Object-oriented programming

Session 2

20th of February 2019
Ljubov Jaanuska
How many hours did you spend on the course last week?
How you succeed in this course?
A reminder

• In-class sessions (max 12 points)
• Self-study at home (max 12 points)
• Material:
  – https://courses.cs.ut.ee/2019/oopn/spring/Main/Materials
  – Moodle

• To get an access to the exam, a student must have:
  * at least 10p. for Test 1;
  * at least 12p. for home assignments & group works;
  * at least 4p. for the in-class sessions

• To pass the exam, at least 15p. must be obtained.
I would like to start with
Numbers and data types

- **Whole numbers**
  - **byte**
    - -128 kuni 127
  - **short**
    - -32 768 kuni 32 767
  - **int**
    - -2 147 483 648 kuni 2 147 483 647
  - **long**
    - -9 223 372 036 854 775 808 kuni 9 223 372 036 854 775 807

- **Fractions**
  - **float**
  - **double**
Data type conversion

• Computation in expressions
  – If one operand is a `double`, then the second one is converted into a `double`
  – Otherwise, if one operand is a `long`, then the second one is converted into a `long`
  – Otherwise, both are converted into an `int`

1.0/2
1/2
Data type conversion

• If we want to convert into a narrower data type, then we have to specify it:

```cpp
double f = 4;
double d = 6.4;
int i = (int) d; // is not changed automatically, (6)
double u = i; // is changed automatically (6.0)
```
String

- The string is not a primitive data type in Java

```java
String s1 = "Is";
String s2 = "February";
String s3 = s1 + " " + s2;
```
What is the output?

```java
int i = 1, j = 2;
System.out.println("i + j = "+i+j);
```
What is the output?

```java
int i = 1, j = 2;
System.out.println("i + j = " + (i + j));
```
# Logical operators

<table>
<thead>
<tr>
<th>Java</th>
<th>Python</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>!</code></td>
<td><code>not</code></td>
<td>Logical NOT (unary)</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td><code>and</code></td>
<td>Logical AND (binary)</td>
</tr>
<tr>
<td>`</td>
<td></td>
<td>`</td>
</tr>
<tr>
<td>Operation</td>
<td>Precedence</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td>$a++$ $a--$</td>
<td>High precedence</td>
<td></td>
</tr>
<tr>
<td>$++a$ $--a$ $+a$ $-a$ $\sim$ $!$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$*$ $/$ $%$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+$ $-$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$&lt;&lt;$ $&gt;&gt;$ $&gt;&gt;&gt;$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$&lt;$ $&gt;$ $&lt;=$ $=&gt;$ $instanceof$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$==$ $!=$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$&amp;$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$^$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>$</td>
<td></td>
</tr>
<tr>
<td>$&amp;&amp;$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$||$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$?$ $:$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+= -$ $*= /= %= &amp;= ^=</td>
<td>&lt;=$ $&lt;&lt;= &gt;&gt;= &gt;&gt;&gt;= $</td>
<td>$</td>
</tr>
</tbody>
</table>

We are not going to use all of them in this course.
If operators are of the same level

• Assignment
  – from right to left

\[ x = y \ += z = 1; \]

• The rest binary operators
  – from left to right

\[ 5 \ % \ 2 \ * \ 3 \ / \ 6 \]
What is the output?

```java
int x, y, z;
x = y += z = 1;
System.out.println(x);
```
What is the output?

```java
int x, y, z;
y = 0;
x = y += z = 1;
System.out.println(x);
```
What is the output?

System.out.println(5 % 2 * 3 / 6);
Conditional statements

```java
if (BooleanExpression) {
    statement(s) if expression is true;
}
else {
    statement(s) if expression is false;
}
```
if (sum >= 91)
    grade = 'A';
else
    if (sum >= 81)
        grade = 'B';
    else
        if (sum >= 71)
            grade = 'C';
        else
            if (sum >= 61)
                grade = 'D';
            else
                if (sum >= 51)
                    grade = 'E';
                else
                    grade = 'F';

if (sum >= 91)
    grade = 'A';
else if (sum >= 81)
    grade = 'B';
else if (sum >= 71)
    grade = 'C';
else if (sum >= 61)
    grade = 'D';
else if (sum >= 51)
    grade = 'E';
else
    grade = 'F';

If a block contains only one statement, the curly brackets are not compulsory!
What is the output?

```java
int i = 1, j = 2, k = 3;
if (i > j)
    if (i > k)
        System.out.println("A");
else
    System.out.println("B");
```
Indentation does not matter, but strongly recommended!

