Technische Hochschule Ingolstadt
Fakultät für Elektrotechnik und Informatik

Praktikum „WaWi“
Vorführaufgabe 3

Behandelte Themen

Exceptions, Java FX

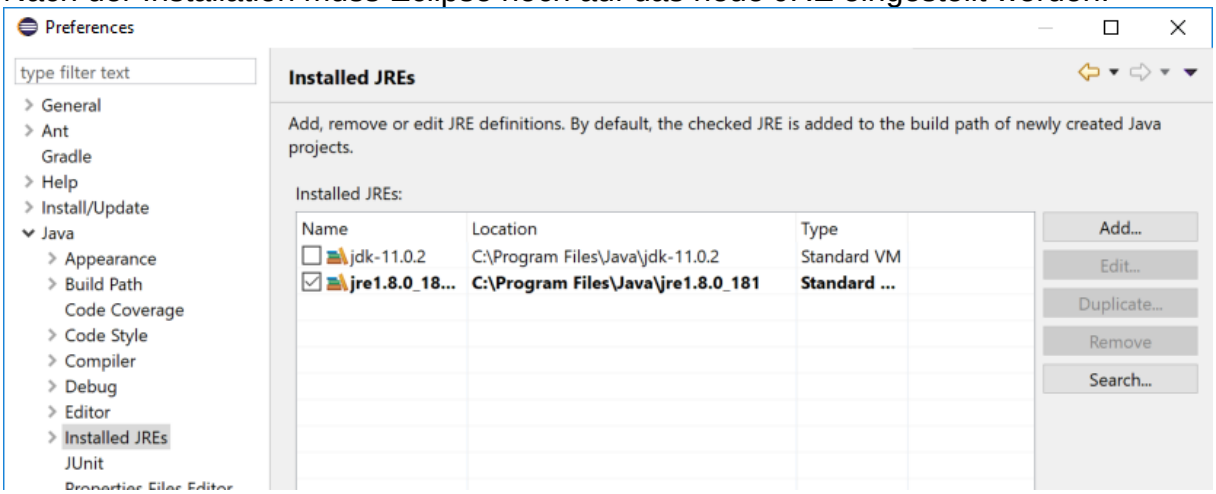
Vorbereitung

Kopieren Sie bitte Ihr bisheriges Eclipse-Projekt und benennen Sie die Kopie „`WaWi03`“ (nutzen Sie hierzu einfaches „copy and paste“ im „Package Explorer“ von Eclipse) und erweitern Sie die Inhalte des kopierten Projekts entsprechend nachfolgender Aufgaben.

