Programmieren 1

Dr. Arne Schmidt

Wintersemester 2023/2024
Abgabe: 02.02.2024
Rückgabe: ab 05.02.2024

Übungsblatt 5

Abgabe der Lösungen bis zum 02.02.2024 um 18:00 Uhr im Gitlab vom IBR. Es werden nur ausführbare und rechtzeitig eingereicht Lösungen bewertet. Python-Programme müssen mit python3 interpretierbar, und Java-Programme mit Java 11 kompilierbar sein. Sofern nicht anders in der Aufgabe angegeben, dürfen **keine** Standardfunktionen benutzt oder Bibliotheken importiert werden, wenn sie nicht in der Vorlesung behandelt wurden.

Sofern nicht anders in der Aufgabe beschrieben, soll im Git die Ordnerstruktur wie vom Blatt vorgegeben beibehalten werden. D.h. für eine Aufgabe x von Blatt y sollen die verwendeten Source-Dateien (.py oder .java) in dem Ordner blatt[y]/pflichtaufgabe[x]/ gespeichert werden.

Hausaufgabe 1 (Exceptions): **(6 Punkte)**

In dieser Aufgabe wollen wir einen Temperaturrechner implementieren. Zwei der üblichen Temperatureinheit sind dabei Grad Celsius (°C) und Kelvin (K). Dabei sind x K $= x - 273.15$ °C. Die niedrigsten Temperaturen sind -273.15 °C bzw. 0 K.

Wir definieren eine Klasse `Temperature`, welche als Attribute den Typ (°C, oder K) als String und den eigentlichen Temperaturwert als double enthält. Zusätzlich werden folgende Objektmethoden bereitgestellt:

`void convertToCelsius()` Rechnet die Temperatur in Celsius um.

`void convertToKelvin()` Rechnet die Temperatur in Kelvin um.

`void increaseTemperature(Temperatur diff)` Erhöht die Temperatur um diff. Wirft eine `IncompatibleTemperatureException`, falls diff nicht vom gleichen Typ ist, wie das Temperature-Objekt.

`void reduceTemperature(Temperatur diff)` Reduziert die Temperatur um diff. Wirft eine `IncompatibleTemperatureException`, falls diff nicht vom gleichen Typ ist, wie das Temperature-Objekt. Wirft eine `InvalidTemperatureException`, falls das Ergebnis eine Temperatur ergibt, welche nicht existieren kann (z.B. -7 K).

Temperature(double val, String type) Konstruktor der Temperatur-Klasse. Erzeugt ein Objekt mit Temperatur-Typ type und Temperatur-Wert val. Der Konstruktor wirft eine `InvalidTemperatureException`, falls für den Typ type ein ungültiger Wert übergeben wurde. Wirft außerdem eine `IncompatibleTemperatureException`, falls type nicht einem der zwei möglichen Temperaturtypen entspricht.

Die erwähnten Exceptions sollen dabei eigens implementierte **checked Exceptions** sein.

a) Implementiere die Klasse `Temperature` und die Exceptions im package `temperature`. Eigene Hilfsmethoden dürfen implementiert werden.

b) Implementiere zusätzlich eine toString-Methode.

c) Teste in einer Main-Methode (außerhalb des Packages) die oben genannten Funktionen, indem Temperature-Objekte erstellt werden und die Funktionen einmal fehlerfrei durchgeführt werden und jeweils alle möglichen Exceptions ausgelöst werden. Dabei sollen aussagekräftige Ausgaben auf der Konsole erfolgen. JUnit muss hier nicht genutzt werden.

Hausaufgabe 2 (Unit Tests): **(9 Punkte)**

In dieser Aufgabe wollen wir ein paar Methoden implementieren, welche wir später über C0- und C1-Tests in Verbindung mit JUnit testen.

a) Implementiere eine Utility-Klasse `StringUtils`, welche folgende Klassenmethoden bereitstellt. (Hinweis: Die String-API[1] kann hierfür hilfreich sein und darf genutzt werden. Beachte: Es dürfen keine Klassen importiert werden.)

`boolean isPalindrome(String text)` Überprüft, ob der Eingabestring ein Palindrom ist (vorwärts und rückwärts gelesen ergibt es das gleiche Wort).

`void sort(char[] text)` Sortiert ein Array von Zeichen über Seiteneffekte. (Als Hilfsfunktion für `areAnagrams` gedacht.)

`boolean areAnagrams(String str1, String str2)` Überprüft, ob str1 ein Anagramm von str2 ist. Ein Anagramm ist ein Wort, das durch Vertauschung von Buchstaben entsteht, z.B. ist *Ampel* ein Anagramm von *Palme*. Leerzeichen werden dabei ignoriert, und Großbuchstaben wie Kleinbuchstaben behandelt.

`boolean hasPrefix(String text, String prefix)` Überprüft, ob text das Präfix prefix besitzt (also mit diesen Zeichen beginnt). Beispielsweise ist *ent* ein Präfix von *entscheiden*, aber *apf* nicht von *appellieren*. Auch hier sollen alle Zeichen wieder als Kleinbuchstaben betrachtet.

`boolean hasSuffix(String text, String suffix)` Überprüft, ob text das Suffix suffix besitzt (also mit diesen Zeichen endet). Beispielsweise ist *en* ein Suffix von *beenden*, aber *en* nicht von *entscheidend*. Auch hier sollen alle Zeichen wieder als Kleinbuchstaben betrachtet.

Sorge in jeder Methode dafür, dass besondere Eingaben beachtet werden, bspw. wenn Strings `null` sind (dann kann bspw. einfach `false` zurückgegeben werden), Strings Leerzeichen enthalten, oder String-Längen nicht zueinander passen, etc.

b) Schreibe eine Testklasse `StringTest`, welche JUnit nutzt, um einen C0- und C1-Test für die genannten Methoden (außer `sort`) durchzuführen. (Hinweis: Je nach Implementierung stimmen C0- und C1-Test überein.)

c) Bei einem C2-Test werden alle möglichen Wege des Kontrollflussgraphen getestet. Da es bei Schleifen beliebig viele Iterationen geben kann, und somit auch beliebig zu testende Wege, macht es häufig Sinn, die Anzahl an Iterationen zu beschränken.

Erweitere die Testklasse um einen C2-Test für die Methode `isPalindrom` und `hasPrefix`. Die Anzahl an Iterationen in Schleifen soll dabei auf maximal zwei begrenzt sein.

(Hinweis: Falls JUnit[2] nicht im CLASSPATH hinterlegt ist, kann man dies lokal im selben Ordner hinzufügen.

Zum Kompilieren nutze den Befehl: `javac -cp junit-platform-console-standalone-1.10.1.jar:. [javafiles]`

Zum Ausführen nutze den Befehl:
`java -cp junit-platform-console-standalone-1.10.1.jar:.`
`org.junit.runner.JUnitCore [javatestfile]`)

[1] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html`

[2] Als Standalone bspw. hier herunterladbar: `https://repo1.maven.org/maven2/org/junit/platform/junit-platform-console-standalone/1.10.1/junit-platform-console-standalone-1.10.1.jar`