LAB 01: INTELLIJ, AND "HELLO COMP2013 WORLD"

Content:

1. Preparation
2. Get familiar with IntelliJ
3. Implement a simple object-oriented example
4. Working with existing code

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

NB: We do not expect everyone to solve all the "**Challenges**" listed below immediately. If you could not solve them now, you might want to consider coming back to them later in the module, once you have a bit more experience with the IDE and Java.

Please note that we have used a Windows environment to prepare this task sheet. If you are using a Linux or Mac environment and get stuck, please ask us, and we will try to help.

PART 1: PREPARATION

To maintain software, you need to be familiar with modern Integrated Development Environments (IDEs). As we said in the first lecture, you should use the IntelliJ IDEA IDE[1] in this module. As for the programming language, everything from Java 12 onwards is suitable, but we recommend that you use on of the latest Java Development Kits.

Here are some tips for installing the JDK and IDE on your private machine:

- Java JDK (not JRE)
  - Download open-jdk GA release (https://jdk.java.net/19/)
    - Difference between GA and Reference release: https://stackoverflow.com/questions/50316397/difference-between-openjdk-10-and-java-se-10-reference-implementation
  - System environment > Set this up in System Variables (bottom) rather than User Variables (top) if it is your own machine
    - Windows
      - JAVA_HOME > Path to JDK folder
      - Path > add "%JAVA_HOME%\bin"
  - Check that java installation works, using "java -version"
    - If it shows Java 8, you can use "where java" to check out why
      - Remove "C:\Program Files (x86)\Common Files\Oracle\Java\javapath\java.exe" or move the java path variable ("%JAVA_HOME%\bin") to the top of the list
- IntelliJ IDEA IDE
  - Download IntelliJ Community Edition: https://www.jetbrains.com/idea/download/
  - Run IntelliJ installer > Choose: 64 bit launcher + associate .java with IntelliJ + update context menu (if you want)
  - Install and then run IntelliJ > need to agree to terms > IntelliJ should open

If you encounter any issues when installing the IDE and JDK on your local machine, let us know, and we are trying to help you to fix those issues.

[1] We will also try to provide some support for Eclipse and NetBeans users, but the official IDE for this module is IntelliJ IDEA

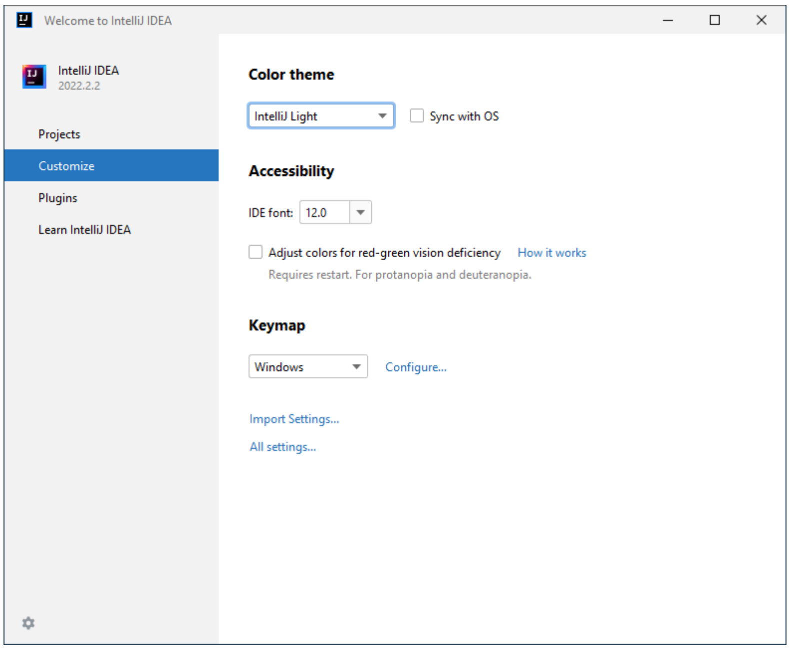
PART 2: GET FAMILIAR WITH INTELLIJ

The following information will guide you through programming a simple "Hello COMP2013 World" program in IntelliJ. Feel free to skip this if you know how to set up a new project in IntelliJ, using Java 12+ and write "Hello COMP2013 World" to the console.

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 should have one additional file (module-info.java) in the new projects we create.

For the following demonstration, I will use IntelliJ IDEA 2022.2.2, which is the version installed on my machine. The university machines and the Virtual Desktop (VD) have IntelliJ IDEA 2022.1.3 installed, but the difference between the two versions is not drastic.

