Verzweigungen

Einführung in die Programmierung 1

Wintersemester 21/22

Überblick

- Motivation
- if/else-Anweisung
- switch-Anweisung
- Programmierstil
- Programmiermuster

Motivation

Elementare Anweisungen (Wiederholung)

- Zuweisung
- Ausdrucksanweisung

```
Ausdruck;
```

  - Nur sinnvoll, wenn der Ausdruck Seiteneffekte hat (z. B. i++)

Elementare Anweisungen (Wiederholung)

- Block
  - Zusammenfassung von aufeinander folgenden Anweisungen

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

- Leere Anweisung

```
;
```

Motivation für Verzweigungen

- Bisher wurden die einzelnen Zeilen eines Programms sequentiell abgearbeitet
- Sehr oft sollen aber bestimmte Anweisungen situationsabhängig ausgewählt werden
- Mit Verzweigungen kann der Ablauf des Programms situationsabhängig gesteuert werden

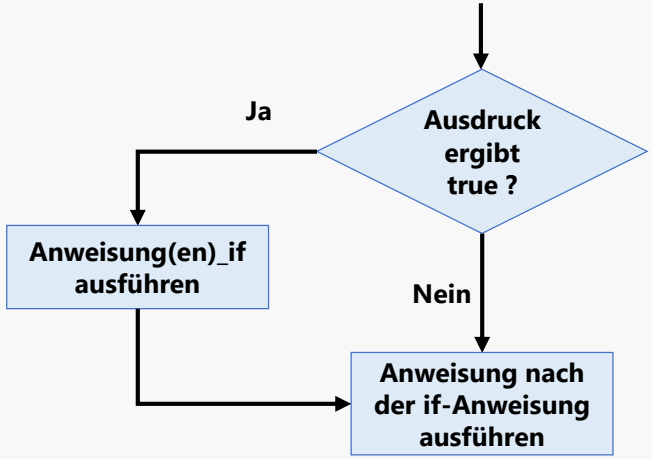
if/else-Anweisung

if-Anweisung

- Allgemeine Form

```
if (Ausdruck)
   Anweisung(en)_if
```

- Anmerkung
  - Ausdruck in Klammern muss einen booleschen Wert erzeugen
- Ablauf



Vergleichsoperatoren

- Vergleiche liefern einen booleschen Wert (true, false)
- Operatoren für Vergleiche (bei einfachen Datentypen)

Java - Notation	Mathematische Notation	Beispiel für true	Beispiel für false
a < b	a < b	2 < 13	2 < 2
a > b	a > b	13 > 2	2 > 13
a <= b	a ≤ b	2 <= 2	3 <= 2
a >= b	a ≥ b	3 >= 2	2 >= 3
a == b	a = b	2 == 2	2 == 3
a != b	a ≠ b	2 != 3	2 != 2

Beispiel (if)

```
public class IfTest1 {

    public static void main(String[] args) {
        int first = (int) (Math.random() * 100);
        int second = (int) (Math.random() * 100);
        if (first > second) {
            System.out.println("first > second");
        }
        System.out.println("first = " + first);
        System.out.println("second = " + second);
    }

}
```

```
first = 39
second = 75
```

```
first > second
first = 64
second = 49
```

Strg + Alt + L bei Problemen

Die Methode random der Klasse Math liefert einen Wert (Zufallszahl) größer oder gleich 0 aber kleiner als 1.0 (vom Typ double) zurück

Hinweis – Klasse Math

- Hilfsklasse für mathematische Operationen
  - `abs`, `min`, `max`, `sin`, `cos`, `log`, `pow`, `sqrt` …
  - Enthält auch Konstanten `E` (Eulersche Zahl) und `PI`
- API
  - https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Math.html

if – Hinweise

- Bei mehreren Anweisungen Block verwenden
  - Beispiel für Unterschied im Ergebnis

```
int x = 5;
int y = 3;
if (x < 10) {
    System.out.println(x + y);
    System.out.println(x - y);
}
if (x < 10)
    System.out.println(x + y);
    System.out.println(x - y);
```

```
8
2
8
2
```

