The impact of inheritance depth on maintenance tasks – Detailed description and evaluation of two experiment replications

Barbara Unger, Lutz Prechelt (unger, prechelt@ira.uka.de)
Fakultät für Informatik
Universität Karlsruhe
D-76128 Karlsruhe, Germany

Technical Report 18/1998

July 17, 1998

Abstract

Inheritance is one of the main concepts of object-oriented technology. It is claimed that the use of inheritance improves productivity and decreases development time.

John Daly et al. reported on two experiments evaluating the effects of inheritance depth on program maintenance. They found that maintenance was performed significantly quicker for software using three levels of inheritance, compared to equivalent 'flattened' software without inheritance. A second experiment found that maintenance for software using five levels of inheritance tended to be slightly slower than for equivalent software without inheritance.

We report on similar experiments on the same question. Our results contradict those mentioned above. Several crucial changes were made to the setup. In particular longer and more complex programs were used, an inheritance diagram was available to the subjects, and we used more and different kinds of maintenance tasks. Furthermore, our experiment design compares zero level, three level and five level inheritance directly in one experiment.

The results suggest that there is a tendency that deeper inheritance may complicate program understanding. But the effect depends rather on other factors such as complexity of the program and type of maintenance task than on inheritance depth. We found a high correlation between maintenance time and the number of methods to trace to gain program understanding. Further work should be done to identify other influence factors.

Contents

Chapter 1

Introduction

The present report is the detailed description and evaluation of two instances of a controlled experiment on the influence of inheritance depth on the maintainability of object-oriented programs.

In this first chapter we will discuss the general topic of the experiment, give a broad overview of the purpose and setup of the experiment, and refer to related work.

The second chapter describes the hypotheses, preparation, setup, and conduct of the experiment. It also discusses possible threats to the internal validity and external generalizability of the experiment. Chapter 3 presents and interprets in detail the results obtained in the experiment and Chapter 4 explains the differences between the related work and our results. Chapter 5 presents the conclusions.

The appendix contains the handouts used in the experiment: questionnaires, task descriptions, and program source code. Furthermore it describes correct solutions for the tasks. The program source is written in German, but an additional appendix provides German to English translations for the program's main terms.

1.1 Experiment introduction and overview

Object-orientation was expected to be a silver bullet for a long time. Vendors made promises that object-orientation enhances productivity, reliability, reuse, life cycle time, etc. In [9] the claims about object orientation are summarized:

- Methodological advantages of object-orientation:
  - natural imitation of the application domain by object-oriented analysis
  - smooth transition from object-oriented analysis to object-oriented design
  - consistent terminology from analysis to implementation
- General advantages of object-orientation:
  - higher productivity and shorter development time through inheritance and reuse
  - higher quality of the products through encapsulation and modularization
  - less effort for maintenance

Much of the general advantages described above is attributed to inheritance: *I*nheritance is expected to reduce overall code size because functionality that is common to more than one class needs only be defined once. Classes can be viewed as implementations of abstract data types, providing an interface. The functionality can be understood in terms of the interface. Understanding derived classes is eased because they are built on the abstractions implemented by their respective superclasses. Inheritance allows reuse of the classes' implementation. If the superclass is trusted to be correct, procedures that are inherited completely from the superclass need not be tested again. Therefore derived classes are less prone to errors.

These effects are expected to multiply if a class is inherited from more than once. The described advantages lead to the assumption that using more abstraction levels the maintenance of a program will be easier and that the quality of maintenance tasks will be higher.

John Daly et al. [4] conducted an experiment to evaluate the influence of inheritance on maintainability. They tested the hypothesis that "the use of a hierarchy of 3 levels of inheritance depth does positively affect the maintainability of object-oriented programs"[1]. Subjects had to implement identical maintenance tasks on object-oriented programs with a hierarchy of 3 levels of inheritance depth and on functionally equivalent programs with no inheritance. The results indicated that inheritance depth does affect the maintainability such that maintaining a program with no inheritance takes significantly more time than maintaining a program with inheritance depth of 3. A replication of the same experiment had the same results.

Daly et al. also performed a similar experiment with a modified hypothesis derived from the results of the questionnaire survey on the object-oriented paradigm mentioned in Section 1.2. They tested whether "the use of a hierarchy of 5 levels of inheritance depth does affect the maintainability of object-oriented programs such that subjects maintaining the inheritance program version will take longer than those subjects maintaining the flat program version" [4]. The same (!) 29 subjects (except for two) as in the first experiment were used for this experiment. The results obtained did not show significant differences between the program versions.

Our criticism of these experiments is that they used rather small and simple programs. Details on the programs size are listed in the table below. Lines of code (*LOC*) are here defined as all lines of all header files and source files except for leading or trailing blank lines in a file. "{"- or "}"-lines are all lines containing only one brace, *comment lines* are lines of code containing only a comment, and *include lines* are lines of code containing an include statement. Programs `university-1` and `literature` were used in the two parts of the first experiment (1a and 1b, university also in the replication 1r) and `university-2`, an extended version of `university-1` was used in the second experiment. All classes of the programs were quite simple. For example in the university programs, each class represented a person group (such as professor, student, secretary) with almost the same class variables (such as last name, first name, age), and similar methods (for assigning values to the variables and for printing the object).

When we first saw the experiment of Daly et al., we hypothesized that more abstraction levels would universally make maintenance easier, if only the subjects were given a class hierarchy diagram. We decided to perform a similar experiment to test this assumption.

We implemented three Java program versions with identical functionality that differed only in the number of abstraction levels (5, 3, and 0 abstraction levels) and created appropriate class hierarchy diagrams.

Subjects of two university courses participated in the experiment, 57 computer science students from a graduate course and 58 computer science students from an undergraduate course. Each course was randomly divided into three groups. The sampling was stratified for balancing experience among the groups. All subjects of one experiment group received the same program version, the appropriate class hierarchy diagram, and two

[1] For a detailed definition of inheritance depth, see 2.8.1

program	levels of inheritance	Number of classes	Number of methods	LOC	”{”- or ”}” lines	Number of comment lines	Number of include lines
university-1	3	5	21	252	52	29	22
university-1	0	3	26	273	58	16	14
literature	3	6	27	323	60	30	26
literature	0	4	35	370	78	21	19
university-2	5	11	56	694	136	78	56
university-2	0	8	96	1007	208	59	41

Table 1.1: Characterization of the programs used by Daly et al.

maintenance tasks. For each subject we measured the time required for completing each task and we rated the delivered solution. We tested whether and how the performance of the three groups differs.

1.2 Related work

A good summary of the related work can be found in the article of John Daly et al. [4] so that we do not reiterate all the references they cite. But one reference we emphasize: a structured interview study on the object-oriented paradigm [3]. These structured interviews, each based on a template of 23 questions, were carried out with 13 experienced object-orientation users to explore problems discussed in the object-oriented literature. For example “Subjects were asked to define how many levels of inheritance it took before their understanding began to become constrained”. Half of the subject indicated a depth of inheritance of 3-4, a quarter of the subjects indicated an inheritance depth of 5-6 and another quarter more than 6 or “as not relevant”. Another topic with respect to inheritance is the delocalization of functionality. “On the effect of spreading method functionality over the inheritance hierarchy on system understanding almost every subject commented that if the inheritance hierarchy is designed properly than it would not be detrimental to understanding” [3]. These structured interviews were an initial study that was followed by a larger questionnaire survey of experience with the object-oriented paradigm [2]. This study also contained the question “ How deep would your inheritance hierarchy be before you became uncomfortable with it?” The answer scale for the subjects was a little different from the structured interviews but the result indicates that the subjects dare more: 12% answered an inheritance depth of 2-3, 31% a depth of 4-6 and 57% a depth more than 6 or even did not see problems with inheritance at all.

Chapter 2

Description of the experiment

2.1 Experiment purpose and hypotheses

Genericity, overloading, inheritance, polymorphism, and encapsulation are the main components of the object-oriented concept. Until now it is an open question how and to which extent these features have to be employed to get better results: better in the sense of shorter development time, higher product quality, cheaper maintenance, etc.

The purpose of this experiment is to answer a small part of the research question: When is the use of inheritance beneficial? Does it matter how deep the inheritance hierarchy is? Does it matter whether programmers have an inheritance diagram where they can trace method and class definitions?

In this experiment we want to test whether the inheritance depth significantly affects the maintainability of programs. The maintainability will be measured by the time required to complete specific maintenance tasks and by the quality of the solutions delivered.

2.1.1 Our hypotheses

We pose the following hypotheses:

1. **Hypothesis 1:** The more abstraction levels are used, the easier the program is to comprehend and the less time is required for maintenance tasks.
2. **Hypothesis 2:** The more abstraction levels are used, the better is the quality of the delivered solutions.

Explanation of the hypotheses:

Hypothesis 1: Using inheritance provides the maintainer with the information that classes inheriting from a superclass have the same functionality as the superclass except for the methods that are overwritten. So it is not necessary to comprehend lines of code twice or detect that there are duplicates. On the other hand, this effect might be reduced by the dispersal of methods declared in more than one superclass. So the maintainer has to switch between different classes to understand what the functionality of a class is. To reduce this impact, we offer class hierarchy diagrams in the experiment.

Another aspect of this hypothesis is the required time for *implementing* changes when more than one class is affected. Adding common functionality to more than one class can be done by adding and testing the functionality once in a superclass, hence maintenance time can be reduced.

Hypothesis 2: First, when using more code from a superclass, less places to insert defects will be given to a maintainer maintaining the appropriate subclass. A smaller part of the functionality is displayed to the maintainer where s/he can make changes and introduce defects when some functionality is provided in a superclass that needs not be touched. This is important in particular if the maintenance tasks only affect concrete classes from which no other classes inherit. Similarly, if a maintainer needs to make changes that affect more than one class, s/he can often implement the changes just once in a common superclass.

For these hypotheses we assume that the design is sound, i.e., each abstraction level in the inheritance hierarchy represents a sensible and useful concept of the program.

2.2 Experiment design

We designed a basic program version with a deep inheritance hierarchy and derived two functionally equivalent versions from it, differing in the inheritance depth. Functionality, programming style, and comments are essentially the same in all three versions. This is explained in detail in Section 2.8.1.

The subjects are assigned into three groups called G0, G3 and G5, each subject works on one of these program versions. We measure the time required by each individual subject and judge their solutions by both a black box test and an examination of the source program.

The independent variable in this experiment is the inheritance depth. The dependent variables are the time required for each task, the quality of the solutions, the approach the subjects chose and additional subjective information from a postmortem questionnaire.

We also administer a short questionnaire (immediately *before* the actual experiment) for gathering statistical information about our subjects, and for assessing their understanding of object-orientation in general and Java in particular.

2.2.1 Subject groups

We conducted the experiment two times. The first run was with students from our JAKK course (see below). The next run (done two parts) was with students from the Informatik II course.

To make sure to have three comparable groups for each run, we used block-randomization. For the JAKK course subjects, we blocked the subjects based on their pre-performance. The pre-performance was determined as the sum of points the students obtained in the five exercises of the JAKK course. They could take part in the experiment if they achieved more than 75 percent of all points. The motivation to take part in the experiment was to receive lab course credit. In the Informatik II course, we asked the subjects to tell us about their programming experience instead, measured in terms of the size of the biggest program they had ever implemented. All students of the Informatik II course were allowed to take part in the experiment, one third of them did. Their motivation was to obtain a small bonus on their exam grade.

The three treatment groups are called G0 for the group with the flat program, G3 for the group with the program of inheritance depth 3 and G5 for the group with the program of inheritance depth 5. The suffixes I and

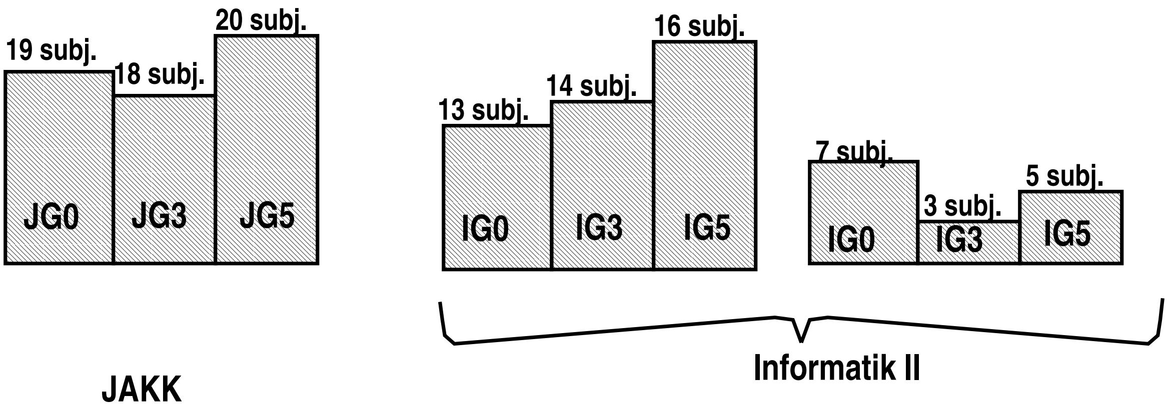


Figure 2.1: Group size and group name for the JAKK and Informatik II subjects. The slight imbalance is due to no-show participants, who registered, but did not appear.

J describe the course: J for the JAKK course and I for the Informatik II course. For the group sizes, see Figure 2.1.

2.2.2 Preparation: The JAKK course

In order to have enough subjects with sufficient knowledge and comparable background, we taught a course, called JAKK ("Java/AWT-Kompaktkurs"), to prepare our subjects. The topic of the course were Java and AWT (Abstract Windowing Toolkit), which attracted over 70 highly motivated graduate students. Participants were required to have at least basic knowledge of OO techniques.[1] It was a five week intensive course with a 90 minute lecture and one exercise per week. The results of the exercises were submitted on disk and were graded by the instructors.

We gave two lectures on Java itself, one on AWT, one on thread programming, and one for discussing frequent errors etc. The course exercises were chosen as follows:

Exercise 1 was to implement a clone of the Unix *wc* program. This exercise was meant for learning how to install and operate the compiler and learning basic Java syntax, file I/O etc.

Exercise 2 introduced reuse of functionality. The exercise was to implement a clone of the Unix *ls* program. Students had to reuse and enhance a given class, and they had to add new classes on their own.

Exercise 3 was on programming with AWT. The task was to display a graphical directory viewer somewhat like the Windows file manager. The students had to reuse their classes from Exercise 2.

Exercise 4 was on programming with threads. The task was to concurrently search files for lines whose acronyms matched the search string or words that were anagrams of the search string. The algorithms were to be implemented using the Template Method design pattern. The number of concurrent threads was to be limited in either of two user-selective ways, using the Strategy design pattern to make the selection transparent. We required all concurrent activity to be properly synchronized and free of race conditions.

Exercise 5 was a simple graphical game. It introduced mouse handling and animation.

[1] We use OO as the abbreviation for object-oriented or object-orientation.

In a questionnaire about the course the course participants indicated that they invested an average of 11.6 hours per week (median: 10 hours) over the 5 weeks of the course. A minimum of 75 percent of all points of the exercises was required for participating in the experiment; 59 of the 73 course participants qualified.

2.2.3 Preparation: The Informatik II course

The subjects were taught as a part of their undergraduate studies. They learned basic computer science knowledge and used the Java programming language for their practical exercises. If they solved enough exercises, they obtained a bonus for their exam.

Among others, practical lab exercises contained a character counter, concatenated lists, different output representations (reuse of the concatenated list), cross sum calculation with reading from standard input, exceptions, read in data with AWT, binary trees, data processing with reading from files, a simple AWT game, and hash tables.

2.3 Experiment format and conduct

We carried out the JAKK run of the experiment in June 1997 on a Tuesday morning, 8 00 to 13 00 hours. The run[2] for the Informatik II course was in June 1997, too, but later in the afternoon, from 16 00 to 22 00 hours. The experiment was conducted in a computer lab equipped with 80 IBM RS6000 workstations running the AIX operating system. The subjects were allowed to use any installed editor. Most of them used either Joe or Emacs. The Java installation was JDK1.1. Although the subjects knew before the experiment that they had to work in an Unix environment and that they should be familiar with it, we could not be sure that they all knew the working environment, so we gave them a short additional handout describing useful commands and tools.

The first handout contained two parts. The first part, a questionnaire, covered personal data such as name, semester, programming experience, etc., see Appendix A.2. The second part was a small test of Java and OO knowledge to see whether the groups had been well balanced, see again Appendix A.2. The subjects had to solve the exercise without using the computer. After they completed this, they were handed out the first work task consisting of a task description (see Appendix A.2), a pretty-printed program listing (see Appendix B.1), a class diagram with the inheritance structure (see Appendix B.2), and the short Unix information mentioned above. At this point, the subjects had to log into their computers. Subjects of the JAKK course found the program in their account, subjects of the Informatik II course had to invoke a script that copied the program into their account.[3] As a subject finished with the task, we collected the exercise, checked the time stamp and whether s/he had invoked the required script for delivering the source code, and then handed out the next work task. We performed the same procedure after the second task and handed out the postmortem questionnaire. The order of the handouts is shown in Figure 2.2

For further details see the actual documents as used in the experiment; they should be self-explanatory and are printed in the appendices starting on page 55.

[2] Actually, this was conducted in two parts on different days, because not all students were available on the same day.

[3] This is because the JAKK subjects had new accounts that we had configured before, whereas the Informatik II subjects used their own accounts.

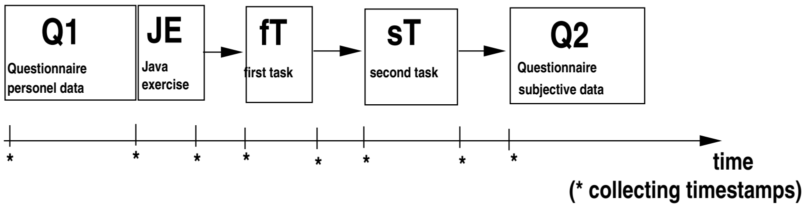


Figure 2.2: Order of the handouts, each arrow between a block means collecting the last handout, controlling of the timestamp and distributing the next handout. At each mark (*) on the time axis a timestamp is collected.

2.4 Experimental subjects JAKK

2.4.1 Education

57 subjects participated in the first run of our experiment. All of them were male Informatics students, and all but 8 of them held a Vordiplom (similar to B.Sc.) degree, one subject did not answer the question. On average, these students were in their eighth semester at the university; see Figure 2.3.

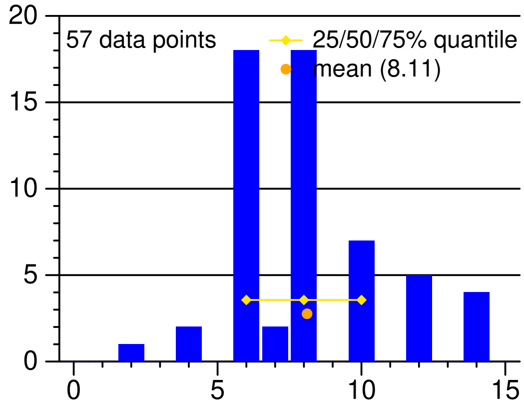


Figure 2.3: [JAKK] Distribution of semester numbers of experimental subjects.

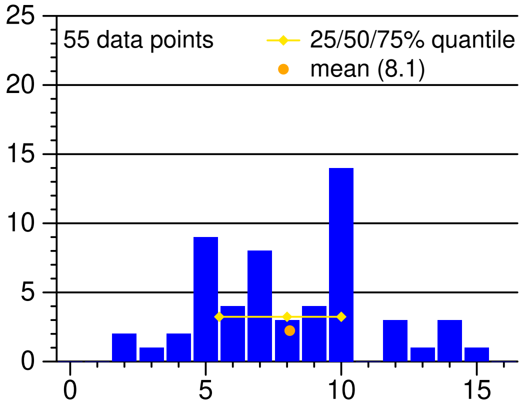


Figure 2.4: [JAKK] Distribution of years of programming experience of subjects.

2.4.2 Programming experience

The students had an average of 8.1 (median: 8) years of programming experience (Figure 2.4), 79% of them had written more or much more than 3000 Lines of Code (LOC) in their life (Figure 2.5). 91% of the subjects claimed they had significant practical experience with object-oriented programming (Figure 2.6) and 47% claimed that they had significant practical experience in programming graphical user interfaces (Figure 2.7). A subject is classified as having significant practical experience if s/he has written more than 300 LOC on a topic. Our subjects had practice with a median of 4 different programming languages (Figure 2.8). The median of the largest program ever written by each of our subjects had 2750 LOC and took 2 person months; see Figures 2.9 and 2.10. 39% of the subjects had also previously participated in a team software project and their median contribution was 1300 LOC and 2 person months to the total median project size of 20 KLOC and 6 person months; see Figures 2.11 to 2.14.

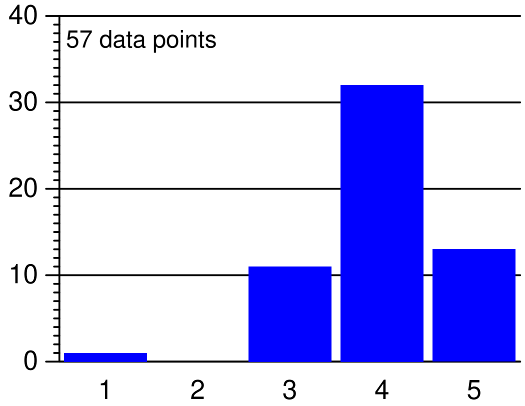


Figure 2.5: [JAKK] Distribution of previous programming knowledge and experience: 1=only theoretical knowledge, 2=less than 300 LOC written, 3=less than 3000, 4=less than 30000, 5=more than 30000.

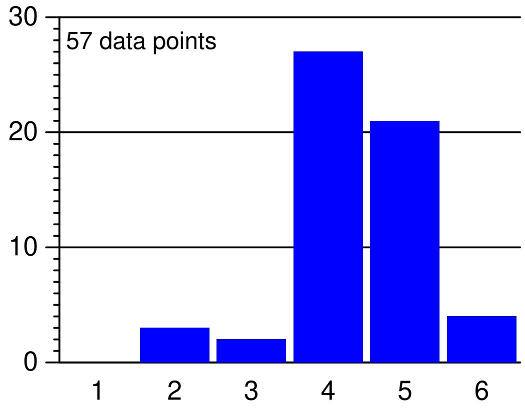


Figure 2.6: [JAKK] Distribution of previous experience in object-oriented programming: 1=no knowledge, 2=only theoretical knowledge, 3=less than 300 LOC written, 4=less than 3000, 5=less than 30000, 6=more than 30000.

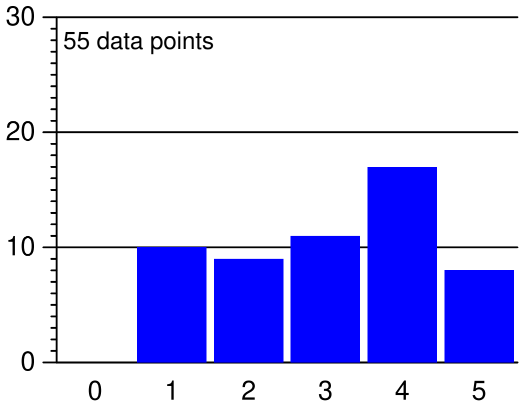


Figure 2.7: [JAKK] Distribution of previous experience programming in graphical user interfaces (GUI). The same encoding is used as in Figure 2.6.

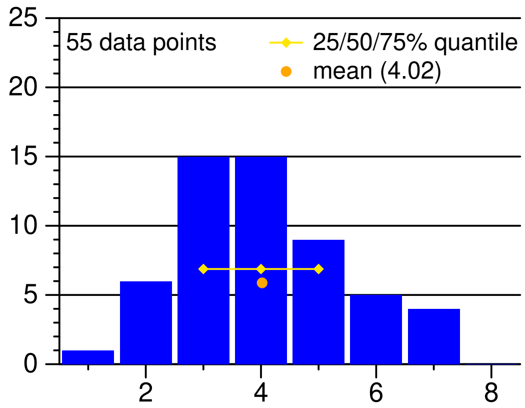


Figure 2.8: [JAKK] Distribution of number of programming languages previously used.

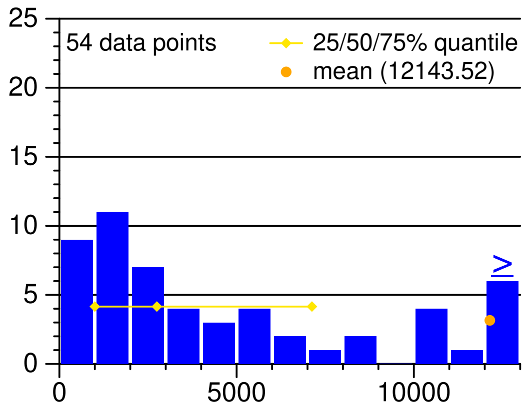


Figure 2.9: [JAKK] Distribution of size (in LOC) of the subject's largest program ever written alone.

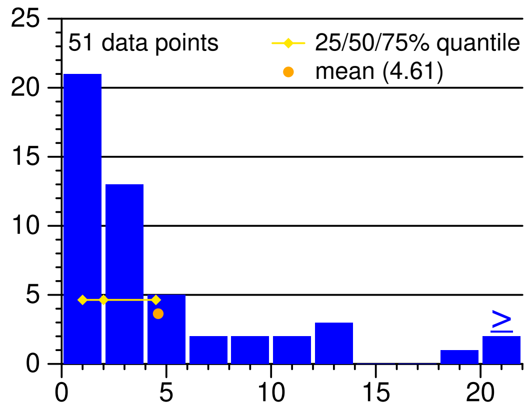


Figure 2.10: [JAKK] Distribution of time (in person months) of largest program ever written alone.

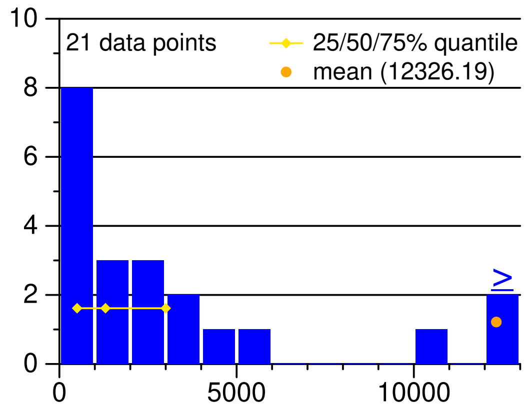


Figure 2.11: [JAKK] Distribution of size (in LOC) of subject's contribution to his/her largest team software project.

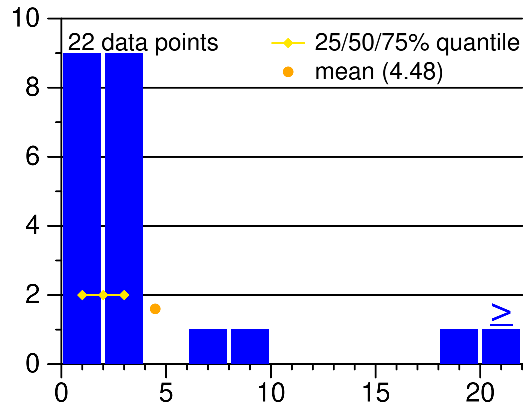


Figure 2.12: [JAKK] Distribution of time (in person months) of subject's contribution to his/her largest team software project.

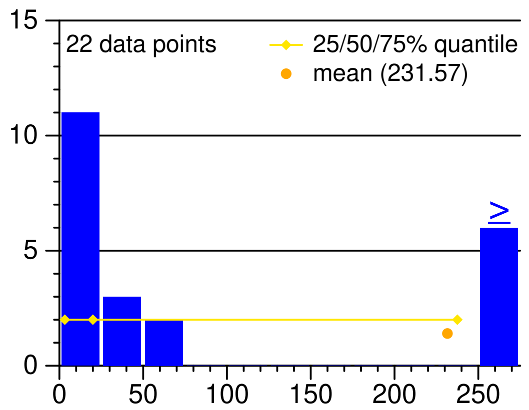


Figure 2.13: [JAKK] Distribution of size (in **K**LOC) of the subject's largest team software project.

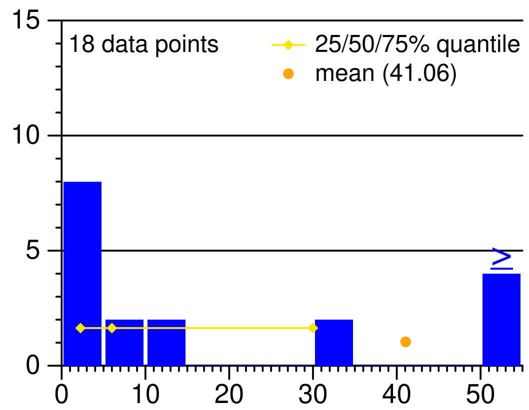


Figure 2.14: [JAKK] Distribution of time (in person months) of subject's largest team software project.

2.5 Experimental subjects Informatik II

2.5.1 Education

58 subjects participated in this run of our experiment. All of them were Informatics undergraduate students (54 male, 4 female) in their Vordiplom (similar to B.Sc.) studies, except two Physics undergraduate students and two Math undergraduate students. All students were in their second semester at the university except one first semester Math student.

2.5.2 Programming experience

The students had a median of 6 years(!) of programming experience (Figure 2.15), 53% of them had written more or much more than 3000 LOC (Figure 2.16). 62% of the subjects claimed that they had significant practical experience with object-oriented programming (Figure 2.17) and 45% claimed that they had significant practical experience in programming graphical user interfaces (Figure 2.18). A subject is again classified as having significant practical experience if s/he has written more than 300 LOC on a topic.

Our subjects had practice with a median of 4 different programming languages (Figure 2.19). The largest program ever written by our subjects had a median size of 2000 LOC and 1.5 person months; see Figures 2.20 and 2.21. 14% of the subjects had also previously participated in a team software project and contributed a median of 350 LOC and 2.0 person months to the total project size of 5.5 KLOC (median) and 2.0 person months (median); see Figures 2.22 to 2.25.

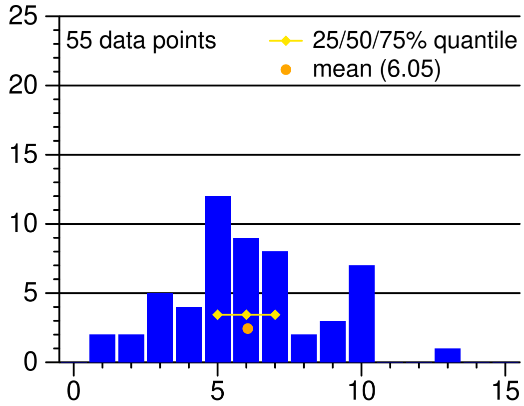


Figure 2.15: [Informatik II] Distribution of years of programming experience of subjects.

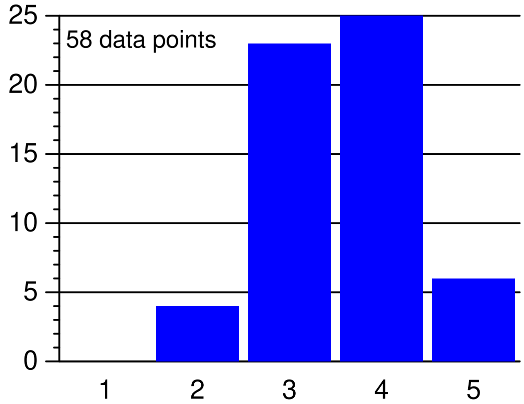


Figure 2.16: [Informatik II] Distribution of previous programming knowledge and experience: 1=only theoretical knowledge, 2=less than 300 LOC written, 3=less than 3000, 4=less than 30000, 5=more than 30000.

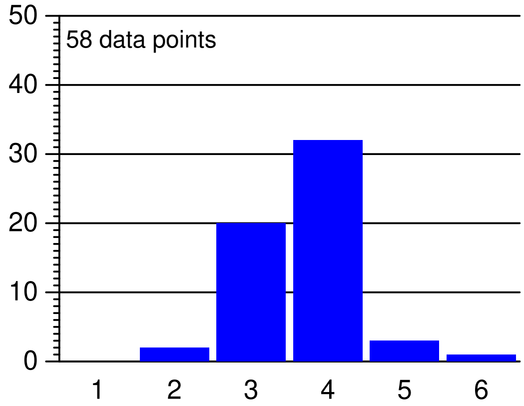


Figure 2.17: [Informatik II] Distribution of previous experience in object-oriented programming: 1=no knowledge, 2=only theoretical knowledge, 3=less than 300 LOC written, 4=less than 3000, 5=less than 30000, 6=more than 30000.

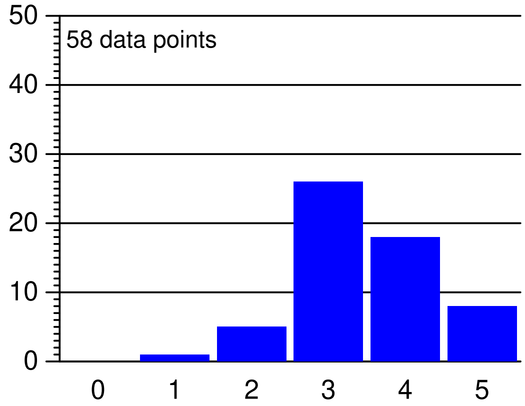


Figure 2.18: [Informatik II] Distribution of previous experience in programming graphical user interfaces (GUI). The same encoding is used as in Figure 2.17.

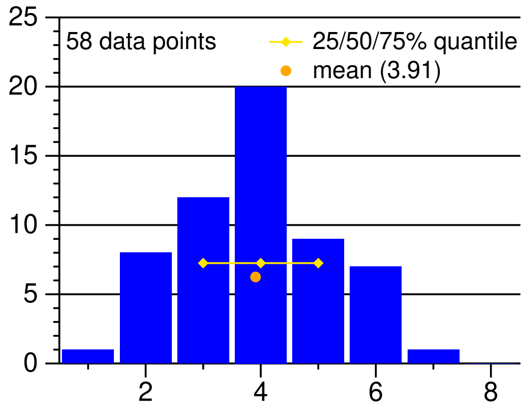


Figure 2.19: [Informatik II] Distribution of number of programming languages previously used.

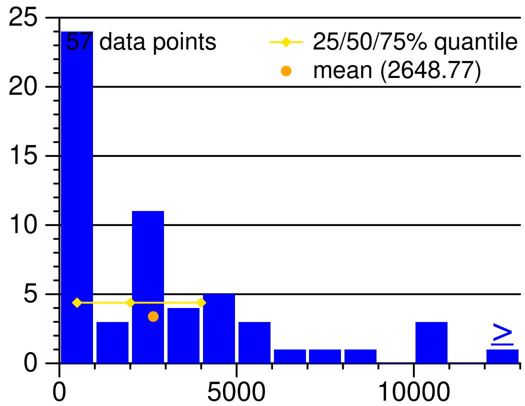


Figure 2.20: [Informatik II] Distribution of size (in LOC) of largest program ever written alone.

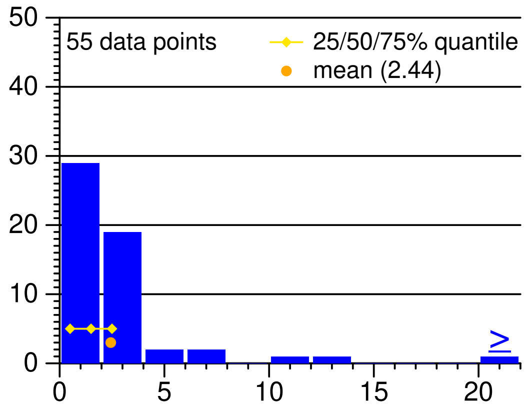


Figure 2.21: [Informatik II] Distribution of time (in person months) of largest program ever written alone.

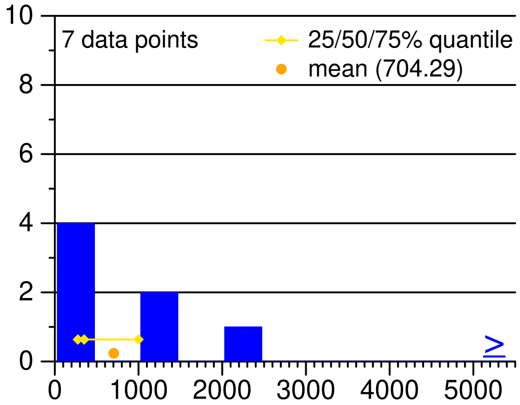


Figure 2.22: [Informatik II] Distribution of size (in LOC) of subject's contribution to his/her largest team software project.

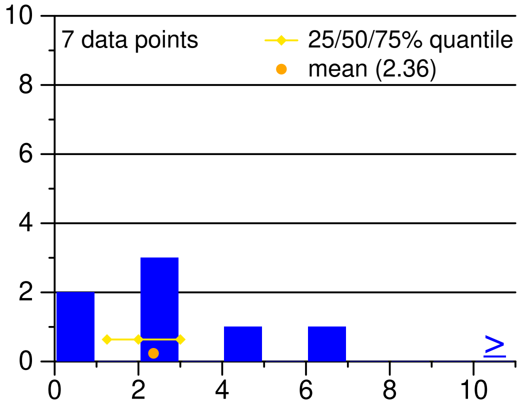


Figure 2.23: [Informatik II] Distribution of time (in person months) of subject's contribution to his/her largest team software project.

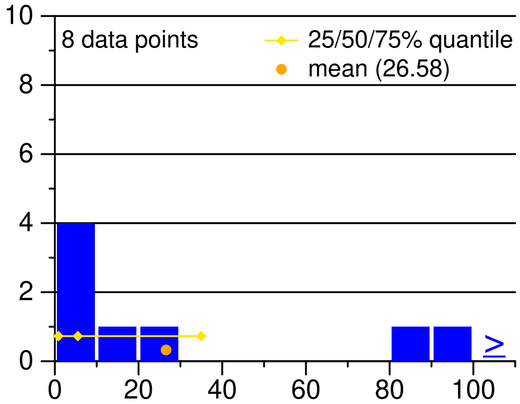


Figure 2.24: [Informatik II] Distribution of size (in **K**LOC) of subject's largest team software project.

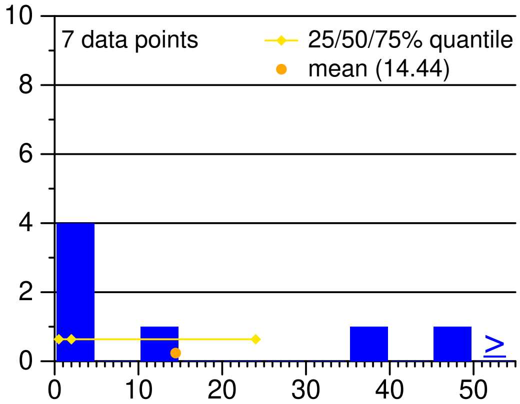


Figure 2.25: [Informatik II] Distribution of time (in person months) of subject's largest team software project.

2.6 Knowledge of Java, inheritance, and polymorphism

Before the actual experiment started, we tried to learn about our subject's knowledge of Java, inheritance, and polymorphism. The introductory questionnaire contained a small test. They were given a tricky Java program consisting of three classes (see Appendix A.2). First the subjects had to mark those lines that would result in a compilation or runtime error. Second they had to determine the output of the program (all without using the computer.)

The answers for the first part, "mark wrong lines" we classified in four different groups: 1 = correct, 2 = subject marked lines wrong because of insufficient understanding of the keyword *static*, 3 = subject marked lines wrong because of insufficient understanding of the keyword *static* **and** marked **one** other line wrong for a different reason, 4 = subject marked **more** than one line wrong not caused by a misunderstanding of the keyword *static*. The *static* (i.e. class variables/methods) feature of Java is not relevant for our experiment. Figures 2.26 and 2.27 show the distribution for the JAKK and the Informatik II subjects.

There are some differences between the three different treatment groups of the JAKK and the Informatik II course but they are not significant, see Figure 2.28.

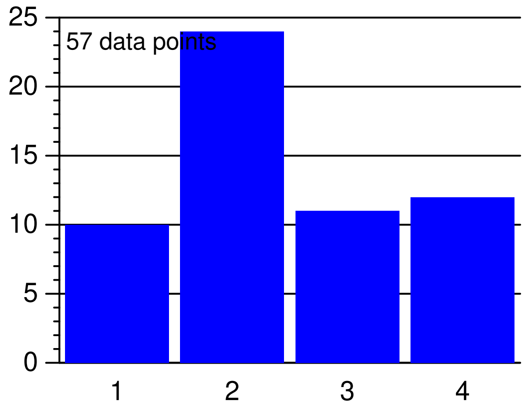


Figure 2.26: Distribution of the answer classes for the JAKK subjects. Answer classification: 1 = correct; 2 = subject marked lines wrong because of insufficient understanding of the keyword *static*, 3 = subject marked lines wrong because of insufficient understanding of the keyword *static* **and** marked **one** other line wrong for a different reason, 4 = subject marked **more** than one line wrong not caused by a misunderstanding of the keyword *static*

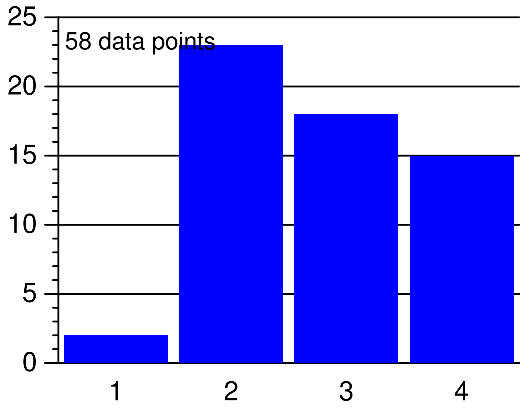


Figure 2.27: Distribution of the answer classes for the Informatik II subjects. The same encoding is used as in Figure 2.26

The second part of the test was evaluated with respect to the first part. We counted the wrong output that was not a consequence of wrongly marked lines. Figures 2.29 shows the distribution for the JAKK and the Informatik II subjects.

2.7 Constraints of the experiment

The tasks and programs used in our experiment had to obey the following constraints:

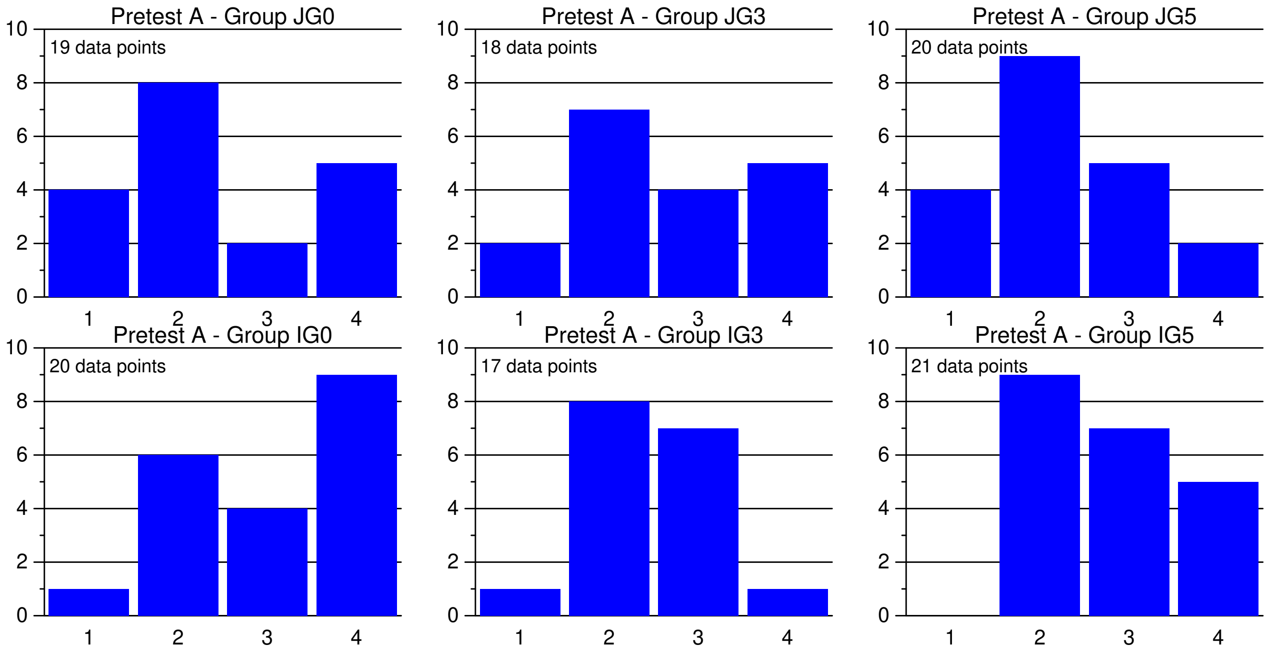


Figure 2.28: Distribution of the answer classes for the JAKK (suffix J) and the Informatik II (suffix I) subjects for the first part of the Java-Test separated into the different treatment groups, where G0 is the group with the flat program version, G3 with the inheritance depth 3, and G5 with the inheritance depth of 5.

