LAB 01: ECLIPSE, INTELLIJ, AND "HELLO SWM WORLD"

Aims:

1. Get familiar with Eclipse, IntelliJ and the Virtual Desktop
2. Compile Java 10 source code using Eclipse and IntelliJ
3. Implement a simple object-oriented example
4. Working with existing code. Some Eclipse/IntelliJ tips.

Everyone has different levels of experience with IDEs and OO coding. Jump in at the section which suits your abilities.

PART 1: ECLIPSE/INTELLIJ BASICS

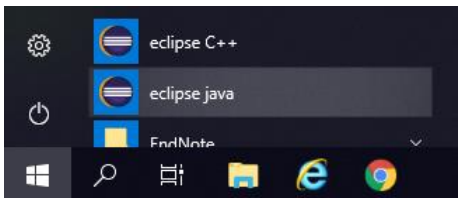
To maintain software you need to be familiar with modern integrated development environments (IDEs). Following the request of students from previous years, this year we support two IDEs in this module: Eclipse and IntelliJ. From now on, when we refer to "IDE's" we mean these two. As for the programming language, we will be using Java 10. You should have all used Java before, but as creating programs with Java 10 is slightly different, this part will guide you through a simple "Hello Java 10 World" program. Feel free to skip this if you know how to set up a new project in Java 10 and write "Hello Java 10 World".

One of the things introduced in Java 9 is formal modularity, in order to create structure inside large software systems. The new hierarchy in a Java project is as follows: Module > Package > Class > Fields and Methods. In one of the later lectures we will talk about the overall concept in more detail. For now we only need to be aware that this concept exists as we will have at least one additional file in new projects we create.

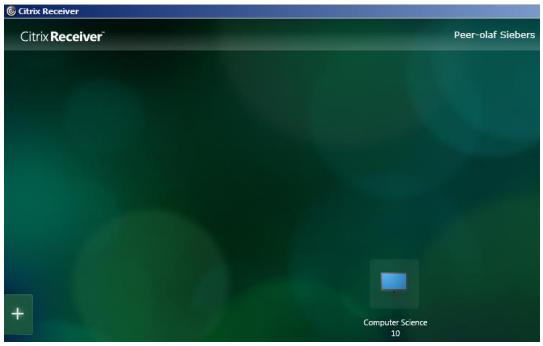
We would recommend that you try out the following in both environments, to familiarise yourself with both of them. Later on you can decide to go for one.

Starting the Eclipse IDE

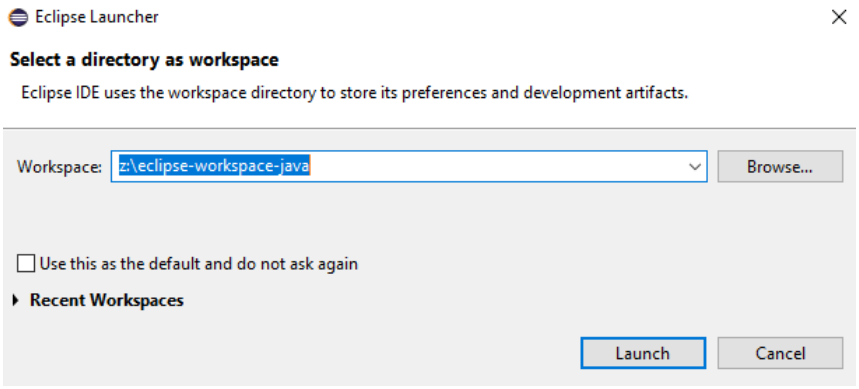
- There are two ways of accessing Eclipse for Java:
    - Directly access it on the lab machines (make sure you choose "eclipse java")



    - Access it through the virtual desktop (choose "Citrix Receiver" from the start menu to start your virtual desktop)

