Weitere Techniken zur Abstraktion

Kapitel 10

Programmieren 1

Übersicht

Konzepte:

- Abstrakte Klassen, Interfaces, multiple Vererbung

Java-Konstrukte:

- `abstract`, `implements`, `interface`

Lernziele:

- Sie wissen, was abstrakte Klassen und Interfaces sind und kennen deren Einsatzgebiete
- Sie können abstrakte Klassen und Interfaces einsetzen um einfache Problemstellungen geeignet zu abstrahieren
- Sie können erklären, wie multiple Vererbung in Java funktioniert

Übersicht

Programm

- Auswertung der Lesekontrollfragen 15‘
- Fragen und Diskussion von Beispielen I 30‘

Pause

- Fragen und Diskussion von Beispielen II 40‘
- Aufträge für das nächste Plenum 5’

Aufgaben

Aufgabe 01-1 – Beam me up, Scotty!

Zürcher Hochschule für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

Überprüfen Sie Ihr Verständnis bez. abstrakter Klassen

1. **Einzelarbeit (3‘ + 3‘)**
   - Studieren Sie den nebenstehenden Code und notieren und korrigieren Sie die darin enthaltenen Fehler und Unschönheiten
   - Evtl. müssen Sie auch zusätzliche Methoden schreiben

   Bemerkung: Die javadoc Kommentare müssen Sie nicht verbessern…

```
/**Diese abstrakte Klasse modelliert einen Energieschild
 * mit einer bei Erzeugung spezifizierten maximalen
 * Energie. Schilde absorbieren Energie, die auf sie
 * trifft. Das Absorptionsverhalten wird durch den
 * jeweiligen konkreten Schildtyp definiert. */
public class Schild
{
  protected int maxEnergie;
  protected int energie;

  /**Erzeugt einen Schild mit der angegebenen maximalen
   * Energie und setzt die Anfangsenergie auf das Max.*/
  public Schild(int maxEnergie) {
    if( maxEnergie < 0 ) {
      throw new IllegalArgumentException(
                  "Maximale Energie muss >= 0 sein");
    }
    this.maxEnergie = maxEnergie;
    this.energie = maxEnergie;
  }
  /**@return true, falls Schild verbraucht */
  public boolean istVerbraucht() {
    return energie <= 0;
  }
  /**Der Schild absorbiert die gegebene Energie gemäss
   * seinem Schadensmodell und meldet zurück, wie viel
   * Energie der Schild nicht absorbieren konnte. */
  public abstract int absorbieren(int energie) {
    return 0;
  }
}
```

Aufgabe 01-2 – Beam me up, Scotty!

Überprüfen Sie Ihr Verständnis für die Implementation von Klassen

1. **Zu Zweit (mit SitznachbarIn) (8‘ + 5‘)**
   - Implementieren Sie Transphasen-schilde gemäss dieser Spezifikation:
     - Ein Transphasenschild ist ein Schild
     - Transphasenschilder haben eine maximale Stärke von 10'000
     - Transphasenschilde setzen das folgende Schadensmodell um:
       - Falls die Energie des Schildes noch grösser als 0 ist, absorbiert der Schild die gesamte Energie, wobei die Schildenergie um einen fixen Wert *schaden* (wird dem Konstruktor übergeben) reduziert wird. Die Schildenergie soll dabei nicht unter 0 sinken
       - Andernfalls wird keine Energie absorbiert

```
/**Diese abstrakte Klasse modelliert einen Energieschild
 * mit einer bei Erzeugung spezifizierten maximalen
 * Energie. Schilde absorbieren Energie, die auf sie
 * trifft. Das Absorptionsverhalten wird durch den
 * jeweiligen konkreten Schildtyp definiert. */
public abstract class Schild
{
  private final int maxEnergie;
  private int energie;

  /**Erzeugt einen Schild mit der angegebenen maximalen
   * Energie und setzt die Anfangsenergie auf das Max.*/
  public Schild(int maxEnergie) {
     …
  }
  /**@return Die verbleibende Energie des Schildes */
  protected int getEnergie() {
    return energie;
  }
  /**Setzt die verbleibende Energie des Schildes auf den
   * gegebenen Wert.
   */
  protected void setEnergie(int energie) {
    this.energie = energie;
  }
  /**@return true, falls Schild verbraucht */
  public boolean istVerbraucht() {
    return energie <= 0;
  }
  /**Der Schild absorbiert die gegebene Energie gemäss
   * seinem Schadensmodell und meldet zurück, wie viel
   * Energie der Schild nicht absorbieren konnte. */
  public abstract int absorbieren(int energie);
}
```

Aufgabe 01-3 – Beam me up, Scotty!

