Programmieren 1

Dr. Arne Schmidt

Wintersemester 2023/2024
Abgabe: 22.12.2023
Rückgabe: ab 08.01.2024

Übungsblatt 3

Abgabe der Lösungen bis zum 22.12.2023 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 (Warenautomat): **(7 Punkte)**
In dieser Aufgabe möchten wir einen Warenautomaten entwickeln. Betrachte dazu das Klassendiagramm in Abbildunng 1.

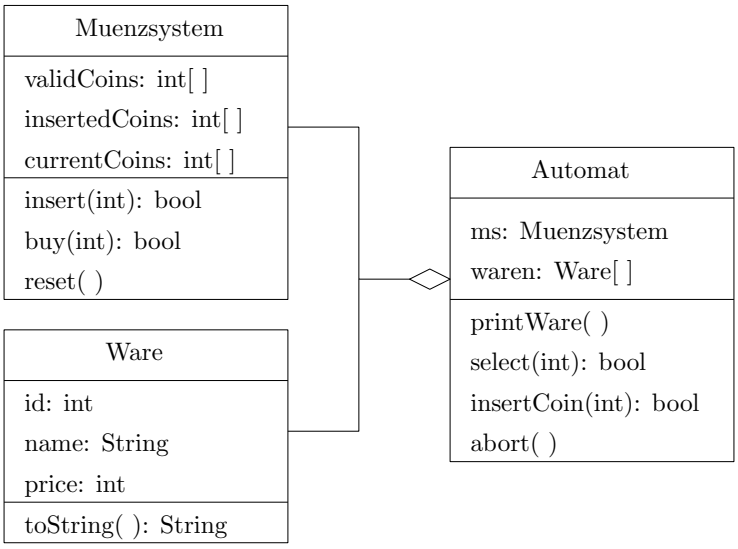

Abbildung 1: Klassendiagramm für den Warenautomaten

Die zu verkaufenden **`Ware`**n werden durch eine `id` (die auch gleich der Nummer entspricht, über welche die Ware gekauft werden kann), sowie einem Namen `name` (Beschreibung der Ware) und einen Preis `price` (der Verkaufspreis der Ware in Cent) beschrieben.

Das **`Muenzsystem`** sorgt dafür, dass (1) Geld in den Automaten eingeworfen werden kann (`insert`); (2) das eingeworfene Geld zurück gibt (`reset`); (3) den Kauf vollzieht, sollte genügend Geld eingeworfen sein (`buy`). Wechselgeld ist dabei nicht vorgesehen. Das Array `validCoins` gibt dabei an, welche Münzen zur Verfügung stehen. Wir wollen hier nur Münzen mit den Werten 1, 2, 5, 10, 20, 50, 100 und 200 nutzen. Das Array `currentCoins` soll angeben, wie viele der jeweiligen Münze vom Nutzer gerade eingeworfen wurden. `insertedCoins` gibt an, wie viele der Münzen über alle Käufe eingeworfen worden sind. (currentCoins[i] bzw. insertedCoins[i] zeigt also die Anzahl an Münzen mit dem Wert validCoins[i] an.)

Der **`Automat`** dient als Schnittstelle für den Nutzer und besteht aus einem Münzssystem (`Muenzsystem ms`) und einem Sortiment an Waren (`Ware[] waren`). Mit `insertCoin(int)` soll eine Münze eingeworfen werden, welches dann über das Münzsystem überprüft wird. Weiter soll eine Liste mit Waren ausgegeben werden können (`getWare()`), sodass der Nutzer erkennt, mit welcher Nummer eine Ware gekauft werden kann. Die Funktion `select(int)` versucht dann die Ware mit der ID des übergebenen Wertes zu kaufen. Der Nutzer kann mit `abort()` alle eingegebenen Münzen zurückfordern.

a) Implementiere die vorgegebenen Konstruktoren, sowie die Funktionen für die Klassen `Muenzsystem`, `Ware` und `Automat` und sinnvolle Get- und Set-Methoden. Eigene sinnvolle Funktionen dürfen beliebig ergänzt werden.

b) Implementiere eine main-Methode in der `main.java`, welche

   (i) einen Automaten mit mindestens 5 Waren mit unterschiedlichen Preisen erstellt.

   (ii) die eingegebenen Waren über den Automaten auf die Konsole ausgibt (`printWare()`).

   (iii) mindestens 3 Münzen in den Automaten wirft, worunter mindestens eine nicht gültige und eine gültige Münze ist.

   (iv) einen reset anfordert, um die gültig eingeworfenen Münzen zurückzubekommen.

