1. **Aufgabenblatt** zur Veranstaltung *Computergrafik und Visualisierung II* im Sommersemester 2020

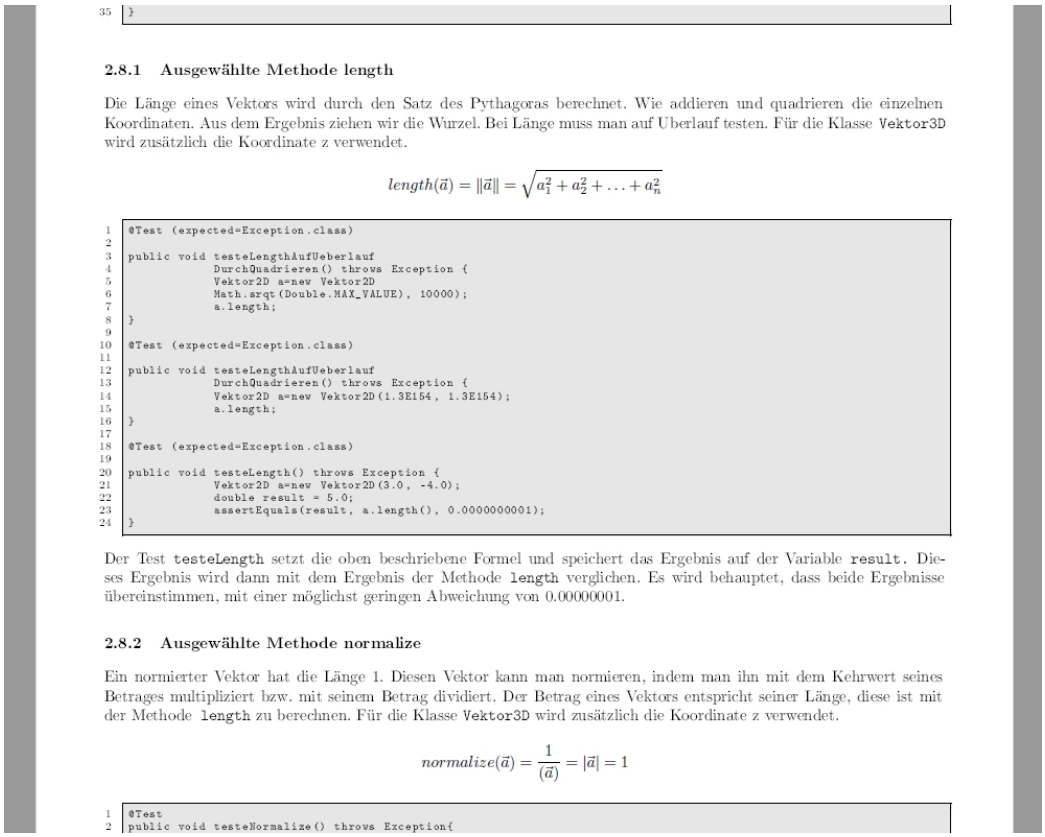
Kommentare und Hinweise

Hochschule für Technik und Wirtschaft Dresden, Friedrich-List-Platz 1, 01069 Dresden
Prof. Dr. Marco Block-Berlitz, Rainer Uhlemann

Allgemeine Kommentare zu den Abgaben

Wir werden uns im Laufe des Semesters immer besser einspielen. Zu den Lösungen in PDF-Form werden auch immer die Quelldateien abzugeben sein. Bei größeren Projekten erwarte ich dann entsprechend Eclipse-Pakete. Alle gepackten Dateien bitte immer als **ZIP-Dateien** packen.

In Zukunft wird parallel zu den Quelldateien (bei denen auch gerne eine Kommentierung vorhanden ist) eine weitere Datei erwartet, bei denen die relevanten Lösungsabschnitte entsprechend erläutert und vorgestellt werden. Ich habe mir erlaubt, ein abgegebenes Beispiel mal hier zu zeigen:



```
35 }
```

2.8.1 Ausgewählte Methode length

Die Länge eines Vektors wird durch den Satz des Pythagoras berechnet. Wie addieren und quadrieren die einzelnen Koordinaten. Aus dem Ergebnis ziehen wir die Wurzel. Bei Länge muss man auf Uberlauf testen. Für die Klasse `Vektor3D` wird zusätzlich die Koordinate z verwendet.

