Check und Prepare

Lösungsvorschläge

TECHNISCHE UNIVERSITÄT DARMSTADT

Autoren: Nhan Huynh und Darya Nikitina
Fachbereich: Informatik
Übungsblatt: 04
Version: 20. November 2021
Semesterübergreifend

V1 Grundbegriffe

★★★

Erklären Sie kurz in eigenen Worten die Unterschiede der folgenden Konzepte zueinander:

1. Klasse vs. Objekt
2. Objekt- vs. Klassenmethoden
3. Abstrakte Klassen vs. Interfaces
4. Überladen von Methoden vs. Überschreiben von Methoden

Lösungsvorschlag:

1. Klasse vs. Objekt

   Klasse
   Eine Klasse ist eine Beschreibung eines Objekts mit seinen Attributen und enthält die Definitionen aller Methoden (genauer: aller Methoden die gegenüber der Basisklasse hinzukommen oder zwar in der Basisklasse schon vorhanden sind, aber überschrieben werden). Sie dient als Vorlage, aus der dann beliebig viele Objekte erzeugt werden können.

   Objekt
   Ein Objekt ist die Instanziierung einer Klasse mit spezifischen Werten für die Attribute einer Klasse.

2. Objekt- vs. Klassenmethoden

   Objektmethoden
   Objektmethoden können nur mithilfe eines Objekts aufgerufen werden.

   Objektmethoden können auf alle Attribute des Objekts zugreifen und diese Lesen und Schreiben.
   Objektmethoden sind individuell für das Objekt.
   Objektmethoden können zudem auch alle Methoden der Klasse aufrufen.

   Klassenmethoden
   Klassenmethoden können direkt über den Namen der Klasse aufgerufen werden, ohne dass zuvor ein Objekt der Klasse generiert wurde.
   Klassenmethoden werden mit dem Modifier `static` deklariert.

   Klassenmethoden dürfen nur auf Klassenattribute zugreifen und nur Klassenmethoden aufrufen.

3. Abstrakte Klassen vs. Interfaces

Abstrakte Klassen	**Interfaces**
Es können keine Objekte instanziiert werden.	Es können keine Objekte instanziiert werden.
Eine abstrakte Klasse kann implementierte Methoden enthalten.	Bei einem Interface sind die Objektmethoden entweder `default` oder nicht implementiert und die Attribute sind mit dem Modifier `static final` (Konstanten) gekennzeichnet. Zudem kann ein Interface kann auch Klassenmethoden haben.
Die Methoden und Attribute einer abstrakten Klassen können beliebige Zugriffsmodifier besitzen.	Der Zugriffsmodifier der Methoden und Attribute eines Interfaces sind immer `public`, weshalb der Zugriffsmodifier auch weggelassen werden kann.
Auf Klassen gibt es generell nur Einfachvererbung.	Ein Interface kann mehrere Interfaces erweitern und eine Klasse kann mehrere Interfaces implementieren.

4. Überladen von Methoden vs. Überschreiben von Methoden

Überladen von Methoden	**Überschreiben von Methoden**
Überladen von Methoden heißt, dass wir mindestens zwei Methoden vom gleichem Namen innerhalb einer Klasse haben. Der Rückgabetyp und die Parameter können verschieden sein, aber die Parameterlisten müssen verschieden sein. Eine Methode wird nicht ausschließlich durch den Namen identifiziert, sondern auch über die Typen, Anzahl und Reihenfolge ihrer Argumente - ihre Signatur. Daher können mehrere Methoden mit demselben Namen in einer Klasse vorhanden sein, solange sich die Signaturen unterscheiden.	Überschreiben von Methoden heißt, dass die Methode aus der direkt abgeleiteten Klasse exakt den gleichen Namen, denselben Rückgabetyp (oder vom Subtyp) und dieselben Parameter wie die Methode aus der Basisklasse hat.
Die gleichen Methodennamen können für semantisch ähnliche Funktionalitäten auf unterschiedlichen Datentypen verwendet werden. (siehe `String.valueOf(...)` für `boolean`, `char`...)	Dadurch erreicht man, dass beim Objekt der abgeleiteten Klasse die neue Implementierung (in der abgeleiteten Klasse) beim Aufruf der Methode aufgerufen wird und nicht die ursprüngliche Methode. Somit kann man bspw. erreichen, dass Methoden, deren Funktionalität in der abgeleiteten Klasse verändert werden soll, einfach überschrieben werden, aber gleichzeitig die anderen Methoden der Oberklasse immer noch verwenden werden können. Die Unterscheidung, welche Methode verwendet wird, wird anhand des dynamischen Typs (= Typ des Objektes) getroffen. Betrachten wir folgende Fälle: Ist der dynamische Typ die abgeleitete Klasse, so wird die Methode von der abgeleiteten Klasse verwendet. Ist er hingegen die Basisklasse, so wird die Methode von der Basisklasse verwendet.