   (v) genügend gültige Münzen einwirft, um anschließend eine Ware zu kaufen.

Hausaufgabe 2 (Bankaccounts): **(8 Punkte)**
In dieser Aufgabe betrachten wir ein Bankensystem. Siehe Abbildung 2 für ein Klassendiagramm.

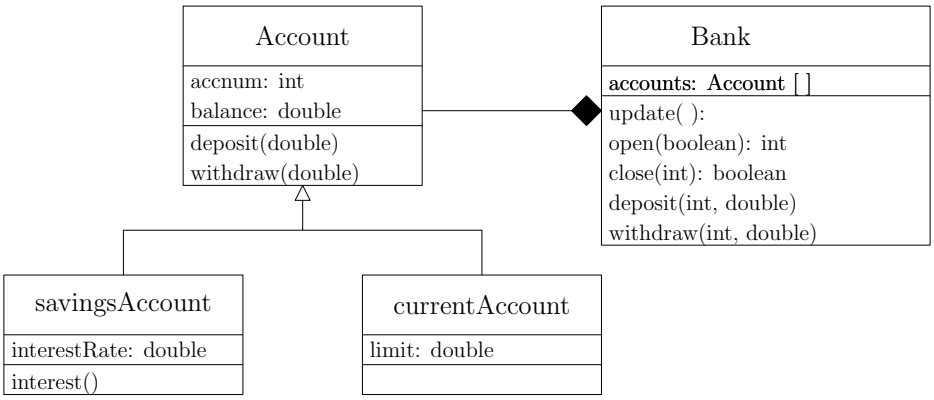

Abbildung 2: Klassendiagramm für das Bank-Szenario.

Ein Konto (`Account`) bei einer Bank besteht üblicherweise aus einer Accountnummer (`accnum`) und einem Geldbetrag (`balance`). Ein Konto bietet folgende Funktionen an, welche nur über die Bank aufgerufen werden sollen:

- Einzahlen (`deposit`) fügt einen nicht-negativen Wert auf den Geldbetrag hinzu.
- Abheben (`withdraw`) entfernt einen nicht-negativen Wert vom Geldbetrag, sofern genügend Geld vorhanden ist.

Nun sollen zwei verschiedene Typen von Konten existieren:

- Sparkonto (`savingsAccount`): Der Geldbetrag muss immer mindestens 100.00€ betragen. Es können über eine neue Funktion Zinsen (`interest`) zu einem für das Konto festgesetzten Zinssatz (`interestRate`) gezahlt werden.

- Girokonto (`currentAccount`): Dieses Konto darf bis zu einem festgelegten Limit (limit) überzogen werden.

Eine `Bank` kann bis zu 30 Accounts halten und kann mit Hilfe einer `update`-Funktion die Konten aktualisieren. Dabei erhält ein Sparkonto die Zinsen und für ein Girokonto soll eine Nachricht ausgegeben werden, falls es überzogen ist (Geldbetrag ist kleiner als 0.00€). Die Bank stellt Funktionen zur Eröffnung (`open`; ein boolean gibt an, welcher Accounttyp erstellt werden soll) und Schließung (`close`; ein Integer gibt an, welches Konto geschlossen werden soll) von Konten bereit. Außerdem kann über die Bank Geld eingezahlt und abgehoben werden, indem die Accountnummer, sowie der Betrag angegeben wird.

a) Implementiere die Klassen Account, savingsAccount und currentAccount mit Hilfe von Vererbung. Achte darauf, dass nur notwendige Funktionen überladen werden.

b) Implementiere die Klasse Bank. Mache dir dabei Polymorphismus zu Nutze. Nutze in der update-Funktion außerdem den Operator `instanceof`. Die Bank gibt auf SavingsAccounts immer 3% Zinsen, und hat für CurrentAccounts ein festes Limit von -1000€.

c) Implementiere eine main-Methode, welche eine Bank mit 20 Accounts verschiedener Typen eröffnet und drei Mal folgende Operationen wiederholt:

   - jedem Konto einen zufälligen Geldbetrag überweisen,
   - versuchen, von jedem Konto Geld abzuheben, sodass Sparkonten manchmal über 100€ bleiben und manchmal nicht, und Girokonten teilweise unter 0€ fallen.
   - die Konten aktualisiert.

   Zum Schluss soll mindestens ein Account wieder geschlossen werden.

   (Hinweis: In Java können Zufallszahlen zwischen 0.0 und 1.0 mit `Math.random()` erzeugt werden.)