$$length(\vec{a}) = \|\vec{a}\| = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$$

```
1  @Test (expected=Exception.class)
2
3  public void testeLengthAufUeberlauf
4              DurchQuadrieren() throws Exception {
5              Vektor2D a=new Vektor2D
6              Math.sqrt(Double.MAX_VALUE), 10000);
7              a.length;
8  }
9
10 @Test (expected=Exception.class)
11
12 public void testeLengthAufUeberlauf
13             DurchQuadrieren() throws Exception {
14             Vektor2D a=new Vektor2D(1.3E154, 1.3E154);
15             a.length;
16 }
17
18 @Test (expected=Exception.class)
19
20 public void testeLength() throws Exception {
21             Vektor2D a=new Vektor2D(3.0, -4.0);
22             double result = 5.0;
23             assertEquals(result, a.length(), 0.0000000001);
24 }
```

Der Test `testeLength` setzt die oben beschriebene Formel und speichert das Ergebnis auf der Variable `result`. Dieses Ergebnis wird dann mit dem Ergebnis der Methode `length` verglichen. Es wird behauptet, dass beide Ergebnisse übereinstimmen, mit einer möglichst geringen Abweichung von 0.00000001.

2.8.2 Ausgewählte Methode normalize

Ein normierter Vektor hat die Länge 1. Diesen Vektor kann man normieren, indem man ihn mit dem Kehrwert seines Betrages multipliziert bzw. mit seinem Betrag dividiert. Der Betrag eines Vektors entspricht seiner Länge, diese ist mit der Methode `length` zu berechnen. Für die Klasse `Vektor3D` wird zusätzlich die Koordinate z verwendet.

$$normalize(\vec{a}) = \frac{1}{(\vec{a})} = |\vec{a}| = 1$$

```
1  @Test
2  public void testeNormalize() throws Exception{
```

Jetzt wird dem einen und anderen sicherlich klarer, warum Lyx bzw. Latex gefragt sind.

Jetzt inhaltlich zum praktischen Teil

1. Vektor2D und Vektor3D [10 Punkte]
Erstellen Sie die zwei Klassen `Vektor2D` und `Vektor3D`, die entsprechend 2D- oder 3D-Vektoren repräsentieren können. Die Koordinaten werden jeweils durch `double` repräsentiert. Weiterhin sollen sinnvolle Hilfsmethoden angeboten werden. Es sind mindestens die folgenden Funktionen zu implementieren: `setPosition`, `isNullVector`, `add`, `sub`, `mult`, `div`, `isEqual`, `isNotEqual`, `length` und `normalize`. Dokumentieren Sie Ihre Entwicklung der Methoden `add` und `normalize` und mittels Test-Driven-Development in geeigneter Weise.

Kommentar zu Booleschen Rückgabewerten

Wir entwickeln viele Methoden, bei denen einfache Überprüfungen (boolesche Ausdrücke) vorgenommen werden. In diesem Beispiel wird geprüft, ob es sich um den NullVektor handelt:

```
1 public boolean isNullVektor() {
2    if (x==0.0 && y==0.0)
3       return true;
4    else
5       return false;
6 }
```

Die Lösung liefert das korrekt Ergebnis. Aus softwareästhetischer Sicht wäre folgende Lösung vorzuziehen:

```
1 public boolean isNullVektor() {
2    return (x==0.0 && y==0.0);
3 }
```

In beiden Fällen wird ein boolescher Ausdruck ausgewertet und das Ergebnis direkt zurückgeliefert. Analog bitte auch die Methoden `isEqual` und `isNotEqual` prüfen.

Verkettung von Methoden

Das ist ein wichtiges Werkzeug, um Redundanz und damit die Fehleranfälligkeit von Programmen zu verringern. Diese Lösung

```
1  public boolean isEqual(Vektor2D a) {
2     if (this.x==a.x && this.y==a.y)
3        return true;
4     return false;
5  }
6
7  public boolean isNotEqual(Vektor2D a) {
8     if (this.x==a.x && this.y==a.y)
9        return false;
10    return true;
11 }
```

wird analog zum vorhergehenden Kommentar und unter Anwendung der Methodenverkettung wie folgt verbessert:

```
1 public boolean isEqual(Vektor2D a) {
2    return (this.x==a.x && this.y==a.y)
3 }
4
5 public boolean isNotEqual(Vektor2D a) {
6    return !isEqual(a);
7 }
```

2. Lineare Algebra [20 Punkte]
Schreiben Sie eine Klasse `LineareAlgebra`, die folgende statische Methoden (für 2D und 3D) zur Verfügung stellt: `add`, `sub`, `mult`, `div`, `isEqual`, `isNotEqual`, `length`, `normalize`, `euklDistance`, `manhattanDistance`, `crossProduct`, `dotProduct`, `cosEquation`, `sinEquation`, `angleRad`, `angleDegree`, `radToDegree`, `degreeToRad`, `determinante`, `abs` und `show`. Hinweis: Da die Klasse nur statische Methoden enthält, sollten sie die Einschränkung der Konstruktoren beachten.

