Software and Programming I

More on Loops and Expressions

Roman Kontchakov
Birkbeck, University of London
Outline

- The **while**, **for** and **do** Loops
  - Sections 4.1, 4.3 and 4.4
- Return Statement
  - Section 5.4
- Expressions and Types
- Operation Precedence
Loops and Assignments

```java
int i = 6;
while (i >= 0) {
    System.out.println(i - 1);
    i = i - 2;
}
```

<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>i &gt;= 0</th>
<th>i - 1</th>
<th>i - 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>true</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>true</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>true</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>true</td>
<td>-1</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>false</td>
<td>END OF THE LOOP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The for Loop

1 for (int k = 2; k <= 9; k++) {
2     String s = s0;
3     if (k % 2 == 1)
4         s = s1;
5     System.out.println(k + " is " + s);
6 }

initialisation (statement)   condition boolean expression   update (statement)
And The while Loop

initialisation (statement)

condition boolean expression

update (statement)

1 \texttt{int} k = 2;
2 \texttt{while} (k <= 9) {
3 \hspace{1em} \texttt{String} s = \texttt{s0};
4 \hspace{1em} \texttt{if} (k \% 2 == 1)
5 \hspace{2em} s = \texttt{s1};
6 \hspace{1em} \texttt{System.out.println}(k + " is " + s);
7 \hspace{1em} k++;
8 }
The do Loop

the do loop is appropriate when
the loop body must be executed at least once

```java
int value;
do {
    System.out.println("Enter an integer < 100: ");
    value = in.nextInt();
} while (value >= 100);
```

**NB:** do not forget the semicolon ; after
Return Values

The `return` statement terminates a method call and yields the method result.

```java
public static double cubeVolume(double sideLength) {
    if (sideLength < 0)
        return 0;
    // more code
    return sideLength * sideLength * sideLength;
}
```
Fibonacci Numbers: Maths

\[ f_1 = 1 \]
\[ f_2 = 1 \]
\[ f_n = f_{n-1} + f_{n-2}, \text{ for } n > 2 \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_n )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>21</td>
</tr>
</tbody>
</table>
Fibonacci Numbers

```java
public static int fibonacci(int k) {
    if (k <= 2) {
        return 1; // exit
    }
    int fn2 = 1; // f_{n-2} = 1 for n = 3
    int fn1 = 1; // f_{n-1} = 1 for n = 3
    int fn = fn1 + fn2; // f_n = 3 for n = 3
    for (int n = 4; n <= k; n++) {
        fn2 = fn1;
        fn1 = fn;
        fn = fn1 + fn2;
    }
    return fn;
}
```

<table>
<thead>
<tr>
<th>n</th>
<th>fn2</th>
<th>fn1</th>
<th>fn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>13</td>
<td>21</td>
</tr>
</tbody>
</table>
Loop Termination

Collatz conjecture

Lothar Collatz, 1937

The sequence \( a_{n+1} = \begin{cases} \frac{a_n}{2}, & \text{if } a_n \text{ is even} \\ 3a_n + 1, & \text{if } a_n \text{ is odd} \end{cases} \)

eventually reaches 1 regardless of which positive integer \( a_0 \) is chosen.

```java
while (a > 1) {
    if (a % 2 == 1)
        a = 3 * a + 1;
    else
        a = a / 2;
}
```
suppose $a$ is 4 and $b$ is 5. what is the value of $b > a$?

```java
public static boolean greater(int a, int b) {
    return b > a; // returns true if b > a
}

boolean found = false;
while (!found) {
    ... // do something
    if (...) // if the condition is met
        found = true;
    ... // do something else
}
```
Suppose there are two variables:

```c
int a, b; // the same as int a, int b
```

How do you swap the contents of `a` and `b`?

```c
int t = a;
```

```c
a = b;
```

```c
b = t;
```
Expressions

assignment statement  \[ \text{cansPerPack} = 8 \; ; \]

an **expression** is a combination of variable names, literals, method calls and operators

the **type** of an expression is known at compile-time:

- 8 is of type `int`
- 10.2 and -12.3e-45 are of type `double`
- "foo^=\nbar" is of type `String`
- false and true are of type `boolean`

**NB:** types of variables are declared (initialisation is, for example, `=` 9)
suppose $expr_1$ and $expr_2$ are expressions

- the type of $expr_1 + expr_2$ is
  - $int$ if the type of both $expr_1$ and $expr_2$ is $int$
  - $double$ if the type of one of $expr_1$ or $expr_2$ is $double$
    and the other type is numerical, i.e., $int$ or $double$
  - $String$ if the type of one of $expr_1$ or $expr_2$ is $String$
  - otherwise, it is a compile-time error

**NB:** what is the type of $false + 1$?

- similar rules apply to -, *, / and %
  except they are **not** defined on $String$
suppose \( expr_1 \) and \( expr_2 \) are expressions

- \( expr_1 < expr_2, \ expr_1 \leq expr_2, \ expr_1 \geq expr_2 \)
  and \( expr_1 \geq expr_2 \) are of type \texttt{boolean}

both \( expr_1 \) and \( expr_2 \) must be of \texttt{numerical} datatypes

compile-time error otherwise

\textbf{NB:} what is the type of \( 60 \leq marks \leq 69 \)?

- \( expr_1 \mid \mid expr_2, \ expr_1 \&\& expr_2 \) and \( ! expr_1 \)
  are of type \texttt{boolean}

both \( expr_1 \) and \( expr_2 \) must be of type \texttt{boolean}

compile-time error otherwise

\textbf{NB:} what is the type of \( 60 \leq marks \&\& \leq 69 \)?
Operation Precedence

- `()` method call  
- `!,(type)` type cast  
- `*,,/,%` multiplicative  
- `+,-` additive  
- `<,<=,>=,>` relational  
- `==,!=` equality  
- `&&` logical AND  
- `||` logical OR
Operation Precedence

What is the value of \( 2 + a \mod 3 \) if \( a \) is 11?

- \( 2 \times 6 + a \mod 3 + 1 < 10 \) && \( a > 3 \) ?
- \( 2 \times 6 + a \mod 3 + 1 < 10 \) && \(!a > 3\) ?
- \( 2 + a / 3 \) ?
- \( 2 + (\text{double}) a / 3 \) ?
Take Home Messages

- The **while** loop executes instructions repeatedly while a condition is true.
- The **for** is used when a value runs from a starting point to an ending point with a constant increment.
- The **return** statement terminates a method call and yields the method result.
- Variables can have the same name provided their scopes do not overlap.