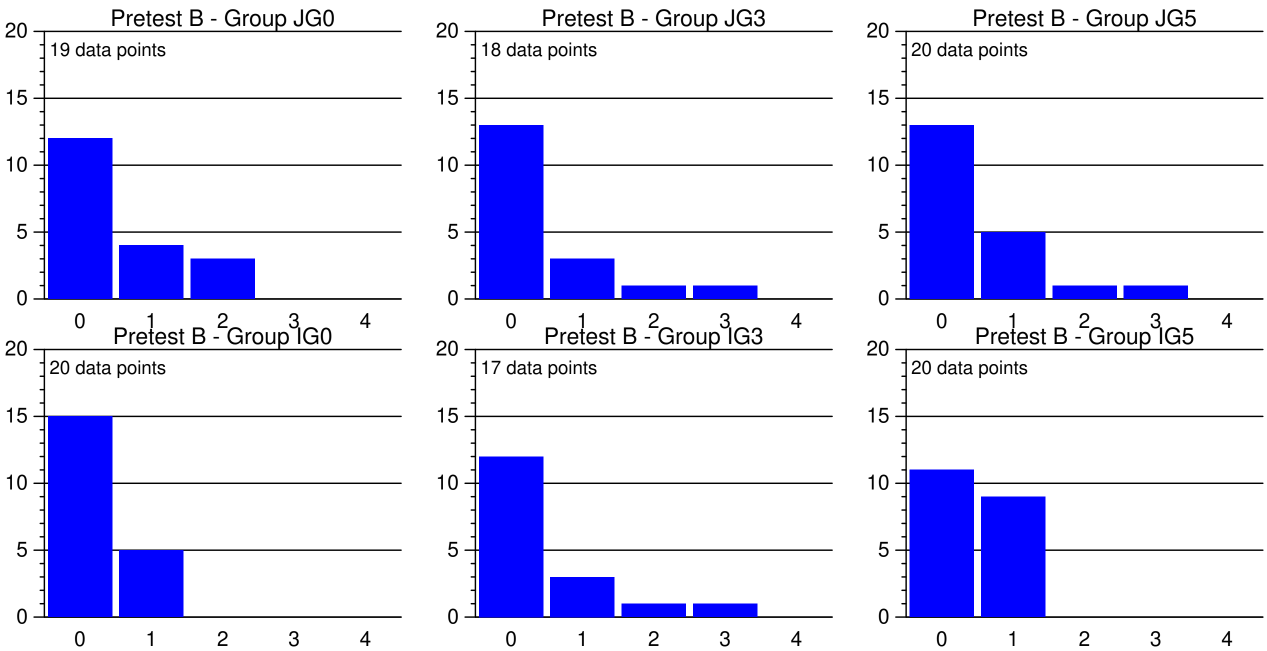


Figure 2.29: Distribution of the number of wrong answers for the JAKK and the Informatik II subjects for the second part of the Java-Test separated into the different treatment groups.

1. The experiment has to be carried out in a single time interval. We assume that four hours is a reasonable maximum, due to the subjects' concentration ability. Therefore, the tasks should not consume more than about two hours for an average subject.
2. The application domains of the programs had to be well understandable by all subjects. Therefore, we could only use domains that were either known from the course or were so simple that they could be explained in a few sentences.

We handled these constraints as follows. Constraint 1 rules out large programs that are much larger than the largest program they have ever written alone. (The median of 2750 LOC for the JAKK subjects and 2000 LOC for the Informatik II subjects suggests a program of a size not much larger than 2000 LOC.) Complex and difficult tasks are also ruled out. Therefore, we used modest tasks (described below) and a program of modest size.

For satisfying constraint 2, we selected a domain that is mostly intuitive and easy to explain in a few words. Viewing data points in different manners is common and not difficult to understand even if the subjects do not know the correct technical terms. Assuming that some subjects were not familiar with the technical terms we provided descriptions of the domain and the meaning of the technical terms used in the program, see Appendix A.2.

2.8 Programs and Tasks

This section will shortly describe the programs and tasks and explain why we chose them. The tasks for the JAKK subjects differ from the tasks for the Informatik II subjects but the programs are almost the same, see below. One can find the task descriptions translated into English in Appendix A.2 and the corresponding program listings in Appendix B.1 (not translated, but see the translations of technical terms given in Appendix C).

2.8.1 Programs

For testing the hypotheses we need programs differ only with respect to inheritance depth. Therefore we developed a base version of an object-oriented program with a deep inheritance hierarchy and derived two new program versions from it by eliminating inheritance levels.

The base version is a Java application using the Abstract Window Toolkit. The application, called "Boerse", is for presenting stock market data that are stored in two files. The stock market data can be displayed in different kinds of tables (data of one day, data of the last week, data of the last month, the percentage of changes between two prices) or charts (for selected stock data prices of a day or the last month). For more details about the domain and functionality, see Chapter B.1 or A.2.

The base version consists of 28 classes (7 abstract classes, 1 interface and 20 concrete classes) with an inheritance depth of 5, where inheritance depth is defined as the number of extension of the longest chain of extensions (*B extends A*)[4]. For a visualization of that definition, see Figure 2.30. Some of the abstract classes use the design pattern "Template Method", (as in a quicksort algorithm that is implemented in the abstract superclass except for the compare method for the different data types. This base version with an inheritance depth of 5 will be called Boerse5 in the following.

[4] A maintenance task on a program with an inheritance depth of n that requires adding a new abstraction level is defined to have inheritance depth $n+1$, because maintainers have to understand n inheritance levels plus the one they have to add.

```
class B extends A{
}
class C extends A{
}
class D extends C{
}
class E extends C{
}
class F extends D{
}
class G extends E{
}
class H extends E{
}
class I extends H{
}
```

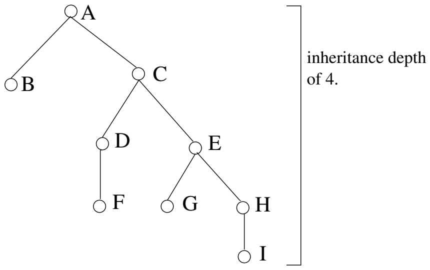


Figure 2.30: Inheritance depth is defined as the number of pairs in the longest chain of extensions ("A extends B") of a class. In this example: C extends A; E extends C; H extends E; I extends H. This is an inheritance depth of 4.

	JAKK			Informatik II		
program version	LOC	# classes	#methods	LOC	# classes	#methods
Boerse5	1200	28	80	1187	28	79
Boerse3	1344	27	100	1317	27	96
Boerse0	2470	20	158	2465	20	160

Table 2.1: Size characterization of the programs used for the experiment

The version with an inheritance depth of 3 was derived from the base version by making all concrete classes inherit from only abstract classes following the suggestions of Gamma et al. [7] to prefer inheriting interfaces over inheriting implementations. The functionality (methods) and the comments defined in inherited concrete superclasses were moved into the derived class by "cut and paste". Functionality and comments that are not used by the derived class were omitted. In the following, we call this version Boerse3.

The flat version, called Boerse0, was built from the inheritance version Boerse3. All abstract classes were removed and the functionality of the superclasses were added into the concrete classes. Useless functionality that was defined in the superclass but not used in derived classes was removed. Comments were moved along with the code as appropriate.

There are only small differences between the programs for the JAKK course and the Informatik II course. The programs for JAKK subjects are written in the programming style of JDK 1.0.2 (using the now deprecated event handling) because most subjects learned this. The Informatik II subjects learned JDK1.1 with the new event handling mechanism. So we modified the programs accordingly and added some comments concerning the event handling. For details, see Appendix B.3.

2.8.2 Tasks

In total, we had three different tasks, called "Task 1", "Task 2A", and "Task 2B". Subjects from JAKK had to work on all three tasks. For them, we did not measure Task 2A and Task 2B separately; the combination is called "Task 2".

Trying to satisfy constraint 1 (time) we omitted Task 1 for the Informatik II subjects because we assumed that the Informatik II subjects were less experienced programmers than the JAKK course subjects.

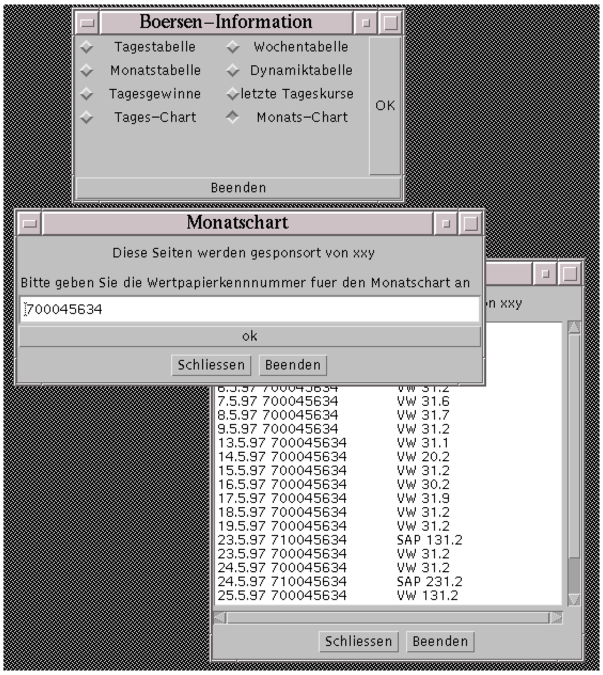


Figure 2.31: Screenshot: The upper window is the main menu window. The middle window is for selecting the stock number. A user can select a stock number and press the OK button. The lower left window displays a month table of all stock data of the last 30 days.

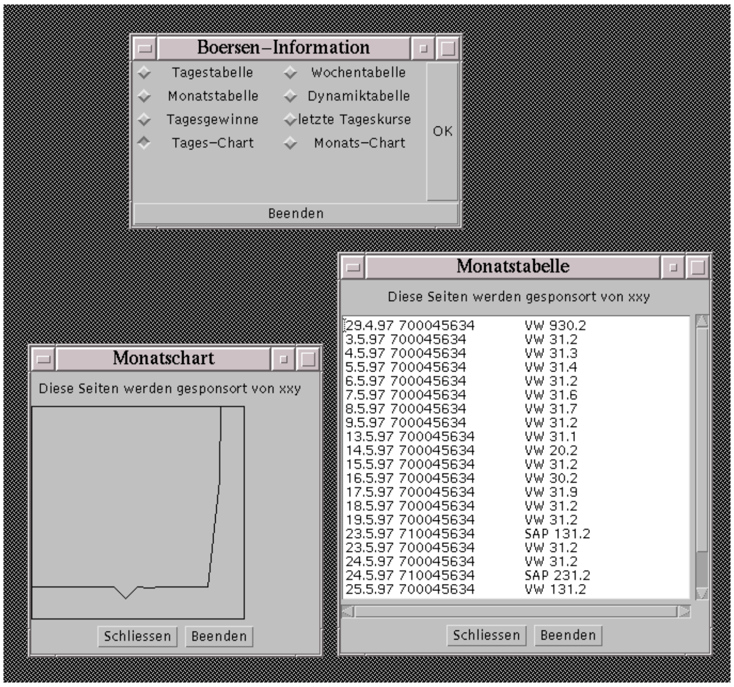


Figure 2.32: The scene from Figure 2.31 after pressing the OK button.

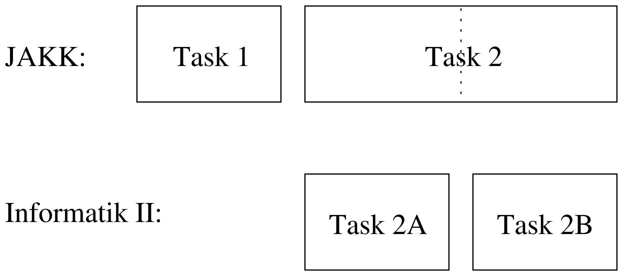


Figure 2.33: Task order for the two groups. Because of the time constraints we omitted Task 1 for the Informatik II group and split Task 2 into two subtasks 2A and 2B instead.

Task 1 – Y2K-Problem

The first task addresses the Y2K-problem (Year 2000). The format of the date in the original input files is “dd.mm.yy”. We provided a second set of input files that differed only in the date format: “dd.mm.**yy**yy”. The subjects had to change the program so that instead of the original input files with the old date format the new files with the extended date format are handled by the program.

Extract from the file KURSE (PRICES), where the first line is the translation of the row names. (This line was not in the original files.)

```
#date           time    stocknumber ID      name            price
#DATUM          UHRZEIT WERTPAPIERKENNUMMER NAME        KURS
01.03.97        11:51   700045634           Daimler Benz    130.70
01.03.97        12:34   700045634           Daimler Benz    134.50
01.03.97        14:26   700045634           Daimler Benz     34.20
```

Extract of the file KURSE_2000:

```
#date           time    stocknumber ID      name            price
#DATUM          UHRZEIT WERTPAPIERKENNUMMER NAME        KURS
01.03.1997      11:51   700045634           Daimler Benz    130.70
01.03.1997      12:34   700045634           Daimler Benz    134.50
01.03.1997      14:26   700045634           Daimler Benz     34.20
```

Extract from the file KASSAKURSE (AVG_PRICES):

```
#date           stocknumber ID          name    day-average price
#DATUM          WERTPAPIERKENNUMMER     NAME    KASSAKURS
02.04.97        700045634               VW      131.20
03.04.97        700045634               VW      231.20
03.05.97        700045634               VW       31.20
```

Extract from the file KASSAKURSE_2000:

```
#date           stocknumber ID          name    day-average price
#DATUM          WERTPAPIERKENNUMMER     NAME KASSAKURS
02.04.1997      700045634               VW      131.20
03.04.1997      700045634               VW      231.20
03.05.1997      700045634               VW       31.20
```

Task 2A – Time Interval Price Display

The given program has two ways of displaying stock charts and six sorts of table-based stock data displays. All table displays present data for fixed time intervals (day, week, or month) and there is no way to focus on a particular stock number. In contrast, the chart output asks the user to enter a stock number. Then the chart for that particular stock in a fixed time interval will be displayed.

For Task 2A the subjects had to add a new table-based output. But instead of a fixed time interval, the user should be prompted (similar to the stock number selection for chart displays) to specify the time interval for which the data was to be displayed.

Task 2B – Time Interval Gain/Loss Display

Task 2B is very similar to Task 2A. Again, a given table display, this time a gain/loss table, must be extended to accept user input for selecting a specific time interval.

2.8.3 Solving the tasks

Solving Task 1

In this section it will be discussed that Task 1 does not require a full understanding of the programs and the inheritance. Most JAKK subjects will complete Task 1 without even trying to fully understand the code. Others will gain a deeper understanding of the code before tackling Task 1.

A straightforward way to solve this task is to trace the data flow through the program. The lines where the input files are read are easily found in all versions of the code. In Boerse0 there are 8 identical code fragments that handle the file input. In Boerse3 and Boerse5 there is only a single position in the code. With the inheritance diagram at hand, the spot where input files are read is easy to find, for example Program Listing B.1.3, lines 184-200.

From the file input method called `leseDaten [readData]` individual lines of input are read. These lines are then processed in two different methods. One method, called `selektiere [select]` (Program Listing B.1.3, lines 378, 466, 556, and 794) analyses the line and determines whether it will be included into the output. This method is invoked by another method, called `erzeugeDatenvector [generateDataVector]` (Program Listing B.1.3, lines 368, 459, 563, and 800) that constructs a special data type from the input line. In both methods the input line is segmented with `substring` and explicit line positions. With that understood, the subjects could conclude that they had to change the explicit positions in the `substring` operations in both methods.

In the flat version Boerse0 and in Boerse3, each of the two methods `selektiere [select]` and `erzeugeDatenvector [generateDataVector]` appears 8 times. In Boerse5 the methods appear only 4 times. However, if the concept is understood, a single pass through the file with the "search next" mode of the editor is sufficient to complete the task since all appearances look much alike.

In addition, the special data type that is used to store the date had to be adopted to the four digit year format. This change was identical in all three versions.

The difference with respect to inheritance between the flat version and the other versions was locality. In the flat version, the name of the data file is stored into an instance variable. In immediate proximity in the code, this variable is used to open the file. This is different in the other two versions. The method that handles the files (`leseDaten [readData]`, class `KostenloseInfos [FreeInfos]`) uses the same instance variable to access the file name, however, that instance variable is initialized in the sub-sub-classes (class `ZeitraumTabelle [IntervalTable]` in Boerse5), see Figure 2.34.

A solution for this task is given in Appendix A.3.1

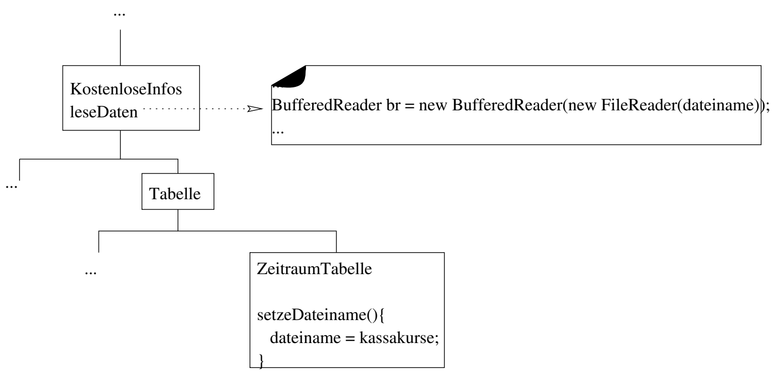


Figure 2.34: Initialization and access of the input file name for Boerse5

Solving Task 2A

From the task description and from executing the original program, the subjects can conclude that almost all of the required functionality is already present in the program. The new table-based output must be quite similar to existing table-based output. In addition, for implementing the required user input the subjects can find an example from the chart implementations where the user is prompted to enter a stock number. Moreover, the main menu must be extended to offer the new option in close analogy to its existing entries.

Gaining comprehension for Boerse0: For the flat version Boerse0, the subjects can gain an understanding of the charts and tables by selecting an example class of each; one table-based class and one chart class. Due to the flat code, a single class contains all code needed to implement the table output or the chart output. The class implementations are organized homogeneously. Thus, only one class has to be understood fully, e. g. `WochenTabelle [WeekTable]`. If the maintainer realized that the functionality he needed is already there, a line-by-line comparison reveals differing aspects between chart and table implementations: some new instance variables are needed for the user input and three methods are different, namely, the event handling method (`myaction`[5]), result preparation method (`selektiere [select]`) and result display method (`stelleDar [present]`). Given an understanding of one table class and the differences of the two class types (table classes and chart classes) the solution of the task becomes quite simple: A new class is created as a copy of an existing table class; then the three methods mentioned above are altered according to the differences in comparison with the chart class.

Gaining comprehension for Boerse3 and Boerse5: Doing the same maintenance task for Boerse3 or Boerse5 this is more complicated. First, one of the table classes has to be understood and the differences between the table-based classes and the chart classes have to be found. However, for gaining an understanding of the class `WochenTabelle [WeekTable]` or `MonatsTabelle [MonthTable]`, one must seek and find the functionality in the inheritance hierarchy. This is shown in Figures 2.35 and 2.36.

[5] Only for the JAKK subjects. Informatik II subjects had event handling using inner classes, see Section B.3

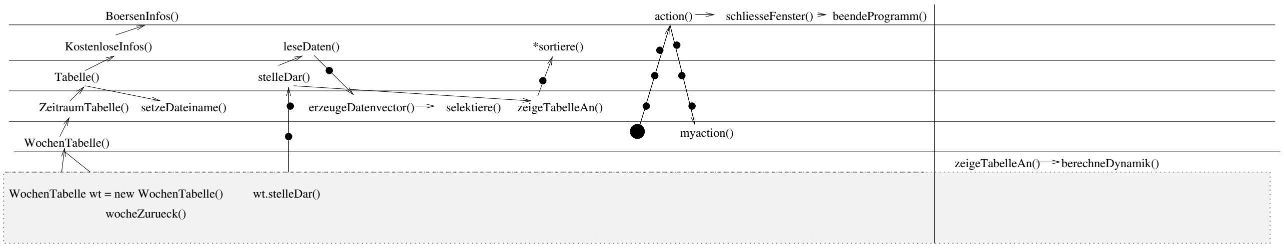


Figure 2.35: For Boerse5 program comprehension must be gained by searching through the inheritance hierarchy, starting in the main program. The arrows mean an invocation of a method. An arrow crossing a horizontal line means that the method invoked is located in a super- or subclass. The shaded region in the lower part are method calls from classes outside the main hierarchy. The black dot in the figure is a starting point for searching the method "action" that is invoked implicitly by the AWT. Methods are marked with a star if they invoke other methods but due to meaningful names method invocations do not need to be traced. The left side of this figure is for Task 2A and the right side for Task 2B.

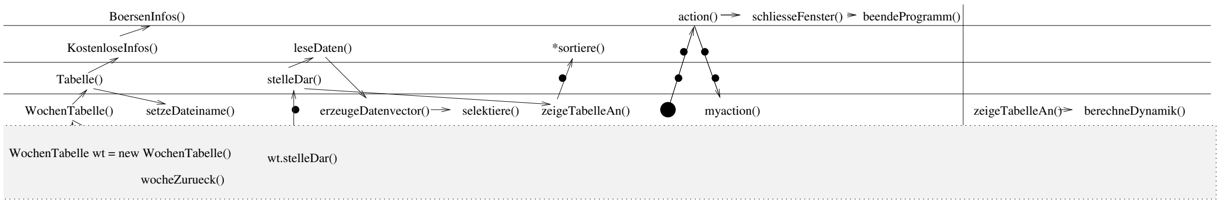


Figure 2.36: Same as Figure 2.35 for Boerse3.

The differences between the charts and tables can be found by comparing the two types of classes including the functionality that is realized in superclasses. So the maintainer has not only to search two classes for differences, like the Boerse0 case; instead, s/he has to search the whole inheritance path to the root.

As discussed for Boerse0, three methods differ. In Boerse3, result display is implemented in the abstract table super-classes of the concrete table classes. Result preparation is implemented on the concrete class level. Therefore, for all three methods different classes must be compared to understand the differences between table output and chart output.

In Boerse5 this becomes even more complicated since result preparation occurs at different levels of the class hierarchy.

In Table 2.2 the methods that have to be compared, copied and/or changed are listed with the class levels in which the classes are located that contain such a method.

	`myaction()`		`selektiere()` [`select()`]		`stelleDar()` [`present()`]	
	Chart	Table	Chart	Table	Chart	Table
Boerse0	0	0	0	0	0	0
Boerse3	3	2	3	3	2	2
Boerse5	4	2	3	3	3	3

Table 2.2: Search levels where the differing methods can be found.

Solving: Whereas the structure of the given code suggests to copy and modify an existing table class for Boerse0, there are different solution types for Boerse3 and Boerse5. The cut-and-paste-and-modify approach does not work, since the code to be modified is spread across super-classes as well. Therefore, some methods must be redefined to hide unwanted implementations in super-classes. For Boerse3 the new table can be added at the same level as existing tables, or it can be added as a subclass of an existing table class. For Boerse5 the same is true; however, in this case different types of tables are at different levels of the inheritance graph. Subjects must be more careful to pick the appropriate ancestor. The most useful ancestor class is the interval table display class `ZeitraumTabelle [IntervalTable]` that implements large parts of the new functionality already.

Solving Task 2B

Since the task is similar, the cut-and-paste-and-modify approach can be used again for Boerse0. In comparison with Task 2A, analogous changes must be repeated (three methods and some instance variables) and analogous code lines must be kept unchanged. For Boerse3 and Boerse5 a new class must be introduced, but the body of that class is completely identical to the class introduced during Task 2A.

A solution for Tasks 2A and 2B is given in Appendix A.3.2

2.9 Internal validity

There are two sources of threats to the interval validity of an experiment[6]: Insufficient control of relevant variables, and inaccurate data gathering or processing.

As far as we can see, all relevant external variables have been appropriately controlled in this experiment. In particular, there is no significant bias in the random group sampling, the subjects seemed willing to perform as best as they could in all three experimental conditions, there was acceptable mortality in the JAKK group (7 subjects did not start the second part of Task 2, 2 subjects of group JG0, 3 subjects of JG3, and 2 subjects of JG5). The mortality in the Informatik II group was biased (11 subjects did not start with Task 2B, 3 subjects of group IG0, 2 subjects of IG3, and 6 subjects of IG5) so the results presented have to be interpreted carefully. The environmental conditions were essentially the same for all subjects.

We minimized data gathering errors by exercising utmost care. Data processing was almost completely automated. Manual and automated consistency checks were applied for detecting various kinds of possible mistakes in data gathering or processing.

2.10 External validity

There are four causes for differences between the experimental situation and real software maintenance situations that limit the generalization (external validity) of the experiment: (1) subjects with more experience, (2) programs of different size, structure, or programming language, (3) familiarity with the program, and (4) tasks of different kind or complexity.

[6]Definition from [1]: "*Internal validity* refers to the extent to which we can accurately state that the independent variable produced the observed effect."

1. The most frequent concern with controlled experiments using student subjects is that the results cannot be generalized to professional software engineers because the latter are so much more experienced. Because of the replication with different groups (JAKK and Informatik II), we can estimate the effect of having programmers with different experience. If the data have the same trend for more experienced subjects as for less experienced subjects (which ours do), the results may be also transferable to professional programmers.Furthermore, our JAKK subjects are quite experienced, anyway; see Section 2.4.1.

2. Another obvious difference between our experiment and reality is program size and structure. Compared to typical industrial programs, the experiment programs are rather small and are not as complex as (large size) industrial programs. The programming language is Java, so we can, for instance, not determine the effects of multiple inheritance. The influence of both factors is unknown.

3. Usually a programmer maintains a program more than once. So after a learning phase the programmer works on a familiar program. Our subjects work on a unfamiliar program and they are perhaps not acquainted with the programming style. It is unclear whether or how program familiarity interacts with inheritance depth.

4. Finally, the kind of task and its complexity may be different. In the experiment, the tasks were relatively simple program changes or program extensions that can be done mainly by adding some completely new classes, and subjectively the complexity was rather low. The experiment does not tell us much about other kinds of tasks.

Chapter 3

Experiment results and discussion

This chapter presents and interprets the results of the experiment. Section 3.1 explains the methods of statistical analysis and result presentation that we used and explains why they were chosen. Section 3.2 presents the main results (subjects' measured performance) and Section 3.3 adds data from the postmortem questionnaire for understanding some underlying effects.

3.1 Statistical methods

Most of our formal statistical reasoning is for comparing the means of pairs of distributions. Hardly any of these distributions are "normal distributions": Some of them are discrete and coarse-grained, several of them have two or more peaks, and many are heavily skewed or even monotone. Therefore, statistical analysis must not use a parametric test such as the t-test that assumes a normal distribution. On the other hand, classical non-parametric tests, such as the Wilcoxon rank sum test, cannot perform inference for the mean but only for the median, which is less relevant for our purpose. Moreover, rank sum tests cannot provide confidence intervals.

In this report, we thus use resampling statistics (bootstrap) to compare the means of arbitrary distributions non-parametrically. The basic idea of resampling is described in [8]. For all resampling statistics we used 10000 trials.

The other statistical test used is the χ^2 test on a four field table for testing the significance of frequency differences of a binary attribute. The application is comparing the incidence of a certain event in two experimental groups. We report the Pearson's chi-square test with Yates' continuity correction and the Fisher exact p statistic in addition to the p-value of the Pearson's χ^2 test; the exact p is more reliable in some cases (Fisher's exact test does not require the cell counts to be large). These tests were performed using Statistica 5.0 (Windows NT 4.0) and S-Plus 3.4 (Sun OS 5.5).

We consider a test result significant if p is less or equal to 0.1.

3.2 Performance on the tasks

In this section, we compare the performance of the groups with different inheritance depths. For each task, we first consider the individual types of errors that occurred and then investigate global quantitative effects with respect to time required and solution quality obtained.

3.2.1 Metrics employed

In the evaluation below, the following measurements and criteria will be used. Each class of them is described by the following terms [6]: A measurement can be either objective (and therefore in principle completely reproducible) or subjective (and therefore subject to debate); it can be either direct or be derived from other measurements; it can be on a nominal, ordinal, interval, cardinal, or absolute scale; it can have limited precision and limited accuracy even if it is objective.

Groups (objective, direct, nominal scale, completely accurate): The groups were used for comparing performance with maintenance tasks on programs of different inheritance depth. The groups are named by the course (JAKK = J, Informatik II = I) and the program version they worked on (flat version = G0, inheritance version with depth 3 = G3, inheritance version with depth 5 = G5).

Time measurements (objective, direct, cardinal scale, precision 1 minute, inaccuracy should be about 1 minute): The subjects noted start and end times on each page of the experiment materials. We computed the difference between the end of the last page of a task and the start of the first page of the task as the work time measurement; the subjects did not make major breaks that had to be subtracted. We also gathered time data by scripts run on the subject's computer during the experiment so that we could do consistency checks in case that the start time of a new task was not close to the end time of the previous task. We used the time data only on the task level (as opposed to the subtask level as for the JAKK subjects between the first and the second part of Task 2) as it is the one with the clearest interpretation and the one that was validated upon collection of each part of the experiment materials.

Points achieved (subjective/objective, direct, cardinal scale, precision 1 point): We graded the solutions by assigning points, using a penalty system where possible (subtracting a fixed number of points for each kind of error). The individual penalties are explained in the actual results sections below. We consider the differences of numbers of points between the groups for each subtask individually and for the whole task. Points are meant to characterize the quality of a solution, but this interpretation must be applied only with care.

Incidence counts (subjective, direct, absolute scale): Incidence counts reflect how often a particular event occurs in a group. We considered incidence counts for various classes of errors in the solutions delivered by the subjects. In a few of the cases it is debatable whether a certain solution is an instance of the event or not, so there is some amount of subjectivity in the data. Except for subjectivity, the incidence data is considered accurate.

Productivity (objective, derived, cardinal scale, precision $\frac{1}{60}$ pt/hour): A measure that is meant to characterize productivity was derived by computing points per time unit. Due to the restrictions of the point measure, points per time also have to be interpreted with care.

3.2.2 Notation

For the discussion of the results we introduce a short and compact notation to summarize the group comparisons. This will help the reader to gain a quick overview.

Definitions:

$G_i, G_j \in \{IG0, IG3, IG5, JG0, JG3, JG5\}$
$R \in \{t, pt, pr\}$ where t is mean time; pt is mean points; pr is mean productivity

$R(G_i) < R(G_j)$, if the result R of G_i is **significantly smaller** than the result R of G_j, $p < 0.1$.

$R(G_i) \leq R(G_j)$, if the result R of G_i is **smaller** than the result R of G_j, but $0.1 \leq p \leq 0.2$

$R(G_i) \approx R(G_j)$, if the p-value is larger than 0.2.

We also use the symbols $>$ and $\geq$ accordingly.

3.2.3 Performance on the tasks – JAKK

Task 1 – "Y2K"

This task can be divided into two dependent subtasks, program comprehension and program changing. The program comprehension does not have to be extensive to get the task done because the way how the date is read and manipulated is easy to discover. See 2.8 for a detailed discussion of Task 1.

Time: It is remarkable that JG5 and JG0 are significantly faster than JG3 ($t(JG5) \approx t(JG0) < t(JG3)$, see Table 3.2, lines 4 - 6). Figures 3.1 and 3.2 show the time distribution for all subjects and by treatment groups. You can see from these figures that there are some "outliers" that may distort the results. We can eliminate them by cutting off 10 or 20% of the right side of the distributions but the results remain similar. The obtained results seem to contradict Hypothesis 1. This will be discussed in the Paragraph *Conclusion* below.

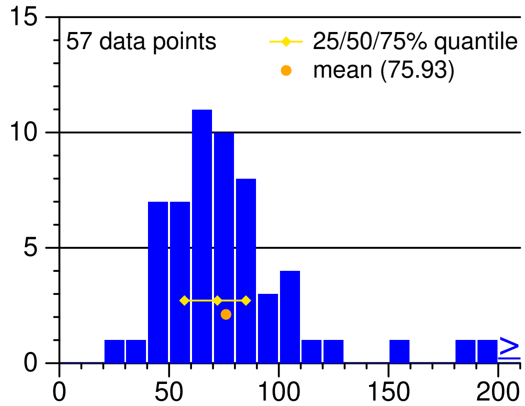


Figure 3.1: [JAKK] Distribution of the time required for the task "Y2K"

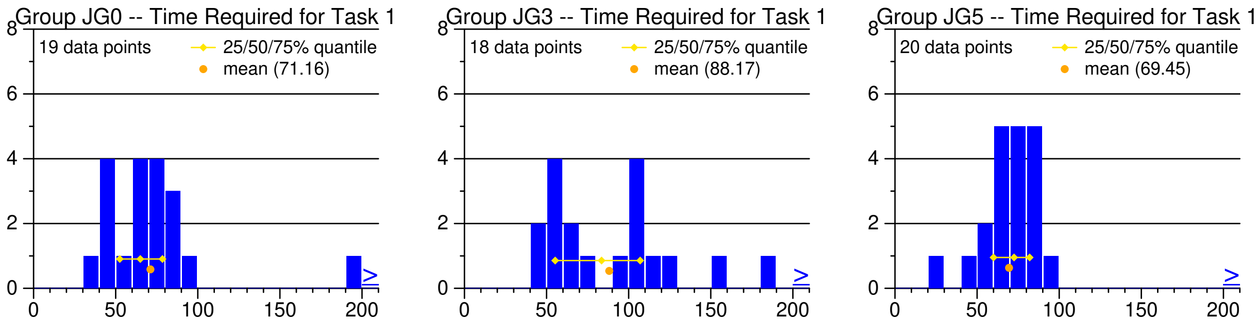


Figure 3.2: Distribution of the time required for the task "Y2K" separately for the three treatment groups.

Points: The quality of the solutions differs between the groups. The maximum of 4 points was given for correct solutions as determined by a black box test. Errors cost between 1 and 4 points depending on the error

group	no. of subjects	no. of correct solutions
JG0	19	100%
JG3	18	83%
JG5	20	70%

groups	χ^2	p	Fisher exact p
JG0 - JG3	1.57	0.21	0.105
JG0 - JG5	4.63	**0.031**	**0.020**
JG3 - JG5	0.34	0.56	0.454

Table 3.1: [JAKK] Fraction of correct solutions per group (left) and significance p of the group differences (right) for Task 1.

class. For errors of class B (one/two/more tables or charts are empty) we reduced the points by 1, 2, or 4; errors of class C (at least one table or chart wrong; subsequent substring indices not corrected) costed 1 point, and errors of class D (forgot to change a hard-coded date variable) costed 2 points. Error class X for miscellaneous errors was judged individually for each case. As you can see from Figure 3.3, all subjects from JG0 produced correct solutions while some subjects from the groups JG3 and JG5 made some errors. A significance test for differences in correctness between JG0 and JG3 or between JG0 and JG5 is not useful because in group JG0 there is no variance at all. An incidence count on the number of correct solutions shows a significant difference between the groups (see Table 3.1; for this table the Fisher exact p-value is more reliable, because some of the counts in the four field table are very small).

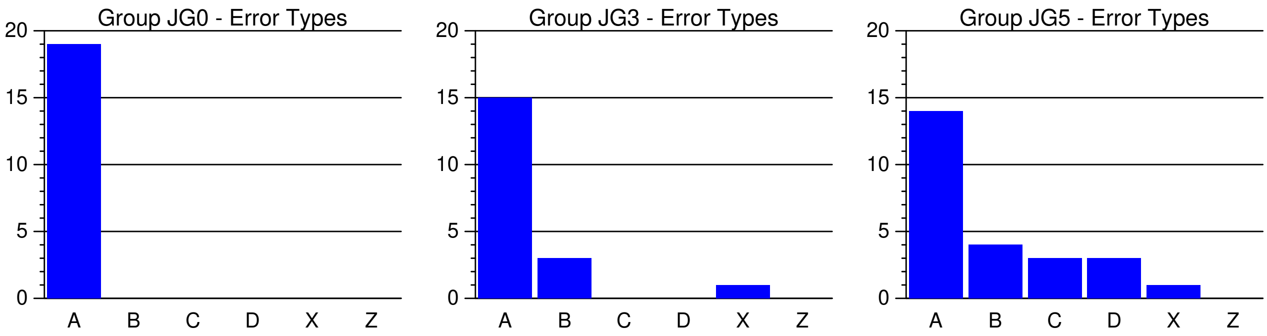


Figure 3.3: Frequency of different error types for task "Y2K". Codes: A = everything OK; B = one/two/more tables or charts are empty; C = at least one table or chart wrong (subsequent indices not corrected); D = forgot to correct 'heute' (today), a hard coded date; X = other; Z = nothing done but delivered. (A solution can have more than one error class.)

Productivity: When comparing time and points results, there is only one obvious conclusion: JG0 was more productive than the other two groups. Comparing JG3 and JG5 there is no significant difference in the productivity: ($pr(JG0) > pr(JG3) \approx pr(JG5)$).

Conclusion: From the time results $t(JG0) < t(JG3)$ and knowing from Section 2.8.2 that there are the same number of changes for both program versions we can conclude that the program Boerse3 is harder to understand than Boerse0. So Hypothesis 1 has to be rejected. Comparing the time results from Boerse5 with the other two groups it may seem strange that they are almost as fast as Boerse0. On a closer look, this can be explained: In Boerse5 there was only half as many of changes than in Boerse3 or Boerse0. So longer program comprehension time was compensated by less work to be done. We can conclude from this that Boerse5 was harder to understand than Boerse0 but we can not make a decision between Boerse3 and Boerse5. Considering the points obtained and the defects made, we see a trend that with deeper inheritance the number of defects is growing.

So for this work task we have to reject both hypotheses.

	Variable	Groups	Range min ... max	mean1	mean2	means difference (90% confid.)	significance p
1	JT time	G0, G3	5...76	15.9	17.5	-51%...24%	0.381
2	-	G0, G5	7...32	15.9	16.8	-23%...13%	0.313
3	-	G3, G5	5...76	17.5	16.8	-28%...46%	0.470
4	T1 time	G0, G3	32...194	71.1	88.1	-41%...3.0%	**0.075**
5	-	G0, G5	25...194	71.1	69.4	-16%...24%	0.447
6	-	G3, G5	25...185	88.1	69.4	4.4%...51%	**0.023**
7	-(10%)	G0, G3	32...125	62.7	78.2	-37%...-3.4%	**0.022**
8	-	G0, G5	25...85	62.7	67.0	-19%...5.9%	0.200
9	-	G3, G5	25...125	78.2	67.0	-2.2%...36%	**0.073**
10	-(20%)	G0, G3	32...107	60.0	72.2	-34%...-0.2%	**0.046**
11	-	G0, G5	25...82	60.0	64.7	-20%...5.9%	0.172
12	-	G3, G5	25...107	72.2	64.7	-6.5%...31%	0.153
13	T1 points	G0, G3	0 ...4	4.0	3.6	(1.5%...23%)	(0.000)
14	-	G0, G5	0 ...4	4.0	3.2	(9.3%...42%)	(0.000)
15	-	G3, G5	0 ...4	3.6	3.2	-7.1%...33%	0.142
16	T1 productivity	G0, G3	0...7	3.8	3.0	2.7%...54%	**0.030**
17	-	G0, G5	0...7	3.8	2.8	14%...64%	**0.003**
18	-	G3, G5	0...6	3.0	2.8	-20%...35%	0.339
19	T2 time	G0, G3	64...200	115.7	132.2	-24%...-0.8%	**0.038**
20	-	G0, G5	78...175	115.7	134.8	-23%...-5.1%	**0.005**
21	-	G3, G5	64...200	132.2	134.8	-14%...10%	0.393
22	-(10%)	G0, G3	64...173	112.1	124.1	-21%...1.3%	**0.071**
23	-	G0, G5	78...162	112.1	130.4	-23%...-4.7%	**0.006**
24	-	G3, G5	64...173	124.1	130.4	-16%...6.2%	0.239
25	-(20%)	G0, G3	64...152	108.4	118.6	-19%...2.4%	**0.097**
26	-	G0, G5	78...150	108.4	126.6	-23%...-5.2%	**0.005**
27	-	G3, G5	64...152	118.6	126.6	-18%...5.2%	0.185
28	T2 points	G0, G3	0...8	6.3	6.0	-15%...28%	0.313
29	-	G0, G5	0...8	6.3	6.4	-18%...17%	0.495
30	-	G3, G5	0...8	6.0	6.4	-26%...13%	0.299
31	T2 productivity	G0, G3	0...6	3.4	2.9	-11%...45%	0.156
32	-	G0, G5	0...6	3.4	3.0	-11%...36%	0.178
33	-	G3, G5	0...5	2.9	3.0	-29%...22%	0.412

Table 3.2: Results of the JAKK subjects; (left to right:) line number, name of variable, groups that are compared (group1, group2), smallest and biggest value in the combined sample, arithmetic average of group1, arithmetic average of group2, 90% confidence interval I for the difference group1 − group2 (measured in percent of group2), significance p of this difference (one sided). I and p were computed using resampling with 10000 trials. Parentheses around intervals and p values indicate dubious results due to zero variance in at least one of the samples. JT time means the time for answering the Java-test on the first questionnaire. T1 and T2 indicate tasks 1 and 2, respectively. (10%) or (20%) means the result that is obtained if 10 or 20 percent are cut off of the right side of the distribution to eliminate possible outliers.

Task 2 – "Time Interval Price Display and Gain/Loss Display"

Task 2 combines Tasks 2A and 2B as discussed in Section 2.8. We gathered timestamps only at the start and the end of the task but not between the two parts.

Time: For this task the time of group JG0 is significantly shorter than for the other two groups, on average 15 percent less: $t(JG0) < t(JG3) \approx t(JG5)$. Figures 3.4 and 3.5 show the time distribution for all subjects and by treatment group. In the group of Boerse3 some data points may be considered outliers so we recalculated the statistics with 10 and 20% cut off of the right side of the distribution. But the results did not change much. For details, see Table 3.2, lines 19 - 27.

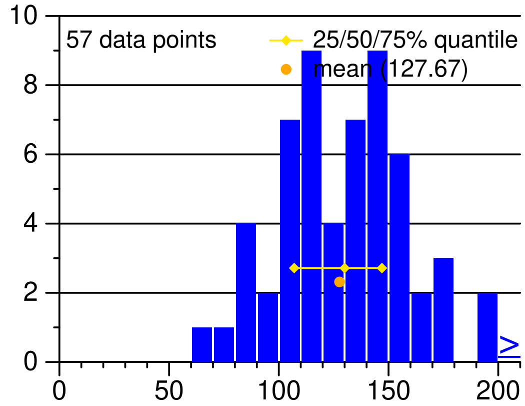


Figure 3.4: [JAKK] Distribution of the time required for all subjects for task "Time Interval Price Display and Gain/Loss Display".

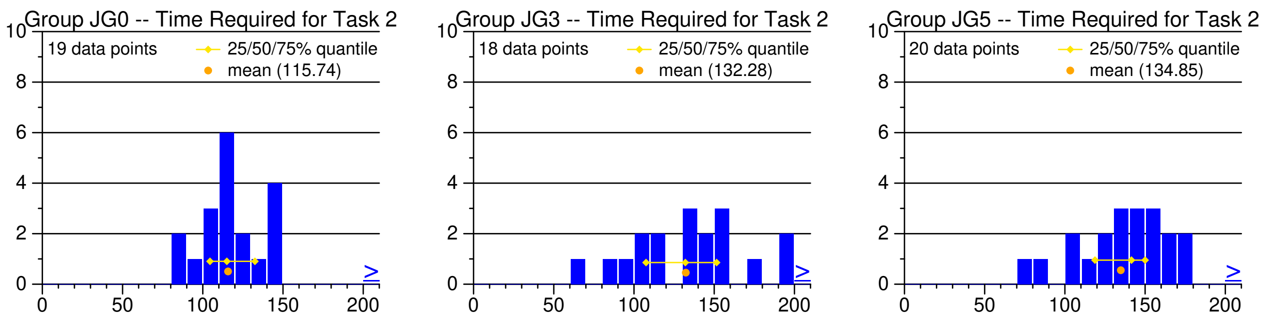


Figure 3.5: Distribution of the time required by group.

Points: No significant differences were observed between the groups, see Figures 3.6, 3.7, 3.8, and Table 3.2, lines 28 - 30.

Incidence counts: The number of correct solutions does not differ significantly between the groups, see Tables 3.3.

Types of solutions: For this task it is interesting to know which type of solution the subjects chose — whether they preserved the design or "destroyed" its structure.

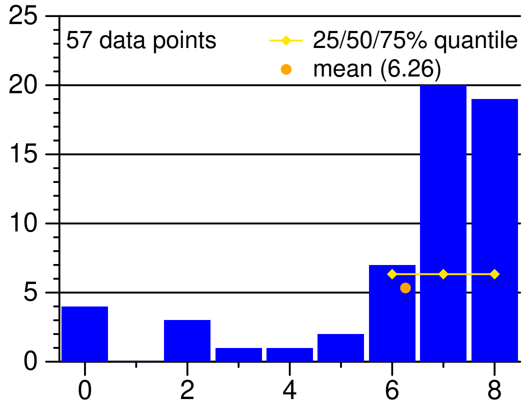


Figure 3.6: [JAKK] Distribution of the points for all subjects; the maximum achievable is 8.

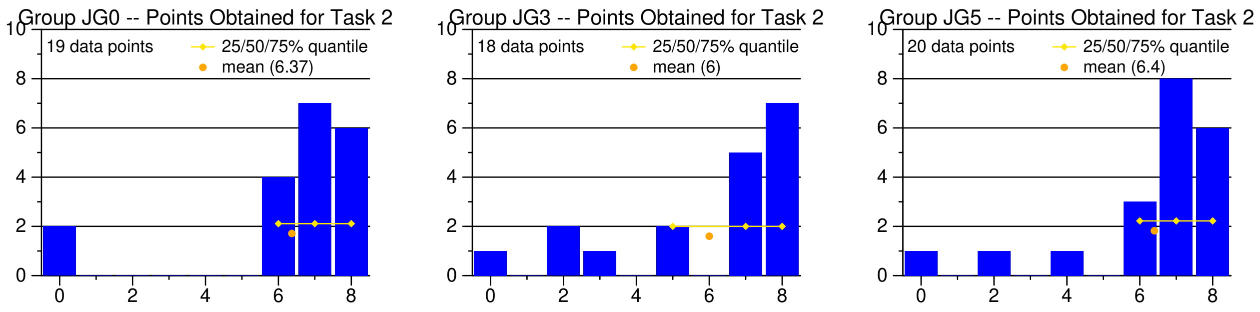


Figure 3.7: Distribution of points obtained by group.