Absichtlich falsche Formatierung

```
int x = 15;
int y = 3;
if (x < 10) {
    System.out.println(x + y);
    System.out.println(x - y);
}
if (x < 10)
    System.out.println(x + y);
    System.out.println(x - y);
```

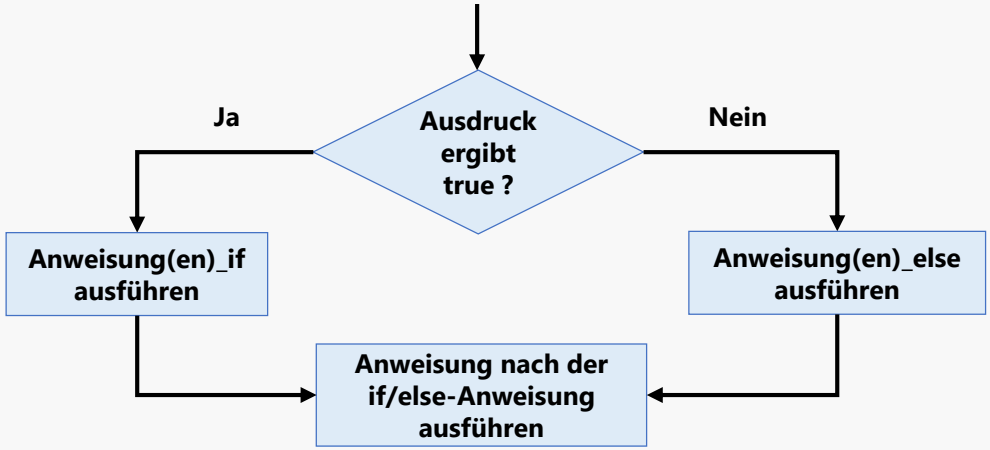
```
12
```

if-else

- Allgemeine Form

```
if (Ausdruck)
   Anweisung(en)_if
else
   Anweisung(en)_else
```

- Ablauf



Beispiel (if-else)

```
public class IfTest2 {

    public static void main(String[] args) {
        int first = (int) (Math.random() * 100);
        int second = (int) (Math.random() * 100);
        if (first > second) {
            System.out.println("first = " + first);
            System.out.println("second = " + second);
            System.out.println("first > second");
        } else {
            System.out.println("first = " + first);
            System.out.println("second = " + second);
            System.out.println("first <= second");
        }
    }

}
```

```
first = 59
second = 53
first > second
```

```
first = 49
second = 71
first <= second
```

Beispiel (Maximum dreier Zahlen)

- 2 verschiedene Versionen
  - Eine Version mit mehreren verschachtelten if-Anweisungen (`MaximumOfThree.java`)
  - Eine Version mit weniger Vergleichen aber mehr Zuweisungen (`MaximumOfThree2.java`)
- Hinweis
  - Längere Programme werden ab jetzt in IntelliJ gezeigt
  - Alle Programme werden in TUWEL angeboten
    - Einzelne zip-Dateien zu den jeweiligen Kapiteln

Vergleich der zwei Versionen (1)

```
…
if (firstNumber > secondNumber) {
    if (firstNumber > thirdNumber) {
        maximum = firstNumber;
    } else {
        maximum = thirdNumber;
    }
} else {
    if (secondNumber > thirdNumber) {
        maximum = secondNumber;
    } else {
        maximum = thirdNumber;
    }
}
…
```

Beispiel: Diese Zuweisung wird nur erreicht, falls firstNumber > secondNumber **und** firstnumber > thirdnumber gilt

```
…
maximum = firstNumber;
if (secondNumber > maximum) {
    maximum = secondNumber;
}
if (thirdNumber > maximum) {
    maximum = thirdNumber;
}
…
```

Diese if-Anweisungen sind unabhängig voneinander. Es kann eine beliebige Kombination davon ausgeführt werden.

