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 10

Abgabedatum: 09.01.2021

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 **(7 Punkte)** mit diesem Übungsblatt erreichen.

Aufgabe 1 (Schnittstellen)

2 Punkte

Java erlaubt nur Einfachvererbung von Klassen. Das bedeutet eine Klasse kann nur genau von einer anderen Klasse erben. Sie ist also eine Spezialisierung der Oberklasse. Dies kann zwar sequentiell wiederholt werden, es kann also z. B. Klasse1 von Klasse2 und Klasse2 von Klasse3 erben, allerdings kann eine Klasse nicht gleichzeitig von zwei verschiedenen Klassen erben. Eine Klasse kann aber eine beliebige Anzahl Schnittstellen implementieren und auf diese Weise gleichzeitig Fähigkeiten aus verschiedenen Bereichen erhalten.

Lesen Sie sich die Abschnitte 5.13.1 - 5.13.6 aus dem Buch „Java ist auch eine Insel“[1] zum Thema Schnittstellen durch.

1. Erstellen Sie eine Klasse `KioskInhalt.java`. Jeder `KioskInhalt` soll über einen Namen verfügen, der über eine Methode `getName` abgerufen werden können soll.
2. Erstellen Sie weiter eine Klasse `Kiosk.java` die wiederum über einen Namen verfügt und zusätzlich eine Liste von KioskInhalten beinhaltet.
3. Legen Sie zwei Interfaces `Elektronisch.java` und `Kaufbar.java` an. `Elektronisch.java` soll über eine Methode `getStromverbrauch()` und `Kaufbar.java` über zwei Methoden `getPreis()` und `setPreis(double preis)` verfügen.
4. Erstellen Sie die Klassen `Zeitung.java`, `Kasse.java` und `EBookReader.java`. Zeitungen und EBookReader stellen Kioskinhalte dar und sind verkaufbar. Kassen und EBookReader sollen zudem elektronisch sein.

   Implementieren Sie die benötigten Methoden derart, dass eine Kasse einen Stromverbrauch von 0,5 und ein EBook einen Stromverbrauch von 0,25 (Kilowattstunden) hat. Des Weiteren soll eine Zeitung 50 Cent kosten und ein EBookReader 55 Euro.

[1] `http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_05_013.htm`

5. Überschreiben Sie die `toString`-Methode der Klasse `Kiosk.java` derart, dass diese den Namen des Kiosks und all seiner Objekte mit deren Preisen und Stromverbrauch ausgibt, falls diese über solche verfügen. Schreiben Sie eine `Main.java`-Klasse mit einer `main`-Methode, welche einen neuen Kiosk erzeugt, der eine Zeitung, eine Kasse und einen EBookReader als Objekte enthält und diese mittels der `toString`-Methode des Kiosks in der Konsole ausgibt. Eine mögliche Ausgabe könnte z. B. wie folgt aussehen:

```
Svens Kiosk Inventar:

Svens Kasse, Stromverbrauch: 0.5
Käseblatt, Preis: 0.5
EBookReader, Preis: 55.0, Stromverbrauch: 0.25
```

Aufgabe 2 (GUI) 3 Punkte

Es existieren verschiedene Java-Bibliotheken um eine grafische Nutzerschnittstelle (Graphical User Interface - GUI) zu erzeugen. In dieser Aufgabe soll mit Hilfe der `JavaFX`-Bibliothek eine primitive GUI erstellt werden.

Voraussetzung Bevor Sie die `JavaFX`-Bibliothek nutzen können, müssen Sie sicherstellen, dass die Bibliothek als *Dependency* für ihr Projekt vorhanden ist. Zwei Möglichkeiten hierfür finden Sie im Anhang.

1. Legen Sie eine Klasse `GUI.java` an, die von der `Application`-Klasse erbt. Schreiben Sie in dieser Klasse eine `main`-Methode die als einzigen Code

   ```
   launch(args);
   ```

   beinhaltet. Überschreiben Sie des Weiteren die `start`-Methode der `Application`-Klasse wie folgt

   ```
   @Override
   public void start(Stage primaryStage) throws Exception {
           primaryStage.setTitle("GUI");

           // weiterer Code

           primaryStage.show();
   }
   ```

   Ihre Anwendung erzeugt jetzt bereits ein leeres Grafikfenster.

