SENG275 – Lab 1
Due Friday May 12, 11:59 pm

Welcome to SENG275. Today we'll be familiarizing ourselves with the software we'll be using throughout the course, and writing some basic unit tests. We'll be working inside a gitlab repository, and we'll finish up by pushing our changes to gitlab.csc.uvic.ca.

All the software you'll need for the course is already installed on the lab machines, but you may wish to install it on your personal computers as well. We cannot offer individual technical support for your own personal computers, but the first half of this document covers the process in detail.

The work required for each lab will be due on Friday night at 11:59 pm. Only the files present in your gitlab repository at that time will be marked.

Quick summary of what you need to do:

1. Create your lab repository with lab01 in Assignments on https://gitlab.csc.uvic.ca, and open it in IntelliJ.
2. Write some basic unit tests using the Junit platform and API.
3. Push your lab01 repository to gitlab.

Step-by-Step:

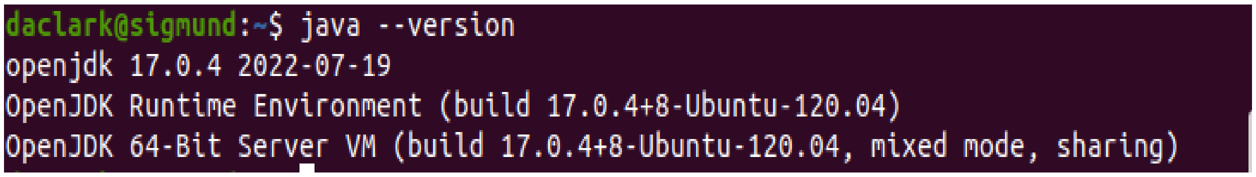
If you'd like to install the software on your own personal computer, follow the instructions in this section. If you're comfortable working on the lab machines, you can skip this part.

If it's not already present on your personal computer, you may need to install the following:

1. Java
2. IntelliJ IDEA Ultimate
3. Git

Proceed through the list at your own pace, and please let your TA know if you're encountering anything confusing. If you're not done by the end of the class, please submit your screenshot by next Monday at 5 pm Victoria time. If you have any questions after the class, send them to **@TeachingTeam** on Microsoft Teams.

Installing Java

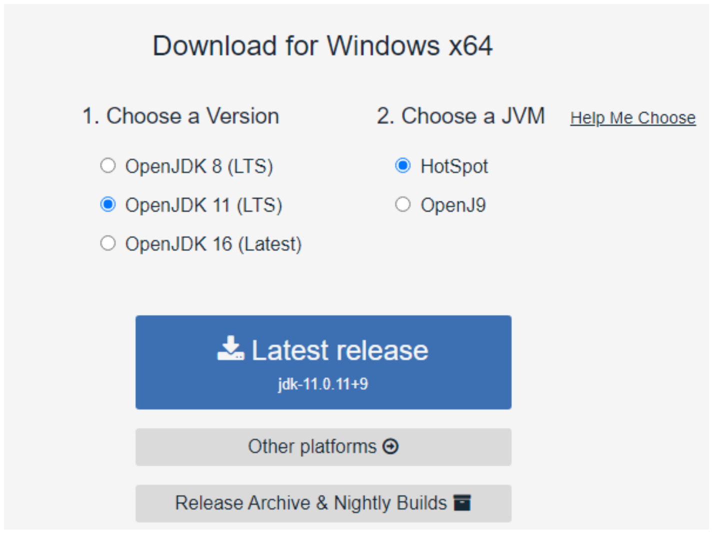


```
daclark@sigmund:~$ java --version
openjdk 17.0.4 2022-07-19
OpenJDK Runtime Environment (build 17.0.4+8-Ubuntu-120.04)
OpenJDK 64-Bit Server VM (build 17.0.4+8-Ubuntu-120.04, mixed mode, sharing)
```

We have tested the course software with Java 11 and Java 17; other versions may work as well, but if you're having trouble running some of the software, try installing one of those versions.

First, check to see if you already have a Java JDK installed on your machine. Try the following two commands at the terminal or command line: `**java -version**` and `**java --version**`. If either of them gives you a version number, then you have java installed, and can proceed to the next section.

If both of these commands gave you an error message, then you don't have Java installed on your machine. You need to download a Java JDK from one of the open-source implementations - I suggest https://adoptium.net/. Your choice of version and JVM will depend on your platform; I recommend choosing Java 11 and the HotSpot JVM, but any version between 11 and 17 should work fine. You may have to reboot your computer to complete the installation - repeat the `**java -version**` and `**java --version**` terminal commands to confirm that your installation went ok.