Vergleich der zwei Versionen (2)

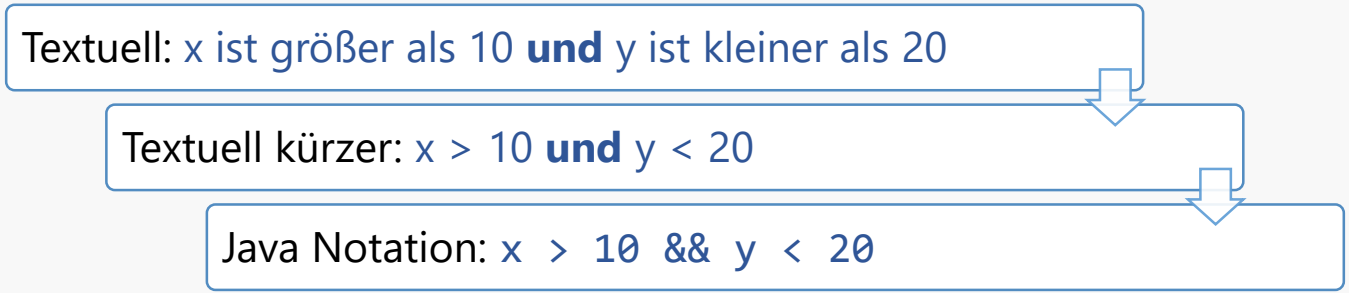
```
…
if (firstNumber > secondNumber) {
    if (firstNumber > thirdNumber) {
        maximum = firstNumber;
    } else {
        maximum = thirdNumber;
    }
} else {
    if (secondNumber > thirdNumber) {
        maximum = secondNumber;
    } else {
        maximum = thirdNumber;
    }
}
…
```

```
…
maximum = firstNumber;
if (secondNumber > maximum) {
    maximum = secondNumber;
}
if (thirdNumber > maximum) {
    maximum = thirdNumber;
}
…
```

- Vergleiche
  - Die 2. Version (rechts) ist textuell kürzer und leichter lesbar
  - 1. Version braucht immer 2 Vergleiche und 1 Zuweisung
  - 2. Version braucht immer 2 Vergleiche und im Durchschnitt 2 Zuweisungen

Komplexere Ausdrücke

- Mehrere Vergleiche in einem Ausdruck verknüpfen
- Wird mit Hilfe von logische Operatoren realisiert
- Beispiel



  - Linker bzw. rechter Vergleich liefern jeweils true oder false (logische Werte)

Logische Operatoren

- Beispiele für logische Operatoren
  - Negation: !
  - Oder: || (|)
  - Und: && (&)
  - XOR: ^

a	!a
true	false
false	true

a	b	a && b	a || b	a ^ b
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

Beispiele für logische Operatoren

```
int counter = 1;
while (counter <= 15) {
    if ((counter % 2 == 0) && (counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
6
12
```

```
int counter = 1;
while (counter <= 15) {
    if (!(counter % 2 == 0) && !(counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
1
5
7
11
13
```

```
int counter = 1;
while (counter <= 15) {
    if ((counter % 2 == 0) || (counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
2
3
4
6
8
9
10
12
14
15
```

```
int counter = 1;
while (counter <= 15) {
    if (!(counter % 2 == 0) || !(counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
1
2
3
4
5
7
8
9
10
11
13
14
15
```

```
int counter = 1;
while (counter <= 15) {
    if ((counter % 2 == 0) ^ (counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
2
3
4
8
9
10
14
15
```

```
int counter = 1;
while (counter <= 15) {
    if (!(counter % 2 == 0) ^ !(counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
2
3
4
8
9
10
14
15
```