   Erzeugen Sie ein neues `VBox`-Objekt, zu diesem Objekt können mehrere Grafikkomponenten hinzugefügt werden, die dann in dem Fenster untereinander angezeigt werden. Dieses können wir der `primaryStage` allerdings nicht direkt übergeben. Wir müssen zunächst eine neue `Scene` erzeugen, der wir im Konstruktor das `VBox`-Objekt übergeben. Als weitere Übergabeparameter können die Breite und die Höhe des Fensters übergeben werden. Setzen Sie das Attribut `Scene` der `primaryStage` auf den Wert der von Ihnen erzeugten `Scene`. Beachten Sie, dass `primaryStage.show();` am Ende der `start`-Methode ausgeführt werden soll.

2. Erzeugen Sie ein `Label`, zwei `Buttons` und ein `TextField` und fügen Sie diese mittels der `getChildren().add`-Methode der zuvor erzeugten `VBox` hinzu. Das `Label` soll dabei den Text „0 Klicks“, der erste `Button` den Text „Klick“, der zweite `Button` den Text „Setze Klicks“ und das `TextField` den Text „5“ zugewiesen bekommen.

3. Legen Sie eine neue Klasse `ButtonClickHandler` an, die das Interface `EventHandler<ActionEvent>` implementiert. Überschreiben Sie die `handle(ActionEvent event)`-Methode zunächst ohne dort Code einzufügen. In Ihrer `GUI`-Klasse können Sie nun ein Objekt der Klasse `ButtonClickHandler` anlegen und dieses mit der `setOnAction(EventHandler<ActionEvent>)`-Methode an den ersten `Button` übergeben. Die zuvor überschriebene `handle(ActionEvent event)`-Methode wird dadurch genau dann ausgeführt, wenn der `Button` geklickt wird.

   Bearbeiten Sie die `ButtonClickHandler` Klasse so, dass ein Klick auf den `Button` dafür sorgt, dass das zuvor erzeugte `Label` die Anzahl der Klicks auf den Button anzeigt. Falls Sie dazu weitere Funktionen oder Konstruktoren benötigen, dann legen Sie diese an.

4. Die Funktionalität des Buttons soll mit einem sogenannten *Lambda*-Ausdrucks implementiert werden. Rufen sie hierfür auf ihrem zweiten Button

   ```
   meinZweiterButton.setOnAction(actionEvent -> {
       // ihre Implementierung
   });
   ```

   auf. Anstelle von `// ihre Implementierung` können sie nun das Verhalten des Buttons implementieren.

   Wenn der zweite `Button` geklickt wird, soll der Text des zuvor erstellten `TextField`s ausgelesen werden. Es soll versucht werden den Text in einen Integer zu konvertieren und wenn dies gelingt, soll die Anzahl der Klicks auf diesen Wert gesetzt werden. Zum konvertieren kann die `Integer.parseInt(String s)`-Methode genutzt werden. Bedenken Sie, dass die Methode eine Exception wirft, wenn der Input nicht korrekt ist. Fangen Sie diesen Fall ab, und behandeln Sie ihn entsprechend.

5. Erzeugen Sie in der `GUI`-Klasse ein `Canvas`-Objekt der Größe 400 mal 200 Pixel und fügen Sie es der `VBox` hinzu. Lesen Sie vor dem Hinzufügen zur `VBox` den `GraphicsContext` des `Canvas`-Objekts mit der `getGraphicsContext2D()`-Methode aus. Mit `fill...`- und `stroke...`-Methoden können Sie dann auf dem `GraphicsContext`-Objekt zeichnen. Zeichnen Sie auf diese Weise einen Kreis.

6. Ändern Sie das Programm so ab, dass in das `Canvas` immer genau so viele Kreise gezeichnet werden, wie der Aktuelle Klickstand ist. Füllen Sie die Kreise dabei in unterschiedlichen Farben aus.

Aufgabe 3 (Kassenzettel) 2 Punkte

Es soll ein Programm zum Erstellen von Kassenzetteln geschrieben werden. Zu jedem Einkauf wird ein Kassenzettel erstellt. Ein Kassenzettel enthält mehrere Einträge sowie eine Gesamtsumme. Ein Eintrag besteht aus dem Namen eines Produkts, der gekauften Menge sowie dem Preis.

1. Fertigen Sie eine geeignete objektorientierte Modellierung für den Kassenzettel und dessen Einträge an. Erstellen Sie dazu die beiden Klassen `Kassenzettel.java` und `Eintrag.java`. Legen Sie nur Attribute mit dem Modifikator `private` an und erstellen Sie gegebenenfalls geeignete `setter`- und `getter`-Methoden um aus anderen Klassen auf diese zugreifen zu können. Ein Kassenzettel soll über eine Liste von Einträgen verfügen.

