Java I – Vorlesung 3
Objektorientiertes Programmieren

10.5.2004

Die restlichen Kontrollstrukturen
Grundlagen des OOP
Klassen und Objekte

`switch`

- Syntax:

```
switch ( expr )
{
   case const1 :   [statements]
                   [break;]
   case const2 :   [statements]
                   [break;]
   ...
  [default:        statements]
}
```

- Der Ausdruck expr muss Typ `char`, `byte`, `short` oder `int` haben.
- Die konstanten Ausdrücke $const_i$ (keine Variablen!) müssen verschiedene Werte haben.

switch

- Semantik:
  1. Werte `expr` aus.
  2. Wenn es eine `case`-Marke für diesen Wert gibt, springe zur Marke und führe ab dort alle Anweisungen aus.
  3. Wenn keine `case`-Marke für den Wert gibt, springe zur `default`-Marke, wenn es eine gibt; sonst tue nichts.

Ein Beispiel

```
int zahl123 = 2;

switch(zahl123) {
  case 1: System.out.println("eins");
  case 2: System.out.println("zwei");
  case 3: System.out.println("drei");
  default: System.out.println("fertig!");
}
```

break

- Standardmäßig führt switch alle Fälle ab dem passenden aus ("fall-through").
- Ausgabe des Beispielprogramms:

```
zwei
drei
fertig
```

- Meistens will man aber nur genau einen Fall ausführen.
- Dafür verwendet man die `break`-Anweisung, um den ganzen `switch`-Block zu verlassen.

switch mit break

```
int monat = Integer.parseInt(args[1]),
    anfangsWochentag;

switch(monat) {
  case 1: anfangsWochentag = 3;
          break;
  case 2: anfangsWochentag = 6;
          break;
  // usw.
  default: anfangsWochentag = -1; // Fehlerwert
}
```

while-Schleife

- Syntax: `while ( expr )`
  `statement`
- Semantik:
  1. Der Ausdruck *expr* (mit Typ `boolean`) wird ausgewertet.
  2. Trifft er zu (Wert `true`), wird *statement* ausgeführt. Sonst wird der Programmfluss hinter der Schleife fortgesetzt.
  3. Gehe zu 1.

while

```
int i = 0;
double summe = 0, durchschnitt;

while( i < 5 ) {
  summe += Integer.parseInt(args[i]);
  i++;
}

durchschnitt = summe/5;
```

`break` und `continue`

- Die `break`-Anweisung verlässt die aktuelle Schleife, ohne die Bedingung auszuwerten.
- Die `continue`-Anweisung überspringt den Rest des aktuellen Durchlaufs, wertet die Bedingung neu aus und setzt ggf. die Schleife fort.
- Durch Verwendung von Labels kann man Schleifen aus tieferen Schachtelungsebenen abbrechen.

`continue`

```
int i = 0, aktueller_wert;
double summe = 0, durchschnitt;

while( i < 5 ) {
  aktueller_wert = Integer.parseInt(args[i]);
  i++;

  if( aktueller_wert < 0 )
    continue;

  summe += aktueller_wert;
}

durchschnitt = summe/5;
```

`do-while`-Schleife

- Syntax:
```
do
     statement
while ( expr );
```
- Semantik:
  1. Zunächst wird *`statement`* ausgeführt.
  2. Dann wird *`expr`* (mit Typ `boolean`) ausgewertet.
  3. Trifft er zu (Wert `true`), gehe zu 1.
  4. Sonst beende die Schleife.
- Also genau wie `while`, führt aber statement vor der ersten Auswertung von *`expr`* einmal aus.

`for`-Schleife

- Syntax:
```
for(init-anw; test-expr; update-anw)
     statement
```
- Semantik:
  1. Zunächst wird *`init-anw`* ausgeführt.
  2. Falls die *`test-expr`* (Typ `boolean`) zu wahr auswertet, führe *`statement`* aus. Sonst beende die Schleife.
  3. Führe *`update-anw`* aus.
  4. Gehe zu 2.

for

```
int i;
double summe = 0, durchschnitt;

for( i = 0; i < 5; i++ )
  summe += Integer.parseInt(args[i]);

durchschnitt = summe/5;
```


for als while

```
for( init-anw; test-expr; update-anw )
  statement;
```

```
init-anw;

while( test-expr ) {
  statement;
  update-anw;
}
```

Schleifen: Übersicht

- Alle Schleifen können als `while`-Schleifen geschrieben werden.
- `do-while` führt zuerst einmal die Anweisung aus und wertet erst dann die Bedingung aus.
- `for` ist abkürzende Schreibweise; sehr häufig!
- Alle Schleifen können mit `break` abgebrochen und mit `continue` im nächsten Durchlauf fortgesetzt werden.