Teilweise Auswertung („*Short circuit evaluation*“)

- `x && y` ist immer `false`, falls `x == false`
  - Sobald x auf false auswertet, wird die gesamte Auswertung beendet
- `x || y` ergibt immer `true`, falls `x == true`
  - Sobald x auf true auswertet, wird die gesamte Auswertung beendet

Teilweise auswerten

- Erhöhte Robustheit
- y darf Berechnungen enthalten, die ungültig sind im Fall
  - (`x && y`) bei `x == false`
  - (`x || y`) bei `x == true`
- Beispiel
  - `(x != 0 && y/x > 2)` – keine Division durch 0 möglich
- Vollständige Auswertung mit & und |

Dangling else (1)

- Ein `else`-Zweig bei zwei `if`-Anweisungen
  - Wohin gehört der `else`-Zweig?
- Beispiel (irreführend formatiert)

```
if (counter < 5)
  if (counter % 2 == 0)
    System.out.println("HERE1");
else
  System.out.println("HERE2");
```

Dangling else (2)

- Beispiel (irreführend formatiert)

```
if (counter < 5)
  if (counter % 2 == 0)
    System.out.println("HERE1");
else
  System.out.println("HERE2");
```

- Auflösung durch Compiler
  - `else` gehört zum textuell letzten freien `if` im selben Block
  - `counter` mit Wert `7` führt zu keiner Ausgabe
  - `counter` mit Wert `3` führt zur Ausgabe `"HERE2"`

Bedingungsoperator

- Ternärer Operator und rechtsassoziativ
- Form

```
Bedingung ? Ausdruck1 : Ausdruck2;
```

- Beispiel

```
int max, a, b;
…
max = (a > b) ? a : b;
```

```
if (a > b) {
    max = a;
} else {
    max = b;
}
```

