Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Prof. Dr. A.B. Cremers

Vorlesung Objektorientierte Softwareentwicklung
Sommersemester 2010
Dr. Günter Kniesel

Aufgabenblatt 11

— Veröffentlicht am 13.07.2010 —

Abgabe bis 20.07.2010, 13:00 per SVN

Aufgabe 1 *(Vermischte Aufgaben - 10 Punkte)*

Überlegen Sie sich für jede Aussage ein Argument, mit dem Sie eindeutig bestimmen können, ob die Aussage wahr oder falsch ist. Für jede richtig angekreuzte Antwort werden 0,5 Punkte angerechnet, für jede falsch angekreuzte Antwort 0,5 Punkte abgezogen. Nicht angekreuzte Antworten geben keinen Punktabzug.

wahr	falsch	
		Jede Java-Klasse muss eine statische Methode mit dem Namen `main` zur Verfügung stellen.
		Nicht gekennzeichnete Ganzzahlen (z. B. `5`) sind in Java vom Typ `integer`, nicht gekennzeichnete Dezimalzahlen (z. B. `5.0`) vom Typ `double`.
		Folgende Deklaration ist in Java gültig: `int final=100;`
		Eine Java-Klasse kann zwei Methoden mit dem selben Namen besitzen.
		Auf öffentliche Klassenvariablen kann zugegriffen werden, ohne ein Objekt der Klasse zu erzeugen.
		Bei der *Garbage Collection* werden alle Objekte, die nicht direkt vom *Stack* aus referenziert sind, aus dem Speicher entfernt.
		Jede Java-Klasse, die einen *Konstruktor* definiert, muss auch über einen *Destruktor* verfügen.
		Der Wert eines als *final* definierten Java-Datenfeldes kann nachträglich nicht mehr verändert werden.
		Mit `super.m();` kann in Java explizit auf die Methode `m()` der Oberklasse zugegriffen werden, selbst wenn diese in der Unterklasse überschrieben wurde.
		Alle objektorientierten Sprachen nutzen Klassen als Objekt-Schablonen.

Aufgabe 2 *(Statisches und Dynamisches Binden - 8 Punkte)*

Gegeben seien folgende Java-Klassen:

```
class C1 {
  static int s=1;
  void f1(){
    System.out.println("C1::f1");
  }
  void f2(){
    System.out.println("C1::f2");
  }
}

class C2 extends C1 {
  static int s=2;
  void f1(){
    System.out.println("C2::f1");
  }
  void f3(){
    System.out.println("C2::f3");
  }
}
```

```
public class Test {
  static public void main ( String[ ] args ) {
    C1 a=new C2();
    C2 b=new C2();
    // Stelle 1
  }
}
```

Geben Sie an, was das Ausgabeergebnis von folgenden Codefragmenten ist, wenn sie an `Stelle 1` eingefügt werden. Neben der Ausgabe – die auch leer sein kann und in diesem Fall durch $\emptyset$ gekennzeichnet werden soll – sind auch *Compilierfehler* bzw. *Laufzeitfehler* mögliche Antworten.

1. `((C2) a).f1();`
2. `((C1) b).f1();`
3. `System.out.println(a.s);`
4. `b.f1();`
5. `b.f2();`
6. `b.f3();`
7. `C1 c = new C1(); c.f3();`
8. `C1 c = new C1(); ((C2) c).f1();`

Aufgabe 3 *(Programmierfehler - 3 Punkte)*

Student Hans hat folgendes Codefragment geschrieben, um in einer Zahlen-Liste nach der Zahl `25` zu suchen. Leider befinden Sich noch drei Fehler in dem Programm. Helfen Sie Hans, die Fehler zu finden und markieren Sie diese!

```
int[] zahlen = {1, 2, 3, 10, 25, 36, 70};

for (int i=1; i<=zahlen.length; i++) {

      if (zahlen[i] = 25) {

            System.out.println("Die Zahl wurde gefunden!");

      }

}
```

Aufgabe 4 *(Identität und Gleichheit - 13 Punkte)*

3+4+5+1 Diese Aufgabe beschäftigt sich mit dem Klonen von `Schaf`-Objekten. Unter dem *Klon* O_k eines Objektes O versteht man ein unabhängiges Objekt, welches den gleichen Typ wie O hat und dessen interner Zustand mit dem Zustand von O übereinstimmt. Damit gilt also: O_k ist gleich O, aber nicht identisch.