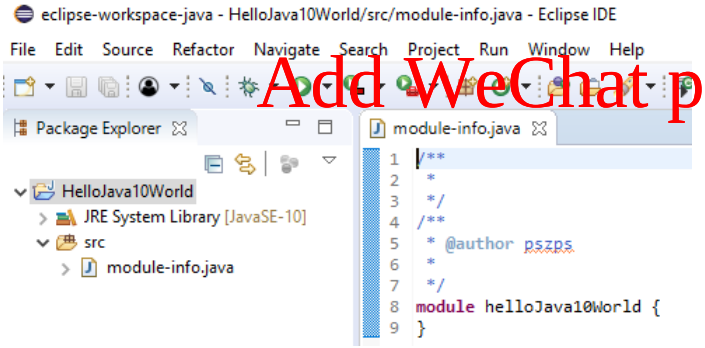


- Choose a workspace: Always make sure you are using your personal network storage (e.g. "z:\" as below).



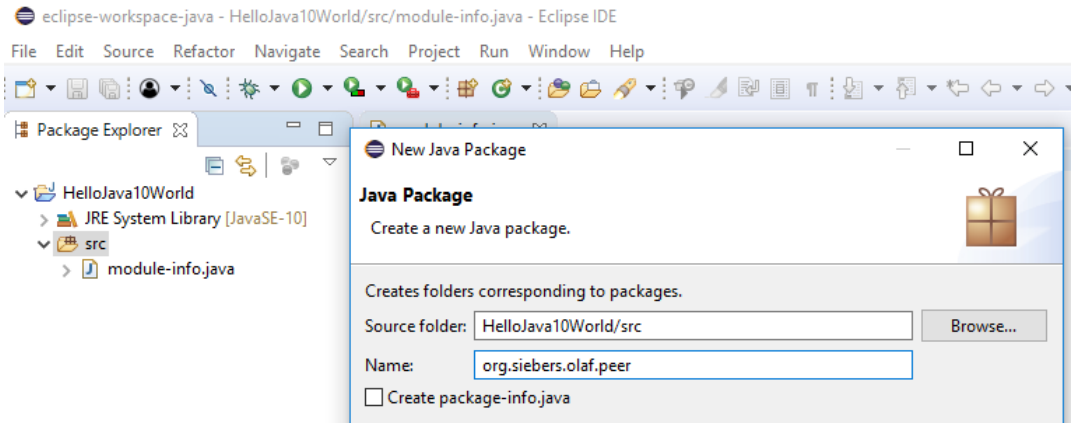
Create a new Java project

- Choose {File > New > Java Project}
- Call it "HelloJava10World" and make sure JavaSE-10 is selected as the JRE. Click {Finish}
- Eclipse will by default save the project in your workspace folder
- You will then be asked if you want to create a module-info.java file. This is for storing information about project modularity later. Change the module name to "helloJava10World" and click {Create}
- Your project should then appear in the Package Explorer (Note that we have not create any actual source code files yet)



