Javakurs für Anfänger

Einheit 05: Programmablauf

Kyrill Schmid
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

1. Teil: Nutzereingaben

- EVA Prinzip
- Eingaben über die Konsole
- Eingaben über GUI
- Programmieraufgabe: Kreisberechnung II

2. Teil: Programmablauf

- 3 Grundstrukturen von Algorithmen
- Wie bisher: Anweisungsfolgen
- Blöcke und Sichtbarkeitsbereiche

Lernziele

- Erweiterte Kontrollstrukturen kennenlernen
- Sichtbarkeitsbereiche von Variablen verstehen
- Mit Auswahlstrukturen umgehen können

Programmieraufgabe (Letzte Stunde)

Lösen Sie bitte selbstständig die folgende Programmieraufgabe zu *Klassen & Objekten* in Java:

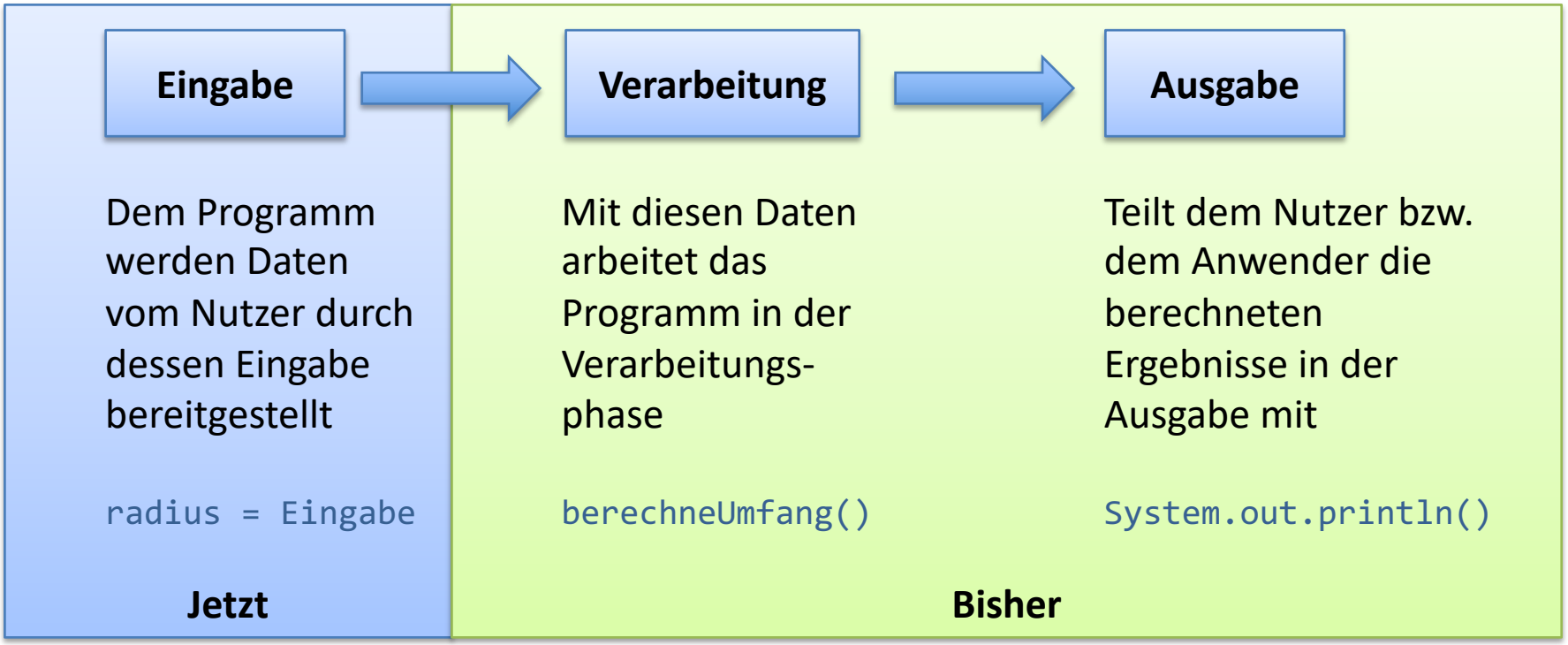
- Erstellen Sie ein neues Projekt „Uebung03“ mit einer Klasse `Kreis`
- Die Klasse `Kreis` besitzt:
  - Ein Attribut `radius`, welches den Radius des Kreises als Kommazahl `Double` speichert
  - Einen Konstruktor mit leerer Parameterliste, der den Radius mit 0 initialisiert
  - Einen zweiten Konstruktor, dem als Parameter eine Kommazahl zur Initialisierung des Radius übergeben wird.
  - Getter und Setter für den Radius
  - Eine Methode `double getUmfang()`, welche den Umfang des Kreises berechnet und zurückliefert
    - Hinweis: Kreisumfang = 2*pi*radius
    - Hinweis: pi kann als `double` mit 3.14 oder mittels `Math.PI` angegeben werden
  - Eine Methode `double getFlaeche()`, welche die Fläche des Kreises berechnet und zurückliefert
    - Hinweise: Kreisfläche = r^2*pi