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V2 Brumm, Brumm, Brumm

Schreiben Sie eine Klasse `Car` zur Repräsentation von Autos, die folgende Anforderungen erfüllen soll:

- Ein Auto hat einen Namen vom Typ `String` und einen Kilometerstand (`mileage`) vom Typ `double`. Beide Attribute sollen `private`, nicht `public`, sein.
- Der Konstruktor soll einen `String` als Parameter erhalten, der den Namen des Autos angibt. Der Konstruktor soll den Namen des Autos setzen und den Kilometerstand auf `0.0` setzen.
- Schreiben Sie die Methoden `public double getMileage()` und `public String getName()`. Diese liefern die entsprechenden Attribute der Klasse `Car` zurück.
- Schreiben Sie die Methode `public void drive(double distance)`, die eine Distanz in Kilometern als Argument erhält und auf den alten Kilometerstand addiert.

Lösungsvorschlag:

```
1  /**
2   * Represents a simple real world car.
3   */
4  public class Car {
5
6    /**
7     * The name of this car.
8     */
9    private String name;
10
11   /**
12    * The current mileage of this car.
13    */
14   private double mileage;
15
16   /**
17    * Constructs a car with the specified name.
18    *
19    * @param name the name of the car
20    */
21   public Car(String name) {
22     this.name = name;
23     mileage = 0;
24   }
```

```
25    /**
26     * Drives the car by the given distance in kilometre.
27     *
28     * @param distance the distance in kilometre that the car should drive
29     */
30    public void drive(double distance) {
31      mileage += distance;
32    }
33
34    /**
35     * Returns the mileage of this car.
36     *
37     * @return the mileage of this car
38     */
39    public double getMileage() {
40      return mileage;
41    }
42
43    /**
44     * Returns the name of this car.
45     *
46     * @return the name of this car
47     */
48    public String getName() {
49      return name;
50    }
51  }
```

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V3 Interfaces

Gegeben sei folgendes Interface

```
1 public interface I1 {
2
3   void m1();
4
5   int[] m2(double param1);
6 }
```

Schreiben Sie nun eine Klasse `C1`, die das Interface `I1` implementiert. Sofern im Body einer Methode eine Rückgabe erwartet wird, geben sie `null` zurück.

Lösungsvorschlag:

```
1  public class C1 implements T1 {
2
3    @Override
4    public void m1() {
5    }
6
7    @Override
8    public int[] m2(double[] param1) {
9      return null;
10   }
11
12 }
```

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V4 Vererbung

Gegeben seien folgende zwei Klasse, die sich im gleichen Package befinden:

```
1  public class B1 {
2
3    public float f;
4    private boolean b;
5    protected byte by;
6
7    int m1() {
8      return -1;
9    }
10
11   private int m2() {
12     return -2;
13   }
14 }
15
16 public class B2 extends B1 {
17
18   int i;
19   public double d;
20
21   protected int m2() {
22     return 2;
23   }
24
25   public static void main(String[] args) {
26     B2 obj = new B2();
27   }
28 }
```

Betrachten Sie die `main`-Methode der Klasse `B2`. Auf welche Attribute können Sie mit dem Objekt `obj` zugreifen? Welche Methoden können Sie aufrufen und welchen Wert geben die Methoden zurück?

Lösungsvorschlag:

- Wir können auf die folgende Attribute mit dem Objekt `obj` zugreifen:
  - `f`, `by`, `i` und `d`
- Wir können die folgenden Methoden mit dem Objekt `obj` aufrufen:
  - `m1`: Sie gibt den Wert `-1` zurück, da die Methode in der Klasse `B1` definiert ist und die Klasse `B2` `B1` erweitert.
  - `m2` der Klasse `B2`: Sie gibt den Wert `2` zurück, da die Methode `m2` von `B1` in der Klasse `B2` überschrieben wurde.

