Objektorientierung II

Javakurs 2009 - LE6

Florian Streibelt
Freitagsrunde

SoSe 2009

9. März 2010

Agenda

- Wiederholung zu Objekten
- Kapselung und Information Hiding im Detail
- Vergleich von Objekten
- Vererbung
- kleine API-Beispiele
- Objekte ausserhalb von Java

Was sind Objekte?

Was sind Objekte?

Was sind Objekte

Objekte ...

- sind Instanzen von Klassen
- sind zusammenhängende Einheiten von Attributen und Methoden
- führen logisch zusammenhängenden Code zusammen
- strukturieren meinen Code
- ermöglichen auch komplexen Code einfach zu verstehen
- erlauben das einfache Wiederverwenden von Code

Ein Objekt

```
1   class Human{
2
3     public String name;
4     public int age;
5
6     public Human(String name, int age){
7       this.name = name;
8       this.age  = age;
9     }
10
11    public void describe(){
12      System.out.println("Name:  "+this.name);
13      System.out.println("Alter: "+this.age);
14    }
15
16  }
```

Was bedeutet static?

Eine statische Methode, gekennzeichnet durch `static`, ist nicht an an konkretes Objekt gebunden sondern erfüllt eine Aufgabe die auf den Typ bezogen ist und kann ohne konkretes Objekt aufgerufen werden.
Beispiel: `int i = Integer.parseInt(String s);`
ist ungefähr so definiert:

```
1     public class Integer {
2       ...
3       public static int parseInt(String a){
4        ...
5        return i;
6       }
7       ...
8     }
```

For your eyes only...

Kapselung und Information Hiding...

Eine einfache Punktklasse

```
1     class Point{
2
3       public double dist;
4       public double angle;
5
6     }
```

Was passiert bei folgendem Programmfragment?

```
1     Point p = new Point();
2     p.dist=-12;
3     p.angle=1234;
```

Wie kann man so etwas verhindern?

Eine gekapselte Punktklasse

```
1   class Point{
2
3     private double dist;    // neu: nur noch aus
4     private double angle;   // Point zugreifbar!
5
6     public Point(double d, double a){
7      this.setAngle(a);
8      this.setDist(d);
9     }
10
11    public void setAngle(double a){
12      this.angle = a % 360;
13    }
14
15    public void setDist(double d){
16      if (d >= 0) {          // illegale Werte
17        this.dist = d;       // verhindern
18      }
19    }
20    //[...] (getAngle, getDist)
21  }
```

Was bedeutet Kapseln?

- Getter-/Setter-Methoden für alle Instanz-Variablen
- alle Instanz-Variablen sind 'private'
- interne Hilfs-Methoden werden 'private' deklariert

Welche Vorteile hat dieses Vorgehen?

$\Rightarrow$ Wir können uns an neue Anforderungen anpassen, z.B. wenn wir kartesische Koordinaten benötigen.

Information-Hiding

```
1
2         /*
3          * Hilfsmethode fuer kartesische Koordinaten
4          */
5         private void setXY(int x, int y){
6
7            this.dist = Math.sqrt(x*x + y*y )
8
9            if ( y >= 0){
10             this.angle= Math.acos(x/d);
11           }else{
12             this.angle= - Math.acos(x/d);
13           }
14
15        }
```

Information-Hiding

```
1        // [...]
2
3        public Point (int x, int y){
4          this.setXY(x,y);
5        }
6
7        public int getX(){
8         return this.dist * Math.cos(angle);
9        }
10       public int getY(){
11        return this.dist * Math.sin(angle);
12       }
13
14       public void setX(int x){
15         int y = this.getY();
16         this.setXY(x,y);
17       }
18       public void setY(int y){
19         int x = this.getX();
20         this.setXY(x,y);
21       }
```

Was haben wir erreicht?