2. Überschreiben Sie die `toString()`-Methoden der `Kassenzettel`- und der `Eintrag`-Klasse so, dass die `toString()`-Methode bei einem Eintrag den Namen des Produkts, die Menge des gekauften Produkts und dessen Preis als String zurückliefert. Die `toString()`-Methode des

Kassenzettels soll alle Einträg auflisten und den Gesamtpreis als String zurückgeben. Eine Ausgabe könnte z. B. wie folgt aussehen.

```
-------------------------
   Programmierpraktikum
      Einkaufsliste
-------------------------
Toast
  1x                1.49
Salami 200g
  2x                1.29
Saftschinken 200g
  2x                1.39
Sandwichkäse 200g
  4x                0.99
                    =====
Summe EUR           10.81
                    =====
```

3. Schreiben Sie eine Klasse `Main.java` die ein Objekt `Kassenzettel` erzeugt und den Nutzer solange nach den gekauften Produkten, deren Anzahl und den Preis fragt, bis dieser mit der Eingabe „x“ symbolisiert, dass der Kassenzettel vollständig ist. Anschließend soll der Kassenzettel ansprechend formatiert ausgegeben werden.

JavaFX Dependency

Es gibt verschiedene Möglichkeiten Abhängigkeiten - d.h. Code-Bibliotheken u.ä., die zum Compilieren und Ausführen des eigenen Codes nötig sind - zu verwalten. Häufig wird diese Aufgabe vom *Build-System* übernommen.

Build System Aufgrund der Komplexität heutiger Software, wird der Prozess des Kompilierens von Software heutzutage häufig von einem sogenannten Build System übernommen. Je nach Ausführung übernimmt das Build System mindestens die Aufgabe, alle nötigen Quelldateien zu kompilieren und zu einem ausführbaren Programm zusammenzufügen. Häufig wird auch das Verwalten von Abhängigkeiten von Build-System übernommen.

Wir haben bisher (wenn auch vllt. unbewusst) das integrierte Build-System von Eclipse genutzt. Alternativ wurde zu Beginn des Moduls auch erläutert, wie man ein Programm von der Kommandozeile aus kompiliert und ausführt. Die bisher benötigten Code-Bibliotheken waren alle in der sogenannten `Java Runtime Environment`, den Standard-Bibliotheken Java's, gegeben. Weiterer Code wurde direkt als Quellcode vorgegeben.

JavaFX wiederum ist nicht (mehr) Teil der `JRE`. Folgend sollen zwei Möglichkeiten beschrieben werden, wie JavaFX als Abhängigkeit zu einem Projekt hinzugefügt werden kann. Eine nutzt das integrierte Build-System von Eclipse, das andere nutzt das weit verbreitete Build-System `Maven`[2].

Als Bibliothek in Eclipse einfügen

1. Laden Sie unter `https://openjfx.io/` das für ihr System passende JavaFX SDK herunter. **Achtung:** Sie müssen ein Häkchen bei „Include older versions“setzen und bei „JavaFX version“die 11 wählen.

[2]Ggf. bekannt aus dem Modul *Entwicklung von Softwaresysteme*.

Entpacken Sie das Archiv an eine Stelle, an der Sie die Dateien für die Zeit ihres Software-Projekts liegen lassen können (z.B. ihren Eclipse-Workspace).

