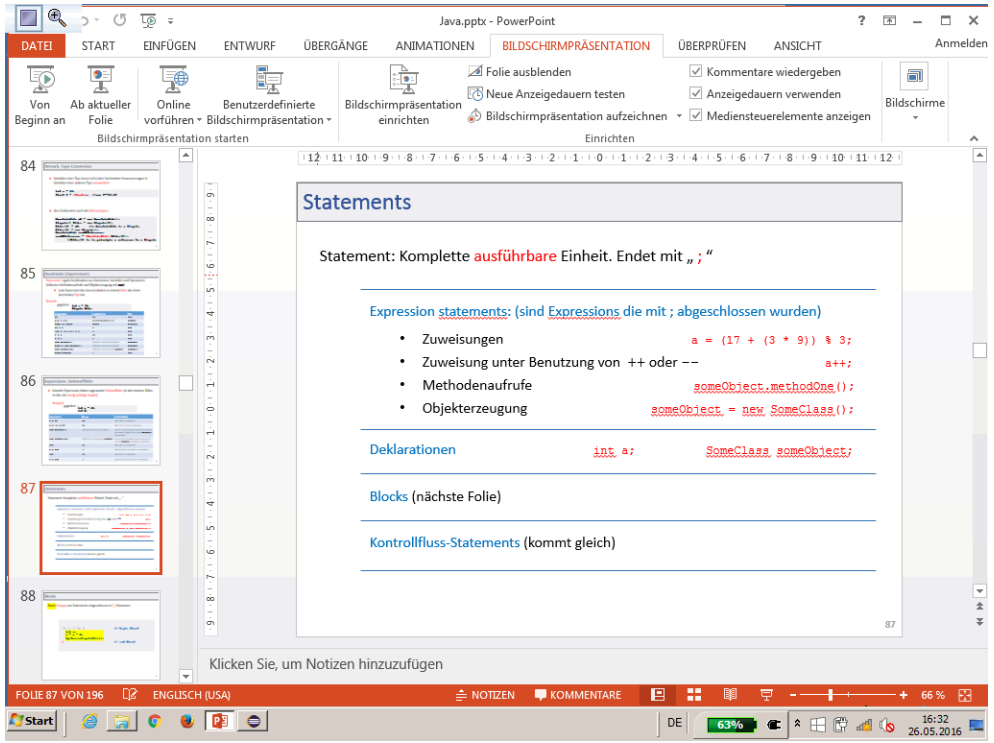
Script **generated by TTT**

Title: groh: profile1 (26.05.2016)

Date: Thu May 26 16:32:57 CEST 2016

Duration: 26:04 min

Pages: 40



Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder -- `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder -- `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

87

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

87

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

87

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder −− `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

87

Statements

Statement: Komplette ausführbare Einheit. Endet mit „ ; “

Expression statements: (sind Expressions die mit ; abgeschlossen wurden)

- Zuweisungen `a = (17 + (3 * 9)) % 3;`
- Zuweisung unter Benutzung von ++ oder -- `a++;`
- Methodenaufrufe `someObject.methodOne();`
- Objekterzeugung `someObject = new SomeClass();`

Deklarationen `int a;` `SomeClass someObject;`

Blocks (nächste Folie)

Kontrollfluss-Statements (kommt gleich)

Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

```
if (a != b) {              // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                          // end block
```


Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

```
if (a != b) {              // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                          // end block
```

88

Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

```
if (a != b) {              // begin block
  int c;
  c = a * b;
  System.out.println(c);
}                          // end block
```

88

Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

- In einem Block deklarierte Variablen sind nur innerhalb des Blocks sichtbar:

```
int a = 7, b = 6;

if (a != b) {                 // begin block
   int c;
   c = a * b;
   System.out.println(c);
}                             // end block

System.out.println(c);        // ERROR: c unavailable
```


Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

- In einem Block deklarierte Variablen sind nur innerhalb des Blocks sichtbar:

```
int a = 7, b = 6;

if (a != b) {                 // begin block
   int c;
   c = a * b;
   System.out.println(c);
}                             // end block

System.out.println(c);        // ERROR: c unavailable
```

89

Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

- In einem Block deklarierte Variablen sind nur innerhalb des Blocks sichtbar:

```
int a = 7, b = 6;

if (a != b) {                 // begin block
   int c;
   c = a * b;
   System.out.println(c);
}                             // end block

System.out.println(c);        // ERROR: c unavailable
```

89

Blocks

Block: Gruppe von Statements eingeschlossen in { } Klammern

- In einem Block deklarierte Variablen sind nur innerhalb des Blocks sichtbar:

```
int a = 7, b = 6;

if (a != b) {                 // begin block
   int c;
   c = a * b;
   System.out.println(c);
}                             // end block

System.out.println(c);        // ERROR: c unavailable
```

89

Kontrollfluss-Statements

Kontrollfluss-Statements erlauben es, von der sequentiellen Abfolge der Abarbeitung der Statements abzuweichen.