- Zugriff nur noch über feste Schnittstellen
- wir können Parameter 'filtern'
- wie wir die Koordinaten intern speichern ist egal
- wir können nach aussen weitere Schnittstellen hinzufügen
- der Code ist damit flexibler einsetzbar

Schnittstelle/Interface unseres Objekts

```
1
2       public Point (double d, double a);
3       public Point (int x, int y);
4
5       public int setX (int x);
6       public int setY (int y);
7
8       public int getX ();
9       public int getY ();
10
11      public void setDist (double d);
12      public void setAngle (double a);
13
14      public double getDist ();
15      public double getAngle ();
```

Wir haben jetzt sogar zwei Konstruktoren:

```
1       Point p1 = new Point ( 0.79d, 2.83d);
2       Point p2 = new Point ( 2 , 2);
```

Java findet den passenden Konstruktor anhand der Parametertypen.

Kapselung und Information Hiding

Was bedeutet Kapselung und Information Hiding?

- verstecken interner Datenstrukturen
- klare Trennung zwischen Daten und Logik

Warum wollen wir das?

- kann helfen inkonsistente Daten zu verhindern
- ermöglicht leichtes Umstellen interner Strukturen
- sorgt für klare Schnittstellen
- macht Aufgaben im Team verteilbar

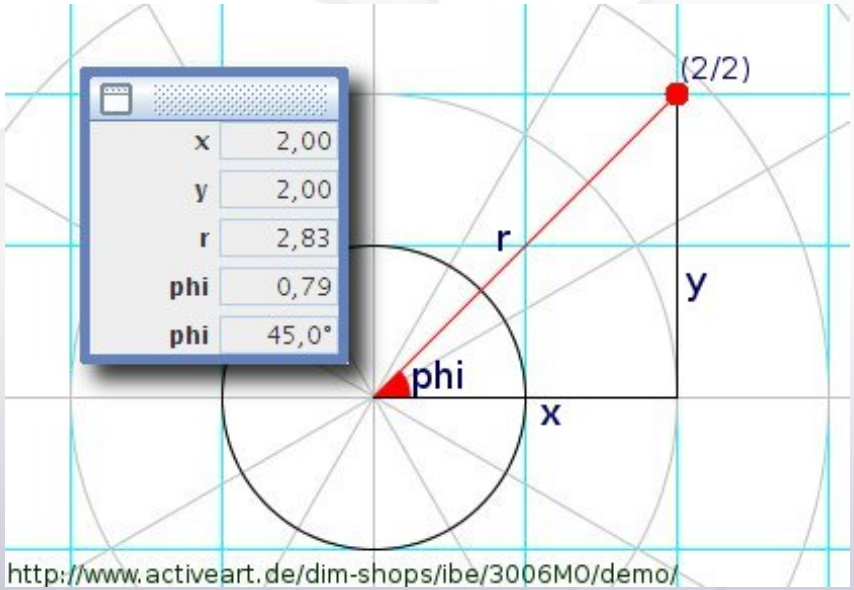
Vergleichen

Sally == Fiona ?

Vergleichen von Objekten

```
1        Point p1 = new Point ( 0.79d, 2.83d);
2        Point p2 = new Point ( 2 , 2);
```

Die beiden Punkte sind auf der selben Stelle im Koordinatensystem, was passiert beim Vergleichen?



Vergleich per ==

```
1        Point p1 = new Point ( 0.79d, 2.83d);
2        Point p2 = new Point ( 2 , 2);
3        if (p1 == p2){
4          System.out.println("p1==p2");
5        }else{
6          System.out.println("p1!=p2");
7        }
```

Was wird ausgegeben?

p1!=p2

Warum? $\Rightarrow$ p1 und p2 sind unterschiedliche Objekte!

Was müssen wir vergleichen? $\Rightarrow$ Die Koordinaten.

Die Lösung: Wir vergleichen mit einer eigenen Methode, dazu erweitern wir die Klasse wieder.