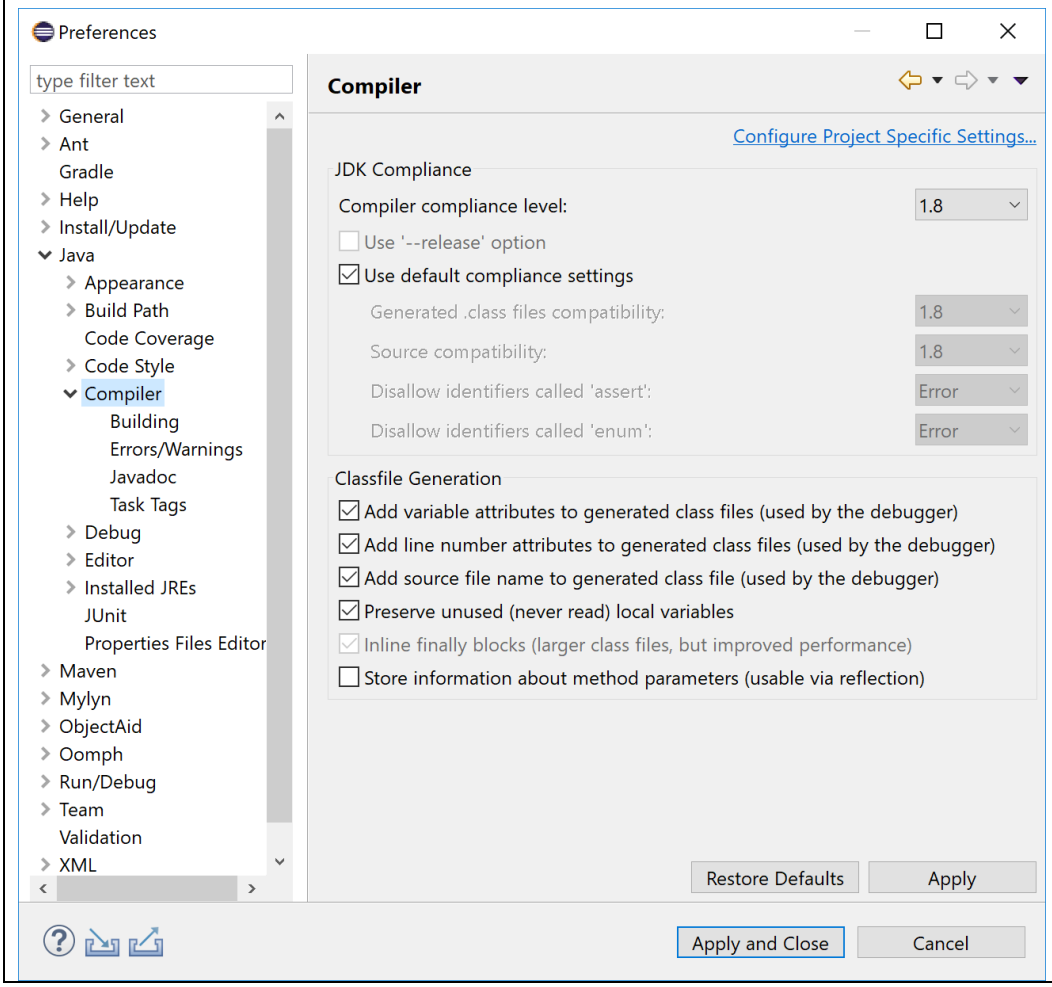
Für diese Vorführaufgabe steht ein neues Archiv „`cert.zip`“ (siehe Moodle-Kursseite zum Praktikum) zur Verfügung. Ersetzen Sie den Inhalt des Source-Folders „`cert`“ durch den Inhalt des neuen Archivs. Danach erhalten Sie eine Test-Klasse zum Test der Aufgabe 1 in diesem Ordner.

Wichtiger Hinweis zu JavaFX: JavaFX wurde ab JDK 11 ausgelagert. Verwenden Sie bitte ein JRE der Version 8 (hier ist das JavaFX dabei!), siehe z.B. https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html.

Nach der Installation muss Eclipse noch auf das neue JRE eingestellt werden:



Wichtig ist auch, dass der Compiler Compliance Level auf 1.8 eingestellt ist:



Aufgabe 1 – Exceptions

Bisher wurden auftretende Fehler (z. B. nicht ausreichender Lagerbestand beim Auslagern) nicht behandelt. Dies soll nun durch Einsatz von zwei Exception-Klassen verbessert werden. Legen Sie vorbereitend hierfür das Package „`exceptions`“ an.

Die Klasse `BookingException` stellt eine Möglichkeit zur Behandlung von Buchungsfehlern zur Verfügung.

- Eine BookingException ist eine Exception.
- Ein Objekt der Klasse BookingException kann nicht ohne eine Fehlermeldung angelegt werden.

Die Klasse `OutOfStockException` soll einen nicht ausreichenden Lagerbestand beim Auslagern anzeigen.

- Eine `OutOfStockException` ist eine Exception.
- Eine `OutOfStockException` hat ein Produkt.
- Ein Objekt der Klasse `OutOfStockException` kann nicht ohne ein Produkt und eine Fehlermeldung angelegt werden.

