Vererbung Grundlagen

Lernziele

- Sie verstehen die Zuweisungsregeln im Zusammenhang mit Vererbung (Subtyping) und können diese richtig anwenden.
- Sie können für eine einfache Problemstellung eine geeignete Vererbungshierarchie konzipieren und diese auch umsetzen.
- Sie können in einem bestehenden Programm ein schlechtes Klassendesign bezüglich Vererbung erkennen, darauf basierend ein besseres Design ausarbeiten und das Programm entsprechend anpassen.

Aufgabe 1 (auf Papier)

Im Zusammenhang mit Vererbung ist es wichtig, das Subtyping genau zu verstehen, d.h. welche Objekte welchen Variablen zugewiesen bzw. bei welchen Parametern verwendet werden dürfen. Eng damit verbunden ist das Verständnis, welche Methoden von einer Objektvariablen aufgerufen werden dürfen und was das Casting von Objektvariablen in diesem Zusammenhang für eine Rolle spielt. Diese Aufgabe bietet Ihnen die Möglichkeit, Ihre Kenntnisse diesbezüglich zu überprüfen. Studieren Sie zuerst die vier nachfolgend vorgegebenen Klassen, damit Sie deren Zusammenhang verstehen.

```
public class Sportler {
  private int alter;

  public Sportler (int alter) {
    this.alter = alter;
  }

  public int gibAlter() {
    return alter;
  }

  public boolean istAelterAls(Sportler andererSportler) {
    if (alter > andererSportler.alter) {
      return true;
    } else {
      return false;
    }
  }
}
```

```
public class Tennisspieler extends Sportler {
  private int ranking;

  public Tennisspieler(int alter, int ranking) {
    super(alter);
```

```
    this.ranking = ranking;
  }

  public int gibRanking() {
    return ranking;
  }

  public boolean istBesserKlassiertAls(
          Tennisspieler andererTennisspieler) {
    if (ranking < andererTennisspieler.ranking) {
      return true;
    } else {
      return false;
    }
  }
}

public class Leichtathlet extends Sportler {
  public Leichtathlet(int alter) {
    super(alter);
  }
}

public class Sprinter extends Leichtathlet {
  private double zeit;

  public Sprinter(int alter, double zeit) {
    super(alter);
    this.zeit = zeit;
  }

  public void sprinte() {
    System.out.println("Lockere " + zeit + " Sekunden!");
  }
}
```

Gehen Sie nun durch alle Anweisungen in der folgenden Methode `test()` und überlegen Sie sich, welche Zeilen nicht zulässig sind und markieren Sie diese mit „Kompilierfehler“ oder „Laufzeitfehler“. Bitte beachten Sie, dass dieser Code natürlich auch unschön ist, weil die Rückgabewerte der Methoden nicht verarbeitet werden, darüber wollen wir in dieser Aufgabe aber hinwegsehen.

```
public void test()
{
  Tennisspieler ivan = new Tennisspieler(30, 4); korrekt
  Tennisspieler john = new Tennisspieler(28, 2); korrekt
  Tennisspieler pete = new Tennisspieler(33, 1); korrekt

  Sprinter carl = new Sprinter(26, 9.83); korrekt
  Sprinter usain = new Sprinter(25, 9.58);korrekt
  Sprinter asafa = new Sprinter(29, 9.71);korrekt

  Leichtathlet athlet;korrekt, obwohl nicht Clean Code (Zugriffsmodifikator fehlt)
  Sportler sportler;  korrekt, obwohl nicht Clean Code (Zugriffsmodifikator fehlt)
  Tennisspieler bjoern; korrekt, obwohl nicht Clean Code (Zugriffsmodifikator fehlt)

  carl.gibAlter();korrekt (Methoden von Superklasse kann aufgerufen werden)
  john.gibAlter();korrekt (Methoden von Superklasse kann aufgerufen werden)
  athlet = asafa; korrekt, da Subklasse einer Superklasse zugewiesen werden kann
  asafa.gibAlter();korrekt
  athlet.gibAlter();korrekt
  athlet.sprinte(); Kompilierfehler, da Methode nicht aufgerufen werden kann
  ((Sprinter) athlet).sprinte(); korrekt

  Object obj = ivan;korrekt
  obj.equals(pete);korrekt (jedoch ist der Returnwert false)
  obj.equals(carl); korrekt (jedoch ist der Returnwert false)
  ivan.equals(pete); korrekt (jedoch ist der Returnwert false)
  bjoern = obj;Kompilierfehler
  bjoern = (Tennisspieler) obj;korrekt
  sportler = (Tennisspieler) obj; korrekt

  ivan.istAelterAls(john); korrekt
  ivan.istAelterAls(asafa);korrekt
  ivan.istBesserKlassiertAls(pete);korrekt
  ivan.istBesserKlassiertAls(usain); Kompilierfehler
  ivan.istBesserKlassiertAls((Tennisspieler) usain); Laufzeitfehler
  sportler = usain; korrekt
  ivan.istBesserKlassiertAls((Tennisspieler) sportler); Laufzeitfehler

  athlet = carl;korrekt
  sportler = john; korrekt
  sportler.istAelterAls(athlet); korrekt (jedoch ist der Returnwert false)
  athlet.istAelterAls(sportler); korrekt
}
```