Vergleich per equals

```
1  class Point{
2  // [...]  Code wie bisher
3
4   public boolean equals (Point otherPoint){
5
6      if (otherPoint == null) {  // Sonst Null-Pointer Exception!
7         return false;
8      }
9
10     return (   this.getX() == otherPoint.getX()
11            && this.getY() == otherPoint.getY()  );
12
13  }
14
15 }
```

Vergleich per equals (Beispiel)

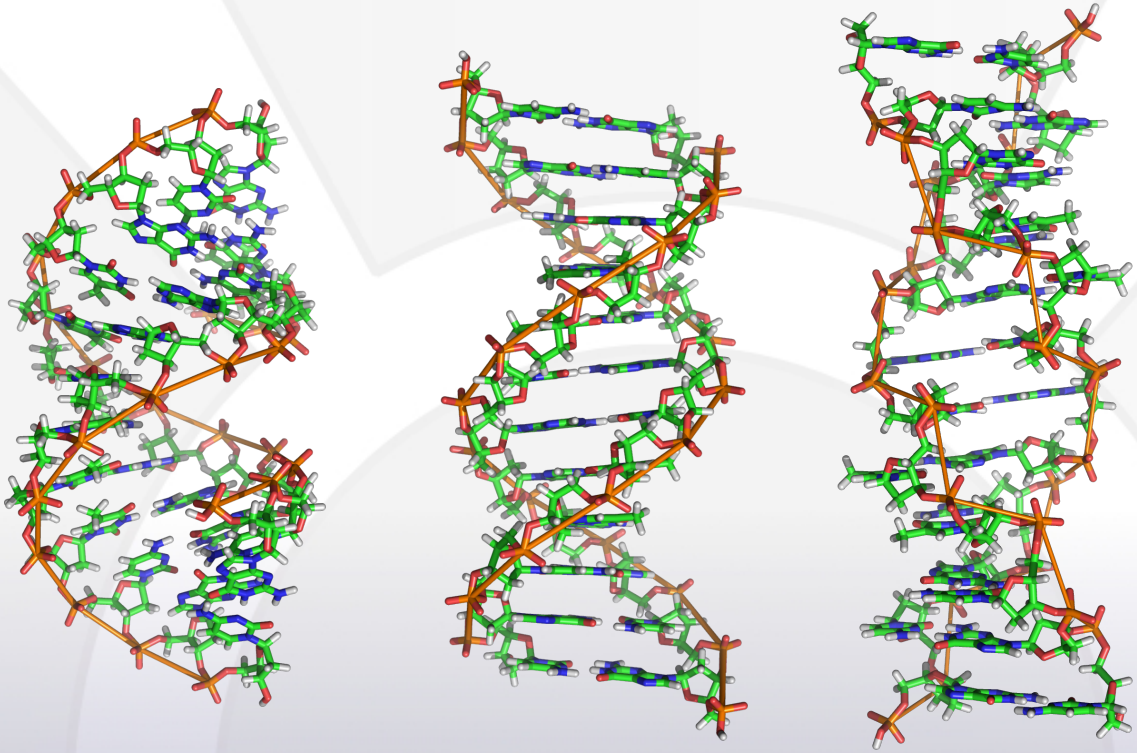
```
1
2         Point p1 = new Point ( 0.79d, 2.83d);
3         Point p2 = new Point ( 2 , 2);
4
5         if ( p1.equals(p2) ){
6           System.out.println("p1 equals p2");
7         }else{
8           System.out.println("p1 differs from p2");
9         }
```

Was wird diesmal ausgegeben?

```
p1 equals p2
```

Warum? $\Rightarrow$ Es wird die Vergleichsmethode von p1 genutzt, diese vergleicht die X und Y-Koordinaten.