- Bedingte Verzweigungen (conditionals): `if`, `if else`, `switch`
- Schleifen (loops): `while`, `do while`, `for`
- Verzweigungen (branches): `break`, `continue`, `return`

Bedingte Verzweigungen: `if` `if else` `switch`

```
if (speed > 10) {
        speed = speed - 2;
} else {
        if (speed > 0) {
                speed--;
        } else {
                System.err.println("The bicycle has already stopped!");
        }
}
```


Bedingte Verzweigungen: `if` `if else` `switch`


```
if (speed > 10) {
        speed = speed - 2;
} else {
        if (speed > 0) {
                speed--;
        } else {
                System.err.println("The bicycle has already stopped!");
        }
}
```


Bedingte Verzweigungen: `if` `if else` `switch`


```
if (speed > 10) {
        speed = speed - 2;
} else {
        if (speed > 0) {
                speed--;
        } else {
                System.err.println("The bicycle has already stopped!");
        }
}
```

Bedingte Verzweigungen: `if` `if else` `switch`

```
if (speed > 10) {
        speed = speed - 2;
} else if (speed > 0) {
        speed--;
} else {
        System.err.println("The bicycle has already stopped!");
}
```


Schleifen: `while` `do while`

`while`: Mache etwas, *solange* eine Bedingung gilt (zu `true` evaluiert)

```
int count = 1;
while (count < 8) {
        System.out.print("#:" + count + " ");
        count++;
}
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

Schleifen: `while` `do while`

`while`: Mache etwas, *solange* eine Bedingung gilt (zu `true` evaluiert)

```
int count = 1;
while (count < 8) {
        System.out.print("#:" + count + " ");
        count++;
}
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

Schleifen: `while` `do while`

`while`: Mache etwas, *solange* eine Bedingung gilt (zu `true` evaluiert)

```
int count = 1;
while (count < 8) {
        System.out.print("#:" + count + " ");
        count++;
}
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

Schleifen: while do while

`while`: Mache etwas, *solange* eine Bedingung gilt (zu `true` evaluiert)

```
int count = 1;
while (count < 8) {
       System.out.print("#:" + count + " ");
       count++;
}
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

`do while`: Ähnlich zu `while`, aber überprüfe Bedingung NACH der Ausführung von etwas (anstelle VOR der Ausführung)

```
int count = 1;
do {
       System.out.print("#:" + count + " ");
       count++;
} while (count < 8)
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

Schleifen: for

`for`: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
       System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

Schleifen: while do while

`while`: Mache etwas, *solange* eine Bedingung gilt (zu `true` evaluiert)

```
int count = 1;
while (count < 8) {
       System.out.print("#:" + count + " ");
       count++;
}
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

`do while`: Ähnlich zu `while`, aber überprüfe Bedingung NACH der Ausführung von etwas (anstelle VOR der Ausführung)

```
int count = 1;
do {
       System.out.print("#:" + count + " ");
       count++;
} while (count < 8)
```

⇨ Ausgabe: `#:1 #:2 #:3 #:4 #:5 #:6 #:7`

98

Schleifen: for

`for`: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
       System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

99

Schleifen: for

`for`: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

Schleifen: for

`for`: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

Schleifen: for

`for`: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

Schleifen: for

`for`: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
      System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: `#:0 #:1 #:2 #:3 #:4 #:5 #:6`

Allgemeine Form:

```
for (initialization; termination; update) {
      statements
}
```

- *initialization* Expression: Wird einmal vor Beginn der Schleife ausgeführt
- *termination* Expression (Bedingung): Wenn `true` dann führe Statements aus, ansonsten verlasse Schleife
- update Expression: Wird nach jeder Iteration der Schleife ausgeführt.