Die beiden oben beschriebenen Klassen sollen folgendermaßen verwendet werden:

- Beim Auslagern von Artikeln sollte eine `OutOfStockException` geworfen werden, wenn der Lagerbestand nicht ausreicht.

- Beim Hinzufügen einer Rechnungsposition zu einer Rechnung sollte eine `OutOfStockException` geworfen werden, wenn der Lagerbestand nicht ausreicht. Bitte beachten Sie, dass dies nur für Artikel gilt!
- Beim Hinzufügen einer Rechnungsposition zu einer Rechnung bzw. beim Buchen einer Rechnung sollte eine `BookingException` geworfen werden, wenn die Rechnung nicht den Status `IN_ERSTELLUNG` hat.
- Die geworfenen Exceptions sollten im Bereich der GUI (s. Aufgabe 2) gefangen werden, im Bereich der Geschäftsobjekte (z. B. beim Buchen einer Rechnung) werden Exceptions einfach weiter geworfen. Ergänzen Sie entsprechende `throws`-Klauseln in den Methodenköpfen.

Aufgabe 2 – Entwicklung der Grafischen Benutzeroberfläche

Vorbereitung

- Package `gui`: Erstellen Sie ein Package namens „gui“ und befüllen Sie es aus dem entsprechenden Archiv im Kursraum (Klasse `WaWiAppFragment`). Benennen Sie die Klasse `WaWiAppFragment` in `WaWiApp` um. Die Klasse `WaWiApp` ist die Startklasse für die Anwendung, d.h. sie enthält eine main-Methode. `WaWiApp` startet Dialoge durch Instanziierung von Dialog-Klassen (hier: `POSDialog`) und führt die Ereignisbehandlung durch (z.B. Reaktion auf Button-Klick).
- Package `gui.posDialog`: Erstellen Sie ein Package namens „`gui.posDialog`“ und befüllen Sie es aus dem entsprechenden Archiv im Kursraum. Dieses Package enthält alle Klassen und Interfaces für die Realisierung des POS-Dialogs. Benennen Sie die Klasse `POSDialogFragment` in `POSDialog` um. Der POS-Dialog (siehe Abbildung 1) ermöglicht die Auswahl eines Produkts (Artikel oder Dienstleistung) über spezielle Buttons. Jeder dieser Buttons ist eine Instanz der vorgegebenen Klasse `ProduktButton`. Die Interfaces werden für die Behandlung der 4 Ereignisse benötigt und werden im Quellcode erläutert.
- Die Bilddateien für die Produktbuttons befinden sich in dem Archiv „`pic.zip`“ (siehe Moodle-Kursraum). Entpacken Sie dieses Archiv in ihr Projekt unter dem neu anzulegenden Folder namens „`pic`“ unter „src“.

Funktionsweise des Dialogs

Hauptziel dieser Aufgabe ist die Definition einer Klasse namens `POSDialog`. `POSDialog` realisiert den Dialog für POS (englisch *Point of Sale*) und bietet eine Benutzeroberfläche für die Auswahl der abzurechnenden Produkte und die schlussendliche Abrechnung („Checkout“).