- Erstellen Sie ein Testprogramm `KreisTest`, das:
  - Einen Kreis mit `radius` = 5 erzeugt
  - Und anschließend den Radius, den Umfang und die Fläche auf der Konsole ausgibt.

Benutzereingabe während des Programmablaufs

- Wir wollen das Kreis-Attribut `radius` nicht fest im Programmcode verbauen (*hardcoded*), sondern es erst zur Laufzeit durch eine Benutzereingabe einlesen!

Der Dialog mit dem Anwender (EVA Prinzip)



Nutzereingaben

Nutzereingabe über die Tastatur:

- Über die Konsole (Standardeingabe)
- Über eine grafische Benutzerschnittstelle **GUI** (*graphical user interface*)

1. Benutzereingabe über Konsole:

- `System.in` liefert uns den `InputStream` der Standardeingabe
  - analog zu `System.out` für `OutputStream` der Standardausgabe
- Wir bedienen uns der Java Klasse `Scanner` aus dem Package `java.util`, der wir im Konstruktor den `InputStream` der Standardeingabe übergeben:
  - `Scanner scan = new Scanner(System.in);`
- Mit den Methoden des Scanners können Eingabewerte gelesen werden
  - `scan.next()`: gibt nächstes Token als String zurück
    - Eine Zahl als Eingabe ist zunächst auch String
    - Kann mittels `Integer.parseInt(eingabe)` zu Integer umgewandelt werden
    - Analog für Kommazahl-Eingabe: `Double.parseDouble(eingabe)`
  - `Scan.nextInt()`: gibt nächstes Token als int zurück

Benutzereingabe über Konsole

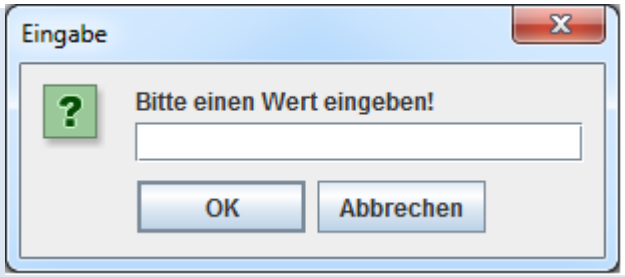

```
// Java Beispiel für Nutzereingabe über die Konsole

import java.util.Scanner; // Klasse Scanner muss importiert werden

// Objekt der Klasse Scanner mit Standardeingabe erzeugen
Scanner scan = new Scanner(System.in);

// Nutzer zur Eingabe auffordern:
System.out.println(„Bitte geben Sie einen Wert ein: “);

// Nutzereingabe lesen und als String speichern
String eingabe = scan.next();

// Nutzereingabe ausgeben
System.out.println(„Sie haben “+eingabe+“ eingegeben“);

// Oder Nutzereingabe in Integer umwandeln, falls Eingabe eine korrekte Zahl
int zahl_eingabe = Interger.parseInt(eingabe)
```

Programmieraufgabe:
Schreiben Sie Ihr Programm zur Kreisberechnung von vorhin so um, dass der Benutzer aufgefordert wird den Radius in der Konsole einzugeben. Das Programm soll dann den eingegebenen Wert verwenden, um den Radius, den Umfang und die Fläche auf der Konsole auszugeben!

