LAB 1: ECLIPSE, BASIC OO PROGRAMMING, AND WORKING WITH EXISTING CODE

Aims:

1. Get used to the Virtual Desktop, and compiling Java using Eclipse within this environment
2. Implement a simple object-oriented example
3. Working with existing code. Some Eclipse tips.

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

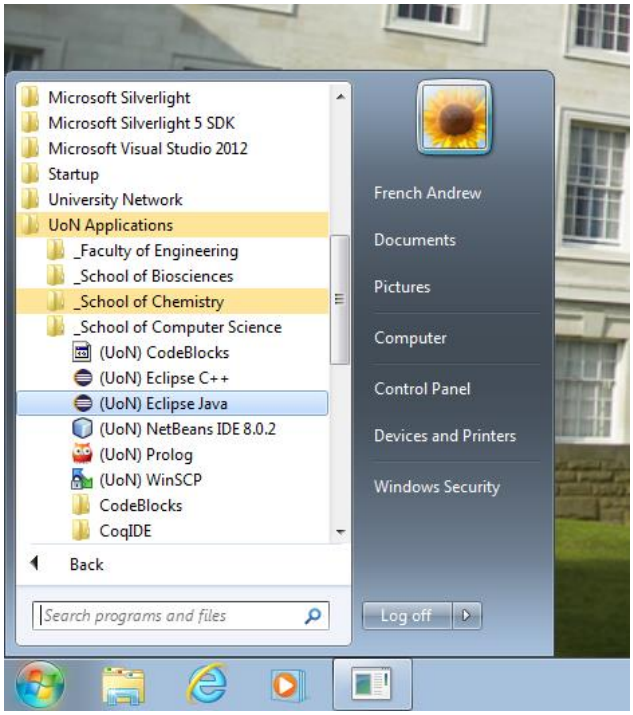
PART 1: ECLIPSE BASICS

To maintain software you need to be familiar with modern integrated development environments (IDEs). As we're using Java, we will use the Eclipse IDE. You all should have used this before, but as a refresher, this part of the lab will guide you through a simple Hello World program. Feel free to skip this if you know how to set up a new Java project and write Hello World.

There are two ways of accessing Eclipse for Java:

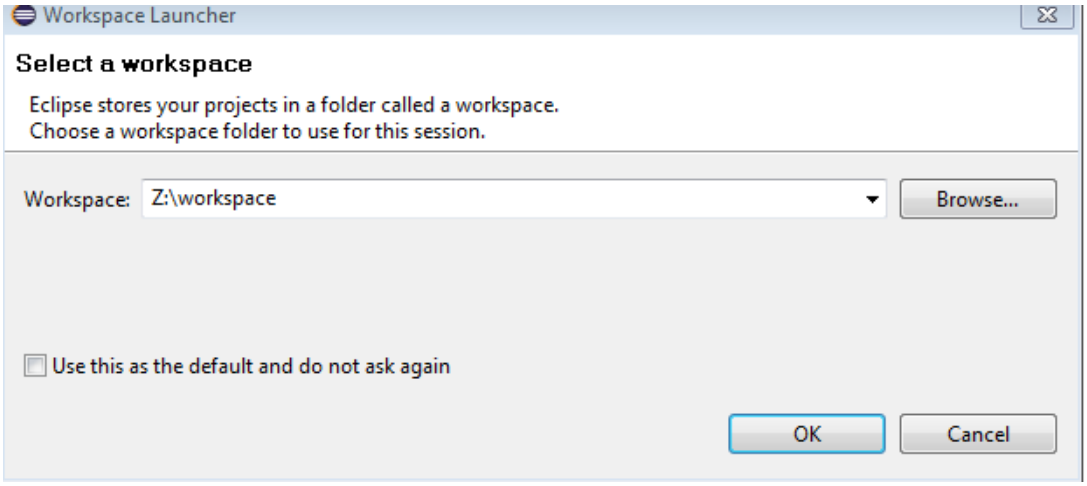
- You can directly access it on the lab machines (make sure you choose "eclipse java")

- You can access it through the virtual desktop



NOTE: If you receive an error about missing links when you launch Eclipse, give the system a minute and then try again. It seems sometimes there is a delay at the moment when you first log in to the system.

Workspace. Warning. For workspace, by default it will store in C:\... This seems to work fine for Eclipse, BUT note due to restriction in the virtual desktop you will ***not*** be able to see/access the files in Explorer (!). Therefore, change it to your personal network store (e.g. z:\, as below). This seems to work fine, and means you can access the files ok.