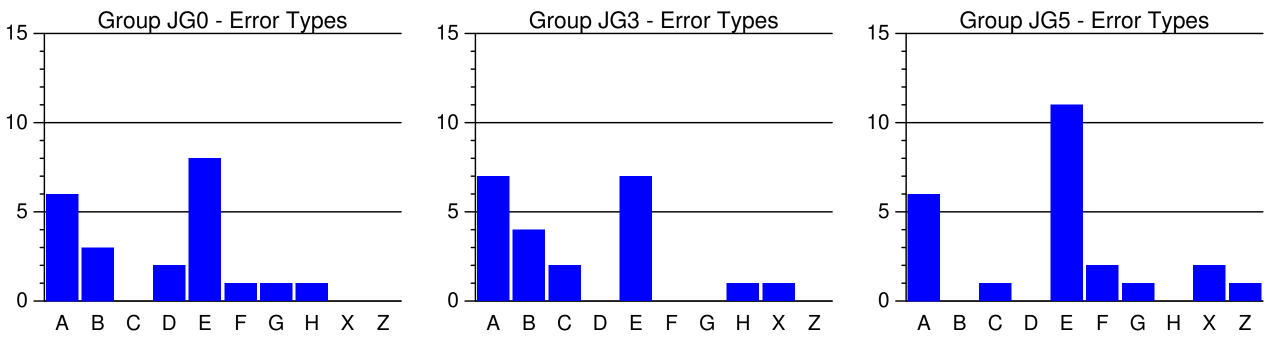


Figure 3.8: Frequency of different errors for task “Time Interval Price Display and Gain/Loss Display”. Codes: A = everything OK; B = one/both of the tables unimplemented; C = only dialog implemented; D = dialog not implemented; E = start day missing; F = end day missing; G = records appear twice; H = uses KURSE instead of KASSAKURSE; X = other; Z = nothing done but delivered.(A solution can have more than one error class.)

group	no. of subjects	fraction of correct solutions
JG0	19	37%
JG3	18	39%
JG5	20	30%

groups	χ^2	p	Fisher exact p
JG0 - JG3	0.044	0.83	1
JG0 - JG5	0.013	0.91	0.74
JG3 - JG5	0.055	0.81	0.73

Table 3.3: [JAKK] Fraction of correct solutions per group (left) and significance p of the group differences (right) for Task 2.

Task 2A, JG0: 17 out of 19 subjects from group JG0 chose the solution type as described in Section 2.8.3 and thus preserved the structure: A new class is created as a copy of an existing table class. Only 2 subjects inherited from an existing class and overwrote the functionality that had to be changed.

Task 2B, JG0: 15 out of 17 created the solution by duplicating and modifying an existing class. Only 2 subjects inherited from the class implemented in Task 2A.

Task 2A, JG3: 11 out of 18 subjects retained the structure and inherited from the abstract table class while 5 of the subjects inherited from a concrete class that is derived from the abstract table class and 2 subjects inherited from the abstract common superclass of the charts and the tables.

Task 2B, JG3: 7 subjects out of 15 inherited from the abstract table class, 2 subjects inherited from the class implemented in Task 2A, 5 subjects inherited form a concrete table class that has similar functionality but this destroys the design structure (the design decision was: only inheriting from abstract classes), and one subject inherited from the abstract class that is the superclass of the charts and the tables.

Task 2A, JG5: 18 out of 20 subjects from group JG5 extended the abstract class for time interval table displays and added the selection dialog functionality for user defined time intervals, 1 subject extended a concrete table class, and 1 subject extended the Java library class Dialog and tried to implement the class from scratch.

Task 2B, JG5: 9 subjects out of 18 inherited from the class implemented in the first part and the other 9 subjects inherited form a concrete table class that has similar functionality.

	Task 2A		Task2B	
	group size	fraction of subjects retaining the structure	group size	fraction of subjects retaining the structure
JG0	19	89%	17	88%
JG3	18	61%	15	47%
JG5	20	90%	18	100%

Table 3.4: [JAKK] Fraction of subjects that retained the design structure. The group sizes differ between Task 2A and Task 2B because of mortality in the groups.

Conclusion: Differences in time required are quite small between groups JG3 and JG5; JG5 required on average 2 percent more time. This is plausible: For this task it was only necessary to find the differences between the classes “MonatsTabelle [MonthTable]” and “Chart”. In Boerse3 as well as in Boerse5 the code had to be found across several hierarchy levels; in Boerse5 a few more superclasses had to be taken into consideration. Once the differences between the two classes were detected, the exercise could be solved easily by inheritance and/or copy-and-paste in all three groups. The actual changes were similar for all for all three groups, so we can assume that the time differences result from program comprehension. As in Task 1 we can conclude that with increasing inheritance depth comprehension tends to become more difficult and the time required is increasing correspondingly.

3.2.4 Performance on the tasks - Informatik II

Keep in mind that the tasks are not the same as for the JAKK course. Task 1 from the JAKK course is skipped and Task 2 is split into two tasks, called Task 2A and 2B. We did this because of the time constraints described in Section 2.7.

Task 2A – “Time Interval Price Data Display”

	Variable	Groups	min … max	mean1	mean2	means difference (90% confid.)	significance p
1	JT time	G0, G3	5 … 20	12.7	13.1	-19%… 11%	0.349
2	-	G0, G5	5 … 25	12.7	13.2	-19%… 11%	0.337
3	-	G3, G5	7 … 25	13.1	13.2	-16%… 15%	0.487
4	T2A time	G0, G3	82 … 305	152.3	154.0	-19%… 16%	0.456
5	-	G0, G5	93 … 330	152.3	180.8	-31%… -0.9%	**0.038**
6	-	G3, G5	82 … 330	154.0	180.8	-30%… 0.5%	**0.055**
7	-(10%)	G0, G3	82 … 187	138.7	139.8	-13%… 11%	0.460
8	-	G0, G5	93 … 238	138.7	168.3	-28%… -7.0%	**0.003**
9	-	G3, G5	82 … 238	139.8	168.3	-29%… -5.6%	**0.007**
10	-(20%)	G0, G3	82 … 179	133.6	136.4	-14%… 9.9%	0.386
11	-	G0, G5	93 … 219	133.6	161.1	-28%… -6.7%	**0.003**
12	-	G3, G5	82 … 219	136.4	161.1	-27%… -4.0%	**0.012**
13	T2A points	G0, G3	0 … 8	5.9	5.4	-20%… 39%	0.295
14	-	G0, G5	0 … 8	5.9	5.0	-14%… 47%	0.185
15	-	G3, G5	0 … 8	5.4	5.0	-27%… 38%	0.374
16	T2A productivity	G0, G3	0 … 5	2.6	2.4	-26%… 47%	0.319
17	-	G0, G5	0 … 5	2.6	1.9	-0.3%… 81%	**0.051**
18	-	G3, G5	0 … 5	2.4	1.9	-16%… 72%	0.144
19	T2B time	G0, G3	10 … 84	27.0	30.9	-47%… 18%	0.263
20	-	G0, G5	11 … 64	27.0	19.1	12%… 72%	**0.006**
21	-	G3, G5	10 … 84	30.9	19.1	19%… 111%	**0.004**
22	-(10%)	G0, G3	10 … 41	22.9	24.0	-28%… 18%	0.385
23	-	G0, G5	11 … 38	22.9	18.2	4.2%… 47%	**0.021**
24	-	G3, G5	10 … 41	24.0	18.2	5.0%… 58%	**0.021**
25	-(20%)	G0, G3	10 … 41	20.9	22.4	-30%… 15%	0.311
26	-	G0, G5	11 … 31	20.9	17.3	0.9%… 41%	**0.042**
27	-	G3, G5	10 … 41	22.4	17.3	3.8%… 56%	**0.027**
28	T2B points	G0, G3	0 … 3	2.3	2.2	-24%… 30%	0.422
29	-	G0, G5	0 … 3	2.3	2.5	-30%… 19%	0.349
30	-	G3, G5	0 … 3	2.2	2.5	-34%… 17%	0.298
31	T2B productivity	G0, G3	0 … 60	14.5	16.0	-79%… 59%	0.404
32	-	G0, G5	0 … 60	14.5	20.9	-81%… 21%	0.167
33	-	G3, G5	0 … 60	16.0	20.9	-77%… 32%	0.244

Table 3.5: Results of the Informatik II subjects; (left to right:) line number, name of variable, groups that are compared (group1, group2), smallest and biggest value in the combined sample, arithmetic average of group1, arithmetic average of group2, 90% confidence interval I for difference group1 $-$ group2 (measured in percent of group2), significance p of the difference (one sided). I and p were computed using resampling with 10000 trials. JT time means the time for answering the Java-test on the first questionnaire. T2A means Task 2A, “Time Interval Price Data Display”, T2B means Task 2B, “Time Interval Gain/Loss Display”, (10%) or (20%) means the result that is obtained if 10 or 20 percent are cut off of the right side of the distribution to eliminate possible outliers.

Time: Regarding the average time of the groups there is one obvious conclusion: IG5 needed more time ($\approx$ 18 percent more) than IG3 and IG0: $t(IG0) \approx t(IG3) < t(IG5)$. The Histograms 3.9 and 3.10 suggest some data points to be outliers. However, by cutting the right end (10% and 20%) off of the distribution of each group, we still get similar results. Subjects of IG5 needed significantly much more time than the other two groups.

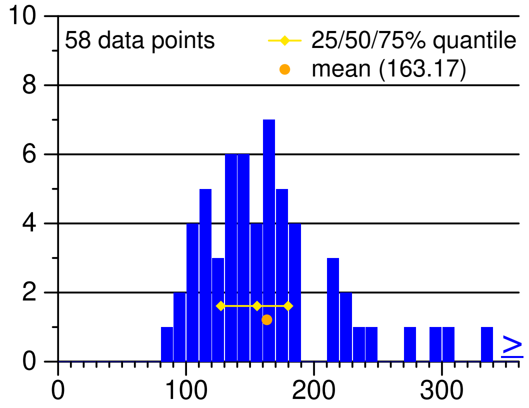


Figure 3.9: [Informatik II] Distribution of the time required for the task "Time Interval Price Display".

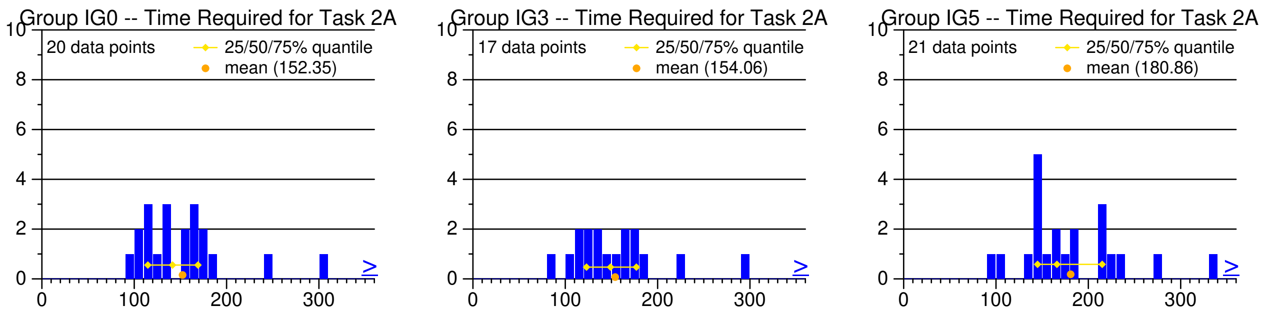


Figure 3.10: [Informatik II] Distribution of the time required for the task "Time Interval Price Display" for each of the three treatment groups.

Points: No significant quality differences can be observed between the three groups. The number of correct solutions in each group is not significantly different, either; see Figures 3.11, 3.12, 3.13 and Table 3.6.

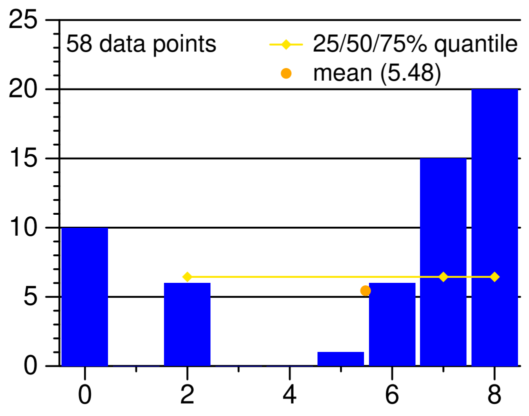


Figure 3.11: [Informatik II] Distribution of the points for all subjects; the maximum achievable is 8.

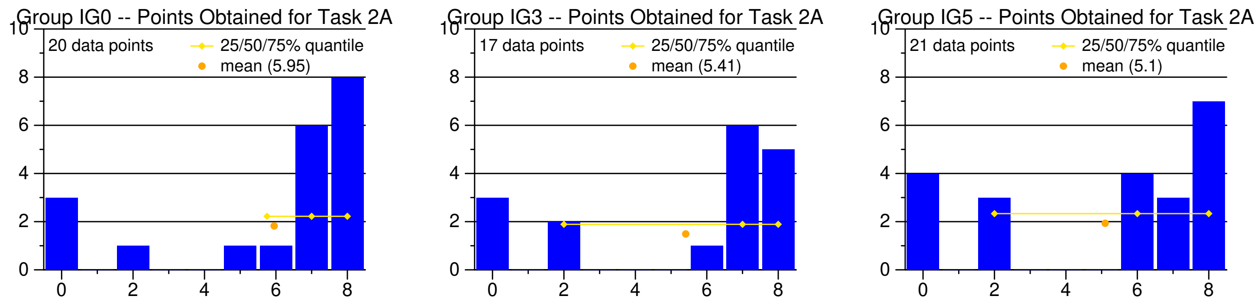


Figure 3.12: [Informatik II] Distribution of points obtained by group.

group	no. of subjects	fraction of correct solutions	groups	χ^2	p	Fisher exact p
IG0	20	40%	IG0 - IG3	0.107	0.74	0.73
IG3	17	29%	IG0 - IG5	0.014	0.91	0.75
IG5	21	33%	IG3 - IG5	0.009	0.93	1

Table 3.6: [Informatik II] Fraction of correct solutions per group (left) and significance p of the group differences (right) for Task 2A.

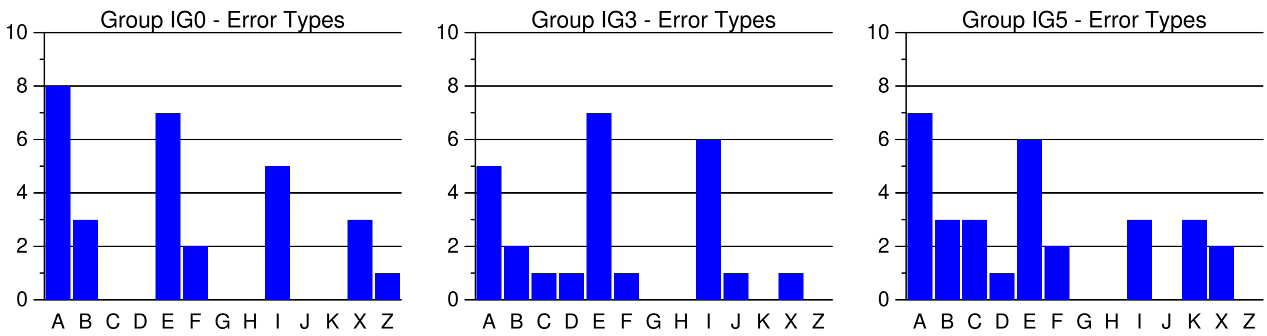


Figure 3.13: [Informatik II] Frequency of different errors for task "Time Interval Price Display". Codes: A = everything OK; B = table unimplemented, only checkbox appears; C = only dialog implemented, no table appears; D = dialog unimplemented, some other table with new time range appears; E = start day missing; F = end day missing; G = records appear twice; H = uses KURSE instead of KASSAKURSE; I = requires two-digit years; J = start day wrong (e.g., wrong year), but not just E; K = end day wrong (e.g., wrong year), but not just F; X = other; Z = nothing done but delivered. (A solution can have more than one error class.)

Type of solutions: For this task, it is interesting to know which type of solution the subjects chose – whether they preserved the design or "destroyed" its structure.

Task 2A, IG0: 17 out of 20 chose the solution type as described in Section 2.8. A new class is created as a copy of an existing table class. Only 2 subjects inherited from an existing class and overwrote the functionality that had to be changed. (1 subject did not deliver a solution.)

Task 2A, IG3: 14 out of 17 subjects retained the structure and inherited from the abstract table class, 2 subjects retained the structure, but inherited from the abstract class that is the superclass of the charts and the tables. Only 1 subject inherited from a concrete table display class.

Task 2A, IG5: 17 out of 21 subjects extended the abstract class for fixed time interval table displays and added the selection dialog functionality for user defined time intervals, 3 subjects also extended an abstract class but only for table displays which is the superclass for the fixed time interval table display. Only 1 subject inherited from the abstract class that is the superclass of the charts and the tables.

	group size	fraction of subjects retaining the structure
JG0	20	85%
JG3	17	82%
JG5	21	81%

Table 3.7: [Informatik II] Fraction of subjects that retained the design structure.

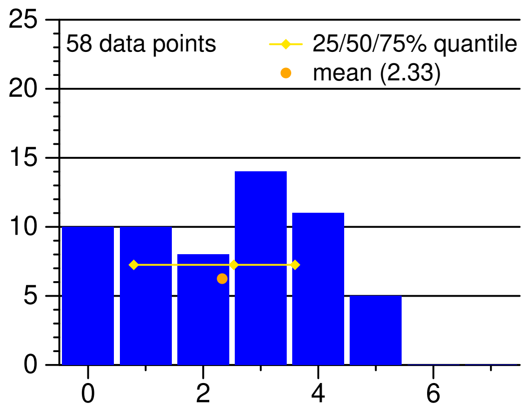


Figure 3.14: [Informatik II] Distribution of the productivity for all subjects.

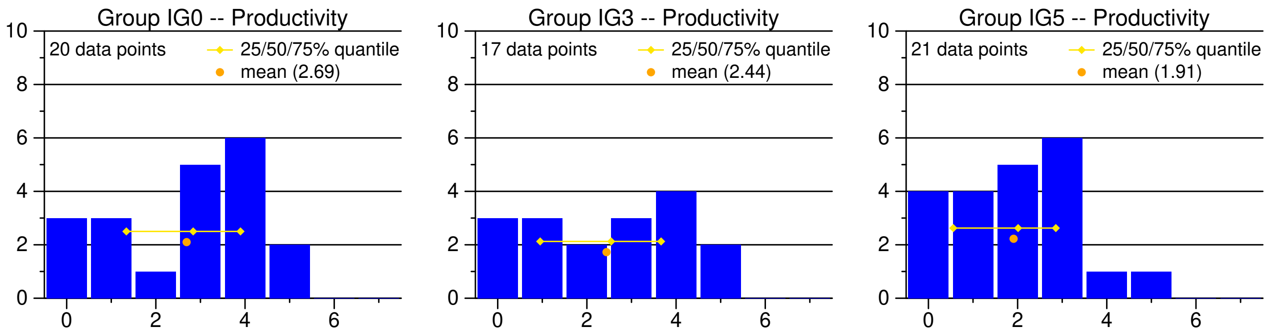


Figure 3.15: [Informatik II] Distribution of the productivity by group.

Conclusion: In contrast to the work task of the JAKK experiment with its “warm-up” task 1, all program comprehension must be gained in this task. But we still observe the same trend as in the JAKK run: with an increasing inheritance hierarchy the maintenance time is increasing, too. The quality of the solutions does not differ much between the groups so we have to reject both of our hypotheses.

Task 2B – “Time Interval Gain/Loss Display”

Task 2B differs from all other tasks in this experiment – the subjects just worked on a quite similar task and gained almost the complete program understanding required for this task beforehands in Task 2A. So the time results might give information about the working time required for the different program versions.

Caveat: The time results presented for this task are quite unreliable. When looking on the mortality it is salient that the mortality in group IG5 is higher than in the two other groups (IG5 = 29%; IG3 = 12%; IG0 = 10%). The table below lists for each subject the time for the Task 2A so that it is obvious that more slow subjects quit in IG5 than in IG3 and IG0. From these considerations we expect that more faster subjects remained in the group IG5 and that the groups are not balanced anymore.

Conclusion: We may conclude for this task that more subjects working on a program with a deep inheritance structure found it too difficult than in the other two groups.

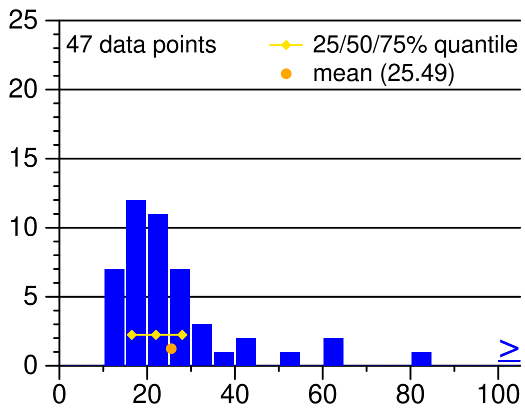


Figure 3.16: [Informatik II] Distribution of the time required for Task 2B for all subjects.

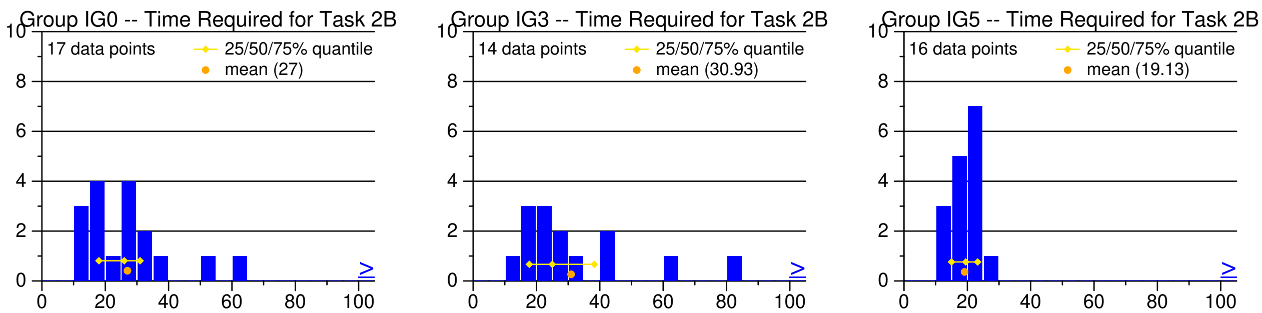


Figure 3.17: [Informatik II] Distribution of the time required by group.

	IG0	IG3	IG5
1	96	82	93
2	102	101	109
3	108	110	**138**
4	110	112	140
5	111	123	143
6	116	126	145
7	123	131	147
8	131	135	149
9	133	**149**	157
10	133	***157***	166
11	***150***	163	166
12	154	165	179
13	162	177	***180***
14	167	179	**182**
15	**167**	187	212
16	**175**	**223**	**215**
17	179	299	219
18	180		220
19	245		**238**
20	305		**270**
21			**330**

Table 3.8: [Informatik II] Times required for each subject for Task 2A. Boldfaced and underlined entries are for subjects which gave up in the second task. Times marked with *** are the time closest to the mean in the group. From this table we can see that the mortality is unbalanced.

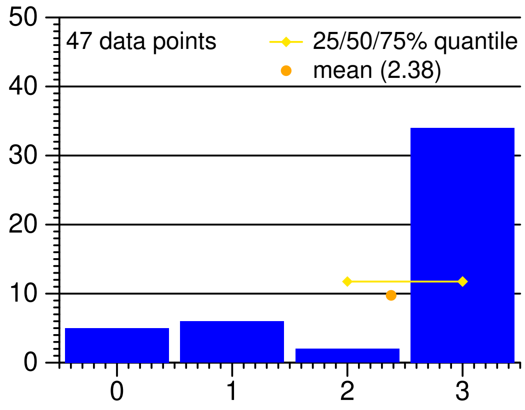


Figure 3.18: [Informatik II] Distribution of the obtained points for all subjects; the maximum is 3.

group	no. of subjects	fraction of correct solutions
IG0	20	35%
IG3	17	24%
IG5	21	29%

groups	χ^2	p	Fisher exact p
IG0 - IG3	0.16	0.69	0.5
IG0 - IG5	0.011	0.92	0.74
IG3 - IG5	0.0	0.98	1

Table 3.9: [Informatik II] Fraction of correct solutions per group (left) and significance p of the group differences (right) for Task 2B.

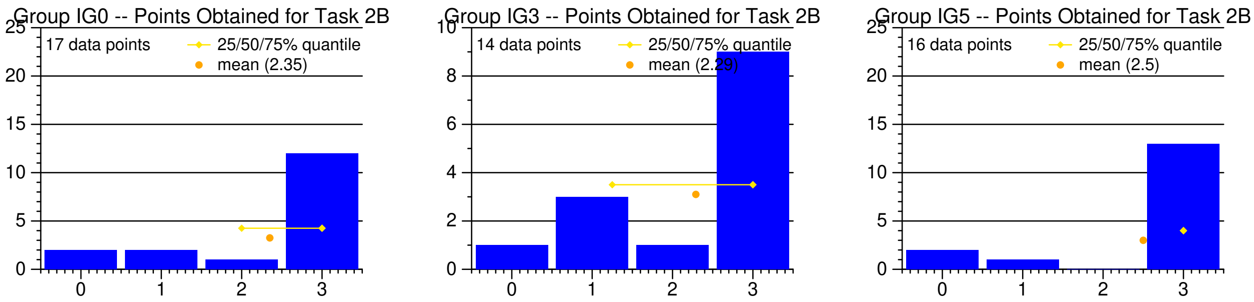


Figure 3.19: [Informatik II] Distribution of the points obtained by group.

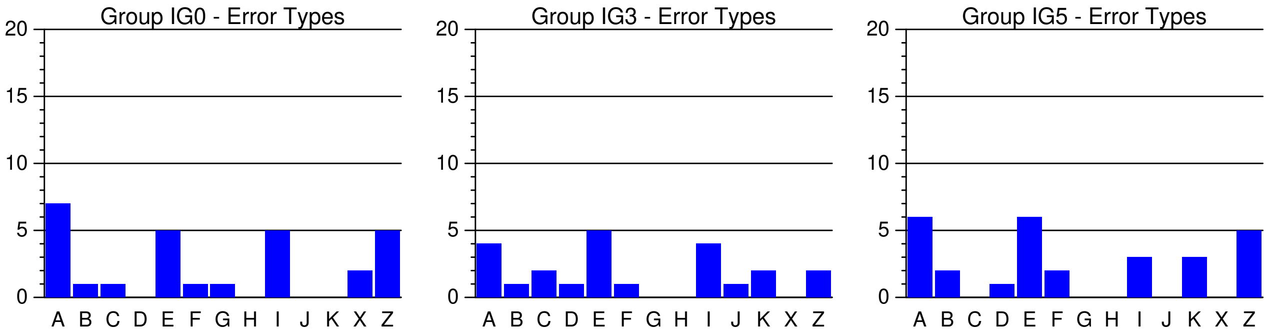


Figure 3.20: [Informatik II] Frequency of different errors for task “Time Interval Gain/Loss Display”. Codes: A = everything OK; B = table unimplemented, only checkbox appears; C = only dialog implemented, no table appears; D = dialog unimplemented, some other table with new time range appears; E = start day missing (but was correct in 2A, otherwise no deduction); F = end day missing (but was correct in 2A, otherwise no deduction); G = records appear twice (but was correct in 2A, otherwise no deduction); H = uses KURSE instead of KASSAKURSE (but was correct in 2A, otherwise no deduction); I = requires two-digit years; J = start day wrong (e.g., wrong year), but not only E; K = end day wrong (e.g., wrong year), but not only F; X = other; Z = nothing done but delivered. (A solution can have more than one error class.)

3.3 Subjects' experience

In the final questionnaire we wanted to learn about our subjects' impressions: how they assess, for example, the program structure, the use of inheritance, the difficulty of the tasks, their concentration, etc.

3.3.1 Inheritance use

With the first question we wanted to know how the subjects assess the use of inheritance. Our subjects had five choices from "inheritance is used much too much" to "inheritance is used much too little". The median of the answers from the JAKK subjects were "inheritance is used too little" for group JG0 and "inheritance is used in the right degree" for the groups JG3 and JG5. The distribution is shown in Figure 3.21.

There is an obvious difference between the performance of the subjects and the subjective impression they have. Regarding that JG0 was on average the best and fastest group, it is curious that most of these subjects graded the inheritance use as "too little" or "much too little". The discrepancy between subjective impression and objective results demonstrates how necessary controlled experiments are.

We have similar answers from the Informatik II subjects, except for IG0. These subject graded the inheritance use as "used in the right degree" (median).

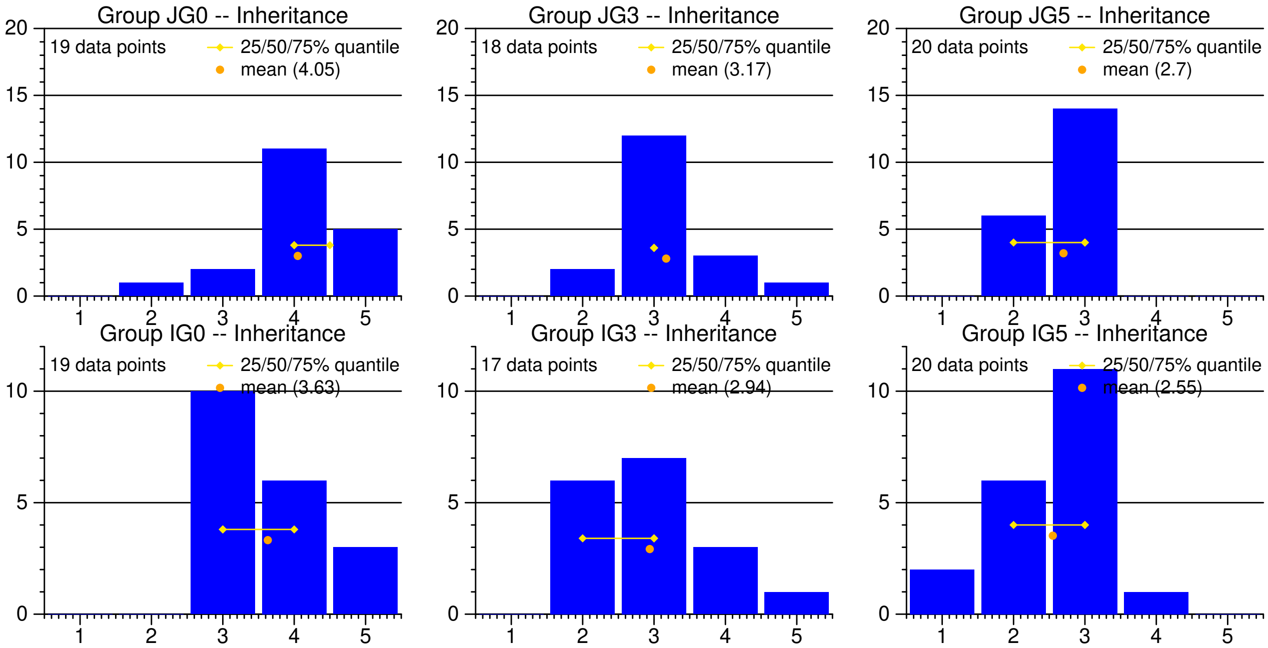


Figure 3.21: Answers from the JAKK subjects (first row) and for the Informatik II subjects (second row) on how they assess the use of inheritance, split into treatment groups. The subjects had the following choices: Inheritance is used much too much (1), is used too much (2), is used in the right degree (3), is used too little (4), or is used much too little (5).

3.3.2 Program structure

In the next question the subjects were asked how they rated the quality of the program structure for a program of this size. They had five choices from "very clear structure" to "unclear structure". The median of the answers of the JAKK and Informatik II subjects were between “clear structure’ and “medium structure’. The distribution is shown in Figure 3.22. We expected that the group JG0 would grade the structure as less clear than the other groups because of the code replication and the resulting code extension. However, comparing the actual subjective impressions reported with the objective results, it is odd that more than half of the subjects of JG5 graded the structure as a ”very clear structure” or ”clear structure”, while of JG3 and JG0 less than half graded it as ”clear or very clear structure”, even though they received better results. Probably the judgement was mostly based on the OMT diagrams, where JG5 clearly appears the most structured. Overall, the Informatik II subjects judged the structure more realistically than the JAKK subjects.

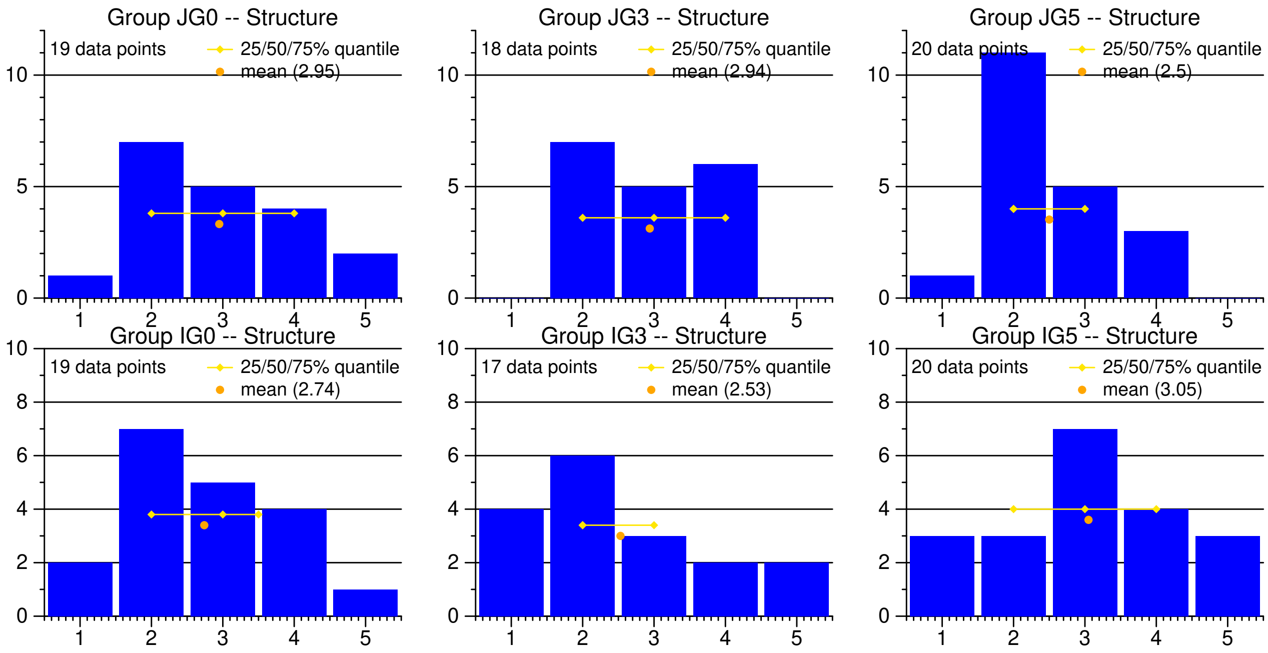


Figure 3.22: Answers from the JAKK subjects (first row) and for the Informatik II subjects (second row) on how they assess the structure of the program, split into treatment groups. The subjects had the following choices: very clear structure (1), clear structure (2), medium structure (3), less clear structure(4), unclear structure (5).

3.3.3 Simplicity of the tasks

This question investigated how difficult the subjects found the tasks. We expected from the time results obtained and the comprehension steps counted that the group with the flat program version found the tasks easier than the other groups but no large differences between the groups were observed. The distributions are shown in Figures 3.23 to 3.24.

3.3.4 Concentration of the subjects

How well the subjects were subjectively able to concentrate on the tasks is shown in Figures 3.25 to 3.26.The median of all groups was that their concentration was “high”, except for the inheritance groups of the JAKK experiment in Task 2. They probably realized that they had problems with tracing the methods to gain program understanding and they put it down to their concentration ability.

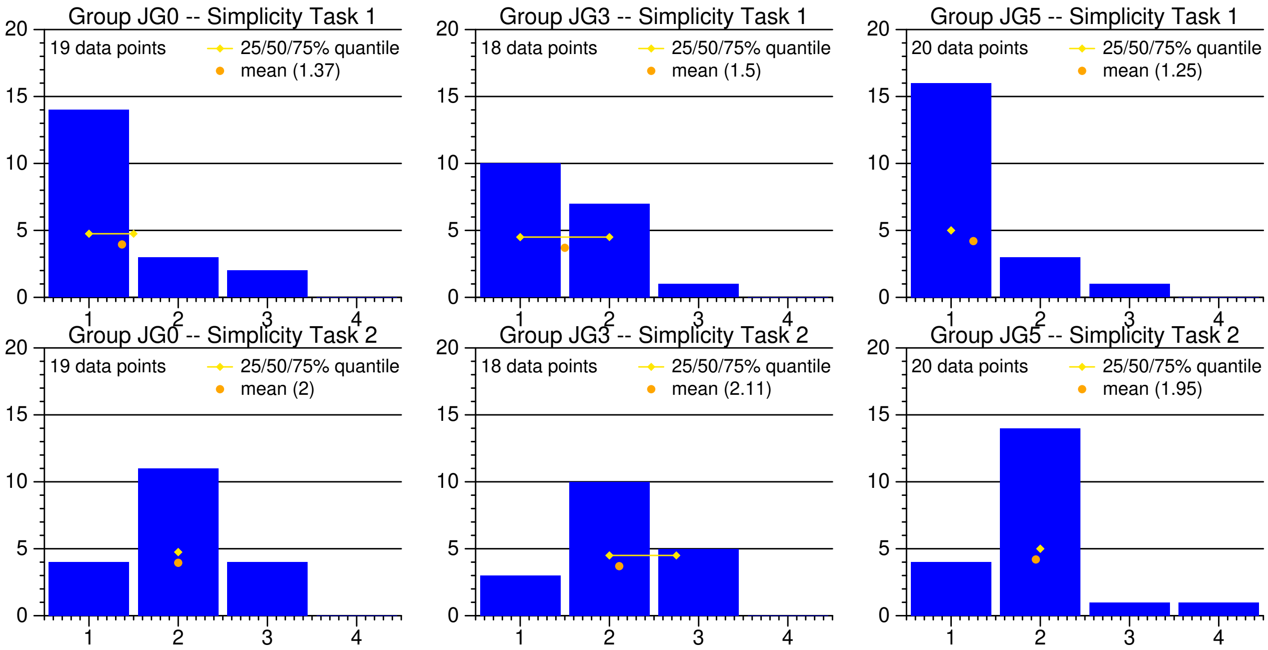


Figure 3.23: Answers from the JAKK subjects on how they assess the simplicity of Task 1 (first row) and Task 2 (second row) of the experiment, split into treatment groups. The subjects had the following choices: pretty simple (1), not quite so simple (2), pretty difficult (3), difficult (4).

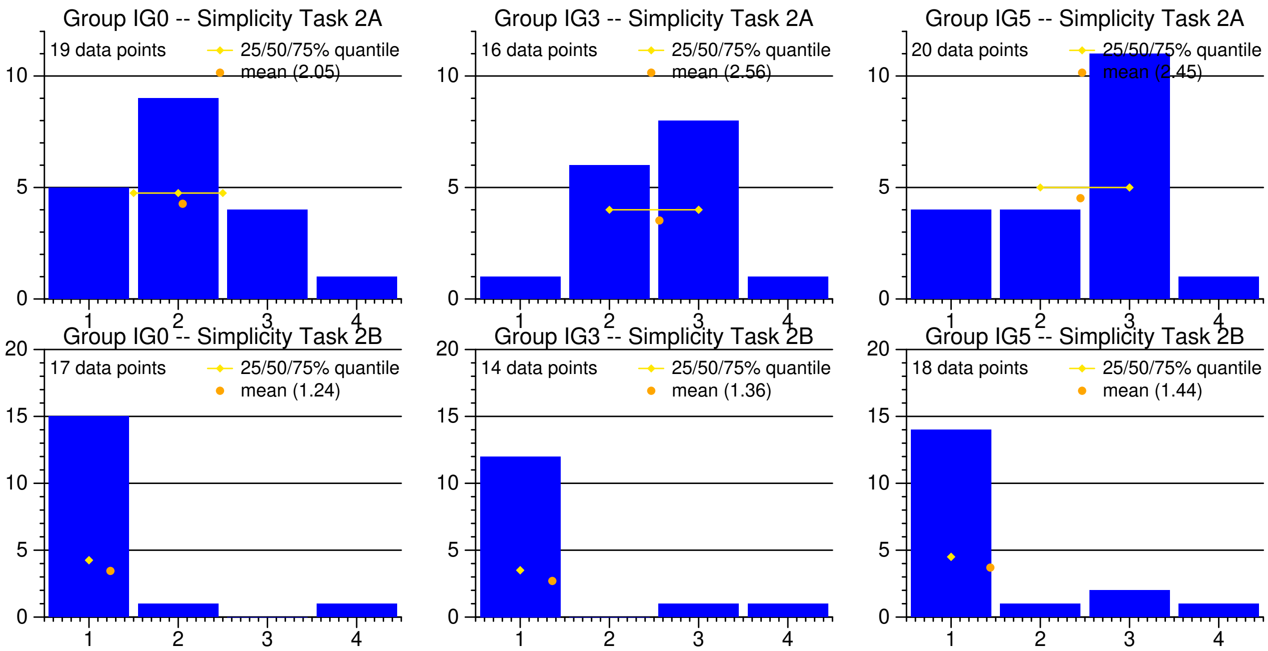


Figure 3.24: Answers from the Informatik II subjects on how they assess the simplicity of Task 2A (first row) and Task 2B (second row) of the experiment, split into treatment groups. For a legend see Figure 3.23.

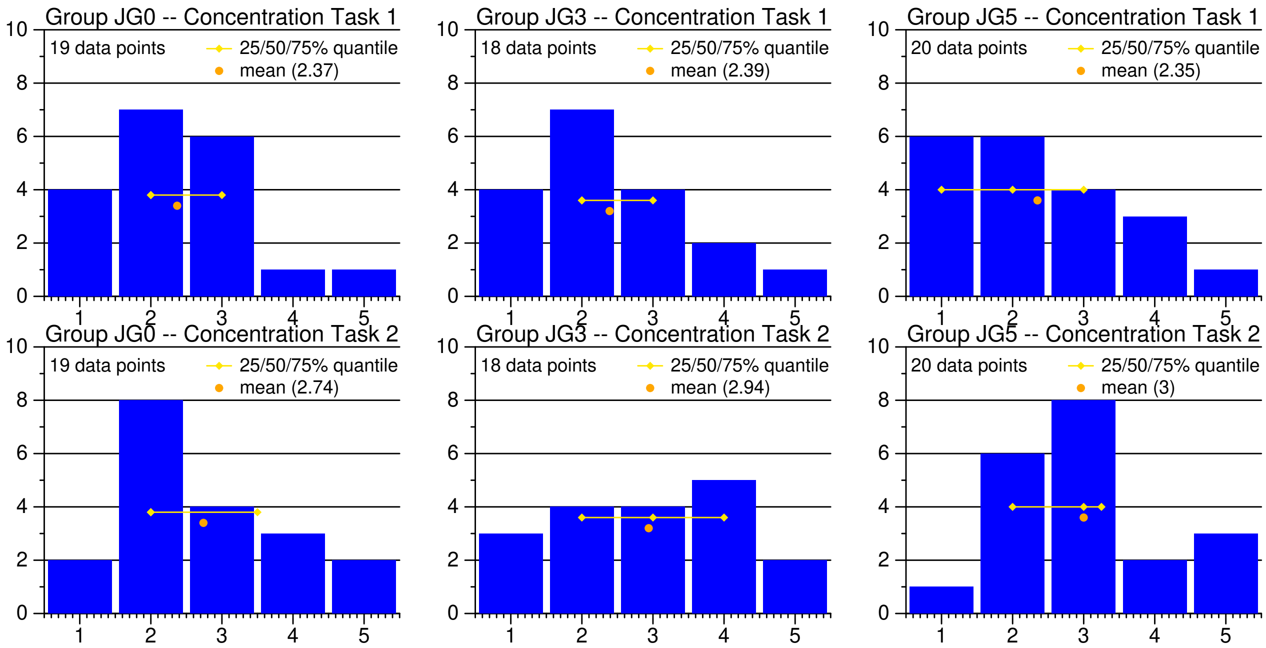


Figure 3.25: Answers from the JAKK subjects on how they assess their ability of concentration during Task 1 (first row) and Task 2 (second row), split into treatment groups. The subjects had the following choices: very high (1), high (2), OK (3), not so high (4), low (5).

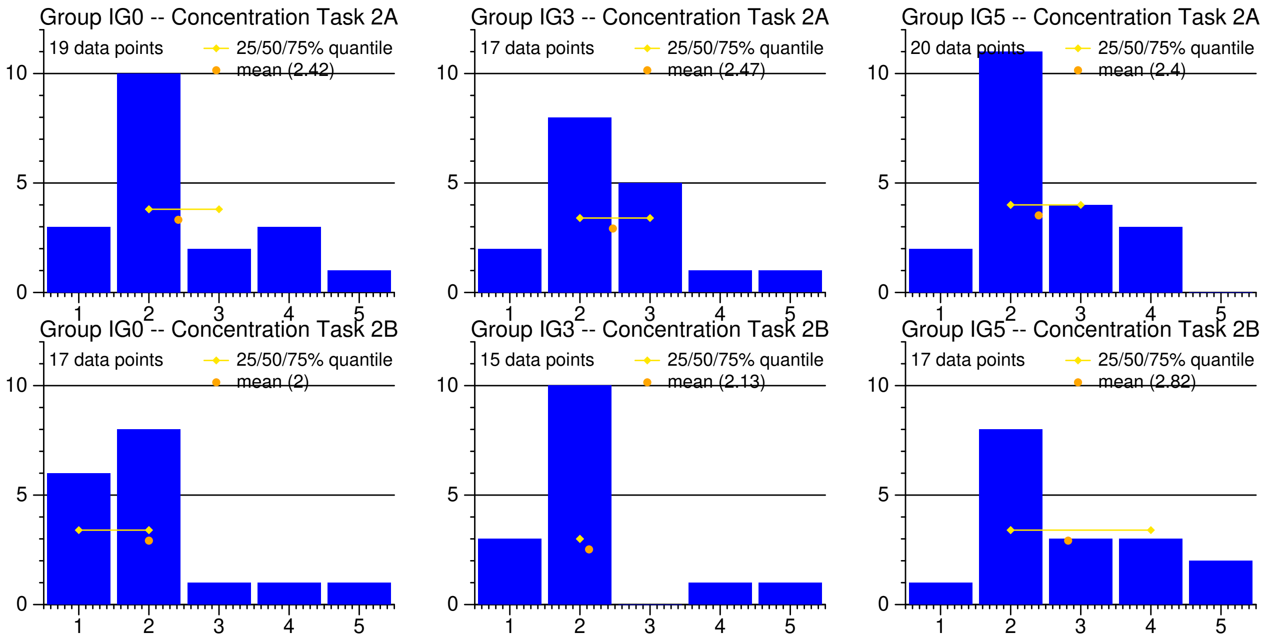


Figure 3.26: Ditto for the Informatik II subjects for Task 2A (first row) and Task 2B (second row).