Benutzereingabe über GUI

Die Klasse `JOptionPane` aus dem Packet `javax.swing` besitzt die Methode `showMessageDialog`:

- Zeigt ein grafisches Fenster mit einer Eingabeaufforderung
- Liefert nach klicken auf „ok“ die Eingabe als String zurück




```
// Beispiel für Nutzereingabe über GUI

// Klasse JOptionPane muss importiert werden!
import javax.swing.JOptionPane;

public class MeineKlasse{
   public static void main(String[] args){

      // Fenster zur Werteingabe erzeugen und Eingabe als String speichern
      String eingabe = JOptionPane.showInputDialog(„Bitte einen Wert eingeben!“);

      // Dann kann man wieder mit der Eingabe als String arbeiten. Bsp.:
      System.out.println(„Ihre Eingabe war “+eingabe);
   }
}
```

Programmieraufgabe:
Verwenden Sie nun das graphische Interface für die Abfrage des Radius und lassen Sie sich wieder den Radius, den Umfang und die Fläche des Kreises ausgeben.

Steuerung des Programmablaufs: Einführung in Kontrollstrukturen

Bisher: Einfacher sequentieller Ablauf von Befehlen

- Beispiel Student (letzte Stunde):

```
public class Hauptprogramm {

   //Main-Methode
   public static void main(String[] args){

      Student studi1 = new Student("Peter",3);
      Student studi2 = new Student("Hanna",5);

      studi1.lernen();
      studi2.lernen();
   }
}
```

In Zukunft wollen wir **komplexere** Abläufe programmieren können!

Komplexere Abläufe verlangen Kontrollstrukturen. Beispiele:

- Wenn ein Student im höheren Semester ist, dann…
- Ein Student lernt solange bis,….
- Alle Studenten machen ….

Überblick über Kontrollstrukturen

Ein Algorithmus lässt sich durch 3 Grundstrukturen beschreiben:

- Anweisungsfolge bzw. Sequenz
  - Wie bisher: Schrittweise Abarbeitung von Befehlen von oben nach unten
- Auswahlstruktur bzw. Selektion
  - Ermöglicht die **bedingte** Ausführung von Anweisungen
  - Bsp.: `if, else, switch-case`
- Wiederholungsstruktur bzw. Iteration oder Schleife
  - **Mehrmalige** Ausführung der gleichen Anweisungen
  - Beispiele: `while-, do-, for-`Schleifen

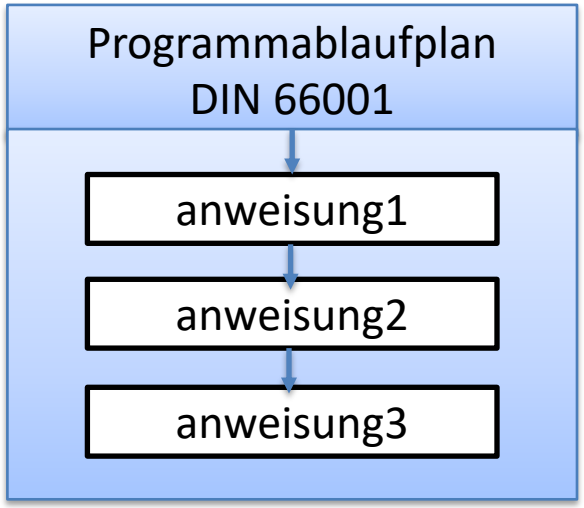
Anweisungsfolge (Sequenz)

Wie bei unseren bisherigen Programmen:

- Die im Quellcode vorhandenen Anweisungen werden sequentiell von oben nach unten abgearbeitet.
- Die Sequenz ist damit die einfachste Kontrollstruktur
- Beispiel:

```
anweisung1;

anweisung2;

anweisung3;
```

Darstellung als Programmablaufplan



Blöcke

Anweisungsfolgen können auch in Blöcken (mit geschweifte Klammern) zusammengefasst werden

- Werden von außen wie eine Einheit betrachtet
- Ein Block kann selbst Variablen definieren, die nur innerhalb des Blocks sichtbar sind
- Ein Block ist selbst wieder eine Anweisung
- Vorteile: Struktur und geschlossene Einheit
- Beispiel:

```
// Beispiel für Blöcke

anweisung1;
{               //Begin Block
   anweisung2;
   anweisung3;
}               //End Block
anweisung4;
```

Anweisungen 2 und 3 bilden eine Einheit

Sichtbarkeit von Variablen

Die Verwendung von Blöcken wirkt sich auf die **Sichtbarkeit** von Variablen aus!

