# Java Crashkurs

Kim-Manuel Klein (kmk@informatik.uni-kiel.de)

May 7, 2015

# Quellen und Editoren

- Internet Tutorial: z.B. http://www.java-tutorial.org

Editoren

- Normaler Texteditor (Gedit, Scite oder ähnliche)
- Eclipse (etwas komplexer Aufbau)

# Vor- und Nachteile von Java

Vorteile:

- Unabhängig von Plattform: Durch Übersetzung in virtuelle Maschine (JVM)
- Netzwerkfähig, nebenläufig
- Sicherheitskonzept

Nachteile:

- Laufzeithandicap durch Interpretation der JVM

# Aufbau von Java-Programmen

Ein imperatives Java-Programm besteht aus

- Klassendeklaration mit einer einzigen Methode namens "main":

```
public class <KlassenName>
{
    public static void main(String[] args)
    {
    <Anweisungen>
    }
}
```

# Ausgabe

- Ausgabe in Java erfolgt mit "System.out.print("Text");"
- Zeilenumbruch erfolgt mit "\n"
- Der Befehl "System.out.println("Text");" macht Zeilenumbruch hinter "Text"
- "System.out.print()" funktioniert mit beliebigen Datentypen

# Erstes Beispielprogramm

```
public class HelloWorld
{
    public static void main(String[] args)
    {
    System.out.println("Hello␣World");
    }
}
```

Gibt "Hello World" aus.

# Kompilierung und Ausführung:

- Speichern der Datei als "HelloWorld.java"
- Kompilieren mit "javac HelloWorld.java" (im entsprechenden Verzeichnis)
- Ausführen mit "java HelloWorld"

# Das Testprogramm

- Ausführung mit "python test.py <Problem> <Pfad>"
- Als Pfad akzeptiert das Programm Dateien im Format zip, tar, tar.gz
  Beispiel: "python test.py Triangle /Kim/workspace/triangle.zip"
- Die gepackten Dateien enthalten die .java-Datei sowie ein .cfg-File
- Das .cfg-File enthält die Namen der Autoren: <Vorname1> <Nachname1>, <Vorname2> <Nachname2>
- Wichtig: Die .java-Datei muss denselben Namen wie das Problem haben. Hier: "Triangle.java"

# Verfügbarkeit des Testprogramms:

- Installierbar unter Linux (benötigt wird Python 2.7 oder höher)
- Vorinstalliert im Pool des Rechenzentrums im Verzeichnis "/home/fku/Public/ads"
- Per SSH oder ThinClient erreichbar

# Variablen Deklaration

Eine Deklaration lokaler Variablen hat die Form

```
<Type>  <VarName>;
```

Normale Zuweisung

```
<VarName> = <Expression>;
```

Die Variable kann direkt initialisiert werden mit

```
<Type>  <VarName> = <Expression>;
```

# Ganze Zahlen

- byte: Zahlenraum von -128 bis 127
- short: Zahlenraum von -32768 bis 32767
- int: Zahlenraum von -2,147,483,648 bis 2,147,483,647
- long: Zahlenraum von -9,223,372,036,854,775,808 bis 9,223,372,036,854,775,807

Gleitkommazahlen: float, double (nach IEEE-754–Standard)

# Datentyp Boolean

Der Typ boolean hat genau zwei Werte, true und false. Boole‘sche

Operatoren:

- ! strikte Negation
- & & Konjunktion („und“)
- || Disjunktion („oder“)

# Beispiel

```
public class BooleanTest
{
    public static void main(String[] args)
    {
    boolean b1 = false;
    boolean b2 = true;
    boolean b3 = b1 && b2;
// liefert false
    boolean b4 = b1 || b2;
// liefert true
    }
}
```

# Arithmetische Operationen und Vergleichsoperationen

Arithmetische Operationen

- * Multiplikation, / Division, % Modulo (Rest)
- + Addition, - Subtraktion

Vergleichsoperationen

- $>$ größer, $>=$ größer oder gleich
- $<$ kleiner, $<=$ kleiner oder gleich
- $==$ gleich, $!=$ nicht gleich

Liefert Typ Boolean

# Zeichen und Zeichenketten

- Typ char (für character) bezeichnet Menge der Zeichen aus dem Unicode-Zeichensatz
- char umfasst ASCII-Zeichensatz mit kleinen und großen Buchstaben, Zahlen und verschiedenen Darstellung von Zeichen durch Umrahmung mit Apostroph Sonderzeichen
- Darstellung von Zeichen durch Umrahmung mit Apostroph Beispiel: ‘a‘ , ‘A‘ , ‘1‘ , ‘9‘
- Zeichenketten: werden mit Doppelapostroph umrahmt und sind vom Typ String (eine Klasse)

# Fallunterscheidung

Die Fallunterscheidung in Java hat die Form

```
if ( <Bool Expression> )
     <Statement>
```

bzw.

```
if ( <Bool Expression> )
     <Statement>
else
     <Statement>
```

# Beispiel

Die Fallunterscheidung in Java hat die Form

```
int i =5; int j = 6;
if ( i > j )
     i=j+1;
else
{
     System.out.println("i␣kleiner␣oder␣gleich␣j");
     i = j;
}
```

Wichtig im Fall von Mehrfachanweisungen sind Klammern (siehe else-Zweig).