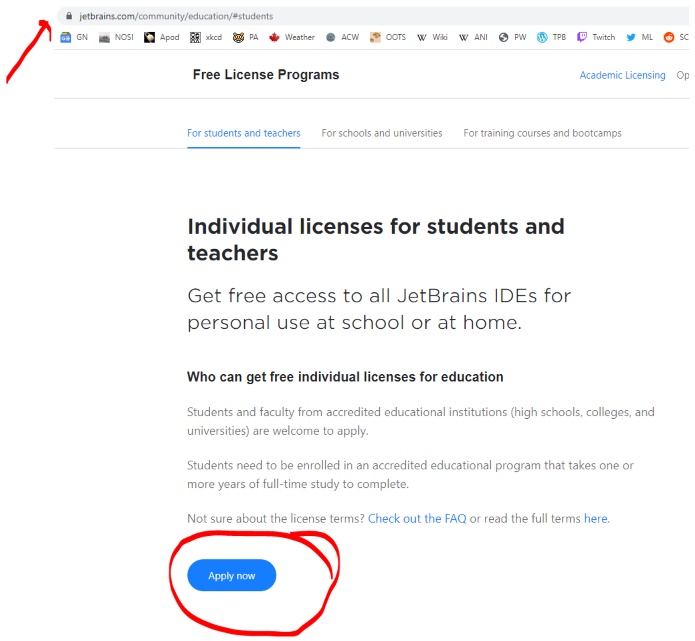
Download for Windows x64

1. Choose a Version
   - OpenJDK 8 (LTS)
   - [✓ OpenJDK 11 (LTS)]
   - OpenJDK 16 (Latest)

2. Choose a JVM — Help Me Choose
   - [✓ HotSpot]
   - OpenJ9

Installing IntelliJ

IntelliJ is a commercial development environment that is provided in two versions: the Ultimate edition, and the Community edition. We'll eventually be using some of the code coverage tools and automated testing capabilities that are only included in the Ultimate edition, so it's best to use that. Fortunately, every student at UVic has access to the Ultimate edition through NetBrains' Free Educational License program.


jetbrains.com/community/education/#students
GN NOSI Apod xkcd PA Weather ACW OOTS Wiki ANI PW TPB Twitch ML SC
Free License Programs
Academic Licensing
For students and teachers
For schools and universities
For training courses and bootcamps
Apply now

Individual licenses for students and teachers

Get free access to all JetBrains IDEs for personal use at school or at home.

Who can get free individual licenses for education

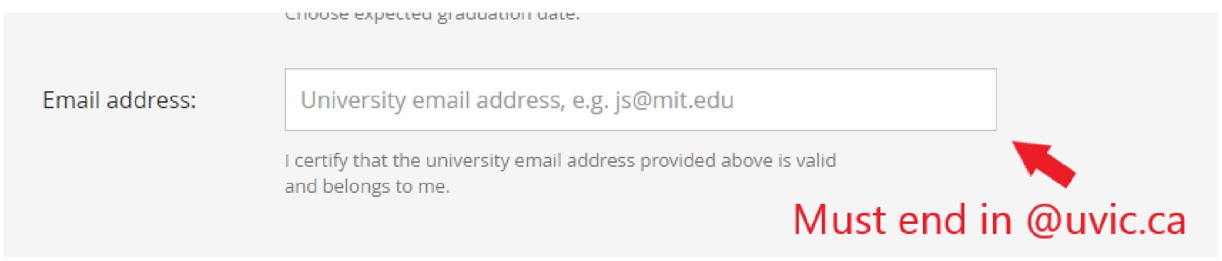
Students and faculty from accredited educational institutions (high schools, colleges, and universities) are welcome to apply.

Students need to be enrolled in an accredited educational program that takes one or more years of full-time study to complete.

Not sure about the license terms? Check out the FAQ or read the full terms here.

To install IntelliJ:

1. Go to the JetBrains website at https://www.jetbrains.com/community/education/#students
2. Click on the Apply Now button.
3. Fill out the JetBrains Products for Learning form. Where it asks to enter your email address, **make sure you provide your UVic email address!** (This will be your netlink login name, followed by @uvic.ca). Renewals and other information from JetBrains will come to your UVic email account, so make sure to check there for further information.



Choose expected graduation date.

