P. Thiemann, L. Fennell Sommersemester 2017

Programmieren in Java

`http://proglang.informatik.uni-freiburg.de/teaching/java/2017/`

unit-tests

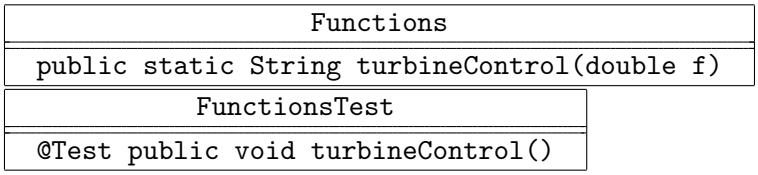
Unit-Tests

Woche 05 Aufgabe 1/4

Herausgabe: 2017-05-22
Abgabe: 2017-06-02

Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project `unit-tests`
Package `unittests`
Klassen



In dieser Aufgabe sollen für vorhandene Funktionen Unit-Tests mit JUnit4 geschrieben werden. Im Paket befindet sich ein vorkonfiguriertes IntelliJ-Projekt. Dort finden Sie die Klasse `unittests.Functions`, die die Implementierung der schon aus Aufgabe `conditional-functions` bekannten Funktion `turbineControl` enthält.

```
public static String turbineControl(double f)
```

Abhängig von der gegebenen Frequenz `f` werden Kommandos für eine Turbine als String ausgegeben.

- `"DISCONNECT"` falls $f \leq 49$ oder $f \geq 51$,
- `"MORE_WATER"` falls $f \lesssim 50$,
- `"LESS_WATER"` falls $f \gtrsim 50$ und
- `"STEADY"` falls $f \approx 50$.

Zwei Zahlen sind annähernd gleich ($\approx$), falls ihr Unterschied weniger als 0.001 beträgt. Entsprechend sind auch $\lesssim$ („signifikant kleiner") und $\gtrsim$ („signifikant größer") definiert.

Benutzen Sie JUnit4 um Tests für `turbineControl` zu schreiben. Die Tests müssen in einer Klasse `unittests.FunctionsTest` stehen. Ihre Testfälle müssen so beschaffen sein, dass sie für Methoden `turbineControl` Statement-Abdeckung herstellen. Das heisst, jedes Statement der Methoden muss durch mindestens einen Testfall ausgeführt werden. *Verändern sie auf*

keinen Fall die Implementierungen in der Klasse ***`Functions`***; bearbeiten Sie ausschließlich die Datei `test/unittests/FunctionsTest.java`.

Empfehlung: Nutzen Sie diese Aufgabe um sich mit dem Ausführen und Schreiben von JUnit-Tests vertraut zu machen.

Damit Jenkins die Codeabdeckung allerdings gut prüfen kann, sind die beiden Implementierungen mit Kontrollausgaben versehen (das sind die `log`-Aufrufe).

Die Ausgaben können Sie durch Ausführen von `test/unittests/Main.java` betrachten. Bei folgender Ausgabe ist Ihre Abdeckung in Ordnung:

```
Messages:
turbineControl:DISCONNECT:false
turbineControl:DISCONNECT:true
turbineControl:LESS_WATER
turbineControl:MORE_WATER
turbineControl:STEADY
```

P. Thiemann, L. Fennell Sommersemester 2017

Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2017/

line-intersection

Schnittpunkt zweier Geraden

Woche 05 Aufgabe 2/4

Herausgabe: 2017-05-22
Abgabe: 2017-06-02

Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project `line-intersection`
Package `lineintersection`
Klassen

Geometry

`public static String lineIntersection(`
`  double a1, double b1`
`  double a2, double b2)`

GeometryTest

`@Test public void lineIntersection()`
`public boolean checkIntersection(`
`  double[] resultPoint,`
`  double a1, double b1,`
`  double a2, double b2,`
`  double deltaX, double deltaY)`

In dieser Aufgabe sollen für vorhandene Funktionen Unit-Tests mit JUnit4 geschrieben werden. Im Paket befindet sich ein vorkonfiguriertes IntelliJ-Projekt. Dort finden Sie die Klasse `lineintersection.Geometry`, die die Implementierung der folgenden Funktion enthält:

```
public static double[] lineIntersection(double a1, double b1, double a2,
                                        double b2)
```

Sie berechnet den Punkt, an dem sich zwei gegebene Geraden schneiden. Die Geraden ergeben sich aus den Gleichungen $y = \texttt{a1} * x + \texttt{b1}$ und $y = \texttt{a2} * x + \texttt{b2}$. Das Ergebnis ist ein Array mit zwei Elementen, dass die x-Koordinate als erstes Element und die y-Koordinate als zweites Element enthält. Gibt es keinen Schnittpunkt, so soll `null` zurückgegeben werden.

