Programmiermethodik 1
Programmiertechnik

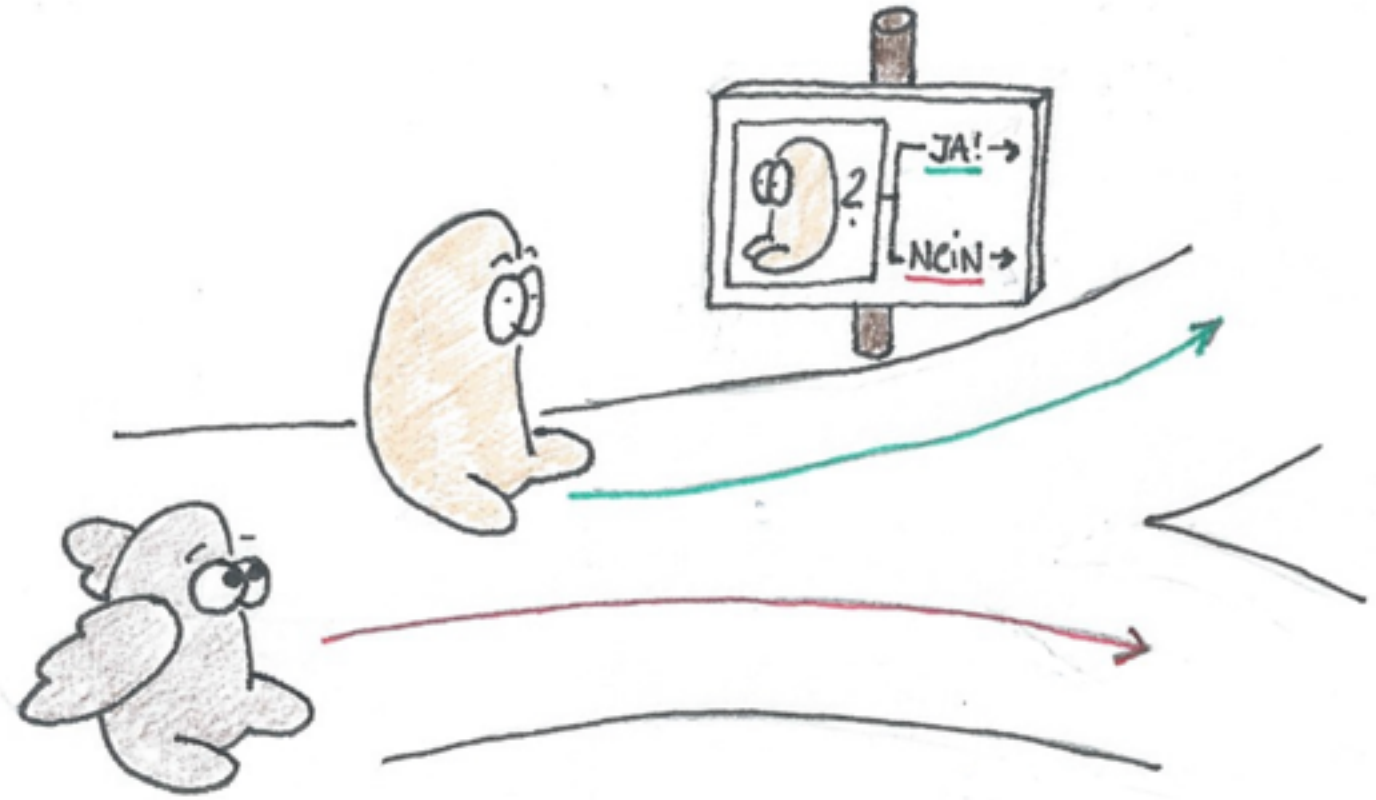
Wahrheitswerte, Bedingte Anweisungen

Wiederholung

- Arithmetische Ausdrücke
- Fließkommazahlen
- Kompatibilität

Ausblick

Worum gehts?