3.3.5 Quality assessment

The subjects were asked how they subjectively assess the quality of their solutions to the two tasks. For each task they had the following choices of answers: no more errors or omissions (1), at most one error or omission (2), probably several errors or omissions (3), or don't know (4). G0 had most confidence in correctness(!) except for IG5 in Task 2B (but in Task 2B mortality distorts these results, anyway). This is unexpected from the hypotheses but expected from examining the tasks: G0 had all code local in a class and that resulted in a better overview of what they were doing. The distributions are shown in the Figures 3.27 and 3.28.

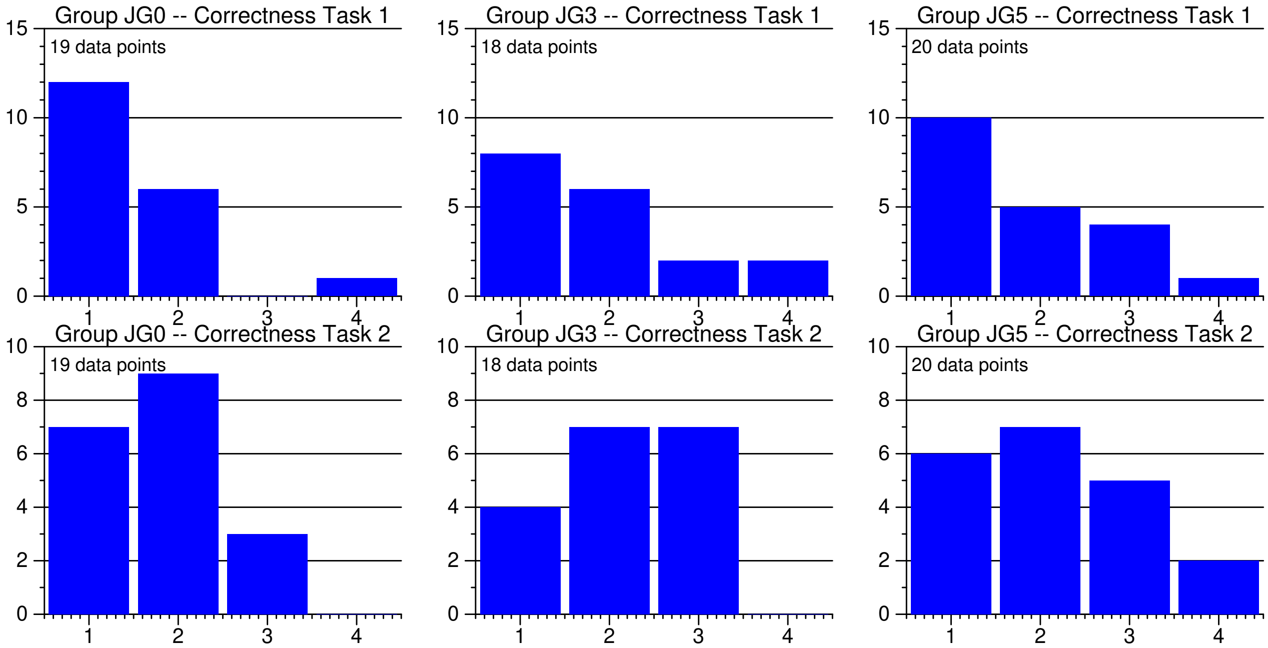


Figure 3.27: Answers from the JAKK subjects on how they assess their solutions for Task 1 (first row) and Task 2 (second row), split into treatment groups. The subjects had the following choices: no more errors or omissions (1), at most one error or omission (2), probably several errors or omissions (3), don't know (4).

3.3.6 Usefulness of OO knowledge

We asked the subjects whether they think that their OO knowledge was helpful. We expected that the subjects found it the more helpful the more inheritance is used in the program. They had the following choices of answers: no, not at all (1), only a little (2), can't decide (3), yes, somewhat (4), or yes, very much (5). We expected to observe a huge difference between the flat program version and the deep inheritance version because no inheritance knowledge is needed in the flat program version but for understanding Boerse3 and Boerse5 this knowledge is essential. We were surprised that no large differences were observed between the groups, see Figures 3.29 and 3.30. We speculate that to a large degree the concept of "OO knowledge" was equated with general knowledge of Java and its API, which is roughly as relevant in all three groups.

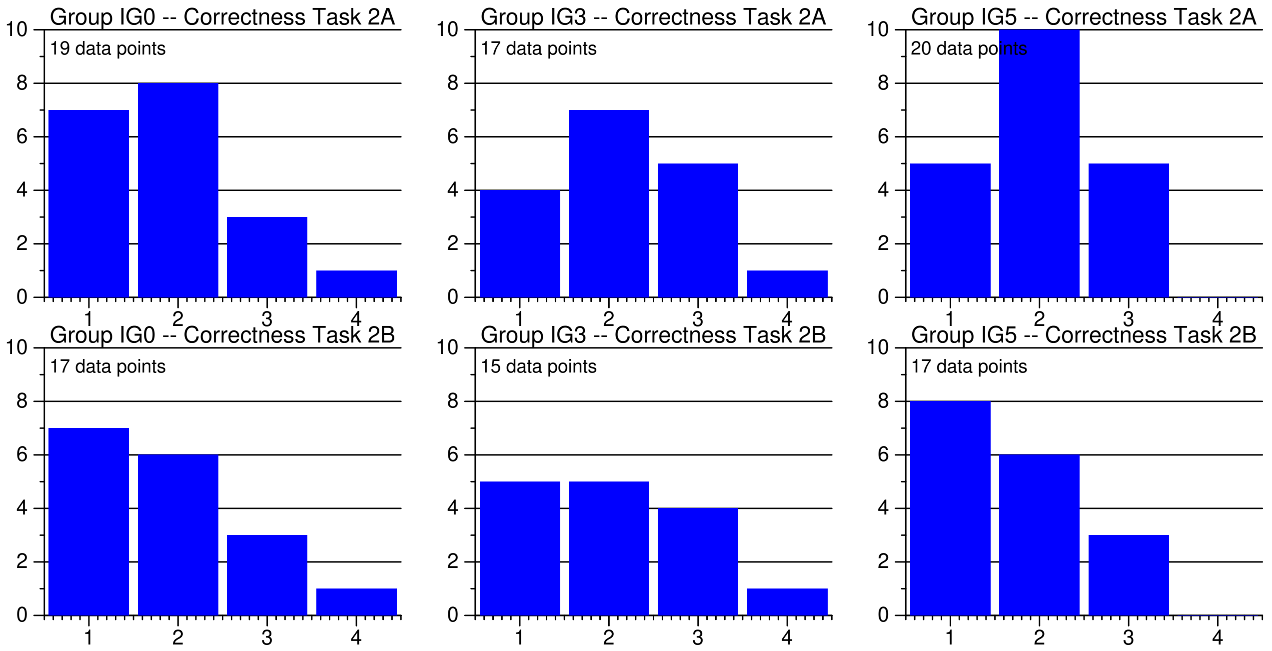


Figure 3.28: Ditto for the Informatik II subjects for Task 2A (first row) and Task 2B (second row).

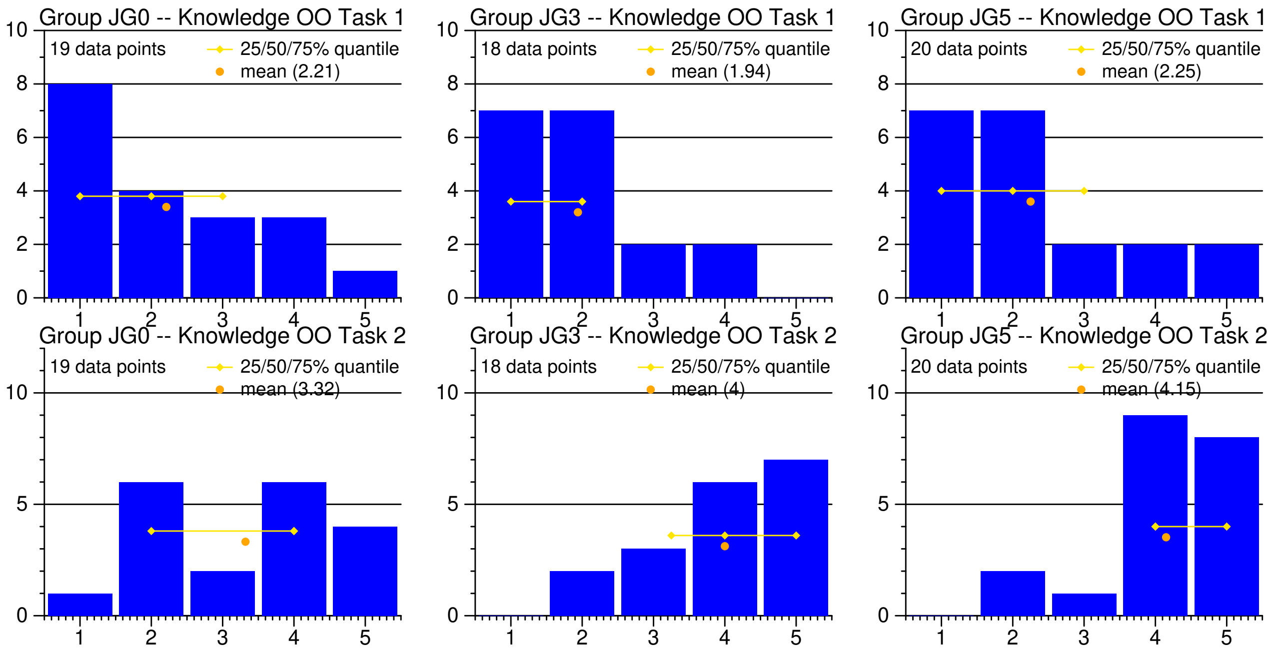


Figure 3.29: Answers from the JAKK subjects on how they assess the usefulness of OO knowledge for Task 1 (first row) and Task 2 (second row), split into treatment groups. The subjects had the following choices: no, not at all (1), only a little (2), can't decide (3), yes, somewhat (4), yes, very much (5).

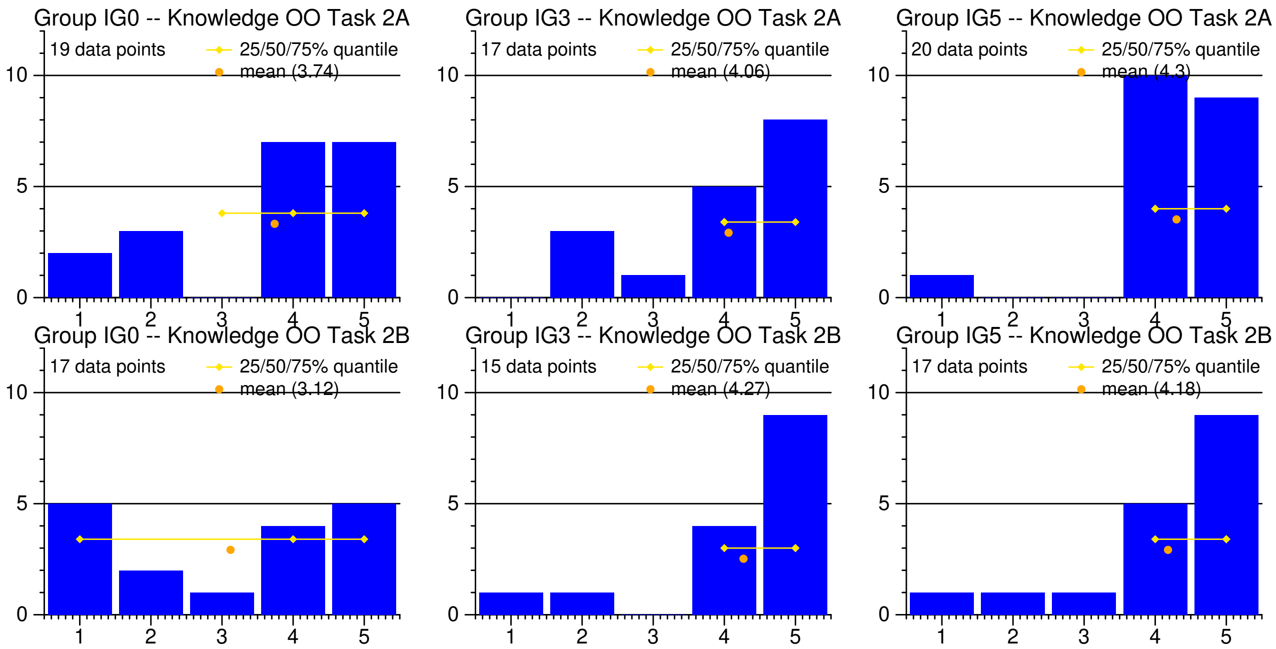


Figure 3.30: Ditto for the Informatik II subjects for Task 2A (first row) and Task 2B (second row).

Chapter 4

Analysis of the different results of the previous and our new experiment

The results found in this experiment and the results found by Daly et al. contradict. Daly et al. found that the program with inheritance depth of 3 is faster to maintain than the flat version and that the time for maintaining the program with inheritance depth of 5 is not significantly different from the equivalent flat program. In contrast, we found that for Task 2A (Informatik II) and Task 2 (JAKK) subjects with an inheritance program are slower than subjects maintaining the flat program. Task 2B could not be assessed because of the high mortality rate in the group with the deep inheritance hierarchy. For Task 1 the fastest group is the group with the deepest inheritance hierarchy (and hence the smallest number of places needing change).

What causes the differences?

Examining the programs of Daly et al. and our programs, we find several differences. Program length, number of classes, and number of methods differ, see Tables 1.1 and 2.1. The programs and maintenance tasks do not only differ in these respects, they also differ in complexity. Trying to find a measure for complexity we counted the number of methods that have to be understood to perform the task and the distance of the methods to their invocation, e. g. if a method M is invoked in the main program on an object of type C but this inherits the method from class A, that is the super-superclass of C, we count one external method invocation (call from main), two hierarchy changes (switch to super-superclass) and one method to trace (namely M). This is shown in Table 4.1. The process of gaining program understanding is sketched in Figures 2.35 and 2.36 on page 25 for Task 2 of our experiment and in Figure 4.1 for the original experiment. Each arrow across horizontal lines means that the method that is invoked is located in one of the superclasses or one of the subclasses.

	Boerse0	Boerse3	Boerse5	Univ0(3)	Univ3	Univ0(5)	Univ5
methods to trace	15	18	19	3	5	4	7
hierarchy changes	0	17	21	0	2	0	5
external method invocations	3	3	3	2	2	2	2

Table 4.1: Comparison of task complexity between our experiment and the ones of Daly et al. *Methods to trace* is the number of methods that must be investigated for gaining enough program understanding for solving the tasks. *hierarchy changes* is the number of inheritance steps that have to be followed to find the required methods (see Figure 4.1, and *external method invocations* are method invocations from outside the main hierarchy (e.g. from the main program).

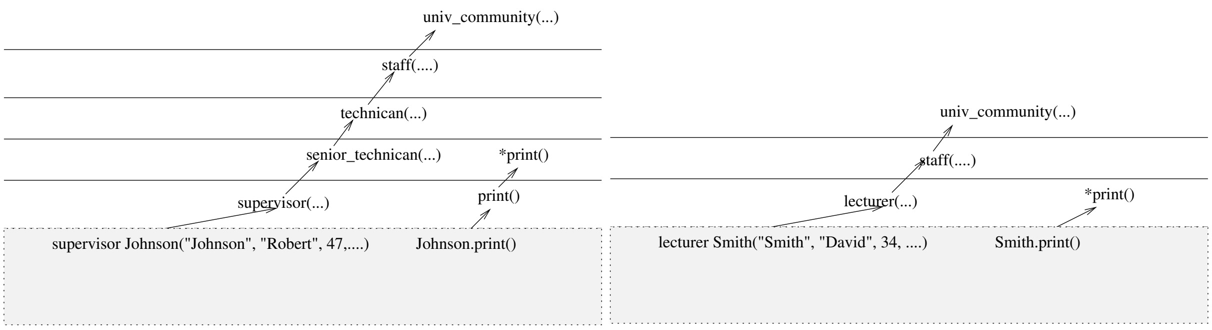


Figure 4.1: For University5 program with inheritance (left) and University3 program with inheritance (right) comprehension must be gained by searching through the inheritance hierarchy, starting in the main program. The arrows mean an invocation of a method. An arrow crossing a horizontal line means that the method invoked is located in a super- or subclass. The shaded region in the lower part are method calls from classes outside the main-hierarchy tree. Methods are marked with a star if they invoke other methods but (due to meaningful names) the method invocations do not need to be traced.

Analyzing the complexity metrics (number of classes in program, number of methods in program, LOC, number of methods to trace for the maintenance task, number of hierarchy changes to gain program understanding) we found two metrics having a high correlation with the required time. The average maintenance time correlates highly with the number of trace methods (r = 0.98) and the number of classes in the program (r = 0.97)[1] This is visualized in Figure 4.2.

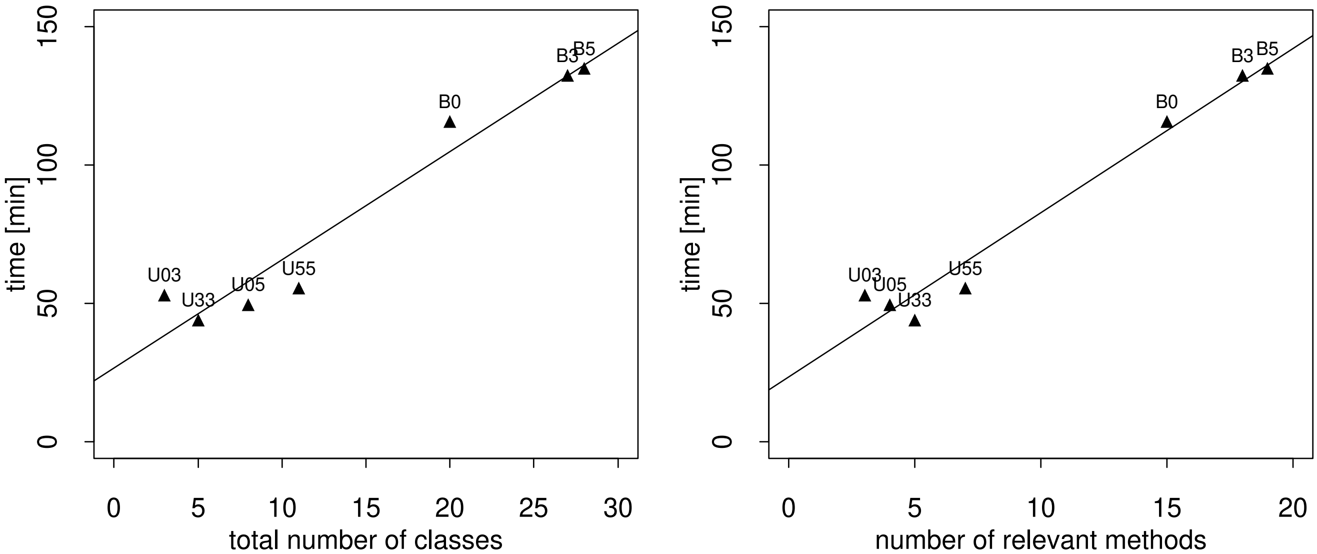


Figure 4.2: Correlation between the maintenance time and the number of relevant classes (left) and the number of relevant methods that have to be traced (right).

As we see, the university-3 program behaves against the trend of all the others: despite an increasing number of classes and trace methods the maintenance time decreases for the university-3 program. In the light of the

[1]This correlation comprise only the University03-programs, the University05-programs and our JAKK-programs. The correlations change to 0.96 (time vs. classes) and 0.96 (time vs. trace methods) if we add the data points of the Literature-programs of Daly et al. We leave these data points out because the Literature programs are quite similar to the University03-program.

other data points and the strength of the relation, this is unexpected and unexplainable for us. So these results suggest that the main data point in Dalys conclusion may be some sort of outlier. It would be best to analyze their data subject-by-subject, but unfortunately we do not have access to it; according to John Daly, the raw data was lost.

Our conclusion: Not the inheritance depth itself is a good predictor of maintenance effort; crucial attributes in our experiment are related ones such as the number of methods to be understood.

In this experiment a lot of (probably) relevant factors could not be investigated: (1) We considered predominantly comprehension-bound tasks. What is the effect of more implementation-bound tasks? (2) Much of the code was relevant for the task. But in reality only a small fraction of a program will typically be investigated during any single maintenance task. What is the effect of this? (3) Polymorphism was not really essential for the program structure, inheritance was employed for reuse only. Are our results valid for designs using polymorphism as well?

So future work should address these questions, try to determine the relevant factors related to inheritance, and quantify their impact on maintenance time.

Chapter 5

Conclusion

In this experiment we tested the impact of inheritance hierarchy on maintainability. The experiment was inspired by a similar experiment conducted by Daly et al.

For testing our hypotheses, (H1) the more abstraction levels are used, the easier the program is to comprehend and the less time is required for maintenance tasks and (H2) the more abstraction levels are used, the better is the quality of the delivered solutions, we divided the groups into three balanced groups working on equivalent programs with zero, three, or five levels of inheritance.

In both groups, JAKK and Informatik II, we recognized similar effects. The "flat" program version was faster to comprehend than the other two program versions with inheritance depth 3 and 5. In Task 1 (Y2K-Problem), the subjects with the flat program version had more modifications to apply than the group with the 5-level program, but they were almost as fast (2.4 percent slower, not significant). The 3-level group had the same number of changes as the 0-level group but was much slower (23.9 percent, significant).

In Task 2 and Task 2A (adding new classes), the trend differed because here overall program comprehension was required to be able to do the program changes in contrast to Task 1 where it was sufficient to comprehend only a small part of the program. The time required by the subjects increased with the inheritance depth. Subjects with the flat version were significantly faster than subjects of the 5-level group. Although they were faster, their solutions were on average more correct than in the other groups.

In Task 2B, mortality was high and non-uniform over the groups so we can not properly evaluate this task.

Hypothesis 1 and hypothesis 2 have to be rejected. Contrary effects have been observed in this controlled experiment. The flat version of the program was easier to comprehend and to maintain.

Regarding the subjective experience of the subjects collected in the final questionnaire, it is obvious that the subject's experience do not fit to the objective results. This indicates that experiments have to be conducted instead of relying on intuitive judgements. The software community should not rely on subjective experiences as was suggested by Al Davis in [5].

Our results contradict the results of the previous experiments by John Daly et al. It is most likely that the source of the differences is somewhere in the size and complexity of the programs and maintenance tasks; however, we can not point out exactly where.

We found that inheritance depth is not a good predictor for maintenance time. Instead we found two attributes that are good predictors – at least for the programs and "add a class" tasks in the given experiments: these are the total number of classes in the program and the number of methods relevant for understanding. Further work is required for finding other relevant attributes that allow to predict maintenance time and for creating a quantitative model of maintenance that is applicable to a broader set of contexts.

Appendix A

Tasks and Solutions

A.1 Handling descriptions

This appendix contains the original materials given to the subjects. They were handed out in four parts:

1. a questionnaire collecting personal information and a small test of the subjects' knowledge of Java, inheritance, and polymorphism
2. the explanation of the first task together with the program listing and a some Unix help information
3. the explanation of the second task
4. a postmortem questionnaire

The subjects had to give each part back to the experimenters (except for the listing and the Unix info) when they received the next part. Each subject could promptly do this at any time.

A.2 Original questionnaire (translated into English)

Instruction and Questionnaire for the JAKK experiment

Barbara Unger, Lutz Prechelt, Michael Philippsen
Fakultät für Informatik, Universität Karlsruhe
June 10, 1997

Please read and follow these instructions **carefully**. Fill in your student ID and time correctly when required. Invoke the command `delivering` for delivering the source code only when you are finished with one of the tasks.

Take as much time as you need. We planned sufficient working time. For us, it is more important that you work well rather than fast.

Student ID:

Questionnaire Part 1: Personal information

Please fill in this part of the questionnaire before reading further and before the actual experiment starts. *All information will be considered confidential.*

Completeness and correctness of your information (please print!) are important for the accuracy of the scientific results of the experiment. Therefore, please answer all questions.

____________________ lastname ____________________ firstname

________ sex: m/f

________ time

I am a ____________ (subject) major in my ______-th semester,

☐ with Vordiplom, ☐ without Vordiplom.

Before the Java course started I had the following programming experience (overall):

- ☐ only theoretical knowledge.
- ☐ wrote less than 300 lines of code myself.
- ☐ wrote less than 3,000 lines of code myself.
- ☐ wrote less than 30,000 lines of code myself.
- ☐ wrote more than 30,000 lines of code myself.

I have been programming for about ______ years now and used predominantly the following programming languages (ordered by decreasing experience):

____________________ language 1, language 2, . . .

Before the Java course started I had the following experience in object oriented programming:

- [ ] no knowledge
- [ ] only theoretical knowledge.
- [ ] wrote less than 300 lines of code myself.
- [ ] wrote less than 3,000 lines of code myself.
- [ ] wrote less than 30,000 lines of code myself.
- [ ] wrote more than 30,000 lines of code myself.

Before the Java course started I had the following experience in programming graphical user interfaces (GUIs):

- [ ] no knowledge
- [ ] only theoretical knowledge.
- [ ] wrote less than 300 lines of code myself.
- [ ] wrote less than 3,000 lines of code myself.
- [ ] wrote less than 30,000 lines of code myself.
- [ ] wrote more than 30,000 lines of code myself.

The longest program I have written *alone* had about ____________ (LOC) lines of code. It was written in ____________ (Programming language) and consumed an effort of ____________ (person month) person month.

Answer the following question only if you have already worked in a team software project or are doing this currently. The largest program in whose construction I have *participated* had about ____________ (LOC) lines of code altogether and consumed an effort of ____________ (person months) person months. My own contribution was about ____________ (LOC) lines of code or ____________ (person months) person months, respectively.

My understanding of object-oriented concepts in Java is as follows:
(Enter a number between 1 and 5 for each concept. The number indicates how well you subjectively believe to understand the concept.
1: I understand and apply it very well,
2: I understand it well,
3: I understand it roughly,
4: I am beginning to understand it,
5: I do not understand it)

______ Class

______ Object (instance)

______ Polymorphism

______ Inheritance (*extends*)

______ Overloading and overwriting methods

______ abstract class

______ abstract method

______ Interface (*implements*)

______ Difference value type (e. g. *int*) and object type

Now please enter your student id and time.

Student ID: Time:

Questionnaire Part 2: A small Java test

Consider the following Java program:

```
class Superclass {
  int i = 7;
  void m() {
     System.out.println("Superclass i = "+i);
  }
}
class DerivedClass extends Superclass {
  static int j = 9;
  void m() {
     System.out.println("DerivedClass j = "+j);
  }
  static void mi() {
     System.out.println("DerivedClass i = "+i);
  }
}

class Test {
  public static void main(String args[]) {
     Superclass  o1 = new Superclass();
     Superclass  o2;
     DerivedClass u1 = new DerivedClass();
     DerivedClass u2;
     o2.m();
     o2 = o1;
     Superclass.m();
     DerivedClass.mi();
     o1.i++;
     o2.m();
     u2 = u1;
     u1.j++;
     u2.m();
     u1.j++;
     o2 = u1;
     o1.i = 0;
     o2.i = 1;
     o2.m();
     o2.mi();
  }
}
```

a.) Some of the instructions in the above program are illegal, that is, they result in compile time errors or run time errors. Mark all of these instructions in the listing above. Please do not use the computer for solving this task.

b.) Some of the instructions in the program produce output lines. Note in the listing behind these lines which output will be produced. Assume that the lines marked as wrong in a.) are removed from the program. Please do *not* use the computer for solving this task.

Tip: The simplest method is to write down step by step all changes of the objects and variables.

When you have finished this task **invoke the command** `deliver`, enter the actual time here, and request new materials. Time:

Now please enter the time. Time:
Student id: 000000 **group: BoerseX**

The exercise: Program “Boerse”

You received a listing of the program “Boerse” and a corresponding class diagram indicating the inheritance structure.

The source code is located on your computer:

Your Login: `Test`

Your Password: `pwd`

Directory: `~/Boerse`

File: `Boerse.java` (and in case of need a copy is stored in `Boerse.java.orig`)

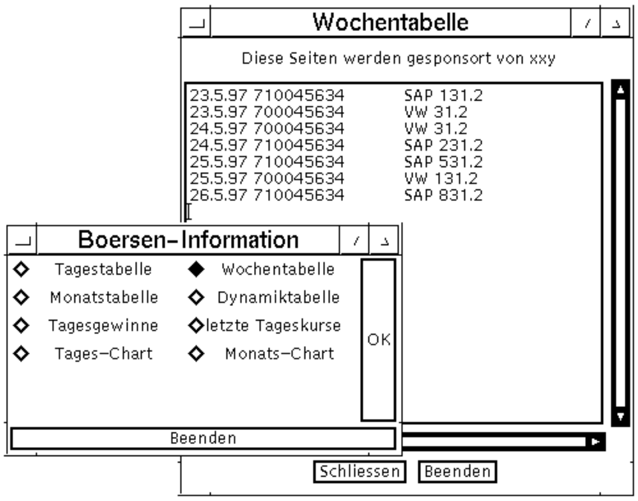
The purpose of the program is to show a selection of display types of various stock shares. For simplification the data are read from a file and the displays are simple, too.

There are two files that provide the data. One file called AVG_PRICES contains one entry per day and stock. Each entry consists of a date, the stock ID number (that identifies the stock share unambiguously), the appropriate (maybe ambiguous) name of the company and the average price of the share for that day. The second file called PRICES contains possibly several price values per day for the same stock, described by date, time, stock ID number, company name and share value (price).

The user can chose between various displays of the prices in tables or graphs with different selections and contents. For example, the DayTable displays all share values for all shares of one day, while a DayChart displays these data graphically for just one share.

A WeekTable displays all share values for all shares of the last seven days, a MonthTable accordingly for the last 30 days, etc. A MonthDynamicsTable shows for any two successive days of one month the change of the average share value in percent, etc.

As an example the following screen shot displays a view of the program while creating a week table:



Read and comprehend the program `Boerse.java` so that you can process the following two work tasks. The first task is described below, the second task will be handed out on a different sheet after completing the first one.

If you think that comprehending some parts of the program is not necessary, you do not need to investigate them.

Please do not write the solutions into the program listing. Implement your solution directly in the source code of `Boerse.java`, compile it and test it.

However, if you want to, you can write marks and notes into the listing. After the second work task deliver the listing together with the other handouts.

Work task 1 for program "'Boerse'"

Starting with the year 2000, it will be insufficient to code the year number with two digits.

Enhance `Boerse.java` in that manner that the program can process the new files `PRICES_2000` and `AVG_PRICES_2000` correctly. These files contain four digit year numbers instead of two digit year numbers (the rest of the format and content is identical). Test your program with these files.

When you have finished this task or if you think, you can't do better to complete this task, **invoke the command** `deliver`, enter the actual time here, and request new materials. Time:

Now please enter the time. Time:
Student id: 000000 **group: BoerseX**

Work task 2 for program "'Boerse"'

The tables implemented yet all cover fixed time intervals, for example one week back or one month back starting from today. This is too restrictive for the users; they want to choose the time interval themselves.

Enhance the program so that a new menu item "AnyIntervalTable" is offered. "AnyIntervalTable" should display a new table covering a user-specified time interval. The user is asked (via a dialog box) for a time interval (start date and end date, format `dd.mm.yyyy-dd.mm.yyyy`, for example `17.02.1997-26.03.1997`). .

Analogous to (for example) a WeekTable, all average prices (date, stock ID number, company name and share price) should be displayed for the specified time interval.

For solving the work task you can introduce new classes or you can change existing classes – as you prefer. For the dialog box use the same `Frame` as for the table created subsequently (only change its contents), just as it is done for the chart displays.

In much the same manner, introduce another new menu item "'AnyDynamicIntervalTable"', that displays a dynamics table (analogous to month dynamics table) of a time interval specified in a dialog box.

When you have finished this task or if you think, you can't do better to complete this task, **invoke the command** `deliver`, enter the actual time here, and request new materials. Time:

Now please enter your student id and time.

Student ID: Time:

Questionnaire Part 3: Experience in the program

For proper evaluation of the experiment, we need your cooperation.

This part of the questionnaire is meant to complement the answers you gave above by subjective background information in order to allow for a better analysis of the experiment results.

Most questions have some additional space below for arbitrary comments; such information would be very useful for us.

I think that in the program "Boerse.java" inheritance is

☐ used much too much
☐ is used too much
☐ is used in the right degree
☐ is used too little
☐ is used much too little

comment:

Measured on programs of this size, I think that "Boerse.java" has a

☐ very clear structure
☐ clear structure
☐ medium clear structure
☐ not so clear structure
☐ unclear structure

comment:

The questions below have two checkboxes for each answer: [1] for Task 1 (Year 2000) and [2] for Task 2 (Time interval).
Please check exactly one box per column for each question.

Overall and in the given situation I found task 1 and 2

[A] [B] pretty simple
[A] [B] not quite so simple
[A] [B] pretty difficult
[A] [B] difficult

comment:

During task 1 and 2 my ability to concentrate on the work was

- A B very high
- A B high
- A B OK
- A B not so high
- A B low

comment:

I believe that my solutions for task 1 and 2 have

- A B no errors or omissions
- A B at most one error or omission
- A B probably several errors or omissions
- A B (don't know)

comment:

I think that for solving task 1 and 2 my previous knowledge of object oriented programming (as opposed to general programming knowledge) was helpful:

- A B no, not at all
- A B only a little
- A B can't decide
- A B yes, somewhat
- A B yes, very much

comment:

This is how much effort I invested in various aspects while solving task 1 and task 2 (Please fill in each line a number between 1 and 5:
1: very much, 2: much, 3: medium, 4: somewhat, 5: only a little)

- A B Implementing a well understandable program
- A B Implementing a program that is well extensible
- A B Implementing a well testable program
- A B Implementing a defect-free program
- A B Implementing an efficient program
- A B Solving the task with little effort

comment:

I think the purpose of this experiment was:
(Note: If you don't have an idea, just skip this question.)

Something else I would like to say (e. g. what I found particularly difficult, unclever in the experiment setup, interesting, etc.):

Thank you!

Many thanks for participating in our experiment.

A.3 Solutions

The solution described below is for the JAKK program version with inheritance depth 5. The solutions for the other program versions are analogous.

A.3.1 Solution for Task 1 – ”Y2K”

line	original	new
49	static Datum heute = new Datum(27,5,97);	static Datum heute = new Datum(27,5,1997);
50	static String kurse = ”KURSE”;	static String kurse = ”KURSE_2000”;
51	static String kassakurse = ”KASSAKURSE”;	static String kassakurse = ”KASSAKURSE_2000”;
380	d.setzeDatum(s.substring(0,8));	d.setzeDatum(s.substring(0,10));
382	return (wkn.compareTo(s.substring(15,24))==0);	return (wkn.compareTo(s.substring(17,26))==0);
416	uhrzeit = new Uhrzeit(s.substring(9,15));	uhrzeit = new Uhrzeit(s.substring(11,17));
468	d.setzeDatum(s.substring(0,8));	d.setzeDatum(s.substring(0,10));
470	return (wkn.compareTo(s.substring(9,18))==0);	return (wkn.compareTo(s.substring(11,20))==0);
499	datum = new Datum(s.substring(0,9));	datum = new Datum(s.substring(0,10));
558	d.setzeDatum(s.substring(0,8));	d.setzeDatum(s.substring(0,10));
565	Datum datum = new Datum(s.substring(0,8));	Datum datum = new Datum(s.substring(0,10));
566	Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));	Uhrzeit uhrzeit = new Uhrzeit(s.substring(11,16));
568	String wkn = s.substring(14,24);	String wkn = s.substring(16,26);
569	String name = s.substring(25,s.length()-7);	String name = s.substring(27,s.length()-7);
665	jahr = new Integer(s.substring(6,8)).intValue();	jahr = new Integer(s.substring(6,10)).intValue();
677	jahr = new Integer(s.substring(6,8)).intValue();	jahr = new Integer(s.substring(6,10)).intValue();
796	d.setzeDatum(s.substring(0,8));	d.setzeDatum(s.substring(0,10));
802	Datum datum = new Datum(s.substring(0,8));	Datum datum = new Datum(s.substring(0,10));
803	String wkn = s.substring(9,18);	String wkn = s.substring(11,20);
804	String name = s.substring(19,s.length()-7);	String name = s.substring(21,s.length()-7);

A.3.2 Solution for Task 2 - ”Time Interval Price Display and Gain/Loss Display”

```
class BelTabelle extends ZeitraumTabelle {
  Datum bisDatum;
  Label l = new Label();
  TextField t = new TextField("29.11.1996-29.11.1997",21);
  Button ok = new Button("ok");

  BelTabelle(String titel) {
    super(titel);
  }

  boolean selektiere(String s) {
    Datum d = new Datum();
    d.setzeDatum(s.substring(0,10));
    return (vonDatum.kleinergleich(d) && d.kleinergleich(bisDatum));
  }

  void stelleDar(){                                  //ueberdeckt stelleDar aus Tabelle.
    p2.add("North",l);
    p2.add("Center",t);
    p2.add("South",ok);
    pack();
    show();
```

```
    }


    boolean myaction(Event ev, Object target){
        if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
            String zeitraum = t.getText();
            vonDatum = new Datum(zeitraum.substring(0,10));
            bisDatum = new Datum(zeitraum.substring(11,21));
            p2.removeAll();
            leseDaten(dateiname);
            zeigeTabelleAn();
            return true;
        }
        return false;
    }
}


class BelDynamikTabelle extends MonatsDynamikTabelle {
    Datum bisDatum;
    Label l = new Label();
    TextField t = new TextField("29.11.1996-29.11.1997",21);
    Button ok = new Button("ok");

    BelDynamikTabelle(String titel) {
        super(titel);
    }

    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,10));
        return (vonDatum.kleinergleich(d) && d.kleinergleich(bisDatum));
    }

    void stelleDar(){                              //ueberdeckt stelleDar aus Tabelle.
        p2.add("North",l);
        p2.add("Center",t);
        p2.add("South",ok);
        pack();

        show();
    }


    boolean myaction(Event ev, Object target){
        if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
            String zeitraum = t.getText();
            vonDatum = new Datum(zeitraum.substring(0,10));
            bisDatum = new Datum(zeitraum.substring(11,21));
            p2.removeAll();
            leseDaten(dateiname);
            zeigeTabelleAn();
            return true;
        }
        return false;
    }
}
```

/////////Add in class Auswahl:

```
Checkbox cbbt = new Checkbox("Bel.   Tabelle", cbg, false);
Checkbox cbbdt = new Checkbox("Bel.Dynamiktabelle", cbg, false);

  p.add(cbbt);
  p.add(cbbdt);

     } else if(cbg.getSelectedCheckbox() == cbbt) {
       BelTabelle bt = new BelTabelle("Bel.   Tabelle");
       bt.stelleDar();
       return true;
     } else if(cbg.getSelectedCheckbox() == cbbdt) {
       BelDynamikTabelle bdt = new BelDynamikTabelle("Bel.Dynamiktabelle");
       bdt.stelleDar();
       return true;
```

