Object Oriented Programming
(236703)

IDEs + git Workshop

Oren Afek

oren.afek@cs.technion.ac.il

Technion – Israel Institute of Technology

April 2017

Agenda

This workshop will help you To :

- Get started with the most advanced standard (*De Facto.*) programming tools today.
- Save a lot of time while working.
- Acquire a much more productive and organized way to develop.
- Learn how to have fun while programming!

JetBrains

Awesome Powerful IDEs

- The most emerging company for development tools and solutions
- Developed a tool for almost every programming need:

  Java, C/C++, Python, C#, Swift (iOS, OSX) App Development, Ruby, PHP, JavaScript and more….

IntelliJ
Java

Rider
C#(.NET)

CLion
C/C++

PyCharm
Python

AppCode
Swift

JetBrains

Awesome Powerful IDEs

- All based on the basic original IntelliJ framework.
- **Everything looks and feels the same!**

IntelliJ
Java

Rider
C#(.NET)

CLion
C/C++

PyCharm
Python

AppCode
Swift

JetBrains

Even the mighty Google got it:

- At first, Android development was offered by using an Eclipse plugin
- Google created a brand new IDE for Android development, **powered by IntelliJ platform**.
- So even Android is actually developed using JetBrains' (based) product

Android Studio
Android Development

JetBrains

Other Development tools

- Other than IDEs, JetBrains also offer tons of innovative development tools such as:
- TeamCity – Continues Integration.
- YouTrack – Issue tracking and project management tools.
- UpSource – Code review and repository browsing.

JetBrains

There is also a downside ☹

- Most of **JetBrains' tools aren't free** and a bit expensive.
- Even though, there are some **Community Editions** for two of the IDEs:

JetBrains

And There is an extraordinary Bright Side ☺

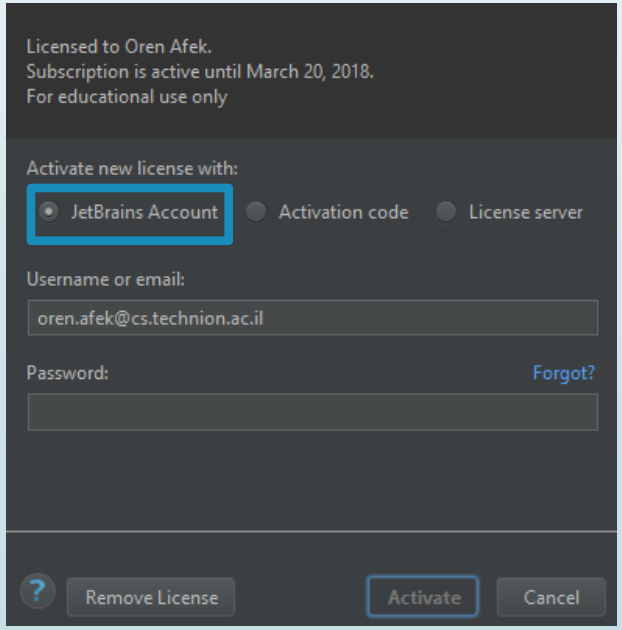
- JetBrains offers all of its IDEs and tools **free for students**
- All you have to do, is register using an academic email like @**campus.technion.ac.il** or @**cs.technion.ac.il** and you're good to go for **a year**.

- You can always **extend for an extra year.** as long you have access to your academic email.

JetBrains

Getting Started

- Apply for a student account: https://www.jetbrains.com/student/
- Sign in: https://account.jetbrains.com/login
- Download any IDE or tool you desire **for free**!
  - Or use the mighty **JetBrains Toolbox** to manage everything in one place
- Upon first use in the IDE, use your credentials to unlock its full features



IntelliJ

Amazingly smart Java IDE

- Getting Started
- Basic Flow
  - Opening a project
  - Writing some code
  - Compiling
  - Running
- Project Structure
- Unit Testing
  - JUnit

IntelliJ

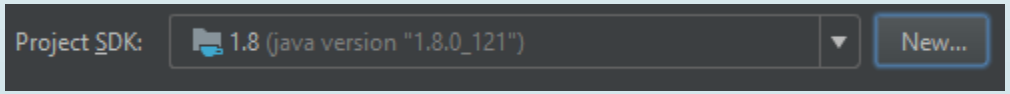
Amazingly smart Java IDE

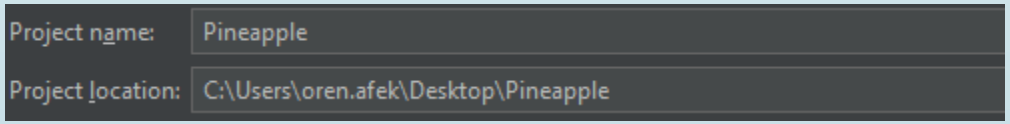
- Intentions (Alt + Enter) Wizard
- Auto Generation
- Snippets
- Debugging
  - Debugger
  - Watches
  - Evaluations in runtime
- Keymap

IntelliJ

Getting Started

- After Installing IntelliJ (and providing the license key),
- Install Java JDK (8) (here's a guide)
- Open a new Project:

