Blöcke, Sichtbarkeit und Methoden

Inhalt - Überblick

Rückblick

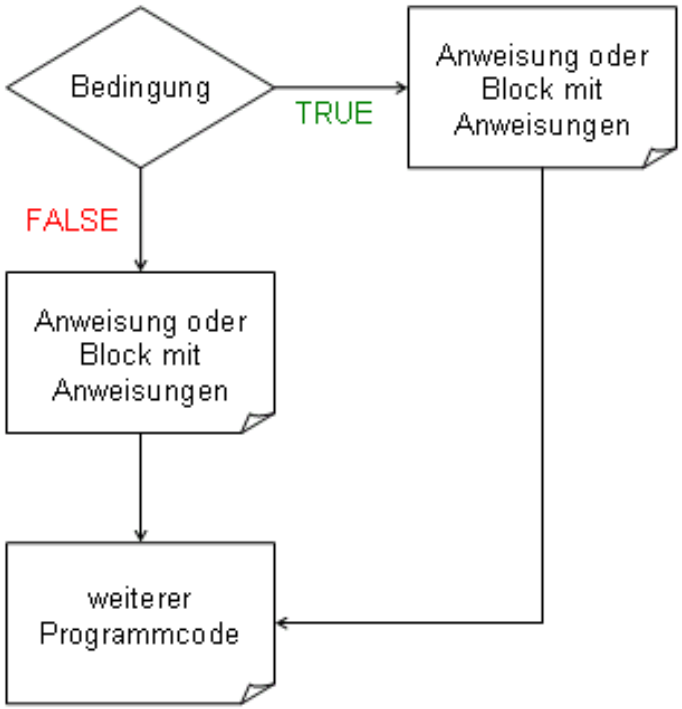
- Variablen und Arrays
- Ausdrücke und Operatoren (+, ++, %, !, …)
- Verzweigungen (if, switch, …)
- Schleifen (while, for, …)

Logische Operatoren (WH)

Verknüpfung von booleschen Werten (true / false)

- Logische Negation: !
- Logisches Und: && (mit short circuit evaluation)
- Logisches Oder: || (mit short circuit evaluation)
- Logisches Und: & (ohne short circuit evaluation)
- Logisches Oder: | (ohne short circuit evaluation)
- Exklusiv-Oder: ^

Ausführen, wenn eine Bedingung wahr ist; ansonsten einen anderen Block ausführen:

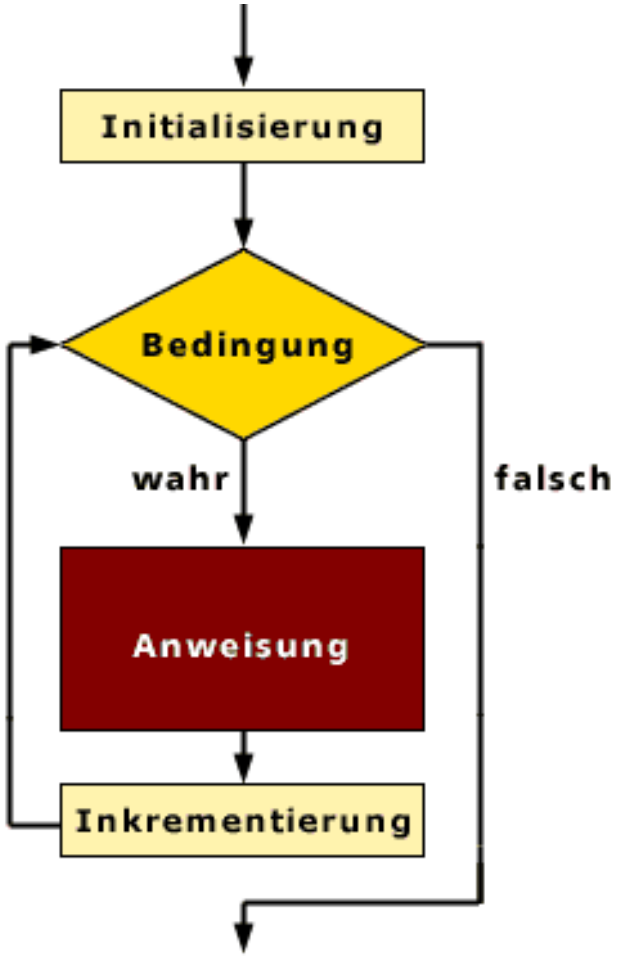



```
if (bedingung){
    anweisung 1;
} else {
    anweisung 2;
}
```

Die Anweisung 1 wird ausgeführt, wenn die Bedingung wahr (true) ist. Ansonsten wird die Anweisung 2 ausgeführt.