Überprüfen Sie Ihr Verständnis für die Implementation von Klassen

1. **Zu Zweit (mit SitznachbarIn) (8‘ + 3‘)**
   - Studieren Sie den nebenstehenden Code und erweitern Sie diesen so, dass folgende Spezifikation erfüllt ist:
     - Raumschiffe können auch als Objekte vom Typ Zerstörbar betrachtet werden
     - Die Panzerung wird bei einem Angriff (Aufruf der Methode `zerstoeren`) um die Angriffsstärke reduziert. Die Panzerung soll nicht kleiner als 0 werden. Sinkt die Panzerung auf 0, ist das Raumschiff zerstört
     - Es sollen keine Objekte vom Typ Raumschiff erzeugt werden können

```
/**Die Klasse modelliert ein Raumschiff. Raumschiffe
 * haben eine Panzerung, die zu Beginn den Wert
 * {@value #PANZERUNG} hat. */
public class Raumschiff {
  private static final int PANZERUNG = 10000;
  private int panzerung;

  /** Erzeugt ein Raumschiff. */
  public Raumschiff() {
    this.panzerung = PANZERUNG;
  }
}

public interface Zerstoerbar
{
  /**Fuehrt einen Angriff mit der angegebenen Staerke
   * durch. Der Zustand des angegriffenen Objektes
   * wird gemaess seinem Schadensmodell entsprechend
   * angepasst. Wird das Objekt durch den Angriff
   * zuerstoert, wird dies zurueckgemeldet.
   * @param staerke Staerke des Angriffs
   * @return true, falls das Objekt zerstoert wurde
   */
   public boolean zerstoeren(int staerke);
}
```

Aufgabe 01-4 – Beam me up, Scotty!

Überprüfen Sie Ihr Verständnis für die Implementation von Klassen

1. **Zu Zweit (mit SitznachbarIn) (12‘ + 6‘)**
   - Implementieren Sie die Klasse Kampfstern:
     - Kampfsterne sind Raumschiffe
     - Kampfsterne haben einen Namen und können mit einer beliebigen Anzahl Schilde ausgestattet werden
     - Nach ihrer Erzeugung haben sie den bei der Erzeugung angegebenen Namen, besitzen jedoch noch keine Schilde. Schilde können aber hinzugefügt werden
   - Das Verhalten bei einem Angriff auf einen Kampfstern verändert sich im Vergleich zum Raumschiff wie folgt:
     - Wird ein Kampfstern angegriffen, werden zuerst die Schilde benutzt, um den Angriff zu absorbieren
     - Erst wenn alle Schilde verbraucht sind, wirkt der Angriff auf die normale Panzerung des Kampfsterns
     - Ist ein Schild verbraucht, ist dieser zu entfernen

Aufgabe 01-4 – Beam me up, Scotty!


```
public abstract class Schild
{
  private final int maxEnergie;
  private int energie;

  /**Erzeugt einen Schild mit der angegebenen
   * maximalen Energie und setzt die
   * Anfangsenergie auf das Max.*/
  public Schild(int maxEnergie) {
     …
  }

  /**@return Die verbleibende Energie des
   *Schildes */
  protected int getEnergie() {
    return energie;
  }

  /**Setzt die verbleibende Energie des Schildes
   * auf den gegebenen Wert. */
  protected void setEnergie(int energie) {
    this.energie = energie;
  }

  /**@return true, falls Schild verbraucht */
  public boolean istVerbraucht() {
    return energie <= 0;
  }

  /**Der Schild absorbiert die gegebene Energie
   * gemäss seinem Schadensmodell und meldet
   * zurück, wie viel Energie der Schild nicht
   * absorbieren konnte. */
  public abstract int absorbieren(int energie);
}
```

```
public abstract class Raumschiff
                      implements Zerstoerbar {
  private static final int PANZERUNG = 10000;
  private int panzerung;

  /** Erzeugt ein Raumschiff. */
  public Raumschiff() {
    this.panzerung = PANZERUNG;
  }

  @Override
  public boolean zerstoeren(int staerke) {
    panzerung =
           Math.max(panzerung - staerke, 0);
    return panzerung <= 0;
  }
}
```

Aufgabe 02 – Klassendiagramm

Überprüfen Sie Ihr Verständnis für die Nutzung der von Vererbungshierarchien, abstrakten Klassen und Interfaces für ein gutes Klassendesign