- Start IntelliJ. If you are using the VD, the easiest way to start IntelliJ IDEA is to type "intellij" into the windows search box.
- In the "Welcome to IntelliJ IDEA" screen choose {Customize > All settings ...} and in the coming screen {System Settings > Default project directory} and choose your workspace directory, where you want to save your projects in. Always make sure you are using your personal network storage (I used "C:\Workspace\Java\IntelliJ", but yours will be different).



Color theme

IntelliJ Light ☐ Sync with OS

Accessibility

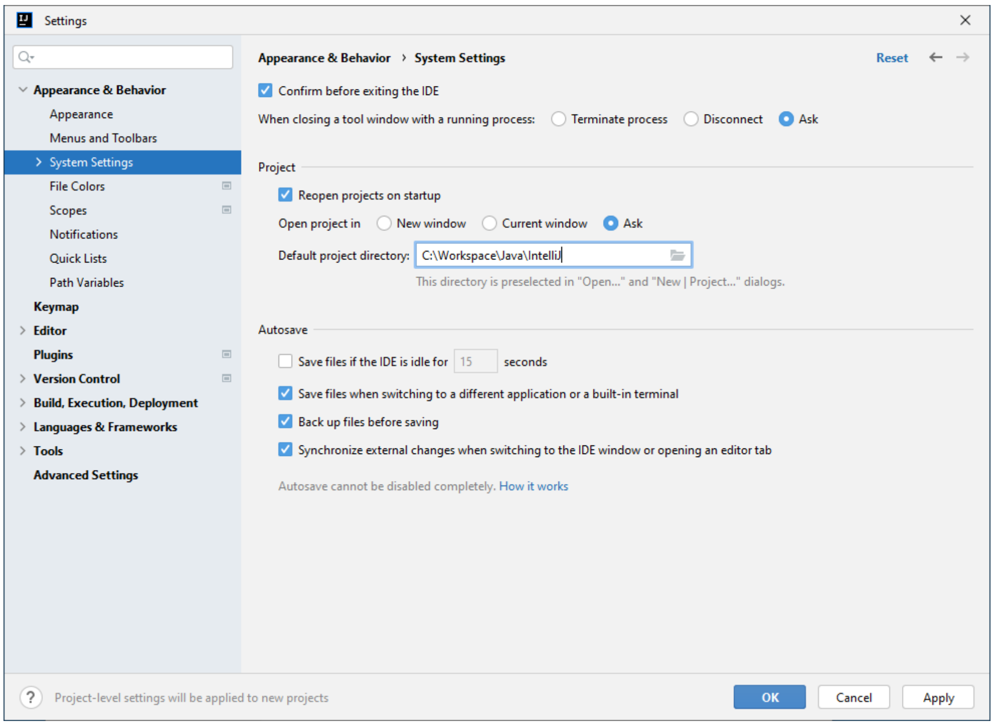
IDE font: 12.0

☐ Adjust colors for red-green vision deficiency

Requires restart. For protanopia and deuteranopia.

Keymap

Windows



Appearance & Behavior › System Settings

[✓] Confirm before exiting the IDE

When closing a tool window with a running process: Terminate process Disconnect [✓ Ask]

Project

[✓] Reopen projects on startup

Open project in New window Current window [✓ Ask]

Default project directory: C:\Workspace\Java\IntelliJ

This directory is preselected in "Open..." and "New | Project..." dialogs.

Autosave

[ ] Save files if the IDE is idle for 15 seconds

[✓] Save files when switching to a different application or a built-in terminal

[✓] Back up files before saving

[✓] Synchronize external changes when switching to the IDE window or opening an editor tab

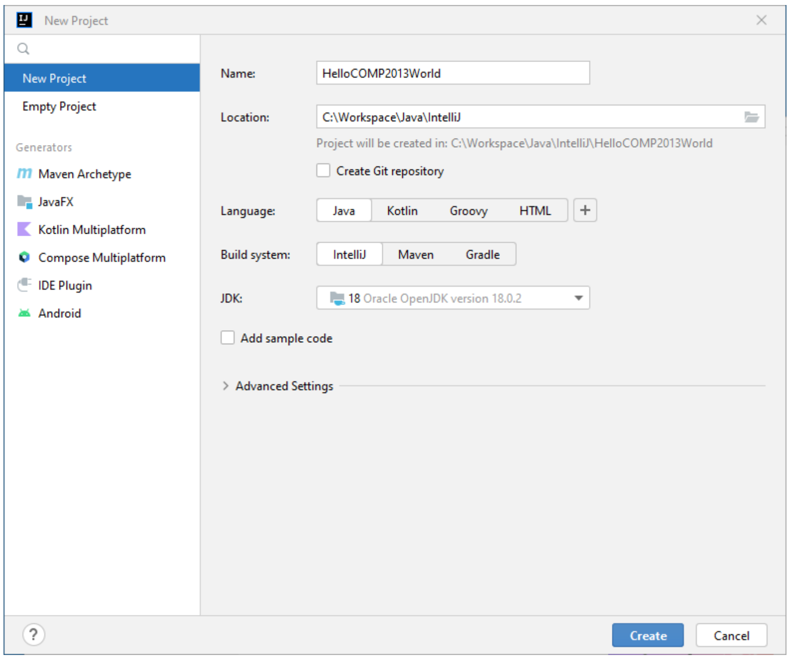
Autosave cannot be disabled completely. How it works