[Class18.java]

For Schleife (WH)




```
for(init; test; update) {
    anweisung;
}
```

- init wird einmal vor dem Start der Schleife ausgeführt
- test (Bedingung) wird vor jedem Durchlauf getestet
- update wird nach jedem Durchlauf ausgeführt
- Alle drei Komponenten sind optional; eine fehlende Laufbedingung (test) wird durch true ersetzt, es entsteht eine Endlosschleife

[Class20.java]

Blöcke

Blöcke

- Zusammenfassung von Anweisungen
- Können dort im Code vorkommen, wo auch einzelne Anweisungen vorkommen
- Ein Block beginnt mit { und endet mit }:

```
{
        Anweisung1;
        Anweisung2;
        …
}
```

[Class18.java]

Beispiel für einen Block

```
// Ein Block
{
    System.out.println("(1) Zahl 1 eingeben: ");
    int zahl1 = sc.nextInt();

    System.out.println("(1) Zahl 2 eingeben: ");
    int zahl2 = sc.nextInt();

    System.out.println("(1) Zahl 1: " + zahl1 + " - Zahl 2: " + zahl2);
}

// Ein weiterer Block
{
    System.out.println("(2) Zahl 1 eingeben: ");
    int zahl1 = sc.nextInt();

    System.out.println("(2) Zahl 2 eingeben: ");
    int zahl2 = sc.nextInt();

    System.out.println("(2) Zahl 1: " + zahl1 + " - Zahl 2: " + zahl2);
}
```

[Class25.java]

Bekannte Blöcke

```
Scanner sc = new Scanner(System.in);

System.out.println("Zahl 1 eingeben: ");
int zahl1 = sc.nextInt();

System.out.println("Zahl 2 eingeben: ");
int zahl2 = sc.nextInt();

if (zahl1 > zahl2) {
    System.out.println("Zahl 1 ist größer als Zahl 2");
    System.out.println("Zahl 3 eingeben: ");

    int zahl3 = sc.nextInt();
    if (zahl3 > zahl2){
        System.out.println("... und Zahl 3 ist noch größer.");
    }
}
```

[Class26.java]

Bekannte Blöcke

```
int i = 0;
while (i < 9) {

    System.out.println("Eingabe: ");

    Scanner sc = new Scanner(System.in);
    array[i] = sc.nextLine();

    System.out.println(array[i]);
    i++;
}
```

[Class20.java]

Blöcke und Variablen-Sichtbarkeit

Sichtbarkeit von Variablen

```
int zahl1 = 10;

{
    int zahl2 = 20;
    System.out.println("(1) Zahl 1: " +  zahl1);
    System.out.println("(1) Zahl 2: " +  zahl2);
}
{
    int zahl3 = 30;
    System.out.println("(2) Zahl 1: " +  zahl1);
    System.out.println("(2) Zahl 3: " +  zahl3);
    {
        int zahl4 = 40;
        System.out.println("(3) Zahl 1: " +  zahl1);
        System.out.println("(3) Zahl 3: " +  zahl3);
        System.out.println("(3) Zahl 4: " +  zahl4);
    }
}
```

[Class27.java]

Sichtbarkeit von Variablen

```
Scanner sc = new Scanner(System.in);

System.out.println("Zahl 1 eingeben: ");
int zahl1 = sc.nextInt();

System.out.println("Zahl 2 eingeben: ");
int zahl2 = sc.nextInt();

if (zahl1 > zahl2) {
    System.out.println("Zahl 1 ist größer als Zahl 2");
    System.out.println("Zahl 3 eingeben: ");

  int zahl3 = sc.nextInt();
  if (zahl3 > zahl2){
      System.out.println("... und Zahl 3 ist noch größer.");
  }
}
```

[Class26.java]

Sichtbarkeit von Variablen

- Die Variablen eines Blocks sind in inneren Blöcken sichtbar
- Das gilt auch für tiefer verschachtelte Blöcke, z. B.

