

**LE - 6**

**Kapselung . toString . equals . Java API**

**Ilya Shabanov  
Kai Dietrich**

**Freitagrunde 4!**

Ilya & Kai

1

12.4.2007

Wiederholung:

**Objektorientierung**

Ilya & Kai

2

12.4.2007

```
class Human {
    int age;
    Head head;

    public Human(int age) {
        this.age = age;
        this.head = new Head();
    }

    public void speak(String what) {
        this.head.shoutOutLoudly(what);
    }
}
```

Ilya & Kai

3

12.4.2007

```
class Vorlesung {
    public static void main(String [] args) {
        Human kai = new Human(23);
        kai.speak("Wiederholung");
    }
}
```

Ilya & Kai

4

12.4.2007

```
Human paul =
  new Human();
```



Human schatzi = paul;

Human papa = paul;

Human officer = paul;



Fragen?

toString(), equals(),  
Kapselung

Ein bisschen Verwirrung :

```
class LangzeitStudent{
  int terms;
  String name;

  public LangzeitStudent( int terms , String name ){
    terms = terms;
    name = name;
  }
}
```

- Die Klasse besitzt zwei Variablen: den Namen und die Anzahl der Semester des Langzeitstudenten, die dem Konstruktor übergeben werden.

Ein bisschen Verwirrung :

```
LangzeitStudent s1 = new LangzeitStudent( 17 , "Nicky" );
LangzeitStudent s2 = new LangzeitStudent( 87 , "Nicky's Mom" );
```

```
System.out.println("s1: " + s1);
System.out.println("s2: " + s2);
```

```
s1: LangzeitStudent@16f0472
s2: LangzeitStudent@18d107f
```

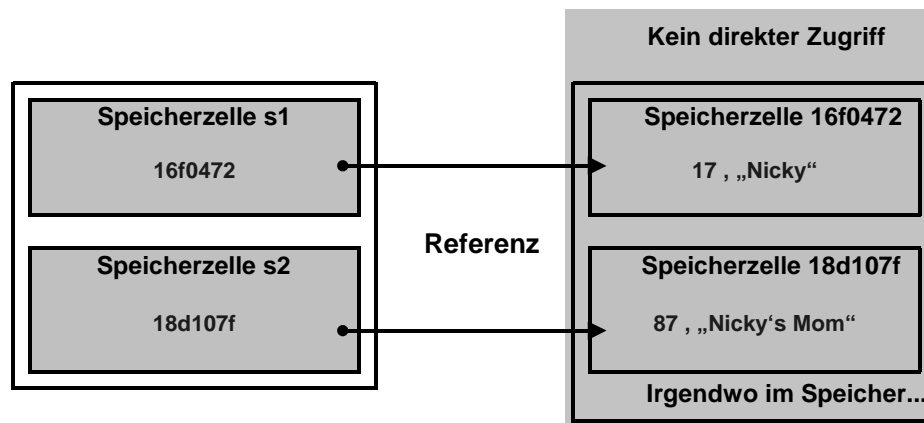


- Den Inhalt von s1 und s2 auszugeben wäre hier sinnvoller.

Was bedeuten die Zahlen? :

```
s1: LangzeitStudent@16f0472
s2: LangzeitStudent@18d107f
```

- Die Zahlen sind Adressen.



Was bedeuten die Zahlen? :

- Tatsächlich macht der Compiler daraus:

```
System.out.println("s1: " + s1.toString() );
System.out.println("s2: " + s2.toString() );
```

- toString() macht aus der Adresse einen String.
- Jede Klasse besitzt eine Standard toString() Methode, die das macht.

- Man kann sich eine eigene toString() Methode definieren! (Überschreiben)

toString() überschreiben :

```
class LangzeitStudent{
    int numOFTerms;
    String studentName;

    public LangzeitStudent( int terms , String name ){
        numOFTerms = terms;
        studentName = name;
    }

    public String toString(){
        String output = "Ich bin " + studentName
            + ". Ich studiere erst seit " + numOFTerms
            + " Semestern!";
        return output;
    }
}
```

```
Ich bin Nicky. Ich studiere erst seit 17 Semestern!
Ich bin Nicky's Mom. Ich studiere erst seit 87 Semestern!
```

Objekte vergleichen :

```
class LangzeitStudent{
    int terms;
    int matNum;
    String name;

    public LangzeitStudent( int terms , String name , int mnr ){
        terms = terms;
        name = name;
        matNum = mnr;
    }
}
```

- Um Studenten zu vergleichen fügen wir eine Variable für die Matrikelnummer ein.