Project-level settings will be applied to new projects

- Click {OK} to get back to the "Welcome to IntelliJ IDEA" screen:

Create a new Java project

- Click on {Projects > New Project}, and provide a {Name} "HelloCOMP2013World".



Name: HelloCOMP2013World

Location: C:\Workspace\Java\IntelliJ

Project will be created in: C:\Workspace\Java\IntelliJ\HelloCOMP2013World

[ ] Create Git repository

Language: [✓ Java] Kotlin Groovy HTML

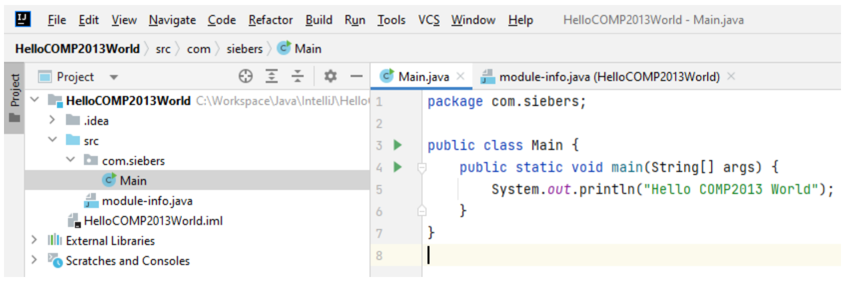
Build system: [✓ IntelliJ] Maven Gradle

JDK: 18 Oracle OpenJDK version 18.0.2

[ ] Add sample code

- Click {Create}
- Now right click on the src folder and create a package via {New > Package}. Name it "com.uon" (or "com.yourname"). We will use "com.siebers". This will be the base package. Now right click on the base package and create a class via {New > Java Class > Class}. Name it "Main".
- An alternative way to generate classes inside packages is to right click on the src folder and create a class via {New > Java Class > Class} and provide relevant package names as a prefix. In this example above that would have been "com.siebers.Main".

- To generate the "main" method inside the "Main" class body just type "main" between the {} and press the "TAB" key. To generate a console printout just type "sout" between the new {} and press the "TAB" key. Finally add a String that contains what should be printed into the console.

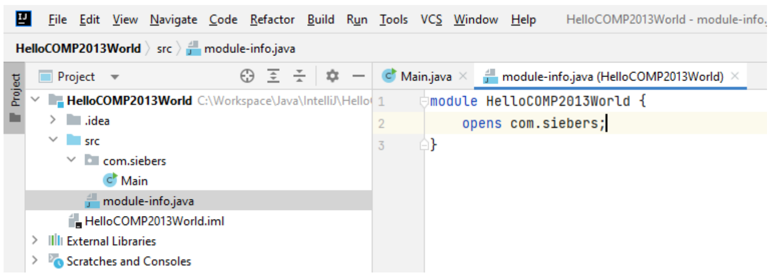

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
HelloCOMP2013World - Main.java
HelloCOMP2013World src com siebers Main
Project
Main.java
module-info.java (HelloCOMP2013World)
HelloCOMP2013World C:\Workspace\Java\IntelliJ\Hello
.idea
src
com.siebers
Main
module-info.java
HelloCOMP2013World.iml
External Libraries
Scratches and Consoles

```java
package com.siebers;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello COMP2013 World");
    }
}
```

Create module-info.java file

- Next, we need to generate a file for storing information about project modularity. Right click on the base package and choose {New > module-info.java}
- Open the module-info.java file, and add "opens com.siebers;". We will learn in a later lecture what this means, but if you are keen to know now, you can have a look at the following url: https://www.oracle.com/uk/corporate/features/understanding-java-9-modules.html
- Now you will need to remove all other main classes from outfile outside the com, otherwise you might get an error.