Agenda

- Wahrheitswerte
- Bedingte Anweisungen
- Weitere Konstrukte

Wahrheitswerte

Wahrheitswerte

- Datentyp boolean
- Ausdruck mit Wahrheitswert als Ergebnis
- eigenständiger Datentyp: boolean
- boolean hat nur zwei Werte:
    - wahr
    - falsch
- boolean-Literale:
    - true (wahr, ja, zutreffend)
    - false (falsch, nein, unzutreffend)
- boolean kein numerischer Typ
    - nicht kompatibel zu int oder double

Wahrheitswerte

- Variablen mit Typ boolean sind zulässig
- Beispiel:
    - boolean istOk;
    - istOk = true;
- ebenso mit Initialisierung:
    - boolean istOk = true;
- Zuweisung von Bedingungen an boolean-Variablen möglich
    - Vergleichsoperatoren liefern Wahrheitswert
    - boolean gueltigeTemperatur = celsius > -273.16;

Relationale Operatoren

- neue Art von Operator notwendig
    - Vergleich von Werten
    - Ergebnis: Wahrheitswert
    - relationaler Operator oder Vergleichsoperator

- relationale Operatoren erwarten numerische Operanden und liefern Wahrheitswerte (boolean)

Relationale Operatoren

Relationale Operatoren

Syntax	Art des Vergleichs
<	echt kleiner
<=	kleiner oder gleich
>	echt größer
>=	größer oder gleich
==	gleich
!=	nicht gleich

- häufige Fehler:
  - Gleichheitsrelation
    - '==' in Java
    - entspricht '=' in der Mathematik
  - Wertzuweisung = in Java hat keine mathematische Entsprechung

Relationale Operatoren

- relationale Operatoren bilden eine neue Gruppe von Operatoren:
  - Operanden sind Zahlen, Ergebnis ist Wahrheitswert
  - Erinnerung: arithmetische Operatoren
  - Operanden sind Zahlen, Ergebnis ist Zahl
- Priorität ist niedriger als bei arithmetische Operatoren
  - Beispiel: 2+3 < 2*3 → 5 < 6 → true
  - Einzelheiten siehe Operatorentabelle (siehe EMIL)

Logische Operatoren

- logische Operatoren verknüpfen Wahrheitswerte
  - Operanden sind Wahrheitswerte, Ergebnis ist Wahrheitswert

Operator	Name	deutsch	Ergebnis ist true genau dann, wenn …
&&	AND	logisches Und	… alle beide Operanden true sind
||	OR	inklusives logisches Oder	… mindestens ein Operand true ist
^	XOR	exklusives logisches Oder	… genau ein Operand true ist
!	NOT	logisches Nicht	… der Operand false ist

Wahrheitstabellen

- Wahrheitstabellen ordnen jeder möglichen Kombination von Operanden ein Ergebnis zu
  - beschreiben logische Operatoren damit vollständig
- Beispiel AND
  - true && true → true
  - true && false → false
  - false && true → false
  - false && false → false

Wahrheitstabellen – OR und XOR

- Beispiel OR
  - true || true → true
  - true || false → true
  - false || true → true
  - false || false → false

- Beispiel XOR
  - true ^ true → false
  - true ^ false → true
  - false ^ true → true
  - false ^ false → false

Logische Ausdrücke

- logische Operatoren dienen zur Formulierung zusammengesetzter Bedingungen
    - sogenannte logische Ausdrücke
- Beispiel: $-5 \leq x < 5$
    - in Worten: x ist größer oder gleich –5 und x ist kleiner als +5
        - als logischer Java-Ausdruck: (x >= -5) && (x < 5)

Übung: Logische Ausdrücke

- Formulieren Sie für die Variablen
  - int zahl1;
  - int zahl2;
  - boolean wahrheitswert;
- folgenden Ausdruck in Java-Syntax:
  - "zahl1 ist größer als zahl2 und außerdem ist wahrheitswert falsch."

Operatorgruppen

- Operatoren fallen (bisher) in drei Gruppen:

Gruppe	Operatoren	Typen
arithmetisch	+, -, *, /, %	numerisch → numerisch
relational	<, >, <=, =>, == , !=	numerisch → boolean
logisch	&&, ||, ^, !, &, |	boolean → boolean

- zusätzlich:
  - Zuweisungsoperator =
  - == und != können alle Datentypen vergleichen

Vergleich von Fließkomma-Werten

- relationale Operatoren sind polymorph
    - können ganze Zahlen und Fließkomma-Werte vergleichen
- gemischte Operanden
    - implizite Typkonversion zu Fließkomma-Werten
    - aber: Rundungsfehler bei Fließkomma-Werten!