Appendix B

Programs

B.1 Programs – JAKK

B.1.1 Boerse.java (no inheritance)

```
/************************************************************************/
/*Programmautor:  Barbara  Unger                                        */
/*Datum:           1997-06-03                                           */
/*Gruppe:          Boerse1                                              */
/************************************************************************/

import java.awt.*;
import java.util.Vector;
import java.util.Enumeration;
import java.util.Date;
import java.io.*;

/* Programm zum Betrachten von Boersenkursen.
   Abfolge der Klassen in dieser Datei:

   interface Konstanten
   class TagesChart extends Frame implements Konstanten
   class TagesChartWert
   class MonatsChart extends Frame implements Konstanten
   class MonatsChartWert
   class TagesTabelle extends Frame implements Konstanten
   class Tagesdaten
   class Zeitraumdaten
   class Datum
   class Uhrzeit
   class Kurs
   class Tagesgewinne extends Frame implements Konstanten
   class LetzteTageskurse extends Frame implements Konstanten
   class WochenTabelle extends Frame implements Konstanten
   class MonatsTabelle extends Frame implements Konstanten
   class MonatsDynamikTabelle extends Frame implements Konstanten
   class TagesChartAnzeiger extends Canvas
   class MonatsChartAnzeiger extends Canvas
   class Auswahl extends Frame
```

```
    class Boerse
*/


interface Konstanten {
//   static Date date = new Date();
//   static Datum heute = new Datum(date.getDay(), date.getMonth(), date.getYear());
   static Datum heute = new Datum(27, 5, 97);
   static String kurse = "KURSE";
   static String kassakurse = "KASSAKURSE";
}


/* Klasse TagesChart

    Implementiert ein Fenster, in dem eine Boersen-Information angezeigt wird.
    Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
    des Programmes, berechnet fuer das Abrufen von Informationen keine
    Kosten (Kosten werden von einem Sponsor getragen).
    Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
    Daten jedoch daraus erzeugt werden, wird durch die Methode
    erzeugeDatenvector() entschieden. Diese entscheidet durch eine
    selektiere()-Methode, welche Datensaetze ueberhaupt ausgewaehlt werden sollen.
    Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
    Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
    werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

    Methoden:
    TagesChart(String)   Konstruktor, der den Titel setzt, einen Sponsor
                              einfuegt, die Kosten auf 0.00 setzt und die Datei
                              mit den Daten angibt.
    leseDaten(String)    liest Daten aus einer Datei und schreibt sie mit Hilfe
                              der Methode erzeugeDatenvector() in den Vector vec
    erzeugeDatenvector(String) bekommt eine Datensatz in Stringform und
                              entscheidet mit Hilfe von selektiere(), ob der
                              Datensatz in den Datenvektor eingefuegt werden soll.
    selektiere(String s) entscheidet, ob der String ausgewaehlt werden
                              soll, konkret: ob er von dem angegebenen Datum ist
                              und die entsprechende Wertpapierkennnummer hat
    qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                              sind Methoden, die einen Sortieralgorithmus zur
                              Verfuegung stellen.
    schliesseFenster()   kehrt zum aufrufenden Fenster zurueck
    beendeProgramm()     beendet das Programm komplett
    action(Event, Object) verarbeitet Interaktionsereignisse
    myaction()           erweitert die action-Methode.
    void stelleDar()     stellt eine Dialogbox dar.
    zeigeCanvasAn()      erzeugt ein Objekt der Canvas-Klasse, in der der
                              Chart aus den selektierten Daten, die im Datenvektor
                              stehen, angezeigt wird.
*/


class TagesChart extends Frame implements Konstanten {
   Panel p1 = new Panel();
   Panel p2 = new Panel();
   Panel p3 = new Panel();
   Button schliessen = new Button("Schliessen");
   Button beenden = new Button("Beenden");
   float kosten;
   String dateiname;
   Label sponsor;
   Vector vec = new Vector();
   int xChartSize = 200;
```

```
    int yChartSize = 200;
    String wkn;
    Label l = new Label();
    TextField t = new TextField("700045634",9);
    Button ok = new Button("ok");
    TagesChartWert tcw;

    /*  Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
          bestimmt das Layout.
          Panel p1 enthaelt die Sponsorinformation
          Panel p2 enthaelt eine Auswahlmoeglichkeit des gewuenschten Wertpapiers
          Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
                    des Programms.
          Die Kosten werden auf 0.00 gesetzt.
    */

    TagesChart(String titel) {
      super(titel);
      this.setLayout(new BorderLayout());
      p1.setLayout(new FlowLayout());
      add("North",p1);
      p2.setLayout(new BorderLayout());
      add("Center",p2);
      p3.setLayout(new FlowLayout());
      add("South",p3);
      p3.add(schliessen);
      p3.add(beenden);
      sponsor = new Label("Diese Seiten werden gesponsort von xxy");
      p1.add(sponsor);
      kosten = 0.0f;
      dateiname = kurse;
      l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Tageschart an");
    }

    /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
         Methode erzeugeDatenvector eine Datenstruktur, in der die
         relevanten Daten gespeichert werden.
     */
    void leseDaten(String dateiname) {
      try{
        BufferedReader br = new BufferedReader(new FileReader(dateiname));
        String s;
        try{
          while((s = br.readLine())≠null) {
            if(!s.startsWith("#")) {
              erzeugeDatenvector(s);
            }
          }
        } catch (IOException e) {
          System.err.println("Lesefehler bei Datei " + dateiname);
        }
      } catch (FileNotFoundException e) {
        System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
      }
    }

  /* bekommt eine Datensatz in Stringform und entscheidet mit Hilfe von selektiere(),
      ob der Datensatz als TagesChartWert-Objekt in den Datenvektor vec
      eingefuegt werden soll.
  */
    void erzeugeDatenvector(String s){
      if(selektiere(s) == true) {
        tcw = new TagesChartWert(s);
        vec.addElement(tcw);
```

```
    }
  }

/* entscheidet, ob der String ausgewaehlt werden soll, konkret: ob er von
    dem angegebenen Datum ist und die entsprechende Wertpapierkennnummer hat
*/
  boolean selektiere(String s) {
    Datum d = new Datum();
    d.setzeDatum(s.substring(0,8));
    if((heute.tag == d.tag) && (heute.monat ==d.monat) && (heute.jahr == d.jahr)) {
      return (wkn.compareTo(s.substring(15,24))==0);
    }
    return false;
  }

  /* Beschreibung des Quicksort siehe oben
  */
  void qsort(int left, int right) {
    int middle;
    if(left < right){
      middle = partition(left, right);
      qsort(left,middle-1);
      qsort(middle+1, right);
    }
  }

  /* Hilfsprozedur fuer den Quicksort
  */
  int partition(int left, int right) {
    Tagesdaten pivot = (Tagesdaten)vec.elementAt(left);
    int l = left;
    int r = right;
    int middle;
    while(l < r) {
      while((l ≤ right) && (((Tagesdaten)vec.elementAt(l)).vergleiche(pivot))){
        l = l+1;
      }
      while((r ≥ left) && (!((Tagesdaten)vec.elementAt(r)).vergleiche(pivot))){
        r = r-1;
      }
      if(l < r) {
        exchange(l, r);
      }
    }
    middle = r;
    exchange (left, middle);
    return middle;
  }

  /* Hilfsprozedur fuer den Quicksort
  */
  void exchange(int i, int j) {
    Tagesdaten t = (Tagesdaten)vec.elementAt(i);
    vec.setElementAt(vec.elementAt(j),i);
    vec.setElementAt(t,j);
  }

  /* Ruft Quicksort auf
  */
  void sortiere() {
    qsort(0, vec.size()-1);
  }

  /* Schliesst das Fenster und gibt seine Ressourcen frei.
```

```
     */
    void schliesseFenster() {
        this.dispose();
    }

    /* Beendet das Programm komplett: alle Fenster werden geschlossen.
    */
    void beendeProgramm() {
        System.exit(0);
    }

    /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
         Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
         anderen Ereignisse wird "myaction" aufgerufen.
    */
    public boolean action(Event event, Object target) {
        if(event.id == Event.ACTION_EVENT) {
            if(event.target == schliessen) {
                schliesseFenster();
                return true;
            }
            else if(event.target == beenden) {
                beendeProgramm();
                return true;
            } else {
                return myaction(event, target);
            }
        } else {
            return false;
        }
    }

    /* myaction erweitert die action-Methode, so dass nun beim Druck des
         Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
         aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
         und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
     */
    boolean myaction(Event ev, Object target){
        if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
            wkn = t.getText();
            wkn.trim();
            p2.removeAll();
            leseDaten(dateiname);
            p2.add("Center",zeigeCanvasAn());
            pack();
            show();
            return true;
        }
        return false;
    }

    void stelleDar(){
        p2.add("North",l);
        p2.add("Center",t);
        p2.add("South",ok);
        pack();
        show();
    }

  /* erzeugt ein Objekt der Canvas-Klasse, in der der Chart aus den
       selektierten Daten, die im Datenvektor stehen, angezeigt wird.
  */
    Canvas zeigeCanvasAn() {
        return new TagesChartAnzeiger(vec, xChartSize, yChartSize);
```

```
  }
}

/*Klasse TagesChartWert
  ist ein Objekttyp, der aus einer Uhrzeit und einem Kurswert besteht.
  Das sind genau die Daten, die man zum Anzeigen eines Tagescharts braucht.
  Die Klasse hat eine vergleiche-Methode, so dass man einen Vector mit
  Objekten dieses Typs mit der sortier()-Methode sortieren kann.

  Methoden:
    TagesChartWert(String) bekommt einen String, in dem die Uhrzeit
                           und der Kurs enthalten sind.
    vergleiche(TagesChartWert t) vergleicht die Uhrzeiten
                           a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
*/


class TagesChartWert {
  Uhrzeit uhrzeit;
  Kurs kurs;

/* bekommt einen String, in dem die Uhrzeit und der Kurs enthalten sind.
 */
  TagesChartWert(String s) {
    uhrzeit = new Uhrzeit(s.substring(9,15));
    kurs = new Kurs(s.substring(s.length()-7, s.length()));
  }

/* vergleicht die Uhrzeiten zweier TagesChartWert-Objekte
 */
  public boolean vergleiche(TagesChartWert t) {
    return ((this.uhrzeit).kleinergleich(t.uhrzeit));
  }
}

/* Klasse MonatsChart

   Implementiert ein Fenster, in dem eine Boersen-Information angezeigt wird.
   Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
   des Programmes, berechnet fuer das Abrufen von Informationen keine
   Kosten (Kosten werden von einem Sponsor getragen).
   Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
   Daten jedoch daraus erzeugt werden, wird durch die Methode
   erzeugeDatenvector() entschieden. Diese entscheidet durch eine
   selektiere()-Methode, welche Datensaetze ueberhaupt ausgewaehlt werden sollen.
   Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
   Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
   werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

   Methoden:
   MonatsChart(String) Konstruktor, der den Titel setzt, einen Sponsor
                       einfuegt, die Kosten auf 0.00 setzt und die Datei
                       mit den Daten angibt.
   leseDaten(String)   liest Daten aus einer Datei und schreibt sie mit Hilfe
                       der Methode erzeugeDatenvector() in den Vector vec
   erzeugeDatenvector(String) bekommt eine Datensatz in Stringform und
                       entscheidet mit Hilfe von selektiere(), ob der
                       Datensatz in den Datenvektor eingefuegt werden soll.
   selektiere(String s) entscheidet, ob der String ausgewaehlt werden
                       soll, konkret: ob er von dem angegebenen Datum ist
                       und die entsprechende Wertpapierkennnummer hat
   qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                       sind Methoden, die einen Sortieralgorithmus zur
                       Verfuegung stellen.
```

```
    schliesseFenster()   kehrt zum aufrufenden Fenster zurueck
    beendeProgramm()     beendet das Programm komplett
    action(Event, Object) verarbeitet Interaktionsereignisse
    myaction()           erweitert die action-Methode.
    void stelleDar()     stellt eine Dialogbox dar.
    zeigeCanvasAn()      erzeugt ein Objekt der Canvas-Klasse, in der der
                           Chart aus den selektierten Daten, die im Datenvektor
                           stehen, angezeigt wird.
*/


class MonatsChart extends Frame implements Konstanten {
  Panel p1 = new Panel();
  Panel p2 = new Panel();
  Panel p3 = new Panel();
  Button schliessen = new Button("Schliessen");
  Button beenden = new Button("Beenden");
  float kosten;
  String dateiname;
  Label sponsor;
  Vector vec = new Vector();
  int xChartSize = 200;
  int yChartSize = 200;
  String wkn;
  Label l = new Label();
  TextField t = new TextField("700045634",9);
  Button ok = new Button("ok");
  MonatsChartWert mcw;
  Datum letzterMonat = new Datum();

  /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
      bestimmt das Layout.
      Panel p1 enthaelt die Sponsorinformation
      Panel p2 enthaelt eine Auswahlmoeglichkeit des gewuenschten Wertpapiers
      Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
               des Programms.
      Die Kosten werden auf 0.00 gesetzt.
  */

  MonatsChart(String titel) {
    super(titel);
    this.setLayout(new BorderLayout());
    p1.setLayout(new FlowLayout());
    add("North",p1);
    p2.setLayout(new BorderLayout());
    add("Center",p2);
    p3.setLayout(new FlowLayout());
    add("South",p3);
    p3.add(schliessen);
    p3.add(beenden);
    sponsor = new Label("Diese Seiten werden gesponsort von xxy");
    p1.add(sponsor);
    kosten = 0.0f;
    l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Monatschart an");
    dateiname = kassakurse;
    letzterMonat = heute.monatzurueck();
  }

  /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
     Methode erzeugeDatenvector eine Datenstruktur, in der die
     relevanten Daten gespeichert werden.
   */
  void leseDaten(String dateiname) {
    try{
```

```
            BufferedReader br = new BufferedReader(new FileReader(dateiname));
            String s;
            try{
                while((s = br.readLine())≠null) {
                    if(!s.startsWith("#")) {
                        erzeugeDatenvector(s);
                    }
                }
            } catch (IOException e) {
                System.err.println("Lesefehler bei Datei " + dateiname);
            }
        } catch (FileNotFoundException e) {
            System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
        }
    }

    /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
       gespeichert sind. Dieses Objekt wird im Vektor vec gepeichert
     */
    void erzeugeDatenvector(String s){
        if(selektiere(s) == true) {
            mcw = new MonatsChartWert(s);
            vec.addElement(mcw);
        }
    }

    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,8));
        if(!d.kleinergleich(letzterMonat)) {
            return (wkn.compareTo(s.substring(9,18))==0);
        }
        return false;
    }

    /* Beschreibung des Quicksort siehe oben
     */
    void qsort(int left, int right) {
        int middle;
        if(left < right){
            middle = partition(left, right);
            qsort(left,middle-1);
            qsort(middle+1, right);
        }
    }

    /* Hilfsprozedur fuer den Quicksort
     */
    int partition(int left, int right) {
        MonatsChartWert pivot = (MonatsChartWert)vec.elementAt(left);
        int l = left;
        int r = right;
        int middle;
        while(l < r) {
            while((l ≤ right) && (((MonatsChartWert)vec.elementAt(l)).vergleiche(pivot))){
                l = l+1;
            }
            while((r ≥ left) && (!((MonatsChartWert)vec.elementAt(r)).vergleiche(pivot))){
                r = r-1;
            }
            if(l < r) {
                exchange(l, r);
            }
        }
```

```
    middle = r;
    exchange (left, middle);
    return middle;
  }

  /* Hilfsprozedur fuer den Quicksort
   */
  void exchange(int i, int j) {
    MonatsChartWert m = (MonatsChartWert)vec.elementAt(i);
    vec.setElementAt(vec.elementAt(j),i);
    vec.setElementAt(m,j);
  }

  /* Ruft Quicksort auf
   */
  void sortiere() {
    qsort(0, vec.size()-1);
  }

  /* Schliesst das Fenster und gibt seine Ressourcen frei.
   */
  void schliesseFenster() {
    this.dispose();
  }

  /* Beendet das Programm komplett: alle Fenster werden geschlossen.
   */
  void beendeProgramm() {
    System.exit(0);
  }

  /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
       Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
       anderen Ereignisse wird "myaction" aufgerufen.
  */
  public boolean action(Event event, Object target) {
    if(event.id == Event.ACTION_EVENT) {
      if(event.target == schliessen) {
        schliesseFenster();
        return true;
      }
      else if(event.target == beenden) {
        beendeProgramm();
        return true;
      } else {
        return myaction(event, target);
      }
    } else {
      return false;
    }
  }

  /* myaction erweitert die action-Methode, so dass nun beim Druck des
       Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
       aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
       und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
   */
  boolean myaction(Event ev, Object target){
    if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
      wkn = t.getText();
      wkn.trim();
      p2.removeAll();
      leseDaten(dateiname);
      p2.add("Center",zeigeCanvasAn());
```

```
            pack();
            show();
            return true;
        }
        return false;
    }

    void stelleDar(){
        p2.add("North",l);
        p2.add("Center",t);
        p2.add("South",ok);
        pack();
        show();
    }

    Canvas zeigeCanvasAn() {
        return new MonatsChartAnzeiger(vec, xChartSize, yChartSize);
    }
}

/*Klasse MonatsChartWert
    ist ein Objekttyp, der aus einem Datum und einem Kassakurswert besteht,
    Daten, die man zum Anzeigen eines Monatschart braucht. Die Klasse hat
    eine vergleiche-Methode, so dass man einen Vector mit diesem Typ mit
    der sortier()-Methode sortieren kann.

    Methoden:
        MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
                                    und der Kurs enthalten sind.
        vergleiche(MonatsChartWert m) vergleicht das Datum
                                    a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
*/

class MonatsChartWert {
    Kurs kassakurs;
    Datum datum;

    MonatsChartWert(String s){
        kassakurs = new Kurs(s.substring(s.length()-6, s.length()));
        datum = new Datum(s.substring(0,9));
    }

    public boolean vergleiche(MonatsChartWert m){
        return ((this.datum).kleinergleich(m.datum));
    }
}

/* Klasse TagesTabelle

     Implementiert ein Fenster, in dem eine tabellarische
     Darstellung aller Kurse eines Tages angezeigt wird.
     Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
     des Programmes, berechnet fuer das Abrufen von Informationen keine
     Kosten (Kosten werden von einem Sponsor getragen).
     Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
     Daten jedoch daraus erzeugt werden, wird durch die Methode
     erzeugeDatenvector() entschieden. Diese entscheidet durch eine
     selektiere()-Methode, welche Datensaetze ueberhaupt ausgewaehlt werden sollen.
     Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
     Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
     werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

     Methoden:
     TagesTabelle(String) Konstruktor, der den Titel setzt, einen Sponsor
```

```
                         einfuegt, die Kosten auf 0.00 setzt und die Datei
                         mit den Daten angibt.
   leseDaten(String)   liest Daten aus einer Datei und schreibt sie mit Hilfe
                         der Methode erzeugeDatenvector() in den Vector vec
   selektiere(String s) entscheidet, ob der String ausgewaehlt werden
                         soll, konkret: ob er von dem angegebenen Datum ist
                         und die entsprechende Wertpapierkennnummer hat
   qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                         sind Methoden, die einen Sortieralgorithmus zur
                         Verfuegung stellen.
   schliesseFenster()  kehrt zum aufrufenden Fenster zurueck
   beendeProgramm()    beendet das Programm komplett
   action(Event, Object) verarbeitet Interaktionsereignisse
   myaction(Event, Object) wird in "action" aufgerufen
*/

class TagesTabelle extends Frame implements Konstanten {
  Panel p1 = new Panel();
  Panel p2 = new Panel();
  Panel p3 = new Panel();
  Button schliessen = new Button("Schliessen");
  Button beenden = new Button("Beenden");
  float kosten;
  String dateiname;
  Label sponsor;
  Vector vec = new Vector();
  TextArea textarea;

  /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
       bestimmt das Layout.
       Panel p1 enthaelt die Sponsorinformation
       Panel p2 dient zum Anzeigen der angeforderten Informationen,
       Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
                des Programms.
       Die Kosten werden auf 0.00 gesetzt.
  */

  TagesTabelle(String titel) {
    super(titel);
    this.setLayout(new BorderLayout());
    p1.setLayout(new FlowLayout());
    add("North",p1);
    p2.setLayout(new BorderLayout());
    add("Center",p2);
    p3.setLayout(new FlowLayout());
    add("South",p3);
    p3.add(schliessen);
    p3.add(beenden);
    sponsor = new Label("Diese Seiten werden gesponsort von xxy");
    p1.add(sponsor);
    kosten = 0.0f;
    setzeDateiname();
  }

  void setzeDateiname(String s) {
    dateiname = s;
  }

  void setzeDateiname() {
    dateiname = kurse;
  }

  /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
      Methode erzeugeDatenvector eine Datenstruktur, in der die
```

```
     relevanten Daten gespeichert werden.
   */
  void leseDaten(String dateiname) {
    try{
      BufferedReader br = new BufferedReader(new FileReader(dateiname));
      String s;
      try{
        while((s = br.readLine())≠null) {
          if(!s.startsWith("#")) {
            erzeugeDatenvector(s);
          }
        }
      } catch (IOException e) {
        System.err.println("Lesefehler bei Datei " + dateiname);
      }
    } catch (FileNotFoundException e) {
      System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
    }
  }

  /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
     gespeichert sind. Dieses Objekt wird im Vektor vec gepeichert
   */
  void erzeugeDatenvector(String s) {
    if(selektiere(s) == true) {
      Datum datum = new Datum(s.substring(0,8));
      Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
      Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
      String wkn = s.substring(14,24);
      String name = s.substring(25, s.length()-7);
      Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
      vec.addElement(tagesdaten);
    }
  }

  boolean selektiere(String s) {
    Datum d = new Datum();
    d.setzeDatum(s.substring(0,8));
    return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
  }

  /* Beschreibung des Quicksort siehe oben
   */
  void qsort(int left, int right) {
    int middle;
    if(left < right){
      middle = partition(left, right);
      qsort(left,middle-1);
      qsort(middle+1, right);
    }
  }

  /* Hilfsprozedur fuer den Quicksort
   */
  int partition(int left, int right) {
    Tagesdaten pivot = (Tagesdaten)vec.elementAt(left);
    int l = left;
    int r = right;
    int middle;
    while(l < r) {
      while((l ≤ right) && (((Tagesdaten)vec.elementAt(l)).vergleiche(pivot))){
        l = l+1;
      }
      while((r ≥ left) && (!((Tagesdaten)vec.elementAt(r)).vergleiche(pivot))){
```

```
            r = r-1;
        }
        if(l < r) {
            exchange(l, r);
        }
    }
    middle = r;
    exchange (left, middle);
    return middle;
}

/* Hilfsprozedur fuer den Quicksort
 */
void exchange(int i, int j) {
    Tagesdaten t = (Tagesdaten)vec.elementAt(i);
    vec.setElementAt(vec.elementAt(j),i);
    vec.setElementAt(t,j);
}

/* Ruft Quicksort auf
 */
void sortiere() {
    qsort(0, vec.size()-1);
}

/* Schliesst das Fenster und gibt seine Ressourcen frei.
 */
void schliesseFenster() {
    this.dispose();
}

/* Beendet das Programm komplett: alle Fenster werden geschlossen.
 */
void beendeProgramm() {
    System.exit(0);
}

/* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
   Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
   anderen Ereignisse wird "myaction" aufgerufen.
*/
public boolean action(Event event, Object target) {
    if(event.id == Event.ACTION_EVENT) {
        if(event.target == schliessen) {
            schliesseFenster();
            return true;
        }
        else if(event.target == beenden) {
            beendeProgramm();
            return true;
        } else {
            return myaction(event, target);
        }
    } else {
        return false;
    }
}

/* Wird von "action" aufgerufen.
*/
boolean myaction(Event ev, Object target){
    return true;
}
```

```
    /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
       zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
       noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
       "stelleDar" aufgerufen.
     */
    void stelleDar(){
      leseDaten(dateiname);
      zeigeTabelleAn();
    }

    void zeigeTabelleAn(){
      textarea = new TextArea(20,40);
      p2.add("Center",textarea);
      sortiere();
      for(int i = 0; i < vec.size(); i++) {
        textarea.append((((Tagesdaten)vec.elementAt(i)).uhrzeit).stunde + ":" +
                        (((Tagesdaten)vec.elementAt(i)).uhrzeit).minute + "  " +
                        ((Tagesdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                        ((Tagesdaten)vec.elementAt(i)).name + "  " +
                        ((Tagesdaten)vec.elementAt(i)).kurs.kurs + "\n");
      }
      if(vec.size() == 0) {
        textarea.append("Fuer das heutige Datum, " + heute.tag + "." +
                        heute.monat + "." + heute.jahr +
                        ", existieren keine Daten\n");
      }
      this.pack();
      this.show();
    }
}


/* Klasse zur Repraesentation eines einzelnen Kursinformations-Datensatzes
   a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
*/
class Tagesdaten {
    Datum datum;
    Uhrzeit uhrzeit;
    String wertpapierkennnummer;
    String name;
    Kurs kurs;

    Tagesdaten(Datum d, Uhrzeit u, String wkn, String n, Kurs k) {
      datum = d;
      uhrzeit = u;
      wertpapierkennnummer = wkn;
      name = n;
      kurs = k;
    }

    public boolean vergleiche(Tagesdaten t) {
      return (this.uhrzeit).kleinergleich(t.uhrzeit);
    }
}

/* Klasse zur Repraesentation eines einzelnen Kassakursinformations-Datensatzes
   a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
*/

class Zeitraumdaten {
    Datum datum;
    String wertpapierkennnummer;
    String name;
    Kurs kassakurs;
```

```
  Zeitraumdaten(Datum d, String wkn, String n, Kurs k) {
    datum = d;
    wertpapierkennnummer = wkn;
    name = n;
    kassakurs = k;
  }

  public boolean vergleiche(Zeitraumdaten z) {
    return (this.datum).kleinergleich(z.datum);
  }
}


/* Klasse zur Repraesentation eines Datums, incl. Berechnung des
   Datums 'vor einer Woche' und 'vor einem Monat'.
   (Anmerkung zum Experiment:
    Diese Routinen sind vereinfacht, um den Code zu verkuerzen.
    Sie funktionieren nicht wirklich korrekt. Das soll uns hier aber mal
    gerade nicht stoeren...)
*/

class Datum {
  int tag;
  int monat;
  int jahr;

  Datum() {
    tag = 0;
    monat = 0;
    jahr = 0;
  }

  Datum(String s){
    tag = new Integer(s.substring(0,2)).intValue();
    monat = new Integer(s.substring(3,5)).intValue();
    jahr = new Integer(s.substring(6,8)).intValue();
  }

  Datum(int t, int m, int j){
    tag = t;
    monat = m;
    jahr = j;
  }

  void setzeDatum(String s) {
    tag = new Integer(s.substring(0,2)).intValue();
    monat = new Integer(s.substring(3,5)).intValue();
    jahr = new Integer(s.substring(6,8)).intValue();
  }

  void setzeDatum(int t, int m, int j) {
    tag = t;
    monat = m;
    jahr = j;
  }

  boolean kleinergleich(Datum d) {
    if(this.jahr < d.jahr) {
      return true;
    } else if(this.jahr == d.jahr){
      if(this.monat < d.monat) {
        return true;
      } else if(this.monat == d.monat){
```

```
            if(this.tag ≤ d.tag) {
                return true;
            }
        }
    }
    return false;
}

Datum monatzurueck() {
    Datum d = new Datum();
    d.tag = tag;
    d.jahr = jahr;
    if(monat == 1) {
        d.monat = 12;
        d.jahr = jahr - 1;
    } else {
        d.monat = monat - 1;
    }
    return d;
}

/* vereinfachte Berechnung: Alle Monate haben 30 Tage
 */
Datum wochezurueck() {
    Datum d = new Datum();
    d.tag = tag - 7;
    d.monat = monat;
    d.jahr = jahr;
    if(d.tag < 1) {
        d.tag = d.tag + 30;
        d.monat = monat - 1;
        if(d.monat < 1) {
            d.monat = d.monat + 12;
            d.jahr = jahr - 1;
        }
    }
    return d;
}

void print() {
    System.out.println(" Datum = " + tag + "." + monat + "." + jahr);
}
}

/* Klasse zur Repraesentation einer Uhrzeit
*/

class Uhrzeit {
    int stunde;
    int minute;

    Uhrzeit(String s) {
        stunde = (new Integer(s.substring(0,2))).intValue();
        minute = (new Integer(s.substring(3,5))).intValue();
    }

    boolean kleinergleich(Uhrzeit u) {
        if(this.stunde < u.stunde) {
            return true;
        } else if(this.stunde == u.stunde) {
            if(this.minute ≤ u.minute) {
                return true;
            }
        }
```

```
        return false;
    }
}


/* Klasse zur Repraesentation eines Kurses oder Kassakurses
*/

class Kurs {
    float kurs;

    Kurs(String s){
        kurs = (new Float(s)).floatValue();
    }
}


/* Klasse Tagesgewinne

     Implementiert ein Fenster, in dem eine tabellarische Darstellung der
     prozentualen Unterschiede zwischen dem ersten und letzten Kurs
     jeder Aktie am heutigen Datum angezeigt wird.
     Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
     des Programmes, berechnet fuer das Abrufen von Informationen keine
     Kosten (Kosten werden von einem Sponsor getragen).
     Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
     Daten jedoch daraus erzeugt werden, wird durch die Methode
     erzeugeDatenvector() entschieden. Diese entscheidet durch eine
     selektiere()-Methode, welche Datensaetze ueberhaupt ausgewaehlt werden sollen.
     Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
     Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
     werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

     Methoden:
     Tagesgewinne(String)  Konstruktor, der den Titel setzt, einen Sponsor
                           einfuegt, die Kosten auf 0.00 setzt und die Datei
                           mit den Daten angibt.
     leseDaten(String)     liest Daten aus einer Datei und schreibt sie mit Hilfe
                           der Methode erzeugeDatenvector() in den Vector vec
     selektiere(String s)  entscheidet, ob der String ausgewaehlt werden
                           soll, konkret: ob er von dem angegebenen Datum ist
                           und die entsprechende Wertpapierkennnummer hat
     qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                           sind Methoden, die einen Sortieralgorithmus zur
                           Verfuegung stellen.
     schliesseFenster()    kehrt zum aufrufenden Fenster zurueck
     beendeProgramm()      beendet das Programm komplett
     action(Event, Object) verarbeitet Interaktionsereignisse
     myaction(Event, Object) wird in "action" aufgerufen
*/

class Tagesgewinne extends Frame implements Konstanten {
    Panel p1 = new Panel();
    Panel p2 = new Panel();
    Panel p3 = new Panel();
    Button schliessen = new Button("Schliessen");
    Button beenden = new Button("Beenden");
    float kosten;
    String dateiname;
    Label sponsor;
    Vector vec = new Vector();
    TextArea textarea;
    Vector v = new Vector();

```

```
/* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
     bestimmt das Layout.
     Panel p1 enthaelt die Sponsorinformation
     Panel p2 dient zum Anzeigen der angeforderten Informationen,
     Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
                 des Programms.
     Die Kosten werden auf 0.00 gesetzt.
*/

Tagesgewinne(String titel) {
  super(titel);
  this.setLayout(new BorderLayout());
  p1.setLayout(new FlowLayout());
  add("North",p1);
  p2.setLayout(new BorderLayout());
  add("Center",p2);
  p3.setLayout(new FlowLayout());
  add("South",p3);
  p3.add(schliessen);
  p3.add(beenden);
  sponsor = new Label("Diese Seiten werden gesponsort von xxy");
  p1.add(sponsor);
  kosten = 0.0f;
  setzeDateiname();
}

void setzeDateiname(String s) {
  dateiname = s;
}

void setzeDateiname() {
  dateiname = kurse;
}

/* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
    Methode erzeugeDatenvector eine Datenstruktur, in der die
    relevanten Daten gespeichert werden.
 */
void leseDaten(String dateiname) {
  try{
    BufferedReader br = new BufferedReader(new FileReader(dateiname));
    String s;
    try{
      while((s = br.readLine())≠null) {
        if(!s.startsWith("#")) {
          erzeugeDatenvector(s);
        }
      }
    } catch (IOException e) {
      System.err.println("Lesefehler bei Datei " + dateiname);
    }
  } catch (FileNotFoundException e) {
    System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
  }
}

/* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
    gespeichert sind. Dieses Objekt wird im Vektor vec gepeichert
 */
void erzeugeDatenvector(String s) {
  if(selektiere(s) == true) {
    Datum datum = new Datum(s.substring(0,8));
    Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
    Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
```

```
            String wkn = s.substring(14,24);
            String name = s.substring(25, s.length()-7);
            Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
            vec.addElement(tagesdaten);
        }
    }

    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,8));
        return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
    }

    /* Beschreibung des Quicksort siehe oben
     */
    void qsort(int left, int right) {
        int middle;
        if(left < right){
            middle = partition(left, right);
            qsort(left,middle-1);
            qsort(middle+1, right);
        }
    }

    /* Hilfsprozedur fuer den Quicksort
     */
    int partition(int left, int right) {
        Tagesdaten pivot = (Tagesdaten)vec.elementAt(left);
        int l = left;
        int r = right;
        int middle;
        while(l < r) {
            while((l <= right) && (((Tagesdaten)vec.elementAt(l)).vergleiche(pivot))){
                l = l+1;
            }
            while((r >= left) && (!((Tagesdaten)vec.elementAt(r)).vergleiche(pivot))){
                r = r-1;
            }
            if(l < r) {
                exchange(l, r);
            }
        }
        middle = r;
        exchange (left, middle);
        return middle;
    }

    /* Hilfsprozedur fuer den Quicksort
     */
    void exchange(int i, int j) {
        Tagesdaten t = (Tagesdaten)vec.elementAt(i);
        vec.setElementAt(vec.elementAt(j),i);
        vec.setElementAt(t,j);
    }

    /* Ruft Quicksort auf
     */
    void sortiere() {
        qsort(0, vec.size()-1);
    }

    /* Schliesst das Fenster und gibt seine Ressourcen frei.
     */
    void schliesseFenster() {
```

```
    this.dispose();
  }

  /* Beendet das Programm komplett: alle Fenster werden geschlossen.
   */
  void beendeProgramm() {
    System.exit(0);
  }

  /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
       Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
       anderen Ereignisse wird "myaction" aufgerufen.
  */
  public boolean action(Event event, Object target) {
    if(event.id == Event.ACTION_EVENT) {
      if(event.target == schliessen) {
        schliesseFenster();
        return true;
      }
      else if(event.target == beenden) {
        beendeProgramm();
        return true;
      } else {
        return myaction(event, target);
      }
    } else {
      return false;
    }
  }

  /* Wird von "action" aufgerufen.
  */
  boolean myaction(Event ev, Object target){
    return true;
  }

  /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
       zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
       noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
       "stelleDar" aufgerufen.
   */
  void stelleDar(){
    leseDaten(dateiname);
    zeigeTabelleAn();
  }

  boolean schonAbgearbeitet(String s) {
    int i = 0;
    while(i < v.size()) {
      if(((String)v.elementAt(i)).compareTo(s)==0) {
        i = v.size();
        return true;
      } else {
        i++;
      }
    }
    v.addElement(s);
    return false;
  }

  float tagesgewinn(String wertpapierkennnummer, int start) {
    float min = 0.0f;
    float max = 0.0f;
    int i = start;
```

```
      while((wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) ≠ 0) && (i <
vec.size())) {
        i = i+1;
      }
      min = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
      while(i < vec.size()) {
        if(wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
          max = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
        }
        i = i+1;
      }
      //      System.out.println("Min = " + min + "      Max = " + max);
      if(min == 0.0f) {
        return 0;
      } else {
        return ((max - min)/min);
      }
    }

    void zeigeTabelleAn() {
      textarea = new TextArea(20,40);
      p2.add("Center",textarea);
      boolean keineDaten = true;
      sortiere();
      for(int i = 0; i < vec.size(); i++) {
        if(! schonAbgearbeitet(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
          textarea.append(((Tagesdaten)(vec.elementAt(i))).name + "  " +
                          ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + "  "
                          + tagesgewinn(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
                          "\n");
          keineDaten = false;
        }
      }
      if(keineDaten == true) {
        textarea.append("Fuer das heutige Datum, " + heute.tag + "." +
                        heute.monat + "." + heute.jahr +
                        ", existieren keine Daten\n");
      }
      this.pack();
      this.show();
    }
}


/* Klasse LetzteTageskurse

   Implementiert ein Fenster, in dem eine tabellarische Darstellung der
   letzten vorhandenen heutigen Kurse fuer alle Aktientitel angezeigt wird.
   Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
   des Programmes, berechnet fuer das Abrufen von Informationen keine
   Kosten (Kosten werden von einem Sponsor getragen).
   Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
   Daten jedoch daraus erzeugt werden, wird durch die Methode
   erzeugeDatenvector() entschieden. Diese entscheidet durch eine
   selektiere()-Methode, welche Datensaetze ueberhaupt ausgewaehlt werden sollen.
   Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
   Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
   werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

   Methoden:
   LetzteTageskurse(String)  Konstruktor, der den Titel setzt, einen Sponsor
                             einfuegt, die Kosten auf 0.00 setzt und die Datei
                             mit den Daten angibt.
   leseDaten(String)     liest Daten aus einer Datei und schreibt sie mit Hilfe
```

```
                              der Methode erzeugeDatenvector() in den Vector vec
    selektiere(String s) entscheidet, ob der String ausgewaehlt werden
                              soll, konkret: ob er von dem angegebenen Datum ist
                              und die entsprechende Wertpapierkennnummer hat
    qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                              sind Methoden, die einen Sortieralgorithmus zur
                              Verfuegung stellen.
    schliesseFenster()   kehrt zum aufrufenden Fenster zurueck
    beendeProgramm()      beendet das Programm komplett
    action(Event, Object) verarbeitet Interaktionsereignisse
    myaction(Event, Object) wird in "action" aufgerufen
*/


class LetzteTageskurse extends Frame implements Konstanten {
  Panel p1 = new Panel();
  Panel p2 = new Panel();
  Panel p3 = new Panel();
  Button schliessen = new Button("Schliessen");
  Button beenden = new Button("Beenden");
  float kosten;
  String dateiname;
  Label sponsor;
  Vector vec = new Vector();
  TextArea textarea;
  Vector v = new Vector();

  /*  Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
      bestimmt das Layout.
      Panel p1 enthaelt die Sponsorinformation
      Panel p2 dient zum Anzeigen der angeforderten Informationen,
      Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
               des Programms.
      Die Kosten werden auf 0.00 gesetzt.
  */

  LetzteTageskurse(String titel) {
    super(titel);
    this.setLayout(new BorderLayout());
    p1.setLayout(new FlowLayout());
    add("North",p1);
    p2.setLayout(new BorderLayout());
    add("Center",p2);
    p3.setLayout(new FlowLayout());
    add("South",p3);
    p3.add(schliessen);
    p3.add(beenden);
    sponsor = new Label("Diese Seiten werden gesponsort von xxy");
    p1.add(sponsor);
    kosten = 0.0f;
    setzeDateiname();
  }

  void setzeDateiname(String s) {
    dateiname = s;
  }

  void setzeDateiname() {
    dateiname = kurse;
  }

  /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
     Methode erzeugeDatenvector eine Datenstruktur, in der die
     relevanten Daten gespeichert werden.
```

```
 */
void leseDaten(String dateiname) {
  try{
    BufferedReader br = new BufferedReader(new FileReader(dateiname));
    String s;
    try{
      while((s = br.readLine())≠null) {
        if(!s.startsWith("#")) {
          erzeugeDatenvector(s);
        }
      }
    } catch (IOException e) {
      System.err.println("Lesefehler bei Datei " + dateiname);
    }
  } catch (FileNotFoundException e) {
    System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
  }
}

/* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
   gespeichert sind. Dieses Objekt wird im Vektor vec gepeichert
 */
void erzeugeDatenvector(String s) {
  if(selektiere(s) == true) {
    Datum datum = new Datum(s.substring(0,8));
    Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
    Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
    String wkn = s.substring(14,24);
    String name = s.substring(25, s.length()-7);
    Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
    vec.addElement(tagesdaten);
  }
}

boolean selektiere(String s) {
  Datum d = new Datum();
  d.setzeDatum(s.substring(0,8));
  return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
}

/* Beschreibung des Quicksort siehe oben
 */
void qsort(int left, int right) {
  int middle;
  if(left < right){
    middle = partition(left, right);
    qsort(left,middle-1);
    qsort(middle+1, right);
  }
}

/* Hilfsprozedur fuer den Quicksort
 */
int partition(int left, int right) {
  Tagesdaten pivot = (Tagesdaten)vec.elementAt(left);
  int l = left;
  int r = right;
  int middle;
  while(l < r) {
    while((l ≤ right) && (((Tagesdaten)vec.elementAt(l)).vergleiche(pivot))){
      l = l+1;
    }
    while((r ≥ left) && (!((Tagesdaten)vec.elementAt(r)).vergleiche(pivot))){
      r = r-1;
```

```
      }
      if(l < r) {
        exchange(l, r);
      }
    }
    middle = r;
    exchange (left, middle);
    return middle;
}

/* Hilfsprozedur fuer den Quicksort
 */
void exchange(int i, int j) {
    Tagesdaten t = (Tagesdaten)vec.elementAt(i);
    vec.setElementAt(vec.elementAt(j),i);
    vec.setElementAt(t,j);
}

/* Ruft Quicksort auf
 */
void sortiere() {
    qsort(0, vec.size()-1);
}

/* Schliesst das Fenster und gibt seine Ressourcen frei.
 */
void schliesseFenster() {
    this.dispose();
}

/* Beendet das Programm komplett: alle Fenster werden geschlossen.
 */
void beendeProgramm() {
    System.exit(0);
}

/* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
   Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
   anderen Ereignisse wird "myaction" aufgerufen.
*/
public boolean action(Event event, Object target) {
    if(event.id == Event.ACTION_EVENT) {
      if(event.target == schliessen) {
        schliesseFenster();
        return true;
      }
      else if(event.target == beenden) {
        beendeProgramm();
        return true;
      } else {
        return myaction(event, target);
      }
    } else {
      return false;
    }
}

/* Wird von "action" aufgerufen.
*/
boolean myaction(Event ev, Object target){
    return true;
}

/* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
```

```
      zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
      noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
      "stelleDar" aufgerufen.
   */
  void stelleDar(){
    leseDaten(dateiname);
    zeigeTabelleAn();
  }

  boolean schonAbgearbeitet(String s) {
    int i = 0;
    while(i < v.size()) {
      if(((String)v.elementAt(i)).compareTo(s)==0) {
        i = v.size();
        return true;
      } else {
        i++;
      }
    }
    v.addElement(s);
    return false;
  }

  float letztetageskurse(String wertpapierkennnummer, int start) {
    float letzter = 0.0f;
    int i = start;
    while(i < vec.size()) {
      if(wertpapierkennnummer.compareTo(
             ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
        letzter = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
      }
      i = i+1;
    }
    return letzter;
  }

  void zeigeTabelleAn() {
    textarea = new TextArea(20,40);
    p2.add("Center",textarea);
    sortiere();
    for(int i = 0; i < vec.size(); i++) {
      if(!schonAbgearbeitet(
               ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
        textarea.append(((Tagesdaten)(vec.elementAt(i))).name + "  " +
                        ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + "  " +
                        letztetageskurse(
                        ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
                        "\n");
      }
    }
    if(vec.size() == 0) {
      textarea.append("Fuer das heutige Datum, " +
                      heute.tag + "." + heute.monat + "." +
                      heute.jahr + ", existieren keine Daten\n");
    }
    this.pack();
    this.show();
  }
}


/* Klasse WochenTabelle

   Implementiert ein Fenster, in dem eine tabellarische Darstellung der
```

```
   Kassakurse aller Aktientitel fuer die letzten 7 Tage angezeigt wird.
   Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
   des Programmes, berechnet fuer das Abrufen von Informationen keine
   Kosten (Kosten werden von einem Sponsor getragen).
   Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
   Daten jedoch daraus erzeugt werden, wird durch die Methode
   erzeugeDatenvector() entschieden. Diese entscheidet durch eine
   selektiere()-Methode, welche Datensaetze ueberhaupt ausgewaehlt werden sollen.
   Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
   Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
   werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

   Methoden:
   WochenTabelle(String)  Konstruktor, der den Titel setzt, einen Sponsor
                          einfuegt, die Kosten auf 0.00 setzt und die Datei
                          mit den Daten angibt.
   leseDaten(String)      liest Daten aus einer Datei und schreibt sie mit Hilfe
                          der Methode erzeugeDatenvector() in den Vector vec
   selektiere(String s)   entscheidet, ob der String ausgewaehlt werden
                          soll, konkret: ob er von dem angegebenen Datum ist
                          und die entsprechende Wertpapierkennnummer hat
   qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                          sind Methoden, die einen Sortieralgorithmus zur
                          Verfuegung stellen.
   schliesseFenster()     kehrt zum aufrufenden Fenster zurueck
   beendeProgramm()       beendet das Programm komplett
   action(Event, Object)  verarbeitet Interaktionsereignisse
   myaction(Event, Object) wird in "action" aufgerufen
*/

class WochenTabelle extends Frame implements Konstanten {
  Panel p1 = new Panel();
  Panel p2 = new Panel();
  Panel p3 = new Panel();
  Button schliessen = new Button("Schliessen");
  Button beenden = new Button("Beenden");
  float kosten;
  String dateiname;
  Label sponsor;
  Vector vec = new Vector();
  Datum vonDatum;
  TextArea textarea;

  /*  Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
      bestimmt das Layout.
      Panel p1 enthaelt die Sponsorinformation
      Panel p2 dient zum Anzeigen der angeforderten Informationen,
      Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
               des Programms.
      Die Kosten werden auf 0.00 gesetzt.
  */

  WochenTabelle(String titel) {
    super(titel);
    vonDatum = heute.wochezurueck();
    this.setLayout(new BorderLayout());
    p1.setLayout(new FlowLayout());
    add("North",p1);
    p2.setLayout(new BorderLayout());
    add("Center",p2);
    p3.setLayout(new FlowLayout());
    add("South",p3);
    p3.add(schliessen);
    p3.add(beenden);
```

```
        sponsor = new Label("Diese Seiten werden gesponsort von xxy");
        p1.add(sponsor);
        kosten = 0.0f;
        setzeDateiname();
    }

    void setzeDateiname(String s) {
        dateiname = s;
    }

    void setzeDateiname() {
        dateiname = kassakurse;
    }

    /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
       Methode erzeugeDatenvector eine Datenstruktur, in der die
       relevanten Daten gespeichert werden.
     */
    void leseDaten(String dateiname) {
        try{
            BufferedReader br = new BufferedReader(new FileReader(dateiname));
            String s;
            try{
                while((s = br.readLine())≠null) {
                    if(!s.startsWith("#")) {
                        erzeugeDatenvector(s);
                    }
                }
            } catch (IOException e) {
                System.err.println("Lesefehler bei Datei " + dateiname);
            }
        } catch (FileNotFoundException e) {
            System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden"
);
        }
    }

    /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
       gespeichert sind. Dieses Objekt wird im Vektor vec gepeichert
     */
    void erzeugeDatenvector(String s){
        if(selektiere(s) == true) {
            Datum datum = new Datum(s.substring(0,8));
            String wkn = s.substring(9,18);
            String name = s.substring(19, s.length()-7);
            Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
            Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
            vec.addElement(zrd);
        }
    }

    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,8));
        return (!d.kleinergleich(vonDatum));
    }

    /* Beschreibung des Quicksort siehe oben
     */
    void qsort(int left, int right) {
        int middle;
        if(left < right){
            middle = partition(left, right);
            qsort(left,middle-1);
```

```
        qsort(middle+1, right);
    }
}

/* Hilfsprozedur fuer den Quicksort
 */
int partition(int left, int right) {
    Zeitraumdaten pivot = (Zeitraumdaten)vec.elementAt(left);
    int l = left;
    int r = right;
    int middle;
    while(l < r) {
        while((l <= right) && (((Zeitraumdaten)vec.elementAt(l)).vergleiche(pivot))){
            l = l+1;
        }
        while((r >= left) && (!((Zeitraumdaten)vec.elementAt(r)).vergleiche(pivot))){
            r = r-1;
        }
        if(l < r) {
            exchange(l, r);
        }
    }
    middle = r;
    exchange (left, middle);
    return middle;
}

/* Hilfsprozedur fuer den Quicksort
 */
void exchange(int i, int j) {
    Zeitraumdaten z = (Zeitraumdaten)vec.elementAt(i);
    vec.setElementAt(vec.elementAt(j),i);
    vec.setElementAt(z,j);
}

/* Ruft Quicksort auf
 */
void sortiere() {
    qsort(0, vec.size()-1);
}

/* Schliesst das Fenster und gibt seine Ressourcen frei.
 */
void schliesseFenster() {
    this.dispose();
}

/* Beendet das Programm komplett: alle Fenster werden geschlossen.
 */
void beendeProgramm() {
    System.exit(0);
}

/* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
   Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
   anderen Ereignisse wird "myaction" aufgerufen.
*/
public boolean action(Event event, Object target) {
    if(event.id == Event.ACTION_EVENT) {
        if(event.target == schliessen) {
            schliesseFenster();
            return true;
        }
        else if(event.target == beenden) {
```

```
            beendeProgramm();
            return true;
         } else {
            return myaction(event, target);
         }
      } else {
         return false;
      }
   }

   /* Wird von "action" aufgerufen.
   */
   boolean myaction(Event ev, Object target) {
      return true;
   }

   /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
       zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
       noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
       "stelleDar" aufgerufen.
    */
   void stelleDar(){
      leseDaten(dateiname);
      zeigeTabelleAn();
   }

   void zeigeTabelleAn(){
      textarea = new TextArea(20,40);
      p2.add("Center",textarea);
      sortiere();
      for(int i = 0; i < vec.size(); i++) {
         textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + "." +
                                 (((Zeitraumdaten)vec.elementAt(i)).datum).monat + "." +
                                 (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + "  " +
                                 ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                                 ((Zeitraumdaten)vec.elementAt(i)).name + "  " +
                                 ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
      }
      if(vec.size() == 0) {
         textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");
      }
      this.pack();
      this.show();
   }
}


/* Klasse MonatsTabelle

    Implementiert ein Fenster, in dem eine tabellarische Darstellung der
    Kassakurse aller Aktientitel fuer die letzten 30 Tage (genauer gesagt:
    seit dem gleichen Tag im Vormonat) angezeigt wird.
    Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
    des Programmes, berechnet fuer das Abrufen von Informationen keine
    Kosten (Kosten werden von einem Sponsor getragen).
    Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
    Daten jedoch daraus erzeugt werden, wird durch die Methode
    erzeugeDatenvector() entschieden. Diese entscheidet durch eine
    selektiere()-Methode, welche Datensaetze ueberhaupt ausgewaehlt werden sollen.
    Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
    Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
    werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

    Methoden:
```

```
    MonatsTabelle(String) Konstruktor, der den Titel setzt, einen Sponsor
                          einfuegt, die Kosten auf 0.00 setzt und die Datei
                          mit den Daten angibt.
    leseDaten(String)     liest Daten aus einer Datei und schreibt sie mit Hilfe
                          der Methode erzeugeDatenvector() in den Vector vec
    selektiere(String s) entscheidet, ob der String ausgewaehlt werden
                          soll, konkret: ob er von dem angegebenen Datum ist
                          und die entsprechende Wertpapierkennnummer hat
    qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                          sind Methoden, die einen Sortieralgorithmus zur
                          Verfuegung stellen.
    schliesseFenster()  kehrt zum aufrufenden Fenster zurueck
    beendeProgramm()      beendet das Programm komplett
    action(Event, Object) verarbeitet Interaktionsereignisse
    myaction(Event, Object) wird in "action" aufgerufen
*/

class MonatsTabelle extends Frame implements Konstanten {
  Panel p1 = new Panel();
  Panel p2 = new Panel();
  Panel p3 = new Panel();
  Button schliessen = new Button("Schliessen");
  Button beenden = new Button("Beenden");
  float kosten;
  String dateiname;
  Label sponsor;
  Vector vec = new Vector();
  Datum vonDatum;
  TextArea textarea;

  /*  Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
      bestimmt das Layout.
      Panel p1 enthaelt die Sponsorinformation
      Panel p2 dient zum Anzeigen der angeforderten Informationen,
      Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
               des Programms.
      Die Kosten werden auf 0.00 gesetzt.
  */

  MonatsTabelle(String titel) {
    super(titel);
    vonDatum = heute.monatzurueck();
    this.setLayout(new BorderLayout());
    p1.setLayout(new FlowLayout());
    add("North",p1);
    p2.setLayout(new BorderLayout());
    add("Center",p2);
    p3.setLayout(new FlowLayout());
    add("South",p3);
    p3.add(schliessen);
    p3.add(beenden);
    sponsor = new Label("Diese Seiten werden gesponsort von xxy");
    p1.add(sponsor);
    kosten = 0.0f;
    setzeDateiname();
  }

  void setzeDateiname(String s) {
    dateiname = s;
  }

  void setzeDateiname() {
    dateiname = kassakurse;
  }
```

```

/* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
   Methode erzeugeDatenvector eine Datenstruktur, in der die
   relevanten Daten gespeichert werden.
 */
void leseDaten(String dateiname) {
   try{
      BufferedReader br = new BufferedReader(new FileReader(dateiname));
      String s;
      try{
         while((s = br.readLine())≠null) {
            if(!s.startsWith("#")) {
               erzeugeDatenvector(s);
            }
         }
      } catch (IOException e) {
         System.err.println("Lesefehler bei Datei " + dateiname);
      }
   } catch (FileNotFoundException e) {
      System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
   }
}

/* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
   gespeichert sind. Dieses Objekt wird im Vektor vec gepeichert
 */
void erzeugeDatenvector(String s){
   if(selektiere(s) == true) {
      Datum datum = new Datum(s.substring(0,8));
      String wkn = s.substring(9,18);
      String name = s.substring(19, s.length()-7);
      Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
      Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
      vec.addElement(zrd);
   }
}

boolean selektiere(String s) {
   Datum d = new Datum();
   d.setzeDatum(s.substring(0,8));
   return (!d.kleinergleich(vonDatum));
}

/* Beschreibung des Quicksort siehe oben
 */
void qsort(int left, int right) {
   int middle;
   if(left < right){
      middle = partition(left, right);
      qsort(left,middle-1);
      qsort(middle+1, right);
   }
}

/* Hilfsprozedur fuer den Quicksort
 */
int partition(int left, int right) {
   Zeitraumdaten pivot = (Zeitraumdaten)vec.elementAt(left);
   int l = left;
   int r = right;
   int middle;
   while(l < r) {
      while((l ≤ right) && (((Zeitraumdaten)vec.elementAt(l)).vergleiche(pivot))){
         l = l+1;
```

```
        }
        while((r ≥ left) && (!((Zeitraumdaten)vec.elementAt(r)).vergleiche(pivot))){
            r = r-1;
        }
        if(l < r) {
            exchange(l, r);
        }
    }
    middle = r;
    exchange (left, middle);
    return middle;
}

/* Hilfsprozedur fuer den Quicksort
 */
void exchange(int i, int j) {
    Zeitraumdaten z = (Zeitraumdaten)vec.elementAt(i);
    vec.setElementAt(vec.elementAt(j),i);
    vec.setElementAt(z,j);
}

/* Ruft Quicksort auf
 */
void sortiere() {
    qsort(0, vec.size()-1);
}

/* Schliesst das Fenster und gibt seine Ressourcen frei.
 */
void schliesseFenster() {
    this.dispose();
}

/* Beendet das Programm komplett: alle Fenster werden geschlossen.
 */
void beendeProgramm() {
    System.exit(0);
}

/* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
   Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
   anderen Ereignisse wird "myaction" aufgerufen.
*/
public boolean action(Event event, Object target) {
    if(event.id == Event.ACTION_EVENT) {
        if(event.target == schliessen) {
            schliesseFenster();
            return true;
        }
        else if(event.target == beenden) {
            beendeProgramm();
            return true;
        } else {
            return myaction(event, target);
        }
    } else {
        return false;
    }
}

/* Wird von "action" aufgerufen.
*/
boolean myaction(Event ev, Object target) {
    return true;
```

```
    }

    /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
       zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
       noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
       "stelleDar" aufgerufen.
    */
    void stelleDar(){
      leseDaten(dateiname);
      zeigeTabelleAn();
    }

    void zeigeTabelleAn(){
      textarea = new TextArea(20,40);
      p2.add("Center",textarea);
      sortiere();
      for(int i = 0; i < vec.size(); i++) {
        textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + "." +
                        (((Zeitraumdaten)vec.elementAt(i)).datum).monat + "." +
                        (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + "  " +
                        ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                        ((Zeitraumdaten)vec.elementAt(i)).name + "  " +
                        ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
      }
      if(vec.size() == 0) {
        textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");
      }
      this.pack();
      this.show();
    }
}


/* Klasse MonatsDynamikTabelle

   Implementiert ein Fenster, in dem eine tabellarische Darstellung
   der prozentualen Unterschiede zwischen aufeinanderfolgenden
   Kassakursen aller Aktientitel fuer den letzten Monat angezeigt wird.
   Die Klasse enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
   des Programmes, berechnet fuer das Abrufen von Informationen keine
   Kosten (Kosten werden von einem Sponsor getragen).
   Des weiteren stellt sie eine Methode zum Dateneinlesen bereit. Welche
   Daten jedoch daraus erzeugt werden, wird durch die Methode
   erzeugeDatenvector() entschieden. Diese entscheidet durch eine
   selektiere()-Methode, welche Datensaetze ueberhaupt ausgewaehlt werden sollen.
   Diese Klasse enthaelt darueberhinaus Methoden zur Darstellung des Charts.
   Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
   werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

   Methoden:
   MonatsDynamikTabelle(String) Konstruktor, der den Titel setzt, einen Sponsor
                                einfuegt, die Kosten auf 0.00 setzt und die Datei
                                mit den Daten angibt.
   leseDaten(String)     liest Daten aus einer Datei und schreibt sie mit Hilfe
                                der Methode erzeugeDatenvector() in den Vector vec
   selektiere(String s) entscheidet, ob der String ausgewaehlt werden
                                soll, konkret: ob er von dem angegebenen Datum ist
                                und die entsprechende Wertpapierkennnummer hat
   qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                                sind Methoden, die einen Sortieralgorithmus zur
                                Verfuegung stellen.
   schliesseFenster()   kehrt zum aufrufenden Fenster zurueck
   beendeProgramm()       beendet das Programm komplett
   action(Event, Object) verarbeitet Interaktionsereignisse
```

```
   myaction(Event, Object) wird in "action" aufgerufen
*/

class MonatsDynamikTabelle extends Frame implements Konstanten {
   Panel p1 = new Panel();
   Panel p2 = new Panel();
   Panel p3 = new Panel();
   Button schliessen = new Button("Schliessen");
   Button beenden = new Button("Beenden");
   float kosten;
   String dateiname;
   Label sponsor;
   Vector vec = new Vector();
   Vector dyn = new Vector();
   Datum vonDatum;
   TextArea textarea;

   /*  Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und
       bestimmt das Layout.
       Panel p1 enthaelt die Sponsorinformation
       Panel p2 enthaelt eine Auswahlmoeglichkeit des gewuenschten Wertpapiers
       Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden
                des Programms.
       Die Kosten werden auf 0.00 gesetzt.
   */

   MonatsDynamikTabelle(String titel) {
      super(titel);
      vonDatum = heute.monatzurueck();
      this.setLayout(new BorderLayout());
      p1.setLayout(new FlowLayout());
      add("North",p1);
      p2.setLayout(new BorderLayout());
      add("Center",p2);
      p3.setLayout(new FlowLayout());
      add("South",p3);
      p3.add(schliessen);
      p3.add(beenden);
      sponsor = new Label("Diese Seiten werden gesponsort von xxy");
      p1.add(sponsor);
      kosten = 0.0f;
      setzeDateiname();
   }

   void setzeDateiname(String s) {
      dateiname = s;
   }

   void setzeDateiname() {
      dateiname = kassakurse;
   }

   /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
      Methode erzeugeDatenvector eine Datenstruktur, in der die
      relevanten Daten gespeichert werden.
    */
   void leseDaten(String dateiname) {
      try{
         BufferedReader br = new BufferedReader(new FileReader(dateiname));
         String s;
         try{
            while((s = br.readLine())≠null) {
               if(!s.startsWith("#")) {
                  erzeugeDatenvector(s);
```

```
            }
        }
    } catch (IOException e) {
        System.err.println("Lesefehler bei Datei " + dateiname);
    }
  } catch (FileNotFoundException e) {
    System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
  }
}

/* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
   gespeichert sind. Dieses Objekt wird im Vektor vec gepeichert
 */
void erzeugeDatenvector(String s){
  if(selektiere(s) == true) {
    Datum datum = new Datum(s.substring(0,8));
    String wkn = s.substring(9,18);
    String name = s.substring(19, s.length()-7);
    Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
    Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
    vec.addElement(zrd);
  }
}

boolean selektiere(String s) {
  Datum d = new Datum();
  d.setzeDatum(s.substring(0,8));
  return (!d.kleinergleich(vonDatum));
}

/* Beschreibung des Quicksort siehe oben
 */
void qsort(int left, int right) {
  int middle;
  if(left < right){
    middle = partition(left, right);
    qsort(left,middle-1);
    qsort(middle+1, right);
  }
}

/* Hilfsprozedur fuer den Quicksort
 */
int partition(int left, int right) {
  Zeitraumdaten pivot = (Zeitraumdaten)vec.elementAt(left);
  int l = left;
  int r = right;
  int middle;
  while(l < r) {
    while((l <= right) && (((Zeitraumdaten)vec.elementAt(l)).vergleiche(pivot))){
      l = l+1;
    }
    while((r >= left) && (!((Zeitraumdaten)vec.elementAt(r)).vergleiche(pivot))){
      r = r-1;
    }
    if(l < r) {
      exchange(l, r);
    }
  }
  middle = r;
  exchange (left, middle);
  return middle;
}
```

```
/* Hilfsprozedur fuer den Quicksort
 */
void exchange(int i, int j) {
   Zeitraumdaten z = (Zeitraumdaten)vec.elementAt(i);
   vec.setElementAt(vec.elementAt(j),i);
   vec.setElementAt(z,j);
}

/* Ruft Quicksort auf
 */
void sortiere() {
   qsort(0, vec.size()-1);
}

/* Schliesst das Fenster und gibt seine Ressourcen frei.
 */
void schliesseFenster() {
   this.dispose();
}

/* Beendet das Programm komplett: alle Fenster werden geschlossen.
 */
void beendeProgramm() {
   System.exit(0);
}

/* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
    Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
    anderen Ereignisse wird "myaction" aufgerufen.
*/
public boolean action(Event event, Object target) {
   if(event.id == Event.ACTION_EVENT) {
      if(event.target == schliessen) {
         schliesseFenster();
         return true;
      }
      else if(event.target == beenden) {
         beendeProgramm();
         return true;
      } else {
         return myaction(event, target);
      }
   } else {
      return false;
   }
}

/* Wird von "action" aufgerufen.
*/
boolean myaction(Event ev, Object target) {
   return true;
}

/* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
    zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
    noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
    "stelleDar" aufgerufen.
 */
void stelleDar(){
   leseDaten(dateiname);
   zeigeTabelleAn();
}

float berechneDynamik(int i) {
```

```
    //suche den letzen Wert des gleichen Wertpapiers, um die prozentuale
    //Abweichung des neuen Wertes vom alten zu bestimmen.
    int j = i-1;
    boolean gefunden = false;
    while ((j ≥ 0) && !gefunden) {
      if ((((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer).compareTo(
            ((Zeitraumdaten)vec.elementAt(j)).wertpapierkennnummer) == 0) {
        gefunden = true;
        return ((((((Zeitraumdaten)vec.elementAt(i)).kassakurs).kurs -
                    (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)/
                     (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)*100);
      }
      j--;
    }
    return 0.0f;
  }

  void zeigeTabelleAn(){
    textarea = new TextArea(20,40);
    p2.add("Center",textarea);
    sortiere();
    for(int i = 0; i < vec.size(); i++) {
      textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + "." +
                      (((Zeitraumdaten)vec.elementAt(i)).datum).monat + "." +
                      (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + "  " +
                      ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                      ((Zeitraumdaten)vec.elementAt(i)).name + "  " +
                      berechneDynamik(i) + "%\n");
    }
    if(vec.size() == 0) {
      textarea.append("Fuer diesen Zeitraum existieren keine Daten\n");
    }
    this.pack();
    this.show();
  }

}


/* graphische Darstellung aller Kurse eines Aktientitels fuer den heutigen
   Tag.
*/
class TagesChartAnzeiger extends Canvas {
  Vector v;
  int xSize;
  int ySize;
  int abstand;

  TagesChartAnzeiger(Vector vec, int xSize, int ySize) {
    v = vec;
    this.xSize = xSize;
    this.ySize = ySize;
    if(vec.size() ≠ 0) {
      abstand = xSize/vec.size();
    }
    setSize(xSize, ySize);
  }

  public void paint(Graphics g) {
    g.drawRect(0, 0, xSize-1, ySize-1);
    for(int i = 0; i < v.size()-1; i++) {
      drawLine(g, i);
    }
  }
```

```

  void drawLine(Graphics g, int i){
    g.drawLine(i*abstand,
               ySize - (int)((((TagesChartWert)(v.elementAt(i))).kurs).kurs),
               (i+1)*abstand,
               ySize - (int)((((TagesChartWert)(v.elementAt(i+1))).kurs).kurs));
  }
}


/* graphische Darstellung aller Kassakurse eines Aktientitels fuer die
   letzten 30 Tage.
*/
class MonatsChartAnzeiger extends Canvas {
  Vector v;
  int xSize;
  int ySize;
  int abstand;

  MonatsChartAnzeiger(Vector vec, int xSize, int ySize) {
    v = vec;
    this.xSize = xSize;
    this.ySize = ySize;
    if(vec.size() ≠ 0) {
      abstand = xSize/vec.size();
    }
    setSize(xSize, ySize);
  }

  public void paint(Graphics g) {
    g.drawRect(0, 0, xSize-1, ySize-1);
    for(int i = 0; i < v.size()-1; i++) {
      drawLine(g, i);
    }
  }

  void drawLine(Graphics g, int i){
    g.drawLine(i*abstand,
               ySize - (int)((((MonatsChartWert)(v.elementAt(i))).kassakurs).kurs),
               (i+1)*abstand,
               ySize - (int)((((MonatsChartWert)(v.elementAt(i+1))).kassakurs).kurs));
  }
}


/* Kern des Hauptprogramms: Auswahlmenue der einzelnen Darstellungsformen.
*/
class Auswahl extends Frame {
  Button beenden = new Button("Beenden");
  Button ok = new Button("OK");
  Panel p = new Panel();
  CheckboxGroup cbg = new CheckboxGroup();
  Checkbox cbtt = new Checkbox("Tagestabelle", cbg, true);
  Checkbox cbwt = new Checkbox("Wochentabelle", cbg, false);
  Checkbox cbmt = new Checkbox("Monatstabelle", cbg, false);
  Checkbox cbdt = new Checkbox("Dynamiktabelle", cbg, false);
  Checkbox cbtg = new Checkbox("Tagesgewinne", cbg, false);
  Checkbox cbltk = new Checkbox("letzte Tageskurse", cbg, false);
  Checkbox cbtc = new Checkbox("Tages-Chart", cbg, false);
  Checkbox cbmc = new Checkbox("Monats-Chart", cbg, false);

  Auswahl(String titel) {
    super(titel);
    setLayout(new BorderLayout());
```

```
    p.setLayout(new GridLayout(6,1));
    p.add(cbtt);
    p.add(cbwt);
    p.add(cbmt);
    p.add(cbdt);
    p.add(cbtg);
    p.add(cbltk);
    p.add(cbtc);
    p.add(cbmc);
    add("Center",p);
    add("East",ok);
    add("South",beenden);
    pack();
    show();
  }

  public boolean action(Event ev, Object target){
    if(ev.id == Event.ACTION_EVENT) {
      if(ev.target == ok) {
        if(cbg.getSelectedCheckbox() == cbtt) {
          TagesTabelle tt = new TagesTabelle("Tagestabelle");
          tt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbwt) {
          WochenTabelle wt = new WochenTabelle("Wochentabelle");
          wt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbmt) {
          MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
          mt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbdt) {
          MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
          dt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbtg) {
          Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
          tg.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbltk) {
          LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
          ltg.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbtc) {
          TagesChart tc = new TagesChart("Tageschart");
          tc.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbmc) {
          MonatsChart mc = new MonatsChart("Monatschart");
          mc.stelleDar();
          return true;
        }
      } else if(ev.target == beenden) {
        System.exit(0);
        return true;
      }
    }
    return false;
  }
}


/* Programmeinstiegspunkt
*/
```

```
class Boerse {
    public static void main(String args[]) {
        Auswahl a = new Auswahl("Boersen-Information");
    }
}
```

