University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

Informatik I – Eprog HS11

Übung 3

1 Aufgabe: Begriffe der Objektorientierung

1.1 Lernziele

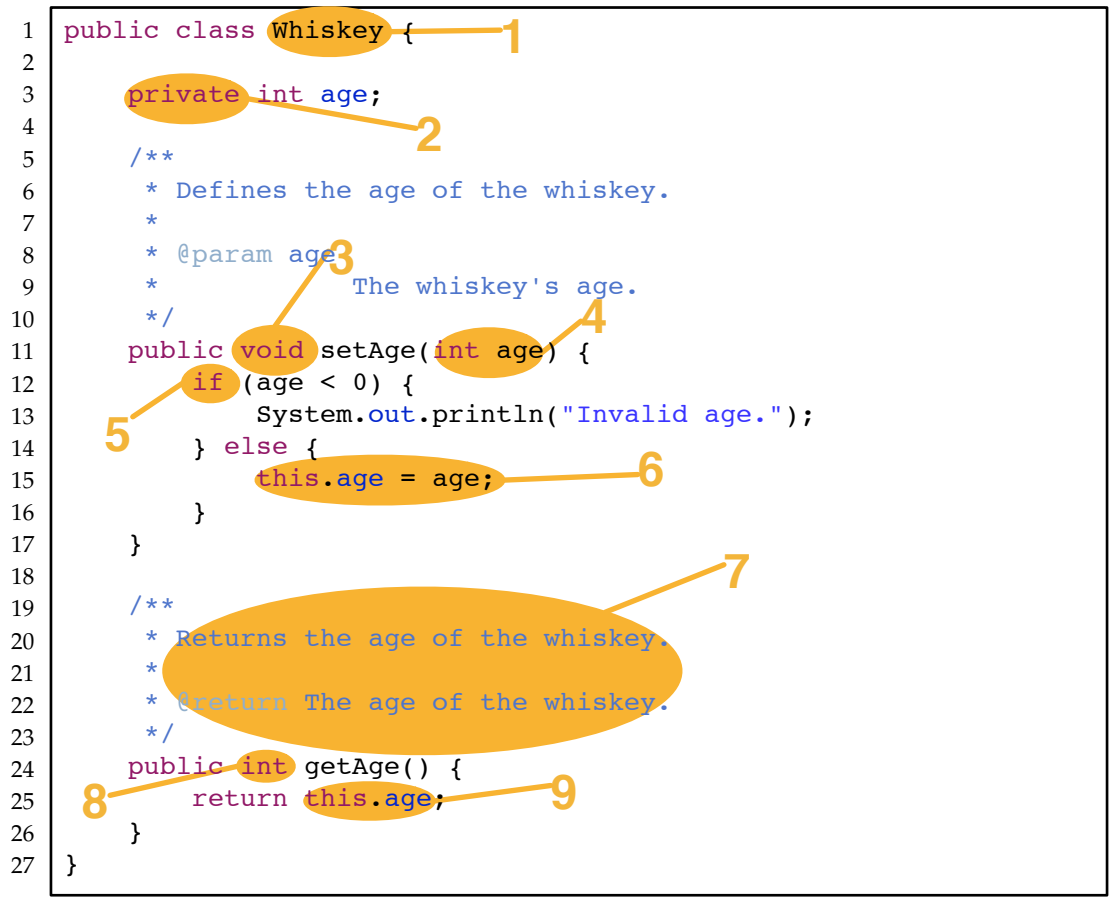
1. Gängige Begriffe der objektorientierten Programmierung definieren und zuordnen können.

1.2 Aufgabenstellung

Ordnen Sie folgende Begriffe den jeweiligen Nummern aus dem untenstehenden Java-Code (Figure 1) zu.

a) Zuordnung

(a) Methodenkommentar
(b) Parameter
(c) Leerer Rückgabewert
(d) Instanzvariable
(e) Klassenname
(f) Initialisierung
(g) Condition
(h) Visibility Modifier
(i) Deklaration des Rückgabewertes




```
1  public class Whiskey {
2
3      private int age;
4
5      /**
6       * Defines the age of the whiskey.
7       *
8       * @param age
9       *            The whiskey's age.
10      */
11     public void setAge(int age) {
12         if (age < 0) {
13             System.out.println("Invalid age.");
14         } else {
15             this.age = age;
16         }
17     }
18
19     /**
20      * Returns the age of the whiskey.
21      *
22      * @return The age of the whiskey.
23      */
24     public int getAge() {
25         return this.age;
26     }
27 }
```

Figure 1: Java Terminologie

2 Aufgabe: Klassen

2.1 Lernziele

1. Eine Klasse mit Zustand (Instanzvariablen) und setter-Methoden implementieren können.
2. Anwendung und Vertiefung der String-API.

