PS Programming Methodology

University of Innsbruck - Department of Computer Science

Blaas A., Hachmeier S., Huaman E., Kaltenbrunner L., Lanzinger P., Löscher M., Moosleitner M., Priller S., Simsek U., Zech P.

27.05.2021

Midtermtest 2

Bei diesem 2. Midtermtest wird das Lösen von Programmieraufgaben am Computer von Ihnen erwartet. Die Formalitäten bzw. Regeln sollten Ihnen bereits bekannt sein. Falls nicht, können Sie sie unter `midterm2InfoSheet.pdf` (OLAT) nachlesen. Bitte lesen Sie sich die Aufgaben genau durch und lösen Sie die einzelnen Teilaufgaben. Für den Test stehen Ihnen 90 Minuten zur Verfügung. Insgesamt können Sie 20 Punkte + Bonuspunkte erreichen. Beachten Sie die kommunizierten Abgabeformalitäten und laden Sie am Ende Ihre Abgabe auf OLAT hoch. Ersetzen Sie im geforderten package-Namen die Beispiel-c-Kennung `cxxxxxxx` durch **Ihre eigene c-Kennung**. Planen Sie zur Sicherheit ein paar Minuten mehr für die Abgabe ein und kontrollieren Sie **gründlich**, ob auch wirklich **alle geforderten Dateien** in Ihrer Abgabe enthalten sind sowie alle Abgaberegeln eingehalten wurden. Abgaben nach Verstreichen der Deadline bzw. außerhalb von OLAT werden **NICHT** akzeptiert.

Aufgabe 1 (Vererbung) [9 Punkte]

Importieren Sie zunächst alle bereitgestellten Klassen aus dem package `at.pm.cxxxxxxx.ex1` aus dem `midterm2.zip`-Archiv (OLAT) in Ihre Entwicklungsumgebung. Die Employee-Klasse (`Employee.java`) ist für die Darstellung verschiedener Mitarbeitertypen verantwortlich. Die Klasse verfügt über die folgenden Attribute und Methoden:

- `name`: Name eines Mitarbeiters
- `type`: Typ eines Mitarbeiters. Er kann einen von drei Werten haben: `STUDENT`, `ADMINISTRATIVE`, `PROFESSOR`
- `hourlyRate`: Stundenlohn eines Mitarbeiters
- `publicTransportationBonus`: ein wöchentlicher Pendelbonus für öffentliche Verkehrsmittel
- `getSalary()`: berechnet das wöchentliche Gehalt eines Mitarbeiters.
- `toString()`: gibt eine Zeichenfolgendarstellung eines Mitarbeiterobjekts zurück.

In der Employee-Klasse gibt es sowohl im Konstruktor als auch in der `getSalary()`-Methode Switch-Case-Blöcke, die die Variablenzuweisungen und Gehaltsberechnungen für jeden Mitarbeitertyp steuern.
Diese Implementierung folgt nicht den objektorientierten Prinzipien. Um dieses Problem zu lösen, müssen Sie die Verwendung von Switch-Case-Blöcken mithilfe von Polymorphismus eliminieren. Die resultierenden Klassen sollten dieselbe Funktionalität wie die angegebene Employee-Klasse bieten.
Implementieren Sie die PolymorphicEmployee-Klasse im Paket `at.pm.cxxxxxxx.ex1`. Jede andere erforderliche Klasse muss ebenfalls im selben Paket implementiert sein. Achten Sie auf die Sichtbarkeit von Methoden und Attributen.

In der `printPolymorphicEmployees()` -Methode in `App.java` sind drei PolymorphicEmployee-Referenzen definiert. Initialisieren Sie diese mit Mitarbeitern, die mit dem Variablennamen übereinstimmen (z. B. muss die `PolymorphicEmployee professor` Variable mit einem Professor initialisiert werden). Stellen Sie sicher, dass die Ausgaben der Methoden `printEmployees()` und `printPolymorphicEmployees()` in `App.java` identisch sind.

Submit

- `at.pm.cxxxxxxx.ex1.PolymorphicEmployee.java`
- `at.pm.cxxxxxxx.ex1.App.java`
- `at.pm.cxxxxxxx.ex1.Employee.java`
- `at.pm.cxxxxxxx.ex1.EmployeeType.java`

und alle anderen Klassen, die Sie im Package `at.pm.cxxxxxxx.ex1` erstellten

Aufgabe 2 (JUnit-Tests) [5 Punkte]

Importieren Sie zunächst alle bereitgestellten Klassen aus dem package `at.pm.cxxxxxxx.ex2` aus dem `midterm2.zip`-Archiv in Ihre Entwicklungsumgebung. Sie realisieren die stark vereinfachte Funktionalität eines shopping-carts. Die Klasse `ShoppingCart.java` enthält folgende Methoden:

- `addItem(Item item, long amount)`: Fügt ein item vom Typ `Item` amount-mal in das Shopping-cart ein. Wird für den Parameter `amount` eine Zahl kleiner gleich 0 ausgewählt, wird eine `IllegalArgumentException` geworfen.
- `removeItem(Item item, long amount)`: Entfernt die gegebene Anzahl an items (`amount`) aus dem shopping-cart. Bei einem `amount` kleiner gleich 0 wird eine `IllegalArgumentException` geworfen. Befindet sich das übergebene `item` nicht im shopping-cart oder wird versucht, eine höhere Anzahl an items aus dem shopping-cart zu entfernen, als vorhanden sind, wird eine `ItemRemovalException` geworfen.
- `calculatePrice()`: Berechnet den Gesamtpreis aller sich im shopping-cart befindlichen items. Befindet sich ein item mehrmals im shopping-cart, so wird der Preis des Items mit seiner Anzahl multipliziert.