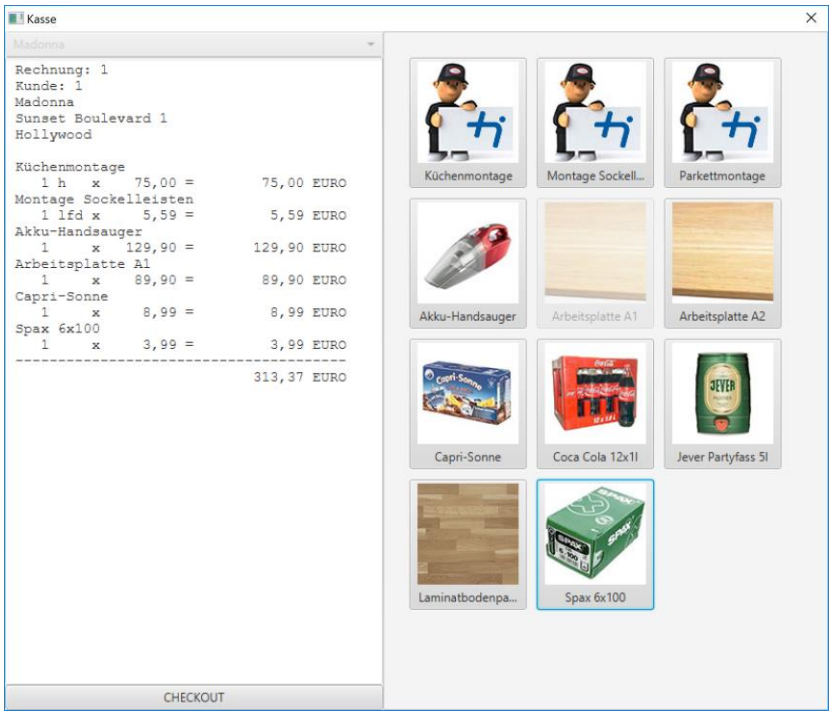


Abbildung 1: Kassendialog im Produktauswahlmodus

`Der POS-Dialog` befindet sich stets in einem von zwei Modi: der *Kundenauswahl* (siehe Abbildung 2) oder der *Produktauswahl* (siehe Abbildung 1).

Der gesamte Ablauf wird im Folgenden beschrieben:

- Nach dem Erzeugen des Dialogs erscheint dieser im Modus *Kundenauswahl*. Während der *Kundenauswahl* ist nur die Auswahl des Barverkaufs (Eintrag „Barverkauf“) oder eines Kunden (Eintrag ist der jeweilige Kundenname) über die Choicebox rechts oben möglich, d. h. die Produktbuttons sowie der Checkout-Button sind gesperrt. Der erste Eintrag der Choicebox lautet „<bitte auswählen>“ und ist im Kundenauswahlmodus zunächst selektiert.
- Mit Auswahl eines Eintrags ungleich „<bitte auswählen>“ wird ein neues Rechnungsobjekt angelegt, welchem der ausgewählte Kunde übergeben wird. Im Falle eines Barverkaufs wir das Rechnungsobjekt ohne Parameter erzeugt. Die noch leere Rechnung wird im Textfeld dargestellt und sämtliche Buttons, für deren Produkte ein Lagerbestand vorhanden ist, werden entsperrt (gilt nur für Artikel!). Die Kundenauswahl-Choicebox wird gesperrt.
  Hinweis: beim Aufbau der TextArea für die Rechnung sollten Sie die Schriftart setzen, um die Rechnung korrekt formatiert anzuzeigen, etwa so:
  ```
  rechnungTextArea.setFont(new Font("Courier New", 14));
  ```
- Wird ein Produktbutton gedrückt, so wird das gewählte Produkt als neue Rechnungsposition hinzugefügt bzw. in eine bestehende entsprechend erweitert. Nutzen Sie hierzu die entsprechende Methode der Klasse Rechnung. Anschließend wird die Rechnungsanzeige im Textfeld aktualisiert. Für den Fall, dass der Lagerbestand eines Artikels durch die Aktion auf 0 reduziert wurde, wird der Produktbutton gesperrt.
- Wird der Checkout-Button gedrückt, so wird die aktuelle Rechnung gebucht. Anschließend werden alle Produktbuttons und der Checkout-Button gesperrt. Die Kundenauswahl-Choicebox zeigt auf den ersten Eintrag („<bitte auswählen>“) und wird entsperrt. Die Anwendung befindet sich damit wieder im Modus *Kundenauswahl*.
- Über den Schließen-Systembutton („X-Button“ in der rechten oberen Ecke, siehe Abbildung 1) kann die Anwendung jederzeit beendet werden.