Benutzen Sie JUnit4 um Tests für die Funktion zu schreiben. Die Tests müssen in der Klasse `lineintersection.GeometryTest` stehen. Außerdem sollen sie in den Tests ein so genanntes *Testorakel* verwenden. Ein Testorakel bestimmt, ob ein berechnetes Ergebnis zu einer gegebenen Eingabe korrekt ist. In dieser Aufgabe ist das Testorakel die oben genannte Funktion `lineintersection.GeometryTest.checkIntersection`. Sie soll `true` zurückgeben, genau dann wenn der Punkt `resultPoint` ein korrekter Schnittpunkt von für die

Geraden $y = \texttt{a1} * x + \texttt{b1}$ und $y = \texttt{a2} * x + \texttt{b2}$ ist. Da bei `double`-Werten leicht Rundungsfehler auftreten können, erlaubt das Orakel zusätzlich die Angabe der Toleranzwerte `deltaX` und `deltaY`. Die x-Koordinate von `resultPoint` ist demnach korrekt, wenn sie weniger als `deltaX` vom genauen Schnittpunkt abweicht. Analog dazu ist die y-Koordinate von `resultPoint` korrekt, wenn sie weniger als `deltaY` vom genauen Schnittpunkt abweicht.

Die Tests sollen das Orakel nach folgendem Muster verwenden:

```
1 assertTrue(checkIntersection(Geometry.lineIntersection(a1, b1, a2, b2),
2                              a1, b1, a2, b2, ...));
```

Wird dieses Muster nicht befolgt, gibt es keinen Punkt für Codequalität.

Ihre Testfälle müssen außerdem wieder so beschaffen sein, dass sie für die Methode `lineIntersection` Statement-Abdeckung herstellen. Das heisst, jedes Statement der Methoden muss durch mindestens einen Testfall ausgeführt werden. Sie dürfen hier wieder ausschließlich die Datei `test/lineintersection/GeometryTest.java` bearbeiten. Die Abdeckung kann wieder durch Ausführen von `test/lineintersection/Main.java` betrachten. Bei folgender Ausgabe ist Ihre Abdeckung in Ordnung:

```
Messages:
lineIntersection:intersect
lineIntersection:parallel
```

P. Thiemann, L. Fennell Sommersemester 2017

Programmieren in Java

`http://proglang.informatik.uni-freiburg.de/teaching/java/2017/`

format-si

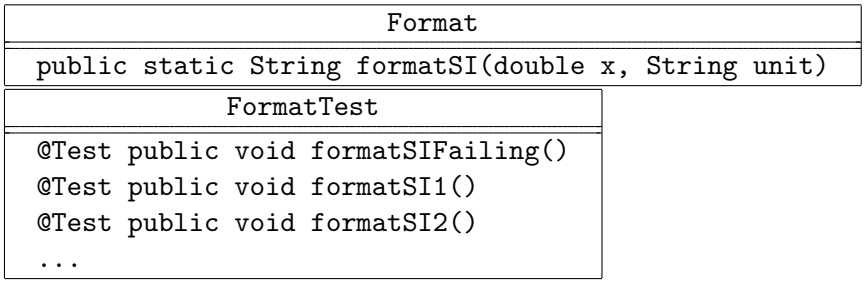
Formatierung von physikalischen Messgrößen

Woche 05 Aufgabe 3/4

Herausgabe: 2017-05-22
Abgabe: 2017-06-02

Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project `format-si`
Package `formatsi`
Klassen



Ähnlich wie in Aufgabe `unit-tests` sollen Sie Tests für eine gegebene Funktion schreiben. Die Funktion `Format.formatSI` finden Sie im mitgelieferten Projekt. Sie gibt einen leserlichen String für eine gegebene physikalische Messgröße mit gegebener Einheit aus. Näheres können Sie der Javadoc-Dokumentation der Funktion entnehmen.

