Lehrstuhl für Angewandte Informatik I
Prof. Dr. Bernhard Westfechtel
Sandra Greiner

WS 2017/18
11. Januar 2018

Konzepte der Programmierung

Lösungsskizzen – Übungsblatt 11

Aufgabe 11.1

s. e-Learning

Aufgabe 11.2 (Vererbungshierarchien und dynamische Typbestimmung)

(a) Vererbungshierarchie (ohne Dateinamen):

```
1 public interface Lebewesen {}
2 public abstract class Tier implements Lebewesen {}
3 public abstract class Mensch implements Lebewesen {}
4 public abstract class Säugetier extends Tier {}
5 public class Reptil extends Tier {}
6 public abstract class Haustier extends Säugetier {}
7 public class Hund extends Haustier {}
8 public class Schwein extends Haustier {}
```

(b) **Mehrfachvererbung** auf Klassenebene ist in Java nicht möglich, d.h. eine Klasse kann maximal eine Super-Klasse haben. Somit kann die Super-Klasse von `Schlange` nicht gleichzeitig `Reptil` und `Haustier` sein.
Ausblick: Man könnte den Sachverhalt aber auf die Mehrfachvererbung von **Interfaces** herunterbrechen, indem man `Haustier` als Interface definiert und die Schlage das Interface implementieren lässt.

(c) In Zeile 3 wird der Variablen `ht` vom Typ `Haustier` der Inhalt der Variablen `l` vom Typ `Lebewesen` zugewiesen. Die Substituierbarkeit ist nicht mehr gegeben, da `l` per se nicht typkompatibel ist mit `Haustier`. Die dynamischen Typen wären zwar kompatibel, jedoch kann der Compiler dies zur Übersetzungszeit nicht entscheiden. Durch eine **explizite Typverengung** könnte man dieses Problem wie folgt lösen:

```
3 Haustier ht = (Haustier) l;
```

Die Typumwandlung ist an dieser Stelle möglich, weil der dynamische Typ von `l` `Hund` und somit mit `Haustier` kompatibel ist. Der statische Typ, `Lebewesen`, ist an dieser Stelle überdeckt und daher nicht ausschlaggebend für die Umwandlung. Wäre er nicht überdeckt, würde die Typverengung zu einem Laufzeitfehler führen. Dies kann jedoch nicht geschehen, da weder `Lebewesen` noch `Haustier` instantiierbar sind.

(d) **Statischer** Typ: Durch **Deklaration** festgelegt. Dynamischer Typ: Durch die Klasse, aus der das aktuell referenzierte Objekt **instantiiert** wurde festgelegt.

- `s1`
  - (Z. 4): statisch `Säugetier`, dynamisch `Schwein`
  - (Z. 7): statisch `Säugetier`, dynamisch `Hund`
- `t` (Z. 2): statisch `Tier`, dynamisch `Reptil`

- `ht`

  (Z. 3): statisch `Haustier`, dynamisch `Hund`

  (Z. 7): statisch `Haustier`, dynamisch `Hund`

  (Z. 8): statisch `Haustier`, dynamisch `Hund`

  Statische und dynamische Typen spielen bei der Vererbung eine Rolle. Der statische Typ kann zur Laufzeit vom dynamischen Typ überdeckt werden.
  Bsp: Man hat eine Klasse `Hund`, die von der abstrakten Klasse Das Tier `t` hat durch die Zuweisung den dynamischen Typ `Hund`. Der statische Typ ist und bleibt `Tier`, er wird dabei temporär überdeckt.

Aufgabe 11.3

- Statisches und dynamisches Binden bezieht sich auf Methoden. Beim statischen Binden steht zur Übersetzungszeit fest, welche Methode ausgeführt wird. Beim dynamischen Binden geschieht dies zur Laufzeit und ist abhängig vom Typ des Objekts, auf dem sie aufgerufen werden. Dies ist bei der Redefinition von Methoden relevant. Bsp:

```
1  public class Tier{
2      public void sprich(){ Out.println("..."); }
3  }
4  public class Hund extends Tier{
5      public void sprich(){ Out.println("wau, wuff, knurr"); }
6  }
7  ...
8  public static void main(String[] args){
9      Tier t = new Hund();
10     t.sprich();
11 }
```

  Obwohl es sich bei `t` eigentlich um ein Tier handelt, ist die Methode `sprich()` in der Klasse `Hund` redefiniert. Da es sich bei `t` zur Laufzeit um einen Hund handelt, erscheint durch das dynamische Binden beim Aufruf der Text *"wau, wuff, knurr"* statt der drei Punkte, die beim statischen Binden erscheinen würden.

- Wird ein Feld oder eine Methode als `final` deklariert, sind keine Änderungen (Redefinition) mehr möglich. Bei `final` Feldern handelt es sich daher um Konstanten, die nach der ersten Wertzuweisung nicht veränderbar sind. Bei final-Methoden sind keine Redefinitionen in erbenden Klassen erlaubt.

  Beispiele:

```
1  public class MathUtil {
2      public static final double PI = 3.4;
3  }
```

  Der Wert von π ist konstant und wird sich nicht mehr ändern. Ein anderes Beispiel sind String-Konstanten oder Ungenauigkeitswerte.
  Ein Beispiel für eine `final`-Methode ist im folgenden Beispiel gegeben. Die Methode `gibGroesse()` soll sich egal für welche Listenvariante immer gleich verhalten, daher wird sie als `final` deklariert. Eine Redefinition ist somit nicht mehr möglich.

```java
public abstract class Liste{
    private int groesse;
    public final int gibGroesse() { return groesse; }
}
```

Aufgabe 11.4 (Java-Klassen zur Ein- und Ausgabe)

Writer Abstrakte Klasse zum Schreiben von Characters.

- **OutputStreamWriter** Klasse, die Character-Zeichen in Bytes konvertiert und in einen Byte-Stream schreibt.
  - **FileWriter** Unterklasse von *OutputStreamWriter*, welche die Daten in eine Datei schreibt.
- **BufferedWriter** Character-Writer mit Buffer-Funktionalität

Reader Abstrakte Klasse zum Lesen von Characters.

- **InputStreamReader** Klasse, welche Bytes aus einem Byte-Stream liest und Character-Zeichen zurückliefert.
  - **FileReader** Unterklasse von *InputStreamReader* welche die Daten aus einer Datei liest.
- **BufferedReader** Character-Writer mit Buffer-Funktionalität

OutputStream Abstrakte Klasse zum Schreiben von Byte-Werten.

- **FileOutputStream** Klasse, um Byte-Streams in Dateien zu schreiben. Anwendung: Bild-Daten

InputStream Abstrakte Klasse zum Lesen von Byte-Werten.

- **FileInputStream** Klasse, um Byte-Streams aus Dateien zu lesen.