Javakurs FSS 2012

Lehrstuhl Stuckenschmidt

Tag 3 - Objektorientierung

Warum Objektorientierung

- Daten und Funktionen möglichst eng koppeln und nach außen kapseln
- Komplexität der Software besser modellieren
- Menschen denken objektorientiert
- Wiederverwendbarkeit von Softwaremodulen

Objekte

- haben eine Identität
- haben einen Zustand
- zeigen ein Verhalten

Klassen

- Beschreibung wie ein Objekt aussehen soll
- Name der "Top-Level-Klasse" muss gleich dem Dateinamen sein
- Besitzt:
  - Attribute
  - Operationen

Beispiel:

Klassenname	Punkt
Attribute	X- und y-Koordinate
Operationen	Position festlegen

Konstruktor

- Ähnlich einer Methode, Unterschiede sind:
  - Name des Konstruktors muss der Klassenname sein
  - Kein Rückgabewert
- Mehrere Konstruktoren für eine Klasse möglich

Konstruktor

```
class Point {
  int x;
  int y;

  point () {
  }

  point (int x, int y) {
    this.x = x;      //Zugriff auf die Objektvariable
    this.y = y;      //durch this
  }
}
```

Objekt erzeugen

- Ausdrückliche Erzeugung mit dem new-Operator
- new-Operator liefert dann eine Referenz auf das Objekt zurück
- Konstruktor wird ausgeführt

```
Point p;
p = new Point();
p = new Point(2, 4);
```

Objektvariablen und -methoden

- In einer Klasse deklarierte Variablen
- Zugriff darauf mit "."

Objektvariablen und -methoden

```
class Point {
  int x; //Objekvariable
  int y; //Objektvariable

  Point () {
  }

  void setPosition (int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

```
Point p = new Point();
p.x = 2;
p.y = 4;
p.setPosition(4, 2);
```

Referenzen

- *null* ist die Referenzvariable, die auf kein Objekt zeigt
- Auf ein Objekt kann mehrfach referenziert werden
- Bei Objekten werden nicht die Werte kopiert, sondern die Referenzen

Referenzen

```
Point p = null; //p referenziert auf kein Objekt
Point q = new Point(1, 2);
p = q; //Kopie der Referenz und nicht des Objekts

System.out.println(p.x); //1
q.x = 2;
System.out.println(p.x); //2
p.y = 3;
System.out.println(q.y); //3
```

Objekte vergleichen

- Zum Vergleich von Objekten wird die equals Methode verwendet
- Bei eigenen Klassen muss die Methode equals überschrieben werden um die Vergleichsfunktionalität einzubauen
- "==" vergleicht die Referenz des Objekts

```
String s1 = "abc"
System.out.println("Vergleich von def mit s1: " + s1.equals("def"));
//false
System.out.println("Vergleich von abc mit s1: " + s1.equals("abc"));
//true
```

equals

```
class Point {
  public int x;
  public int y;

  @Override
  public boolean equals (Object obj) {
    Point p;
    if (obj instanceof Point){
      p = (Point) obj;
    }
    else {
      return false;
    }
    if (p.x == x && p.y == y)
      return true;
    }
    return false;
  }
}
```

toString

- toString Methode wird verwendet um ein Objekt als String zu repräsentieren
- Bei eigenen Klassen muss die toString Methode überschrieben werden, wenn man die Funktionalität braucht

```
Point p = new Point(2, 3);
System.out.println(p);              //Point@3fbefab0
System.out.println(p.toString()); //Point@3fbefab0
```

toString

```
class Point {

  @Override
  public String toString() {
    String s = "X: " + x + " Y: " + y;
    return s;
  }

}
```

```
Point p = new Point(2, 3);
System.out.println(p);              //Point@3fbefab0
System.out.println(p.toString()); //X: 2 Y: 3
```

Pakete

- Zum Bündeln von Klassen
- Bildet eigenen Namensraum
- Können Unterpakete enthalten
- Im Dateisystem werden Pakete in gleichnamigen Ordnern abgelegt

```
package java;
package java.lang;
package java.awt;

package meinPaket;
```

import

- Zugriff auf Klassen aus anderen Paketen
- Importieren einzelner Klassen oder ganzer Pakete möglich
- Das Paket java.lang wird automatisch in jeder Klasse importiert

