Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
11. Dezember 2017

8. Übung zur Vorlesung

Einführung in die Programmierung

A8-1 ***Vererbung*** Überlegen Sie sich ohne Hilfe eines Computers, welche Ausgabe von folgendem Programm erzeugt wird:

```
public class A {
  public int f(int i){ return i+1; }
  public int f(String s){ return 7;}
  public int g() { return h()+f(0);}
  public int h() { return 20; }
  public int h(Object o){return 15;}
}
```

```
public class B extends A {
  public int f(int i) { return i+2; }
  public int f(Object o){ return 99;}
  public int g(){return super.g()*2;}
  public int h(int x){return 30 + x;}
  public int h(String s){ return 52;}
}
```

```
public class ABC {
  public static void main(String[] args) {
  String s = "Inheritance";
    A a = new A();
    System.out.println("a.f(0) = " + a.f(0));
    System.out.println("a.g()  = " + a.g()) ;
    System.out.println("a.f(s) = " + a.f(s));
    System.out.println("a.h(s) = " + a.h(s));
    System.out.println("-----");
    a = new B();
    System.out.println("a.f(0) = " + a.f(0));
    System.out.println("a.g()  = " + a.g()) ;
    System.out.println("a.f(s) = " + a.f(s));
    System.out.println("a.h(s) = " + a.h(s));
    System.out.println("-----");
    B b = (B) a;
    System.out.println("b.f(0) = " + b.f(0));
    System.out.println("b.g()  = " + b.g()) ;
    System.out.println("b.f(s) = " + b.f(s));
    System.out.println("b.h(s) = " + b.h(s));
} }
```

A8-2 ***Vererbung vs. Aggregation*** Die beiden EiP-Teilnehmer Stella Schnitt und Erwin Erber wollen eine Verwaltung für einen Heimtier-Dienstleister in Java realisieren. Die Aufgabe haben sie aufgeteilt: Stella schreibt die Klassen zur Modellierung der Tiere, z.B. Katzen und Hunde; während Erwin Klassen zur Verwaltung der angebotenen Dienstleistung schreibt, z.B. Katzenpension und Hundeschule. Natürlich soll man zu einer Hundeschule ausschließlich Hunde anmelden können, etc. Bis jetzt haben die beiden folgenden Code geschrieben:

```
public interface Tier {
    String gibLaut();
}

public class Katze implements Tier {
    @Override
    public String gibLaut() {
        return "Miau!";
}    }

public class Hund implements Tier {
    @Override
    public String gibLaut() {
        return "Wau! Wau!";
}   }
```

```
import java.util.ArrayList;

public class Hundeschule extends ArrayList<Tier>
    @Override
    public boolean add(Tier t){
        if (t.gibLaut().equals("Wau! Wau!")) {
            return super.add(t);
        } else {
            return false;
    }   }
    public void presentation() {
       for (Tier t : this) {
          System.out.println(t.gibLaut());
}   }   }
```

Ein erster Test schlägt jedoch leider fehl:

```
public static void main(String[] args) {
  ArrayList<Tier> tiere = new ArrayList<Tier>();
  tiere.add(new Katze());
  tiere.add(new Hund());
  Hundeschule hundeschule = new Hundeschule();
  hundeschule.add(new Hund());
  hundeschule.add(new Katze());
  hundeschule.addAll(tiere);
  hundeschule.presentation():
} }
```

Ausgabe:

```
Wau! Wau!
Miau!
Wau! Wau!
```

a) Skizzieren Sie `Tier`, `Katze`, `Hund`, `Hundeschule`, `ArrayList` mit einem UML Klassendiagramm. Beschränken Sie sich auf die für Erwin und Stella relevanten Dinge.

b) Überlegen Sie sich kurz, wie das Test-Programm abläuft und warum sich in diesem Test eine `Katze` erfolgreich in eine `Hundeschule` einschmuggeln konnte!

c) Der Hunde-Test mittels String-Vergleich und `gibLaut()` in der Klasse `Hundeschule` ist sehr dilettantisch und fehleranfällig: Was ist, wenn wir später weitere Unterklassen von `Hund` bekommen, welche etwas anders bellen, z.B.:

```
public class Chihuahua extends Hund {
  @Override public String gibLaut() { return "Wiff! Wiff!"; }
}
```

In der Vorlesung wurde eine Möglichkeit vorgestellt, wie man den Subtyp einer Variable bestimmen kann. Verwenden Sie dies, um die `add`-Methode in der Klasse `Hundeschule` zu verbessern!
Hinweis: Dies löst aber noch nicht das ursprüngliche Problem! (Warum?)

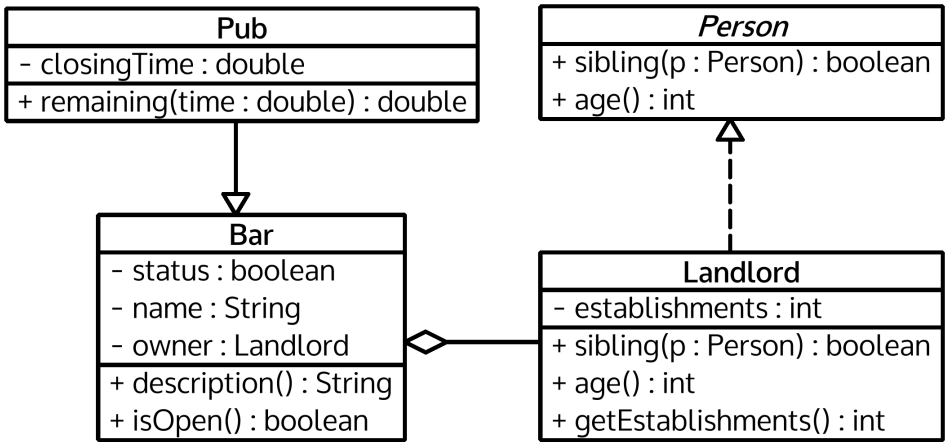
d) Erwin dachte fälschlicherweise, seine Aufgabe schnell durch Vererbung lösen können. Dies war hier jedoch völlig fehl am Platze und führte zu Problemen. Es ist oft problematisch von Klassen zu erben, welche darauf nicht vorbereitet wurden und deren Dokumentation nicht speziell das Erstellen von Unterklassen erklärt. Wenn z.B. eine neue Version der fremden Oberklasse plötzlich neue Methoden mitbringt, können solche Probleme wie in der vorangegangenen Teilaufgabe entstehen; dies führte auch in der Realität schon zu Sicherheitslücken[1].

