Programming 1

Christian Grévisse (`christian.grevisse@uni.lu`)
Gilles Neyens (`gilles.neyens@uni.lu`)
UNIVERSITÉ DU LUXEMBOURG

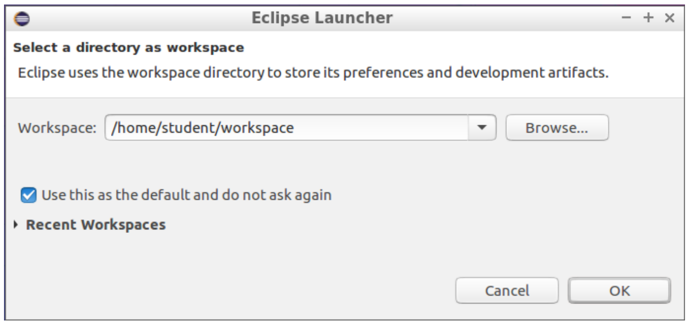
Lab 1 – "Hello, World!", Variables & Basic Sequence of Instructions

2017–09–26

Exercise 1 – Getting started with Eclipse

In this exercise, you will have some first hands-on experience with the Eclipse IDE. The *Hello, World!* example is a typical entry point when learning a new programming language.

1° Open Eclipse. When first opened, it will ask you where to put your workspace, i.e. the directory where all your future projects shall be stored.



Eclipse Launcher

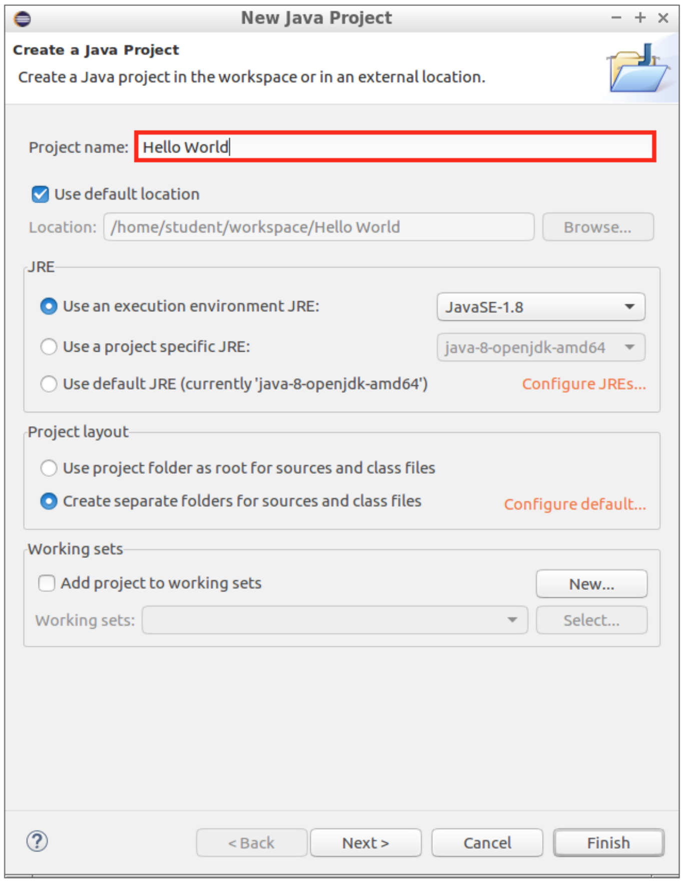
Select a directory as workspace

Eclipse uses the workspace directory to store its preferences and development artifacts.

Workspace: /home/student/workspace

☑ Use this as the default and do not ask again

2° Select `File` → `New` → `Java Project`. In the appearing dialog, give the project a name (e.g. `Hello World`) and click on `Finish`.