Vererbung



Vererbung

Gemeinsamkeiten entdecken

- Erinnert Ihr Euch noch an die Joggerin am Strand mit ihrem Hund?
- Hund und Joggerin haben etwas gemeinsam: Sie können joggen.
- Sie haben noch viel mehr gemeinsam:

  beide sind Säugetiere, Landbewohner, haben einen Namen...
- Auch bei unseren Objekten gibt es so etwas
- Kann man solche Gemeinsamkeiten nicht benutzen, um Dinge zu vereinfachen?
- JA!

Vererbung - Das Grundprinzip

Was ist das grundsätzliche Muster?

- Es gibt eine allgemeine Klasse, z.B. Säugetier
- Diese definiert Methoden, die alle Säugetiere haben müssen, z.B. `breathe()`
- Die von dieser Oberklasse vorgegebenen Methoden kann eine Unterklasse verfeinern/ersetzen
- Eine Unterklasse (Mensch) erweitert die Oberklasse ausserdem um eigene Methoden, z.B. `writeName()`
- Dasselbe gilt für Attribute - eine Unterklasse kann eigene Attribute einführen

Ein Beispiel

```java
class Mammalian{

    private String name;

    public Mammalian(String myName){
        this.name=myName;
    }
    public void breathe(){
     //...
    }
    public String getName(){
        return this.name;
    }
}
```

```java
class Human extends Mammalian{  // extends ist neu!

    private String diary;

    public void writeName(){
      System.out.println("Hi, my Name is "+this.getName());
    }
}
```

Erweiterung des Punkts zum Kreis

Ein etwas besseres Beispiel:

```
1  class Circle extends Point{
2
3    private int radius;
4
5    public Circle(int x, int y, int r){
6     super(x,y);     // auch neu!
7     this.setRadius(r);
8    }
9
10   public void setRadius(int r){
11     this.radius=r;
12   }
13
14   public int getRadius(){
15     return this.radius;
16   }
17
18 }
```

Erweiterung des Punkts zum Kreis

Ein etwas besseres Beispiel:

```
1  class Circle extends Point{
2
3    private int radius;
4
5    public Circle(int x, int y, int r){
6     super(x,y);     // Ruft den Konstruktor der Superklasse
7     this.setRadius(r);
8    }
9
10   public void setRadius(int r){
11     this.radius=r;
12   }
13
14   public int getRadius(){
15     return this.radius;
16   }
17
18 }
```

Zusammenfassung: Vererbung

- Eine (beliebige) Klasse kann mittels Vererbung erweitert werden
- Die so entstehende Klasse erbt alle Attribute und Methoden
- Methoden können neu definiert werden und vorhandene überschreiben
- per `super.`*`funktionsname`*`()` kann überschriebene Methode aus Unterklasse aufgerufen werden
- Zugriffsbeschränkung:
  - `public` - voller Zugriff, keinerlei Schutz
  - `private` - kein Zugriff von Aussen, auch nicht aus Unterklassen
  - `protected` - Zugriff aus den Unterklassen und Package
- private mit Gettern und Settern ist das 'sicherste' aber aufwendigste

Zusammenfassung: Vererbung

- So wie wir immer von Objekten reden macht es auch Java,
- alle Objekte haben eine gemeinsame Oberklasse: `Object`
- `Object` hat eine vordefinierte `equals()`-Methode, die wir überschreiben können...

Vergleich per equals - jetzt als Object

```
1  class Point{
2  // [...]  Code wie bisher
3
4   public boolean equals (Object other){
5
6      if (other == null) {  // Sonst Null-Pointer Exception!
7         return false;
8      }
9
10     if ( ! other instanceof Point){
11        return false; // Das andere ist kein Punkt
12     }
13
14     Point otherPoint = (Point) other; // zum Punkt casten
15
16     return (  this.getX() == otherPoint.getX()
17           && this.getY() == otherPoint.getY() );
18
19  }
20
21 }
```

Standardmethoden

[Picture totally unrelated]

einige Methoden von Object

- Die `equals`-Methode ist eine 'Standard'-Methode von Object
- Es gibt eine ganze Reihe solcher Methoden
- Name und Zweck sind festgelegt, Beispielimplementierungen vorgegeben
- Sie sind in der Api-Dokumentation (Javadocs) beschrieben
- Sie werden zum Teil auch Java-intern benutzt (z.B. bei best. Listen)
- Sind bei eigenen Klassen sinnvoll zu überschreiben

Auswahl an typischen 'default-Methoden'

- `public boolean equals(Object o);` - gerade Beschrieben
- `public int compareTo(Object o);` - zum Sortieren
- `public String toString();` - ???

