LAB 01: ECLIPSE, INTELLIJ, AND "HELLO JAVA 10 WORLD"

Aims:

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

PART 1: 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. Write a line of code to change the speed of the bicycle you just created.
- Write a suitable constructor for the Bicycle class, so that you can set the initial gear, light status and speed values.
- Add a method called "switchLight" to the Bicycle class that turns the light on if it is off, and vice versa
- Add a method "currentState" to the Bicycle class to print out the current speed, gear and light state in the console
- Test its functionality by adding some code to the BikeApp class

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, 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. [In IntelliJ it seems to be more manual labour (at least the way I did it). I created a "MountainBike" class, added "… extends Bicycle" manually in the class specification, went with the mouse over the appearing red line and pressed {Ctrl + Shift + Enter} on the keyboard, which automatically created the required constructor for me]

Hey presto, your subclass is created.

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. Initialize the gear, speed and light status by calling the superclass constructor.
- Add a method called isFullSuspension which returns *true* only if the bike has both front and rear suspension.

- In the BikeApp class create objects for two different mountain bikes, one with full suspension, one without. Print out the current state of the bikes.

- In addition to the above, print out statement on the type of bicycle and whether or not with full suspension. E.g. "MountainBike; isFullSuspension: true"

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 "DiamondHunterEclipse.zip" from Moodle, unzip it, and import the resulting folder into Eclipse. For this, 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 root directory} and provide the location where you stored the source code. 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}. [In IntelliJ you might have to define the Project SDK. Use {File > Project Structure > Project Settings > Project > Project SDK} and choose the one you want to use from the pull-down menu]

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 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. [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/IntelliJ Help to find out more

Both IDEs 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.