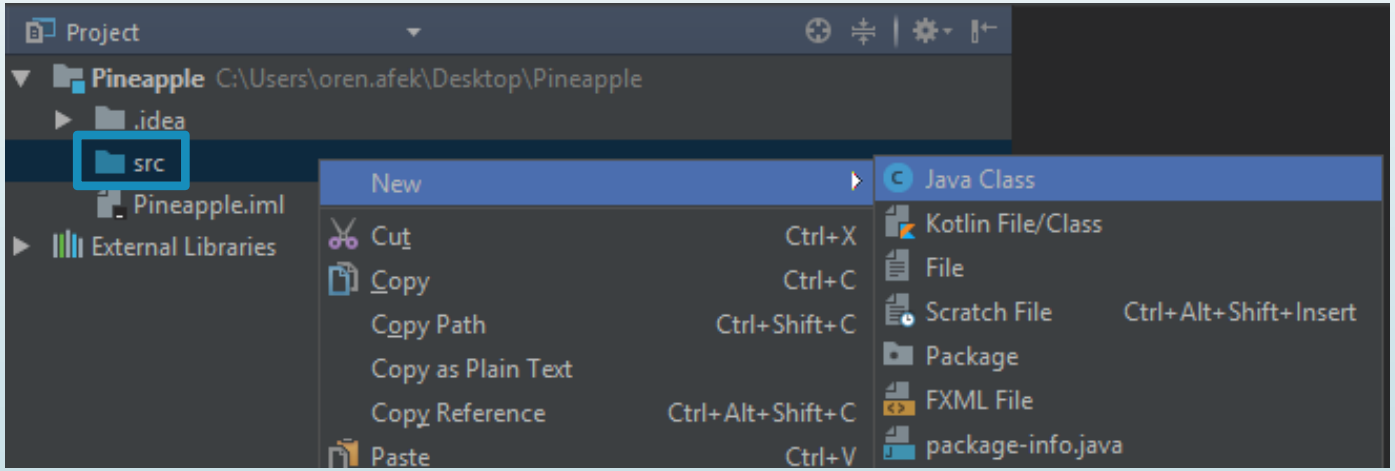




IntelliJ

Basic Flow

- Create a new Java Class:



IntelliJ

Basic Flow

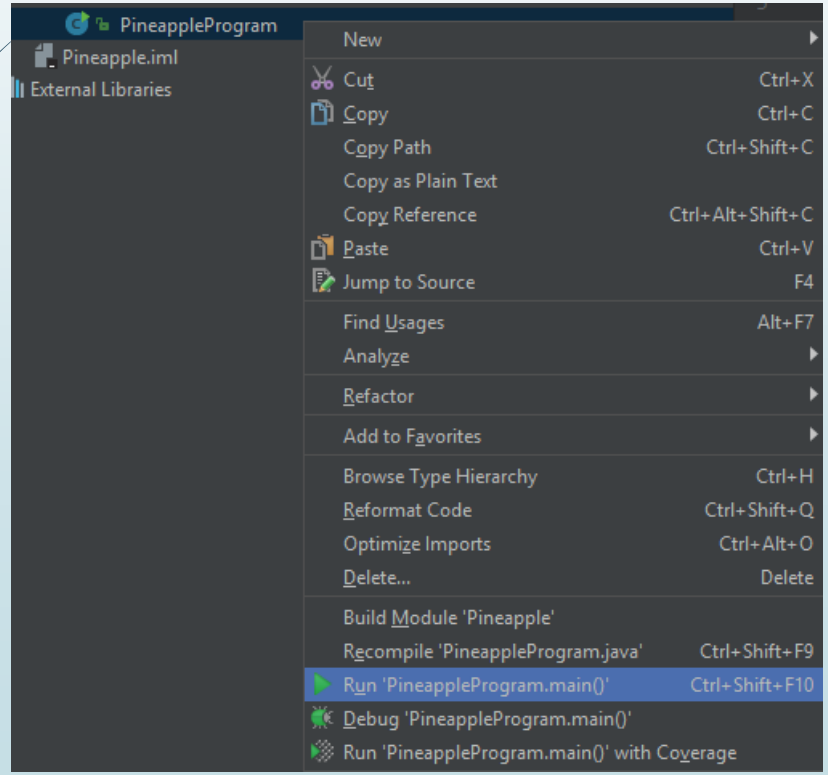
- Write Some Code

```java
/**
 * @author Oren Afek
 * @since 4/10/2017.
 */
public class PineappleProgram {


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