  – Die Variablen des Blocks der Ebene 1 sind auch in den Blöcken der Ebene 2 und 3 sichtbar
- Die Variablen eines Blocks sind nicht in äußeren oder benachbarten Blöcken sichtbar

  – Verwendete Variablennamen können an diesen Stellen für andere Zwecke werden

Wiederverwendung des Variablennamens **zahl1** in zwei benachbarten Blöcken

```
{
    int zahl1 = 20;
    System.out.println("(1) Zahl 1: " +  zahl1);
}
{
    int zahl1 = 40;
    System.out.println("(2) Zahl 1: " +  zahl1);
}
```

[Class28.java]

Funktionen/Methoden (Teil 1)

Methoden/Funktionen

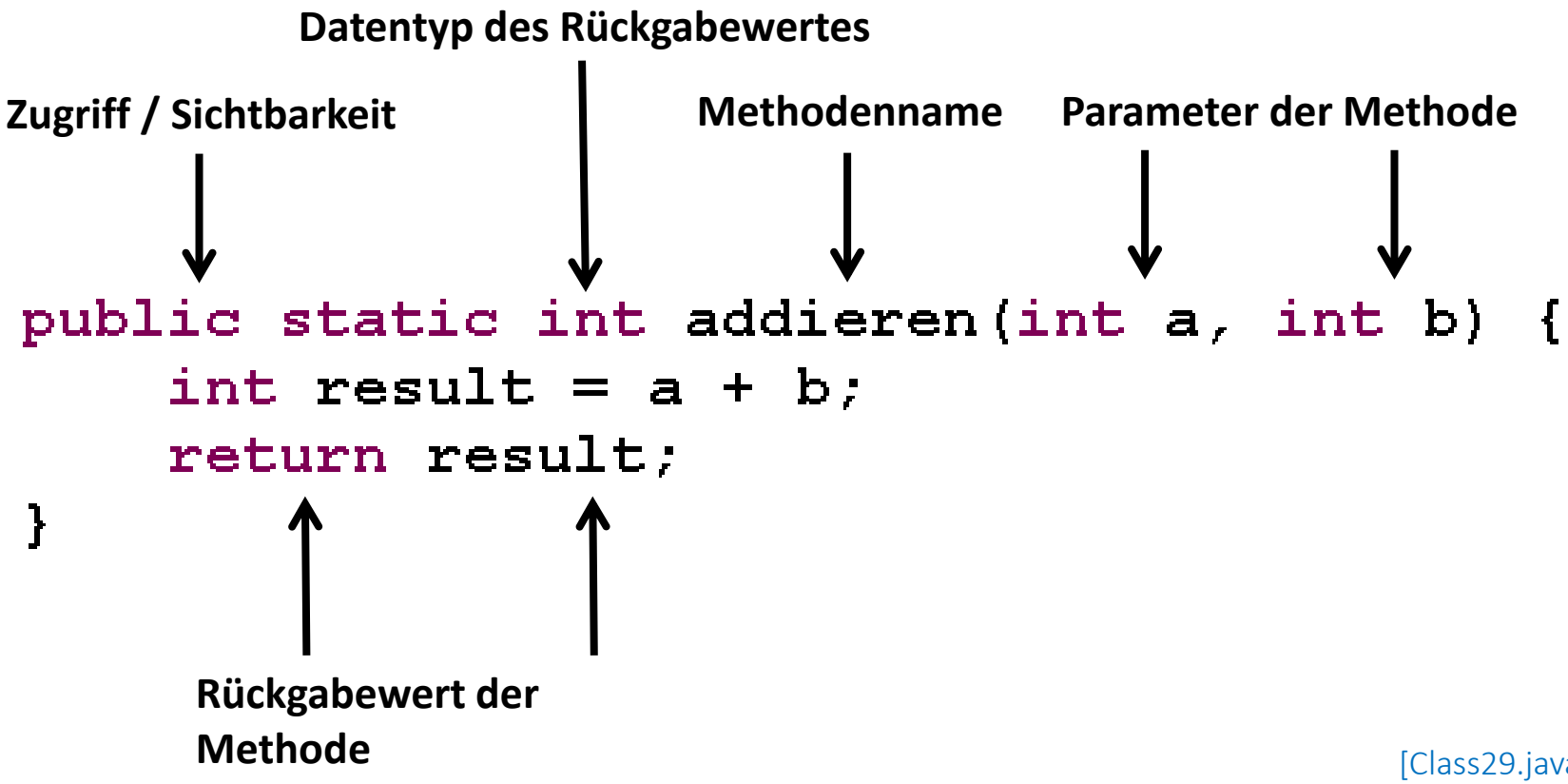
- Methoden bzw. Funktionen lösen Teilaufgaben eines Programms