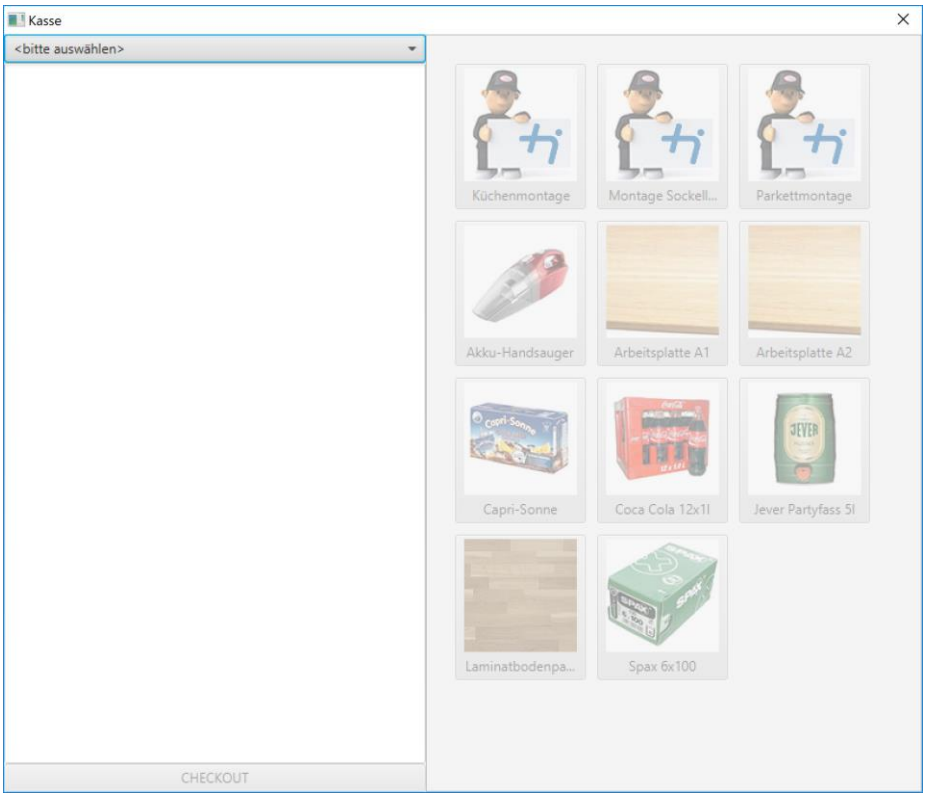


Abbildung 2: Kassendialog im Kundenauswahlmodus

Erläuterungen zum Zusammenwirken der Klassen WaWiApp und POSDialog

Die **Klasse WaWiApp** hat die Aufgabe, eine Instanz von POSDialog zu erzeugen und die Behandlung der Ereignisse des Dialogs vorzunehmen. Man trennt die Darstellung von Daten häufig von der Verarbeitungslogik, um eine Unabhängigkeit der Verarbeitungslogik von der jeweils eingesetzten UI-Technologie (hier: JavaFX) zu erreichen. Der Vorteil dieser Trennung liegt in der einfachen Austauschbarkeit der UI-Technologie sowie in einer besseren Wartbarkeit. Fehler in der Verarbeitungslogik können nicht vermischt werden mit Fehlern in der Darstellung der Daten (hier: mit JavaFX).

Aufgabe der **Klasse POSDialog** ist es, den POS-Dialog anzuzeigen und Eingaben zu erlauben (je nach Modus). Die Verarbeitung der Eingaben sowie die Behandlung weiterer Ereignisse wird jedoch aufgrund der oben erklärten Modularisierung an eine WaWiApp-Instanz delegiert. Ein Beispiel für diese Delegation ist weiter unten unter Punkt 4. zu sehen.

Die **Delegation** wird technisch über **Lambda-Ausdrücke** vorgenommen, d.h. die App-Instanz definiert einen Lambda-Audruck zur Behandlung eines bestimmten Ereignisses und macht den Ausdruck beim Dialog bekannt (per Setter-Aufruf).