```

IntelliJ

Basic Flow

- (compile and) Run your program



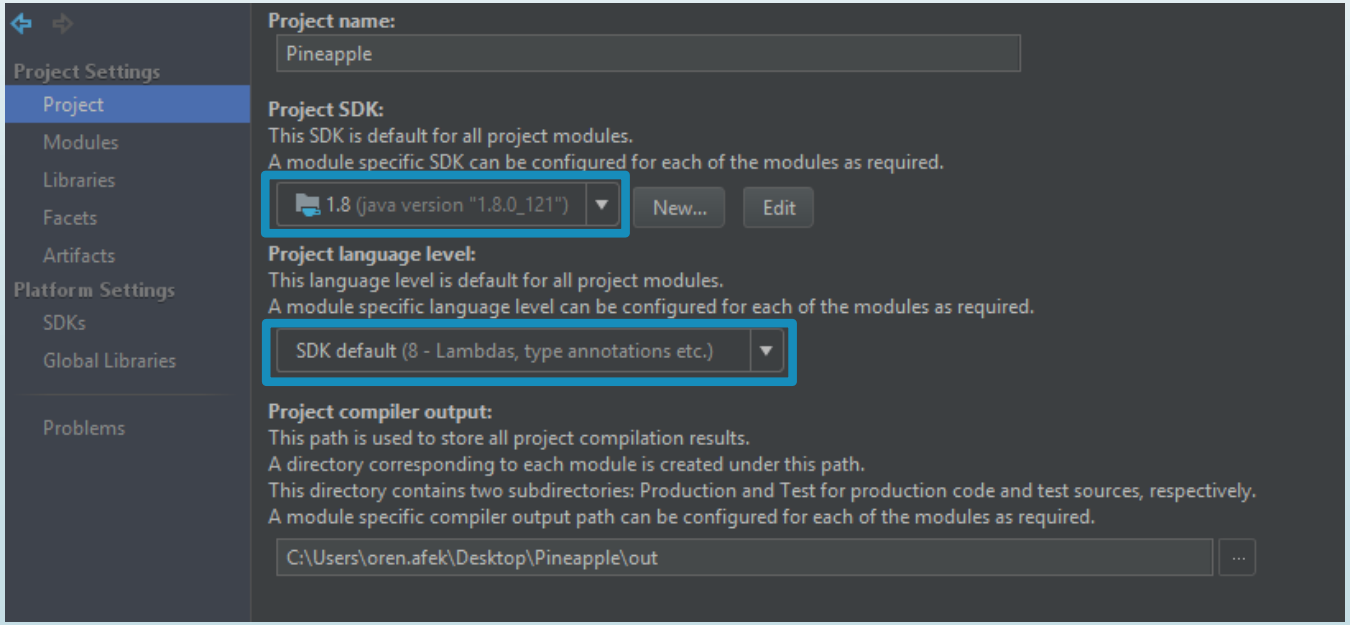

```java
/**
 * @author Oren Afek
 * @since 4/10/2017.
 */
public class PineappleProgram {


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

Ctrl + Shift + F10

IntelliJ

Project Structure

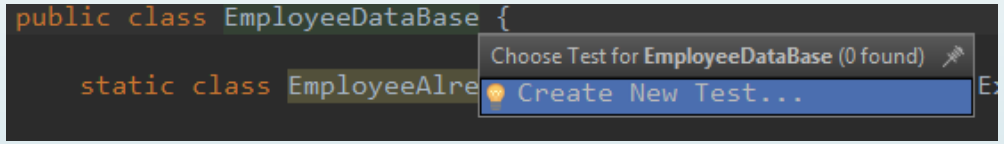
- Define project's SDK
- Define Java's language Level **(Java 8 recommended)**
- Add dependencies of external libraries

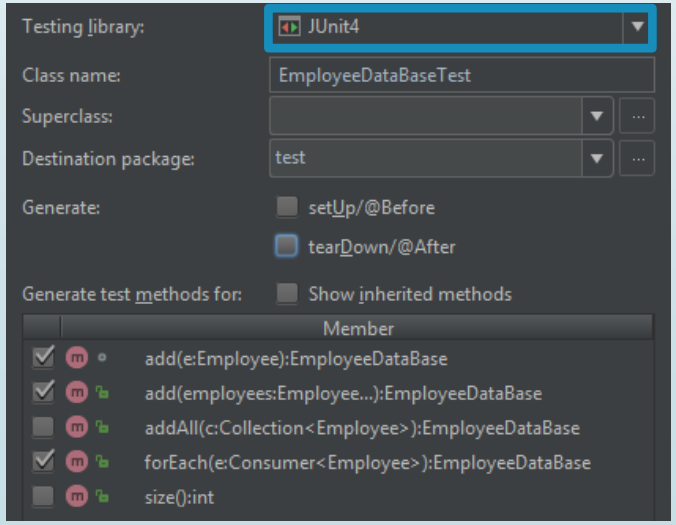


IntelliJ

Unit Testing

- Create a new Test:
  - Select **Navigate** –> **Test** or Hit **Ctrl + Shift + T** on a Class Name





  - Select **JUnit4** library
  - Mark methods to be tested.

IntelliJ

Unit Testing

- **@BeforeClass**

  Annotated **`public static void`** method will **run once**, before **any** of the tests run.

  Use this to allocate test objects and heavy connections like remote server connections etc.

```
public class EmployeeDataBaseTest {

    public static EmployeeDataBase $;
    private static int initialSize;
    private static Employee homer, marge, lisa;

    @BeforeClass
    public static void init() {

        homer = new Employee("33", "Homer", "Simpson");
        marge = new Employee("34", "Marge", "Simpson");
        lisa = new Employee("35", "Lisa", "Simpson");

        $.add(homer, marge, lisa);
        initialSize = $.size();
    }
```

IntelliJ

Unit Testing

- **@AfterClass**

  Annotated **`public static void`** `method` will run **once**, after **all** tests finished to run.

```
@AfterClass
public static void done(){
    dumpBackToSpringfield($);
}

private static void dumpBackToSpringfield(EmployeeDataBase db) {
    System.out.println("Back to Springfield all of you: " + db);
}
```


JUnit 5

IntelliJ

Unit Testing

- **@Test**

  Each test method should be **`public void`** and annotated with **@Test**.