Beispiele:

- Zwei Zahlen addieren
- Eine Textausgabe erzeugen
- Eine Datei kopieren und umbenennen
- …

- Methoden sollten einen selbstsprechenden Namen haben. Sie beginnen in der Regel mit einem Kleinbuchstaben.

Aufbau einer Methode



[Class29.java]

Methoden: Zugriff/Sichtbarkeit

… später

Methoden: Rückgabewert

- Eine Methode übernimmt eine bestimmte (Teil-)Aufgabe. Als Ergebnis kann sie einen Rückgabewert liefern.
- In der Deklaration der Methode wird angegeben, welchen Datentyp der Rückgabewert hat

```
public static int addieren(int a, int b) {
    int result = a + b;
    return result;
}
public static String baueUhrzeit() {
    Date date = new Date();
    String zeit = date.toString();
    return zeit;
}
```

[Class29.java]

Methoden: Rückgabewert

- Die Methode kann einen Rückgabewert errechnen und diesen als Ergebnis liefern. Dieser entspricht dem Datentyp aus der Methoden-Deklaration
- Um einen Rückgabewert als Ergebnis zu liefern wird das Schlüsselwort **return** genutzt

```
public static int addieren(int a, int b) {
    int result = a + b;
    return result;
}
public static String baueUhrzeit() {
    Date date = new Date();
    String zeit = date.toString();
    return zeit;
}
```

[Class29.java]

Methoden: Rückgabewert

- Methoden, die keinen Rückgabewert haben, nutzen das Schlüsselwort **void** anstelle des Datentyps
- Diese dürfen kein Ergebnis mit Hilfe von return liefern

```
public void textAusgabe() {
    System.out.println("Textausgabe");
}
```

[Class29.java]

Methoden: Parameter

- Jeder Methode können Parameter übergeben werden
- Für jeden übergebenen Parameter wird der Datentyp angegeben
- Die einzelnen Parameter werden durch „,“ getrennt angegeben und mit einem Namen versehen, der für den Parameterzugriff innerhalb der Methode verwendet wird

```
public static int addieren(int a, int b) {
    int result = a + b;
    return result;
}
```

Parameter-Zugriff (vgl. Variablen)

[Class29.java]

Aufrufen von Methoden

- Eine Methode wird durch Angabe des Methodennamens aufgerufen
- Falls die Methode Parameter besitzt, müssen diese übergeben werden (mit dem korrekten Datentyp)
- Methoden können andere (Sub-)Methoden aufrufen. So entsteht eine Aufrufhierarchie.

