PROGRAMMIER-
TECHNIK I
ÜBUNG

9

Bitte lösen Sie die Übungsaufgaben selbstständig z.B. während der Übungszeit. Die Übungen am 09/10 Dezember finden asynchron statt, d.h. Sie können die Aufgaben in Ihrem Tempo an einem beliebigen Zeitpunkt lösen. Wir besprechen das Aufgabenblatt 9 und das Aufgabenblatt 10 gemeinsam am 16/17 Dezember.

Aufgabe 9.1 (Konstruktoren-Experimente, argumentlos)

a) Erstellen Sie eine Oberklasse Obst und eine Unterklasse Apfel. Definieren Sie in der Klasse Obst einen argumentlosen Konstruktor, der einfach die Zeile „Ich mache ein Obst“ ausgibt. Erstellen Sie eine Klasse ObstKonstruktorTest und erzeugen Sie einen Apfel. Was ist die Ausgabe?

b) Erstellen Sie eine **Unterklasse Birne** (erbt von Obst) und definieren Sie hier einen argumentlosen Konstruktor, der einfach die Zeile „Ich mache eine Birne“ ausgibt. Erzeugen Sie in Ihrer ObstKonstruktorTest-Klasse eine Birne. Was wird ausgegeben?

c) Versuchen Sie super() nach der Ausgabezeile im Konstruktor Birne aufzurufen? Was passiert? Warum? Wie müssen Sie Ihren Konstruktor ändern? Macht es einen Unterschied ob der super()-Aufruf im Birnen-Konstruktor aufgerufen wird oder dieser weggelassen wird?

Aufgabe 9.2 (Konstruktoren) – Gegeben sei das nachfolgende Programm. Bestimmen Sie dessen Ausgaben. In welcher Reihenfolge werden Konstruktoren aufgerufen? Was können Sie über den Konstruktionsprozess ableiten?

```
01 public class Mahlzeit {
02     Mahlzeit() {
03             System.out.println ("Mahlzeit");
04     };
05 }
06
07 public class Abendessen extends Mahlzeit {
08     public Abendessen () {
09             System.out.println ("Abendessen");
10     }
11 }
12
13 public class Italiener extends Abendessen {
14     public Italiener() {
15                     System.out.println ("Italiener");
16     }
17 }
18
19 public class Pizza extends Italiener {
20     Boden steinofenBoden = new Boden();
21     Tomatensosse frischeTomate = new Tomatensosse();
22     Belag superBelag = new Belag();
23
24     public Pizza() {
25             System.out.println ("Pizza");
26     }
27 }
28
29 public class Boden {
30     public Boden () {
31             System.out.println ("Boden");
32     }
33 }

public class Belag {
        public Belag() {
                System.out.println("Belag");
        }
}

public class Tomatensosse {
        public Tomatensosse() {
                System.out.println("Tomatensoße");
        }
}

public class WirHabenHunger {
        public static void main(String[] args) {
                Pizza meinAbendeessen = new Pizza();
        }
}
```

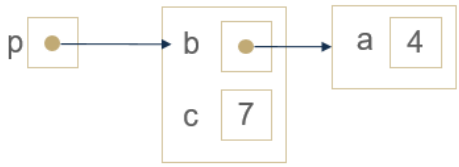
Aufgabe 9.3 (Referenzen) – Gegeben sei das nachfolgende Programm.

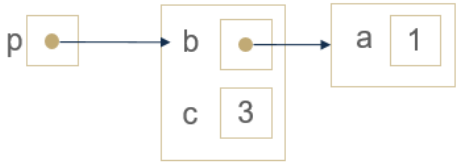
```
01 public class IntClass {
02     public int a;
03     public IntClass (int a) {
04           this.a = a;
05     }
06 }
07
08 public class RefIntClass {
09     public IntClass b;
10     public int c;
11
12     public RefIntClass (int b, int c) {
13           this.b = new IntClass (b);
14           this.c = c;
15     }
16 }
17
18 public class IntRefClassTest {
19     public static void copy1 (RefIntClass x, RefIntClass y) {
20           y.b.a = x.b.a;
21           y.c = x.c;
22     }
23
24     public static void copy2 (RefIntClass x, RefIntClass y) {
25           y.b = x.b;
26           y.c = x.c;
27     }
28
29     public static void copy3 (RefIntClass x, RefIntClass y) {
30           y = x;
31     }
32
33     public static void main(String[] args) {
34           RefIntClass p = new RefIntClass (4, 7);
35           RefIntClass q = new RefIntClass (1, 3);
36           HIER JEWEILS ERSETZEN
37 }
```

Das Ausgangsbild mit den Referenzen nach Ausführung der Zeile 35 sieht folgendermaßen aus. Welches Bild ergibt sich, wenn in der Zeile 36 jeweils eine der folgenden Zeilen eingefügt wird:

a) copy1 (p, q)
b) copy2 (p,q)
c) copy3 (p,q);
d) q = p;

Zeichnen Sie die Referenzen und Werte nach der Kopieraktion!