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
        System.out.print("#:" + i + " ");
}
```

⇒ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

Allgemeine Form:

```
for (initialization; termination; update) {
        statements
}
```

- *initialization* Expression: Wird einmal vor Beginn der Schleife ausgeführt
- *termination* Expression (Bedingung): Wenn `true` dann führe Statements aus, ansonsten verlasse Schleife
- update Expression: Wird nach jeder Iteration der Schleife ausgeführt.

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
        System.out.print("#:" + i + " ");
}
```

⇒ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

allgemeines `for` ist äquivalent zu `while`:

```
initialization;
while (termination) {
        statements
        update;
}
```

Allgemeine Form:

```
for (initialization; termination; update) {
        statements
}
```

- *initialization* Expression: Wird einmal vor Beginn der Schleife ausgeführt
- *termination* Expression (Bedingung): Wenn `true` dann führe Statements aus, ansonsten verlasse Schleife
- update Expression: Wird nach jeder Iteration der Schleife ausgeführt.

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
        System.out.print("#:" + i + " ");
}
```

⇒ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

allgemeines `for` ist äquivalent zu `while`:

```
initialization;
while (termination) {
        statements
        update;
}
```

Allgemeine Form:

```
for (initialization; termination; update) {
        statements
}
```

- *initialization* Expression: Wird einmal vor Beginn der Schleife ausgeführt
- *termination* Expression (Bedingung): Wenn `true` dann führe Statements aus, ansonsten verlasse Schleife
- update Expression: Wird nach jeder Iteration der Schleife ausgeführt.

Schleifen: for

for: (Üblicherweise:) Mache etwas eine festgelegte Anzahl von Malen (Iterationen)

```
for (int i=0; i<7; i++) { // loop will be executed 7 times
        System.out.print("#:" + i + " ");
}
```

⇒ Ausgabe: #:0 #:1 #:2 #:3 #:4 #:5 #:6

allgemeines `for` ist äquivalent zu `while`:

```
initialization;
while (termination) {
        statements
        update;
}
```

Allgemeine Form:

```
for (initialization; termination; update) {
        statements
}
```

- *initialization* Expression: Wird einmal vor Beginn der Schleife ausgeführt
- *termination* Expression (Bedingung): Wenn `true` dann führe Statements aus, ansonsten verlasse Schleife
- update Expression: Wird nach jeder Iteration der Schleife ausgeführt.

Verzweigungen: return break continue

- **`return`**: Beende die gerade ausgeführte Methode und gehe im Kontrollfluss an die Stelle zurück, wo sie aufgerufen wurde.
(Mehr Details folgen gleich)

- **`break`**: Erzwinge die Beendigung einer Schleife
- **`continue`**: Überspringe die aktuelle Iteration einer Schleife

Man kommt fast immer ohne das aus (und sollte es auch!!)

```
for (int i = 0; i < 10; i++) {
	if (i == 8) {
		break;
	} else if (i % 2 == 0) {
		continue;
	}
	System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: `#:1 #:3 #:5 #:7`

Verzweigungen: return break continue

- **`return`**: Beende die gerade ausgeführte Methode und gehe im Kontrollfluss an die Stelle zurück, wo sie aufgerufen wurde.
(Mehr Details folgen gleich)

- **`break`**: Erzwinge die Beendigung einer Schleife
- **`continue`**: Überspringe die aktuelle Iteration einer Schleife

Man kommt fast immer ohne das aus (und sollte es auch!!)

```
for (int i = 0; i < 10; i++) {
	if (i == 8) {
		break;
	} else if (i % 2 == 0) {
		continue;
	}
	System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: `#:1 #:3 #:5 #:7`

103

Verzweigungen: return break continue

- **`return`**: Beende die gerade ausgeführte Methode und gehe im Kontrollfluss an die Stelle zurück, wo sie aufgerufen wurde.
(Mehr Details folgen gleich)

- **`break`**: Erzwinge die Beendigung einer Schleife
- **`continue`**: Überspringe die aktuelle Iteration einer Schleife

Man kommt fast immer ohne das aus (und sollte es auch!!)

```
for (int i = 0; i < 10; i++) {
	if (i == 8) {
		break;
	} else if (i % 2 == 0) {
		continue;
	}
	System.out.print("#:" + i + " ");
}
```

⇨ Ausgabe: `#:1 #:3 #:5 #:7`

103

LIVE DEMO:
- minimum
- fakultaet (mit while und for)
- exp
- power mit while und for
- testIfAllZero
- dotProduct
- multMatVect

LIVE DEMO:

- minimum
- fakultaet (mit while und for)
- exp
- power mit while und for
- testIfAllZero
- dotProduct
- multMatVect