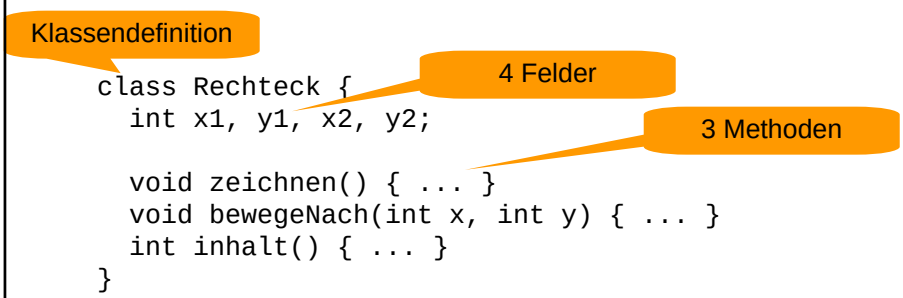
Objektorientierte Programmierung

- Ein Objekt ist eine Kombination von Daten mit Operationen auf diesen Daten.
- Daten werden in Feldern (≈ Variablen) gespeichert; Methoden (≈ Funktionen) sind Sequenzen von Anweisungen, die Felder nutzen.
- Jedes Objekt ist Instanz einer Klasse: Klassen definieren gleichartige Objekte mit ihren Methoden.
- Klasse ist (Referenz-)Datentyp des Objekts.

Eine Klasse für Rechtecke

- Felder:
  - X- und Y-Position der beiden Ecken
- Methoden (was kann man mit Rechtecken machen?):
  - Auf den Bildschirm zeichnen
  - Vom Bildschirm löschen
  - An andere Position bewegen
  - Stauchen und dehnen
  - Flächeninhalt berechnen

Rechtecke in Java-Syntax




```
class Rechteck {
  int x1, y1, x2, y2;

  void zeichnen() { ... }
  void bewegeNach(int x, int y) { ... }
  int inhalt() { ... }
}
```

Warum OOP?

- Objektorientierte Programmierung (OOP) ermutigt den Programmierer dazu, Programme in Klassen aufzuteilen.
- Für viele Projekte sind Klassen gute Gliederungsebene, die zu Dingen in der wirklichen Welt passen.
- Klassen haben typischerweise überschaubare Komplexität.

Warum OOP: Kapselung

- Kapselung (encapsulation): Man kann Implementierungsdetails von Klassen nach außen verstecken:
  - Rechteck mit Breite/Höhe implementieren.
- Damit verstecken wir Komplexität der Implementierung: Andere Leute müssen nur über sichtbare Aspekte nachdenken.
- Implementierung kann jederzeit verändert werden, ohne das Gesamtprogramm zu stören.

Warum OOP: Vererbung

- Klassen können von anderen Klassen abgeleitet werden:
  - "Zeichnen" und "Flächeninhalt" auch für andere geometrische Figuren definiert.
- Abgeleitete Klasse erbt alle Felder und Methoden der Mutter, kann neue dazutun und die geerbten Methoden überschreiben.
- Objekte der abgeleiteten Klasse können überall eingesetzt werden, wo Objekte der Mutterklasse akzeptiert werden.

Warum OOP: Wiederverwendbarkeit

- Abhängigkeiten zwischen verschiedenen Klassen werden in OO-Sprachen explizit gemacht:
  - hängt Rechteck-Klasse von Klassen zum Zeichnen von Linien ab?
- Das ermutigt Programmierer, diese Abhängigkeiten klein zu halten.
- Klassen mit wenigen Abhängigkeiten können in neuen Kontexten eingesetzt werden.
- Solche Klassen werden also einmal entwickelt und können dann gut wiederverwendet werden.

Klassen in Java

- Eine Klasse in Java enthält Members, und zwar
  - Felder (= Variablen; enthalten Daten)
  - Methoden (= Funktionen/Prozeduren; führen Anweisungen aus)
- Klassen werden mit dem Schlüsselwort `class` definiert.
- Klassen sind Datentypen.
- Typischerweise eine Klasse pro Datei.