```
@Test
public void testAddAlreadyInDataBase() {
    assertEquals(initialSize, $.size());
    try {
        $.add(new Employee("33", "AA", "BB"));
        fail();

    } catch (EmployeeAlreadyInDataBaseException ignored) {/**/}

    assertEquals(initialSize, $.size());
}
```

IntelliJ

Unit Testing

- **@Before**

  Annotated **`public void`** method will run **once**, before **each test**. Use this to make the tests distinctive.

```
@Before
public void resetDB(){
    $ = new EmployeeDataBase($_original);
}
```

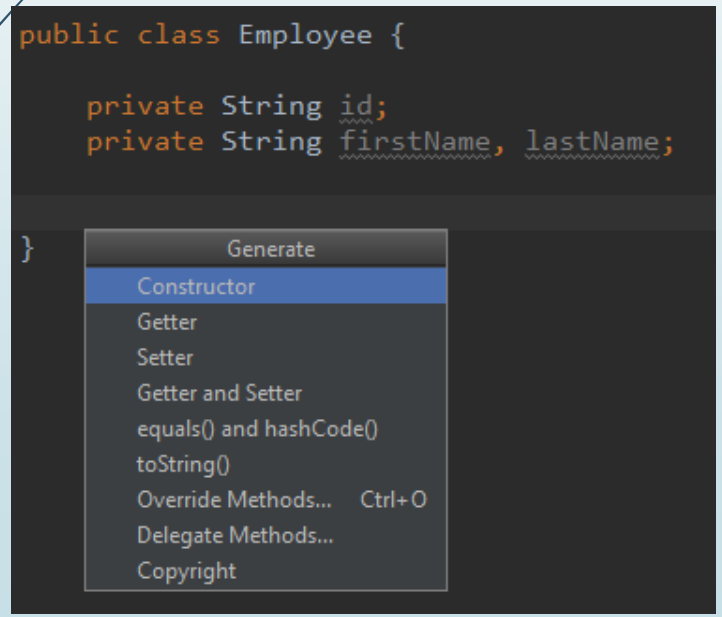
- **@After**

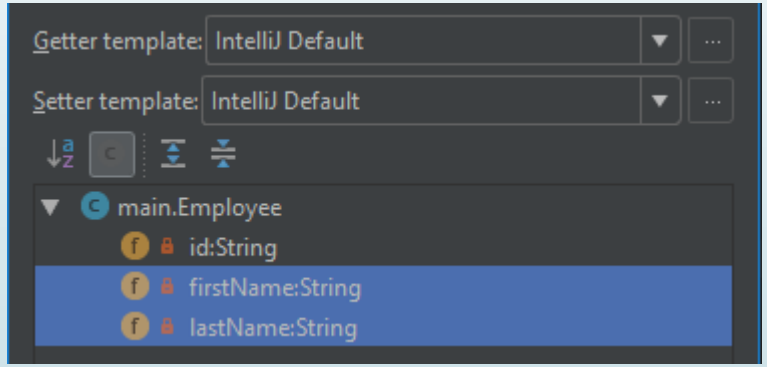
  Annotated **`public void`** method will run **once**, after **each test** (in between). Use this to make conclusion after a test.

IntelliJ

Auto Generation – Let IntelliJ write for you!

- Hit **Alt + Insert** to let IntelliJ generate boiler plate code for you. For example, getters/setters, constructors, headers of interface methods etc.





IntelliJ

Auto Generation – Let IntelliJ write for you!

- Hit **Alt + Insert** to let IntelliJ generate boiler plate code for you. For example, getters/setters, constructors, headers of interface methods etc.

```
public class Employee {

    private String id;
    private String firstName, lastName;

    public String getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
```

IntelliJ

Intentions - IntelliJ helps you in time of need!

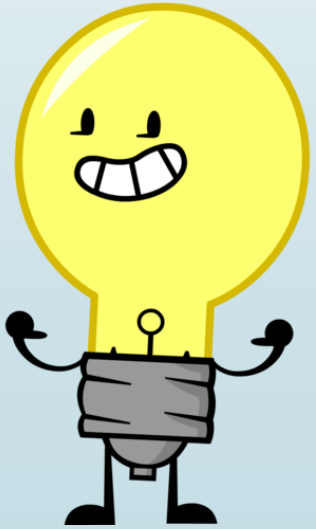
- For example, in the use of classes that haven't been imported, or methods that haven't been written etc.
- Even if you don't need it, IntelliJ will help you to improve your code to meet a safer more "Java standard" code.
- Hit **Alt + Enter** whenever IntelliJ introduces you to a solution or an enhancement to your code.