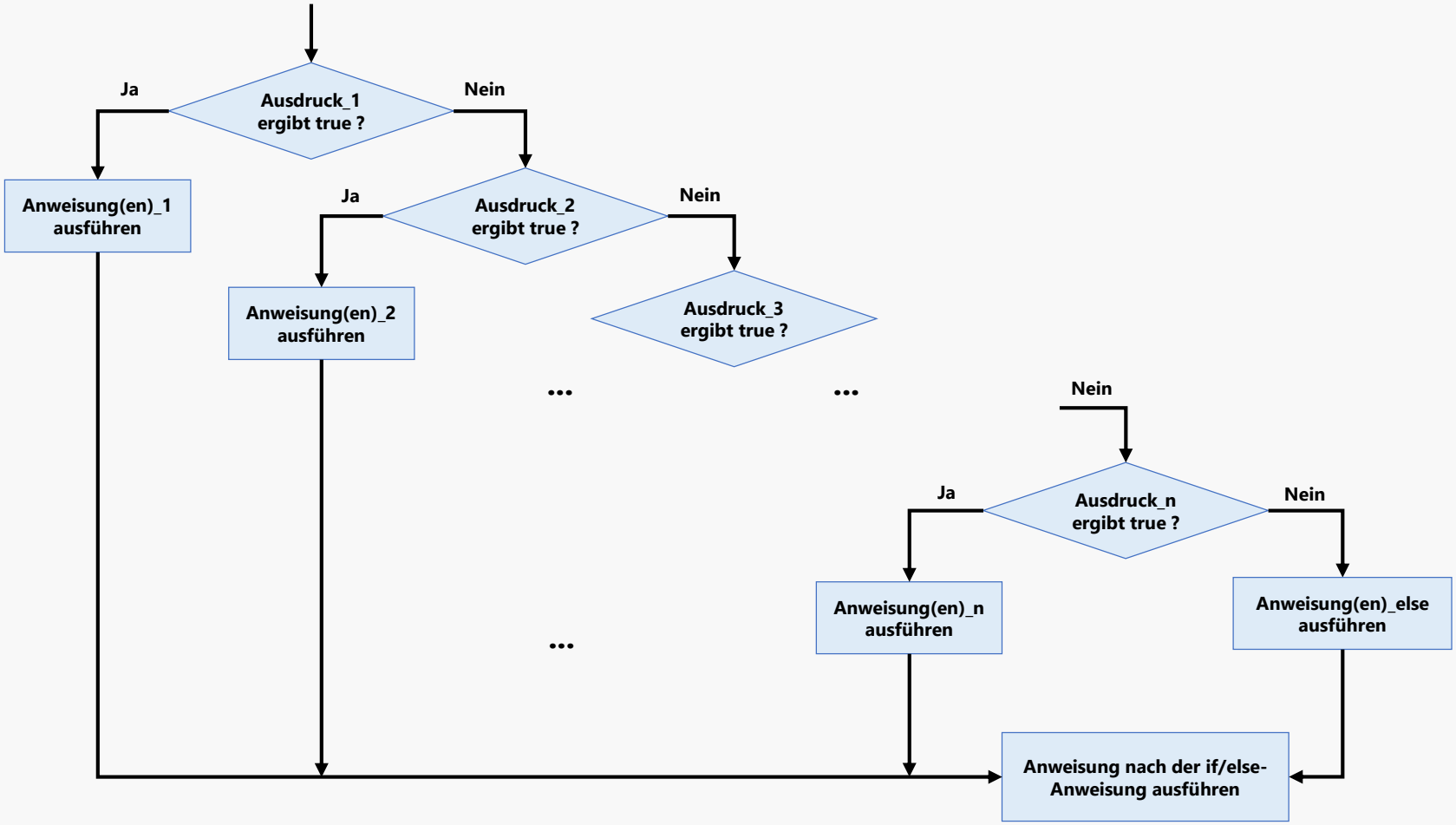
Mehrfachverzweigung bei if

- Allgemeine Form

```
if (Ausdruck_1)
   Anweisung(en)_1
else if (Ausdruck_2)
   Anweisung(en)_2
…
else if (Ausdruck_n)
   Anweisung(en)_n
else
   Anweisung(en)_else
```

- Ein Vergleich wird nach dem anderen durchgeführt (Seiteneffekte beachten!)
- Bei der **ersten Bedingung**, die wahr ist, wird die Anweisung (Anweisungsfolge) abgearbeitet und dann die Anweisung nach der Verzweigung ausgeführt

Mehrfachverzweigung – Ablauf



Beispiel (Mehrfachverzweigung)

```java
public class SequentialChoice {

    public static void main(String[] args) {
        int x = 3;   // Test data
        int result;

        if (x < 0) {
            result = -1;
        } else if (x > 0) {
            result = 1;
        } else {
            result = 0;
        }

        System.out.println(result);
    }

}
```

1

Weiteres Beispiel für Mehrfachverzweigung

```java
public class GradingExample1 {

    public static void main(String[] args) {
        int grade = 2;  // Test data

        String text;
        if (grade == 1) {
            text = " excellent ";
        } else if (grade == 2) {
            text = " good ";
        } else if (grade == 3) {
            text = " satisfactory ";
        } else if (grade == 4) {
            text = " sufficient ";
        } else if (grade == 5) {
            text = " insufficient ";
        } else {
            text = "no grade";
        }

        System.out.println("Your grade:" + text);
    }

}
```

```java
public class GradingExample1 {

    public static void main(String[] args) {
        int grade = 2;  // Test data

        String text = " no grade ";
        if (grade == 1) {
            text = " excellent ";
        } else if (grade == 2) {
            text = " good ";
        } else if (grade == 3) {
            text = " satisfactory ";
        } else if (grade == 4) {
            text = " sufficient ";
        } else if (grade == 5) {
            text = " insufficient ";
        }

        System.out.println("Your grade:" + text);
    }

}
```

```
Your grade: good
```