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: Hello World

☑ Use default location

Location: /home/student/workspace/Hello World

JRE

● Use an execution environment JRE: JavaSE-1.8

○ Use a project specific JRE: java-8-openjdk-amd64

○ Use default JRE (currently 'java-8-openjdk-amd64')

Project layout

○ Use project folder as root for sources and class files

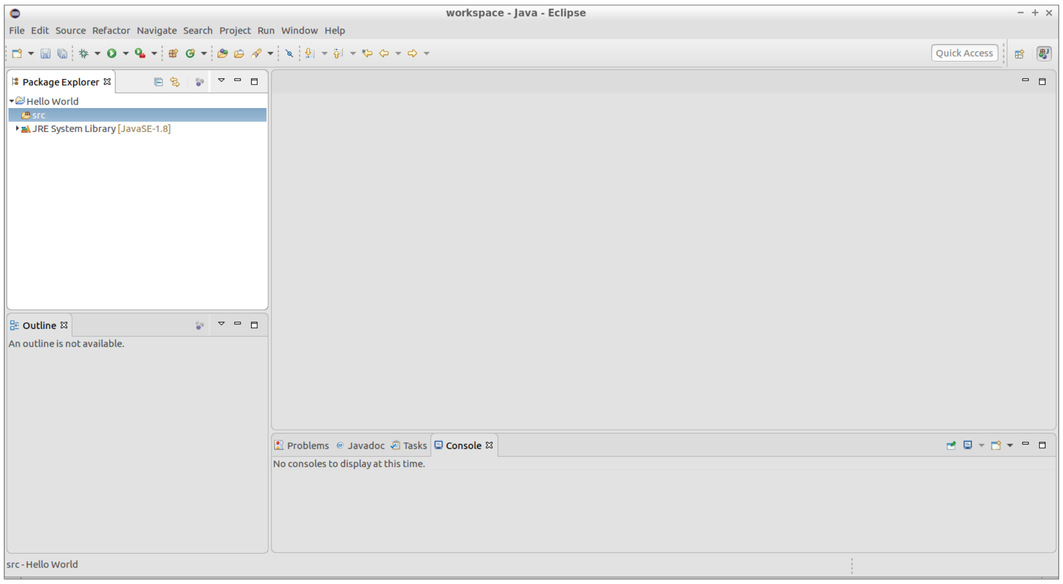
● Create separate folders for sources and class files

Working sets

☐ Add project to working sets

Working sets:

As you can see in the `Package Explorer` in the upper left corner, a new, empty project is created. Code will be stored in the `src` folder.

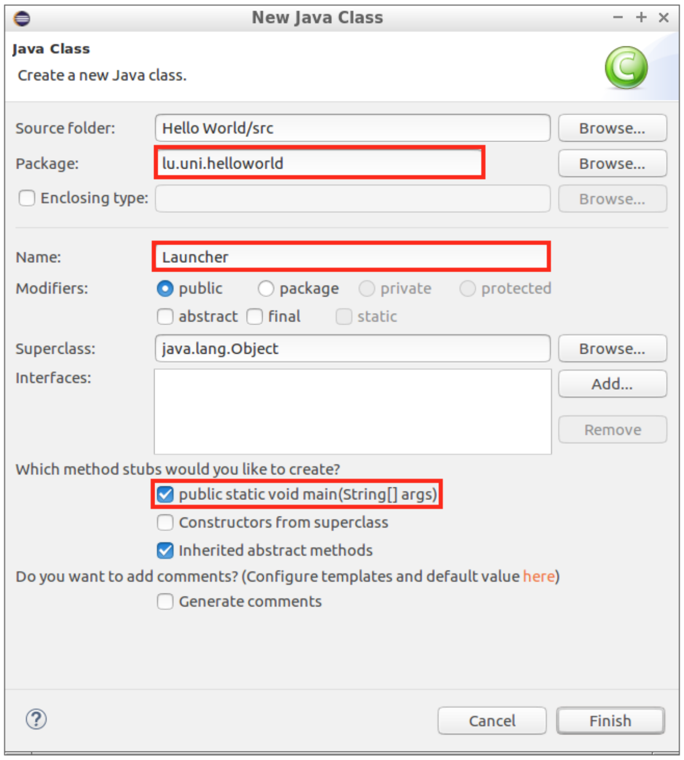

workspace - Java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
Hello World
src
JRE System Library [JavaSE-1.8]
Outline
Problems
Javadoc
Tasks
Console
src - Hello World

An outline is not available.

No consoles to display at this time.

3° In the menu `Window` $\rightarrow$ `Show View`, you can enable additional views, such as the `Console View` (which outputs console logs).