Machen Sie es nun richtig: Eine Hundeschule *ist keine* Liste, sondern eine Hundeschule *besitzt* eine interne Liste! Implementieren Sie die Klasse `Hundeschule` neu, so dass im obigen Test nur noch bellen zu vernehmen und auch noch folgendes Interface implementiert wird:

```
public interface TierSchule {
    boolean add(Tier tier);
    boolean addAll(ArrayList<Tier> tiere);
    boolean remove(Tier tier);
    ArrayList<Tier> getTiere();
}
```

e) Erklären Sie in 2–4 Sätzen, warum das Problem aus der ersten Teilaufgabe nach dieser Änderung nun nicht mehr auftreten kann; insbesondere auch dann, falls später einmal `ArrayList` in einer neuen Version weitere Methoden zum Hinzufügen von Elementen mitbringen würde.

H8-1 ***UML Klassendiagramm*** (6 Punkte; Abgabe: `H8-1.txt` oder `H8-1.pdf`)
Gegeben ist folgendes UML Klassendiagramm:



Geben Sie eine möglichst vollständige, passende Implementierung an! Geben Sie aber keine Konstruktoren und keine Methodenrümpfe an!

Hinweis: Klassen brauchen nicht als `abstract` deklariert werden.

[1] Siehe z.B. Kapitel 4, Item 16 in Joshua Bloch, "Effective Java", 2nd Ed., Addison Wesley

H8-2 ***Schnittstellen*** (6 Punkte; Verzeichnis `Flaechen`)
In dieser Aufgabe wollen wir uns mit Schnittstellen (Interfaces) beschäftigen. Der Aufgabe sollten 2 Dateien beiliegen: `Demo.java` zur Demonstration. Ein Interface `HatFlaeche.java` für alle Dinge mit einer Fläche (vgl. Folie 7.26). Wenn Sie alle Teilaufgaben richtig bearbeitet haben, so sollte sich `Demo.java` kompilieren lassen und folgende Ausgabe erzeugen:

```
a)
Es gibt folgende Dinge:
  * Quadrat mit den Eckpunkt (4.0,7.0) und Größe 3.0
  * Kreis mit den Mittelpunkt (3.0,3.0) und Radius 3.0
  * Quadrat mit den Eckpunkt (3.0,3.0) und Größe 6.0
  * Kreis mit den Mittelpunkt (2.0,2.0) und Radius 2.0
Mit Gesamtflaeche: 85.9682
b)
Es gibt folgende Dinge:
  * Quadrat mit den Eckpunkt (4.0,7.0) und Größe 3.0
  * Kreis mit den Mittelpunkt (2.0,2.0) und Radius 2.0
Mit Gesamtflaeche: 21.605600000000003
c)
Es gibt folgende Dinge:
  * Kreis mit den Mittelpunkt (3.0,3.0) und Radius 3.0
  * Quadrat mit den Eckpunkt (3.0,3.0) und Größe 6.0
Mit Gesamtflaeche: 64.3626
```

In Ihrer Abgabe im Verzeichnis `Flaechen` sollten am Ende 7 Dateien sein: `Quadrat.java`, `Kreis.java`, `Test.java`, `KleinerTest.java`, `Negation.java`, `Demo.java` (von Ihnen editiert), `HatFlaeche.java` (unverändert).

a) Implementieren Sie die Klassen `Quadrat` und `Kreis`! Diese müssen das vorgegebene Interface `HatFlaeche` implementieren und einen passenden Konstruktor bieten, wie im vorhandenen Code von `Demo.java` verwendet.

b) Schreiben Sie ein funktionales Interface `Test`, welches eine Methode `triffzu` bietet. Diese Methode bekommt ein Objekt mit Flächeneigenschaft und liefert eine Entscheidung darüber.

c) Vervollständigen Sie die Methode `filter` in der Klasse `Demo`, so dass diese Klasse eine `ArrayList` zurück liefert, welche nur die Dinge mit Fläche enthält, welche den übergebenen Test bestanden haben.

d) Entwerfen Sie eine Klasse `KleinerTest`, welche das Interface `Test` implementiert, und Gegenstände akzeptiert, deren Fläche (strikt) kleiner ist als ein gewisser Wert, der dem Konstruktor übergeben wurde.

e) Schreiben Sie ein Klasse `Negation`, welche das Interface `Test` implementiert, und einen im Konstruktor übergebenen anderen `Test` negiert.

Abgabe: Lösungen zu den Hausaufgaben können bis Sonntag, den 17.12.17, mit UniWorX nur als `.zip` abgegeben werden. Aufgrund des Klausurbonus müssen die Hausaufgaben von Ihnen alleine gelöst werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen. Bitte beachten Sie auch die Hinweise zum Übungsbetrieb auf der Vorlesungshomepage (`www.tcs.ifi.lmu.de/lehre/ws-2017-18/eip/`).