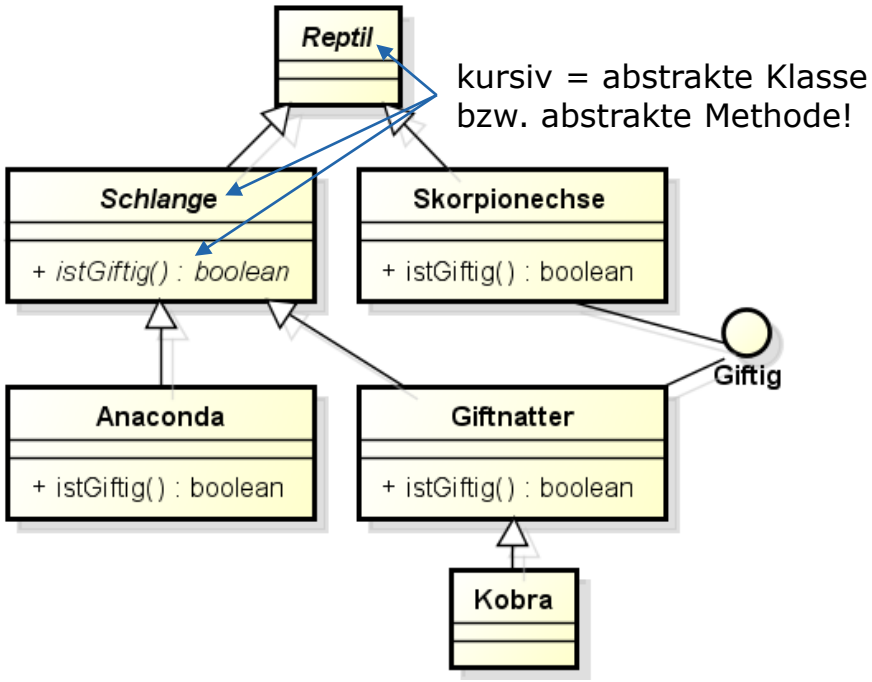
1. **In Zweierteams (10')**
   - Skizzieren Sie das Klassendiagramm für eine Anwendung wie nebenstehend beschrieben. Notieren Sie die Signaturen von für das Design zentralen Methoden.
2. **Zwei Zweierteams (5')**
   - Setzen Sie sich mit einem anderen Zweierteam zusammen und diskutieren Sie Ihre Klassendiagramme. Was ist gleich, was ist anders? Wieso haben Sie das so entworfen und nicht Anders?

- Es gibt verschiedene Tiere: Hunde, Katzen, Kühe und Hühner.
- Tiere können mit einem der folgenden Futtertypen gefüttert werden: Fleisch, Getreide, Gras. Ein Tier kann mit unpassendem Futter gefüttert werden, es wird dann einfach nichts essen.
- Tiere machen abhängig vom konkreten Tier zum Tier passende Geräusche.
- Speziell trainierte Tiere (z.B. Rennhund, Rennkuh etc.) können für Rennen verwendet werden. Ein solches Exemplar kann als Objekt vom Typ Sprinter betrachtet werden.

Aufgabe 03 – Polymorphie

Überprüfen Sie Ihr Verständnis für Polymorphie

1. **Im Plenum (20')**
   - Welche Zeilen sind OK? Wo gibt es Kompilier- oder Laufzeitfehler?
   - Zeilen mit Fehlern können für die folgenden Zeilen ignoriert werden
   - Das Interface `Giftig` beinhaltet die Methode `istGiftig()`




```
Reptil reptil; Giftnatter giftnatter;
Giftig giftig; Schlange schlange;
schlange = new Giftnatter();
giftig = new Giftnatter();
reptil = giftig;
reptil = (Schlange) giftig
reptil = (Reptil) giftig;
giftnatter = giftig;
giftig = new Skorpionechse();
schlange = (Schlange) giftig;
Reptil reptil2 = schlange;
giftnatter = (Giftnatter) new Anaconda();
schlange = new Kobra();
giftig = (Giftig) schlange;
giftnatter = (Giftig) schlange;
giftnatter = schlange;
schlange = new Schlange();
giftig.istGiftig();
reptil.istGiftig();
schlange = new Anaconda();
schlange.istGiftig();
((Giftig) schlange).istGiftig();
giftig = (Giftig) new Anaconda();
```

Aufgabe 04-1 – Klassendesign

Überprüfen Sie Ihr Verständnis für ein gutes Klassendesign

1. **Im Plenum (5‘)**
   - Ziel ist es, das nebenstehende Codestück ausführen zu können
   - Studieren Sie den Code auf der nächsten Slide und diskutieren Sie, was am Klassendesign unschön ist und welche Anpassungen nötig sind, damit der nebenstehende Code funktioniert

   Bemerkung: javadoc Kommentare sind aus Platzgründen verkürzt oder gar nicht dargestellt. Diese müssen Sie nicht verbessern.

```
public class TesteFiguren {

  public static void main(String[] args) {
    List<Figur> figuren =
                    new ArrayList<Figur>();
    figuren.add(new Kreis(10));
    figuren.add(new Quadrat(80));
    figuren.add(new Kreis(100));

    for(Figur figur : figuren) {
      System.out.println(
                      figur.gibFlaeche());
    }
  }
}
```