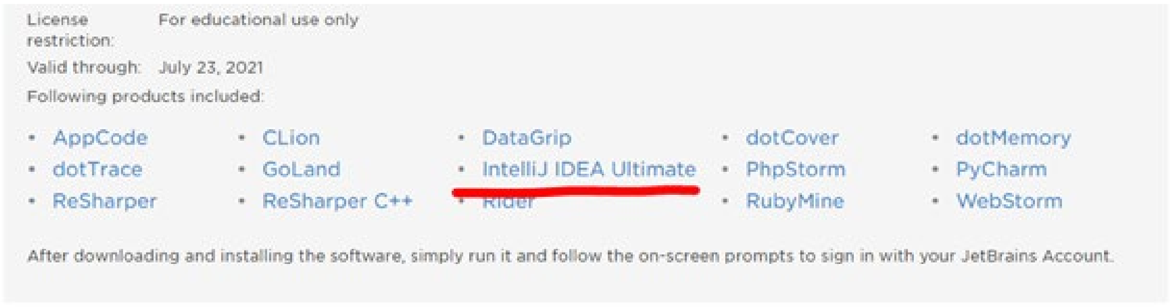
Email address: University email address, e.g. js@mit.edu

I certify that the university email address provided above is valid and belongs to me.

Must end in @uvic.ca

4. The confirmation email should arrive shortly - click on the link in the email to confirm your submission.
5. You'll have to scroll down to the bottom and approve the license agreement.
6. Now you can create a JetBrains account -

   Do NOT use your UVic name and password!
7. Once you've signed into the JetBrains website, you'll see a list of software that you're able to download and install for free. We want the IntelliJ IDEA Ultimate edition - it's available for Windows, Mac and Linux machines, so make sure you choose the appropriate version.



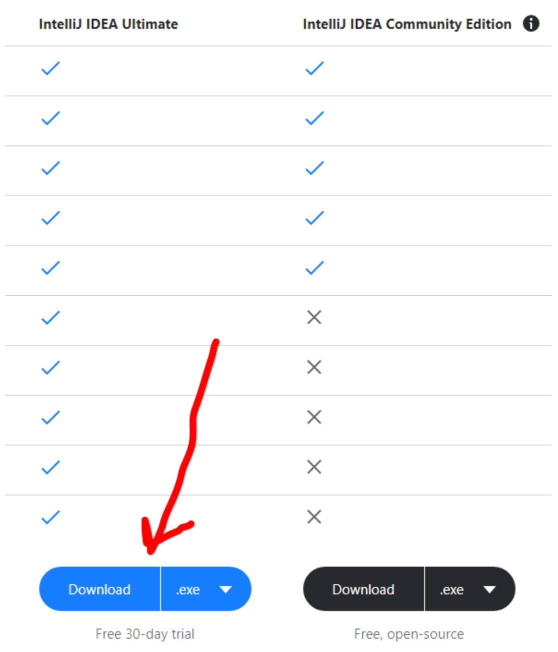
License restriction: For educational use only

Valid through: July 23, 2021

Following products included:

AppCode	CLion	DataGrip	dotCover	dotMemory
dotTrace	GoLand	IntelliJ IDEA Ultimate	PhpStorm	PyCharm
ReSharper	ReSharper C++	Rider	RubyMine	WebStorm

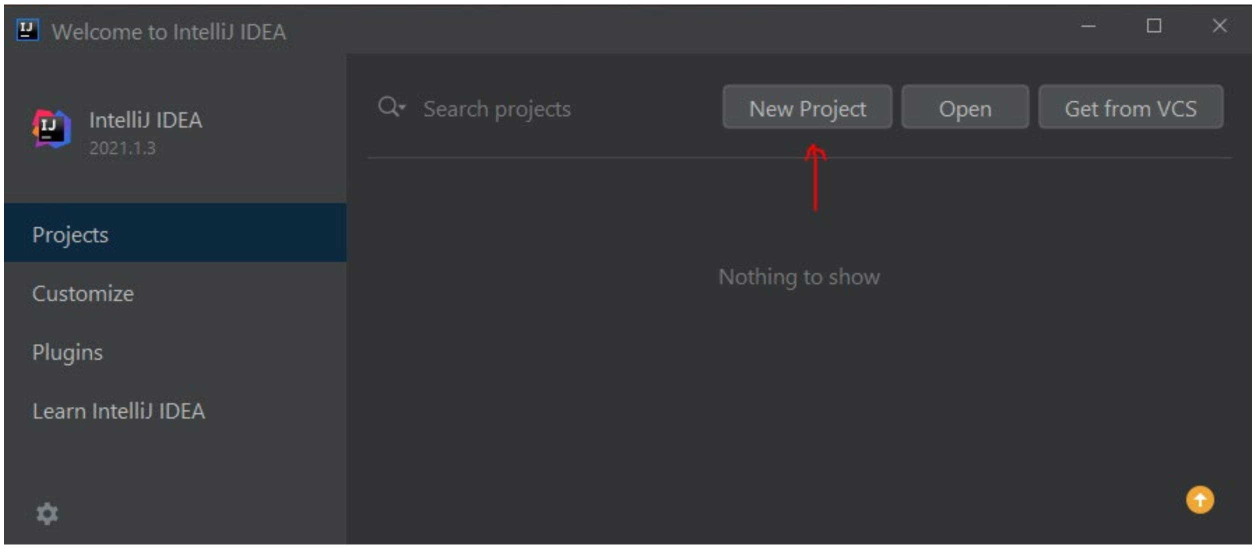
After downloading and installing the software, simply run it and follow the on-screen prompts to sign in with your JetBrains Account.