Objekte

- Klassen sind Datentypen. Die Werte dieses Datentyps heißen Objekte.
- Jedes Objekt hat eine eindeutige Klasse.
- Jedes Objekt hat seinen eigenen Satz von Datenfeldern, deren Werte unabhängig von den Werten in den anderen Objekten sind.
- Neue Objekte werden mit dem Schlüsselwort `new` erzeugt.

Klassen und Objekte: Erstes Beispiel

```
class Demo {
  int x;

  public static void main(String[] args) {
    Demo obj1 = new Demo(), obj2 = new Demo();

    obj1.x = 27;
    obj2.x = -3;

    System.out.println(obj1.x);
    System.out.println(obj2.x);
  }
}
```

Datenfeld x vom Typ int

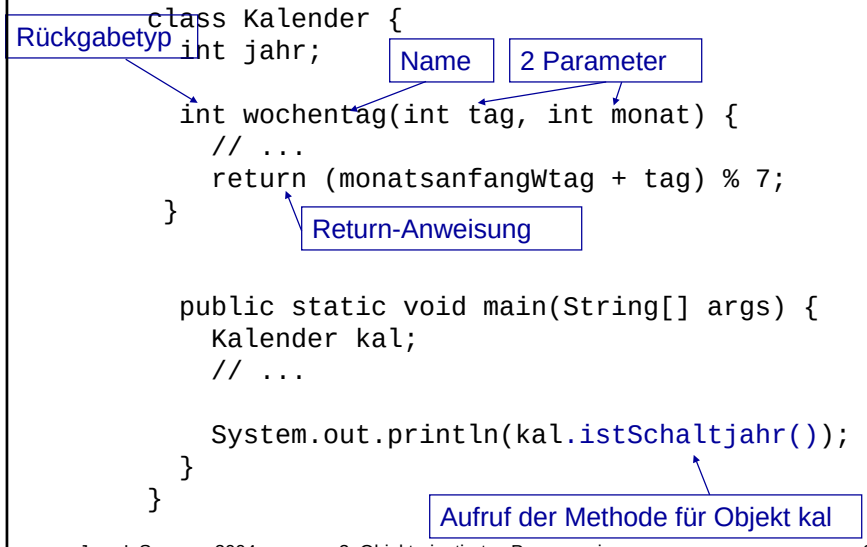
2 Objekte von Klasse Demo erzeugen

Zugriff auf Members: . (Punkt)

Methoden

- Methoden haben Argumente und berechnen daraus einen Wert.
- Methoden werden für eine ganze Klasse definiert, aber (normalerweise) für ein bestimmtes Objekt aufgerufen. Sie verwenden die Felder des Objekts.
- Methodenaufrufe sind Ausdrücke, die in den Methoden der Klasse selbst oder in anderen Klassen verwendet werden können.

Methoden




```
class Kalender {
  int jahr;

  int wochentag(int tag, int monat) {
    // ...
    return (monatsanfangWtag + tag) % 7;
  }

  public static void main(String[] args) {
    Kalender kal;
    // ...

    System.out.println(kal.istSchaltjahr());
  }
}
```


Methoden: Definition

```
returntyp meth(typ1 var1, ..., typn varn) {
  ... Anweisungen ...
}
```

- Methoden haben beliebig viele (oder gar keine) Parameter.
- Für jeden Parameter muss ein Datentyp angegeben werden.
- Jede Methode hat einen eindeutigen Rückgabetyp.

Methodenkörper

- Der Körper der Methode ist ein einziger Block von Anweisungen.
- Im Körper können die Parameter sowie die Felder der Klasse wie normale Variablen verwendet werden.
- Eine return-Anweisung verlässt die Methode und gibt den Wert des Aufrufs an:

```
return expr;
```


Methoden: Auswertung

`obj.meth(expr`$_1$`, ..., expr`$_n$`)`

- Versuche in der Klasse von `obj` eine Methode mit Namen `meth` und Parametertypen zu finden, die zu den Typen der Argumente `expr`$_i$ passen.
- Typen der Argumente können erweiternd in Typen der Parameter konvertiert werden.

Methoden: Auswertung

`obj.meth(expr`$_1$`, ..., expr`$_n$`)`

- Dann führe den Körper der Methode aus. Werte der Argumente werden den Parametern zugewiesen.
- Return-Anweisung beendet Ausführung der Methode und gibt den Wert des Aufruf-Ausdrucks an.
- Methodenaufruf ist ein Ausdruck, dessen Typ der Rückgabetyp der Methode ist.