Aufgabe 9.4 (Klassen, Kunde)

Gegeben sei folgende Klasse.

```
public class Kunde {
	private String name;
	private int alter;

}
```

1. Was passiert in den folgenden Anweisungen?

```
01 Kunde meinKunde;
02 meinKunde = new Kunde();
```

2. Schreiben Sie Konstruktoren für die Klasse Kunde?
   a. Parameterlos (privat)
   b. Mit name als Argument (public)
   c. Mit name und alter als Argument (public)

3. Erweitern Sie die Klasse Kunde um eine Instanzvariable `vorherigerKunde`, die eine Referenz auf einen weiteren Kunden speichert.
4. Erweitern Sie die Klasse um eine Instanzvariable `kundenNummer`, die es ermöglicht, jedem Kunden eine eindeutige, ganzzahlige Nummer zuzuweisen.
5. Erweitern Sie die Klasse um eine Klassenvariable `folgeNummer`, die die nächste zu vergebene Nummer enthält.
6. Erweitern Sie Ihre bisherigen Konstruktoren um die
   a. Vergabe der kundenNummer
   b. Aktualisierung der folgeNummer

   Tipp: Über this() kann der private parameterlose Konstruktor aufgerufen werden. Erweitern Sie die Punkte a. und b. im privaten parameterlosen Konstruktor und rufen Sie diesen in allen anderen Konstruktoren auf.

7. Stellen Sie einen weiteren Konstruktor bereit, der es neben dem Namen und dem Alter ermöglicht, einen Vorgänger anzugeben.
8. Erweitern Sie Ihre Klasse Kunde um eine Methode istErster, die true liefert falls das Kundenobjekt keinen Vorgänger in der Warteliste hat.
9. Schreiben Sie getter und setter für Ihre Klasse. Welche setter sollten Sie auf gar keinen Fall schreiben? Warum?
10. Erweitern Sie Ihre Klasse Kunde um eine Methode printKunde, die die wichtigsten Infos zum Kunden auf der Konsole ausgibt.

11. Gegeben sei folgende Methode. Was bewirkt die Methode? Warum? Schauen Sie sich die Beispielaufrufe in den Zeilen 05 – 07 an. Laden Sie die Codezeilen in Ihr Programm. Hat sich Ihre Vermutung bestätigt?

```
public String toString() {
	String ausgabe = name + " " + kundenNummer + "";
	if (vorherigerKunde != null) {
		ausgabe = ausgabe + " kommt nach " + vorherigerKunde;
	}
	return ausgabe;
}
```

```
// Aufruf

01 Kunde kunde1 = new Kunde("Anton", 54);
02 Kunde kunde2 = new Kunde("Berta", 32, kunde1);
03 Kunde kunde3 = new Kunde("Cäsar", 19, kunde2);
04
05 System.out.println(kunde1.toString());
06 System.out.println(kunde2.toString());
07 System.out.println(kunde3.toString());
```

12. Erläutern Sie den Aufbau ihrer Daten aus Zeile 01 – 03 grafisch analog zur Aufgabe 3.

Aufgabe 9.4 (Polymorphie, Boote)

1. Erstellen Sie eine Klasse Boot, mit der privaten Instanzvariablen `groesse` (gibt die Länge des Boots in cm an) und der privaten Instanzvariablen `name`.
2. Stellen Sie einen Konstruktor bereit, der das Boot mit der übergebenen Größe und dem Namen initialisiert. Es soll weder ein parameterloser Konstruktor noch ein Konstruktor, der lediglich den Namen oder die Größe initialisiert, definiert sein.
3. Stellen Sie einen `getter` und einen `setter` für die Instanzvariable groesse bereit.
4. Definieren Sie 3 Klassen, die sich direkt von der Klasse Boot ableiten und definieren Sie in den jeweiligen Klassen einen Konstrukor:
   - RuderBoot
   - SegelBoot
   - MotorBoot

5. Definieren Sie eine Methode
   `public String toString ()`
   die alle Eigenschaften eines Boots als String zurückgibt.
6. Definieren Sie eine Klasse Hafen, die eine private Instanzvariable hat, die ein Array von Booten darstellt (das sind die Liegeplätze am Hafen).
   - Definieren Sie einen Konstruktor, der mit der Anzahl der Liegeplätze aufgerufen wird und das Array initialisiert. Einen parameterlosen Konstruktor soll es nicht geben.
     Definieren Sie eine private Klassen-Variable `nochFreiePlaetze`, die die Anzahl freier Plätze speichert. Erweitern Sie den Konstruktor derart, dass beim Erstellen des Hafens, die Klassenvariable `nochFreiePlaetze` initialisiert wird.
     Schreiben Sie eine Methode `getAnzahlLiegeplaetze`, die die Anzahl noch freier Liegeplätze angibt.
     Schreiben Sie eine Methode `nochFreiePlaetze`, die true zurückliefert, falls mindestens ein Liegeplatz im Hafen frei ist.

- Schreiben Sie eine Methode
  - public int addBoot (Boot b), die dem Boot einen Liegeplatz zuweist falls noch ein Liegeplatz frei ist. Die Methode gibt die Liegeplatznummer zurück wenn ein Platz frei ist, -1 sonst.
- Schreiben Sie eine Methode, die das größte Boot und eine Methode die das kleinste Boot zurückgibt.
- Schreiben Sie eine Methode, die die Boote nach der Größe sortiert. Verändern Sie dabei aber nicht die Liegeplatzzuordnung im Hafen, der Hafenmeister wird es Ihnen danken. Verwenden Sie hierfür den Bubble-Sort-Algorithmus. Greifen Sie dabei NICHT auf eine vordefinierte Java-Bibliothek zurück.