Öffnen Sie nun die Klasse `ShoppingCartTest.java`. Ihre Aufgabe ist es, JUnit-Tests für die beschriebenen Methoden bereitzustellen. Sie müssen hierfür nicht Zeile für Zeile des zur Verfügung gestellten Codes durchgehen. Die im vorherigen Abschnitt bereitgestellte Beschreibung der Methoden sollte ausreichend sein, damit Sie die Tests ausimplementieren können. Lösen Sie nun die folgenden Aufgaben, indem Sie die bereits zur Verfügung gestellten Test-Methoden ausimplementieren:

a) *1 Punkt* Implementieren Sie die Methode `testAddItem()`: Rufen Sie die `addItem(..)`-Methode des shopping-carts auf und überprüfen Sie danach, ob das Item in der gewünschten Anzahl (`amount`) hinzugefügt wurde. Sie können hierfür die beiden Methoden `getItemEntry(..)`- bzw. `getTotalAmountOfItems()` der ShoppingCart-Klasse verwenden.

b) *1 Punkt* Implementieren Sie die Methode `testAddItemIllegalArgument()`: Rufen Sie die `addItem(..)`-Methode mit einem negativen `amount` auf. Der Test ist dann erfolgreich, wenn eine `IllegalArgumentException` geworfen wird.

c) *1 Punkt* Implementieren Sie die Methode `testRemoveItem()`: Fügen Sie ein item mit einer beliebigen Anzahl ins shopping-cart, indem Sie die `addItem(..)`-Methode aufrufen. Rufen Sie dann die `removeItem(..)`-Methode auf und entfernen Sie die gesamte Anzahl des Items bzw. einen Teil davon. Überpüfen Sie dann den Inhalt des shopping-carts mit den bereits erwähnten Hilfsmethoden.

d) *1 Punkt* Implementieren Sie die Methode `testRemoveItemNonExisting()`: Versuchen Sie, ein sich nicht im shopping-cart befindliches Item zu entfernen. Der Test ist erfolgreich, wenn eine `ItemRemovalException` geworfen wird.

e) *1 Punkt* Implementieren Sie die Methode `testCalcPrice()`: Fügen Sie mindestens zwei verschiedene Items zum shopping-cart hinzu. Überprüfen Sie dann, ob der berechnete Gesamtpreis dem erwarteten Wert entspricht.

Submit

- `at.pm.cxxxxxxx.ex2.Item.java`
- `at.pm.cxxxxxxx.ex2.ItemEntry.java`
- `at.pm.cxxxxxxx.ex2.ItemRemovalException.java`
- `at.pm.cxxxxxxx.ex2.ShoppingCart.java`
- `at.pm.cxxxxxxx.ex2.ShoppingCartTest.java`

Aufgabe 3 (Generics) [6 Punkte]

Implementieren Sie für diese Aufgabe eine vereinfachte Version einer sortierten Liste namens `SortedList`. Folgende Anforderungen sind gegeben bzw. Methoden müssen implementiert werden:

- Die Liste muss generisch sein, d.h. alle Typen sind erlaubt, welche das `Comparable`-Interface implementieren.
- `SortedList` darf nach außen hin nur die im Folgenden beschriebenen Methoden bereitstellen.

Die Wahl der zugrunde liegenden Datenstruktur bleibt Ihnen überlassen. Sie sind dabei keinerlei Einschränkungen unterworfen. Als einziges Kriterium gilt: Die beschriebenen Anforderungen müssen erfüllt werden. Lösen Sie nun die folgenden Teilaufgaben:

1. Implementieren Sie die Methode `void addElement(T element)`, welche das übergebene Element `element` zur Liste hinzufügt.
2. Implementieren Sie die Methode `void removeElement(T element)`, welche ein Vorkommnis des übergebenen Elements `element` aus der Liste löscht.
3. Implementieren Sie die Methode `List<T> getSorted()`, welche alle Elemente der Liste in sortierter Reihenfolge zurückgibt.
4. Erstellen Sie eine Klasse `Main.java` in welcher Sie Ihre Implementierung testen. Legen Sie hierfür zwei Instanzen Ihrer implementierten `Sorted`-Klasse mit unterschiedlichem Typ an und fügen Sie jeweils mindestens drei Elemente ein. Rufen Sie dann die `getSorted()`-Methode auf und geben das Ergebnis aus. Löschen Sie jeweils ein Element aus der Liste, rufen Sie dann die `getSorted()`-Methode erneut auf und geben Sie wiederum das Ergebnis aus.

Submit

`at.pm.cxxxxxxx.ex3.SortedList.java`

`at.pm.cxxxxxxx.ex3.Main.java`

Aufgabe 4 (Bonus*) [2 Punkte]

Durch die Lösung dieser Bonusaufgabe können Sie zusätzliche Bonuspunkte sammeln. Das Lösen dieser Aufgabe ist daher optional.
Ersetzen Sie Ihre `SortedList`-Implementierung aus `Aufgabe 3 (Generics)` durch eine effizientere Version mit einer Laufzeitkomplexität von `O(n)` oder besser. **Dies gilt für alle drei Methoden**. Begründen Sie mithilfe von Kommentaren im Source-Code die verbesserte Laufzeitkomplexität Ihrer Implementierung.

Hinweis

Sie müssen hierfür KEINE neue, eigene Version der Klasse `SortedList` abgeben. Sie können die Verbesserung direkt in der Klasse für Aufgabe 3 vornehmen, müssen diese aber entsprechend kommentieren bzw. erklären. Bei einer fehlenden Erklärung erhalten Sie **KEINE** Bonuspunkte.