B.1.2 Boerse.java (3 levels of inheritance)

```
/**************************************************************************/
/*Programmautor:  Barbara  Unger                                          */
/*Datum:           1997-06-03                                             */
/*Gruppe:          Boerse3                                                */
/**************************************************************************/

import java.awt.*;
import java.util.Vector;
import java.util.Enumeration;
import java.util.Date;
import java.io.*;

/* Programm zum Betrachten von Boersenkursen.
   Abfolge der Klassen in dieser Datei:

   interface Konstanten
   abstract class BoersenInfo extends Frame implements Konstanten
   abstract class KostenloseInfos extends BoersenInfo
   abstract class KostenplichtigeInfos extends BoersenInfo
   abstract class Chart extends KostenloseInfos
   class TagesChart extends Chart
   class TagesChartWert implements Comparable
   class MonatsChart extends Chart
   class MonatsChartWert implements Comparable
   abstract class Tabelle extends KostenloseInfos
   class TagesTabelle extends Tabelle
   class Tagesdaten implements Comparable
   class Zeitraumdaten implements Comparable
   class Datum
   class Uhrzeit
   class Kurs
   class Tagesgewinne extends Tabelle
   class LetzteTageskurse extends Tabelle
   class WochenTabelle extends Tabelle
   class MonatsTabelle extends Tabelle
   class MonatsDynamikTabelle extends Tabelle
   interface Comparable
   abstract class ChartAnzeiger extends Canvas
   class TagesChartAnzeiger extends ChartAnzeiger
   class MonatsChartAnzeiger extends ChartAnzeiger
   class Auswahl extends Frame
   class Boerse
*/

interface Konstanten {
//  static Date date = new Date();
//  static Datum heute = new Datum(date.getDay(), date.getMonth(), date.getYear());
  static Datum heute = new Datum(27, 5, 97);
  static String kurse = "KURSE";
  static String kassakurse = "KASSAKURSE";
}


/* abstrakte Klasse BoersenInfo Implementiert Fenster, in denen
   Boersen-Informationen angezeigt werden.  Stellt sicher, dass in
   jeder der daraus abgeleiteten Klassen Knoepfe zum Schliessen des
```

```
   Fensters und zum Beenden des Programmes existieren.

   Methoden:
      BoersenInfo(String)        erzeugt das Fenster und setzt den Titel
      schliesseFenster()         kehrt zum aufrufenden Fenster zurueck
      beendeProgramm()           beendet das Programm komplett
      action(Event, Object)      verarbeitet Interaktionsereignisse
      abstract myaction(Event, Object)
                                 wird in "action" aufgerufen; Unterklassen
                                 koennen hier durch ueberschreiben
                                 spezielle Ereignisbehandlung
                                 implementieren
*/

abstract class BoersenInfo extends Frame implements Konstanten{
   Panel p1 = new Panel();
   Panel p2 = new Panel();
   Panel p3 = new Panel();
   Button schliessen = new Button("Schliessen");
   Button beenden = new Button("Beenden");
   float kosten;

   /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und bestimmt das Layout.
         Panel p1 dient zum Darstellen von Sponsor- oder Kosteninformationen,
         Panel p2 dient zum Anzeigen der angeforderten Informationen,
         Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden des Programms.
   */
   BoersenInfo(String titel) {
      super(titel);
      this.setLayout(new BorderLayout());
      p1.setLayout(new FlowLayout());
      add("North",p1);
      p2.setLayout(new BorderLayout());
      add("Center",p2);
      p3.setLayout(new FlowLayout());
      add("South",p3);
      p3.add(schliessen);
      p3.add(beenden);
   }

   /* Schliesst das Fenster und gibt seine Ressourcen frei.
    */
   void schliesseFenster() {
      this.dispose();
   }

   /* Beendet das Programm komplett: alle Fenster werden geschlossen.
    */
   void beendeProgramm() {
      System.exit(0);
   }

   /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
      Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
      anderen Ereignisse wird "myaction" aufgerufen.
   */
   public boolean action(Event event, Object target) {
      if(event.id == Event.ACTION_EVENT) {
         if(event.target == schliessen) {
            schliesseFenster();
            return true;
         }
         else if(event.target == beenden) {
            beendeProgramm();
```

```
            return true;
         } else {
            return myaction(event, target);
         }
      } else {
         return false;
      }
   }

   /* Wird von "action" aufgerufen. Kann in Unterklassen ueberschrieben
      werden, um eine spezielle Ereignisbehandlung zu
      implementieren. (Diese Konstruktion verhindert, dass die Behandlung
      der Knoepfe "Schliessen" und "Beenden" jedesmal neu implementiert
      werden muss.
   */
   abstract boolean myaction(Event ev, Object target);
}


/* abstrakte Klasse KostenloseInfos
   Stellt sicher, dass fuer das Abrufen von Informationen durch alle
   aus ihr abgeleiteten Klassen keine Kosten anfallen. (Kosten werden
   von einem Sponsor getragen).
   Des weiteren stellt sie eine Methode zum Dateneinlesen
   bereit. Welche Daten jedoch daraus erzeugt werden ist zu diesem
   Zeitpunkt unklar und wird erst in einer Unterklasse entschieden
   durch die Methode erzeugeDatenvector(). Ausserdem wird nun schon
   ein Sponsorlabel fuer alle kostenlosen Infos gesetzt.

   Methoden:
   KostenloseInfos(String) Konstruktor, setzt Fenstertitel, fuegt einen Sponsor ein
                           und setzt die Kosten auf 0.00.
   leseDaten(String)  liest Daten aus einer Datei und schreibt sie mit Hilfe
                           der Methode erzeugeDatenvector() in den Vector vec
   qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                           sind Methoden, die einen Sortieralgorithmus fuer alle
                           Unterklassen zur Verfuegung stellen. Da in jeder
                           Unterklasse unterschiedliche Datentypen zu sortieren
                           sind, wurde Schablonenmethode angewandt, die
                           vergleiche()-Methode ist nur abstrakt realisiert
                           und wird vom jeweiligen Datentyp erst implementiert.
*/

abstract class KostenloseInfos extends BoersenInfo {
   String dateiname;
   Label sponsor;
   Vector vec = new Vector();

   /* Konstruktor, setzt den Titel des Fensters, fuegt einen Sponsor in das
        Fenster ein (in p1) und setzt die Kosten auf 0.00
     */
   KostenloseInfos(String titel) {
      super(titel);
      sponsor = new Label("Diese Seiten werden gesponsort von xxy");
      p1.add(sponsor);
      kosten = 0.0f;
   }

   /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
        abstrakte Methode erzeugeDatenvector eine Datenstruktur, in der die
        relevanten Daten gespeichert werden.
     */
   void leseDaten(String dateiname) {
      try{
```

```
            BufferedReader br = new BufferedReader(new FileReader(dateiname));
            String s;
            try{
                while((s = br.readLine())≠null) {
                    if(!s.startsWith("#")) {
                        erzeugeDatenvector(s);
                    }
                }
            } catch (IOException e) {
                System.err.println("Lesefehler bei Datei " + dateiname);
            }
        } catch (FileNotFoundException e) {
            System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
        }
    }

    /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
       gespeichert sind. Dieses Objekt wird im Vektor vec gepeichert
     */
    abstract void erzeugeDatenvector(String s);

    /* Beschreibung des Quicksort siehe oben
     */
    void qsort(int left, int right) {
        int middle;
        if(left < right){
            middle = partition(left, right);
            qsort(left,middle-1);
            qsort(middle+1, right);
        }
    }

    /* Hilfsprozedur fuer den Quicksort
     */
    int partition(int left, int right) {
        Comparable pivot = (Comparable)vec.elementAt(left);
        int l = left;
        int r = right;
        int middle;
        while(l < r) {
            while((l ≤ right) && (((Comparable)vec.elementAt(l)).vergleiche(pivot))){
                l = l+1;
            }
            while((r ≥ left) && (!((Comparable)vec.elementAt(r)).vergleiche(pivot))){
                r = r-1;
            }
            if(l < r) {
                exchange(l, r);
            }
        }
        middle = r;
        exchange (left, middle);
        return middle;
    }

    /* Hilfsprozedur fuer den Quicksort
     */
    void exchange(int i, int j) {
        Comparable c = (Comparable)vec.elementAt(i);
        vec.setElementAt(vec.elementAt(j),i);
        vec.setElementAt(c,j);
    }

    /* Ruft Quicksort auf
```

```
    */
  void sortiere() {
    qsort(0, vec.size()-1);
  }
}


/* Klasse KostenpflichtigeInfos
    enthaelt einen Rahmen fuer Informationen, die vom Kunden bezahlt werden
    muessen, wie beispielsweise Prognosen, etc.
 */
/*Wird nicht ausprogrammiert.*/
abstract class KostenplichtigeInfos extends BoersenInfo {

  KostenplichtigeInfos(String titel) {
    super(titel);
    kosten = 1.0f;
  }
}


/* abstrakte Klasse Chart
    Diese Klasse enthaelt allgemeine Methoden zur Darstellung eines Charts.
    Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
    werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

    Methoden:
    Chart(String)          Konstruktor, setzt den Fenstertitel
    void stelleDar()       stellt eine Dialogbox dar.
    myaction()             erweitert die action-Methode der Oberklasse.
    abstract zeigeCanvasAn()   liefert ein Canvas zurueck.
*/

abstract class Chart extends KostenloseInfos {
  int xChartSize = 200;
  int yChartSize = 200;
  String wkn;
  Label l = new Label();
  TextField t = new TextField("700045634",9);
  Button ok = new Button("ok");

  /* Konstruktor, ruft den Konstruktor von KostenloseInfos auf, und fuegt in den
      freien Rahmen p2 eine Auswahlmoeglichkeit des gewuenschten Wertpapiers ein
   */
  Chart(String titel) {
    super(titel);
  }

  void stelleDar(){
    p2.add("North",l);
    p2.add("Center",t);
    p2.add("South",ok);
    pack();
    show();
  }


  /* myaction erweitert die action-Methode, so dass nun beim Druck des
      Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
      aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
      und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
   */
  boolean myaction(Event ev, Object target){
    if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
```

```
            wkn = t.getText();
            wkn.trim();
            p2.removeAll();
            leseDaten(dateiname);
            p2.add("Center",zeigeCanvasAn());
            pack();
            show();
            return true;
        }
        return false;
    }

/*  liefert ein Canvas zurueck. Was das Canvas enthaelt, wird erst in der Unterklasse
    bestimmt.
 */
    abstract Canvas zeigeCanvasAn();
}


/* Klasse TagesChart
   In dieser Klasse wird das Erzeugen des Datenvektors implementiert.
   Diese entscheidet durch eine selektiere()-Methode, welche
   Datensaetze ueberhaupt ausgewaehlt werden sollen.

   Methoden:
     TagesChart(String)            Konstruktor, der den Titel setzt und die Datei
                                   mit den Daten angibt.
     erzeugeDatenvector(String)  bekommt eine Datensatz in Stringform und
                                   entscheidet mit Hilfe von selektiere(), ob der
                                   Datensatz in den Datenvektor eingefuegt
                                   werden soll.
     selektiere(String s)          entscheidet, ob der String ausgewaehlt werden
                                   soll, konkret: ob er von dem angegebenen
                                   Datum ist und die entsprechende
                                   Wertpapierkennnummer hat
     zeigeCanvasAn()                    erzeugt ein Objekt der Canvas-Klasse, in der der
                                   Chart aus den selektierten Daten, die im Datenvektor stehen,
                                   angezeigt wird.
*/

class TagesChart extends Chart {
    TagesChartWert tcw;

/* Konstruktor, der den Titel setzt und die Datei mit den Daten angibt
 */
    TagesChart(String titel) {
        super(titel);
        dateiname = kurse;
        l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Tageschart an");
    }

/* bekommt eine Datensatz in Stringform und entscheidet mit Hilfe von selektiere(),
   ob der Datensatz in den Datenvektor eingefuegt werden soll.
 */
    void erzeugeDatenvector(String s){
        if(selektiere(s) == true) {
            tcw = new TagesChartWert(s);
            vec.addElement(tcw);
        }
    }

/* entscheidet, ob der String ausgewaehlt werden soll, konkret: ob er von dem angegebenen
   Datum ist und die entsprechende Wertpapierkennnummer hat
 */
```

```
    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,8));
        if((heute.tag == d.tag) && (heute.monat ==d.monat) && (heute.jahr == d.jahr)) {
            return (wkn.compareTo(s.substring(15,24))==0);
        }
        return false;
    }

/* erzeugt ein Objekt der Canvas-Klasse, in der der Chart aus den selektierten Daten, die im
      Datenvektor stehen, angezeigt wird.
  */
    Canvas zeigeCanvasAn() {
        return new TagesChartAnzeiger(vec, xChartSize, yChartSize);
    }
}



/*Klasse TagesChartWert
    ist ein Objekttyp, der aus einer Uhrzeit und einem Kurswert besteht.
    Das sind genau die Daten, die man zum Anzeigen eines Tagescharts braucht. Die Klasse hat
    eine vergleiche-Methode, so dass man einen Vector mit Objekten dieses Typs mit
    der sortier()-Methode aus der Klasse KostenloseInfos sortieren kann.

    Methoden:
        TagesChartWert(String) bekommt einen String, in dem die Uhrzeit
                               und der Kurs enthalten sind.
        vergleiche(Comparable c) vergleicht die Uhrzeiten
*/

class TagesChartWert implements Comparable {
    Uhrzeit uhrzeit;
    Kurs kurs;

/* bekommt einen String, in dem die Uhrzeit und der Kurs enthalten sind.
  */
    TagesChartWert(String s) {
        uhrzeit = new Uhrzeit(s.substring(9,15));
        kurs = new Kurs(s.substring(s.length()-7, s.length()));
    }

/* vergleicht die Uhrzeiten zweier TagesChartWert-Objekte
  */
    public boolean vergleiche(Comparable c) {
        return ((this.uhrzeit).kleinergleich(((TagesChartWert)c).uhrzeit));
    }
}


/* Klasse MonatsChart
      In dieser Klasse wird das Erzeugen des Datenvectors implementiert.
      Diese entscheidet durch eine selektiere()-Methode, welche
      Datensaetze ueberhaupt ausgewaehlt werden sollen.

      Methoden:
          MonatsChart(String)          Konstruktor, der den Titel setzt und die Datei
                                           mit den Daten angibt.
          erzeugeDatenvector(String)  bekommt eine Datensatz in Stringform und
                                           entscheidet mit Hilfe von slektiere(), ob der
                                           Datensatz in den Datenvector eingefuegt
                                           werden soll.
          selektiere(String s)         entscheidet, ob der String ausgewaehlt werden
                                           soll, konkret: ob er von dem angegebenen
```

```
                                      Datum ist und die entsprechende
                                      Wertpapierkennnummer hat
     zeigeCanvasAn()                     erzeugt ein Objekt der Canvas-Klasse, in der der
                                      Chart angezeigt wird.
*/

class MonatsChart extends Chart {
  MonatsChartWert mcw;
  Datum letzterMonat = new Datum();

  MonatsChart(String titel) {
    super(titel);
    l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Monatschart an");
    dateiname = kassakurse;
    letzterMonat = heute.monatzurueck();
  }

  void erzeugeDatenvector(String s){
    if(selektiere(s) == true) {
      mcw = new MonatsChartWert(s);
      vec.addElement(mcw);
    }
  }

  boolean selektiere(String s) {
    Datum d = new Datum();
    d.setzeDatum(s.substring(0,8));
    if(!d.kleinergleich(letzterMonat)) {
      return (wkn.compareTo(s.substring(9,18))==0);
    }
    return false;
  }

  Canvas zeigeCanvasAn() {
    return new MonatsChartAnzeiger(vec, xChartSize, yChartSize);
  }
}


/*Klasse MonatsChartWert
  ist ein Objekttyp, der aus einem Datum und einem Kassakurswert besteht,
  Daten, die man zum Anzeigen eines Monatschart braucht. Die Klasse hat
  eine vergleiche-Methode, so dass man einen Vector mit diesem Typ mit
  der sortier()-Methode aus der Klasse KostenloseInfos sortieren kann.

  Methoden:
    MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
                                       und der Kurs enthalten sind.
    vergleiche(Comparable c)         vergleicht das Datum
*/

class MonatsChartWert implements Comparable {
  Kurs kassakurs;
  Datum datum;

  MonatsChartWert(String s){
    kassakurs = new Kurs(s.substring(s.length()-6, s.length()));
    datum = new Datum(s.substring(0,9));
  }

  public boolean vergleiche(Comparable c){
    return ((this.datum).kleinergleich(((MonatsChartWert)c).datum));
  }
}
```

```


/* abstrakte Klasse Tabelle.
   Gemeinsame Oberklasse fuer alle tabellarischen Darstellungen von
   Kursen oder Kassakursen.
*/

abstract class Tabelle extends KostenloseInfos {
  String dateiname;

  Tabelle(String titel) {
    super(titel);
    setzeDateiname();
  }

  /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
     zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
     noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
     "stelleDar" aufgerufen.
   */
  void stelleDar(){
    leseDaten(dateiname);
    zeigeTabelleAn();
  }

  abstract void setzeDateiname();
  abstract void zeigeTabelleAn();
}

/* Klasse zur tabellarischen Darstellung aller Kurse eines Tages
*/

class TagesTabelle extends Tabelle {
  TextArea textarea;

  TagesTabelle(String titel) {
    super(titel);
  }

  void setzeDateiname(String s) {
    dateiname = s;
  }

  void setzeDateiname() {
    dateiname = kurse;
  }

  boolean selektiere(String s) {
    Datum d = new Datum();
    d.setzeDatum(s.substring(0,8));
    return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
  }


  void erzeugeDatenvector(String s) {
    if(selektiere(s) == true) {
      Datum datum = new Datum(s.substring(0,8));
      Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
      Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
      String wkn = s.substring(14,24);
      String name = s.substring(25, s.length()-7);
      Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
      vec.addElement(tagesdaten);
    }
```

```
  }

  boolean myaction(Event ev, Object target){
    return true;
  }

  void zeigeTabelleAn(){
    textarea = new TextArea(20,40);
    p2.add("Center",textarea);
    sortiere();
    for(int i = 0; i < vec.size(); i++) {
      textarea.append((((Tagesdaten)vec.elementAt(i)).uhrzeit).stunde + ":" +
                      (((Tagesdaten)vec.elementAt(i)).uhrzeit).minute + "  " +
                      ((Tagesdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                      ((Tagesdaten)vec.elementAt(i)).name + "  " +
                      ((Tagesdaten)vec.elementAt(i)).kurs.kurs + "\n");
    }
    if(vec.size() == 0) {
      textarea.append("Fuer das heutige Datum, " + heute.tag + "." +
                      heute.monat + "." + heute.jahr +
                      ", existieren keine Daten\n");
    }
    this.pack();
    this.show();
  }
}


/* Klasse zur Repraesentation eines einzelnen Kursinformations-Datensatzes
*/

class Tagesdaten implements Comparable {
  Datum datum;
  Uhrzeit uhrzeit;
  String wertpapierkennnummer;
  String name;
  Kurs kurs;

  Tagesdaten(Datum d, Uhrzeit u, String wkn, String n, Kurs k) {
    datum = d;
    uhrzeit = u;
    wertpapierkennnummer = wkn;
    name = n;
    kurs = k;
  }

  public boolean vergleiche(Comparable c) {
    return (this.uhrzeit).kleinergleich(((Tagesdaten)c).uhrzeit);
  }
}


/* Klasse zur Repraesentation eines einzelnen Kassakursinformations-Datensatzes
*/

class Zeitraumdaten implements Comparable {
  Datum datum;
  String wertpapierkennnummer;
  String name;
  Kurs kassakurs;

  Zeitraumdaten(Datum d, String wkn, String n, Kurs k) {
    datum = d;
    wertpapierkennnummer = wkn;
```

```
    name = n;
    kassakurs = k;
  }

  public boolean vergleiche(Comparable c) {
    return (this.datum).kleinergleich(((Zeitraumdaten)c).datum);
  }
}


/* Klasse zur Repraesentation eines Datums, incl. Berechnung des
   Datums 'vor einer Woche' und 'vor einem Monat'.
   (Anmerkung zum Experiment:
    Diese Routinen sind vereinfacht, um den Code zu verkuerzen.
    Sie funktionieren nicht wirklich korrekt. Das soll uns hier aber mal
    gerade nicht stoeren...)
*/

class Datum {
  int tag;
  int monat;
  int jahr;

  Datum() {
    tag = 0;
    monat = 0;
    jahr = 0;
  }

  Datum(String s){
    tag = new Integer(s.substring(0,2)).intValue();
    monat = new Integer(s.substring(3,5)).intValue();
    jahr = new Integer(s.substring(6,8)).intValue();
  }

  Datum(int t, int m, int j){
    tag = t;
    monat = m;
    jahr = j;
  }

  void setzeDatum(String s) {
    tag = new Integer(s.substring(0,2)).intValue();
    monat = new Integer(s.substring(3,5)).intValue();
    jahr = new Integer(s.substring(6,8)).intValue();
  }

  void setzeDatum(int t, int m, int j) {
    tag = t;
    monat = m;
    jahr = j;
  }

  boolean kleinergleich(Datum d) {
    if(this.jahr < d.jahr) {
      return true;
    } else if(this.jahr == d.jahr){
      if(this.monat < d.monat) {
        return true;
      } else if(this.monat == d.monat){
        if(this.tag ≤ d.tag) {
          return true;
        }
      }
```

```
    }
    return false;
  }

  Datum monatzurueck() {
    Datum d = new Datum();
    d.tag = tag;
    d.jahr = jahr;
    if(monat == 1) {
      d.monat = 12;
      d.jahr = jahr - 1;
    } else {
      d.monat = monat - 1;
    }
    return d;
  }

  /* vereinfachte Berechnung: Alle Monate haben 30 Tage
   */
  Datum wochezurueck() {
    Datum d = new Datum();
    d.tag = tag - 7;
    d.monat = monat;
    d.jahr = jahr;
    if(d.tag < 1) {
      d.tag = d.tag + 30;
      d.monat = monat - 1;
      if(d.monat < 1) {
        d.monat = d.monat + 12;
        d.jahr = jahr - 1;
      }
    }
    return d;
  }

  void print() {
    System.out.println(" Datum = " + tag + "." + monat + "." + jahr);
  }
}

/* Klasse zur Repraesentation einer Uhrzeit
*/

class Uhrzeit {
  int stunde;
  int minute;

  Uhrzeit(String s) {
    stunde = (new Integer(s.substring(0,2))).intValue();
    minute = (new Integer(s.substring(3,5))).intValue();
  }

  boolean kleinergleich(Uhrzeit u) {
    if(this.stunde < u.stunde) {
      return true;
    } else if(this.stunde == u.stunde) {
      if(this.minute <= u.minute) {
        return true;
      }
    }
    return false;
  }
}
```

```
/* Klasse zur Repraesentation eines Kurses oder Kassakurses
*/

class Kurs {
   float kurs;

   Kurs(String s){
      kurs = (new Float(s)).floatValue();
   }
}


/* Tabellarische Darstellung der prozentualen Unterschiede zwischen dem
    ersten und letzten Kurs jeder Aktie am heutigen Datum.
*/

class Tagesgewinne extends Tabelle {
   TextArea textarea;
   Vector v = new Vector();

   Tagesgewinne(String titel) {
      super(titel);
   }

   void setzeDateiname(String s) {
      dateiname = s;
   }

   void setzeDateiname() {
      dateiname = kurse;
   }

   boolean selektiere(String s) {
      Datum d = new Datum();
      d.setzeDatum(s.substring(0,8));
      return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
   }

   void erzeugeDatenvector(String s) {
      if(selektiere(s) == true) {
         Datum datum = new Datum(s.substring(0,8));
         Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
         Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
         String wkn = s.substring(14,24);
         String name = s.substring(25, s.length()-7);
         Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
         vec.addElement(tagesdaten);
      }
   }

   boolean myaction(Event ev, Object target){
      return true;
   }

   boolean schonAbgearbeitet(String s) {
      int i = 0;
      while(i < v.size()) {
         if(((String)v.elementAt(i)).compareTo(s)==0) {
            i = v.size();
            return true;
         } else {
            i++;
         }
      }
```

```
      v.addElement(s);
      return false;
   }

   float tagesgewinn(String wertpapierkennnummer, int start) {
      float min = 0.0f;
      float max = 0.0f;
      int i = start;
      while((wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) != 0) && (i <
vec.size())) {
         i = i+1;
      }
      min = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
      while(i < vec.size()) {
         if(wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
            max = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
         }
         i = i+1;
      }
      //      System.out.println("Min = " + min + "        Max = " + max);
      if(min == 0.0f) {
         return 0;
      } else {
         return ((max - min)/min);
      }
   }

   void zeigeTabelleAn() {
      textarea = new TextArea(20,40);
      p2.add("Center",textarea);
      boolean keineDaten = true;
      sortiere();
      for(int i = 0; i < vec.size(); i++) {
         if(! schonAbgearbeitet(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
            textarea.append(((Tagesdaten)(vec.elementAt(i))).name + "  " +
                                ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + "  "
                                + tagesgewinn(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
                                "\n");
            keineDaten = false;
         }
      }
      if(keineDaten == true) {
         textarea.append("Fuer das heutige Datum, " + heute.tag + "." +
                           heute.monat + "." + heute.jahr +
                           ", existieren keine Daten\n");
      }
      this.pack();
      this.show();
   }
}

/* tabellarische Darstellung der letzten vorhandenen heutigen Kurse fuer alle
    Aktientitel.
*/

class LetzteTageskurse extends Tabelle {
   TextArea textarea;
   Vector v = new Vector();

   LetzteTageskurse(String titel) {
      super(titel);
   }

   void setzeDateiname(String s) {
```

```
        dateiname = s;
    }

    void setzeDateiname() {
        dateiname = kurse;
    }

    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,8));
        return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
    }

    void erzeugeDatenvector(String s) {
        if(selektiere(s) == true) {
            Datum datum = new Datum(s.substring(0,8));
            Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
            Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
            String wkn = s.substring(14,24);
            String name = s.substring(25, s.length()-7);
            Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
            vec.addElement(tagesdaten);
        }
    }

    boolean myaction(Event ev, Object target){
        return true;
    }

    boolean schonAbgearbeitet(String s) {
        int i = 0;
        while(i < v.size()) {
            if(((String)v.elementAt(i)).compareTo(s)==0) {
                i = v.size();
                return true;
            } else {
                i++;
            }
        }
        v.addElement(s);
        return false;
    }

    float letztetageskurse(String wertpapierkennnummer, int start) {
        float letzter = 0.0f;
        int i = start;
        while(i < vec.size()) {
            if(wertpapierkennnummer.compareTo(
                    ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
                letzter = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
            }
            i = i+1;
        }
        return letzter;
    }

    void zeigeTabelleAn() {
        textarea = new TextArea(20,40);
        p2.add("Center",textarea);
        sortiere();
        for(int i = 0; i < vec.size(); i++) {
            if(!schonAbgearbeitet(
                        ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
                textarea.append(((Tagesdaten)(vec.elementAt(i))).name + "  " +
```

```
                                ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + "  " +
                                letztetageskurse(
                                ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
                                "\n");
            }
        }
        if(vec.size() == 0) {
            textarea.append("Fuer das heutige Datum, " +
                              heute.tag + "." + heute.monat + "." +
                              heute.jahr + ", existieren keine Daten\n");
        }
        this.pack();
        this.show();
    }
}


/* tabellarische Darstellung der Kassakurse aller Aktientitel fuer die
   letzten 7 Tage. Benutzt die Kassakurse.
*/

class WochenTabelle extends Tabelle {
    Datum vonDatum;
    TextArea textarea;

    WochenTabelle(String titel) {
        super(titel);
        vonDatum = heute.wochezurueck();
    }

    void setzeDateiname(String s) {
        dateiname = s;
    }

    void setzeDateiname() {
        dateiname = kassakurse;
    }

    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,8));
        return (!d.kleinergleich(vonDatum));
    }

    void erzeugeDatenvector(String s){
        if(selektiere(s) == true) {
            Datum datum = new Datum(s.substring(0,8));
            String wkn = s.substring(9,18);
            String name = s.substring(19, s.length()-7);
            Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
            Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
            vec.addElement(zrd);
        }
    }

    boolean myaction(Event ev, Object target) {
        return true;
    }

    void zeigeTabelleAn(){
        textarea = new TextArea(20,40);
        p2.add("Center",textarea);
        sortiere();
        for(int i = 0; i < vec.size(); i++) {
```

```
            textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + "." +
                            (((Zeitraumdaten)vec.elementAt(i)).datum).monat + "." +
                            (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + "  " +
                            ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                            ((Zeitraumdaten)vec.elementAt(i)).name + "  " +
                            ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
        }
        if(vec.size() == 0) {
            textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");
        }
        this.pack();
        this.show();
    }
}


/* tabellarische Darstellung der Kassakurse aller Aktientitel fuer die
   letzten 30 Tage (genauer gesagt: seit dem gleichen Tag im Vormonat).
   Benutzt die Kassakurse.
*/

class MonatsTabelle extends Tabelle {
    Datum vonDatum;
    TextArea textarea;

    MonatsTabelle(String titel) {
        super(titel);
        vonDatum = heute.monatzurueck();
    }

    void setzeDateiname(String s) {
        dateiname = s;
    }

    void setzeDateiname() {
        dateiname = kassakurse;
    }

    boolean selektiere(String s) {
        Datum d = new Datum();
        d.setzeDatum(s.substring(0,8));
        return (!d.kleinergleich(vonDatum));
    }

    void erzeugeDatenvector(String s){
        if(selektiere(s) == true) {
            Datum datum = new Datum(s.substring(0,8));
            String wkn = s.substring(9,18);
            String name = s.substring(19, s.length()-7);
            Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
            Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
            vec.addElement(zrd);
        }
    }

    boolean myaction(Event ev, Object target) {
        return true;
    }

    void zeigeTabelleAn(){
        textarea = new TextArea(20,40);
        p2.add("Center",textarea);
        sortiere();
        for(int i = 0; i < vec.size(); i++) {
```

```
        textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + "." +
                        (((Zeitraumdaten)vec.elementAt(i)).datum).monat + "." +
                        (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + "  " +
                        ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                        ((Zeitraumdaten)vec.elementAt(i)).name + "  " +
                        ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
      }
      if(vec.size() == 0) {
        textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");
      }
      this.pack();
      this.show();
  }
}


/* tabellarische Darstellung der prozentualen Unterschiede zwischen
   aufeinanderfolgenden Kassakursen aller Aktientitel fuer den
   letzten Monat. Benutzt die Kassakurse.
*/

class MonatsDynamikTabelle extends Tabelle {
  Vector dyn = new Vector();
  Datum vonDatum;
  TextArea textarea;


  MonatsDynamikTabelle(String titel) {
    super(titel);
    vonDatum = heute.monatzurueck();
  }

  void setzeDateiname(String s) {
    dateiname = s;
  }

  void setzeDateiname() {
    dateiname = kassakurse;
  }

  boolean selektiere(String s) {
    Datum d = new Datum();
    d.setzeDatum(s.substring(0,8));
    return (!d.kleinergleich(vonDatum));
  }

  void erzeugeDatenvector(String s){
    if(selektiere(s) == true) {
      Datum datum = new Datum(s.substring(0,8));
      String wkn = s.substring(9,18);
      String name = s.substring(19, s.length()-7);
      Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
      Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
      vec.addElement(zrd);
    }
  }

  boolean myaction(Event ev, Object target) {
    return true;
  }

  float berechneDynamik(int i) {
    //suche den letzen Wert des gleichen Wertpapiers, um die prozentuale
    //Abweichung des neuen Wertes vom alten zu bestimmen.
```

```
    int j = i-1;
    boolean gefunden = false;
    while ((j ≥ 0) && !gefunden) {
      if ((((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer).compareTo(
            ((Zeitraumdaten)vec.elementAt(j)).wertpapierkennnummer) == 0) {
        gefunden = true;
        return ((((((Zeitraumdaten)vec.elementAt(i)).kassakurs).kurs -
                    (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)/
                     (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)*100);
      }
      j--;
    }
    return 0.0f;
  }

  void zeigeTabelleAn(){
    textarea = new TextArea(20,40);
    p2.add("Center",textarea);
    sortiere();
    for(int i = 0; i < vec.size(); i++) {
      textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + "." +
                      (((Zeitraumdaten)vec.elementAt(i)).datum).monat + "." +
                      (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + "  " +
                      ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                      ((Zeitraumdaten)vec.elementAt(i)).name + "  " +
                      berechneDynamik(i) + "%\n");
    }
    if(vec.size() == 0) {
      textarea.append("Fuer diesen Zeitraum existieren keine Daten\n");
    }
    this.pack();
    this.show();
  }
}


/* Schnittstelle fuer 'vergleichbare' Datentypen (zum Sortieren).
   a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
*/

interface Comparable {
  public boolean vergleiche(Comparable c);
}

/* Oberklasse aller graphischen Anzeigeklassen
*/

abstract class ChartAnzeiger extends Canvas {
  Vector v;
  int xSize;
  int ySize;
  int abstand;

  ChartAnzeiger(Vector vec, int xSize, int ySize) {
    v = vec;
    this.xSize = xSize;
    this.ySize = ySize;
    if(vec.size() ≠ 0) {
      abstand = xSize/vec.size();
    }
    setSize(xSize, ySize);
  }

  public void paint(Graphics g) {
```

```
    g.drawRect(0, 0, xSize-1, ySize-1);
    for(int i = 0; i < v.size()-1; i++) {
      drawLine(g, i);
    }
  }

  abstract void drawLine(Graphics g, int i);
}


/* graphische Darstellung aller Kurse eines Aktientitels fuer den heutigen
   Tag.
*/

class TagesChartAnzeiger extends ChartAnzeiger {

  TagesChartAnzeiger(Vector vec, int xSize, int ySize) {
    super(vec, xSize, ySize);
  }

  void drawLine(Graphics g, int i){
    g.drawLine(i*abstand,
               ySize - (int)((((TagesChartWert)(v.elementAt(i))).kurs).kurs),
               (i+1)*abstand,
               ySize - (int)((((TagesChartWert)(v.elementAt(i+1))).kurs).kurs));
  }
}


/* graphische Darstellung aller Kassakurse eines Aktientitels fuer die
   letzten 30 Tage.
*/

class MonatsChartAnzeiger extends ChartAnzeiger {

  MonatsChartAnzeiger(Vector vec, int xSize, int ySize) {
    super(vec, xSize, ySize);
  }

  void drawLine(Graphics g, int i){
    g.drawLine(i*abstand,
               ySize - (int)((((MonatsChartWert)(v.elementAt(i))).kassakurs).kurs),
               (i+1)*abstand,
               ySize - (int)((((MonatsChartWert)(v.elementAt(i+1))).kassakurs).kurs));
  }
}

/* Kern des Hauptprogramms: Auswahlmenue der einzelnen Darstellungsformen.
*/

class Auswahl extends Frame {
  Button beenden = new Button("Beenden");
  Button ok = new Button("OK");
  Panel p = new Panel();
  CheckboxGroup cbg = new CheckboxGroup();
  Checkbox cbtt = new Checkbox("Tagestabelle", cbg, true);
  Checkbox cbwt = new Checkbox("Wochentabelle", cbg, false);
  Checkbox cbmt = new Checkbox("Monatstabelle", cbg, false);
  Checkbox cbdt = new Checkbox("Dynamiktabelle", cbg, false);
  Checkbox cbtg = new Checkbox("Tagesgewinne", cbg, false);
  Checkbox cbltk = new Checkbox("letzte Tageskurse", cbg, false);
  Checkbox cbtc = new Checkbox("Tages-Chart", cbg, false);
  Checkbox cbmc = new Checkbox("Monats-Chart", cbg, false);