```
public EmployeeDataBase printAll(){
    System.out.println("Employees : ");
    employees.forEach(new Consumer<Employee>() {
        @Override
        public void accept(Employee
```

Replace with lambda
Change class type parameter

```
public EmployeeDataBase printAll(){
    System.out.println("Employees : ");
    employees.forEach(employee -> System.out.println(employee));
    return this;
}
```

Replace lambda with method reference
Go to super method Ctrl+U



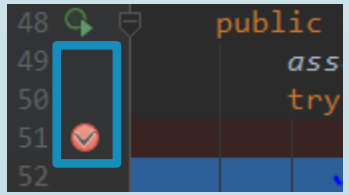
IntelliJ

Debugging – Super powerful debugger

- JetBrains's debugger offers great runtime information :
  - Override `toString()` to get an in place overview of the objects

```
@Test
public void testAddAlreadyInDataBase() {
    assertEquals(initialSize, $.size());
    try {
        Employee bart = new Employee("33", "Bart", "Simpson");  bart: "Bart Simpson (33)"
        $.add(bart);  $: "[Marge Simpson (34), Lisa Simpson (35), Homer Simpson (33)]"  bart: "Bart Simpson (33)"
        fail();
```

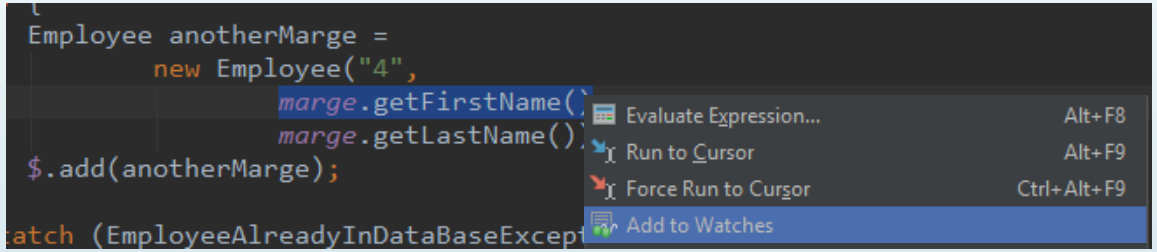

  - Toggle breakpoints by touching the blank area between the line no. and the workspace

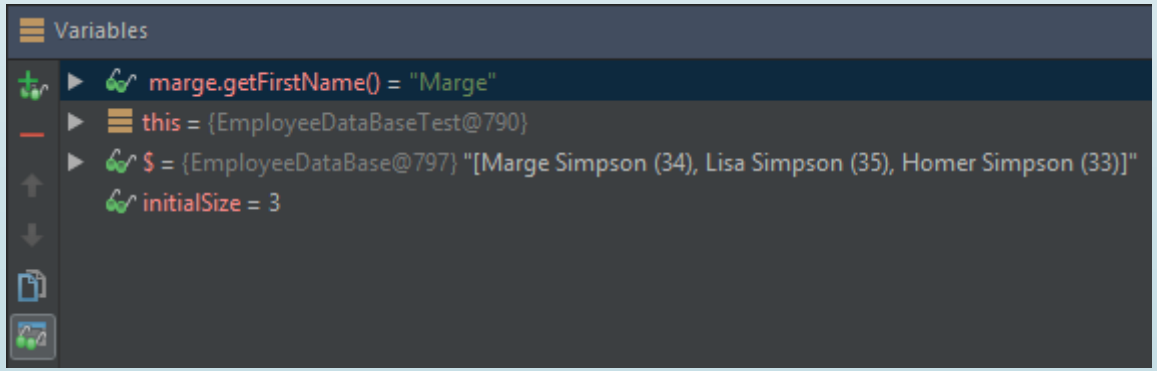


IntelliJ

Debugging – Super powerful debugger

- JetBrains's debugger offers great runtime information :
  - Add intermediate expressions as watches:





IntelliJ

Debugging – Super powerful debugger

- Evaluate code snippets in runtime to see the future (maybe another one)

```
Employee anotherMarge =
        new Employee("4",
                marge.getFirstName(),
                marge.getLastName(),
$.add(anotherMarge);

catch (EmployeeAlreadyInDataBaseExcept
```

Evaluate Expression... Alt+F8
Run to Cursor Alt+F9
Force Run to Cursor Ctrl+Alt+F9
Add to Watches

```
marge.getFirstName().replaceAll("M","L")
```

Use Ctrl+Shift+Enter to add to Watches

Result:
result = "Large"
value = {char[5]@1035}
hash = 0

IntelliJ

KeyMap

- Some Important Keyboard Shortcuts:
  - **Alt** + **Enter** – Intentions
  - **Ctrl** + **D** – Duplicate
  - **Alt** + **Insert** – Auto Generate
  - **Ctrl** + **Alt** + **L** – Auto Reformatting
  - **Shift** + **Shift** – Search Everywhere
  - **Ctrl** + **Shift** + **N** – Search for a File
  - **Ctrl** + **O** / **Ctrl** + **I** – Methods to Override / Implement
  - **Ctrl** + **Shift** + **F10** – Run
  - **Ctrl** + **F9** – Debug
  - **Ctrl** + **Shift** + **–** – Collapse All
  - **Ctrl** + **Shift** + **T** – Create a Test
  - **Ctrl** + **P** – Show expected parameters

IntelliJ

KeyMap

- Full Key map Scheme

  https://resources.jetbrains.com/storage/products/intellij-idea/docs/IntelliJIDEA_ReferenceCard.pdf

CLion

Awesome C/C++ IDE

- Getting Started
  - Download MinGW if you're using windows
    - Linux / OSX – no need ☺
  - **Do exactly like you've done in IntelliJ**
    - It's all the same!!

CLion

Awesome C/C++ IDE

- CMake
  - Unified way to mange your builds, includes and dependencies
    - CMake will make the Makefiles for you (if you're in linux / osx, or the Visual Studio build files if needed).
  - Write your code in **CMakeLists.txt** file and let CLion do the rest!

CLion

Awesome C/C++ IDE

- CMake
  - Lets have this simple C++ program:

```
#include <iostream>

using std::cout;
using std::endl;

int main() {
    cout << "I ♥ Watermelon!" << endl;
    return 0;
}
```

  - The appropriate **CMakeLists.txt** file to match:

```
cmake_minimum_required(VERSION 3.7)
project(Hello)
add_executable(Watermelon main.cpp)
```

CMake min. version

Project's name

Create an exec. File named **Watermelon** using the source file **main.cpp**

CLion

Awesome C/C++ IDE

- CMake
  - Use the **set** command to:
    - Set pre-existing values
      - like the C++ Standard:

```
set(CMAKE_CXX_STANDARD 11) # C++11
```

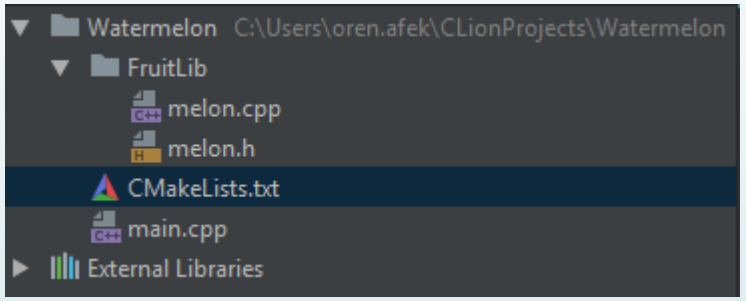
    - Create your own macros for future use:

```
set(Sources main.cpp melon.cpp melon.h)
add_executable(WaterMelon ${Sources})
```

CLion

Awesome C/C++ IDE

- CMake
  - Use the **`include_directories()`** command to **#include** files from other directories




```
#include <iostream>
#include "melon.h"
using std::cout;
using std::endl;