- Beispiel: Siehe Class30.java

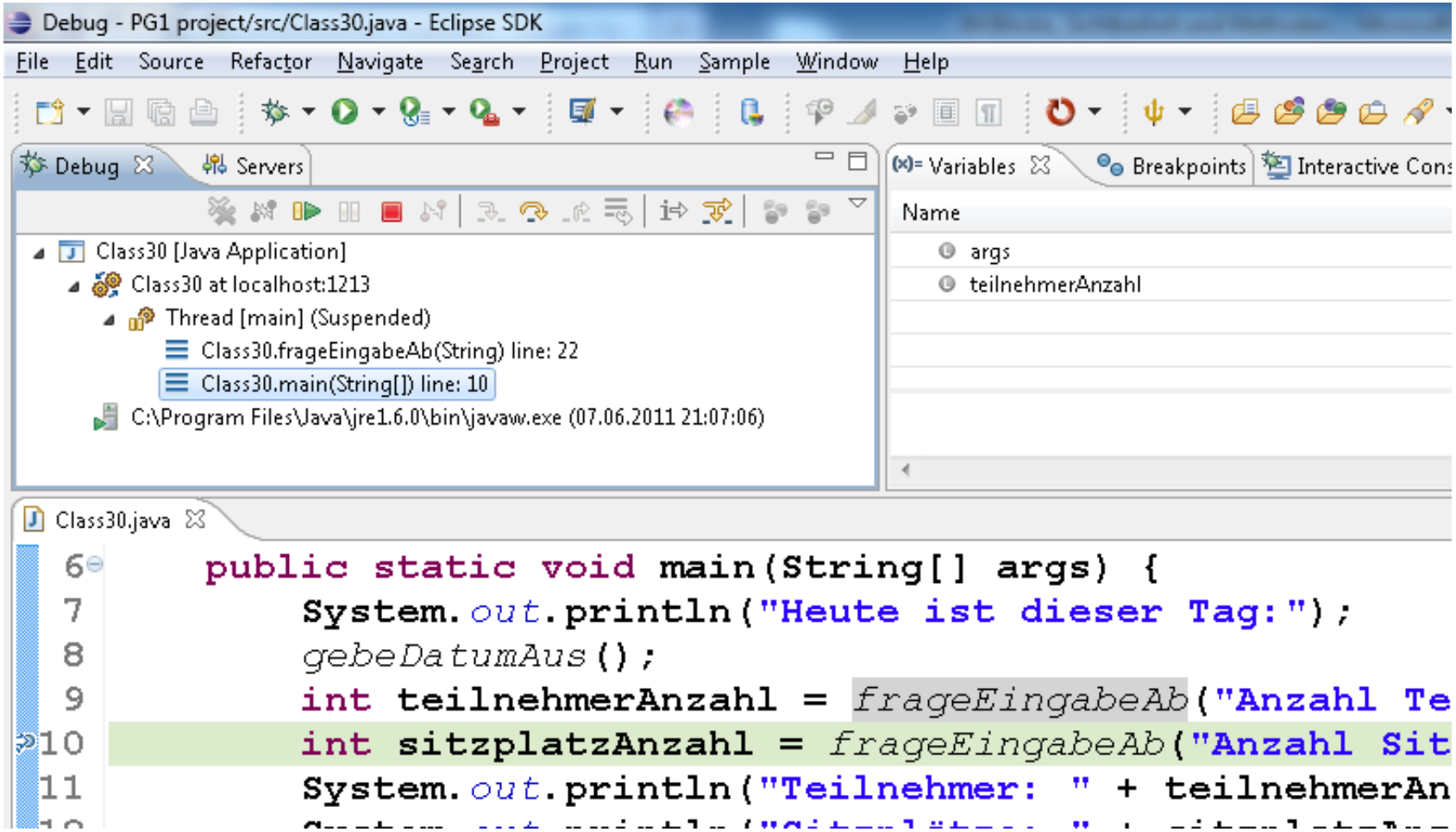
[Class30.java]

Beispiel

```
public static void main(String[] args) {
    System.out.println("Heute ist dieser Tag:");
    gebeDatumAus();
    int teilnehmerAnzahl = frageEingabeAb("Anzahl Teilnehmer");
    int sitzplatzAnzahl = frageEingabeAb("Anzahl Sitzplätze");
    System.out.println("Teilnehmer: " + teilnehmerAnzahl);
    System.out.println("Sitzplätze: " + sitzplatzAnzahl);
}
public static void gebeDatumAus() {
    Date date = new Date();
    String dateString = date.toString();
    System.out.println(dateString);
}
public static int frageEingabeAb(String aufforderung) {
    Scanner sc = new Scanner(System.in);
    int result = 0;
    System.out.print("Bitte geben Sie ");
    System.out.print(aufforderung);
    System.out.println(" ein: ");
    result = sc.nextInt();
    return result;
}
```

[Class30.java]

Aufrufhierarchie / Debuggen



[Class30.java]

Aufrufhierarchie bei Exceptions

```
public static int frageEingabeAb(String aufforderung) {
    Scanner sc = new Scanner(System.in);
    int result = 0
    System.out.print("Bitte geben Sie ");
    System.out.print(aufforderung);
```

Stacktrace (siehe auch http://de.wikipedia.org/wiki/Stacktrace)

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Syntax error, insert ";" to complete LocalVariableDeclarationStatement

        at Class30.frageEingabeAb(Class30.java:23)
        at Class30.main(Class30.java:9)
```

[Class30.java]