2.2 Aufgabenstellung

a) Employee

Implementieren Sie eine Klasse `Employee` mit drei Instanzvariablen `firstName`, `lastName` und `salary`. Implementieren Sie für die ersten beiden Instanzvariablen jeweils eine Methode, die einen Parameter vom Typ `String` erwartet und diesen Wert der entsprechenden Instanzvariablen zuweist. Man spricht in diesem Zusammenhang auch von setter-Methoden und bildet sie nach folgendem Schema:

```
1 public void setInstanzvariable(Type value) {
2    ...
3    this.instanzvariable = value;
4    ...
5 }
```

z.B.:

```
1
2 public void setLastName(String lastName) {
3    ...
4    this.lastName = lastName;
5    ...
6 }
```

Erstellen Sie zusätzlich die setter-Methode der Instanzvariable `salary` welche einen `Integer`-Wert zwischen 3500 und 8000 erwartet. Falls eine Zahl ausserhalb dieses Wertebereiches eingegeben wird, so soll der Wert nicht gespeichert und eine entsprechende Fehlermeldung auf der Konsole ausgegeben werden.

Ergänzen Sie die Klassendefinition um eine Methode `toString()`, die ein `String`-Objekt in folgendem Format zurückliefert:

LastName FirstName, Salary

Zusätzlich soll eine Methode `getInitials()` implementiert werden, welche die Initialen der Person in folgendem Format zurückliefert:

F. L

Verwenden Sie dabei zwei unterschiedliche Wege um Strings zu konkatenieren. Erstellen Sie einen TestDriver um drei unterschiedliche Instanzen der Klasse `Employee` zu erzeugen und testen Sie die implementierten Methoden, indem Sie die Rückgabewerte auf der Konsole ausgeben.

3 Aufgabe: Encapsulation, Interaction, Conditionals

3.1 Lernziele

1. Eine Ausgangslage nach objektorientierten Kriterien vollständig analysieren können.
2. Analysiertes sinnvoll in Java abbilden und implementieren.

3.2 Aufgabenstellung

Im Rahmen einer Zusammenarbeit des Instituts für Informatik und des Swiss Banking Instituts zur Entwicklung einer Banking-Simulation, sind Sie dafür verantwortlich, dass die Kontoführung gemäss den nachfolgenden Vorgaben einwandfrei funktioniert. Implementieren Sie gemäss Ihrer Analyse der Ausgangslage entsprechende Klassen, Eigenschaften und Methoden in Java.[1]

Jedes Konto verfügt über einen Kontostand (inkl. Rappen) und kennt den Namen seines Besitzers, welcher festgelegt werden kann. Ein Konto verfügt ausserdem über eine ganzzahlige Monatslimite. Diese legt fest, wieviel Geld im selben Monat insgesamt maximal abgehoben werden kann. Bei Eröffnung eines Kontos beträgt die Standardlimite Fr. 2000. Ein Konto kann nicht überzogen werden, d. h. es kann nicht mehr Geld abgehoben werden als sich auf dem Konto befindet.

Der Name des Kontobesitzers soll jederzeit abgefragt werden können, ebenso der Kontostand. Des weiteren kann die Monatslimite auf den nächsten Monat hin angepasst werden.

Auf das Konto können beliebige positive Einzahlungen (unter Angabe des Betrags und des Transaktionsdatums[2]) gemacht werden, sofern das Datum der Transaktion neuer als dasjenige der letztmaligen Transaktion ist.

Beim Abheben wird ebenfalls zunächst überprüft, ob die Transaktion am selben Tag oder nach der vorhergehenden Transaktion stattfindet. Es sollen natürlich auch nur positive Beträge abgehoben werden können. Dann wird die verbleibende Limite ermittelt. Diese wird dabei unter Umständen – falls die zu tätigende Transaktion mindestens im Folgemonat der zuletzt getätigten ist – auf die Monatslimite zurückgesetzt. Falls alle Bedingungen erfüllt sind und ausreichend Geld auf dem Konto vorhanden ist, werden Kontostand, verbleibende Limite sowie letztes Transaktionsdatum entsprechend angepasst.

[1] Denken Sie daran, Eigenschaften und Methoden im Normalfall – ausser bei Schnittstellen – `private` zu halten und wiederverwendbaren Code in Hilfsmethoden zu schreiben.

[2] Sie dürfen die bereit gestellte Klasse `Date` benutzen. Aus dieser Klasse könenn Sie die Methoden `isNewerThan` und `isSameYearMonth` zum Vergleichen zweier Gregorianischen Kalender verwenden.