Aufgabe 2

Forken Sie für diese Aufgabe das Projekt https://github.engineering.zhaw.ch/prog1-kurs/08_Praktikum_Hochschule. Nutzen Sie Eclipse um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.

An einer Hochschule sollen eine gewisse Zahl von Studierenden jeweils von einem Dozierenden betreut werden. Dazu soll ein kleines Programm entwickelt werden. Die Klasse `Betreuungsverhaeltnis` ist vorgegeben. Ihre Aufgabe ist es, die von dieser Klasse benötigten Klassen `Dozent` und `Student` zu entwickeln, damit das Programm korrekt funktioniert. Die Klassen sind wie folgt definiert:

- Ein Dozierender hat einen Namen, eine ID, eine Büronummer und eine Telefonnummer (alles Strings). Eine Methode `gibInfo` liefert den Namen und die ID (z.B. „Albert Einstein, ID 1234-5678“) und zwei Methoden `gibBuero` und `gibTelefonnummer` liefern die Büro- und Telefonnummer.
- Ein Studierender hat einen Namen, eine ID (beides Strings) und eine Anzahl Credits, die er bisher erreicht hat (int). Eine Methode `gibInfo` liefert den Namen und die ID, eine Methode `gibCredits` liefert die Anzahl erworbener Credits und eine Methode `erhoeheCredits` erhöht die Credits um einen spezifizierten Wert.

Halten Sie sich an diese Vorgaben, damit die vorgegebene Klasse `Betreuungsverhaeltnis` mit Ihren Klassen funktioniert. Studieren Sie ebenfalls die Klasse `Betreuungsverhaeltnis`, damit Sie genau verstehen, wie diese die Klassen `Dozent` und `Student` verwendet.

Überlegen Sie sich zuerst, wie Sie die Klassen `Student` und `Dozent` am besten implementieren, um Code Duplizierung zu vermeiden. Sie müssen die Gültigkeit der übergebenen Parameter nicht prüfen.

Testen können Sie entweder mit der vorgegebenen Klasse `Simulation` oder indem Sie eine eigene Klasse (mit main-Methode) schreiben.

Aufgabe 3

Forken Sie für diese Aufgabe das Projekt https://github.engineering.zhaw.ch/prog1-kurs/08_Praktikum_Fahrzeugverwaltung. Nutzen Sie Eclipse um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.

Das Projekt dient einem Fahrzeughändler, der Autos, Motorräder und Fahrräder verkauft, zur Verwaltung der Fahrzeuge. Die Klasse `Fahrzeugverwaltung` ist die zentrale Klasse, welche die Fahrzeuge verwaltet. Die Klasse `Simulation` (mit der main-Methode) dient dazu, automatisch Kunden und Fahrzeuge zu erzeugen und einige Verkäufe zu tätigen.

a) Studieren Sie das Programm. Führen Sie es auch aus um zu sehen, was bei der Ausführung der Simulation ausgegeben wird. Welche Probleme identifizieren Sie bei der Struktur des Programms?