Zweite Version rechts ohne else-Zweig
text muss dabei korrekt initialisiert werden

Weiteres Beispiel – schlechte Alternative

- Mit verschachtelten Bedingungsoperatoren
- Zeigt was möglich ist
  - Repräsentiert schlechten Stil!

```
public class GradingExample2 {

    public static void main(String[] args) {
        int grade = 2;  // Test data

        String text;
        text = grade == 1 ? " excellent " :
                grade == 2 ? " good " :
                        grade == 3 ? " satisfactory " :
                                grade == 4 ? " sufficient " :
                                        grade == 5 ? " insufficient " :
                                                " no grade ";

        System.out.println("Your grade:" + text);
    }

}
```

```
Your grade: good
```

switch-Anweisung

switch

- Allgemeine Form

```
switch (Ausdruck) {
   case Wert_1:
          Anweisung(en)_1
          break;
   case Wert_2:
          Anweisung(en)_2
          break;
   …
   case Wert_n:
          Anweisung(en)_n
          break;
   default:
          Anweisung(en)_default
          break;          /* Nicht notwendig, aber hilfreich */
}
```

switch – allgemeine Beschreibung

- Ausdruck wird ausgewertet
  - Erlaubte Datentypen (von den bisher besprochenen): `short`, `byte`, `int`, `char` oder `String`
- Auswahl einer Alternative
  - Wert des Ausdrucks entspricht dem **konstanten Wert** hinter einer `case`-Marke

```
switch (Ausdruck) {
    case Wert_1:
        Anweisung(en)_1
        break;
    case Wert_2:
        Anweisung(en)_2
        break;
    …
    case Wert_n:
        Anweisung(en)_n
        break;
    default:
        Anweisung(en)_default
        break;
}
```

switch – break

- Sorgt dafür, dass **nicht** zur folgenden `case`-Marke weitergesprungen wird
  - `switch`-Anweisung wird verlassen
- Ohne `break` können zwei oder mehrere `case`-Marken verbunden werden
  - Alle nachfolgenden Anweisungen werden bis zum nächsten `break` oder dem Ende der `switch`-Anweisung abgearbeitet
  - „Fall-through

switch – default

- `default`-Marke
  - ist optional
  - wird ausgeführt, wenn keine `case`-Marke passt
  - ohne `default` ist es möglich, dass keine Anweisungen ausgeführt werden
- Hinweis
  - Jeder Wert darf nur einmal bei einer `case`-Marke vorkommen

Beispiele (switch)

- Beispiel mit switch statt Mehrfachverzweigung
  - Variante von GradingExample1.java

```
Your grade: good
```

```java
public class GradingExample3 {

    public static void main(String[] args) {
        int grade = 2;  // Test data

        String text;
        switch (grade) {
            case 1:
                text = " excellent ";
                break;
            case 2:
                text = " good ";
                break;
            case 3:
                text = " satisfactory ";
                break;
            case 4:
                text = " sufficient ";
                break;
            case 5:
                text = " insufficient ";
                break;
            default:
                text = " no grade ";
        }

        System.out.println("Your grade: " + text);
    }

}
```

Beispiele (switch)

- Alternatives Beispiel mit switch
  - Benutzt „Fall-through“

```
Your grade: passed
```

```java
public class GradingExample4 {

    public static void main(String[] args) {
        int grade = 2;  // Test data
        String text;

        switch (grade) {
            case 1:
                text = " with distinction ";
                break;
            case 2:
            case 3:
            case 4:
                text = " passed ";
                break;
            case 5:
                text = " not passed ";
                break;
            default:
                text = " no grade ";
        }

        System.out.println("Your grade:" + text);
    }

}
```

switch-Ausdruck (Java 14)