```

```
  Auswahl(String titel) {
    super(titel);
    setLayout(new BorderLayout());
    p.setLayout(new GridLayout(6,1));
    p.add(cbtt);
    p.add(cbwt);
    p.add(cbmt);
    p.add(cbdt);
    p.add(cbtg);
    p.add(cbltk);
    p.add(cbtc);
    p.add(cbmc);
    add("Center",p);
    add("East",ok);
    add("South",beenden);
    pack();
    show();
  }

  public boolean action(Event ev, Object target){
    if(ev.id == Event.ACTION_EVENT) {
      if(ev.target == ok) {
        if(cbg.getSelectedCheckbox() == cbtt) {
          TagesTabelle tt = new TagesTabelle("Tagestabelle");
          tt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbwt) {
          WochenTabelle wt = new WochenTabelle("Wochentabelle");
          wt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbmt) {
          MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
          mt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbdt) {
          MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
          dt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbtg) {
          Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
          tg.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbltk) {
          LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
          ltg.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbtc) {
          TagesChart tc = new TagesChart("Tageschart");
          tc.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbmc) {
          MonatsChart mc = new MonatsChart("Monatschart");
          mc.stelleDar();
          return true;
        }
      } else if(ev.target == beenden) {
        System.exit(0);
        return true;
      }
    }
    return false;
  }
}
```

```
/* Programmeinstiegspunkt
*/

class Boerse {
  public static void main(String args[]) {
    Auswahl a = new Auswahl("Boersen-Information");
  }
}
```

B.1.3 Boerse.java (5 levels of inheritance)

```
/************************************************************************/
/*Programmautor:  Barbara  Unger                                        */
/*Datum:          1997-06-03                                            */
/*Gruppe:         Boerse5                                               */
/************************************************************************/

import java.awt.*;
import java.util.Vector;
import java.util.Enumeration;
import java.util.Date;
import java.io.*;

/* Programm zum Betrachten von Boersenkursen.
   Abfolge der Klassen in dieser Datei:

   interface Konstanten
   abstract class BoersenInfo extends Frame impelements Konstanten
   abstract class KostenloseInfos extends BoersenInfo
   abstract class KostenplichtigeInfos extends BoersenInfo
   abstract class Chart extends KostenloseInfos
   class TagesChart extends Chart
   class TagesChartWert implements Comparable
   class MonatsChart extends Chart
   class MonatsChartWert implements Comparable
   abstract class Tabelle extends KostenloseInfos
   class TagesTabelle extends Tabelle
   class Tagesdaten implements Comparable
   class Zeitraumdaten implements Comparable
   class Datum
   class Uhrzeit
   class Kurs
   abstract class ZeitraumTabelle extends Tabelle
   class Tagesgewinne extends LetzteTageskurse
   class LetzteTageskurse extends TagesTabelle
   class WochenTabelle extends ZeitraumTabelle
   class MonatsTabelle extends ZeitraumTabelle
   class MonatsDynamikTabelle extends MonatsTabelle
   interface Comparable
   abstract class ChartAnzeiger extends Canvas
   class TagesChartAnzeiger extends ChartAnzeiger
   class MonatsChartAnzeiger extends ChartAnzeiger
   class Auswahl extends Frame
   class Boerse
*/

interface Konstanten {
//  static Date date = new Date();
//  static Datum heute = new Datum(date.getDay(), date.getMonth(), date.getYear());
  static Datum heute = new Datum(27, 5, 97);
  static String kurse = "KURSE";
  static String kassakurse = "KASSAKURSE";
}


/* abstrakte Klasse BoersenInfo Implementiert Fenster, in denen
   Boersen-Informationen angezeigt werden.  Stellt sicher, dass in
```

```
   jeder der daraus abgeleiteten Klassen Knoepfe zum Schliessen des
   Fensters und zum Beenden des Programmes existieren.

   Methoden:
      BoersenInfo(String)        erzeugt das Fenster und setzt den Titel
      schliesseFenster()         kehrt zum aufrufenden Fenster zurueck
      beendeProgramm()           beendet das Programm komplett
      action(Event, Object)      verarbeitet Interaktionsereignisse
      abstract myaction(Event, Object)
                                 wird in "action" aufgerufen; Unterklassen
                                 koennen hier durch ueberschreiben
                                 spezielle Ereignisbehandlung
                                 implementieren
*/

abstract class BoersenInfo extends Frame implements Konstanten{
   Panel p1 = new Panel();
   Panel p2 = new Panel();
   Panel p3 = new Panel();
   Button schliessen = new Button("Schliessen");
   Button beenden = new Button("Beenden");
   float kosten;

   /* Konstruktor. Setzt den Titel des Fensters, fuegt drei Panels ein und bestimmt das Layout.
        Panel p1 dient zum Darstellen von Sponsor- oder Kosteninformationen,
        Panel p2 dient zum Anzeigen der angeforderten Informationen,
        Panel p3 enthaelt Knoepfe zum Schliessen des Fensters und zum Beenden des Programms.
   */
   BoersenInfo(String titel) {
      super(titel);
      this.setLayout(new BorderLayout());
      p1.setLayout(new FlowLayout());
      add("North",p1);
      p2.setLayout(new BorderLayout());
      add("Center",p2);
      p3.setLayout(new FlowLayout());
      add("South",p3);
      p3.add(schliessen);
      p3.add(beenden);
   }

   /* Schliesst das Fenster und gibt seine Ressourcen frei.
    */
   void schliesseFenster() {
      this.dispose();
   }

   /* Beendet das Programm komplett: alle Fenster werden geschlossen.
    */
   void beendeProgramm() {
      System.exit(0);
   }

   /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
      Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
      anderen Ereignisse wird "myaction" aufgerufen.
   */
   public boolean action(Event event, Object target) {
      if(event.id == Event.ACTION_EVENT) {
         if(event.target == schliessen) {
            schliesseFenster();
            return true;
         }
         else if(event.target == beenden) {
```

```
            beendeProgramm();
            return true;
         } else {
            return myaction(event, target);
         }
      } else {
         return false;
      }
   }

   /* Wird von "action" aufgerufen. Kann in Unterklassen ueberschrieben
      werden, um eine spezielle Ereignisbehandlung zu
      implementieren. (Diese Konstruktion verhindert, dass die Behandlung
      der Knoepfe "Schliessen" und "Beenden" jedesmal neu implementiert
      werden muss.
   */
   abstract boolean myaction(Event ev, Object target);
}


/* abstrakte Klasse KostenloseInfos
   Stellt sicher, dass fuer das Abrufen von Informationen durch alle
   aus ihr abgeleiteten Klassen keine Kosten anfallen. (Kosten werden
   von einem Sponsor getragen).
   Des weiteren stellt sie eine Methode zum Dateneinlesen
   bereit. Welche Daten jedoch daraus erzeugt werden ist zu diesem
   Zeitpunkt unklar und wird erst in einer Unterklasse entschieden
   durch die Methode erzeugeDatenvector(). Ausserdem wird nun schon
   ein Sponsorlabel fuer alle kostenlosen Infos gesetzt.

   Methoden:
   KostenloseInfos(String) Konstruktor, setzt Fenstertitel, fuegt einen Sponsor ein
                           und setzt die Kosten auf 0.00.
   leseDaten(String)   liest Daten aus einer Datei und schreibt sie mit Hilfe
                           der Methode erzeugeDatenvector() in den Vector vec
   qsort(int, int), partition(int, int), exchange(int, int), sortiere()
                           sind Methoden, die einen Sortieralgorithmus fuer alle
                           Unterklassen zur Verfuegung stellen. Da in jeder
                           Unterklasse unterschiedliche Datentypen zu sortieren
                           sind, wurde Schablonenmethode angewandt, die
                           vergleiche()-Methode ist nur abstrakt realisiert
                           und wird vom jeweiligen Datentyp erst implementiert.
*/

abstract class KostenloseInfos extends BoersenInfo {
   String dateiname;
   Label sponsor;
   Vector vec = new Vector();

   /* Konstruktor, setzt den Titel des Fensters, fuegt einen Sponsor in das
        Fenster ein (in p1) und setzt die Kosten auf 0.00
    */
   KostenloseInfos(String titel) {
      super(titel);
      sponsor = new Label("Diese Seiten werden gesponsort von xxy");
      p1.add(sponsor);
      kosten = 0.0f;
   }

   /* Liest die gewuenschten Daten aus einer Datei und erzeugt durch die
        abstrakte Methode erzeugeDatenvector eine Datenstruktur, in der die
        relevanten Daten gespeichert werden.
    */
   void leseDaten(String dateiname) {
```

```
    try{
      BufferedReader br = new BufferedReader(new FileReader(dateiname));
      String s;
      try{
        while((s = br.readLine())≠null) {
          if(!s.startsWith("#")) {
            erzeugeDatenvector(s);
          }
        }
      } catch (IOException e) {
        System.err.println("Lesefehler bei Datei " + dateiname);
      }
    } catch (FileNotFoundException e) {
      System.err.println("Die Datei " + dateiname + " konnte nicht geoeffnet werden");
    }
  }

  /* Erzeugt aus einem String ein Objekt in dem die relevanten Daten
     gespeichert sind. Dieses Objekt wird im Vektor vec gepeichert
   */
  abstract void erzeugeDatenvector(String s);

  /* Beschreibung des Quicksort siehe oben
   */
  void qsort(int left, int right) {
    int middle;
    if(left < right){
      middle = partition(left, right);
      qsort(left,middle-1);
      qsort(middle+1, right);
    }
  }

  /* Hilfsprozedur fuer den Quicksort
   */
  int partition(int left, int right) {
    Comparable pivot = (Comparable)vec.elementAt(left);
    int l = left;
    int r = right;
    int middle;
    while(l < r) {
      while((l ≤ right) && (((Comparable)vec.elementAt(l)).vergleiche(pivot))){
        l = l+1;
      }
      while((r ≥ left) && (!((Comparable)vec.elementAt(r)).vergleiche(pivot))){
        r = r-1;
      }
      if(l < r) {
        exchange(l, r);
      }
    }
    middle = r;
    exchange (left, middle);
    return middle;
  }

  /* Hilfsprozedur fuer den Quicksort
   */
  void exchange(int i, int j) {
    Comparable c = (Comparable)vec.elementAt(i);
    vec.setElementAt(vec.elementAt(j),i);
    vec.setElementAt(c,j);
  }
```

```
  /* Ruft Quicksort auf
   */
  void sortiere() {
    qsort(0, vec.size()-1);
  }
}


/* Klasse KostenpflichtigeInfos
     enthaelt einen Rahmen fuer Informationen, die vom Kunden bezahlt werden
     muessen, wie beispielsweise Prognosen, etc.
 */
/*Wird nicht ausprogrammiert.*/
abstract class KostenplichtigeInfos extends BoersenInfo {

  KostenplichtigeInfos(String titel) {
    super(titel);
    kosten = 1.0f;
  }
}


/* abstrakte Klasse Chart
     Diese Klasse enthaelt allgemeine Methoden zur Darstellung eines Charts.
     Dazu muss erst einmal ein Wertpapier ausgewaehlt werden, das angezeigt
     werden soll, und nach der Auswahl sollte der Chart angezeigt werden.

     Methoden:
     Chart(String)           Konstruktor, setzt den Fenstertitel
     void stelleDar()        stellt eine Dialogbox dar.
     myaction()              erweitert die action-Methode der Oberklasse.
     abstract zeigeCanvasAn()  liefert ein Canvas zurueck.
*/

abstract class Chart extends KostenloseInfos {
  int xChartSize = 200;
  int yChartSize = 200;
  String wkn;
  Label l = new Label();
  TextField t = new TextField("700045634",9);
  Button ok = new Button("ok");

  /* Konstruktor, ruft den Konstruktor von KostenloseInfos auf, und fuegt in den
       freien Rahmen p2 eine Auswahlmoeglichkeit des gewuenschten Wertpapiers ein
   */
  Chart(String titel) {
    super(titel);
  }

  void stelleDar(){
    p2.add("North",l);
    p2.add("Center",t);
    p2.add("South",ok);
    pack();
    show();
  }


  /* myaction erweitert die action-Methode, so dass nun beim Druck des
       Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
       aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
       und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
   */
  boolean myaction(Event ev, Object target){
```

```
    if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
      wkn = t.getText();
      wkn.trim();
      p2.removeAll();
      leseDaten(dateiname);
      p2.add("Center",zeigeCanvasAn());
      pack();
      show();
      return true;
    }
    return false;
  }

/*  liefert ein Canvas zurueck. Was das Canvas enthaelt, wird erst in der Unterklasse
    bestimmt.
 */
  abstract Canvas zeigeCanvasAn();
}


/* Klasse TagesChart
   In dieser Klasse wird das Erzeugen des Datenvektors implementiert.
   Diese entscheidet durch eine selektiere()-Methode, welche
   Datensaetze ueberhaupt ausgewaehlt werden sollen.

   Methoden:
     TagesChart(String)           Konstruktor, der den Titel setzt und die Datei
                                  mit den Daten angibt.
     erzeugeDatenvector(String)   bekommt eine Datensatz in Stringform und
                                  entscheidet mit Hilfe von selektiere(), ob der
                                  Datensatz in den Datenvektor eingefuegt
                                  werden soll.
     selektiere(String s)         entscheidet, ob der String ausgewaehlt werden
                                  soll, konkret: ob er von dem angegebenen
                                  Datum ist und die entsprechende
                                  Wertpapierkennnummer hat
     zeigeCanvasAn()                  erzeugt ein Objekt der Canvas-Klasse, in der der
                                  Chart aus den selektierten Daten, die im Datenvektor stehen,
                                  angezeigt wird.
*/

class TagesChart extends Chart {
  TagesChartWert tcw;

/* Konstruktor, der den Titel setzt und die Datei mit den Daten angibt
 */
  TagesChart(String titel) {
    super(titel);
    dateiname = kurse;
    l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Tageschart an");
  }

/* bekommt eine Datensatz in Stringform und entscheidet mit Hilfe von selektiere(),
   ob der Datensatz in den Datenvektor eingefuegt werden soll.
 */
  void erzeugeDatenvector(String s){
    if(selektiere(s) == true) {
      tcw = new TagesChartWert(s);
      vec.addElement(tcw);
    }
  }

/* entscheidet, ob der String ausgewaehlt werden soll, konkret: ob er von dem angegebenen
   Datum ist und die entsprechende Wertpapierkennnummer hat
```

```
  */
  boolean selektiere(String s) {
    Datum d = new Datum();
    d.setzeDatum(s.substring(0,8));
    if((heute.tag == d.tag) && (heute.monat ==d.monat) && (heute.jahr == d.jahr)) {
      return (wkn.compareTo(s.substring(15,24))==0);
    }
    return false;
  }

/* erzeugt ein Objekt der Canvas-Klasse, in der der Chart aus den selektierten Daten, die im
   Datenvektor stehen, angezeigt wird.
  */
  Canvas zeigeCanvasAn() {
    return new TagesChartAnzeiger(vec, xChartSize, yChartSize);
  }
}



/*Klasse TagesChartWert
  ist ein Objekttyp, der aus einer Uhrzeit und einem Kurswert besteht.
  Das sind genau die Daten, die man zum Anzeigen eines Tagescharts braucht. Die Klasse hat
  eine vergleiche-Methode, so dass man einen Vector mit Objekten dieses Typs mit
  der sortier()-Methode aus der Klasse KostenloseInfos sortieren kann.

  Methoden:
    TagesChartWert(String) bekommt einen String, in dem die Uhrzeit
                           und der Kurs enthalten sind.
    vergleiche(Comparable c) vergleicht die Uhrzeiten
*/

class TagesChartWert implements Comparable {
  Uhrzeit uhrzeit;
  Kurs kurs;

/* bekommt einen String, in dem die Uhrzeit und der Kurs enthalten sind.
  */
  TagesChartWert(String s) {
    uhrzeit = new Uhrzeit(s.substring(9,15));
    kurs = new Kurs(s.substring(s.length()-7, s.length()));
  }

/* vergleicht die Uhrzeiten zweier TagesChartWert-Objekte
  */
  public boolean vergleiche(Comparable c) {
    return ((this.uhrzeit).kleinergleich(((TagesChartWert)c).uhrzeit));
  }
}


/* Klasse MonatsChart
   In dieser Klasse wird das Erzeugen des Datenvectors implementiert.
   Diese entscheidet durch eine selektiere()-Methode, welche
   Datensaetze ueberhaupt ausgewaehlt werden sollen.

   Methoden:
     MonatsChart(String)           Konstruktor, der den Titel setzt und die Datei
                                   mit den Daten angibt.
     erzeugeDatenvector(String)    bekommt eine Datensatz in Stringform und
                                   entscheidet mit Hilfe von slektiere(), ob der
                                   Datensatz in den Datenvector eingefuegt
                                   werden soll.
     selektiere(String s)          entscheidet, ob der String ausgewaehlt werden
```

```
                                      soll, konkret: ob er von dem angegebenen
                                      Datum ist und die entsprechende
                                      Wertpapierkennnummer hat
     zeigeCanvasAn()                      erzeugt ein Objekt der Canvas-Klasse, in der der
                                      Chart angezeigt wird.
*/

class MonatsChart extends Chart {
  MonatsChartWert mcw;
  Datum letzterMonat = new Datum();

  MonatsChart(String titel) {
    super(titel);
    l.setText("Bitte geben Sie die Wertpapierkennnummer fuer den Monatschart an");
    dateiname = kassakurse;
    letzterMonat = heute.monatzurueck();
  }

  void erzeugeDatenvector(String s){
    if(selektiere(s) == true) {
      mcw = new MonatsChartWert(s);
      vec.addElement(mcw);
    }
  }

  boolean selektiere(String s) {
    Datum d = new Datum();
    d.setzeDatum(s.substring(0,8));
    if(!d.kleinergleich(letzterMonat)) {
      return (wkn.compareTo(s.substring(9,18))==0);
    }
    return false;
  }

  Canvas zeigeCanvasAn() {
    return new MonatsChartAnzeiger(vec, xChartSize, yChartSize);
  }
}


/*Klasse MonatsChartWert
  ist ein Objekttyp, der aus einem Datum und einem Kassakurswert besteht,
  Daten, die man zum Anzeigen eines Monatschart braucht. Die Klasse hat
  eine vergleiche-Methode, so dass man einen Vector mit diesem Typ mit
  der sortier()-Methode aus der Klasse KostenloseInfos sortieren kann.

  Methoden:
    MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
                                      und der Kurs enthalten sind.
    vergleiche(Comparable c)       vergleicht das Datum
*/

class MonatsChartWert implements Comparable {
  Kurs kassakurs;
  Datum datum;

  MonatsChartWert(String s){
    kassakurs = new Kurs(s.substring(s.length()-6, s.length()));
    datum = new Datum(s.substring(0,9));
  }

  public boolean vergleiche(Comparable c){
    return ((this.datum).kleinergleich(((MonatsChartWert)c).datum));
  }
```

```
}


/* abstrakte Klasse Tabelle.
   Gemeinsame Oberklasse fuer alle tabellarischen Darstellungen von
   Kursen oder Kassakursen.
*/

abstract class Tabelle extends KostenloseInfos {

  Tabelle(String titel) {
    super(titel);
    setzeDateiname();
  }

  /* Diese Methoden koennen nicht im Konstruktor aufgerufen werden, da
     zum Zeitpunkt der Ausfuehrung des Konstruktors manche Variablen
     noch nicht gesetzt wurden. Daher werden sie in eine eigene Methode
     "stelleDar" aufgerufen.
   */
  void stelleDar(){
    leseDaten(dateiname);
    zeigeTabelleAn();
  }

  abstract void setzeDateiname();
  abstract void zeigeTabelleAn();
}

/* Klasse zur tabellarischen Darstellung aller Kurse eines Tages
*/

class TagesTabelle extends Tabelle {
  TextArea textarea;

  TagesTabelle(String titel) {
    super(titel);
  }

  void setzeDateiname(String s) {
    dateiname = s;
  }

  void setzeDateiname() {
    dateiname = kurse;
  }

  boolean myaction(Event ev, Object target){
    return true;
  }

  boolean selektiere(String s) {
    Datum d = new Datum();
    d.setzeDatum(s.substring(0,8));
    return ((heute.tag == d.tag) && (heute.monat == d.monat) && (heute.jahr == d.jahr));
  }


  void erzeugeDatenvector(String s) {
    if(selektiere(s) == true) {
      Datum datum = new Datum(s.substring(0,8));
      Uhrzeit uhrzeit = new Uhrzeit(s.substring(9,14));
      Kurs kurs = new Kurs(s.substring(s.length() - 7, s.length()));
      String wkn = s.substring(14,24);
```

```
            String name = s.substring(25, s.length()-7);
            Tagesdaten tagesdaten = new Tagesdaten(datum, uhrzeit, wkn, name, kurs);
            vec.addElement(tagesdaten);
        }
    }

    void zeigeTabelleAn(){
        textarea = new TextArea(20,40);
        p2.add("Center",textarea);
        sortiere();
        for(int i = 0; i < vec.size(); i++) {
            textarea.append((((Tagesdaten)vec.elementAt(i)).uhrzeit).stunde + ":" +
                            (((Tagesdaten)vec.elementAt(i)).uhrzeit).minute + "  " +
                            ((Tagesdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                            ((Tagesdaten)vec.elementAt(i)).name + "  " +
                            ((Tagesdaten)vec.elementAt(i)).kurs.kurs + "\n");
        }
        if(vec.size() == 0) {
            textarea.append("Fuer das heutige Datum, " + heute.tag + "." +
                            heute.monat + "." + heute.jahr +
                            ", existieren keine Daten\n");
        }
        this.pack();
        this.show();
    }
}


/* Klasse zur Repraesentation eines einzelnen Kursinformations-Datensatzes
*/

class Tagesdaten implements Comparable {
    Datum datum;
    Uhrzeit uhrzeit;
    String wertpapierkennnummer;
    String name;
    Kurs kurs;

    Tagesdaten(Datum d, Uhrzeit u, String wkn, String n, Kurs k) {
        datum = d;
        uhrzeit = u;
        wertpapierkennnummer = wkn;
        name = n;
        kurs = k;
    }

    public boolean vergleiche(Comparable c) {
        return (this.uhrzeit).kleinergleich(((Tagesdaten)c).uhrzeit);
    }
}


/* Klasse zur Repraesentation eines einzelnen Kassakursinformations-Datensatzes
*/

class Zeitraumdaten implements Comparable {
    Datum datum;
    String wertpapierkennnummer;
    String name;
    Kurs kassakurs;

    Zeitraumdaten(Datum d, String wkn, String n, Kurs k) {
        datum = d;
        wertpapierkennnummer = wkn;
```

```
        name = n;
        kassakurs = k;
    }

    public boolean vergleiche(Comparable c) {
        return (this.datum).kleinergleich(((Zeitraumdaten)c).datum);
    }
}


/* Klasse zur Repraesentation eines Datums, incl. Berechnung des
   Datums 'vor einer Woche' und 'vor einem Monat'.
   (Anmerkung zum Experiment:
     Diese Routinen sind vereinfacht, um den Code zu verkuerzen.
     Sie funktionieren nicht wirklich korrekt. Das soll uns hier aber mal
     gerade nicht stoeren...)
*/

class Datum {
    int tag;
    int monat;
    int jahr;

    Datum() {
        tag = 0;
        monat = 0;
        jahr = 0;
    }

    Datum(String s){
        tag = new Integer(s.substring(0,2)).intValue();
        monat = new Integer(s.substring(3,5)).intValue();
        jahr = new Integer(s.substring(6,8)).intValue();
    }

    Datum(int t, int m, int j){
        tag = t;
        monat = m;
        jahr = j;
    }

    void setzeDatum(String s) {
        tag = new Integer(s.substring(0,2)).intValue();
        monat = new Integer(s.substring(3,5)).intValue();
        jahr = new Integer(s.substring(6,8)).intValue();
    }

    void setzeDatum(int t, int m, int j) {
        tag = t;
        monat = m;
        jahr = j;
    }

    boolean kleinergleich(Datum d) {
        if(this.jahr < d.jahr) {
            return true;
        } else if(this.jahr == d.jahr){
            if(this.monat < d.monat) {
                return true;
            } else if(this.monat == d.monat){
                if(this.tag <= d.tag) {
                    return true;
                }
            }
```

```
        }
        return false;
    }

    Datum monatzurueck() {
        Datum d = new Datum();
        d.tag = tag;
        d.jahr = jahr;
        if(monat == 1) {
            d.monat = 12;
            d.jahr = jahr - 1;
        } else {
            d.monat = monat - 1;
        }
        return d;
    }

    /* vereinfachte Berechnung: Alle Monate haben 30 Tage
     */
    Datum wochezurueck() {
        Datum d = new Datum();
        d.tag = tag - 7;
        d.monat = monat;
        d.jahr = jahr;
        if(d.tag < 1) {
            d.tag = d.tag + 30;
            d.monat = monat - 1;
            if(d.monat < 1) {
                d.monat = d.monat + 12;
                d.jahr = jahr - 1;
            }
        }
        return d;
    }

    void print() {
        System.out.println(" Datum = " + tag + "." + monat + "." + jahr);
    }
}

/* Klasse zur Repraesentation einer Uhrzeit
*/

class Uhrzeit {
    int stunde;
    int minute;

    Uhrzeit(String s) {
        stunde = (new Integer(s.substring(0,2))).intValue();
        minute = (new Integer(s.substring(3,5))).intValue();
    }

    boolean kleinergleich(Uhrzeit u) {
        if(this.stunde < u.stunde) {
            return true;
        } else if(this.stunde == u.stunde) {
            if(this.minute ≤ u.minute) {
                return true;
            }
        }
        return false;
    }
}
```

```
/* Klasse zur Repraesentation eines Kurses oder Kassakurses
*/

class Kurs {
   float kurs;

   Kurs(String s){
      kurs = (new Float(s)).floatValue();
   }
}


/* abstrakte Klasse ZeitraumTabelle.
    Gemeinsame Oberklasse fuer alle tabellarischen Darstellungen, die
    ueber mehrere Tage gehen. Benutzt die Kassakurse.
*/

abstract class ZeitraumTabelle extends Tabelle {
   Datum vonDatum;
   TextArea textarea;

   ZeitraumTabelle(String titel) {
      super(titel);
   }

   void setzeDateiname(String s) {
      dateiname = s;
   }

   void setzeDateiname() {
      dateiname = kassakurse;
   }

   boolean selektiere(String s) {
      Datum d = new Datum();
      d.setzeDatum(s.substring(0,8));
      return (!d.kleinergleich(vonDatum));
   }

   void erzeugeDatenvector(String s){
      if(selektiere(s) == true) {
         Datum datum = new Datum(s.substring(0,8));
         String wkn = s.substring(9,18);
         String name = s.substring(19, s.length()-7);
         Kurs kassakurs = new Kurs(s.substring(s.length()-6,s.length()));
         Zeitraumdaten zrd = new Zeitraumdaten(datum, wkn, name, kassakurs);
         vec.addElement(zrd);
      }
   }

   void zeigeTabelleAn(){
      textarea = new TextArea(20,40);
      p2.add("Center",textarea);
      sortiere();
      for(int i = 0; i < vec.size(); i++) {
         textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + "." +
                         (((Zeitraumdaten)vec.elementAt(i)).datum).monat + "." +
                         (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + "  " +
                         ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                         ((Zeitraumdaten)vec.elementAt(i)).name + "  " +
                         ((Zeitraumdaten)vec.elementAt(i)).kassakurs.kurs + "\n");
      }
      if(vec.size() == 0) {
         textarea.append("Fuer der gewuenschten Zeitraum existieren keine Daten\n");
```

```
    }
    this.pack();
    this.show();
  }
}


/* Tabellarische Darstellung der prozentualen Unterschiede zwischen dem
   ersten und letzten Kurs jeder Aktie am heutigen Datum.
*/

class Tagesgewinne extends LetzteTageskurse {

  Tagesgewinne(String titel) {
    super(titel);
  }

  float tagesgewinn(String wertpapierkennnummer, int start) {
    float min = 0.0f;
    float max = 0.0f;
    int i = start;
    while((wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) != 0) && (i < vec.size())) {
      i = i+1;
    }
    min = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
    while(i < vec.size()) {
      if(wertpapierkennnummer.compareTo(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
        max = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
      }
      i = i+1;
    }
    //    System.out.println("Min = " + min + "     Max = " + max);
    if(min == 0.0f) {
      return 0;
    } else {
      return ((max - min)/min);
    }
  }

  void zeigeTabelleAn() {
    textarea = new TextArea(20,40);
    p2.add("Center",textarea);
    boolean keineDaten = true;
    sortiere();
    for(int i = 0; i < vec.size(); i++) {
      if(! schonAbgearbeitet(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
        textarea.append(((Tagesdaten)(vec.elementAt(i))).name + "  " +
                        ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + "  "
                        + tagesgewinn(((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
                        "\n");
        keineDaten = false;
      }
    }
    if(keineDaten == true) {
      textarea.append("Fuer das heutige Datum, " + heute.tag + "." +
                      heute.monat + "." + heute.jahr +
                      ", existieren keine Daten\n");
    }
    this.pack();
    this.show();
  }
}

```

```
/* tabellarische Darstellung der letzten vorhandenen heutigen Kurse fuer alle
   Aktientitel.
*/

class LetzteTageskurse extends TagesTabelle {
  Vector v = new Vector();

  LetzteTageskurse(String titel) {
    super(titel);
  }

  boolean schonAbgearbeitet(String s) {
    int i = 0;
    while(i < v.size()) {
      if(((String)v.elementAt(i)).compareTo(s)==0) {
        i = v.size();
        return true;
      } else {
        i++;
      }
    }
    v.addElement(s);
    return false;
  }

  float letztetageskurse(String wertpapierkennnummer, int start) {
    float letzter = 0.0f;
    int i = start;
    while(i < vec.size()) {
      if(wertpapierkennnummer.compareTo(
            ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer) == 0) {
        letzter = ((Tagesdaten)(vec.elementAt(i))).kurs.kurs;
      }
      i = i+1;
    }
    return letzter;
  }

  void zeigeTabelleAn() {
    textarea = new TextArea(20,40);
    p2.add("Center",textarea);
    sortiere();
    for(int i = 0; i < vec.size(); i++) {
      if(!schonAbgearbeitet(
              ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer)) {
        textarea.append(((Tagesdaten)(vec.elementAt(i))).name + "  " +
                        ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer + "  " +
                        letztetageskurse(
                        ((Tagesdaten)(vec.elementAt(i))).wertpapierkennnummer, i) +
                        "\n");
      }
    }
    if(vec.size() == 0) {
      textarea.append("Fuer das heutige Datum, " +
                      heute.tag + "." + heute.monat + "." +
                      heute.jahr + ", existieren keine Daten\n");
    }
    this.pack();
    this.show();
  }
}


/* tabellarische Darstellung der Kassakurse aller Aktientitel fuer die
```

```
    letzten 7 Tage.
*/

class WochenTabelle extends ZeitraumTabelle {

  WochenTabelle(String titel) {
    super(titel);
    vonDatum = heute.wochezurueck();
  }

  boolean myaction(Event ev, Object target) {
    return true;
  }
}


/* tabellarische Darstellung der Kassakurse aller Aktientitel fuer die
    letzten 30 Tage (genauer gesagt: seit dem gleichen Tag im Vormonat).
*/

class MonatsTabelle extends ZeitraumTabelle {

  MonatsTabelle(String titel) {
    super(titel);
    vonDatum = heute.monatzurueck();
  }

  boolean myaction(Event ev, Object target) {
    return true;
  }
}


/* tabellarische Darstellung der prozentualen Unterschiede zwischen
    aufeinanderfolgenden Kassakursen aller Aktientitel fuer den
    letzten Monat.
*/

class MonatsDynamikTabelle extends MonatsTabelle {
  Vector dyn = new Vector();

  MonatsDynamikTabelle(String titel) {
    super(titel);
  }

  float berechneDynamik(int i) {
    //suche den letzen Wert des gleichen Wertpapiers, um die prozentuale
    //Abweichung des neuen Wertes vom alten zu bestimmen.
    int j = i-1;
    boolean gefunden = false;
    while ((j ≥ 0) && !gefunden) {
      if ((((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer).compareTo(
              ((Zeitraumdaten)vec.elementAt(j)).wertpapierkennnummer) == 0) {
        gefunden = true;
        return ((((((Zeitraumdaten)vec.elementAt(i)).kassakurs).kurs -
                    (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)/
                    (((Zeitraumdaten)vec.elementAt(j)).kassakurs).kurs)*100);
      }
      j--;
    }
    return 0.0f;
  }

  void zeigeTabelleAn(){
```

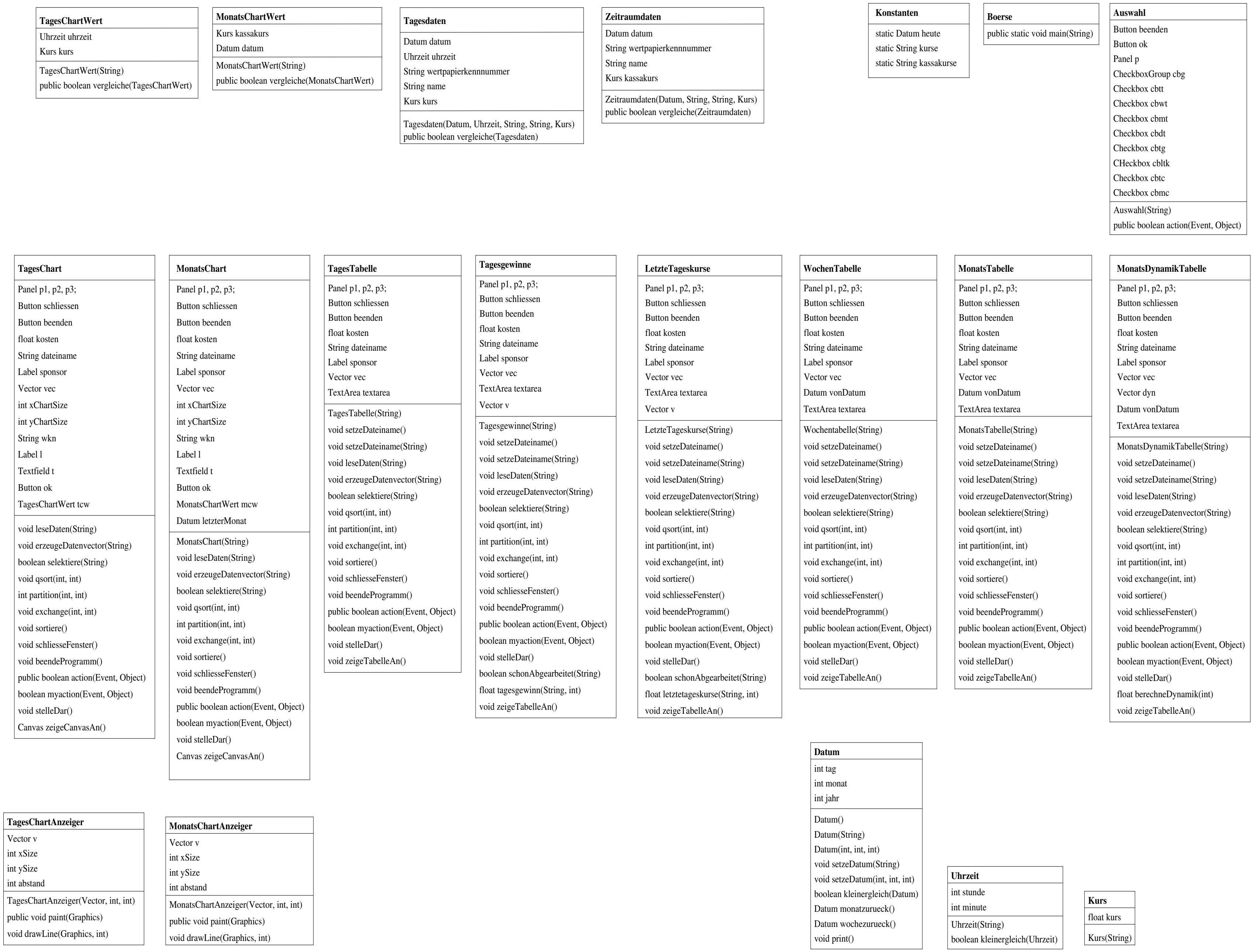
```
    textarea = new TextArea(20,40);
    p2.add("Center",textarea);
    sortiere();
    for(int i = 0; i < vec.size(); i++) {
      textarea.append((((Zeitraumdaten)vec.elementAt(i)).datum).tag + "." +
                      (((Zeitraumdaten)vec.elementAt(i)).datum).monat + "." +
                      (((Zeitraumdaten)vec.elementAt(i)).datum).jahr + "  " +
                      ((Zeitraumdaten)vec.elementAt(i)).wertpapierkennnummer + "  " +
                      ((Zeitraumdaten)vec.elementAt(i)).name + "  " +
                      berechneDynamik(i) + "%\n");
    }
    if(vec.size() == 0) {
      textarea.append("Fuer diesen Zeitraum existieren keine Daten\n");
    }
    this.pack();
    this.show();
  }
}


/* Schnittstelle fuer 'vergleichbare' Datentypen (zum Sortieren).
   a.vergleiche(b) liefert 'a kleiner-oder-gleich b'
*/

interface Comparable {
  public boolean vergleiche(Comparable c);
}

/* Oberklasse aller graphischen Anzeigeklassen
*/

abstract class ChartAnzeiger extends Canvas {
  Vector v;
  int xSize;
  int ySize;
  int abstand;

  ChartAnzeiger(Vector vec, int xSize, int ySize) {
    v = vec;
    this.xSize = xSize;
    this.ySize = ySize;
    if(vec.size() ≠ 0) {
      abstand = xSize/vec.size();
    }
    setSize(xSize, ySize);
  }

  public void paint(Graphics g) {
    g.drawRect(0, 0, xSize-1, ySize-1);
    for(int i = 0; i < v.size()-1; i++) {
      drawLine(g, i);
    }
  }

  abstract void drawLine(Graphics g, int i);
}


/* graphische Darstellung aller Kurse eines Aktientitels fuer den heutigen
   Tag.
*/

class TagesChartAnzeiger extends ChartAnzeiger {

