Session 5: Loops

Softwaretechnologie: Java I

Nils Reiter
nils.reiter@uni-koeln.de

November 9, 2022

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Exercise 4

Section 2

Recap: Switch-Statement

Switch-Statement

- Complex and embedded if-statements quickly become unreadable
- Alternative, if all if-statements compare against the same variable: `switch`-statement

Switch-Statement

- Complex and embedded if-statements quickly become unreadable
- Alternative, if all if-statements compare against the same variable: `switch`-statement

```
1 switch (EXPRESSION) {
2   case CONSTANT:
3     STATEMENT;
4     break;
5   case CONSTANT2:
6   case CONSTANT3:
7     STATEMENT;
8     break;
9   default:
10    STATEMENT
11 }
```

Switch-Statement

- Complex and embedded if-statements quickly become unreadable
- Alternative, if all if-statements compare against the same variable: `switch`-statement

```
1  switch (EXPRESSION) {
2    case CONSTANT:
3      STATEMENT;
4      break;
5    case CONSTANT2:
6    case CONSTANT3:
7      STATEMENT;
8      break;
9    default:
10     STATEMENT
11 }
```

This variant is available in later Java versions:

```
1  // ...
2  switch (EXPRESSION) {
3    // ...
4    case CONSTANT2, CONSTANT3:
5      STATEMENT;
6      break;
7    // ...
8  }
```

Switch-Statement

Example

```
1 static short daysInMonth(byte month) {
2     switch(month) {
3     case 2: return 28; // no break needed, because of return
4     case 4: // fall through to case 11
5     case 6:
6     case 9:
7     case 11: return 30;
8     default: return 31;
9   }
10 }
```

Break vs. Return

- `switch` in a function: Similar effects of `break` and `return`
- But conceptually very different
- `break`
  - Stops execution of the `switch` statement
  - Control flow continues after `}` of `switch`
  - If not used: Following `case` will also be executed
- `return`
  - Stops execution of a function and returns some value
  - Doesn't matter if embedded in other things

Break vs. Return

Two equivalent cases

```
1  static int daysInMonth(int month) {
2    switch (month) {
3    case 2:
4      return 28;
5    case 4:
6    case 6:
7    case 9:
8    case 11:
9      return 30;
10   default:
11     return 31;
12   }
13 }
```

```
1  static int daysInMonth(int month) {
2    float retValue = 31;
3    switch (month) {
4    case 2:
5      retValue = 28;
6      break;
7    case 4:
8    case 6:
9    case 9:
10   case 11:
11     retValue = 30;
12     break;
13   }
14   return retValue;
15 }
```

Section 3

Loops

Introduction

- Executing code repeatedly
- What do we need?
  - The code to be executed (i.e., a code block)
  - Conditions on how often to repeat

While-Loop

- Repeat as long as some expression is true
- Similar to `if`, but with a repeat option
  - `EXPRESSION` must be of type `boolean`
  - If `EXPRESSION` evaluates to `false`, not executed at all
- `EXPRESSION` is evaluated in every iteration before the code block is run
  - I.e., if variables change during execution, the expression can check their state

```
1 while (EXPRESSION) {
2   // some code
3 }
```

demo

Do-While-Loop

- Repeat as long as some expression is true
- Similar to `while`, but code is executed at least once

```
1 do {
2   // some code
3 } while (EXPRESSION);
```

For-Loop

- In many cases, we know in advance how often do repeat code

```
1 // do something for each of 25 days
2 int days = 25;
3 int c = 0;
4 while (c < days) {
5   // do stuff
6   c++; // short form of c = c + 1
7 }
```

For-Loop

- In many cases, we know in advance how often do repeat code

```
1 // do something for each of 25 days
2 int days = 25;
3 int c = 0;
4 while (c < days) {
5   // do stuff
6   c++; // short form of c = c + 1
7 }
```

```
1 // do something for each of 25 days
2 int days = 25;
3 for (int c = 0; c < days; c++) {
4   // do stuff
5 }
```

- For-loops offer a denser notation

For-Loop

```
1 for (INIT; CONDITION; UPDATE) {
2   //
3 }
```

- INIT: Executed before entering the loop for the first time
- CONDITION: An expression, checked before every iteration
  - Must be of type `boolean`
- UPDATE: Executed at the end of each iteration

For-Loop

Scope

- Variables declared within a for loop are not known outside of it
- If variables are declared in INIT, they belong to the scope of for-statement
- This shows a difference to the corresponding while-statement

Example

```
1 int a = 4;
2 for (int b = 0; b < 10; b++) {
3   // b is known
4   // a is known
5 }
6 // a is known
7 // b is not known
```

demo

Break and Continue

- All loops can *also* be controlled by two keywords: `break` and `continue`
- `break`
  - Terminates the entire loop abruptly
  - Execution continues after the closing `}`
- `continue`
  - Terminates the current iteration of the loop
  - Execution continues with the next iteration
    - `for`: Run UPDATE first
    - All loops check their conditions before

Break and Continue

- All loops can *also* be controlled by two keywords: `break` and `continue`
- `break`
  - Terminates the entire loop abruptly
  - Execution continues after the closing `}`
- `continue`
  - Terminates the current iteration of the loop
  - Execution continues with the next iteration
    - `for`: Run UPDATE first
    - All loops check their conditions before
- `break` / `continue` are sometimes useful, but
  - are able to exit a loop independently of the exit condition and thus
  - make code harder to read and understand

Understanding Loops

- Sometimes challenging to understand a loop
- Crucial: Keep track of variable contents
- Variables may change in every iteration
- Conditions/exit conditions can be complex

Understanding Loops

- Sometimes challenging to understand a loop
- Crucial: Keep track of variable contents
- Variables may change in every iteration
- Conditions/exit conditions can be complex

How many ! will be printed?

```
1 int a = 7;
2 while(a > 0) {
3   int f = a % 2;
4   if (f > 0) {
5     a = a - 2;
6   } else {
7     a = a + 1;
8   }
9   System.out.print("!");
10 }
```

Understanding Loops

- Sometimes challenging to understand a loop
- Crucial: Keep track of variable contents
- Variables may change in every iteration
- Conditions/exit conditions can be complex

How many ! will be printed?

```
1  int a = 7;
2  while(a > 0) {
3    int f = a % 2;
4    if (f > 0) {
5      a = a - 2;
6    } else {
7      a = a + 1;
8    }
9    System.out.print("!");
10 }
```

Line	a	f
1	7	
3	7	1
5	5	1
3	5	1
5	3	1

!!

Section 4

Exercise