Vergleich von Fließkomma-Werten

- Beispiel:
- double a = 1.0 / 7.0;
- double b = a + 1.0;
- double c = b - 1.0;
- a == c?
- Ergebnis
  - Bedingung nicht erfüllt, weil das Zwischenergebnis b eine zusätzliche gültige Stelle vor dem Komma braucht und damit am Ende eine Stelle verliert:
  - a: 0.14285714285714285
  - b: 1.1428571428571428
  - c: 0.142857142857142 8

Vergleich von Fließkomma-Werten

- Vergleich von exakten Fließkomma-Werten (==, !=)
    - sehr heikel
- Empfehlung
    - Fließkomma-Werte in Bereichen prüfen, nicht auf Einzelwerte!
- Beispiel:
    - (Math.abs(a – c) < 1e-10)statt (a == c)
        - Ausgabe: a gleich c

Bedingte Anweisungen

Bedingte Anweisung

- Falls <Bedingung> dann <tu-dies> ansonsten <tu-das>.
- if-Anweisung, Alternative, bedingte Anweisung, Verzweigung
- besteht aus
    - 1. Bedingung: (engl. condition)
    - 2. Konsequente: untergeordneter Anweisung
- untergeordnete Anweisung wird nur dann ausgeführt
    - wenn die Bedingung zutrifft
    - andernfalls übergangen
- Syntax:
    - if (<Bedingung>) <Anweisung>

Beispiel

```
double temperatur = …;
if ( temperatur < 0 ){
   System.out.println("Es gibt Schnee!“);
}
```

- Der Text wird nur ausgegeben, wenn die Variable temperatur einen negativen Wert hat

Bedingung

- neue Art von Ausdruck:
    - „trifft zu“ oder „trifft nicht zu“
    - keine dritte Möglichkeit
- Bedingung
    - Ausdruck mit ja/nein-Ergebnis
    - z.B. aufgrund eines Vergleichs
- Ergebnis der Auswertung des Bedingungsausdrucks
    - z.B. Durchführung des Vergleichs
    - keine Zahl, sondern ein Wahrheitswert (true/false)

Beispiel: Berechnung der Tage eines Monats

```
int monat = 7;
int tage = -1;
if ((monat == 4) || (monat == 6) || (monat == 9) ||
   (monat == 11)) {
   tage = 30;
}
if (monat == 2) {
   tage = 28;
}
if ((monat == 1) || (monat == 3) || (monat == 5) ||
   (monat == 7) || (monat == 8) || (monat == 10) ||
   (monat == 12)) {
   tage = 31;
}
System.out.println("Der Monat hat " + tage + " Tage.");
```

Teilweise Auswertung

- Einzelbedingungen in logischen Ausdrücken sind oft voneinander abhängig
- Beispiel
  - if(b != 0 && a/b > 0) …
  - Standard:
    - zuerst beide Operanden auswerten: b != 0 und a/b > 0
    - dann Ergebnisse mit AND verknüpfen
  - hier Problem, falls b == 0:
    - Division durch 0: Programmabbruch!
- Lösung: teilweise Auswertung (shortcut evaluation)
  - Auswertung wird beendet, wenn das Ergebnis nach dem ersten Operanden feststeht
  - der verbleibende, zweite Operand wird dann nicht mehr berechnet

Teilweise Auswertung

Teilweise Auswertung

- bei &&:
  - erster Operand false: logischer Ausdruck ist false
- bei ||:
  - erster Operand true: logischer Ausdruck ist true
- nicht möglich bei ^ (kein vorzeitiges Ergebnis ableitbar)
- Steuerung teilweiser/vollständiger Auswertung durch Operatorenwahl:
  - && und || werten teilweise aus
- Beispiel
  - // 1. Operand ausgewertet, Ausdruck falsch falls b == 0
  - if (b != 0 && a/b > 0) …

Übung: Max3If

- Erstellen Sie ein Programm Max3If, das von 3 übergebenen Integer-Werten den größten Wert ermittelt und ausgibt!