Das Human-Objekt

```
1   class Human{
2
3     private String name;
4     private int age;
5
6     public Human(String name, int age){
7       this.name = name;
8       this.age  = age;
9     }
10
11    public void describe(){
12      System.out.println("Name:␣␣"+this.name);
13      System.out.println("Alter:␣"+this.age);
14    }
15
16  }
```

Ein Objekt

```
1
2     Human somebody = new Human ("Somebody", 18);
3
4     System.out.println(somebody);
```

Die Ausgabe lautet:
`Human@1d9f953d`
Warum?

- Für Java ist es eine Instanz der Human-Klasse mit der ID 1d9f953d.
- Das nützt uns aber nix!
- Die Ausgabe kommt aus der `toString()`-Methode von `Object`
- Diese Methode können wir überschreiben

toString()

```
1   class Human{
2
3     private String name;
4     private int age;
5
6     public Human(String aName, int anAge){
7       this.name = aName;
8       this.age  = anAge;
9     }
10
11    public String toString(){
12      return ("This is a Human named " + this.name
13              + " with the age of " + this.age + ".");
14    }
15
16  }
```

toString() - Beispiel

```
1  // [...]
2
3  Human grandfather = new Human("Grandfather", 92);
4
5  System.out.println(grandfather);
```

Ausgabe:

```
This is a Human named Grandfather with the age of 92.
```

⇒Immer wenn Java einen String erwartet, ruft es die `toString()`-Methode auf!

Blick über den Tellerrand

Andere Sprachen: PHP

```php
<?php

class Human
{
    private $name;

    public function __construct($n){
      $this->name = $n;
    }

    public function getName()
    {
      return $this->name;
    }
}

$h = new Human("Grandfather");
?>
```

Andere Sprachen: Python

```
1
2  class Human(object):
3      def __init__(self, name):
4          self._name = name
5
6      def get_name(self):
7          return self._name
8
9      name = property(get_name)
10
11 h = Human('Grandfather')
12 print h.name
```

Andere Sprachen: C++

```
1  class Human
2  {
3  public:
4      Human( std::string name);
5      std::string getName();
6  private:
7      std::string Name;
8  };
```

```
1  Human::Human(std::string n)
2  {
3      Name = n;
4  }
5
6  std::string Human::getName()
7  {
8     return Name;
9  }
```

Andere Sprachen: C#

```
1
2   class Human
3   {
4         private string m_name;
5
6         public Human(string name) {
7           m_name = name;
8         }
9
10        public string Name {
11          get {
12                return m_name;
13          }
14        }
15  }
```

Thanksto:

- Bernd & Mario für das Ertragen des Probevortrags
- Methyltheobromin (Guaranin) in wässriger Lösung
- #mr-lug auf irc.fu-berlin.de für C++, python und C#
- Hesso und Matthias für das Korrekturlesen in letzer Minute
- 4!

Bildnachweise:
Dream Eyes von Lan Bui, http://www.flickr.com/photos/lanbui/81416656/
Twins #109 von Oude School, http://www.flickr.com/photos/oudeschool/1775934584/
Cat Mandu von eva101, http://www.flickr.com/photos/evapro/385650640/
In die Ferne schauen von donenik, http://www.piqs.de/fotos/5256.html
A-DNA, B-DNA and Z-DNA.png, Richard Wheeler (Zephyris) at en.wikipedia.org