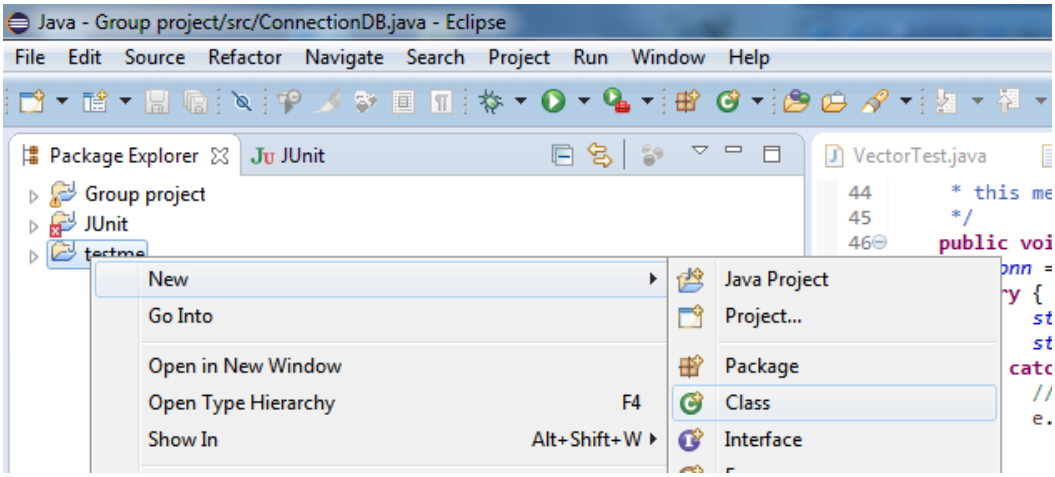
Create a New Java Project

- File→New→Java Project
- Call it any name you like, select version 1.8.
- It will by default save the project in your workspace folder.

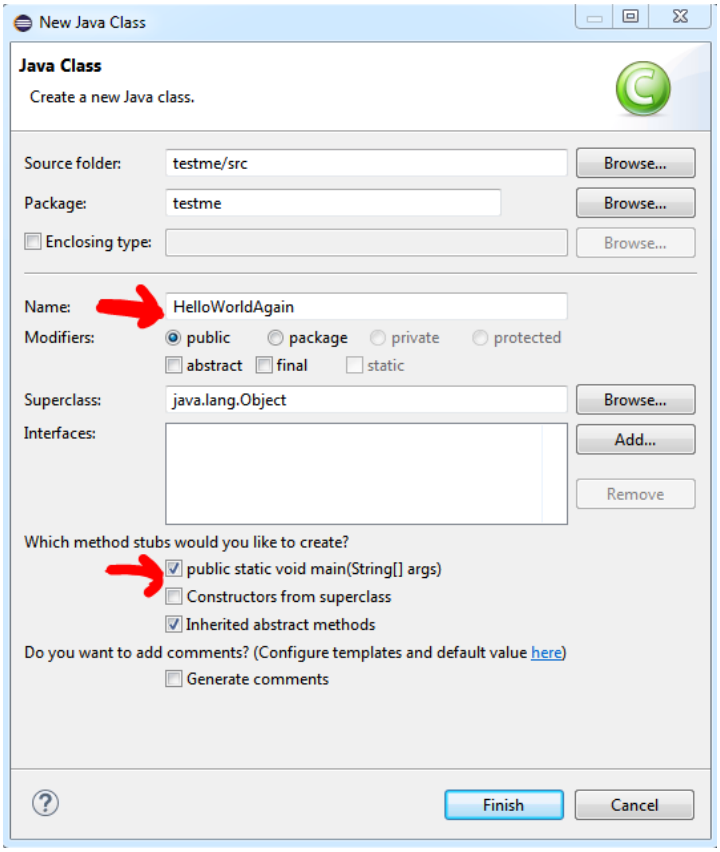
If you get a **Welcome screen**, you should click the icon at the top right called Workbench to move to the main view.

Your project should then appear in the **Package Explorer**. Note we haven't created any actual source files in it yet.

- Add a source file. Right click you project and select new→Class



In the dialog box, add a name, and select to create a method skeleton for main:



Press Finish.

Now, add code to print Hello World:

*HelloWorldAgain.java

```
1  package testme;
2
3  public class HelloWorldAgain {
4
5      public static void main(String[] args) {
6          System.out.println("Hello world!");
7
8      }
9
10 }
11
```

When you run this (e.g. press the  button) your text should appear in the Console window.

OK – you should now be back up to speed on using Eclipse, this time in the Virtual Desktop. You should see that everything should run just as before.

PART 2: AN OBJECT-ORIENTED EXAMPLE

Create a new project. Download the code Bicycle.java from Moodle and import it into your project.

To do this, drag and drop the file in the "Src" folder. Behind the scenes, it will by default copy the file into you project workspace.