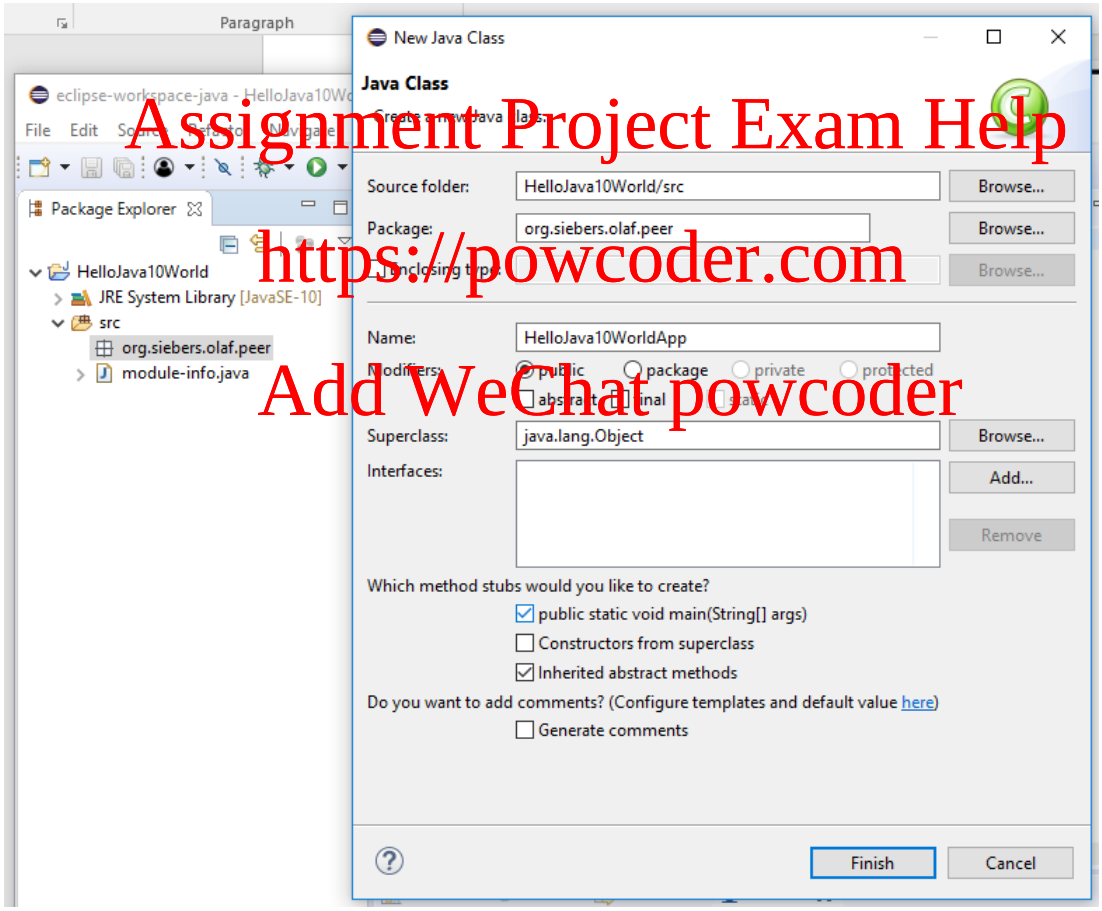
Create a package

- Right click on the "src" folder and choose {New > Package}. You can name it whatever you like but it is common to use a reverse order company name. Click {Finish}.

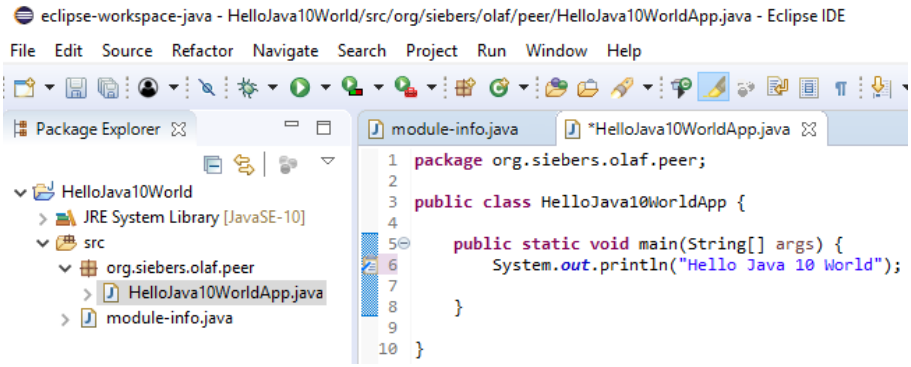


Create a class

- In the Package Explorer right click on the "src\package (package = whatever name you have given your package)" and choose {New > Class}.
- Give it the name "HelloJava10WorldApp" and choose {public static void main(String[] args)} as method stub to be created, and click {Finish}.