- switch kann einen Wert erzeugen
- Beispiel

Hinweis: Beispiel verwendet die Enumeration (ähnlich zu Klasse) Day. Darin sind die Wochentage als Kostanten enthalten.

```
public class SwitchTest {
    public enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY; }
    public static void main(String[] args) {
        int numLetters = 0;
        Day day = Day.WEDNESDAY;
        switch (day) {
            case MONDAY:
            case FRIDAY:
            case SUNDAY:
                numLetters = 6;
                break;
            case TUESDAY:
                numLetters = 7;
                break;
            case THURSDAY:
            case SATURDAY:
                numLetters = 8;
                break;
            case WEDNESDAY:
                numLetters = 9;
                break;
            default:
                numLetters = -1;
        }
        System.out.println(numLetters);
    }
}
```

Seit Java 14

```
public class SwitchTest {

    public enum Day {
        SUNDAY, MONDAY, TUESDAY,
        WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;
    }

    public static void main(String[] args) {
        Day day = Day.WEDNESDAY;
        int numLetters = switch (day) {
            case MONDAY, FRIDAY, SUNDAY -> 6;
            case TUESDAY -> 7;
            case THURSDAY, SATURDAY -> 8;
            case WEDNESDAY -> 9;
            default -> -1;
        };
        System.out.println(numLetters);
    }

}
```

switch-Ausdruck

- Syntaktische Änderung („->“ statt „:“)
- Mehrere Werte hinter case-Klausel
- Kein break mehr
  - Es existiert kein Fall-through!
- switch kann einen Wert zurückgeben
  - Hilfsvariablen können vermieden werden
- Compiler überprüft auf Vollständigkeit
  - Alle möglichen Werte müssen abgedeckt werden (direkt oder über default)

Schlüsselwort yield bei switch

- Zwei Anwendungsfälle

Alternative Schreibweise (ähnlich zu alter Syntax)

```
…
Day day = Day.WEDNESDAY;
int numLetters = switch (day) {
    case MONDAY, FRIDAY, SUNDAY: yield 6;
    case TUESDAY: yield 7;
    case THURSDAY, SATURDAY: yield 8;
    case WEDNESDAY: yield 9;
    default: yield -1;
};
…
```

Rückgabe eines Wertes bei mehreren Anweisungen in einem Block

```
…
Day day = Day.WEDNESDAY;
int numLetters = switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> {
        if (day == Day.SUNDAY) {
            System.out.println("Sunday!");
        }
        yield 6;
    }
    case TUESDAY -> 7;
    case THURSDAY, SATURDAY -> 8;
    case WEDNESDAY -> 9;
    default -> -1;
};
…
```

Programmierstil

Allgemeine Hinweise

- Unterkapitel zu Programmierstil
  - gibt einfache Richtlinien vor
  - zeigt Beispiele für guten Programmierstil
  - soll Diskussionen anregen
- Achtung
  - Die vorgestellten Regeln und Vorgehensweisen müssen nicht immer stur befolgt werden
  - In vielen Fällen werden sie sich aber als hilfreich erweisen

Grundregel für Lesbarkeit

Der Code sollte so geschrieben werden, dass die Zeit, die eine andere Person zum Verständnis benötigt, möglichst gering ist.

Einrückungen

- Erhöhen die Lesbarkeit
  - Programmstruktur sichtbar machen
- Typische Einrückungstiefe
  - 4 Leerzeichen pro Ebene
  - Siehe bisherige Beispiele

Klammerstil

- Zwei bekannte Stile

OTBS („One True Brace Style“)-Stil (Standard in IntelliJ)

```
if (firstNumber > secondNumber) {
    if (firstNumber > thirdNumber) {
        maximum = firstNumber;
    } else {
        maximum = thirdNumber;
    }
} else {
    …
}
```

+ Mehr Code pro Zeile
+ Schließende Klammer steht unter dem Konstrukt, zu dem sie gehört