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V5 Gleicher Abstand

Schreiben Sie eine Methode `static boolean evenlySpaced(int a, int b, int c)`, welche genau dann `true` zurückliefert, wenn der Abstand zwischen dem kleinsten und dem mittleren Element genauso groß ist wie der Abstand zwischen dem mittleren und dem größten Element. Dabei kann jeder der Parameter `a`, `b` oder `c` das kleinste, mittlere oder größte Element sein. Die Klasse, zu der die Methode gehört, muss nicht implementiert werden

Lösungsvorschlag:

```java
/**
 * Returns {@code true} if the distance between the smallest and the middle element is as
 * large as the distance between the middle and the largest element.
 *
 * @param a first integer value to be checked
 * @param b second integer value to be checked
 * @param c third integer value to be checked
 * @return {@code true} if the distance between the smallest and the middle element is as
 * large as the distance between the middle and the largest element
 */
static boolean evenlySpaced(int a, int b, int c) {
  return (2 * a - b - c) * (2 * b - a - c) * (2 * c - a - b) == 0;
}
```

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V6 Zahlen aneinanderreihen

Schreiben Sie eine Methode `static int appendIntegers(int[] a)`. Die Methode bekommt ein `int`-Array übergeben und liefert eine Zahl zurück, die entsteht wenn man alle Zahlen des übergebenen Arrays aneinanderreiht. Der Aufruf `appendIntegers (1,2,3)` liefert `123` zurück. Der Aufruf `appendIntegers(43,2,7777)` liefert `4327777` zurück. Sie dürfen nur Variablen von Typ `int` in ihrer Implementation verwenden, keine `Strings` oder Ähnliches. Schleifen sind natürlich erlaubt.

Lösungsvorschlag:

```
1  /**
2   * Creates an integer by appending the elements of the array after each other
3   *
4   * @param a the array to form a single integer
5   * @return the appended integers
6   */
7  static int appendIntegers(int[] a) {
8    // If array is empty
9    if (a.length == 0) {
10     return 0;
11   }
12   // Starting value
13   int result = a[0];
14   // Appends each integer to the previous one
15   for (int i = 1; i < a.length; i++) {
16     result = append(result, a[i]);
17   }
18   return result;
19 }
20
21 /**
22  * Creates an integer by appending the value y to the value x
23  *
24  * @param x the source value to be appended by another value
25  * @param y the value that should be appended to the other value
26  * @return the appended integer
27  */
28 static int append(int x, int y) {
29   // Offset
30   int d = 10;
31   // Determine shifting
32   while (y / d != 0) {
33     d *= 10;
34   }
35   return x * d + y;
36 }
```

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V7 Zahlen einsortieren

Gegeben sei folgende Klasse

```
1 public class ArrayTuple {
2
3   public int[] iArr;
4   public double[] dArr;
5 }
```

Erweitern Sie diese Klasse um eine `public`-Klassenmethode `ArrayTuple split(double[] a)`. Die Methode liefert ein neues Objekt von Typ `ArrayTuple` zurück, in dessen `int`-Array sich alle ganzen Zahlen aus dem übergebenen Array `A` befinden. Im `double`-Array befinden sich die restlichen Zahlen aus dem übergebenen Array.

Lösungsvorschlag:

```
1  /**
2   * Splits the specified array into integers and doubles.
3   *
4   * @param a the array to split
5   * @return the splitted array as {@link ArrayTuple}
6   */
7  public static ArrayTuple split(double[] a) {
8    // Counters to determine the length of the arrays
9    int integers = 0;
10   int doubles = 0;
11   for (double d : a) {
12     if (d == (int) d) {
13       integers++;
14     } else {
15       doubles++;
16     }
17   }
18
19   // Initialize arrays
20   int[] iArr = new int[integers];
21   double[] dArr = new double[doubles];
22   integers = 0;
23   doubles = 0;
```

```
24    // Copy values
25    for (int i = 0; i < a.length; i++) {
26    double d = a[i];
27      if (d == (int) d) {
28        iArr[integers] = (int) d;
29        integers++;
30      } else {
31        dArr[doubles] = d;
32        doubles++;
33      }
34    }
35
36    ArrayTuple tuple = new ArrayTuple();
37    tuple.iArr = iArr;
38    tuple.dArr = dArr;
39    return tuple;
40  }
```