Kommentar zu Konstruktor

Da die Klasse `LineareAlgebra` nur Methoden anbietet, die statisch sind (siehe Hinweis bei der Aufgabenstellung), sollten wir aus softwareästhetischer Sicht einen leeren privaten Konstruktor angeben. Damit ist dem Anwender der Klasse klar, dass er kein Objekt dieser Klasse erzeugen muss, um mehr Methodik zu erhalten:

```
1 private LineareAlgebra() {}
```

Das war eine Klausuraufgabe bei Programmierung III.

Umgang mit Fehlern bzw. Eingaben, die zu Fehlern führen

In diesem Beispiel wird zwar darauf geachtet, dass eine Division durch 0 zu einer Exception führt und der Anwender entsprechend informiert:

```
1 public static Vector2D div(Vector2D vec, float s) {
2    if (s == 0) {
3       System.out.println("Division durch 0 nicht möglich, es wurde ein
4                           leerer Vector erstellt");
5       return new Vector2D();
6    }
7    return new Vector2D(vec.x / s, vec.y / s);
8 }
```

Aber woher wissen wir, dass es eine Konsole für eine Ausgabe gibt? **Fehlermeldungen sollten in solchen Situationen nie selbst behandelt werden.** Wir geben dem Anwender mit `throw` eine Exception zurück und bitten Ihn, den Fehler selbst zu prüfen und eine Eingabe möglicherweise selbst zu wiederholen.

Unterschied zwischen Vektor2D, Vektor3D und LineareAlgebra

In den Klassen `Vektor2D` und `Vektor3D` gehen wir davon aus, jeweils einen Vektor direkt zu repräsentieren. Damit werden auch alle Operationen auf diesen Vektoren ausgeführt.

Anders verhält es sich bei der Klasse `LineareAlgebra`:

```
1 public static Vektor2D add(Vektor2D vek1, Vektor2D vek2) {
2    return vek1.add(vek2);
3 }
```

Da es sich um statische Methoden handelt, die wie in diesem Beispiel eine Vektoraddition ausführen sollen, darf keiner der beiden Eingabevektoren verändert werden. Es sollte vielmehr ein neuer Vektor erzeugt werden:

```
1 public static Vektor2D add(Vektor2D vec1, Vektor2D vec2) {
2    return new Vektor2D(vec1.x + vec2.x, vec1.y + vec2.y);
3 }
```

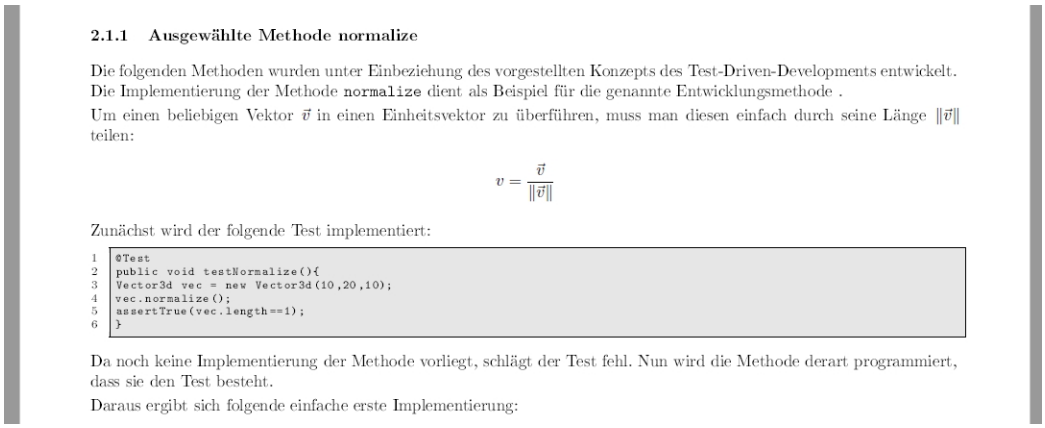
Analog sollte die Lösung in Vektor3D so nicht aussehen:

```
1 public Vektor3D add(Vektor3D vec) {
2    return new Vektor3D(this.x + vec.x, this.y + vec.y, this.z + vec.z);
3 }
```

Jetzt ist klar warum, oder?