- If you want to check where the file it, right click it in the Package Explorer and select Properties.
- You may need to manually create the 'bike' package in Eclipse using  , and drag the Bicycle file into the new package

The project should now compile, but there is no Main so nothing will appear to happen.

```
package bike;
public class Bicycle {
    private int gear;
    private int speed;

    public void setGear(int newValue) {
        gear = newValue;
    }
    public void applyBrake(int decrement) {
        speed -= decrement;
    }
   public void speedUp(int increment) {
        speed += increment;
    }
}
```

Tasks:

- Write a Main class and 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 to print out the current speed, gear and light state.

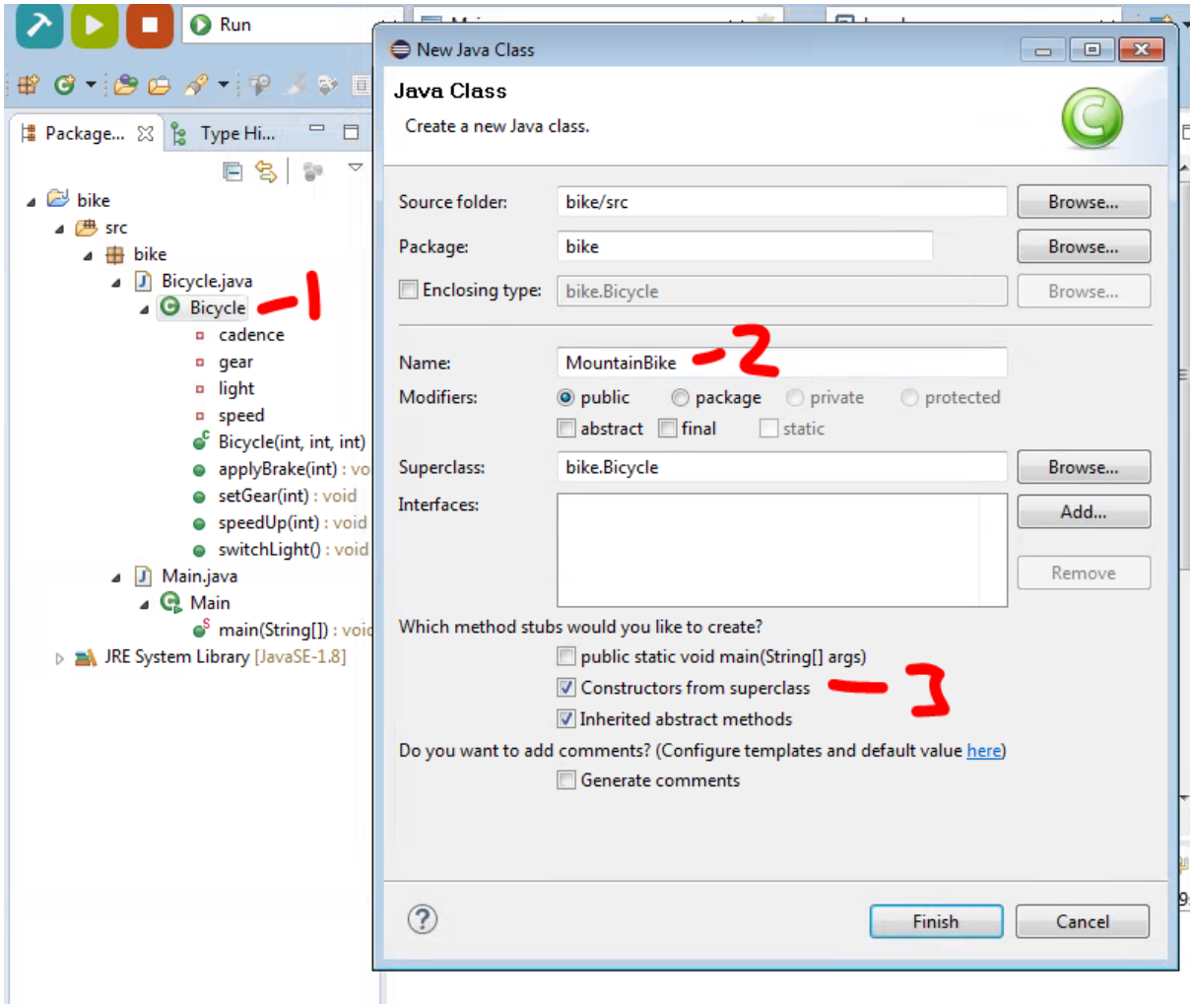
Let's suppose we also want a class for a mountain bike, which has specific features associated with it. It makes sense to create a subclass of a Bicycle i.e. inherit from it.

In Eclipse this is in the package explorer, right click your Bicycle class (1, below) and select New->Class. Change the name (2) and click the box (3) 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.



Advanced 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

Here 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 "DiamondHunter.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".

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. "*").

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. "*").

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. "*").

Challenge: 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.

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.

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

Highlight the method, 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.

Challenge: 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.

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.