Aufgabe 04-1 – Klassendesign


```
public class Figur {
  private final Point position =
                      new Point(0,0);

  /**Setzt die Position der Figur auf den
   * spezifizierten Wert.
   * @param x Die x-Koordinate der Position
   * @param y Die y-Koordinate der Position */
  public void setzePosition(int x, int y) {
    position.x = x;
    position.y = y;
  }
  /**@return Die Flaeche der Figur */
  public double gibFlaeche() {
    return 0;
  }
}

public class Kreis extends Figur {
  private final int radius;

  /**Erzeugt einen Kreis mit Radius radius */
  public Kreis(int radius) {
    this.radius = radius;
  }
  @Override
  /**@return Die Flaeche der Figur */
  public double gibKreisflaeche() {
    return radius * radius * Math.PI;
  }
}
```

```
public class Quadrat extends Figur {
  private final int seitenlaenge;

  /**Erzeugt einen Kreis mit Radius radius */
  public Quadrat(int seitenlaenge) {
    this.seitenlaenge = seitenlaenge;
  }
  @Override
  /**@return Die Flaeche der Figur */
  public double gibQuadratflaeche() {
    return seitenlaenge * seitenlaenge;
  }
}

public class TesteFiguren {

  public static void main(String[] args) {
    List<Figur> figuren =
                      new ArrayList<Figur>();
    figuren.add(new Kreis(10));
    figuren.add(new Quadrat(80));
    figuren.add(new Kreis(100));

    for(Figur figur : figuren) {
      System.out.println(
                    figur.gibFlaeche());
    }
  }
}
```

Aufgabe 04-2 – Objekte Vergleichen

Überprüfen Sie Ihr Verständnis für Objektvergleiche und Interfaces

Ziel: Die Figuren aus 04-1 sollen in einer Liste gespeichert und anschliessend gemäss Ihrer Fläche sortiert werden können. Zum Sortieren soll die von der Klasse `Collections` angebotene statische Methode `sort()` verwendet werden.

1. **Zu Zweit (mit Sitznachbarln) (6‘ + 4‘)**
   - Studieren Sie den nebenstehenden Code. Beispiel 1 funktioniert wie erwartet und gibt die Werte sortiert aus
   - Der auskommentierte Aufruf von sort() führt zu einem Kompilierfehler, wenn er nicht auskommentiert ist
   - Was könnte der Grund hierfür sein? Macht das verhalten Sinn? Studieren Sie die API Dokumentation von sort()

```
//Beispiel 1
List<Integer> zahlen = new ArrayList<Integer>();
zahlen.add(100);
zahlen.add(500);
zahlen.add(200);

Collections.sort(zahlen);
for(Integer zahl : zahlen) {
  System.out.println(
    zahl.getClass().getSimpleName() + ": " + zahl);
}

//Beispiel 2
List<Figur> figuren = new ArrayList<Figur>();
figuren.add(new Kreis(100));
figuren.add(new Quadrat(10));
figuren.add(new Kreis(200));

//Collections.sort(figuren);
for(Figur figur : figuren) {
  System.out.println(
    figur.getClass().getSimpleName() + ": " +
    figur.gibFlaeche());
}
```

Aufgabe 04-3 – Objekte Vergleichen

Ziel: Die Figuren aus 04-1 sollen in einer Liste gespeichert und anschliessend gemäss Ihrer Fläche sortiert werden können. Zum Sortieren soll die von der Klasse `Collections` angebotene statische Methode `sort()` verwendet werden.

Lösungsansätze zur Verwendung von sort():

- `Comparable` Interface implementieren
- Objekt übergeben, das das `Comparator` Interface implementiert

1. **Einzelarbeit (ca. 15‘)**
   - Studieren Sie das Dokument 10_Objektvergleiche.pdf, welches sich auf OLAT im Verzeichnis der Vorlesungsunterlagen befindet
   - Welchen Lösungsansatz würden Sie aus welchem Grund für unser Problem der Sortierung von `Figur` Objekten verwenden?

Aufgabe 04-4 – Objekte Vergleichen

Überprüfen Sie Ihr Verständnis für die Implementation von Interfaces

1. **Zu Zweit (mit SitznachbarIn) (ca. 10‘)**
   - Bei der Ausführung des nebenstehenden Codes sollen die Figuren nun gemäss ihrer Fläche sortiert (kleinste Fläche zuerst) in der Liste drin stehen
   - Erweitern Sie den bisherigen Code um den hierfür notwendigen Code
   - Sie dürfen den nebenstehenden Code dazu nicht verändern!

```
List<Figur> figuren = new ArrayList<Figur>();
figuren.add(new Kreis(100));
figuren.add(new Quadrat(10));
figuren.add(new Kreis(200));

Collections.sort(figuren,
                 new ComparatorFigurflaeche());

for(Figur figur : figuren){
  System.out.println(
    figur.getClass().getSimpleName() + ": " +
    figur.gibFlaeche());
}
```