Test-Driven-Development

Wie wir Test-Driven-Development (TDD) in Java mit JUnit machen, haben wir in *Programmierung III* kennengelernt. Bei Bedarf bitte einfach nochmal die Folien und Übungen bei Opal durcharbeiten (pw: pro). Hier eine Lösung, wie es aussehen könnte, wenn nachvollziehbar dargestellt werden soll, dass TDD angewendet wurde:



2.1.1 Ausgewählte Methode normalize

Die folgenden Methoden wurden unter Einbeziehung des vorgestellten Konzepts des Test-Driven-Developments entwickelt. Die Implementierung der Methode `normalize` dient als Beispiel für die genannte Entwicklungsmethode .
Um einen beliebigen Vektor $\vec{v}$ in einen Einheitsvektor zu überführen, muss man diesen einfach durch seine Länge $\|\vec{v}\|$ teilen:

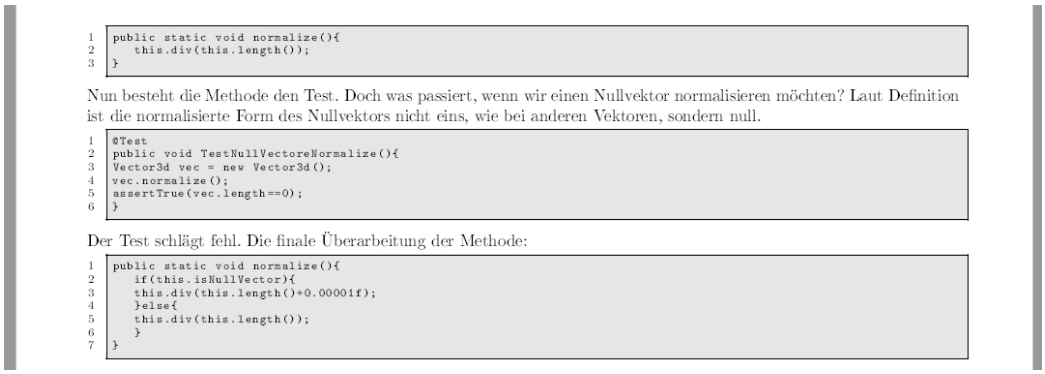
$$v = \frac{\vec{v}}{\|\vec{v}\|}$$

Zunächst wird der folgende Test implementiert:

```
1 @Test
2 public void testNormalize(){
3 Vector3d vec = new Vector3d(10,20,10);
4 vec.normalize();
5 assertTrue(vec.length==1);
6 }
```

Da noch keine Implementierung der Methode vorliegt, schlägt der Test fehl. Nun wird die Methode derart programmiert, dass sie den Test besteht.
Daraus ergibt sich folgende einfache erste Implementierung:

Auf diese Weise kann Schritt für Schritt nachvollzogen werden, wie die Lösung zur Methode `normalize` entsteht:



```
1 public static void normalize(){
2    this.div(this.length());
3 }
```

Nun besteht die Methode den Test. Doch was passiert, wenn wir einen Nullvektor normalisieren möchten? Laut Definition ist die normalisierte Form des Nullvektors nicht eins, wie bei anderen Vektoren, sondern null.

```
1 @Test
2 public void TestNullVectoreNormalize(){
3 Vector3d vec = new Vector3d();
4 vec.normalize();
5 assertTrue(vec.length==0);
6 }
```

Der Test schlägt fehl. Die finale Überarbeitung der Methode:

```
1 public static void normalize(){
2    if(this.isNullVector){
3    this.div(this.length()+0.00001f);
4    }else{
5    this.div(this.length());
6    }
7 }
```

Hier sehen wir auch, warum wir Latex oder Lyx für die Erstellung der Lösung einsetzen wollen. Ausgewählte Codepassagen sollen Eure Arbeit untermauern und plausibel darstellen. Spätestens bei der Abschlussarbeit werdet Ihr Euch damit auseinandersetzen müssen. Auch dort werdet Ihr Formeln und Programmierlösungen vorstellen und erläutern und nicht einfach kommentierten Programmcode abgeben.

Lösungen

Eine vollständige Lösung ist in der Programmsammlung[1] zum Veranstaltungsbuch zu finden [1].

Literatur

[1] Block-Berlitz, M.: *„Warum sich der Dino furchtbar erschreckte: Lehrbuch zu Beleuchtung und Rendering mit Java, LWJGL, OpenGL und GLSL“*, vividus Wissenschaftsverlag, 2019

[1] *http://www.vividus-verlag.de/beleuchtung_und_rendering/index.html*