Programmiervorkurs 2. Tag

Strukturierung von Programmen durch Verzweigungen

FACHSCHAFT INFORMATIK

HS Karlsruhe

Themenüberblick

- Kommentare
- Formulierung von Bedingungen: Boolesche Ausdrücke
- Verzweigungen
  - If-Abfragen
  - Switch/Case

Kommentare

Kommentare

- Erleichtern das Verständnis des Quelltextes
  - Von Menschen für Menschen: Werden vom Compiler entfernt und haben keinen Einfluss auf den Programmablauf
- Programmdokumentation durch Javadoc bzw. entsprechende XML-Dokumentation in C# (Details in der Informatik 1-Vorlesung)
- Implementierungskommentare für Entwickler im Quelltext

Die verschiedenen Arten von Kommentaren

- Javadoc:

```
/**
 * Kommentar (auch über mehrere Zeilen), der
 * automatisch zu html-Dokumentation
 * verarbeitet werden kann
 */
```

- XML-Dokumentation:

```
/// <summary>
/// Äquivalent zu Javadoc für C#
/// </summary>
```

Die verschiedenen Arten von Kommentaren

- Blockkommentar:

```
/*
 * Mehrzeilige Kommentare sind ideal, wenn
 * viele Informationen unterzubringen sind.
 * Es gilt die Devise: so knapp wie
 * möglich, so ausführlich wie nötig.
 */
```

- Zeilenkommentar: `// endet mit Zeilenumbruch`

Verwendung von Kommentaren

- Nachfolgenden Entwicklern Hinweise geben, wie der Quelltext zu verstehen ist
- Sehr praktisch als Gedächtnisstütze: TODOs setzen
- Zum Testen können Teile des Quellcodes zeitweise auskommentiert werden (ersetzt nicht anständiges Debugging!)

Beispiel Java

```java
/**
 * Fuer uns eigentlich uninteressant: Javadoc
 * @author Anna Weisshaar
 */
public class Kommentare {

    /*
     * Die wichtigste Methode fuer uns: hier kommt alles rein, was wir machen
     */
    public static void main(String[] args){

        // Kommentarstatus dieses Programms
        boolean istKommentiert = true;

        // TODO: Warum kommen hier so komische Werte raus?
        /* int testZahl = Integer.MAX_VALUE + 1; */
    }

}
```

Beispiel C#

```
/// <summary>
/// Fuer uns eigentlich uninteressant: XML-Dokumentation
/// </summary>
public class Kommentare {

    /*
     * Die wichtigste Methode fuer uns: hier kommt alles rein, was wir machen
     */
    public static void Main(string[] args){

        // Kommentarstatus dieses Programms
        bool istKommentiert = true;

        // TODO: Warum kommen hier so komische Werte raus?
        /* int testZahl =  int.MaxValue + 1; */
    }

}
```

Formulierung von Bedingungen

Bedingungen

- Werden mit Hilfe von logischen Operatoren ausgedrückt
  - Mehrere logische Operatoren können kombiniert werden
- Ergebnis der Auswertung ist ein boolescher Wert (true oder false)
  - Kann in einer Variable vom Typ bool[ean] gespeichert werden
- Werden zur Entscheidungsfindung verwendet
  - Verzweigungen
  - Abbruchbedingungen
  - Ausführungsbedingungen

Logische Operationen

Vergleiche	Verknüpfungen
größer: >	und: &
kleiner: <	oder: |
größer gleich: >=	nicht: !
kleiner gleich: <=	exklusives Oder: ^
gleich: ==	
ungleich: !=	

Vorsicht beim Vergleich von Gleitkommazahlen!

- `1.0 - (0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1) == 0.0` ?
- `0.02d == 0.02f` ?

  => Um sicherzugehen lieber prüfen, ob die Abweichung nur minimal ist:

  `0.02d - 0.02f < 4.5E-10` ?

Logische Verknüpfungen – Wahrheitstabellen

a	*b*	*&*
0	0	0
0	1	0
1	0	0
1	1	1

a	*b*	*|*
0	0	0
0	1	1
1	0	1
1	1	1

a	*b*	*^*
0	0	0
0	1	1
1	0	1
1	1	0

a	*!*
0	1
1	0

Kurzschlussoperatoren

- UND und ODER gibt es auch als sogenannte Kurzschlussoperatoren: && und ||

  => Der Ausdruck wird nur solange ausgewertet, bis das Ergebnis feststeht

Verwendung von Kurzschlussoperatoren – Seiteneffekte

```java
double a = 0.0;
double b = 0.0;

// Kurzschlussoperator
if ((a != 0) && (b++ / a > 1)) {

    System.out.println(b/a);

}

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

```java
double a = 0.0;
double b = 0.0;

// Ohne Kurzschlussoperator
if ((a != 0) & (b++ / a > 1)) {

    Console.WriteLine(b/a);

