# Programmierpraktikum

Wintersemester 2021/22

*Prof. Dr. rer. nat. habil. Petra Hofstedt*
*Ilja Becker M. Sc., Sven Löffler M. Sc.*
*Sonja Breuß B. Sc., Carlo Bückert B. Sc., Endris Wenzke*

Brandenburgische
Technische Universität
Cottbus - Senftenberg

## Übungsblatt 11

Abgabedatum: 16.01.2022

Die Abgabe Ihrer Lösungen erfolgt vor Ablauf der Abgabefrist digital über die Moodle-Plattform. Erstellen Sie dazu ein PDF-Dokument, das die Lösungen Ihrer schriftlichen Aufgaben enthält. Laden Sie dieses PDF-Dokument und den erarbeiteten Java-Code (.java-Dateien) mit den in den Aufgaben vorgegebenen Namen bei Moodle hoch. Bitte laden Sie die Dateien einzeln hoch, Dateiarchive (z.B. .zip-Dateien) werden nicht akzeptiert.

Sie können maximal **(9 Punkte)** mit diesem Übungsblatt erreichen.

### Aufgabe 1 (Assertions)

2 Punkte

Eine Assertion ist eine Zusicherung über den Zustand eines Computer-Programms oder einer elektronischen Schaltung. Mit Hilfe von Zusicherungen können logische Fehler im Programm oder Defekte in der umgebenden Hard- oder Software erkannt und das Programm kontrolliert beendet werden.

Assertions werden ins Programm eingefügt und dann zur Laufzeit von der JVM (Java Virtual Machine) überwacht. Sie können explizit aktiviert (Entwicklungszeit/Testphase) oder stillgelegt (reguläre Anwendung) werden, ohne dass das Programm neu kompiliert werden muss.

1. Laden Sie sich die Vorlage `WeekdayService.java` von der Moodle-Seite herunter und binden Sie es in Ihr Projekt ein. Erweitern Sie die Klasse `WeekdayService` um eine Methode `public static List<String> getWeekStrings(String language)`. Die Methode sucht aus dem `Weekdaystore` das Array heraus, dass der übergebenen Sprache entspricht, und gibt alle im Array folgenden Elemente raus.

   `getWeekString("Deutsch")` ⇝ ∗ `["Montag", ..., "Sonntag"]`

   Sichern Sie mit einer einfachen Assertion ab, dass die zurückzugebende Liste nicht mehr als 7 Elemente enthält.

2. Schreiben Sie anschließend ein Programm `Assertions`, in deren `main`-Methode zwei Argumente aus den Kommandozeilenargumenten („String[] args"-Parameter der main-Methode) entnommen werden. Das erste Argument ist ein String und stellt eine Sprache dar, das zweite Argument ist auch ein String und stellt einen deutschen Wochentag dar. Schreiben Sie in ihrer `main`-Methode ein Programm zur Übersetzung, der zur eingegebenen Sprache mittels der vorher implementierten Methode vom `WeekdayService` die Liste übersetzter Wochentage abruft und folgendes ausgibt:

   `java Assertions <language> <weekday>` ⇝ ∗
   `"Der <weekday> heißt auf <language> <translated day>."`

`java Assertions Spanisch Montag` ↪ `* "Der Montag heißt auf Spanisch Lunes."`

Sichern Sie mit einer Assertion den Fall ab, dass der eingegebene Wochentag nicht bekannt (d.h. nicht Montag . . . Sonntag) ist[1]. Geben Sie als Assertiontext aus:

`<weekday> ist nicht bekannt.`

3. In Eclipse und in IDEs im Allgemeinen können Sie nicht nur Assertions als Kommandozeilenparameter an- und ausschalten, sondern dem Programm auch Eingaben mitgeben. Testen Sie nacheinander folgende Parameter mittels Run Configurations und beschreiben und begründen Sie die Ergebnisse:

| JVM Parameter | Programm Input | Ergebnis |
|---|---|---|
| | Spanisch Mittwoch | |
| | Utopiasprache Quatschtag | |
| -ea | Spanisch Quatschtag | |
| -ea:assertions.WeekdayService | Utopiasprache Montag | |

4. Nennen Sie zwei Anwendungsfälle in denen man in der Entwicklung mit Java Assertions nutzt. Nennen Sie außerdem zwei Anwendungsfälle in denen man statt Assertions lieber Exceptions nutzen sollte.

**Aufgabe 2 (Exceptions)** 3 Punkte

`Exceptions` sind ein Sprachmittel zum Umgang mit Fehlersituationen. Grundsätzlich gibt es zwei Möglichkeiten mit `Exceptions` umzugehen. Die erste Möglichkeit ist, diese mit einem `try-catch`-Block abzufangen und entsprechend darauf zu reagieren. Die andere Möglichkeit ist es die auftretende `Exception` an die zuvor aufrufende Methode mittels `throws` durch-/hochzureichen.