Objekte vergleichen :

```
LangzeitStudent s1 = new LangzeitStudent( 17 , "Nicky" , 221745 );
LangzeitStudent s2 = new LangzeitStudent( 17 , "Nicky" , 221745 );

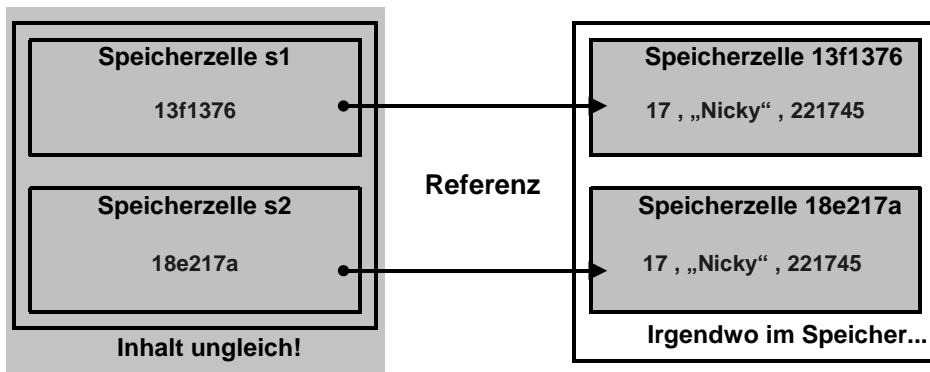
System.out.println( "Vergleich Objekt: " + (s1 == s2) );
```

- s1 und s2 enthalten die gleichen Daten, sind sie aber gleich?

Vergleich Objekt: false

- Trotz gleicher Daten sind s1 und s2 nicht gleich!

Warum sind s1 und s2 nicht gleich? :



- Die Speicherzellen haben zwar gleichen Inhalt, sind aber unterschiedliche Speicherzellen!

- == vergleicht den Inhalt von s1 und s2!

Wie kann man den Inhalt vergleichen? :

- equals ist eine Methode, die genau das macht.

```
public boolean equals( Object o )
```

- Wie toString() gibt es diese Methode bereits für jedes Objekt.

- Um eigene Objekte zu vergleichen muss man jedoch sein eigenes equals schreiben ( also wieder überschreiben ).

equals überschreiben :

```
class LangzeitStudent{
    int terms;
    int matNum;
    String name;

    public LangzeitStudent( int terms , String name , int mnr ){
        terms = terms;
        name = name;
        matNum = mnr;
    }
    public boolean equals( Object o ){
        if(!(o instanceof LangzeitStudent)){return false;}

        LangzeitStudent student = (LangzeitStudent)o;

        if( student.matNr == this.matNr){
            return true;
        }
        return false;
    }
}
```

equals benutzen :

```
LangzeitStudent s1 = new LangzeitStudent( 17 , "Nicky" , 221745 );
LangzeitStudent s2 = new LangzeitStudent( 17 , "Nicky" , 221745 );

System.out.println( "Vergleich Objekt: " + s1.equals( s2 ) );
```

Vergleich Objekt: true

▪ SIEG!

„==“ und „equals“ Zusammenfassung :

equals( Object o )	==
<ul style="list-style-type: none"> <li>▪ Vergleicht Instanzen von Objekten.</li> <li>▪ Muss vom Programmierer implementiert werden.</li> </ul>	<ul style="list-style-type: none"> <li>▪ Vergleicht nur primitive Datentypen wie int, double und boolean.</li> <li>▪ Ist ein Standardkonstrukt von Java.</li> </ul>

Kapselung ( Encapsulation ) :

▪ Eines der OOP Konzepte ist die Bündelung der Daten an ihre Funktionalität.

**Warum eigentlich?**

```
class Apple{
    .....
}
```

▪ Wir versuchen einen Apfel um 5 bit nach links zu shiften.

„Ich esse zwar nur Pizza, habe aber den begründeten Verdacht, dass sich Äpfel nicht um 5 bit shiften lassen...“

– der Langzeitstudent

▪ Moral von der Geschichte' - Äpfel shiftet man nicht.

Kapselung ( Encapsulation ) :

- Jedes Objekt hat daher Methoden, die das beschreiben, was man damit tun kann.

```
class Apple{
    ...
    public boolean isAppleRed(){ ... }
    public void eatApple(){ ... }
    public void dropApple(){ ... }
    public int getWeight(){ ... }
    ...
}
```

- Wir können mit einem Objekt nur das machen, was es uns erlaubt.

Das heißt Kapselung .