int main() {
    melon m(3);
    cout << m << endl;
    return 0;
}
```

```
cmake_minimum_required(VERSION 3.7)
project(Hello)
set(CMAKE_CXX_STANDARD 11) # C++11
include_directories(FruitLib)
set(Sources main.cpp FruitLib/melon.cpp
FruitLib/melon.h)
add_executable(WaterMelon ${Sources})
```

CLion

Awesome C/C++ IDE

- CMake
  - Let CLion create and maintain **CMakeLists.txt** for you.
    - When creating a new file, CLion will add it as a build dependency automatically.
  - In case of a more sophisticated building procedure?
    - like attaching or building static / dynamic libraries

Use this Cmake's [easy start guide](#).

git

The only sane way to work together

- Git is a **version control system** (VCS) for tracking changes in files and coordinating work on those file among multiple people*
- Git allows you to create "milestones" in time, so you can **always go back** to an earlier version of your code.
- Git provides you a simple way to **synchronize your work** with your collaborators without the fear of overwriting.

* Taken from Wikipedia, the free encyclopedia

git

The only sane way to work together

- Installation
- Basic terminology and commands
- GitHub
- JetBrains's VCS GUI

git

Installation

- Windows / OSX
  - Download git from here
- Linux
  - Use this Code Snippet: `sudo apt-get install git`

- Open the Command Line (Windows) or Terminal (OSX/Linux) to start working.

git

Basic Terminology and Commands

- **Repository** – a root folder for files (or project) managed together in git.
  - Create a git repository by using `git init` inside a folder.
- **Tracking** – inclusion of new files to the repo.
  - Track a new file by using `git add <file name>`
  - Track all of the files in the folder (recursively) `git add .`
- **Commit** – a single point in the git history.
  - Commit new changes to the repo by using

    `git commit -am “<commit msg>”`
- **Branch** – an active line of development.

git

Basic Terminology and Commands

- **Cloning** – copying an existing repository from a remote server, to the local server, to start working on it locally
  - Clone a repository by creating a folder and using (within) `git clone <url>`
- **HEAD** - A named reference to the commit at the tip of a branch.
- **Pushing** – copying all of the missing or updated files, from the local repository to the remote repository.