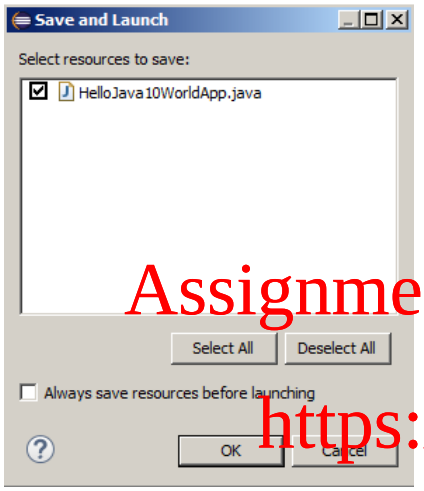


- Now, add some code to the class to print out "Hello Java 10 World"

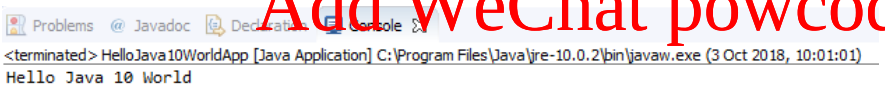


Execute the program

- Choose {Run > Run} to execute the program and click {OK} on the appearing window.



- Your text should then appear in the console window (at the bottom of the Eclipse window).



Challenge: Using IntelliJ for this job

In IntelliJ things work very similar. Try to work out how to build the same program in IntelliJ. You find IntelliJ under "JetBrains" in the Start menu.

Please note, that when you start IntelliJ for the first time you have to scroll down to the bottom of the "JetBrains Privacy Policy" window, before you can click the "Accept" button. Furthermore, when you create your first program, you have to provide the location of your Project SDK. Click {New...} and use the following path "C:\Program Files\ Java\jdk-10.0.1".

PART 2: AN OBJECT-ORIENTED EXAMPLE

The following is written for Eclipse but will work similarly in IntelliJ. Ask a lab helper if you get stuck in IntelliJ.

Create a new project "BikeProject". Add a package "com.theBestBikeShop". Download the code Bicycle.java from Moodle and import it into your project. To do this, drag and drop the file in the "src\ com.theBestBikeShop" package. When asked, choose to copy the file into your project workspace.

- If you want to check where the file is, right click the file in the Package Explorer and select Properties.

When you save the project it will automatically compile and there should be no errors, but it does not do anything, as it does not have a "main()" method yet.

Task: Extend the bike project...

- Create a BikeApp class including the "main()" method which instantiates a Bicycle object. Try changing the speed.
- Write a suitable constructor for this class so you can set the initial gear and speed values
- Add a method called "switchLight", which turns the light on if it is off, and vice versa
- Add a method "currentState" to print out the current speed, gear and light state.

Let's suppose we also want a class for a mountain bike, which has specific features associated with it. It makes sense to create a subclass of a Bicycle i.e. inherit from it.

In Eclipse this is in the package explorer, right click your Bicycle class and select {New > Class}. Change the name and click the box to add the constructor details from the superclass. Note that the superclass box is automatically filled in for you.

Hey presto, your subclass is created.

We will cover more about inheriting from classes in the lectures. But for now, let's just add two variables (Boolean) to store whether a bike has a front suspension and a rear suspension.

- Modify the constructor of MountainBike to take in value for the two suspension variables, and set them in the constructor.
- Add a method called isFullSuspension which returns *true* only if the bike has both front and rear suspension.
- Create objects for two different mountain bikes, one with full suspension, one without.

Challenge: Automating writing getters, setters, constructors etc.

The Eclipse IDE menu "Source" contains a lot of functionality for automating processes such as writing getters and setters and producing constructors. Delete all existing getters and setters and constructors within your bike source code and use "Source / Create Getters and Setters ..." and "Source / Generate Constructor using Fields..." and "Generate Constructors form Superclass ..." to get back to how the code looked before.

PART 3: WORKING WITH EXISTING GAME CODE

The following is written for Eclipse but will work similarly in IntelliJ. We are providing separate projects for Eclipse and IntelliJ. So make sure you are downloading "DiamondHunterIntelliJ.zip" in case you are working with IntelliJ. Ask a lab helper if you get stuck in IntelliJ.

Challenge: Use the Eclipse zip and make it work in IntelliJ.

Tip: For this you will have to fix some path dependencies.

In this part we will download and explore a larger existing codebase. To do this we'll learn some tips in Eclipse for navigating project code.

Let's have a look at some real world source code: a game called "Diamond Hunter". You can find more information about this game here: https://www.youtube.com/watch?v=AA1XpWHhxw0.