Ein `Schaf`-Objekt ist über folgende Klasse definiert:[1]

```
public class Schaf {

  private DNA dna;
  private String name;

  public Schaf(String name) {
    this.name = name;
    this.dna = DNA.getUniqueDNA(); //Erstellt eine eindeutige DNA
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  /**
   * Erstellt eine identische Kopie des aktuellen Objekts
   **/
  public Schaf klone() {
    return this;
  }
}
```

a) Obwohl die Klasse kompiliert und scheinbar funktioniert, ist die Methode `klone()` dennoch konzeptionell fehlerhaft. Beschreiben Sie das Problem und geben Sie ein kurzes Java-Code-Fragment an, mit dem sich das Problem demonstrieren lässt.

b) Geben Sie eine korrekte Implementierung der Methode `klone()` an. Dabei dürfen Original-Objekt und Klon identische Objekte referenzieren:

```
  public Schaf klone() {




  }
```

c) Geben Sie eine korrekte Implementierung für die `equals`-Methode der `Schaf`-Klasse an:
Hinweise:

- *Zwei Objekte sind gleich, wenn Ihr interner Zustand, d. h. alle Instanzvariablen, gleich sind.*
- *So können Sie überprüfen, ob Objekt `O` vom Typ `T` ist:* `if (O instanceof T) ...`

```
  public boolean equals(Object obj) {




  }
```

d) Geben Sie eine allgemeine Bedingung an, mit der überprüft werden kann, ob ein Objekt `objekt_1` ein Klon von `objekt_2` ist. Gehen Sie dabei von der Definition eines *Klons* zu Beginn dieser Aufgabe aus:

[1] `DNA` sei eine Java-Klasse mit der korrekten Klassenmethode `public static DNA getUniqueDNA()`, die ein gültiges `DNA`-Objekt zurückgibt.

```
if (                                                              ) {

    System.out.println("objekt_1 ist ein Klon von objekt_2");
} else {
    System.out.println("objekt_1 ist kein Klon von objekt_2");
}
```

Aufgabe 5 *(Katzenmusik - 9 Punkte)*

Das folgende einfache Programm soll funktionieren:

```
public class Katzenmusik {
  public static void main (String[] args) {
    XXX tier1 = new Katze();
    XXX tier2 = new Hund();
    for (int i=1; i<=10; i++) {
      tier1.gibLaut();
      tier2.gibLaut();
    }
  }
}
```

Wie könnte der Typ *XXX* realisiert werden, damit das Programm funktioniert?

1. Diskutieren Sie zwei verschiedene Alternativen und begründen Sie welche Ihnen hier passender erscheint. (2 Punkte)
2. Implementieren Sie die bevorzugte Alternative für XXX in Java. (2 Punkt)
3. Implementieren Sie die die Klassen `Katze` und `Hund` passend zu der gewählten Alternative. (5 Punkte)

Aufgabe 6 *(Matrizenmultiplikation mit Multithreading - 6* Punkte)*

**Bonusaufgabe*: Durch Lösen dieser Aufgabe können Sie sich zusätzliche Bonuspunkte verdienen!

a) Bei der Multiplikation zweier Matrizen können offensichtlich alle Einträge der Ergebnismatrix unabhängig voneinander berechnet werden. Nutzen Sie diese Erkenntnis aus und versuchen Sie Ihre Implementierung aus der vorigen Aufgabe mithilfe von *Multithreading* zu beschleunigen!

b) Ist es sinnvoll, für jeden Eintrag der Ergebnismatrix einen eigenen Thread zu starten? Oder lässt sich mit weniger Threads eine optimale Rechenzeit erreichen? Testen Sie verschiedene Implementierungen und diskutieren Sie die Ergebnisse! Geben Sie an, wie viele Prozessoren die von Ihnen verwendeten Rechner besitzen! *Hinweis*: Den Zeitbedarf (in Millisekunden) eines Anweisungsblocks können Sie sich folgendermaßen anzeigen lassen[2]:

```
long tix = System.currentTimeMillis();
{
  ... // Anweisungen
}
System.out.println(System.currentTimeMillis()-tix);
```

[2]Die Genauigkeit hängt von der Aktualisierungsrate der Systemzeit und damit vom Betriebssystem ab. Um aussagekräftige Ergebnisse zu erzielen sollten Sie daher die Rechenzeiten für große Anzahlen an hintereinander ausgeführten Multiplikationen vergleichen.