- Anforderungsanalyse
    - Eingabe:
        - Der Benutzer gibt 3 ganzzahlige Werte ein
    - Ausgabe:
        - Der größte der drei Werte wird ausgegeben

- Verwenden Sie für Ihren Algorithmus keine Bibliotheksmethode!

Zweiseitige Bedingte Anweisung

- enthält eine Bedingung und zwei untergeordnete Anweisungen
- wenn die Bedingung zutrifft
    - wird die erste Anweisung ausgeführt ("then-Fall", Konsequente)
    - andernfalls die zweite ("else-Fall", Alternative)
- Syntax:

  if (<Bedingung>) <Anweisung> // Konsequente

  else <Anweisung> // Alternative

Zweiseitige Bedingte Anweisung

- Beispiel
  - x enthält einen beliebigen Wert
  - in a soll dessen Absolutwert (Betrag) berechnet werden:

```
double x = -23;
double absolutWert;
if (x >= 0) {
   absolutWert = x;
   System.out.println("x war positiv oder 0");
} else {
   absolutWert = -x;
   System.out.println("x war negativ");
}
System.out.println("Der Absolutwert lautet: " + absolutWert);
```

- es wird immer genau eine der beiden Anweisungen ausgeführt
  - niemals beide
  - niemals keine

Geschachtelte Bedingte Anweisung

- if-Anweisung ist selbst eine Anweisung
    - kann daher einer anderen untergeordnet werden
    - geschachtelte if-Anweisungen

Beispiel

- Berechnung des Quartals aufgrund der Monatszahl

```
int monat = 7;
int quartal = -1;
if (monat <= 3) {
   quartal = 1;
} else {
   if (monat <= 6) {
      quartal = 2;
   } else {
      if (monat <= 9) {
         quartal = 3;
      } else {
         quartal = 4;
      } // monat <= 9
   } // monat <= 6
} // monat <= 3
System.out.println("Quartal für Monat " +
   monat + ": " + quartal);
```

Best Practices

- hohe Schachtelungstiefe vermeiden !
    - unübersichtlich!
    - es stehen noch andere Konstrukte zur Verfügung
- Alternative und Konsequente immer als Block in geschweifte Klammern setzen!
    - erzeugt Klarheit
    - hilft, falls später Anweisungen hinzukommen
        - z.B. Ausgabeanweisungen zum Testen
    - Ausnahme/Sonderfall: Alternative ist eine bedingte Anweisung
        - dann für bessere Lesbarkeit: auf geschweifte Klammern verzichten

Beispiel

- Berechnung des Quartals aufgrund der Monatszahl

```
if (monat <= 3) {
   quartal = 1;
} else if (monat <= 6) {
   quartal = 2;
} else if (monat <= 9) {
   quartal = 3;
} else {
   quartal = 4;
}
```

Dreistelliger Bedingter Operator

Dreistelliger Bedingter Operator

- ähnlich if-then-else, wertet nur einen von zwei Ausdrücken aus
- Syntax
    - <Bedingung> ? <Konsequente> : <Alternative>
- Ablauf
    - Bedingung auswerten
    - falls wahr: Ja-Ausdruck auswerten und zurückliefern
    - falls falsch: Nein-Ausdruck auswerten und zurückliefern

Dreistelliger Bedingter Operator

- Beispiele:

  ```
  int a = …;
  int b = (a == 0)? 1 : 2;
  System.out.println(b != 1? "ungleich 1": "gleich 1");
  ```

- Problem:
  - leicht unübersichtlich
  - daher: nur mit sehr einfachen (kurzen) Ausdrücken verwenden
  - im Zweifelsfall vermeiden!

Übung: Gerade/Ungerade Zahl

- Gegeben ist eine Variable:
    - int zahl;
- Schreiben Sie eine Anweisung, die entweder "gerade" oder "ungerade" auf der Konsole ausgibt, je nachdem, ob der Wert von zahl gerade oder ungerade ist. Verwenden Sie den dreistelligen bedingten Operator.