class Hello {
  public static void main(String[] args)
  {
    System.out.println("Hi");
  }
}
Conditional expressions

BooleanExpression ? expression1: expression2;

• If BooleanExpression is true, then the value is expression1
• Otherwise, the value is expression2
What is the output?

```c
int i, k;
i = -10;
k = i < 0 ? -i : i;
```
Switch

```java
switch (expression) {
    case value1: statement(s)1; break;
    case value2: statement(s)2; break;
    ...
    case valueN: statement(s)N; break;
    default: default_statement(s);
}
```

- **expression**
  - char, byte, short, int, String
  - `value1,...` - constants (no variables)
  - `break;` - ends work in the block
Loops

• General loop with three expressions - *for*
• Pre-check loop - *while*
• Post-check loop - *do-while*
Loops

```cpp
for (initialization; termination; increment){
    // statements that to be completed
    // until the termination expression becomes true
}

while (termination_expression) {
    // statements that to be completed
    // until the termination expression becomes true
}

do {
    // statements that to be completed
    // until the termination expression becomes true
}
while (termination_expression);
```
Flow diagrams

**for loop**

- initialization
- termination expression
  - Yes: statements
  - No: increment

**while loop**

- termination expression
  - No: statements
  - Yes: statements

**do-while loop**

- statements
- termination expression
  - Yes: statements
Variable scope

- Variables defined inside a block are *local variables*.
- Local variable scope starts with its declaration and ends at the end of the block with the declaration.
- Different blocks can contain different variables with the same name.
- It is not possible to use a variable outside the block where the variable is declared.
What is the output?

```java
int sum = 0;
for(int i = 0; i < 5; i++){
    sum += i;
}
System.out.println(i);
```
What is the output?

```java
int sum = 0;
int i;
for(i = 0; i < 5; i++){
    sum += i;
}
System.out.println(i);
```
What is the output?

```java
int sum = 0;
int i = 0;
for(; i < 5; i++){
    sum += i;
}
System.out.println(i);
```
What is the output?

```java
int sum = 0;
int i = 0;
for(; i < 5; i++) {
    sum += i;
}
System.out.println(sum);
```
Block

- `{ statements }
  - class
  - method
  - loop
  - if
  - ...
  - ...
  - grouped statements

```java
{  
  int j;
  j = 7;
}
System.out.println(j);
```
A little bit more about \texttt{for} loops

- Initialization can contain
  \begin{itemize}
  \item 0 \hspace{1em} \texttt{for} (; \hspace{0.5em} i < 5; \ i++)
  \item 1 \hspace{1em} \texttt{for} (i = 0; \ i < 5; \ i++)
  \item more \hspace{0.5em} \texttt{for} (i = 0, \ j = 0; \ i < 5; \ i++)
  \end{itemize}

- Termination can contain
  \begin{itemize}
  \item 0 \hspace{1em} \texttt{for} (i = 0; \ ; \ i++)
  \end{itemize}

- Increment can contain
  \begin{itemize}
  \item 0 \hspace{1em} \texttt{for} (i = 0; \ i < 5;)
  \item 1 \hspace{1em} \texttt{for} (i = 0; \ i < 5; \ i++)
  \item more
    \begin{itemize}
    \item \texttt{for} (i = 0, \ j = 0; \ i < 5; \ i++, \ j++)
    \end{itemize}
  \end{itemize}
Infinite loops

• `for( ; ; ) {...}
• `while (true) {...}
• `do {...} while (true);
break and continue

• **break** - breaks the iteration and jumps out of the loop

• **continue** - breaks the current iteration and proceed with the next iteration;
Array vs. list

• Array (a finite sequence)
  – Static data structure (length is fixed)

• List
  – Dynamic data structure (length is mutable, elements can be added and deleted)
Array

- Length cannot be changed, elements cannot be added or deleted

```java
int[] b = new int[3];
b[0] = 10;
b[1] = 5;

for (int elem : b){
    System.out.println(elem); // 10 5 0
}

String[] a = {"first", "second", "third"};

//does not work with primitive data types
//does not work with e.g. int[]
System.out.println(Arrays.asList(a));

//System.out.println(Arrays.asList(b)); //[[I@2a139a55]
```
Methods

```java
static void greeting(){
    System.out.println("Hi!");
}
static double sumNum(double a, double b)
{
    return a + b;
}
```

Method signature:
greeting()
sumNum(double, double)
Concepts

• **Overloading** is a concept used to avoid redundant code where the same method name is used multiple times but with a different set of parameters.

• The **signature** of is a description of a method which contains:
  – a method name of the method
  – data types of its parameter
Any questions about home assignment?