Methoden ohne Rückgabewerte

- Manche Methoden geben keine Werte zurück. Dafür gibt es den Rückgabetyp void:

```
void printHelloWorld() { ... }
```

- Aufrufe solcher Methoden werden nur direkt in Anweisungen verwendet (nicht in Ausdrücken).
- Methoden mit Rückgabetyp `void` müssen keine `return`-Anweisungen enthalten, aber sie dürfen -- dann mit `return` ohne Parameter:

```
return;
```

Polymorphe Methoden

- In einer Klasse dürfen mehrere Methoden mit dem gleichen Namen definiert sein, wenn sie verschiedene Parameteranzahlen oder -typen haben.
- Der Compiler sucht die heraus, die zu den Typen der Argumente passt.
- Das wird in der Java-Standardbibliothek öfters eingesetzt: z.B. Methode `println` in `PrintStream` (Package java.io), Methode `append` in `StringBuffer`.

Rekursion

- Methoden können andere Methoden aufrufen.
- Insbesondere kann eine Methode sich selbst aufrufen (Rekursion).
- Rekursion ist ein sehr mächtiges Werkzeug beim Programmieren, mit dem man manchmal sehr elegante Programme bekommt.

Rekursion

```
class Fakultaet {
  int fak(int n) {
    if( n <= 1 )
      return 1;
    else
      return n * fak(n-1);
  }

  public static void main(String[] args) {
    Fakultaet f = new Fakultaet();
    int fak6 = f.fak(6);
  }
}
```


Konstruktoren

- Oft will man schon beim Erzeugen eines Objektes ein Argument übergeben (Beispiel: Kalender für ein bestimmtes Jahr).
- Dazu kann man für jede Klasse Konstruktoren definieren.
- Aufgabe eines Konstruktors: Den Datenfeldern sinnvolle Werte zuweisen, evtl. abhängig von den Argumenten des Konstruktors.

Konstruktoren

- Konstruktor ist eine Methode mit dem gleichen Namen wie die Klasse und keinem Rückgabetyp (nicht mal void).
- Wird bei der Auswertung des `new`-Ausdrucks aufgerufen.
- Polymorphie ist erlaubt: Es kann mehrere Konstruktoren mit verschiedenen Parametern geben.

Konstruktor für Kalender

```
class Kalender {
  int jahr, ersterWochentag;

  Kalender(int welchesJahr) {
    jahr = welchesJahr;
    ersterWochentag = ersterWochentagFuer(jahr);
  }

  public static void main(String[] args) {
    Kalender kal = new Kalender(2004);
    boolean schaltjahr = kal.istSchaltjahr();
    // ...
  }
}
```

Vordefinierte Klassen

- Die Java-Standardbibliothek stellt mehrere tausend vordefinierte Klassen zur Verfügung.
- Verwendet sie!
  Siehe v.a. java.lang, java.util, java.io.
- Dokumentation der Klassen in
  `http://java.sun.com/j2se/1.5.0/docs/api/`
- Klassen aus java.lang sind direkt verfügbar.
  Für Klassen aus anderen Packages (z.B. java.util) zunächst
  ```
  import java.util.*;
  ```


Bisher verwendete Klassen und Objekte

- Die Klasse `String` stellt Zeichenketten dar: Jedes Objekt ist eine unveränderliche Zeichenkette.
- Besonderheiten der Klasse `String`:
  - Es gibt String-Literale: Zeichenketten zwischen doppelten Anführungszeichen, z.B.
    `"Hallo Welt"`, `"abcd\n\u102"`
  - Der Operator `+` kann zwei String-Ausdrücke verknüpfen, indem er ihre Werte aneinanderhängt:
    ```
    "Hallo " + "Welt"  -> "Hallo Welt"
    ```

Bisher verwendete Klassen und Objekte

- `System.out` ist ein Objekt der Klasse `PrintStream` (in java.io). Wichtigste Methoden dieser Klasse sind `println()` mit verschiedenen Parametertypen.
- Die Klassen `Integer`, `Double` usw. repräsentieren primitive Datentypen als Klassen. Objekte entsprechen Werten des primitiven Typs.
  - `x = new Integer(3)` -> Objekt für Wert 3
  - `x.intValue()` -> `int`-Wert für das Objekt
  - `Integer.parseInt(str)` -> int-Wert des Strings

Zusammenfassung

- Restliche Kontrollstrukturen: switch, Schleifen
- Objektorientierte Programmierung
- Klassen und Objekte
- Methoden, Rekursion, Konstruktoren
- Schaut euch die mitgelieferten Klassen an.