  - Push your commits by using `git push`
- **Pulling** – copying all of the missing or updated files, from the remote repository to the local repository
  - Pull the latest changes from server by using `git pull`

git

Basic Terminology and Commands

- **Stashing** – discarding (temporarily) all of the changes since the last push
  - When having conflicts, stash your work, to pull the remote changes by using: `git stash` and use `git stash apply` or `git stash pop` to restore them

- **Checkout / reset** – discarding (permanently) all of the changes since the last push
  - Discard changes in a file by using `git checkout HEAD <file name>`
  - Discard changes in all of the repository `git reset --hard HEAD`

- **Status** – stating the repository condition (changes locally or remotely).
  - Show the repository's status by using `git status`

Basic Terminology and Commands

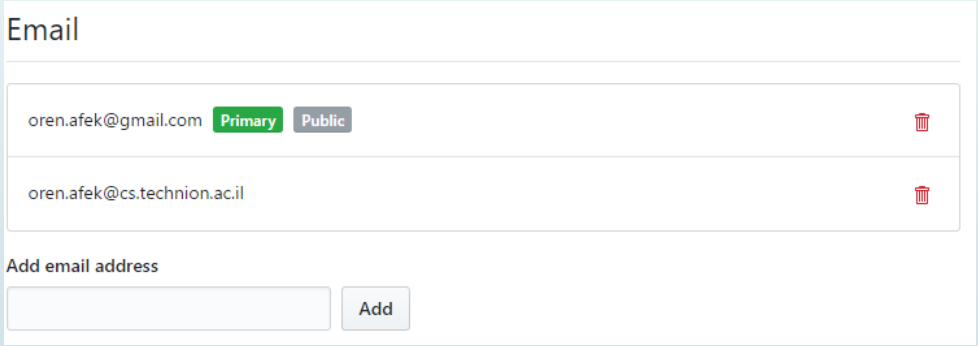
- for other useful git commands, see this git cheet sheet

GitHub – Intuitive way to use git

- GitHub is a web-based Git or version control repository and Internet hosting service
- Store your repositories in GitHub
- Enjoy the easy and organized issue system to help you and your team keep track of your work.

GitHub – Getting Started

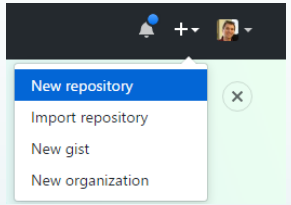
- Create an account in https://github.com/
- Associate an academic email to your account
  - Settings -> Emails -> Add an email

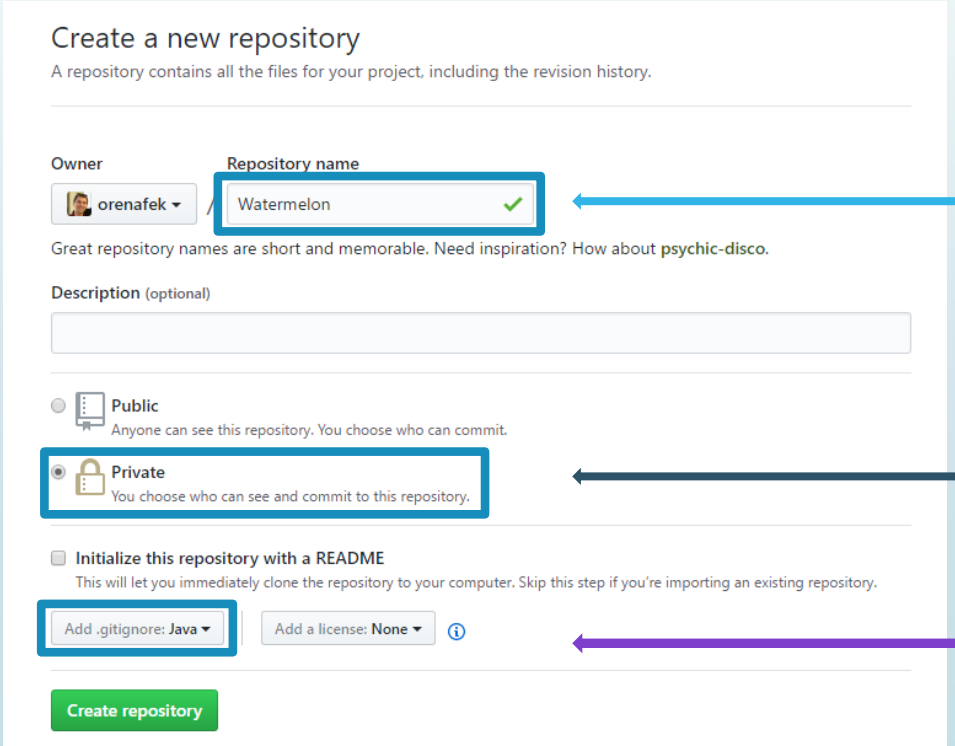


- Apply for the Student Education Pack
  - https://education.github.com/pack

GitHub – Getting Started

- Create a new **private** repository





Repository name

Choose Private!

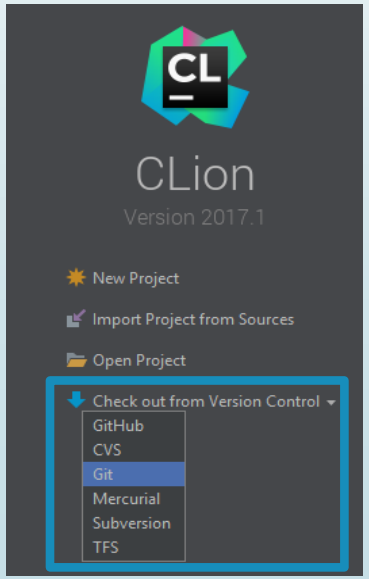
Choose your programming language

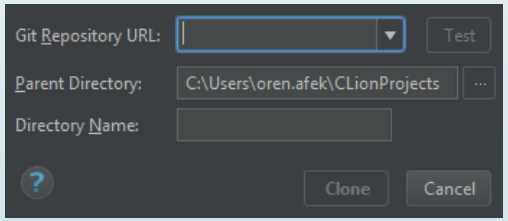
Always use private repositories to store your homework assignments.

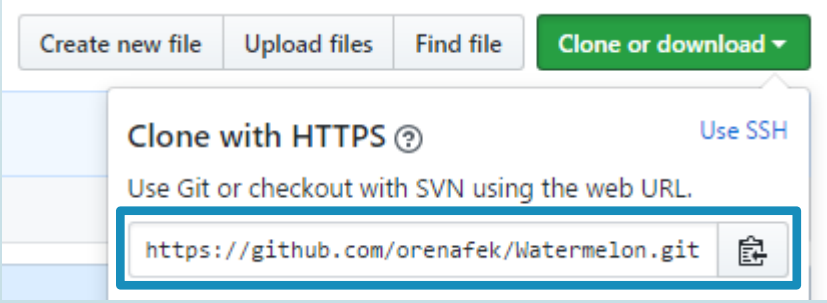
HW stored in public repositories will be graded with an automatic 0

git

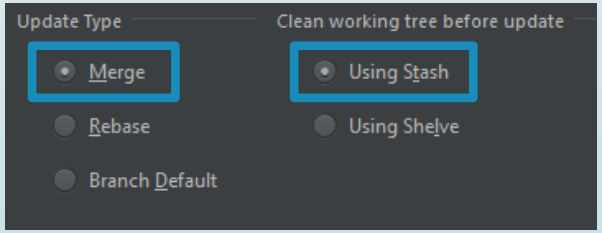
- JetBrains VCS GUI
  - In all of JetBrains's IDEs there is an integrated VCS GUI system.
  - Clone a repository:





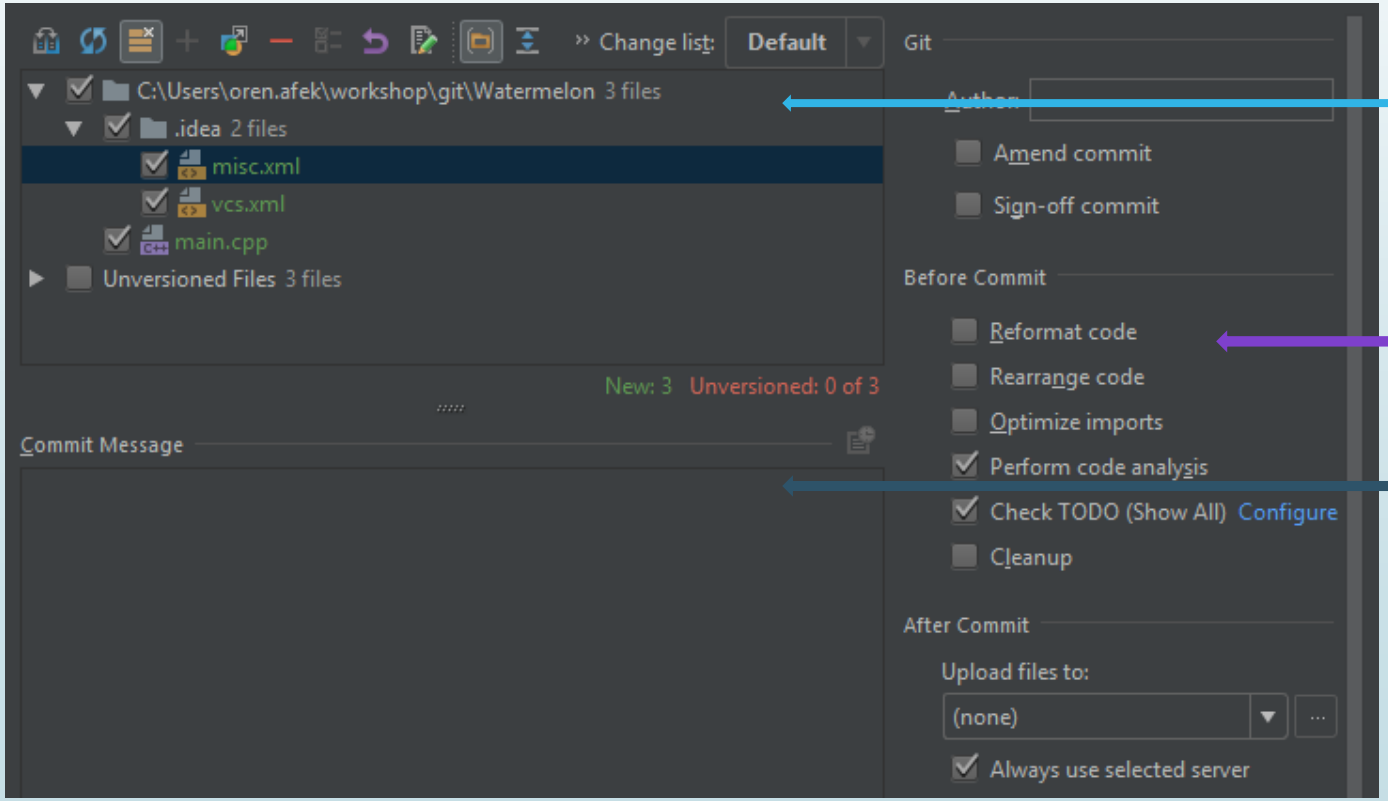


- JetBrains VCS GUI
  - Pull the latest version from the server.



git

- JetBrains VCS GUI
  - Commit latest changes



Staging area

Useful actions to do before committing

Commit message

git

- JetBrains VCS GUI
  - Push your commits