- Instanzvariablen sind mit `this.instanzvariable` innerhalb der kompletten Klasse sichtbar, also auch in allen Blöcken und Unterblöcken
- Lokale Variablen sind nur innerhalb des Blocks (inklusive aller Unterblöcke) in dem sie definiert wurden sichtbar!

```
(…)//Beispiel
public void doMyVeryBest(int meinWert){

   this.meinWert = 7; // Belegt Instanzvariable mit 7

   int x = meinWert; // Belegt lokale Variabel x mit dem Parameterwert

   { //Begin Block

      int y = 5; //Int-Variable y wird definiert. Ist nur im Block sichtbar!
      int z = y+x; //Lokale Variable z = 5+x. x ist hier sichtbar, da Unterblock.

   } // End Block

   x = z; // Fehler: x ist zwar sichtbar, aber z nicht, da außerhalb des Blocks
}
```

Sichtbarkeit von x

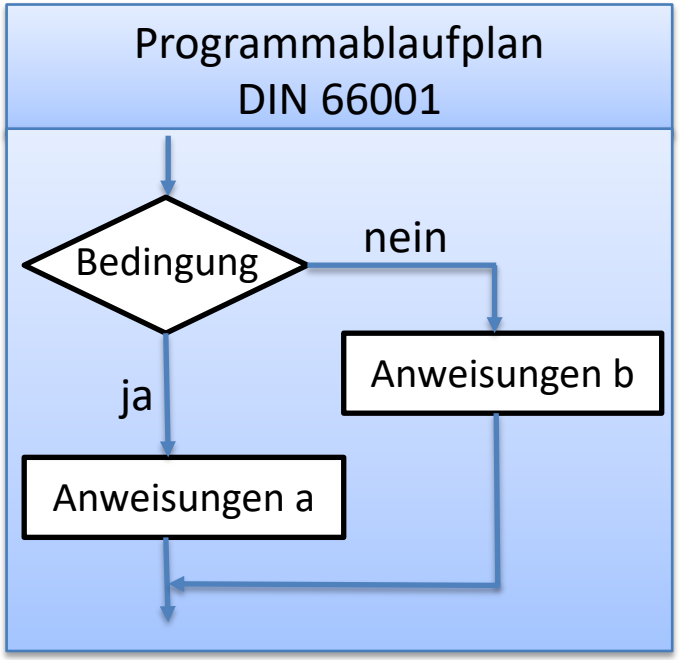
y

z

Auswahlstruktur bzw. Selektion

Auswahlstrukturen ermöglichen die bedingte Ausführung von Anweisungen

- Ausführung einzelner Anweisungen bzw. Blöcke wird von Erfüllung einer **Bedingung** abhängig gemacht
- Beispiele (im Folgenden):
  - `if`-Anweisung
  - Verschachteltet `if`-Anweisungen
  - `Switch-Case`-Anweisungen



Zweiseitige Auswahlstruktur

Was ist eine Bedingung?

Unter einer Bedingung versteht man einen beliebigen Ausdruck, dessen Auswertung einen Wahrheits- bzw. booleschen Wert (`true` oder `false`) liefert.

Was ist eine Bedingung?

Für eine Bedingung kann also entweder

- direkt eine Variable vom Typ `boolean` angegeben werden
- oder einen Methodenaufruf, der einen Wert vom Typ `boolean` zurückgibt
- oder man verwendet **Vergleichsoperatoren** bzw. **logische Operatoren**, wie in den folgenden Tabellen dargestellt:

Vergleichs-Operator	Bedeutung	Priorität
<	kleiner	5
<=	Kleiner gleich	5
>	größer	5
>=	größer gleich	5
==	gleich	6
!=	ungleich	6

Vergleichsoperatoren

Logischer-Operator	Bedeutung	Priorität
!	NICHT	1
&	UND (bitwise)	7
^	XOR (bitwise)	8
|	ODER (bitwise)	9
&&	UND	10
||	ODER	11