}

Console.WriteLine(b);
```

Rangfolge der Operatoren (Auswahl)

++ / --	Inkrement und Dekrement
!	Negation
*, /, %	Multiplikation, Division, Modulo
+, -	Addition und Subtraktion
==, !=	(Un)Gleichheit von Werten
&	Und
^	Xor
|	Oder
&&	Kurzschlussoperator Und
||	Kurzschlussoperator Oder
=	Zuweisung

Beispiel: ein etwas längerer boolescher Ausdruck

Sportwagen: maximal zwei Türen, keine Rücksitze, außerdem:

- Höchstgeschwindigkeit von mindestens 200 km/h und Mindestbeschleunigung von 0 auf 100 km/h in 8 Sekunden
- oder Höchstgeschwindigkeit von mindestens 280 km/h und mindestens 250 PS

Auszuwertende Variablen

- bool[ean] hatRuecksitze; // genau dann true, wenn das Auto Rücksitze hat
- int tueren; // Anzahl Türen
- double beschleunigung; // in Sekunden von 0 auf 100
- double hoechstgeschwindigkeit; // in km/h
- double leistung; // in kW (nicht in PS!)
  - 1 PS ~ 0,735 kW

Verzweigungen

Fallunterscheidung durch If-Abfragen

- Anweisung wird nur dann ausgeführt, wenn eine bestimmte Bedingung erfüllt ist:

```
if (Bedingung) {
   // mach was
} else if (andere Bedingung){
   // mach was anderes
} else {
   // lass es bleiben
}
```

(else if und else optional)

Beispiel 1 – Ergebnis eines Vergleichs als Bedingung

```
double a, b;


...


if (a != 0.0) {
    System.out.print("b/a: ");
    System.out.println(b/a);
} else {
    System.out.println("Division durch 0!");
}
```

Beispiel 2 – Ergebnis einer logischen Verknüpfung als Bedingung

```
bool[ean] istStudent = true;

if (!istStudent) {
    // gewaehre keinen Studentenrabatt
}
```

entspricht der Abfrage folgender Bedingungen:

```
if (istStudent == false) { … }

if (!(istStudent == true)) { … }
```

Verschachtelte If-Abfrage

```
long losnummer;
...

if (losnummer > 3) {
    if (losnummer % 315 == 4) {
        Console.WriteLine("Gewinnerlos");
    }
} else {
    Console.WriteLine("Verliererlos");
}
```

=> Vorsicht! Wird leicht unübersichtlich!

Dangling Else

Oder: Warum Klammern die Lesbarkeit und Verständlichkeit erhöhen und Schachtelungen zu vermeiden sind

```
int a = 2;
int b = 3;
if (a == 1)
    if (b == 1)
        a = 42;
else
    b = 42;
```

Fallunterscheidung durch Switch / Case

- Fallunterscheidung in Abhängigkeit von einer Variablen (ganzzahlige Typen oder char)
- Anweisungen für alle relevanten Werte, die die Variable annehmen kann
  - jeder Wert darf dabei nur einmal vorkommen
  - nur Wert der Variablen abfragen, keine sonstigen Bedingungen

Switch / Case – Syntax Java

```
int fall;
…
switch (fall) {
case 0:
    System.out.println("Operation erfolgreich");
    break;
case 1:
    System.out.println("Abgebrochen");
    // fall through
case 2:
    System.out.println("Operation gescheitert");
    break;
default:
    System.out.println("Unbekannter Fall!");
}
```

Switch / Case – Syntax C#

```
int fall;
…
switch (fall) {
case 0:
    Console.WriteLine("Operation erfolgreich");
    break;
case 1:  // fall through (nur wenn case-Block leer!)
case 2:
    Console.WriteLine("Operation gescheitert");
    break;
default:
    Console.WriteLine("Unbekannter Fall!");
    break;
}
```

Switch / Case – Wie funktioniert's?

- Switch-Ausdruck wird ausgewertet, Wert wird mit den aufgezählten Werten verglichen
- Kommt der Wert in der Aufzählung vor, beginnt die Ausführung bei der entsprechenden Anweisung
  - Java: Läuft durch bis break oder bis zum Ende des Switch
  - C#: Jeder case-Block (auch default) **muss** mit einem break oder einer anderen Sprunganweisung abgeschlossen werden (Ausnahme: leere Anweisungen)
- Kommt der Wert in der Aufzählung nicht vor, wird, so vorhanden, die default-Anweisung ausgeführt

Switch / Case und entsprechende If-Abfrage

```
char auswahl;
...
switch (auswahl) {
case 'r':
   // lese Eingabe
break;
case 'q':
   // beende Programm
break;
case 'n':
   // Neustart
break;
}
```

```
char auswahl;
...

if (auswahl == 'r') {
   // lese Eingabe
} else if (auswahl == 'q') {
   // beende Programm
} else if (auswahl == 'n') {
   // Neustart
}
```

The Ent