```
import java.text.Normalizer;
import java.text.*:
```

Sichtbarkeit

- Kein Schlüsselwort
  - Komponente ist nur innerhalb des Pakets bekannt
- private
  - Komponente ist nur innerhalb der Klasse bekannt
- protected
  - Komponente ist nur innerhalb des Pakts und in allen abgeleiteten Klassen bekannt
- public
  - Komponente ist überall bekannt

Getter und Setter

- Auf Variablen sollte in der Regel nicht direkt von anderen Klassen zugegriffen werden
- Deshalb sollten die meisten Variablen als private deklariert werden
- Zugriff von außen auf diese Variablen erfolgt über Methoden, sogenannte Getter und Setter

Getter und Setter

```
public class Point {
  private int x;
  private int y;

  public int getX() {
    return x;
  }
  public void setX(int i) {
    x = i;
  }
}
```

```
Point p = new Point(2, 3);
System.out.println(p.getX()); //2
p.setPoint(5);
System.out.println(p.getX()); //5
```

static

- Statische Methoden und Variablen gehören zu einer Klasse und sind nicht an ein Objekt gebunden
- Aufruf durch voranstellen des Klassennamens

- Verwendung von static an den ersten beiden Tagen, weil wir ohne Objekte gearbeitet haben

Klassenvariablen

- werden nur einmal angelegt
- können von allen Methoden der Klasse benutzt werden
- Veränderungen in einer Instanz betreffen alle Instanzen
- können als Instanzzähler verwendet werden

Instanzzähler

```
class Point {
  public static int counter;

  public Point() {
    counter++;
  }
}
```

```
Point p1 = new Point();
System.out.println(Point.counter); //1
Point p2 = new Point();
Point p3 = new Point();
System.out.println(Point.counter); //3
System.out.println(p1.counter); //3
System.out.println(p2.counter); //3
System.out.println(p3.counter); //3
```

Klassenmethoden

- Zugriff auf Instanzvariablen nicht möglich
- Besitzen keinen this-Zeiger
- Prominentes Beispiel sind die Methoden der Klasse System

Statische Konstruktoren

- Für komplexere Initialisierungen von Klassenvariablen
- Wird nur einmal beim ersten laden der Klasse ausgeführt und nicht beim instanziieren

```
class Point {
  static int x;
  static int y;

  static{
    x = 2
    y = x * 3;
  }
}
```

Vererbung

- Erfolgt mittels Subklasse *extends* Superklasse
- Alle sichtbaren Eigenschaften werden von der Superklasse auf die Subklasse übertragen
- Mehrfachvererbung ist nicht gültig

```
public class Subklasse extends Superklasse{
}
```

super

- Mit *super* kann auf die Oberklasse zugreifen
- Überschriebene Methoden der Oberklasse können mit einem Aufruf auf super ausgeführt werden

super

```
Public class Subclass extends Superclass{

        public Subclass(){
                super();
                superMethod(15);
        }

        @Override
        public void superMethod(int x){
                if(x > 10){
                        super.superMethod(x);
                }
        }
}
```

Abstrakte Klassen

- Modifizierer *abstract* wird der Klasse vorangestellt
- Nicht instanziierbar
- Allgemein gehaltene Oberklassen
- Unterscheidung in reine und partielle abstrakte Klassen
  - Rein: Die Klasse enthält ausschließlich abstrakte Methoden
  - Partiell: Die Klasse ist abstrakt, enthält aber konkrete Implementierungen

Abstrakte Methoden

- Modifizierer *abstract* wird der Methode vorangestellt
- Methoden geben lediglich die Signatur der Unterklasse vor
- Können nur in abstrakten Klassen vorkommen

Abstrakte Klassen und Methoden

```
abstract class Test {

  abstract boolean isOdd(int number);
}
```

```
abstract class Test {

  abstract boolean isOdd(int number);

  boolean isEven(int number){
    if (number%2 == 0) {
      return true;
    }
    else {
      return false;
    }
  }
}
```

Interfaces

- Schnittstellen
- Abart der abstrakten Klassen
- Enthält Datenelemente und abstrakte Methoden
- Eingesetzt für Mehrfachvererbung, weil Klassen beliebig viele Interfaces implementieren können

Interfaces

```
public interface MyInterface extends Interface1 {

	public static int x = 4;

	public abstract void method();
}
```

```
public class MyClass extends SuperClass implements
MyInterface, AnotherInterface {

	@Override
	public void method() {
	}
}
```

Fragen?

Vielen Dank für ihre Aufmerksamkeit!