Logische Operatoren

Hinweis: Die Prioritäten können mit Hilfe von runden Klammern beeinflusst werden

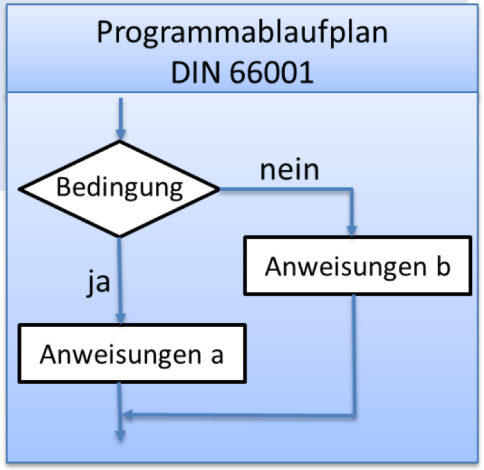
Beispiele


```
// Einfache Beispiele für Vergleiche:

a == 0;      // Ist der Wert von a 0?
b > 10;      // Ist b größer 10?
c != 5;      // Ist c ungleich 5?
zahl <= 100;// Ist zahl kleiner gleich 100?

// Einfache Beispiele für Logische Ausdrücke:

(a < 5) && (b > 2); // Ist a kleiner 5 UND b größer 2?

    // Kann auch ohne Klammern geschrieben werden, wegen Prioritäten!
    // >,< hat Priorität 5 und && hat geringere Priorität 10

zahl >= 100 || zahl < 10; // Ist zahl größer gleich 100 ODER kleiner 10?

a == b || b != c && c < 10; //Ist c ungleich b UND kleiner 10 ODER a gleich b

(a == b || b != c) && c < 10; // Ist a gleich b ODER b ungleich c UND c
                                 kleiner 10
```

Die `if`-Anweisung

Die `if`-Anweisung entscheidet anhand einer Bedingung, welche Anweisungen ausgeführt werden.

```
// Falls Bedingung zutrifft, führe
Anweisungen A1, A2 aus, sonst führe
Anweisungen B1, B2 aus!

if(Bedingung){

   anweisung_A1;
   anweisung_A2;

} else{
   anweisung_B1;
   anweisung_B2;
}
```



Zweiseitige Auswahlstruktur

```
// Konkretes Beispiel
// Entscheidet, ob i<20 oder größer
int i = 10;

if(i<20){

   System.out.println(„i ist kleiner
   20“);

} else{
   System.out.println(„i ist
   groesser gleich 20“);
}
```

Einfache `if`-Anweisungen

Es gibt auch einfache `if`-Anweisungen (ohne `else`-Block)

- Falls die Bedingung `true` ist, wird der `if`-Block ausgeführt, sonst wird dieser einfach übersprungen

```
if (Bedingung){

   anweisung_A1;
   anweisung_A2;

}

anweisung_B1;
anweisung_B2;
//…
```

```
// Konkretes Beispiel
// Falls i > 20, belege Variable s

int i = 10;
String s = „Ist kleiner gleich 20“;

if(i>20){
   s = „Ist groesser 20“;
}

System.out.println(s);
```

Verschachtelte `if`-Anweisungen

`if`-Anweisungen können auch verschachtelt werden

- Falls es mehr als eine Alternative (`else`) geben soll, aber nur **eine** ausgewählt werden darf
- Verwendung von `else if`-Anweisung
- Auch weitere `if`-Anweisungen im else-Block möglich

```
if (Bedingung 1){

   Anweisungen A;

} else if (Bedingung 2){

  Anweisungen B;

} else if (Bedingung XY){

  Anweisungen XYZ;

} else {

  sonstige Anweisungen

}
```

```
// Konkretes Beispiel
// Zahl prüfen

//int zahl ist definiert

if (zahl == 0){

   System.out.println(„Zahl ist 0“);

} else if (zahl > 0){

   Sytem.out.println(„Zahl ist
positiv“);

} else{

   System.out.println(„Zahl ist
negativ“)

}
```

Verschachtelte `if`-Anweisungen

Ein paar Regeln:

- Es kann beliebig viele `else-if` Anweisungen geben
- Der letzte `else`-Block ist wieder optional und bezieht sich auf die letzte `if`-Bedingung

Unterschied zwischen `if` und `else if` Anweisungen:

- Alle `if`-Blöcke werden ausgeführt sobald die entsprechend Bedingung erfüllt ist, also `true` ergibt
- Nur ein `else if` Block wird ausgeführt, wenn die entsprechende Bedingung `true` ergibt
- Mit `else if` Anweisungen können wir also gegenseitige Ausschlusskriterien definieren

Übungen zu `if`-Anweisungen

Erstellen Sie ein neues Projekt in Eclipse mit Namen Uebung05

Aufgabe 1:

- Erstellen Sie in diesem Projekt ein Programm (Main-Methode) mit dem Namen *Zahlentest*
- Der Nutzer soll dabei 2 Zahlen vom Typ integer eingeben
- Nun sollen folgende Fälle geprüft werden:
  - Sind beide Zahlen gleich, dann wird auf der Konsole ausgegeben, dass die Zahlen gleich sind
  - Ist eine Zahl größer als die andere, dann soll die größere der beiden Zahlen mit einem entsprechenden Hinweis (bspw.: „Die größere Zahl lautet:“) auf der Konsole ausgegeben werden.