4° In order to create a new Java *class*, select `File` $\rightarrow$ `New` $\rightarrow$ `Class`. In the dialog, set the package name to `lu.uni.helloworld` and the class name to `Launcher`. Please make sure that the checkbox next to `public static void main(String[] args)` is selected. Click on `Finish`.



Java Class

Create a new Java class.

Source folder: Hello World/src

Package: lu.uni.helloworld

Enclosing type: [ ]

Name: Launcher

Modifiers: [✓ public] package private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

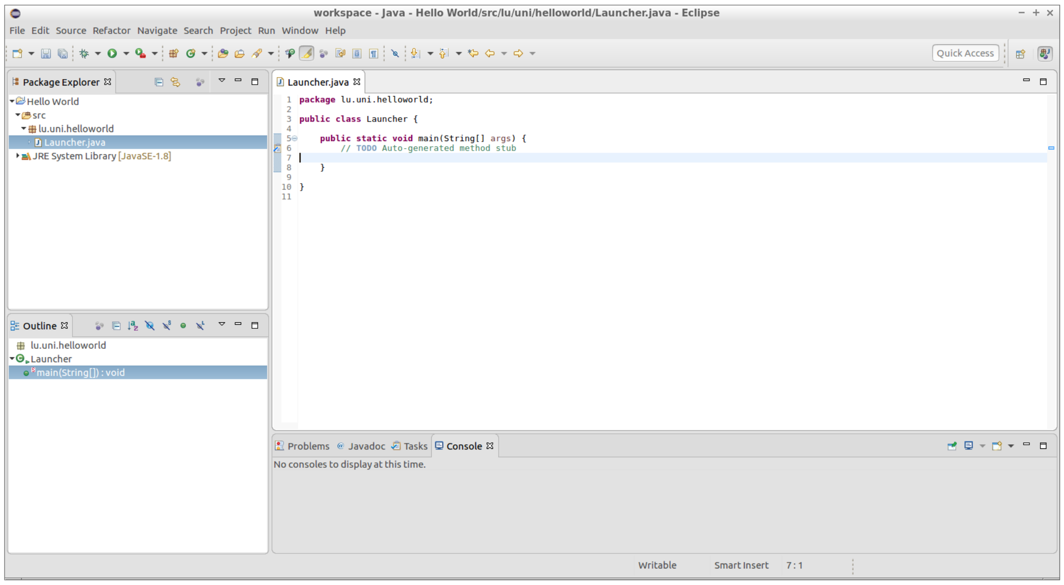
[ ] Constructors from superclass

[✓] Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

As you can see, the editor shows you a file `Launcher.java` for the newly created class `Launcher`. It lies within the package `lu.uni.helloworld`, as you can see in the Package Explorer. The `main`-*method* is the entry point of a Java program. More on packages and the `main`-method in the lecture. Note that the auto-generated stub in this class is a *comment*, i.e. a statement not taken into account by the compiler. Comments are used to make a piece of code more readable to the human eye, by giving explanations on the code.