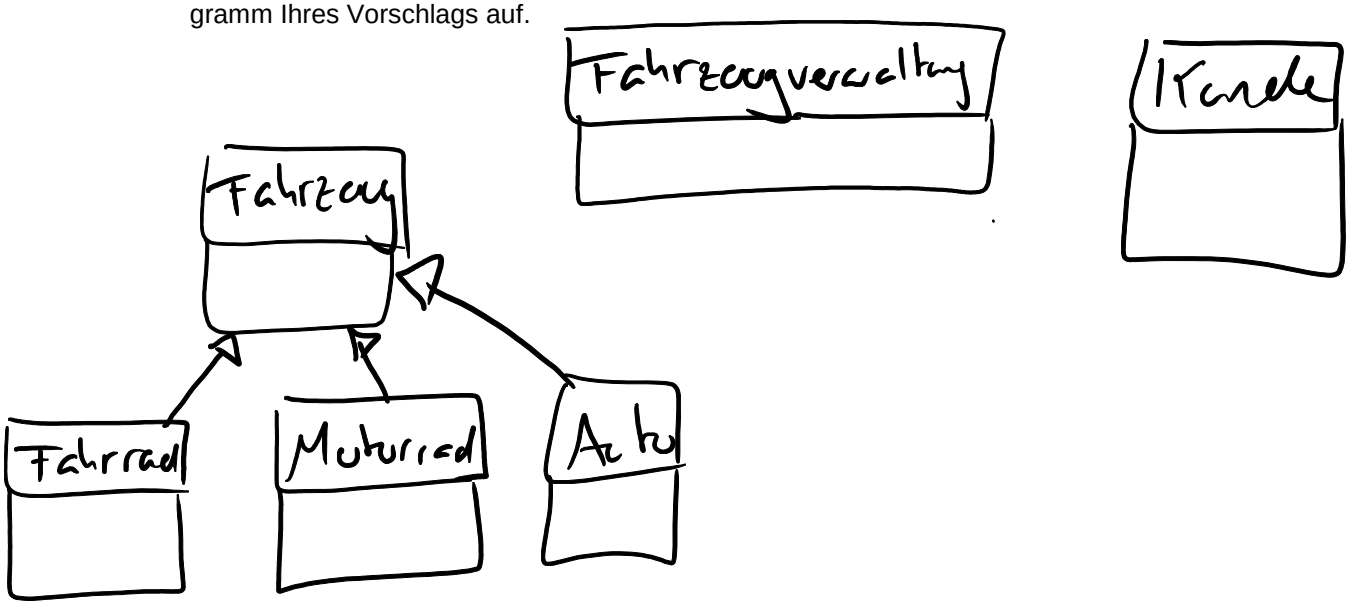
in allen "Fahrzeugsklassen" haben wir identische Datenfelder. Dies könnte vor allem im Falle einer Änderung der Datenfelder zu einer mühsamen und ineffizenten Arbeit werden.

Besser wäre es, wenn es eine superklasse gäbe, mit Subklassen

b) Nehmen Sie an, die Fahrzeuge hätten auch noch eine Farbe. Wie viele Klassen (ausser der Klasse `Simulation`) müssten Sie im aktuellen Design anpassen, damit dies möglich ist?

Wir müssten sämtliche vier "Fahrzeugklassen" + Fahrzeugverwaltung anpassen.

c) Verbessern Sie das Klassendesign unter Berücksichtigung der Vererbung. Überlegen Sie sich dazu zuerst eine geeignete Hierarchie für die Fahrzeugklassen und zeichnen Sie das Klassendiagramm Ihres Vorschlags auf.



d) Implementieren Sie Ihren verbesserten Vorschlag. Dazu kopieren Sie am besten das gesamte Projekt und nehmen dort die Änderungen vor. Fassen Sie momentan die Klassen `Simulation` und `Fahrzeugverwaltung` noch nicht an; Ihr Programm sollte nach den Anpassungen nach wie vor funktionieren. Testen Sie dies und prüfen Sie, ob bei der Durchführung der Simulation immer noch dieselben Ergebnisse ausgegeben werden.

e) Passen Sie nun auch noch die Klasse `Fahrzeugverwaltung` an, damit nur noch eine kaufe-Methode vorhanden ist und nur noch eine Datenstruktur für die Verwaltung der Fahrzeuge verwendet wird. Vermutlich müssen Sie dann auch noch eine kleine Anpassung in der Klasse `Simulation` vornehmen. Prüfen Sie wiederum, ob die Simulation immer noch dieselben Ergebnisse erzeugt.

f) Um den Bogen zu Teilaufgabe b) zu machen: Wie viele Klassen müssten Sie jetzt anpassen, wenn die Fahrzeuge auch eine Farbe haben sollten?

Nun würde es reichen, wenn wir dies in zwei Klassen anpassen

(Fahrzeug und Fahrzeugverwaltung)

g) Nehmen Sie an, Sie müssten in der Klasse Fahrzeugverwaltung eine Methode implementieren, welche durch sämtliche verwalteten Fahrzeuge durchgeht und die Leistung derjenigen Fahrzeuge, die einen Motor haben, ausgibt. Wenn Sie nur die in Kapitel 8 kennengelernten Konstrukte verwenden dürfen, auf welche Probleme stossen Sie dabei?

Ich sehe die Problematik, dass nicht alle Fahrzeuge einen Motor haben und somit müssten wir durch jedes Objekt iterieren und diese wiedergibt, welche einen Motor haben. Dies könnte ziemlick "tricky" werden, um herauszufiltern, welche einen Motor haben und welchen nicht.

Hinweis: Mit den im Kapitel 8 kennengelernten Konstrukten können Sie diese Problemstellung tatsächlich noch nicht befriedigend lösen. Im 9. Kapitel werden Sie geeignete Konstrukte kennenlernen.