1. Schreiben Sie eine Klasse `IOString`, die zwei Methoden `schreibeInDatei` und `leseAusDatei` enthalten soll. Die `schreibeInDatei`-Methode soll dabei zwei Strings als Eingabe erhalten. Der erste String soll den Text angeben, der in die Datei geschrieben werden soll und der zweite soll den Namen der Datei angeben.
   Die `leseAusDatei`-Methode soll einen Dateinamen als String übergeben bekommen und den Text, der in der Datei steht, als String zurückgeben.

   Verwenden Sie für das Schreiben in Dateien sowohl einen `FileWriter` als auch einen `BufferedWriter` und analog dazu beim Lesen aus Dateien einen `FileReader` und einen `BufferedReader`. Die beiden Methoden sollen Fehlermeldungen mittels `throws` an die aurufende Methode übergeben.

2. Schreiben Sie mit `JavaFX` eine GUI, die zwei `Buttons`, ein `TextField`, ein `TextArea` und ein `Label` enthält.

3. Implementieren Sie die Aktionen der `Button`-Klicks mit Lambdas derart, dass durch Klick auf den einen `Button` der Text des `TextAreas` ausgelesen und in eine Datei mit den Namen, des im `TextField` stehenden Textes, geschrieben wird. Analog dazu soll der andere `Button` den Text aus einer Datei auslesen und in das `TextArea` schreiben.
   Auftretende Fehlermeldungen sollen abgefangen und in dem `Label` ausgegeben werden.

[1] **Anmerkung** Es muss erwähnt werden, dass Assertions in der Regel nicht dazu genutzt werden um Benutzereingaben zu kontrollieren, dies dient hier der Aufgabenstellung. Für die Überprüfung von Nutzereingaben und Parametern in öffentlich zugänglichen Methoden nutzt man i.A. Exceptions.

**Aufgabe 3 (Umgekehrt Polnische Notation)** 4 Punkte

Die sogenannte umgekehrt polnische Notation (UPN) ist eine Schreibweise z. B. für arithmetische Ausdrücke, die sich besonders leicht automatisiert auswerten lässt. Insbesondere müssen keine Vorrangregeln für Operatoren berücksichtigt werden, auch Klammerungen sind überflüssig.

Dies wird erreicht, indem die Operanden – insbesondere dann, wenn diese selbst Ausdrücke sind – stets vor dem zugehörigen Operator geschrieben werden. Im Vergleich zur herkömmlichen Infix-Schreibweise ergibt sich zum Beispiel:

| Infix-Notation | UPN |
|---|---|
| 5*3+2 | 5 3 * 2 + |
| 5*(3-2) | 5 3 2 - * |
| (5-2) * (3+4) | 5 2 - 3 4 + * |

Zur Auswertung wird ein Stapelspeicher (engl. *stack*) benötigt. Der Ausdruck in UPN wird schrittweise von links nach rechts abgearbeitet. Dabei werden Zahlen auf den Stapel gelegt, bei Operatoren wird die benötigte Anzahl von Operanden vom Stapel genommen und das Ergebnis der Operation wieder auf dem Stapel abgelegt.

1. Implementieren Sie basierend auf dem `Listen`-Datentyp das folgende Interface `Stack` als Klasse `ListStack`:

```
public interface Stack {
  public void push(Object obj);
  public Object peek();
  public Object pop();
  public boolean isEmpty();
}
```

   Die aufgeführten Methoden haben die folgenden Bedeutungen:

   - Die Methode `push(Object obj)` legt das Objekt obj oben auf dem Stapel ab.
   - Die Methode `peek()` liefert das oben auf dem Stapel liegende Objekt zurück.
   - Die Methode `pop()` liefert das oben auf dem Stapel liegende Objekt zurück und entfernt es dabei vom Stapel.
   - Die Methode `isEmpty()` prüft, ob der Stapel leer ist.

2. Implementieren Sie geeignete Klassen `Operand.java` und `Operator.java`.
3. Legen Sie eine `Main`-Klasse an und formulieren Sie die im obigen Beispiel genannten Ausdrücke in UPN, indem Sie einen Stack von Objekten anlegen, und diese in UPN-Reihenfolge mit `Operanden` und `Operatoren` füllen.
4. Schreiben Sie eine Klasse `UpnAuswerter.java`, welche Ausdrücke in UPN auswerten und das Ergebnis ausgeben kann. Testen Sie Ihre `UpnAuswerter`-Klasse mit den Ausdrücken aus der vorigen Teilaufgabe.