workspace - Java - Hello World/src/lu/uni/helloworld/Launcher.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
Hello World
src
lu.uni.helloworld
Launcher.java
JRE System Library [JavaSE-1.8]
Outline
lu.uni.helloworld
Launcher
main(String[]) : void
Launcher.java
Problems Javadoc Tasks Console
Writable Smart Insert 7 : 1

```java
package lu.uni.helloworld;

public class Launcher {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

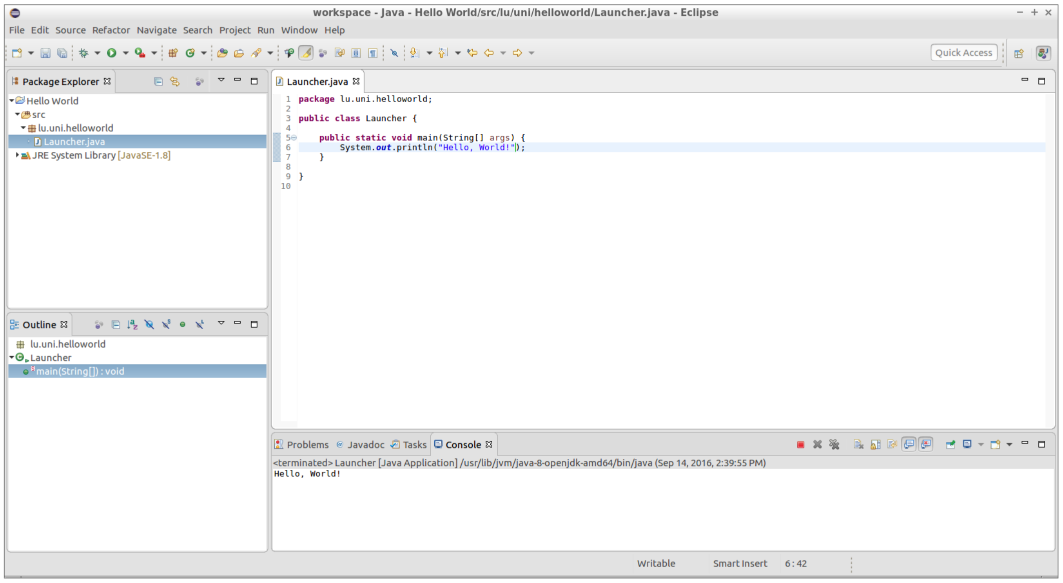
```
No consoles to display at this time.
```

5° Replace the auto-generated method stub by the following statement:

```java
System.out.println("Hello, World!");
```

This statement will print *Hello, World!* to the console. Note that statements in Java are terminated by a semicolon (`;`).

6° Run your project by choosing `Run` → `Run`. This action compiles the Java code into Java bytecode, which will get interpreted by the *Java Virtual Machine (JVM)*. More on this also during the lecture. In the Console view, you will see the output.


workspace - Java - Hello World/src/lu/uni/helloworld/Launcher.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
Hello World
src
lu.uni.helloworld
Launcher.java
JRE System Library [JavaSE-1.8]
Outline
lu.uni.helloworld
Launcher
main(String[]) : void
Launcher.java
Problems Javadoc Tasks Console
Writable Smart Insert 6 : 42

```java
package lu.uni.helloworld;

public class Launcher {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }

}
```

```
<terminated> Launcher [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Sep 14, 2016, 2:39:55 PM)
Hello, World!
```

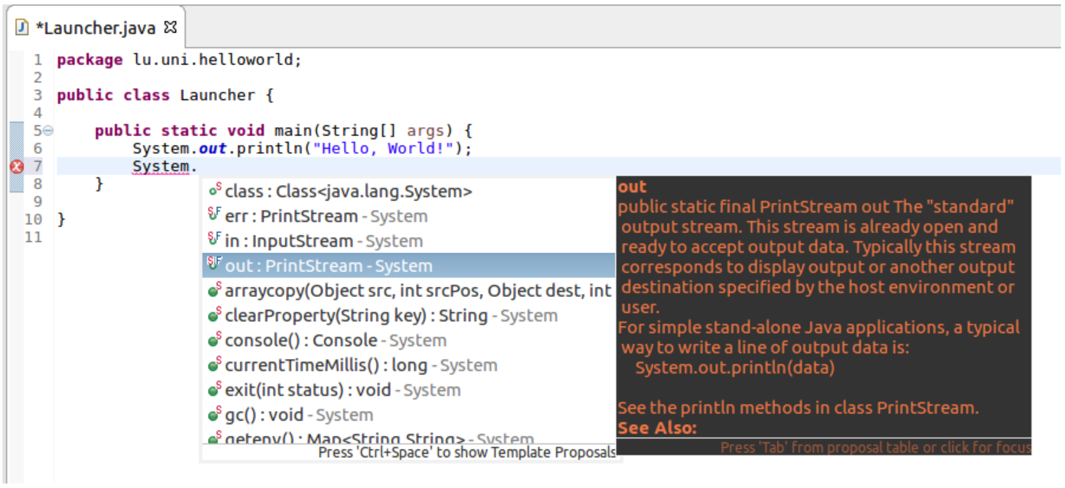
Programming 1

Christian Grévisse (christian.grevisse@uni.lu)
Gilles Neyens (gilles.neyens@uni.lu)

Lab 1 – "Hello, World!", Variables & Basic Sequence of Instructions

Exercise 2 – Autocompletion & Documentation

1° Add another `System.out.println` statement, but this time, only write `System.` and wait for a context menu to open. Alternatively, you can also press `Ctrl + Space` to open this context menu. It will suggest *attributes* and *methods* in the `System` class. The list can be navigated with the arrow keys and an element can be selected by pressing Enter/Return.



*Launcher.java