Information Hiding :

- In Java hat man die Möglichkeit Variablen zu „schützen“, damit sie nur innerhalb der Klasse verändert werden können.

- Das ist immer dann nützlich, wenn man Variablen hat, die z.B. nur positiv sein können ( Alter ).

- Es soll sichergestellt sein, dass niemand von außerhalb die Variable auf einen ungültigen Wert setzt.

```
class LangzeitStudent{
    ...
    private int terms;
    private String name;
    ...
}
```

Auswirkung von private :

```
class LangzeitStudent{
    ...
    private int terms;
    private String name;
    ...
}
```

- Die Variablen sind fortan nur noch innerhalb der Klasse LangzeitStudent sichtbar!

```
LangzeitStudent s1 = new LangzeitStudent( 17 , "Nicky" , 221745 );
System.out.println( "s1's name is: " + s1.name );
```

- Compilerfehler! Man darf nicht auf name zugreifen, da private!

Ein praktisches Beispiel:

```
class CommanderKeen{
    public int positionX;
    public int positionY;
    ...
}
```



Jump

- Die x und y Position darf nur auf bestimmte Art und Weise verändert werden.

- Es muss sichergestellt werden, dass die Bewegung zulässig ist, also dass keine Mauer im Weg ist o. Ä.



Pogo jump



Laufen

Ein praktisches Beispiel:

- Kapselung der x- und y-Variablen mit den Methoden für Bewegung.
- Zugriff nur über diese Methoden.

```
class CommanderKeen{
  private int positionX;
  private int positionY;
  ...
  public void moveRight(){
    if( isMoveRightPossible() )
      positionX += 5;
  }
  public void moveLeft(){ ... }
  public void jump(){
    if( isJumpPossible() )
      //jumping code ...
  }
}
```



Getter und Setter :

- Manchmal ist es dennoch notwendig auf die Variablenwerte direkt zuzugreifen, z.B. wenn das Spiel neugestartet wird.
- Man erstellt dann meist s.g. getter und setter Methoden um die Variablen trotzdem als private beibehalten zu können.

```
class CommanderKeen{
  private int positionX;
  private int positionY;
  ...
  public int getX(){
    return positionX;
  }
  public void setX( int newX ){
    if( ... ) // test value in some way, if necessary
      positionX = newX;
  }
}
```

# Die Java API

# Die Java ... was?

API - Application Programming Interface,  
eng. „Programmierschnittstelle“

= Definition der Syntax und Semantik aller  
Klassen und Methoden der von Java

- Versionen
- Wo finden?
- Wie benutzen?
- Ein kleiner Überblick

## Java gibt es in vielen Versionen:

- Java 1.3.x
- Java 1.4.x
- **Java 1.5.x = Java 5**
- Java 6

Jede Version hat ihre eigene API-Dokumentation

- ▶ bei den Downloads aufpassen
- ▶ Kernfunktionalitäten bleiben gleich, nur Randfunktionen ändern sich oder neue Funktionen werden hinzugefügt

## Wo finden?

a) zum Download: <http://java.sun.com/javase/downloads/>

b) Online:

Java 1.4.2: <http://java.sun.com/j2se/1.4.2/docs/api/index.html>

Java 5: <http://java.sun.com/j2se/1.5.0/docs/api/>

Java 6: <http://java.sun.com/javase/6/docs/api/>

Online ist die Dokumentation mit einer Suchfunktion ausgestattet

# und jetzt in LIVE



Weiter im Text...

```
import java.util.ArrayList;

class PersonalDatabase {
    ArrayList myRoomContent;

    ...
}
```

```
import java.util.*;

class PersonalDatabase {
    ArrayList myRoomContent;
    PriorityQueue tasks;

    ...
}
```

Aber NICHT:

```
import java.*;

class PersonalDatabase {
    ArrayList myRoomContent;
    PriorityQueue tasks;

    ...
}
```

```
cleus@fiesta:~$
Test.java:4: cannot find symbol
symbol   : class ArrayList
location: class PersonalDatabase
    ArrayList myRoom;
    ^
```

Keine Hierarchie – java.util ist eigentlich nur ein String

Wer mehr wissen will:

Thinking in Java  
[http://www.codeguru.com/java/tij/tij\\_c.shtml](http://www.codeguru.com/java/tij/tij_c.shtml)

Java ist auch eine Insel  
<http://www.galileocomputing.de/openbook/javainsel5/>

Java Tutorials  
<http://java.sun.com/javase/reference/tutorials.jsp>

empfehlenswert:

**[www.javadocs.org](http://www.javadocs.org)**

- Schnelle Umleitung auf Klassen in Java 5 Doku
- auch als Searchlet für Firefox verfügbar

**static**

Seid ihr noch da?