V8 Statischer und dynamischer Typ

```
1  public class Alpha {
2
3    protected int v;
4
5    public Alpha(int a) {
6      v = a;
7    }
8  }
9
10 public class Beta extends Alpha {
11
12   public Beta(int b, int c) {
13     super(b);
14     v = c;
15   }
16
17   public Alpha x1() {
18     super.v++;
19     return new Beta(0, v);
20   }
21
22   public int x2(int x) {
23     return x + ++v + v++;
24   }
25 }
```

```
26
27 public class Gamma extends Beta {
28
29   private short y;
30
31   public Gamma(int d, int e) {
32     super(d, e);
33     y = (short) d;
34   }
35
36   public int x2(int x) {
37     return x - y;
38   }
39
40   public static void main(String[] args) {
41     Alpha a = new Alpha(7);
42     Beta b = new Beta(0, 1);
43     Gamma g = new Gamma(9, 2);
44     a = b.x1();
45     int t = b.x2(5);
46     a = new Beta(10, 12).x1();
47     b = g;
48     int r = g.x2(50);
49   }
50 }
```

Hinweis: Nach Zeile X heißt unmittelbar nach X, noch vor Zeile $X + 1$.

(1) Welchen statischen und dynamischen Typ haben `a`, `b` und `g` nach Zeile 40?

(2) Welchen statischen und dynamischen Typ hat `a` und welchen Wert hat `a.v` nach Zeile 41?

(3) Welchen Wert haben `t` und `b.v` nach Zeile 42?

(4) Welchen statischen und dynamischen Typ haben `a`, `b` und welchen Wert hat `a  v` nach Zeile 44?

(5) Welchen Wert haben `r` und `b.v` nach Zeile 45?

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

Lösungsvorschlag:

(1) Welchen statischen und dynamischen Typ haben `a`, `b` und `g` nach Zeile 40?

- Der statische und der dynamische Typ von `a` ist `Alpha`.
- Der statische und der dynamische Typ von `b` ist `Beta`.
- Der statische und der dynamische Typ von `g` ist `Gamma`.

(2) Welchen statischen und dynamischen Typ hat `a` und welchen Wert hat `a.v` nach Zeile 41?

- Der statische Typ von `a` ist `Alpha` und der dynamische Typ `Beta`.
- `a.v` hat den Wert 2.

(3) Welchen Wert haben `t` und `b.v` nach Zeile 42?

- `t` hat den Wert 11.
- `b.v` hat den Wert 4.

(4) Welchen statischen und dynamischen Typ haben `A`, `b` und welchen Wert hat `a.v` nach Zeile 44?

- Der statische Typ von `a` ist `Alpha` und der dynamische Typ `Beta`.
- Der statische Typ von `b` ist `Beta` und de dynamische Typ `Gamma`.
- `a.v` hat den Wert 13.

(5) Welchen Wert haben `r` und `b.v` nach Zeile 45?

- `r` hat den Wert 41.
- `b.v` hat den Wert 2.

Information:
Der statische Typ ist der Typ, der bei der Variablendeklaration angegeben wird. Er ist bei der Kompilierung schon bekannt.

Der dynamische Typ ist der Typ des tatsächlichen Objekts, der zur Laufzeit bekannt ist und kann vom statischen Typ abweichen. Dieser Typ ist entweder gleich dem statischen Typ oder ein Subtyp des statischen Typs.

Der formale Aufbau sieht folgendermaßen aus:

```
<Statischer Typ> Bezeichner = <Dynamischer Typ>
```

Als Beispiel nehmen wir hierzu `FOPBot`. Als statischen Typ nehmen wir die Klasse `Robot`. Somit kann der dynamische Typ entweder `Robot` sein oder ein Subtyp dessen.

- `Robot robot = new Robot(1, 1, UP, 1);`: In diesem Fall ist der dynamische Typ gleich dem statischen Typ.
- `Robot robot = new SymmTurner(1, 1, UP, 1);`: In diesem Fall ist der dynamische Typ ein Subtyp des statischen Typ.