```java
package lu.uni.helloworld;

public class Launcher {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.
    }

}
```

```
class : Class<java.lang.System>
err : PrintStream - System
in : InputStream - System
out : PrintStream - System
arraycopy(Object src, int srcPos, Object dest, int
clearProperty(String key) : String - System
console() : Console - System
currentTimeMillis() : long - System
exit(int status) : void - System
gc() : void - System
getenv() : Map<String, String> - System
Press 'Ctrl+Space' to show Template Proposals
```

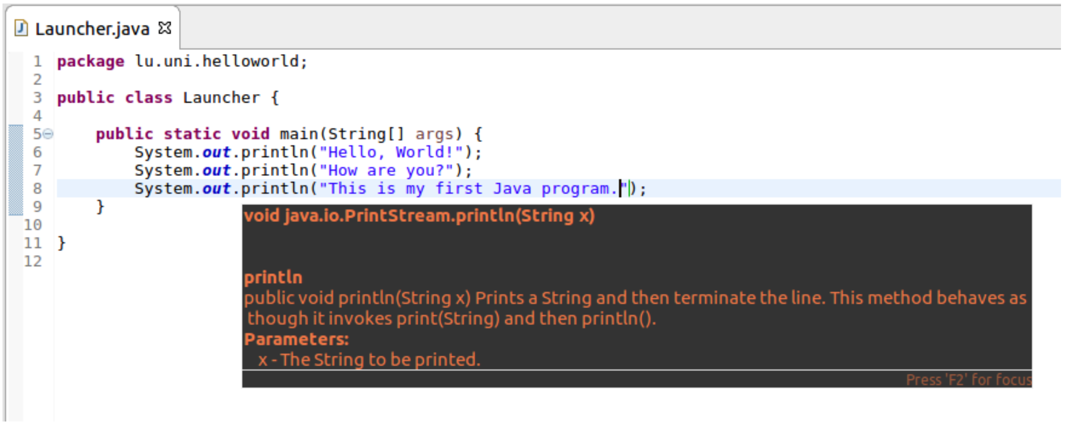
out
public static final PrintStream out The "standard" output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.
For simple stand-alone Java applications, a typical way to write a line of output data is:
  System.out.println(data)

See the println methods in class PrintStream.
See Also:
Press 'Tab' from proposal table or click for focus

This autocompletion feature may increase your coding efficiency, in opposition to typing everything. In addition, it gives you insights to the internal structure of a class.

2° Add yet another `System.out.println` statement, but this time, only write `sysout` and press `Ctrl + Space`, which will expand this shortcut to the full statement.

3° With the cursor, hover over `System`, `out` or `println`. You will see a tooltip with related *Javadoc* comments. In this case, they were defined in the Java SE API[1].



Launcher.java

```java
package lu.uni.helloworld;

public class Launcher {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.out.println("How are you?");
        System.out.println("This is my first Java program.");
    }