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
HelloCOMP2013World - module-info.
HelloCOMP2013World src module-info.java
Project
Main.java
module-info.java (HelloCOMP2013World)
HelloCOMP2013World C:\Workspace\Java\IntelliJ\Hello
.idea
src
com.siebers
Main
module-info.java
HelloCOMP2013World.iml
External Libraries
Scratches and Consoles

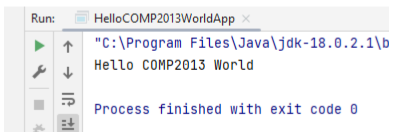
```java
module HelloCOMP2013World {
    opens com.siebers;
}
```

Rename the Main class

- If we wanted to rename the Main class to something more meaningful, we could do this as follows. Right click on the class in the project window, choose {Refactor > Rename...} and change the name to "HelloCOMP2013WorldApp". Then click {Refactor}

Execute the program

- There are several ways to execute the program. You can click one of the two green arrows in the code window border. You can also go to the class containing the main() method, right click it, and choose {Run HelloCOMP2013W....main()}. Another alternative is to run the program by clicking on the green arrow in the top right of the Editor window. Finally, you can run it by choosing {Run > Run HelloCOMP2013WorldApp.java}. Use one of these options to execute the program.
- Your text should then appear in the console window (at the bottom of the IntelliJ IDE). If you do not see the console window, click the Run button at the bottom left.



```
"C:\Program Files\Java\jdk-18.0.2.1\b
Hello COMP2013 World

Process finished with exit code 0
```

PART 3: IMPLEMENT A SIMPLE OBJECT-ORIENTED EXAMPLE

Create a new Java project, name it BikeProject, and add a new package com.theBestBikeShop to the src folder. Create a module-info.java file and add relevant information (opens 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, and then click {Refactor} in the appearing popup.

- If you want to check, where the Bicycle.java file is located, right click the file in the Project Explorer and select {Copy Path > Reference...}. In the popup, you can see the absolute path.

You can compile the project using {Build > Build Project} 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. A neat trick to create the main() method is to type main and use {TAB} for auto-complete.
- Write a line of code to change the speed of the bicycle you just created.
- Write a constructor for the Bicycle class, so that you can set the initial gear and speed values. This will create an error. Go to the BikeApp class and change it, so that the new constructor is used for generating the Bicycle object
- Add a method called switchLightStatus 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 the functionality of both added methods 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 class, i.e. inherit from it.

In the Project Explorer, right click your com.theBestBikeShop package and select {New > Java Class}. Name the new class MountainBike and add "... extends Bicycle" to the class signature. Then hoover with the mouse over the appearing red line and pressed {Alt + Shift + Enter} on the keyboard, which automatically created the required constructor.

Hey presto, your subclass is created.

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

- Modify the constructor of the MountainBike class 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. In the BikeApp class, create objects for two different mountain bikes, one with full suspension, and one without.
- Override the currentState() method in the Bicycle class from within the MountainBike class, providing gear, speed, light status, and suspension state information. Test the functionality of the added method by adding some code to the BikeApp class.

Challenge: Automatically generate getters, setters, and constructors.

The IntelliJ IDE menu {Code} contains a lot of functionality for automating processes such as writing getters and setters and producing constructors. Delete all existing getters, setters, and constructors within your bike source code and use {Code > Generate ...} to get back to how the code looked before.

PART 4: WORKING WITH EXISTING CODE

In this part we will download and explore a larger existing codebase. To do this we will learn some tips for navigating project code. Please note: The code was written for Eclipse, but will work similarly in the IntelliJ IDE.

Challenge: Try to work out how it works in IntelliJ. Some advice is provided in orange.

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 project zip file DiamondHunter.zip from Moodle, unzip it, and import the resulting folder into IntelliJ. In IntelliJ IDEA, just open the project. 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] [If you want a real challenge, download DiamondHunterSrc.zip instead and set up your own project in Intellij IDEA.

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 In IntelliJ use {Navigate > Class...} the class GameState that is listed. This class will then appear in the editor window.

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

Use {View > Tool Window > Structure}

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

In IntelliJ use {Navigate > Type Hierarchy}

Task:

- Try out some of the other options you find in the Navigate menu or when right clicking in the Editor to open a context specific menu.

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

Double click method name and choose {Edit > Find > Show Usages}

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

In IntelliJ double click method name and choose {Navigate> Call Hierarchy}

Task:

- Try out some of the other options you find in the IntelliJ menu.

The 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 later lectures / labs.

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