Switch

Umfangreiche if-else-Kaskaden

- Fallunterscheidungen mit vielen Fällen werden unübersichtlich
    - oft je ein ... else if ... für jeden Fall
- Java bietet Konstrukt, das den Code übersichtlicher macht

switch-Anweisungen

- Syntax
  - switch-Anweisungen ersetzen längere if-Kaskaden

```
switch (<Ausdruck>) {
      case <Konstante>:
             [<Anweisung>]
             [...]
             [break;]
      case <Konstante>:
             [<Anweisung>]
             [...]
             [break;]
      ...
      [default:]
             [<Anweisung>]
             [...]
}
```

switch-Anweisungen

- Ablauf (Semantik)
  - der Wert des Ausdrucks wird einmal berechnet
  - das Ergebnis des Ausdrucks wird nacheinander mit den case-Konstanten ("Labels", "Sprungmarken") verglichen
    - dies können beliebig viele sein
  - case-Konstante mit dem ersten übereinstimmenden Wert:
    - folgende Anweisungen werden ausgeführt
    - dies können beliebig viele sein
  - eine break-Anweisung beendet die switch-Anweisung sofort
    - wie bei Schleifen
- break steht üblicherweise am Ende einer case-Anweisungsfolge
  - ohne break werden die Anweisungen des nächsten case ebenfalls ausgeführt

switch-Anweisungen

- Berechnung der Anzahl Tage im Monat:

```
switch(monat){
    case 1:
        tage = 31;
        break;
    case 2:
        tage = 28;
        break;
    case 3:
        tage = 31;
        break;
    …
    case 12:
        tage = 31;
}
```

case-Konstanten

- case-Konstanten müssen eindeutig sein, doppelte Werte unzulässig
- Wenn keine case-Konstante passt, geschieht nichts
    - dann: ganzes switch wirkt wie eine leere Anweisung
- leere case-Anweisungsfolgen sind zulässig
- Beispiel:

```
switch(monat){
      case 1: case 3: case 5:  case 7:  case 8: case 10: case 12:
              tage = 31;
              break;
      case 2:
              tage = 28;
              break;
      case 4: case 6: case 9: case 11:
              tage = 30;
              break;
}
```

Default-Fall

- default = Standardfall
    - hier: spezielle case-Konstante, passt auf alle übrigen Werte
- default darf nur einmal und nur am Ende genannt werden

```
switch(monat){
      case 2:
             tage = 28;
             break;
      case 4: case 6: case 9: case 11:
             tage = 30;
             break;
      default:  // alle sonstigen Monate
             days = 31;
}
```

Zulässige Datentypen

- Ergebnistyp des Ausdrucks im Kopf der switch-Anweisung und der Typ der case-Konstanten müssen übereinstimmen
- zulässige Typen :
    - einfache ganzzahlige Typen (byte, short, int, char)
    - Aufzählungstypen
    - Strings
- nicht zulässig (u.a.):
    - float, double: Test von exakten Werten problematisch ⇒ Rundungsfehler
    - boolean: nur zwei Werte

Syntaktischer Zucker

- Switch ist ein Beispiel für Syntaktischen Zucker
  - keine neue Funktionalität, erweitert nicht die Sprache
  - aber: Code wird einfacher/lesbarer/übersichtlicher

- Syntaktischer Zucker sind Syntaxerweiterungen in Programmier-sprachen, welche der Vereinfachung von Schreibweisen dienen. Diese Erweiterungen sind alternative Schreibweisen, die aber nicht die Ausdrucksstärke und Funktionalität der Programmiersprache erweitern. (Wikipedia)

Übung: Switch

- Schreiben Sie eine Switch-Anweisung, die für eine Variable
- int zahl;
- folgende Ausgaben auf der Konsole generiert:
  - falls zahl == 1 → "Eins"
  - falls zahl == 2 → "Zwei"
  - ansonsten → "Alles andere"

Zusammenfassung

- Wahrheitswerte
- Bedingte Anweisungen
- Weitere Konstrukte
    - dreistelliger bedingter Operator
    - Switch