}
```

void java.io.PrintStream.println(String x)

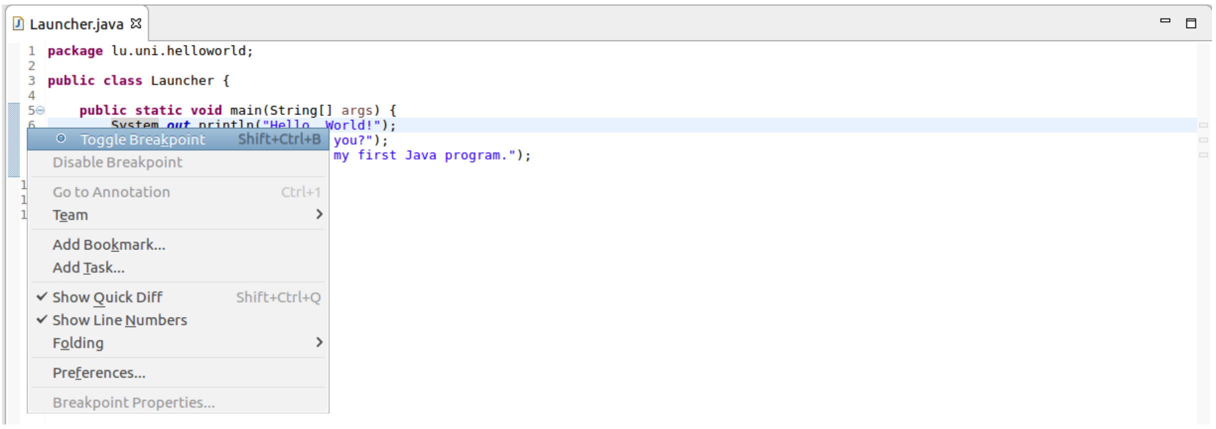
println
public void println(String x) Prints a String and then terminate the line. This method behaves as though it invokes print(String) and then println().
Parameters:
  x - The String to be printed.
Press 'F2' for focus

[1] https://docs.oracle.com/javase/8/docs/api/

Christian Grévisse (christian.grevisse@uni.lu)
Gilles Neyens (gilles.neyens@uni.lu)

Exercise 3 – Debugging

1° Right-click in the margin of the line with the first `System.out.println` statement and choose `Toggle Breakpoint`.

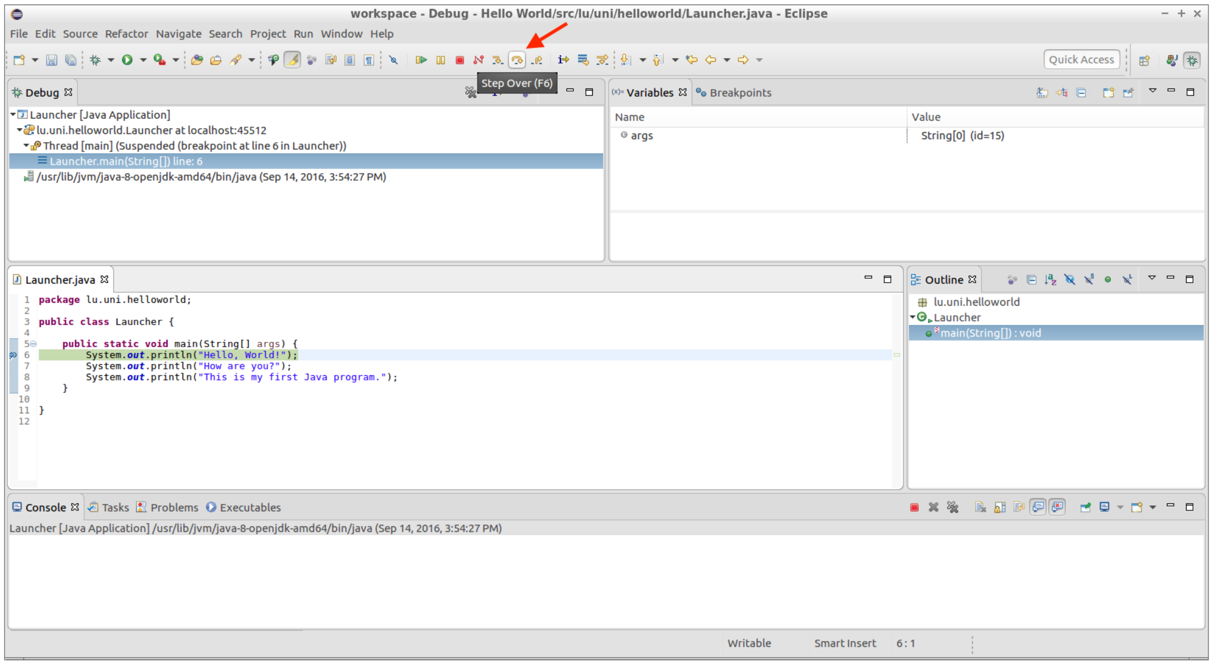