2. Erstellen Sie in Eclipse ein neues Java-Projekt. Nachdem Sie im sich öffenen Dialogfenster einen Namen festgelegt haben, klicken Sie auf *Weiter/Next* und setzen Sie ein Häkchen bei *`module-info.java`-Datei erstellen*. Ausführliche Informationen zum Umgang mit und Nutzen der `module-info.java`-Datei finden Sie hier `https://javabeginners.de/Grundlagen/Module_verwenden.php`.
3. Nachdem Sie ihr Projekt erstellt haben, rechtsklicken Sie auf ihr neu erstelltes Projekt im Package Explorer. Navigieren Sie `Build-Path.../Build-Path konfigurieren`. Navigieren Sie in den Library-Reiter/Tab und klicken Sie auf *Modulepath*. Klicken Sie auf *Externe JARs hinzufügen* und navigieren Sie in den Ordner, in den Sie das Archiv entpackt haben. Wählen Sie alle im Unterordner `lib` vorhandenden `*.jar`-Dateien aus und bestätigen Sie.
4. Überprüfen Sie die Funktionalität, indem Sie eine JavaFX-`Application` basierte Klasse erstellen und ausführen.
   a) Erstellen Sie ein neues Package in ihrem Projekt.
   b) Erstellen Sie eine neue Klasse in ihrem Projekt.
   c) Die Klasse soll von `javafx.application.Application` erben und `public void start(Stage arg0)` implementieren.
   d) Setzen Sie ggf. einen Stage-Titel und sorgen Sie dafür, dass das Fenster angezeigt wird (`show()`).
   e) Stellen Sie sicher, dass in ihrer `module-info.java` folgende Zeilen stehen:

```
exports ihrPackageName;

requires javafx.graphics;
requires javafx.controls;
requires javafx.base;
```

   f) Sollten Konstruktornamen immer noch rot unterstrichen sein, dann gehen Sie mit der Maus über einen solchen Konstruktor. Wird Ihnen als Lösungsvorschlag eine Verschiebung vom „Classpath“ in den „Modulepath“ aufgeführt, dann tun Sie dies für alle entsprechenden Stellen.
   g) Nun sollten Sie ihr Programm wie üblich mit dem grünen Run-Button ausführen können.

Mit dem Build-System Maven

1. Installieren Sie auf ihrem System das Build-System *Apache Maven*. Überprüfen Sie ihre Installation, indem Sie auf der Kommandozeile `mvn -version` aufrufen. Als Ausgabe sollten Details zu ihrer Maven-Installation folgen.
2. Erstellen Sie ein neues Java-Projekt in Eclipse. Entfernen Sie entweder das Häkchen bei *`module-info.java`-Datei erstellen*, oder stellen Sie später sicher, dass der Inhalt der `module-info.java` korrekt ist.
3. Rechtsklicken Sie ihr neu erstelltes Java-Projekt, navigieren Sie zum Menüpunkt *Konvertieren-> Zu Maven-Projekt konvertieren*. Führen Sie die Konvertierung durch. Ihr Projektorder sollte im Ordner-Symbol neben dem kleinen **J** ein kleines **M** aufweisen. Des Weiteren sollte in ihren

Projektorder eine `pom.xml`-Datei hinzugekommen sein. Diese enthält die Konfiguration ihres Maven-Projekts.

4. Fügen Sie oberhalb von <build> folgende Zeilen ein:

```
<properties>
    <project.launcherClass>${mainClass}</project.launcherClass>
</properties>
```

5. Fügen Sie folgendes Plugin unter dem letzten </plugin> hinzu:

```
<plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>1.6.0</version>
        <executions>
                <execution>
                        <goals>
                                <goal>java</goal>
                        </goals>
                </execution>
        </executions>
        <configuration>
                <executable>maven</executable>
                <mainClass>${project.launcherClass}</mainClass>
        </configuration>
</plugin>
```

6. Zu guter Letzt fügen Sie die `JavaFX`-Dependency unterhalb von </build> hinzu:

```
<dependencies>
    <dependency>
      <groupId>org.openjfx</groupId>
      <artifactId>javafx-controls</artifactId>
      <version>11</version>
    </dependency>
</dependencies>
```

7. Öffnen Sie nun ihre Kommandozeile und navigieren Sie in den Projektordner in ihrem Eclipse-Workspace. Führen Sie den Befehl `mvn validate aus`.

8. Überprüfen Sie die Funktionalität, indem Sie eine JavaFX-`Application` basierte Klasse erstellen und ausführen.

   a) Erstellen Sie ein neues Package in ihrem Projekt.

   b) Erstellen Sie eine neue Klasse in ihrem Projekt.

   c) Die Klasse soll von `javafx.application.Application` erben und `public void start(Stage arg0)` implementieren.

   d) Setzen Sie ggf. einen Stage-Titel und sorgen Sie dafür, dass das Fenster angezeigt wird (`show()`).

9. Sollten Sie sich entschieden haben, eine `module-info.java` erstellen zu lassen, stellen Sie sicher, dass die Zeilen

```
exports ihrPackageName;

requires javafx.graphics;
```

enthalten sind. Sie können das Projekt jetzt mit

```
mvn clean compile exec:java -DmainClass=ihrPackageName.IhreKlasse
```

kompilieren und ausführen.