# Schleifen

Allgemein hat eine for-Schleife die Form

for (Initialisierung; Bedingung; Zählerkorrektur) <Statement>

Beispiel:

```
int end = 10;
for (int i=1; i <= end; i++)
{
    System.out.println("tick" + i);
}
```

# Die while-Schleife

Die while-Schleife hat die Form

while (<Boolescher Ausdruck>) <Statement>

Beispiel:

```
int n = 1, end = 10;
while (n <= end)
{
    System.out.println("tick" + n);
    n++;
}
```

# Beispiel

```
int number = 234, length = 0;
while (number != 0)
{
    number = number /2;
    length++;
}
System.out.println(length);
```

# Arrays

Ein Array ist eine Reihung von Elementen eines festen Grundtyps.
Deklaration einer Array- Variablen

```
<Type> [] <VarName>;
```

Initialisierung:

```
<VarName> [] = new <Type> [<Length>];
```

Beispiel:

```
int [] einArray;
einArray = new int[5];
for (int i =0; i< 5;i++)
{
    einArray[i] = i;
}
```

# Beispiel

Einfachere Initialisierung:

```
int [] einArray;
einArray = new int[5] {0,1,2,3,4};
```

# Mehrdimensionale Arrays

Deklaration:

```
<Type> [][] <VarName>;
```

Initialisierung:

```
<VarName> [][] = new <Type> [<Length1>][<Length2>]
```

# Funktionen und Prozeduren

Form:

```
public static <Type> <Name> (parameter)
{body}
```

Möglich sind Typen wie z.B.: void, int, char, long etc.
Mit "return <Type>;" wird der Rückgabewert bestimmt.

# Funktionen und Prozeduren

Beispiel:

```
public static int add(int z1, int z2)
{
    return z1+z2;
}
```

Aufruf erfolgt mit

```
int erg = add(z1,z2);
```

# Parameterübergabe

```
public static void main(String[] args) {
        int i = 10;
        System.out.println(param(i));
// liefert 12
        System.out.println(i);
// liefert 10
    }

    private static double param(int d){
        d = d+2;
        return d;
    }
```

# Arrayübergabe

```
public static void main(String[] args) {
        i[0] = 10;
        System.out.println(param(i));
// liefert 12
        System.out.println(i[0]);
// liefert 12
    }

    private static double param(int[] d){
        d[0] = d[0]+2;
        return d;
    }
```

# Rekursion

Berechnung der n-ten Fibonacci Zahl.

```
public static int fib(int n)
{
if (n==0)
        return 0;
if (n==1)
        return 1;

return fib(n-1) + fib(n-2);
}
```

# Rekursion

Berechnung der Fakultät.

```
static int FakultaetRekursiv(int n) {
if (n == 1)
        return 1;
else
        return FakultaetRekursiv(n - 1) * n;
}
```

# Klassen

Beispiel:

```
public class Student
{
    public String name = "";
    public int matrikelnummer = 0;
    public String fach = "";
}
```

Verwendung:

```
Student student1 = new Student();
student1.name = "Kim";
student1.matrikelnummer = 1;
student1.fach = "Informatik";
```

# Verkettung:

Verweis auf Objekt desselben Typs

```
public class Element
{
        public int inhalt = 0;
        public Element next = null;
}
```

Erzeugung einer Verkettung:

```
Element S = new Element();
S.next = null;
S.inhalt = 5;
Element T = new Element();
T.next = S;
T.inhalt = 3;
System.out.println(T.next.inhalt); //Liefert 5
```

# Listen:

Erzeugung einer Liste der Länge $n$:

```
Element T = new Element(); // Anker der Liste
T.next = null;
T.inhalt = 0;
Element Lauf = T; //Laufzeiger
for(int i=1;i<n;i++)
{
     Element h = new Element();
     h.inhalt = i;
     Lauf.next = h;
     Lauf = Lauf.next;
}
```

# Listen:

Auslesen einer Liste mit Anker $T$:

```
Element Lauf = T;
while(Lauf != null)
{
        System.out.println(Lauf.inhalt);
        Lauf = Lauf.next;
}
```

# Verkettung:

Doppelt verkettete Listen:

```
public class Element
{
        public int inhalt = 0;
        public Element next = null;
        public Element before = null;
}
```

# Verkettung:

Baumknoten:

```
public class Element
{
        public int inhalt = 0;
        public Element left = null;
        public Element right = null;
}
```