```

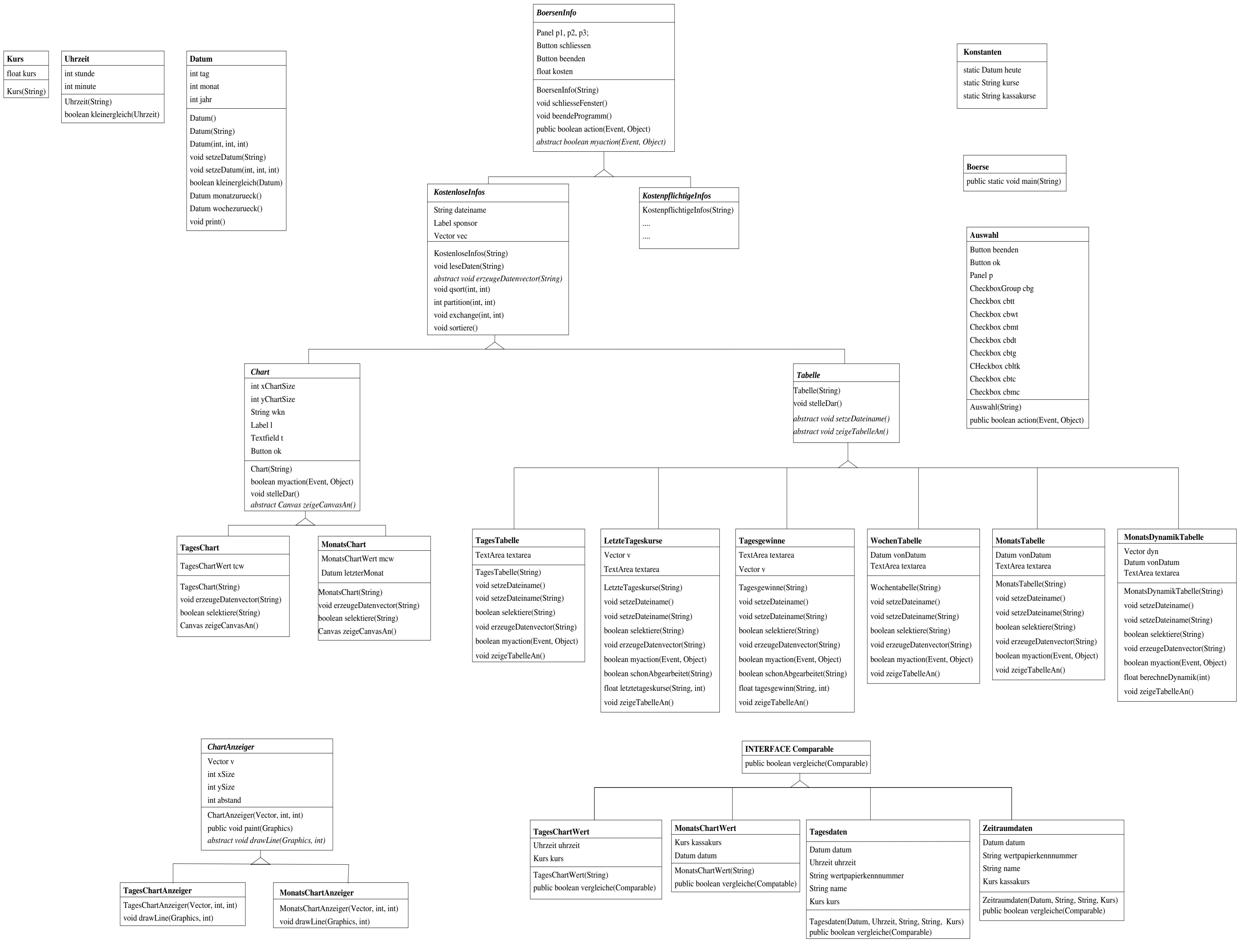
```
    TagesChartAnzeiger(Vector vec, int xSize, int ySize) {
        super(vec, xSize, ySize);
    }

    void drawLine(Graphics g, int i){
        g.drawLine(i*abstand,
                   ySize - (int)((((TagesChartWert)(v.elementAt(i))).kurs).kurs),
                   (i+1)*abstand,
                   ySize - (int)((((TagesChartWert)(v.elementAt(i+1))).kurs).kurs));
    }
}


/* graphische Darstellung aller Kassakurse eines Aktientitels fuer die
   letzten 30 Tage.
*/

class MonatsChartAnzeiger extends ChartAnzeiger {

    MonatsChartAnzeiger(Vector vec, int xSize, int ySize) {
        super(vec, xSize, ySize);
    }

    void drawLine(Graphics g, int i){
        g.drawLine(i*abstand,
                   ySize - (int)((((MonatsChartWert)(v.elementAt(i))).kassakurs).kurs),
                   (i+1)*abstand,
                   ySize - (int)((((MonatsChartWert)(v.elementAt(i+1))).kassakurs).kurs));
    }
}

/* Kern des Hauptprogramms: Auswahlmenue der einzelnen Darstellungsformen.
*/

class Auswahl extends Frame {
    Button beenden = new Button("Beenden");
    Button ok = new Button("OK");
    Panel p = new Panel();
    CheckboxGroup cbg = new CheckboxGroup();
    Checkbox cbtt = new Checkbox("Tagestabelle", cbg, true);
    Checkbox cbwt = new Checkbox("Wochentabelle", cbg, false);
    Checkbox cbmt = new Checkbox("Monatstabelle", cbg, false);
    Checkbox cbdt = new Checkbox("Dynamiktabelle", cbg, false);
    Checkbox cbtg = new Checkbox("Tagesgewinne", cbg, false);
    Checkbox cbltk = new Checkbox("letzte Tageskurse", cbg, false);
    Checkbox cbtc = new Checkbox("Tages-Chart", cbg, false);
    Checkbox cbmc = new Checkbox("Monats-Chart", cbg, false);

    Auswahl(String titel) {
        super(titel);
        setLayout(new BorderLayout());
        p.setLayout(new GridLayout(6,1));
        p.add(cbtt);
        p.add(cbwt);
        p.add(cbmt);
        p.add(cbdt);
        p.add(cbtg);
        p.add(cbltk);
        p.add(cbtc);
        p.add(cbmc);
        add("Center",p);
        add("East",ok);
        add("South",beenden);
        pack();
```

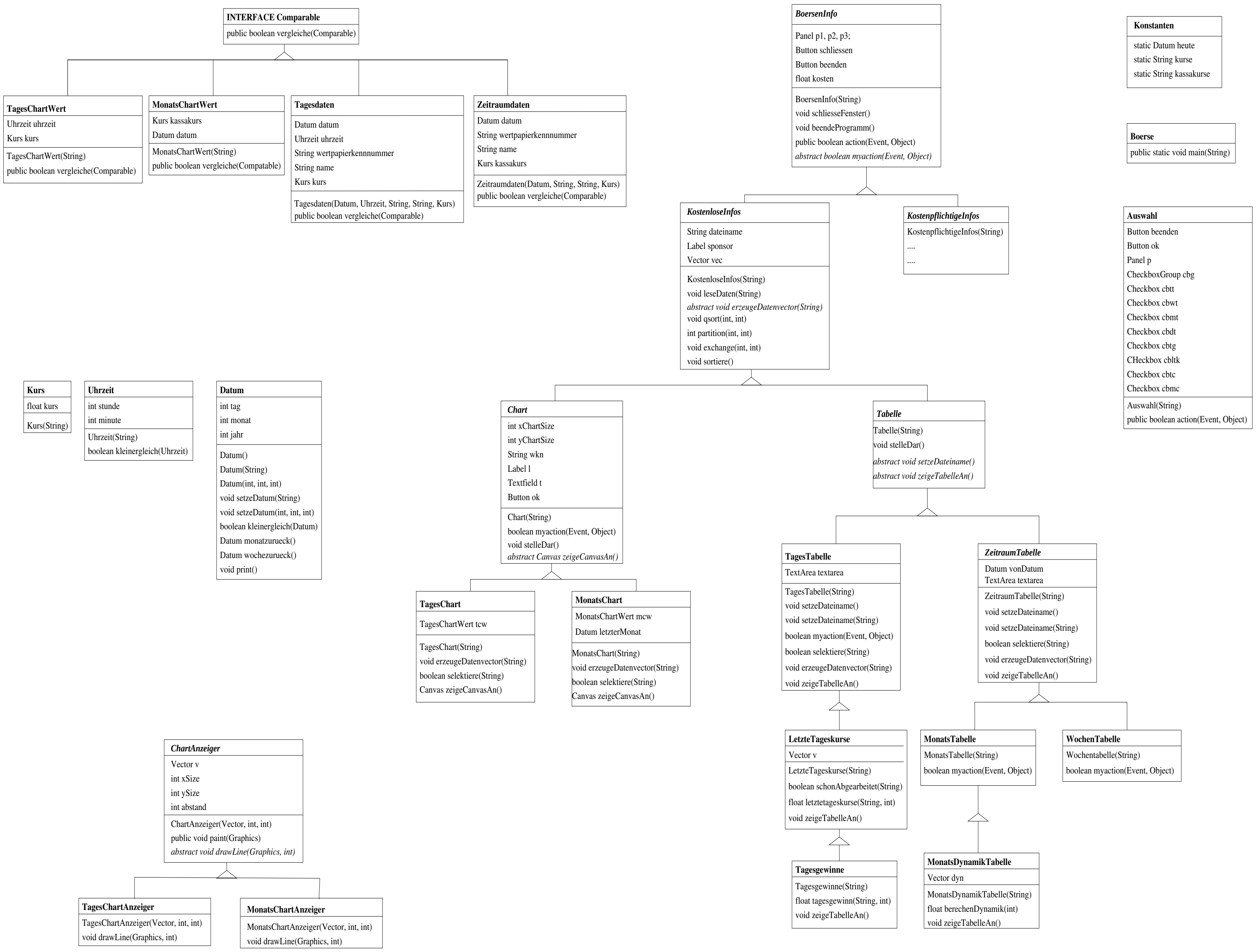
```
    show();
  }

  public boolean action(Event ev, Object target){
    if(ev.id == Event.ACTION_EVENT) {
      if(ev.target == ok) {
        if(cbg.getSelectedCheckbox() == cbtt) {
          TagesTabelle tt = new TagesTabelle("Tagestabelle");
          tt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbwt) {
          WochenTabelle wt = new WochenTabelle("Wochentabelle");
          wt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbmt) {
          MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
          mt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbdt) {
          MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
          dt.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbtg) {
          Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
          tg.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbltk) {
          LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
          ltg.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbtc) {
          TagesChart tc = new TagesChart("Tageschart");
          tc.stelleDar();
          return true;
        } else if(cbg.getSelectedCheckbox() == cbmc) {
          MonatsChart mc = new MonatsChart("Monatschart");
          mc.stelleDar();
          return true;
        }
      } else if(ev.target == beenden) {
        System.exit(0);
        return true;
      }
    }
    return false;
  }
}


/* Programmeinstiegspunkt
*/

class Boerse {
  public static void main(String args[]) {
    Auswahl a = new Auswahl("Boersen-Information");
  }
}
```

B.2 OMT diagrams – JAKK

On the following three pages you find the depictions of the inheritance hierarchy that were given to the subjects working the 0-level, 3-level, or 5-level program version, respectively.


TagesChartWert
Uhrzeit uhrzeit
Kurs kurs
TagesChartWert(String)
public boolean vergleiche(TagesChartWert)
MonatsChartWert
Kurs kassakurs
Datum datum
MonatsChartWert(String)
public boolean vergleiche(MonatsChartWert)
Tagesdaten
Datum datum
Uhrzeit uhrzeit
String wertpapierkennnummer
String name
Kurs kurs
Tagesdaten(Datum, Uhrzeit, String, String, Kurs)
public boolean vergleiche(Tagesdaten)
Zeitraumdaten
Datum datum
String wertpapierkennnummer
String name
Kurs kassakurs
Zeitraumdaten(Datum, String, String, Kurs)
public boolean vergleiche(Zeitraumdaten)
Konstanten
static Datum heute
static String kurse
static String kassakurse
Boerse
public static void main(String)
Auswahl
Button beenden
Button ok
Panel p
CheckboxGroup cbg
Checkbox cbtt
Checkbox cbwt
Checkbox cbmt
Checkbox cbdt
Checkbox cbtg
CHeckbox cbltk
Checkbox cbtc
Checkbox cbmc
Auswahl(String)
public boolean action(Event, Object)
TagesChart
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
String dateiname
Label sponsor
Vector vec
int xChartSize
int yChartSize
String wkn
Label l
Textfield t
Button ok
TagesChartWert tcw
void leseDaten(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
boolean myaction(Event, Object)
void stelleDar()
Canvas zeigeCanvasAn()
MonatsChart
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
String dateiname
Label sponsor
Vector vec
int xChartSize
int yChartSize
String wkn
Label l
Textfield t
Button ok
MonatsChartWert mcw
Datum letzterMonat
MonatsChart(String)
void leseDaten(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
boolean myaction(Event, Object)
void stelleDar()
Canvas zeigeCanvasAn()
TagesTabelle
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
String dateiname
Label sponsor
Vector vec
TextArea textarea
TagesTabelle(String)
void setzeDateiname()
void setzeDateiname(String)
void leseDaten(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
boolean myaction(Event, Object)
void stelleDar()
void zeigeTabelleAn()
Tagesgewinne
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
String dateiname
Label sponsor
Vector vec
TextArea textarea
Vector v
Tagesgewinne(String)
void setzeDateiname()
void setzeDateiname(String)
void leseDaten(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
boolean myaction(Event, Object)
void stelleDar()
boolean schonAbgearbeitet(String)
float tagesgewinn(String, int)
void zeigeTabelleAn()
LetzteTageskurse
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
String dateiname
Label sponsor
Vector vec
TextArea textarea
Vector v
LetzteTageskurse(String)
void setzeDateiname()
void setzeDateiname(String)
void leseDaten(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
boolean myaction(Event, Object)
void stelleDar()
boolean schonAbgearbeitet(String)
float letztetageskurse(String, int)
void zeigeTabelleAn()
WochenTabelle
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
String dateiname
Label sponsor
Vector vec
Datum vonDatum
TextArea textarea
Wochentabelle(String)
void setzeDateiname()
void setzeDateiname(String)
void leseDaten(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
boolean myaction(Event, Object)
void stelleDar()
void zeigeTabelleAn()
MonatsTabelle
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
String dateiname
Label sponsor
Vector vec
Datum vonDatum
TextArea textarea
MonatsTabelle(String)
void setzeDateiname()
void setzeDateiname(String)
void leseDaten(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
boolean myaction(Event, Object)
void stelleDar()
void zeigeTabelleAn()
MonatsDynamikTabelle
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
String dateiname
Label sponsor
Vector vec
Vector dyn
Datum vonDatum
TextArea textarea
MonatsDynamikTabelle(String)
void setzeDateiname()
void setzeDateiname(String)
void leseDaten(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
boolean myaction(Event, Object)
void stelleDar()
float berechneDynamik(int)
void zeigeTabelleAn()
Datum
int tag
int monat
int jahr
Datum()
Datum(String)
Datum(int, int, int)
void setzeDatum(String)
void setzeDatum(int, int, int)
boolean kleinergleich(Datum)
Datum monatzurueck()
Datum wochezurueck()
void print()
TagesChartAnzeiger
Vector v
int xSize
int ySize
int abstand
TagesChartAnzeiger(Vector, int, int)
public void paint(Graphics)
void drawLine(Graphics, int)
MonatsChartAnzeiger
Vector v
int xSize
int ySize
int abstand
MonatsChartAnzeiger(Vector, int, int)
public void paint(Graphics)
void drawLine(Graphics, int)
Uhrzeit
int stunde
int minute
Uhrzeit(String)
boolean kleinergleich(Uhrzeit)
Kurs
float kurs
Kurs(String)


Kurs
float kurs
Kurs(String)
Uhrzeit
int stunde
int minute
Uhrzeit(String)
boolean kleinergleich(Uhrzeit)
Datum
int tag
int monat
int jahr
Datum()
Datum(String)
Datum(int, int, int)
void setzeDatum(String)
void setzeDatum(int, int, int)
boolean kleinergleich(Datum)
Datum monatzurueck()
Datum wochezurueck()
void print()
BoersenInfo
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
BoersenInfo(String)
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
abstract boolean myaction(Event, Object)
Konstanten
static Datum heute
static String kurse
static String kassakurse
Boerse
public static void main(String)
KostenloseInfos
String dateiname
Label sponsor
Vector vec
KostenloseInfos(String)
void leseDaten(String)
abstract void erzeugeDatenvector(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
KostenpflichtigeInfos
KostenpflichtigeInfos(String)
....
....
Auswahl
Button beenden
Button ok
Panel p
CheckboxGroup cbg
Checkbox cbtt
Checkbox cbwt
Checkbox cbmt
Checkbox cbdt
Checkbox cbtg
CHeckbox cbltk
Checkbox cbtc
Checkbox cbmc
Auswahl(String)
public boolean action(Event, Object)
Chart
int xChartSize
int yChartSize
String wkn
Label l
Textfield t
Button ok
Chart(String)
boolean myaction(Event, Object)
void stelleDar()
abstract Canvas zeigeCanvasAn()
Tabelle
Tabelle(String)
void stelleDar()
abstract void setzeDateiname()
abstract void zeigeTabelleAn()
TagesChart
TagesChartWert tcw
TagesChart(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
Canvas zeigeCanvasAn()
MonatsChart
MonatsChartWert mcw
Datum letzterMonat
MonatsChart(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
Canvas zeigeCanvasAn()
TagesTabelle
TextArea textarea
TagesTabelle(String)
void setzeDateiname()
void setzeDateiname(String)
boolean selektiere(String)
void erzeugeDatenvector(String)
boolean myaction(Event, Object)
void zeigeTabelleAn()
LetzteTageskurse
Vector v
TextArea textarea
LetzteTageskurse(String)
void setzeDateiname()
void setzeDateiname(String)
boolean selektiere(String)
void erzeugeDatenvector(String)
boolean myaction(Event, Object)
boolean schonAbgearbeitet(String)
float letztetageskurse(String, int)
void zeigeTabelleAn()
Tagesgewinne
TextArea textarea
Vector v
Tagesgewinne(String)
void setzeDateiname()
void setzeDateiname(String)
boolean selektiere(String)
void erzeugeDatenvector(String)
boolean myaction(Event, Object)
boolean schonAbgearbeitet(String)
float tagesgewinn(String, int)
void zeigeTabelleAn()
WochenTabelle
Datum vonDatum
TextArea textarea
Wochentabelle(String)
void setzeDateiname()
void setzeDateiname(String)
boolean selektiere(String)
void erzeugeDatenvector(String)
boolean myaction(Event, Object)
void zeigeTabelleAn()
MonatsTabelle
Datum vonDatum
TextArea textarea
MonatsTabelle(String)
void setzeDateiname()
void setzeDateiname(String)
boolean selektiere(String)
void erzeugeDatenvector(String)
boolean myaction(Event, Object)
void zeigeTabelleAn()
MonatsDynamikTabelle
Vector dyn
Datum vonDatum
TextArea textarea
MonatsDynamikTabelle(String)
void setzeDateiname()
void setzeDateiname(String)
boolean selektiere(String)
void erzeugeDatenvector(String)
boolean myaction(Event, Object)
float berechneDynamik(int)
void zeigeTabelleAn()
ChartAnzeiger
Vector v
int xSize
int ySize
int abstand
ChartAnzeiger(Vector, int, int)
public void paint(Graphics)
abstract void drawLine(Graphics, int)
TagesChartAnzeiger
TagesChartAnzeiger(Vector, int, int)
void drawLine(Graphics, int)
MonatsChartAnzeiger
MonatsChartAnzeiger(Vector, int, int)
void drawLine(Graphics, int)
INTERFACE Comparable
public boolean vergleiche(Comparable)
TagesChartWert
Uhrzeit uhrzeit
Kurs kurs
TagesChartWert(String)
public boolean vergleiche(Comparable)
MonatsChartWert
Kurs kassakurs
Datum datum
MonatsChartWert(String)
public boolean vergleiche(Compatable)
Tagesdaten
Datum datum
Uhrzeit uhrzeit
String wertpapierkennnummer
String name
Kurs kurs
Tagesdaten(Datum, Uhrzeit, String, String, Kurs)
public boolean vergleiche(Comparable)
Zeitraumdaten
Datum datum
String wertpapierkennnummer
String name
Kurs kassakurs
Zeitraumdaten(Datum, String, String, Kurs)
public boolean vergleiche(Comparable)


INTERFACE Comparable
public boolean vergleiche(Comparable)
TagesChartWert
Uhrzeit uhrzeit
Kurs kurs
TagesChartWert(String)
public boolean vergleiche(Comparable)
MonatsChartWert
Kurs kassakurs
Datum datum
MonatsChartWert(String)
public boolean vergleiche(Compatable)
Tagesdaten
Datum datum
Uhrzeit uhrzeit
String wertpapierkennnummer
String name
Kurs kurs
Tagesdaten(Datum, Uhrzeit, String, String, Kurs)
public boolean vergleiche(Comparable)
Zeitraumdaten
Datum datum
String wertpapierkennnummer
String name
Kurs kassakurs
Zeitraumdaten(Datum, String, String, Kurs)
public boolean vergleiche(Comparable)
BoersenInfo
Panel p1, p2, p3;
Button schliessen
Button beenden
float kosten
BoersenInfo(String)
void schliesseFenster()
void beendeProgramm()
public boolean action(Event, Object)
abstract boolean myaction(Event, Object)
Konstanten
static Datum heute
static String kurse
static String kassakurse
Boerse
public static void main(String)
KostenloseInfos
String dateiname
Label sponsor
Vector vec
KostenloseInfos(String)
void leseDaten(String)
abstract void erzeugeDatenvector(String)
void qsort(int, int)
int partition(int, int)
void exchange(int, int)
void sortiere()
KostenpflichtigeInfos
KostenpflichtigeInfos(String)
....
....
Auswahl
Button beenden
Button ok
Panel p
CheckboxGroup cbg
Checkbox cbtt
Checkbox cbwt
Checkbox cbmt
Checkbox cbdt
Checkbox cbtg
CHeckbox cbltk
Checkbox cbtc
Checkbox cbmc
Auswahl(String)
public boolean action(Event, Object)
Kurs
float kurs
Kurs(String)
Uhrzeit
int stunde
int minute
Uhrzeit(String)
boolean kleinergleich(Uhrzeit)
Datum
int tag
int monat
int jahr
Datum()
Datum(String)
Datum(int, int, int)
void setzeDatum(String)
void setzeDatum(int, int, int)
boolean kleinergleich(Datum)
Datum monatzurueck()
Datum wochezurueck()
void print()
Chart
int xChartSize
int yChartSize
String wkn
Label l
Textfield t
Button ok
Chart(String)
boolean myaction(Event, Object)
void stelleDar()
abstract Canvas zeigeCanvasAn()
Tabelle
Tabelle(String)
void stelleDar()
abstract void setzeDateiname()
abstract void zeigeTabelleAn()
TagesChart
TagesChartWert tcw
TagesChart(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
Canvas zeigeCanvasAn()
MonatsChart
MonatsChartWert mcw
Datum letzterMonat
MonatsChart(String)
void erzeugeDatenvector(String)
boolean selektiere(String)
Canvas zeigeCanvasAn()
TagesTabelle
TextArea textarea
TagesTabelle(String)
void setzeDateiname()
void setzeDateiname(String)
boolean myaction(Event, Object)
boolean selektiere(String)
void erzeugeDatenvector(String)
void zeigeTabelleAn()
ZeitraumTabelle
Datum vonDatum
TextArea textarea
ZeitraumTabelle(String)
void setzeDateiname()
void setzeDateiname(String)
boolean selektiere(String)
void erzeugeDatenvector(String)
void zeigeTabelleAn()
ChartAnzeiger
Vector v
int xSize
int ySize
int abstand
ChartAnzeiger(Vector, int, int)
public void paint(Graphics)
abstract void drawLine(Graphics, int)
LetzteTageskurse
Vector v
LetzteTageskurse(String)
boolean schonAbgearbeitet(String)
float letztetageskurse(String, int)
void zeigeTabelleAn()
MonatsTabelle
MonatsTabelle(String)
boolean myaction(Event, Object)
WochenTabelle
Wochentabelle(String)
boolean myaction(Event, Object)
Tagesgewinne
Tagesgewinne(String)
float tagesgewinn(String, int)
void zeigeTabelleAn()
MonatsDynamikTabelle
Vector dyn
MonatsDynamikTabelle(String)
float berechenDynamik(int)
void zeigeTabelleAn()
TagesChartAnzeiger
TagesChartAnzeiger(Vector, int, int)
void drawLine(Graphics, int)
MonatsChartAnzeiger
MonatsChartAnzeiger(Vector, int, int)
void drawLine(Graphics, int)

B.3 Program differences between JAKK and Informatik II experiments

The Informatik II versions of the Boerse.java program are given here in the form of deltas to the corresponding JAKK versions. These deltas were produced by the Unix utility *diff*.

As one can see, the differences are rather uniform and are restricted to the action() and myaction() methods only.

B.3.1 Boerse.java (no inheritance)

```
7a8
> import java.awt.event.*;
79,80d79
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction() erweitert die action-Methode.
124a124,136
> schliessen.addActionListener(
>     //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>     //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>     //(in diesem Fall der schliessen-Knopf) spezialisierte
>     //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>     //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>     //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
125a138,150
> beenden.addActionListener(
>     //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>     //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>     //(in diesem Fall der beenden-Knopf) spezialisierte
>     //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>     //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>     //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
238,277d262
<     /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<        Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<        anderen Ereignisse wird "myaction" aufgerufen.
<     */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
```

```
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
<                        /∗ myaction erweitert die action-Methode, so dass nun beim Druck des
<    Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
<    aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
<    und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
<  ∗/
< boolean myaction(Event ev, Object target){
< if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
< wkn = t.getText();
< wkn.trim();
< p2.removeAll();
< leseDaten(dateiname);
< p2.add("Center",zeigeCanvasAn());
< pack();
< show();
< return true;
< }
< return false;
< }
<
281a267,285
> ok.addActionListener(
>                        //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                        //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                        //(in diesem Fall der ok-Knopf) spezialisierte
>                        //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                        //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                        //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> wkn = t.getText();
> wkn.trim();
> p2.removeAll();
> leseDaten(dateiname);
> p2.add("Center",zeigeCanvasAn());
> pack();
> show();
> }
> }
> );
357,358d360
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction() erweitert die action-Methode.
403a406,418
> schliessen.addActionListener(
>                        //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                        //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                        //(in diesem Fall der schliessen-Knopf) spezialisierte
>                        //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                        //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                        //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
```

```
> }
> );
404a420,432
> beenden.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der beenden-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
514,553d541
< /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<     Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<     anderen Ereignisse wird "myaction" aufgerufen.
<   */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
< /* myaction erweitert die action-Methode, so dass nun beim Druck des
<     Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
<     aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
<     und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
<   */
< boolean myaction(Event ev, Object target){
< if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
< wkn = t.getText();
< wkn.trim();
< p2.removeAll();
< leseDaten(dateiname);
< p2.add("Center",zeigeCanvasAn());
< pack();
< show();
< return true;
< }
< return false;
< }
<
557a546,564
> ok.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der ok-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
```

```
>                                    //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> wkn = t.getText();
> wkn.trim();
> p2.removeAll();
> leseDaten(dateiname);
> p2.add("Center",zeigeCanvasAn());
> pack();
> show();
> }
> }
> );
574c581
< MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
---
> MonatsChartWert(String) bekommt einen String, in dem das Datum
623,624d629
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
657a663,675
> schliessen.addActionListener(
>                                    //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                    //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                                    //(in diesem Fall der schliessen-Knopf) spezialisierte
>                                    //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                    //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                    //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
658a677,689
> beenden.addActionListener(
>                                    //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                    //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                                    //(in diesem Fall der beenden-Knopf) spezialisierte
>                                    //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                    //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                    //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
776,802d806
<                                    /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<       Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<       anderen Ereignisse wird "myaction" aufgerufen.
<    */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
```

```
< }
< } else {
< return false;
< }
< }
<
<                                                       /* Wird von "action" aufgerufen.
<     */
< boolean myaction(Event ev, Object target){
< return true;
< }
<
1042,1043d1045
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
1077a1080,1092
> schliessen.addActionListener(
>                        //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                         //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                                 //(in diesem Fall der schliessen-Knopf)  spezialisierte
>                             //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                        //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                                                    //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
1078a1094,1106
> beenden.addActionListener(
>                        //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                         //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                                 //(in diesem Fall der beenden-Knopf)  spezialisierte
>                             //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                        //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                                                    //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
1196,1222d1223
<                         /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<         Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<         anderen Ereignisse wird "myaction" aufgerufen.
<     */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
```

```
<
<                                        /* Wird von "action" aufgerufen.
<    */
< boolean myaction(Event ev, Object target){
< return true;
< }
<
1323,1324d1323
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
1359a1359,1371
> schliessen.addActionListener(
>                     //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                     //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                     //(in diesem Fall der schliessen-Knopf) spezialisierte
>                     //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                     //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                     //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
1360a1373,1385
> beenden.addActionListener(
>                     //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                     //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                     //(in diesem Fall der beenden-Knopf) spezialisierte
>                     //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                     //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                     //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
1478,1504d1502
<                     /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<       Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<       anderen Ereignisse wird "myaction" aufgerufen.
<    */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
<                                        /* Wird von "action" aufgerufen.
<    */
< boolean myaction(Event ev, Object target){
< return true;
```

```
< }
<
1596,1597d1593
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
1632a1629,1641
> schliessen.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der schliessen-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
1633a1643,1655
> beenden.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der beenden-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
1639c1661
<
---
>
1751,1777d1772
< /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<     Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<     anderen Ereignisse wird "myaction" aufgerufen.
<   */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
< /* Wird von "action" aufgerufen.
<   */
< boolean myaction(Event ev, Object target) {
< return true;
< }
```

```
<
1839,1840d1833
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
1875a1869,1881
> schliessen.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der schliessen-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
1876a1883,1895
> beenden.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der beenden-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
1993,2019d2011
< /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<     Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<     anderen Ereignisse wird "myaction" aufgerufen.
<  */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
< /* Wird von "action" aufgerufen.
<  */
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
2081,2082d2072
< action(Event, Object) verarbeitet Interaktionsereignisse
< myaction(Event, Object) wird in "action" aufgerufen
2118a2109,2121
```

```
> schliessen.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der schliessen-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
2119a2123,2135
> beenden.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der beenden-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
2236,2262d2251
< /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<     Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<     anderen Ereignisse wird "myaction" aufgerufen.
<   */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
< /* Wird von "action" aufgerufen.
<   */
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
2412a2402,2439
> ok.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der ok-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
```

```
> public void actionPerformed(ActionEvent e){
> if(cbg.getSelectedCheckbox() == cbtt) {
> TagesTabelle tt = new TagesTabelle("Tagestabelle");
> tt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbwt) {
> WochenTabelle wt = new WochenTabelle("Wochentabelle");
> wt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmt) {
> MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
> mt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbdt) {
> MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
> dt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtg) {
> Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
> tg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbltk) {
> LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
> ltg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtc) {
> TagesChart tc = new TagesChart("Tageschart");
> tc.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmc) {
> MonatsChart mc = new MonatsChart("Monatschart");
> mc.stelleDar();
> }
>
> }
> }
> );
2414c2441,2454
< pack();
---
> beenden.addActionListener(
>     //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>     //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>     //(in diesem Fall der beenden-Knopf) spezialisierte
>     //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>     //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>     //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> System.exit(0);
> }
> }
> );
> pack();
2417,2460d2456
<
< public boolean action(Event ev, Object target){
< if(ev.id == Event.ACTION_EVENT) {
< if(ev.target == ok) {
< if(cbg.getSelectedCheckbox() == cbtt) {
< TagesTabelle tt = new TagesTabelle("Tagestabelle");
< tt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbwt) {
< WochenTabelle wt = new WochenTabelle("Wochentabelle");
< wt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmt) {
< MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
< mt.stelleDar();
< return true;
```

```
< } else if(cbg.getSelectedCheckbox() == cbdt) {
< MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
< dt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtg) {
< Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
< tg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbltk) {
< LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
< ltg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtc) {
< TagesChart tc = new TagesChart("Tageschart");
< tc.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmc) {
< MonatsChart mc = new MonatsChart("Monatschart");
< mc.stelleDar();
< return true;
< }
< } else if(ev.target == beenden) {
< System.exit(0);
< return true;
< }
< }
< return false;
< }
2462d2457
<
```

B.3.2 Boerse.java (3 levels of inheritance)

```
7a8
> import java.awt.event.*;
63,68d63
< action(Event, Object) verarbeitet Interaktionsereignisse
< abstract myaction(Event, Object)
< wird in "action" aufgerufen; Unterklassen
< koennen hier durch ueberschreiben
< spezielle Ereignisbehandlung
< implementieren
71c66
< abstract class BoersenInfo extends Frame implements Konstanten{
---
> abstract class BoersenInfo extends Frame implements Konstanten {
93a89,102
> schliessen.addActionListener(
>                                   //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                   //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                                   //(in diesem Fall der schliessen-Knopf) spezialisierte
>                                   //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                   //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                   //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
>
94a104,116
> beenden.addActionListener(
>                                   //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                   //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                                   //(in diesem Fall der beenden-Knopf) spezialisierte
>                                   //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                   //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                   //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
108,136d129
<
<                                   /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<       Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<       anderen Ereignisse wird "myaction" aufgerufen.
<    */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
```

```
< return true;
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
< /* Wird von "action" aufgerufen. Kann in Unterklassen ueberschrieben
<     werden, um eine spezielle Ereignisbehandlung zu
<     implementieren. (Diese Konstruktion verhindert, dass die Behandlung
<     der Knoepfe "Schliessen" und "Beenden" jedesmal neu implementiert
<     werden muss.
<   */
< abstract boolean myaction(Event ev, Object target);
278,279d270
< myaction() erweitert die action-Methode der Oberklasse.
< abstract zeigeCanvasAn() liefert ein Canvas zurueck.
300a292,310
> ok.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der ok-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> wkn = t.getText();
> wkn.trim();
> p2.removeAll();
> leseDaten(dateiname);
> p2.add("Center",zeigeCanvasAn());
> pack();
> show();
> }
> }
> );
306,324d315
< /* myaction erweitert die action-Methode, so dass nun beim Druck des
<     Knopfes "ok" die ausgesuchte Wertpapierkennnummer gesetzt, Daten
<     aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
<     und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
<   */
< boolean myaction(Event ev, Object target){
< if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
< wkn = t.getText();
< wkn.trim();
< p2.removeAll();
< leseDaten(dateiname);
< p2.add("Center",zeigeCanvasAn());
< pack();
< show();
< return true;
< }
< return false;
< }
<
487c478
< MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
---
> MonatsChartWert(String) bekommt einen String, in dem das Datum
571,574d561
```

```
< boolean myaction(Event ev, Object target){
< return true;
< }
<
811,814d797
< boolean myaction(Event ev, Object target){
< return true;
< }
<
913,916d895
< boolean myaction(Event ev, Object target){
< return true;
< }
<
1007,1010d985
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
1071,1074d1045
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
1137,1140d1107
< boolean myaction(Event ev, Object target) {
< return true;
< }
<
1285a1253,1289
> ok.addActionListener(
>                          //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                          //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                          //(in diesem Fall der ok-Knopf) spezialisierte
>                          //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                          //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                          //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> if(cbg.getSelectedCheckbox() == cbtt) {
> TagesTabelle tt = new TagesTabelle("Tagestabelle");
> tt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbwt) {
> WochenTabelle wt = new WochenTabelle("Wochentabelle");
> wt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmt) {
> MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
> mt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbdt) {
> MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
> dt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtg) {
> Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
> tg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbltk) {
> LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
> ltg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtc) {
> TagesChart tc = new TagesChart("Tageschart");
> tc.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmc) {
> MonatsChart mc = new MonatsChart("Monatschart");
> mc.stelleDar();
> }
```

```
> }
> }
> );
1286a1291,1303
> beenden.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der beenden-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> System.exit(0);
> }
> }
> );
1289,1332d1305
< }
<
< public boolean action(Event ev, Object target){
< if(ev.id == Event.ACTION_EVENT) {
< if(ev.target == ok) {
< if(cbg.getSelectedCheckbox() == cbtt) {
< TagesTabelle tt = new TagesTabelle("Tagestabelle");
< tt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbwt) {
< WochenTabelle wt = new WochenTabelle("Wochentabelle");
< wt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmt) {
< MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
< mt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbdt) {
< MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
< dt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtg) {
< Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
< tg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbltk) {
< LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
< ltg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtc) {
< TagesChart tc = new TagesChart("Tageschart");
< tc.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmc) {
< MonatsChart mc = new MonatsChart("Monatschart");
< mc.stelleDar();
< return true;
< }
< } else if(ev.target == beenden) {
< System.exit(0);
< return true;
< }
< }
< return false;
```

B.3.3 Boerse.java (5 levels of inheritance)

```
7a8
> import java.awt.event.*;
64,69d64
< action(Event, Object) verarbeitet Interaktionsereignisse
< abstract myaction(Event, Object)
< wird in "action" aufgerufen; Unterklassen
< koennen hier durch ueberschreiben
< spezielle Ereignisbehandlung
< implementieren
72c67
< abstract class BoersenInfo extends Frame implements Konstanten{
---
> abstract class BoersenInfo extends Frame implements Konstanten {
94a90,102
> schliessen.addActionListener(
>                                 //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                 //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                                 //(in diesem Fall der schliessen-Knopf) spezialisierte
>                                 //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                 //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                 //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> schliesseFenster();
> }
> }
> );
95a104,116
> beenden.addActionListener(
>                                 //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
>                                 //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
>                                 //(in diesem Fall der beenden-Knopf) spezialisierte
>                                 //actionPerformed()-Methode zur Verfuegung. Sie wird immer
>                                 //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
>                                 //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> beendeProgramm();
> }
> }
> );
109,137d129
<
<                                 /* Wird bei Interaktions-Ereignissen aufgerufen. Druecken der
<         Knoepfe "Schliessen" und "Beenden" wird ausgewertet, fuer alle
<         anderen Ereignisse wird "myaction" aufgerufen.
<     */
< public boolean action(Event event, Object target) {
< if(event.id == Event.ACTION_EVENT) {
< if(event.target == schliessen) {
< schliesseFenster();
< return true;
< }
< else if(event.target == beenden) {
< beendeProgramm();
< return true;
```

```
< } else {
< return myaction(event, target);
< }
< } else {
< return false;
< }
< }
<
< /∗ Wird von ”action” aufgerufen. Kann in Unterklassen ueberschrieben
< werden, um eine spezielle Ereignisbehandlung zu
< implementieren. (Diese Konstruktion verhindert, dass die Behandlung
< der Knoepfe ”Schliessen” und ”Beenden” jedesmal neu implementiert
< werden muss.
< ∗/
< abstract boolean myaction(Event ev, Object target);
279d270
< myaction() erweitert die action-Methode der Oberklasse.
301a293,311
> ok.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der ok-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> wkn = t.getText();
> wkn.trim();
> p2.removeAll();
> leseDaten(dateiname);
> p2.add("Center",zeigeCanvasAn());
> pack();
> show();
> }
> }
> );
306,325d315
<
< /∗ myaction erweitert die action-Methode, so dass nun beim Druck des
< Knopfes ”ok” die ausgesuchte Wertpapierkennnummer gesetzt, Daten
< aus der jeweiligen Datei mit erzeugeDatenvector eingelesen,
< und eine Grafik (Canvas) mit der Methode zeigeCanvasAn() angezeigt wird
< ∗/
< boolean myaction(Event ev, Object target){
< if((ev.id == Event.ACTION_EVENT) && (ev.target == ok)) {
< wkn = t.getText();
< wkn.trim();
< p2.removeAll();
< leseDaten(dateiname);
< p2.add("Center",zeigeCanvasAn());
< pack();
< show();
< return true;
< }
< return false;
< }
<
488c478
< MonatsChartAnzeiger(String) bekommt einen String, in dem die Uhrzeit
---
> MonatsChartWert(String) bekommt einen String, in dem das Datum
552,555d541
< boolean myaction(Event ev, Object target){
```

```
< return true;
< }
<
962,964d947
< boolean myaction(Event ev, Object target) {
< return true;
< }
979,981d961
< boolean myaction(Event ev, Object target) {
< return true;
< }
1141a1122,1158
> ok.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der ok-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> if(cbg.getSelectedCheckbox() == cbtt) {
> TagesTabelle tt = new TagesTabelle("Tagestabelle");
> tt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbwt) {
> WochenTabelle wt = new WochenTabelle("Wochentabelle");
> wt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmt) {
> MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
> mt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbdt) {
> MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
> dt.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtg) {
> Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
> tg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbltk) {
> LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
> ltg.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbtc) {
> TagesChart tc = new TagesChart("Tageschart");
> tc.stelleDar();
> } else if(cbg.getSelectedCheckbox() == cbmc) {
> MonatsChart mc = new MonatsChart("Monatschart");
> mc.stelleDar();
> }
> }
> }
> );
1142a1160,1172
> beenden.addActionListener(
> //Die folgende Konstruktion bedeutet, dass eine unbenannte innere
> //Klasse erzeugt wird. Diese stellt eine fuer das aeussere Objekt
> //(in diesem Fall der beenden-Knopf) spezialisierte
> //actionPerformed()-Methode zur Verfuegung. Sie wird immer
> //dann ausgefuehrt, sobald eine Aktion auf dem aeusseren Objekt
> //ausgefuehrt wird.
> new ActionListener(){
> public void actionPerformed(ActionEvent e){
> System.exit(0);
> }
> }
> );
1147,1189d1176
```

```
< public boolean action(Event ev, Object target){
< if(ev.id == Event.ACTION_EVENT) {
< if(ev.target == ok) {
< if(cbg.getSelectedCheckbox() == cbtt) {
< TagesTabelle tt = new TagesTabelle("Tagestabelle");
< tt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbwt) {
< WochenTabelle wt = new WochenTabelle("Wochentabelle");
< wt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmt) {
< MonatsTabelle mt = new MonatsTabelle("Monatstabelle");
< mt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbdt) {
< MonatsDynamikTabelle dt = new MonatsDynamikTabelle("Dynamiktabelle");
< dt.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtg) {
< Tagesgewinne tg = new Tagesgewinne("Tagesgewinne");
< tg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbltk) {
< LetzteTageskurse ltg = new LetzteTageskurse("letzte Tageskurse");
< ltg.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbtc) {
< TagesChart tc = new TagesChart("Tageschart");
< tc.stelleDar();
< return true;
< } else if(cbg.getSelectedCheckbox() == cbmc) {
< MonatsChart mc = new MonatsChart("Monatschart");
< mc.stelleDar();
< return true;
< }
< } else if(ev.target == beenden) {
< System.exit(0);
< return true;
< }
< }
< return false;
< }
```

Appendix C

Translations of German terms

The following translations are given in order to help the reader understand the German Java source code.

abstand	distance
Auswahl	selection
beendeProgramm	terminateProgram
beenden, Beenden	terminate
berechneDynamik	calculateDynamics
Bitte	please
Boerse	stock exchange
Boersen-Information	stock information
BoersenInfo	stock info
ChartAnzeiger	chartDisplay
Datei	file
dateiname	file name
Daten	data
Datum, datum	date
Dynamiktabelle	DynamicsTable
erzeugeDatenvector	generateDataVector
gefunden	found
geoeffnet	opened
gesponsort	sponsored
gewuenschten	desired
heute, heutige	today, today's
jahr	year
KASSAKURSE, kassakurse, kassakurs	spot rates
keine	no, none
keineDaten	noData
kleinergleich	lessThan
kosten	costs
KURSE, Kurs, kurs, kurse	price, prices
Konstanten	constants

KostenloseInfos	FreeInfos
KostenpflichtigeInfos	InfoWithCosts
leseDaten	readData
Lesefehler	ReadError
letzte, letzter	last
letzterMonat	lastMonth
letztetageskurse, LetzteTageskurse	lastDayPrice
monat	month
MonatsChart, Monatschart	MonthChart
MonatsChartAnzeiger	MonthChartDisplay
MonatsChartWert	MonthChartValue
Monatsdaten	data for month
MonatsDynamikTabelle	MonthDynamicTable
MonatsTabelle, Monatstabelle	MonthTable
monatzurueck	month back
nicht	not
schliesseFenster	closeWindow
schliessen, Schliessen	close
schonAbgearbeitet	alreadyDone
Seiten	pages
selektiere	select
setzeDateiname	setFileName
setzeDatum	setDate
sortiere	sort
stelleDar	display, present
stunde	hour
Tabelle	table
tag	day
Tages-Chart, TagesChart, Tageschart	DayChart
TagesChartAnzeiger	DayChartDisplay
TagesChartWert	DayChartValue
Tagesdaten, tagesdaten	data for day
TagesTabelle	DayTable
tagesgewinn, Tagesgewinne	DayGains
Tageskurse	DayPrices
Tagestabelle	DayTable
titel	title
Uhrzeit, uhrzeit	time of day
vergleiche	compare
vonDatum	fromDate
Wertpapierkennummer	stock ID number
WochenTabelle	WeekTable
wochezurueck	week back
zeigeCanvasAn	presentCanvas
zeigeTabelleAn	presentTable
Zeitraum	time interval
ZeitraumTabelle	IntervalTable
Zeitraumdaten	IntervalData

Bibliography

[1] Larry B. Christensen. *Experimental Methodology*. Allyn and Bacon, Needham Heights, MA, 6th edition, 1994.

[2] J. Daly, J. Miller, J. Brooks, M. Roper, and M. Wood. Issues on the object-oriented paradigm: A questionnaire survey. Technical report, University of Strathclyde, EFoCS-8-95, 1995.

[3] J. Daly, M. Wood, J. Brooks, J. Miller, and M. Roper. Structured interviews object-oriented paradigm. Technical report, University of Strathclyde, EFoCS-7-95, 1995.

[4] John W. Daly, Andrew Brooks, James Miller, Marc Roper, and Murray Wood. An empirical study evaluating depth of inheritance on the maintainability of object-oriented software. *Empirical Software Engineering, An international Journal*, 1(2):109–132, 1996.

[5] Al Davis. Eras of software technology transfer. *IEEE Software*, 13(2):4–7, March 1996.

[6] Norman E. Fenton. *Software Metrics: A Rigorous Approach*. Chapman & Hall, 1991.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] Lutz Prechelt, Barbara Unger, and Douglas Schmidt. Replication of the first controlled experiment on the usefulness of design patterns: Detailed description and evaluation. Technical report, Washington University, St. Louis, Dept. of CS, 1997.

[9] Helmut Roderus and Dieter Krißgau. Erfahrungen mit dem Einsatz objektorientierter Methoden. *OBJEKTspektrum*, 1(1):43–47, 94.