Wie in `unit-tests` dürfen Sie hier nur die Klasse `test/formatsi/FormatTest.java` bearbeiten. Anders als in `unit-tests` enthält die Implementierung von `formatSI` einen Fehler, den ihre Tests finden sollen. Schreiben Sie dazu eine Test-Methode `formatSIFailing`, die den Fehler aufdeckt. Beschreiben Sie in `formatSIFailing` (kurz, als Kommentar) was in der Implementierung falsch ist und wie sich der Fehler beheben lässt. Die Beschreibung ist notwendig um den Punkt für Codequalität in dieser Aufgabe zu bekommen.

Trotz des Fehlers sollen Sie dafür sorgen, dass beim Ausführen der Tests alle Statements abgedeckt werden. Schreiben Sie also noch weitere Test-Methoden (z.B. `formatSI1`, `formatSI2`, ..., aber der Name ist eigentlich egal). Wenn Sie `test/formatsi/Main.java` ausführen, Ihre Tests alle Statements abdecken, und der Fehler gefunden wird, sollte folgende Ausgabe zu sehen sein:

```
Test failed: formatSIFailing(formatsi.FormatTest)
Messages:
formatSI:bigger-1
formatSI:no-fit:false
```

```
formatSI:no-fit:true
formatSI:no-symbol
formatSI:smaller-1
formatSI:symbol
formatSI:zero
```

P. Thiemann, L. Fennell Sommersemester 2017

Programmieren in Java

`http://proglang.informatik.uni-freiburg.de/teaching/java/2017/`

subsequence

Teilfolgenordnung auf Strings

Woche 05 Aufgabe 4/4

Herausgabe: 2017-05-22
Abgabe: 2017-06-02

Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project	`subsequence`
Package	`subsequence`
Klassen	

`Main`
`public static PartialOrdering subsequenceCompare(String s1, String s2)`

Achtung: Von jetzt an ist es notwendig, dass Sie ihre Methoden sinnvoll mit JUnit testen, um die Punkte für Codequalität zu bekommen.

Eine Teilfolge eines Strings ist ein String, der entsteht wenn Buchstaben des ursprünglichen Strings weggelassen werden.

`https://de.wikipedia.org/wiki/Teilfolge`

Strings können durch die Teilfolge-Eigenschaft partiell geordnet werden. Die Funktion `subsequenceCompare` berechnet wie die zwei Eingabestrings `s1` und `s2` bezüglich der Teilfolge-Eigenschaft geordnet sind. Das Ergebnis von `subsequenceCompare` ist vom `enum`-Typ `PartialOrdering`:

```
1 public enum PartialOrdering {
2     LESS,
3     EQUAL,
4     GREATER,
5     INCOMPARABLE
6 }
```

Ein `enum`-Typ definiert eine abgeschlossene, endliche Menge von Konstanten, die im Programm eine besondere Bedeutung haben. Im Fall von `PartialOrdering` gibt es die folgenden vier Konstanten. Sie sollen angeben, wie zwei Strings zueinander geordnet sind.

- `PartialOrdering.LESS`: `s1` ist Teilfolge von `s2`, aber nicht gleich `s2`
- `PartialOrdering.EQUAL`: `s1` ist gleich `s2` (und somit auch Teilfolge von `s2`).

- `PartialOrdering.GREATER`: `s2` ist Teilfolge von `s1`, aber nicht gleich `s1`.
- `PartialOrdering.INCOMPARABLE`: `s1` ist weder Teilfolge von `s2`, noch ist `s2` Teilfolge von `s1`

Die Funktion sollte auch für relativ große Strings (z.B. Länge 10000) gut funktionieren.

Beispielaufruf 21: `Main.subsequenceCompare("Hello World", "eloWrld")` ergibt `GREATER` (als `PartialOrdering`)

Beispielaufruf 22: `Main.subsequenceCompare("Hello World", "World Hello")` ergibt `INCOMPARABLE` (als `PartialOrdering`)

Hinweise: Weitere Informationen zu `enum`-Typen finden Sie hier:

`http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html`.