`static`



`java.lang.Math`

`java.lang.Math`

**Mathematik Objekt erzeugen um zu rechnen?**

**Natürlich nicht, denn:**

**Alle Methoden in Math sind `static`**

```
public static double abs(double a)
public static double cos(double a)
public static double log(double a)
public static double random()
public static double sin(double a)
public static double sqrt(double a)
...
```

**static =**

die Methode oder das Attribut  
existiert auch ohne ein Instanz

Wo existiert ein **static** ?

in seiner Klasse

Ein **static** existiert genau einmal pro Java Virtual Machine

Was kann man mit **static** machen?

```
public static void main(String[] args)
```

Funktionen ohne Objekte

Konstanten (`static double java.lang.Math.PI`)

■ ■ ■

Fragen?

And now for something  
completely different ...

Java

```
class CommanderKeen {
    private int Xpos;
    private int Ypos;

    public void moveRight(){
        if( isMoveRightPossible() ){
            XPos += 5;
        }
    }
    public void moveLeft(){ ... }
    public void jump(){
        if( isOnPlatform() && isJumpPossible() ) {
            //jumping code ...
        }
    }
}

class Game {
    public static void main(String[] args) {
        CommanderKeen Keen;
        Keen = new CommanderKeen();
        Keen.jump();
        return;
    }
}
```

```
class CommanderKeen {
    private int Xpos;
    private int Ypos;

    public void moveRight(){
        if( isMoveRightPossible() ){
            XPos += 5;
        }
    }
    public void moveLeft(){ ... }
    public void jump(){
        if( isOnPlatform() && isJumpPossible() ) {
            //jumping code ...
        }
    }
}

class Game {
    public static void Main(string[] args) {
        CommanderKeen Keen;
        Keen = new CommanderKeen();
        Keen.jump();
        return;
    }
}
```

C#

```
class CommanderKeen {
private:    int Xpos;
           int Ypos;

public:
    void moveRight(){
        if( isMoveRightPossible() ){
            XPos += 5;
        }
    }
    void moveLeft(){ ... }
    void jump(){
        if( isOnPlatform() && isJumpPossible() ) {
            //jumping code ...
        }
    }
};

void main(int argc, char* argv[]) {
    CommanderKeen* Keen;
    Keen = new CommanderKeen();
    Keen->jump();
    delete Keen;
    return;
}
```

C++

```
Object subclass: #CommanderKeen
  int Xpos: 0
  int Ypos: 0
```

```
CommanderKeen>>moveRight
  (self isMoveRightPossible) ifTrue:
    [XPos := XPos + 5.]
```

```
CommanderKeen>>moveLeft
  "..."
```

```
CommanderKeen>>jump
  ( self isOnPlatform and: [self isJumpPossible] ) ifTrue:
    ["jumping code ..."]
```

```
Keen := CommanderKeen new.
Keen jump.
```

## Smalltalk

```
class CommanderKeen:
  def __init__(self):
    self.Xpos = 0
    self.Ypos = 0

  def moveRight(self):
    if (self.isMoveRightPossible()):
      self.XPos += 5

  def isMoveRightPossible(self):
    return True

  def jump(self):
    if (self.isMoveRightPossible() and self.isMoveRightPossible()):
      self.XPos += 5

if __name__ == "__main__":
  keen = CommanderKeen()
  print keen.XPos
  keen.jump()
  print keen.XPos
```

## Python

```
(defclass commander-keen ()
  ((x-pos :initform 0 :initarg :x-pos :accessor x-pos)
   (y-pos :initform 0 :initarg :y-pos :accessor y-pos)))

(defgeneric move-right (commander-keen)
  (:documentation "moves commander keen 5 units right"))

(defmethod move-right ((commander-keen commander-keen))
  (with-accessors ((x-pos x-pos)) commander-keen
    (if (is-move-right-possible)
        (setf x-pos (+ x-pos 5)))))

(defgeneric move-left (commander-keen)
  (:documentation "moves commander keen 5 units left"))

(defgeneric jump (commander-keen)
  (:documentation "lets commander keen jump"))

(defmethod jump ((commander-keen commander-keen))
  (with-accessors ((y-pos y-pos)) commander-keen
    (if (and (is-on-platform) (is-jump-possible))
        ;; jumping code
        )))

(defun play ()
  (let ((keen
        (make-instance 'commander-keen)))
    (jump keen)))
```

## Lisp

Viel Spaß in der Übung!