```
package lu.uni.helloworld;

public class Launcher {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
                                  you?");
                                  my first Java program.");
```

Alternatively, you may also double-click in the margin.

2° Choose `Run` → `Debug` to run the program in the *Debug* perspective (when prompted, confirm the perspective switch). The program will enter, as usual, through the `main`-method and will stop before the breakpoint.

3° Click on the `Step Over` icon (or press `F6`) to execute the statement. Do this several times to execute the program line by line. Observe both the highlighted statement in the Editor view and the output on the Console view.



workspace - Debug - Hello World/src/lu/uni/helloworld/Launcher.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Step Over (F6)
Debug
Variables
Breakpoints
Launcher.java
Outline
Console
Tasks
Problems
Executables
Writable
Smart Insert
6 : 1

```
Launcher [Java Application]
  lu.uni.helloworld.Launcher at localhost:45512
    Thread [main] (Suspended (breakpoint at line 6 in Launcher))
      Launcher.main(String[]) line: 6
  /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Sep 14, 2016, 3:54:27 PM)
```

Name	Value
args	String[0] (id=15)

```
package lu.uni.helloworld;

public class Launcher {

    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.out.println("How are you?");
        System.out.println("This is my first Java program.");
    }

}
```

```
lu.uni.helloworld
Launcher
  main(String[]) : void
```

```
Launcher [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Sep 14, 2016, 3:54:27 PM)
```

4° Once the last statement in the `main`-method has been executed, the program will terminate. To go back to the usual Java perspective, choose `Window` → `Perspective` → `Open Perspective` → `Java`.

Programming 1

Christian Grévisse (christian.grevisse@uni.lu)
Gilles Neyens (gilles.neyens@uni.lu)

UNIVERSITÉ DU LUXEMBOURG

Lab 1 – "Hello, World!", Variables & Basic Sequence of Instructions

✎ Plugin Management in Eclipse

As presented in the introduction session last week, we are currently developing a plugin for Eclipse which allows you to retrieve learning material within the IDE. This plugin, called ALMA, is installed on the Ubuntu Virtual Machine. However, as it is still in development, you will probably need to update it on a weekly basis to check for new features and bug fixes.

In general, to **install plugins in Eclipse**, it is necessary to add the repository the plugin is hosted at. To do so, choose Help › Install New Software ... . Choose an already existing repository or click on Add ... to add a new one. The URL of the ALMA plugin repository is `https://alma.uni.lu/eclipse/`. Select the plugin you wish to install (if nothing is shown in the list, deselect Group items by category).

To **uninstall a plugin**, choose Help › About, then Installation Details. Under Installed Software, choose, e.g., *ALMA*, to uninstall the plugin.

If an **update of a plugin** already installed is available, it will show up when choosing Help › Check for Updates.

Note that these actions usually require you to restart the IDE.

✎ Use of the ALMA Plugin

To use the ALMA plugin, you (currently) need to show its view by selecting Window › Show View › Other... . Under the category *ALMA*, choose *ALMA - Learning Aspects* and click Open. When opening a Java class from the package explorer, the view is updated with the *Abstract Syntax Tree* of the class' code.

If you have an issue in understanding an aspect of the code and want to know more about it, you can retrieve related learning material (if available) by selecting the part of code you do not understand (or simply putting the cursor within, e.g., a keyword) and clicking on the 🌐 icon.

On the lab sheets, you will also find sometimes indications like #If on the right-hand side of an exercise title. Use these hashtags in a Java class by putting it as a comment in your code:

```java
public MyFancyClass {
  public static void main(String[] args) {
    // #If
  }
}
```

When you put the cursor on this comment and click the 🌐 icon, material corresponding to the concept *If Conditions* will be shown (this concept will actually be explained during the lecture on Friday).

✎ Reading User Input

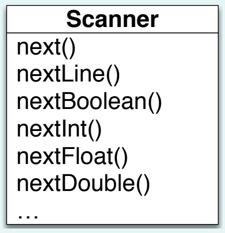
The `Scanner` class can parse primitive types and character strings from input streams. Among others, standard input (i.e. keyboard input from the terminal, in Eclipse the `Console` view) can be captured. From the Java 8 documentation[a], we can read:

> *A Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods.*

Consider the following program. Note that all code is within the `main` method.