Ein **Lambda-Ausdruck** entspricht einer Methode ohne Namen („anonyme Methode“), die ein sogenanntes Funktionales Interfaces (engl. *functional interface*) implementiert. Ein Funktionales Interface ist ein Interface mit genau einer Methode (vgl. Annotation `@FunctionalInterface`). Man ruft ein durch eine Referenz gegebenes Lambda auf, indem man die Interface-Methode auf der Referenz aufruft. Im Beispiel unten ist dies die Anweisung

```
closeHandler.handleCloseEvent();
```

Die Funktionalen Interfaces für die vier Ereignisse sind im Package `gui.posDialog` definiert:

- Interface `POSDialogCheckoutHandler`: Reaktion auf Klick des Checkout-Buttons
- Interface `POSDialogCloseHandler`: Reaktion auf Klick des Dialog-Schließen-Buttons
- Interface `POSDialogKundenauswahlHandler`: Reaktion auf Auswahl eines Kunden in der Kunden-KomboBox
- Interface `POSDialogProduktButtonHandler`: Reaktion auf Klick eines Produkt-Buttons

Empfohlene Vorgehensweise

1. Arbeiten Sie sich in die gegebenen Quellcodes ein und versuchen Sie, die Erläuterungen (s. Kommentare) nachzuvollziehen.
2. Bauen Sie nun schrittweise die Methode `initDialog()` der Klasse `POSDialog` auf, bis der Dialog wie in den Abbildungen dargestellt wird.
3. Ergänzen Sie vorbereitend für Schritt 4. entsprechende Getter/Setter-Methoden für die Attribute von POSDialog.
4. Entwickeln Sie schrittweise die Ereignisbehandlung für die Ereignisse „Dialog schließen“ , „Kunde ausgewählt“, „Produkt-Button geklickt“, und „Checkout-Button geklickt“, indem Sie die Lambdas in `WaWiApp` definieren und als Ereignisbehandlungs-Methode setzen.
   Achtung: Ihre Ereignisbehandlung in `POSDialog` muss das für das jeweilige Ereignis gesetzte Lambda aufrufen und nicht etwa das Ereignis selbst behandeln.

   Beispiel: Ereignis „Dialog schliessen“ nach Klick auf den Schliessen-Button

```
In WaWiApp.showPOSDialog():
...
POSDialogCloseHandler closeHandler = () -> {
    posDialog.hide();
};
posDialog.setCloseHandler(closeHandler);
posDialog.show();
...
```

```
In POSDialog.initDialog():
…
stage.setOnCloseRequest(e -> {
    closeHandler.handleCloseEvent(); // Lambda-Aufruf
});
…
```

   In der Methode `showPOSDialog()` wird durch Aufruf der Methode `show()` der `POSDialog` angezeigt. Vorbereitend wird eine Handler-Methode als Lambda-Ausdruck definiert. Die Handler-Methode behandelt das Ereignis „Dialog-Schließen“, welches durch Klick auf den Schließen-Button (Kreuz-Button) des Dialogs erzeugt wird. Per Setter-Aufruf wird das Lambda dem Dialog bekannt gemacht.
   In der Methode `initDialog()` des `POSDialogs` wird eine Ereignisbehandlung für das Ereignis „Dialog-Schließen“ definiert. Diese behandelt das Ereignis nicht selbst, sondern ruft die Methode `handleCloseEvent()` des zuvor definierten Lambdas auf. Der Methodenname ist im Interface `POSDialogCloseHandler` festgelegt.

Testat

Nach Fertigstellung aller Klassen testen Sie diese gegen alle JUnit-Tests im Ordner *cert*. Erst wenn alle Tests erfolgreich durchlaufen wurden, d. h. „grün“ sind, ist es sinnvoll, Ihre Aufgabe testieren zu lassen.
Im Rahmen des Testats sollten sie die Funktionsweise des Dialogs vorführen und die Implementierung auf Quellcode-Ebene erklären können.