First you need to download the source code zip file from Moodle and import it into Eclipse. The file on Moodle is called "DiamondHunterEclipse.zip". Once you have downloaded the file, open Eclipse and choose {File > Import} and in the appearing pop-up choose {General > Existing Projects into Workspace}. Click on {Next} and in the next pop-up choose {Select archive file} and provide the location where you saved the zip file. Click {Finish}. To run it for the first time, go to the package "com.neet.DiamondHunter.Main" right click the package and choose {Run As > Java Application}.

Let's have a closer look at the Eclipse IDE "Navigate" menu:

HOW DO YOU FIND A SPECIFIC CLASS (OPEN A CLASS BY NAME)?

Let's assume we want to find the class "GameState". An efficient way is to choose {Navigate / Open Type ...} and search for "Game" and then double click the class "GameState" that is listed. This class will then appear in the editor window. The search feature accepts wildcards (e.g. "*"). [In IntelliJ use {Navigate > Class...}]

HOW DO YOU GET AN OVERVIEW OF METHODS AND FIELDS OF A CLASS?

You can find the class in the "Package Explorer", usually located in the left side of the Eclipse IDE. If you click on it will show the methods and fields of the class. Alternatively you can use the Quick Outline feature. If the class "GameState" is open in the "Editor" window you can also right click in the "Editor" window and choose "Quick Outline" and you will be provided with a pop-up window containing a list of all fields and methods of the class. You can use the text box in the top of this pop-up window to filter the list of fields and methods. The filtering feature also accepts wildcards (e.g. "*"). [In IntelliJ use {View > Tool Window > Structure}]

HOW CAN YOU SEE THE INHERITANCE TREE OF A SPECIFIC CLASS?

You can go to "Navigate / Open Type Hierarchy" which will show the inheritance tree of your chosen class in place of the "Package Explorer". In the bottom of that window you can see all the methods and fields that belong to the class you click once above. If you click on a class twice it appears in the "Editor" window. You can also right click in the "Editor" window and choose "Quick Type Hierarchy" and you will be provided with a pop-up window containing the inheritance tree of the class that is currently open. As before, you can use the text box in the top of this pop-up window to filter the list of classes. The filtering feature also accepts wildcards (e.g. "*"). [In IntelliJ use {Navigate > Type Hierarchy}]

Task:

- Try out some of the other options you find in the "Navigation" menu or when right clicking in the "Editor" to open a context specific menu.
- Have a look at the Eclipse Help to find out more

Let's have a closer look at the Eclipse IDE "Search" menu.

HOW CAN YOU FIND OUT IN WHICH CLASSES A SPECIFIC METHOD IS USED?

Let's find out in which classes the method "update()" is used. You could use "Edit / Find/Replace ...". This works like in a normal text editor. But Eclipse also has a more specialised search feature. With it you can find text in multiple files at the same time or you can find specific java constructs (e.g. method names that include a certain string). Let's have a look at the latter. Choose "Search / Search ..." to bring up the search dialog. Type "upd*" in the search string field and choose "Method" in the "Search For" field. Press the "Search" button. This will provide a list of all classes that contain a method whose name includes the string you searched for in the bottom window of the Eclipse IDE. The list also includes methods that call a method with "upd*" in their name. [In IntelliJ double click method name and choose {Edit > Find > Show Usages}]

HOW CAN YOU FIND OUT WHICH METHODS ARE CALLING A SPECIFIC METHOD?

Highlight the methods right click on it and choose {References > Project} from the appearing pop-up menu. A list of methods will then appear in the bottom window of the Eclipse IDE. Double clicking of a name will get you to the specific place where the method is used. [In IntelliJ double click method name and choose {Navigate > Call Hierarchy}]

Task:

- Try out some of the other options you find in the "Search" menu or when right clicking in the "Editor" to open a context specific menu.
- Have a look at the Eclipse Help to find out more

The Eclipse IDE provides many other little functions like these that makes the life of a programmer easier. This includes, amongst others, support for refactoring code and for debugging and testing code. We will cover this in a later lecture / lab.

Finally, don't forget to enjoy yourself and play the game :). Choose {Run > Run} from the menu bar.