Allman-Stil (Standard in Visual Studio)

```
if (firstNumber > secondNumber)
{
    if (firstNumber > thirdNumber)
    {
        maximum = firstNumber;
    }
    else
    {
        maximum = thirdNumber;
    }
}
else
{
    …
}
```

+ Blockbereiche gut erkennbar
- Mehr Zeilen pro Datei

OTBS-Stil

- Vollständige Klammerung
  - Auch bei einzeiligen Blöcken

```
if (a < 10)
    a = 10;
```

```
if (a < 10) {
    a = 10;
}
```

  - Rechte Variante braucht mehr Platz aber erleichtert spätere Erweiterung

Reihenfolge der Operanden bei Vergleichen

- Beispiele für unterschiedliche Formulierungen

`if (length >= 10) ...`	`if (10 < length) ...`
`if (bytesReceived < bytesExpected) ...`	`if (bytesExpected > bytesReceived) ...`

- Einfache Richtlinie

Linke Seite	Rechte Seite
Ausdruck, der abgefragt wird und dessen Wert sich ändert	Ausdruck, gegen dessen Wert verglichen wird und der konstant ist (bzw. weniger oft ändert)

Verkürzungen

- Einfache if-Anweisungen werden manchmal einzeilig geschrieben

```
if (n != 0) x = x/n;
```

  - Nur sehr sparsam verwenden bzw. überhaupt vermeiden

Programmiermuster

Anwendung von if und switch

- Abhängig von der Lesbarkeit und Ausdrucksstärke
- `if`
  - Einfache Auswahlsituationen
    - Anweisung(en) ausführen oder nicht bzw. alternative Anweisungen ausführen
  - Komplexere Ausdrücke in Bedingungen
- `switch`
  - Anhand mehrerer konstanter Werte verzweigen

Bedingungsoperator

- Verkürzung von Programmcode

Beispiel für Verzweigung	Version mit Bedingungsoperator

`int max, a, b;`
`…`
`if (a > b) {`
`  max = a;`
`} else {`
`  max = b;`
`}`
	`int a, b;`
	`…`
	`int max = (a > b) ? a : b;`

- Aber
  - Nur sparsam einsetzen
  - Schlechtes Beispiel siehe `GradingExample2`

if/else – Muster

Muster	Konstrukt	Beispiel für Form

Beliebige Kombination von Anweisungen mit Überprüfung ausführen	Sequentielle if-Anweisungen	`if (Ausdruck_1)`
		`  Anweisung(en)_1`
		`if (Ausdruck_2)`
		`  Anweisung(en)_2`
		`if (Ausdruck_3)`
		`  Anweisung(en)_3`

Keine oder eine Anweisung mit Überprüfung ausführen	Verschachtelte if-Anweisungen ohne else am Ende	`if (Ausdruck_1)`
		`  Anweisung(en)_1`
		`else if (Ausdruck_2)`
		`  Anweisung(en)_2`
		`else if (Ausdruck_3)`
		`  Anweisung(en)_3`

Genau eine Anweisung mit Überprüfung ausführen	Verschachtelte if-Anweisungen mit else am Ende	`if (Ausdruck_1)`
		`  Anweisung(en)_1`
		`else if (Ausdruck_2)`
		`  Anweisung(en)_2`
		`else`
		`  Anweisung(en)_3`

Reihenfolge der Blöcke

- Beispiel

```
if (a == b) {
    // Erster Fall
} else {
    // Zweiter Fall
}
```

```
if (a != b) {
    // Zweiter Fall
} else {
    // Erster Fall
}
```

- Einfache Empfehlungen
  - Positiver Fall zuerst
  - Einfacher Fall zuerst
  - Interessanter Fall zuerst
- Manchmal stehen diese Empfehlungen in Konflikt!

Zusammenfassung

- Motivation
- if/else-Anweisung
- switch-Anweisung
- Programmierstil bei Verzweigungen
- Programmiermuster bei Verzweigungen