```java
package lu.uni.scanner;

import java.util.Scanner;

public class AgeInput {

  public static void main(String[] args) {
    // Initialization of a Scanner object
    Scanner scanner = new Scanner(System.in);

    System.out.print("Please enter your age: ");

    // Reading a number from standard input
    int age = scanner.nextInt();

    System.out.println("You are " + age + " years old.");

    // Streams should be closed
    scanner.close();
  }

}
```

There exists a method for each specific primitive type (and strings), among others:



[a] `https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html`

Christian Grévisse (christian.grevisse@uni.lu)
Gilles Neyens (gilles.neyens@uni.lu)

Exercise 4 – Course Evaluation #VariableDeclaration #VariableInitialization #AssignmentOperator

Teachers are getting really lazy these days. Therefore, you are kindly requested to write a program that

- allows the user to enter the 3 grades for the `Programming 1` course (i.e. Midterm 1, Midterm 2, Final Exam) in the console
- calculates the weighted average (20%, 20%, 60%)
- prints the 3 grades and the average on the console

In a further step, you should ignore grades not belonging to the interval $[0, 20]$. If a number bigger than 20 is read from the console, limit it to 20. Similarly, a number lower than 0 is adjusted to 0. As you do not yet know about decision structures, the methods `Math.min(float a, float b)` and `Math.max(float a, float b)` might come in handy.

Exercise 5 – The bill, please!

Programming makes hungry and thirsty. Write a program that allows a waiter to specify the unit price, the quantity and the VAT rate of an item. The program then calculates and outputs

- the total price without the VAT
- the VAT value
- the total price including the VAT

Exercise 6 – How's the weather?

As you probably know, there are different temperature scales. The most popular one in the United States is Fahrenheit, whereas most of the rest of the world uses Celsius. The formula of conversion is

$$t_{Fahrenheit} = t_{Celsius} \cdot \frac{9}{5} + 32$$

Write a program that reads a temperature on the Celsius scale from the console and calculate the corresponding temperature on the Fahrenheit scale. Output both values.

Exercise 7 – Swapping the Contents of 2 Variables

The purpose of this exercise is to exchange (swap) the values of two variables `a` and `b`, i.e. the value of `a` is assigned to `b` and vice versa.

1° The standard solution in order to swap the contents of two variables `a` and `b` is the following:

```
temp = a;
a = b;
b = temp;
```

Try to understand *why* we need a third variable (`temp`) and why the following answer is wrong:

```
a = b;
b = a;
```

Implement both possibilities and use the debugger (cf. Exercise 3) in order to observe step by step what is going on.

2° Now, write a program that swaps the contents of two variables `a` and `b` **without** using any additional variables.

Exercise 8 – Error Analysis

Try to find and correct the compiler errors in the following statements, first on paper, then verify within Eclipse and get familiar with the IDE's error messages. Indicate each time whether it is a syntactical or semantic error.

```java
public class ErrorAnalysis {
  public static void main(String[] args) {
    // 1.
    int a, b, c, d
    a = 200

    // 2.
    System.out.println((1+2)-3));

    // 3.
    int e = 0.1;

    // 4.
    f = f + 1;
  }
}
```

Exercise 9 – Prefix & Postfix Operators

What is the value of the variables i, j, k respectively the output in the different lines of the following code? Try to reason it first on paper, then try it within Eclipse using the debugger (cf. Exercise 3).

```java
public class Fix {
  public static void main(String[] args) {
    int i = 0, j, k;

    i++;
    System.out.println("i = " + i);
    ++i;
    System.out.println("i = " + i);
    System.out.println("i = " + ++i);
    System.out.println("i = " + i++);
    System.out.println("i = " + i);

    j = i++; System.out.println("i = " + i + ", j = " + j);
    j = ++i; System.out.println("i = " + i + ", j = " + j);

    k = i++ + ++j;
    System.out.println("i = " + i++ + ", j = " + ++j + ", k = " + k++);
    k = ++i + j++;
    System.out.println("i = " + i + ", j = " + j + ", k = " + k);
  }
}
```