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V9 Klassen, Interfaces und Methoden

V9.1

Schreiben Sie ein `public`-Interface `A` mit einer Objektmethode `m1`, die Rückgabetyp `double`, einen `int`-Parameter `n` und einen char-Parameter `c` hat.

Lösungsvorschlag:

```
1 public interface A {
2
3   double m1(int n, char c);
4
5 }
```

V9.2

Schreiben Sie ein `public`-Interface `B`, das von `A` erbt und zusätzlich eine Objektmethode `m2` hat, die keine Parameter hat und einen `String` zurückliefert.

Lösungsvorschlag:

```
1 public interface B extends A {
2
3   String m2();
4
5 }
```

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V9.3

Schreiben Sie eine `public`-Klasse `XY`, die `A` implementiert, aber `m1` nicht. Klasse `XY` soll ein `protected`-Attribut `p` vom Typ `long` haben sowie einen `public`-Konstruktor mit Parameter `q` vom Typ `long`. Der Konstruktor soll `p` auf den Wert von `q` setzen. Weiter soll `XY` eine `public`-Objektmethode `m3` mit Rückgabetyp `void` und Parameter `XY` vom Typ `XY` haben, aber nicht implementieren.

Lösungsvorschlag:

```
1  public abstract class XY implements A {
2
3    protected long p;
4
5    public XY(long q) {
6      p = q;
7    }
8
9    abstract public void m3(XY xy);
10 }
```

V9.4

Schreiben Sie eine `public`-Klasse `YZ`, die von `XY` erbt und `B` implementiert. Die Methode `m1` soll `n+c+p` zurückliefern und `m2` den String `“Hallo“`. `m3` soll den Wert `p` von `XY`auf den Wert `p` des eigenen Objektes addieren. Der Konstruktor von `YZ` ist `public`, hat einen `long`-Parameter `r` und ruft damit den Konstruktor von `XY` auf.

Lösungsvorschlag:

```
1  public class YZ extends XY implements B {
2
3    public YZ(long r) {
4      super(r);
5    }
6
7    public double m1(int n, char c) {
8      return n + c + p;
9    }
10
11   public String m2() {
12     return "Hallo";
13   }
14
15   public void m3(XY xy) {
16     p += xy.p;
17   }
18 }
```

Übungsblatt 04: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V10 Jedes dritte Element

Gegeben sei eine Klasse `X`. Schreiben Sie für diese Klasse die `public`-Objektmethode `foo`. Diese hat ein Array `a` von Typ `int` als formalen Parameter und liefert ein anderes Array `b` vom Typ `int` zurück, das aus `a` entsteht, indem jedes dritte Element gelöscht wird, das heißt, die Elemente von `a` an den Indizes $0, 3, 6, 9, \ldots$ werden nicht nach `b` kopiert, alle anderen Elemente von `a` werden in derselben Reihenfolge, wie sie in `A` stehen, nach `b` kopiert. Weitere Elemente hat `b` nicht. Sie dürfen voraussetzen, dass `A` mindestens Länge `2` hat und ungleich `null` ist. Sie dürfen einfach Operator `=` für das Kopieren von Elementen verwenden.

> **Hinweis:** Überlegen Sie sich die Gesetzmäßigkeit, nach der die Indizes $1, 2, 4, 5, 7, 8, \ldots$ in `a` auf die Indizes $0, 1, 2, 3, 4, 5, \ldots$ in `b` abzubilden sind. Für die Länge von `b` werden Sie eine Fallunterscheidung benötigen, je nachdem, welchen Rest `a.length` dividiert durch `3` ergibt. Denken Sie auch an die letzten beiden Elemente von `a`.

Lösungsvorschlag:

```java
/**
 * Creates an array by removing every third element from the parameter.
 *
 * @param a the array, where every third element should be removed
 * @return the array, where every third element is removed
   */
public int[] foo(int[] a) {
  // Determine length
  int length = (int) (a.length - Math.round(a.length / 3.0));
  int[] b = new int[length];

  // Copy elements
  for (int i = 0, j = 0; i < a.length; i++) {
    // Skip every third element
    if (i % 3 == 0) {
      continue;
    } else {
      b[j] = a[i];
      j++;
    }
  }
  return b;
}
```