8. Install the software. Choose where you want to install it on your computer, and read the installation options - its safe to leave the boxes unchecked.

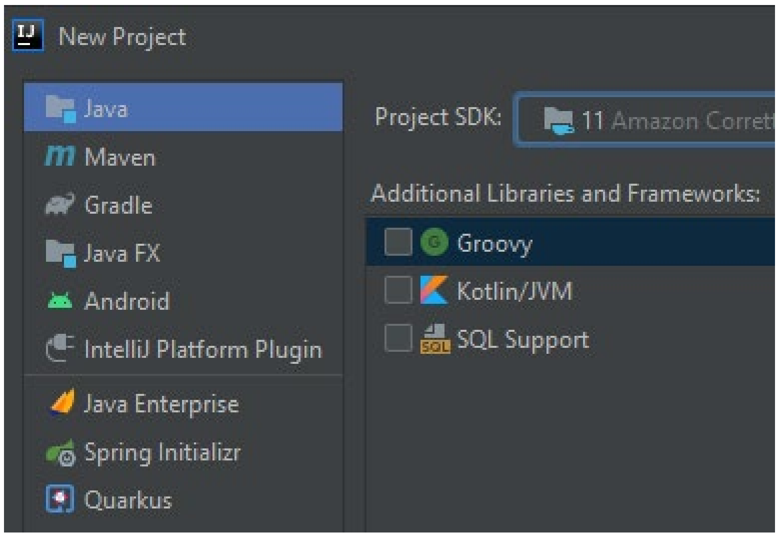
Let's make sure everything worked:

1. Run IntelliJ. You should see the Welcome to IntelliJ IDEA window open up.
2. Click New Project.



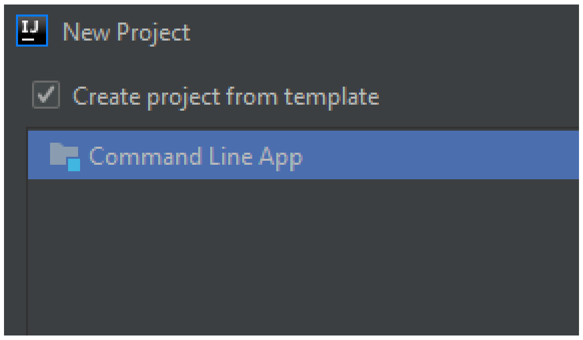
3. You'll see the New Project dialog - note that if you don't have Java installed on your machine, IntelliJ will use its built-in Java SDK, which is currently Java 11. The version of

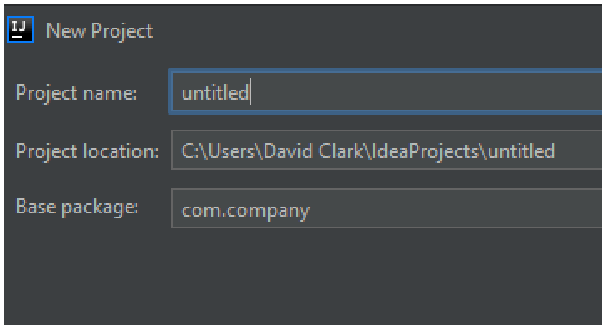
Java that's included with IntelliJ is limited, and JetBrains suggests we install a standalone version, which we did above. Click Next to create a basic Java project.



Project SDK: 11 Amazon Corret

4. Enable Create Project from Template, and choose Command Line App. Click Next, and then Finish.



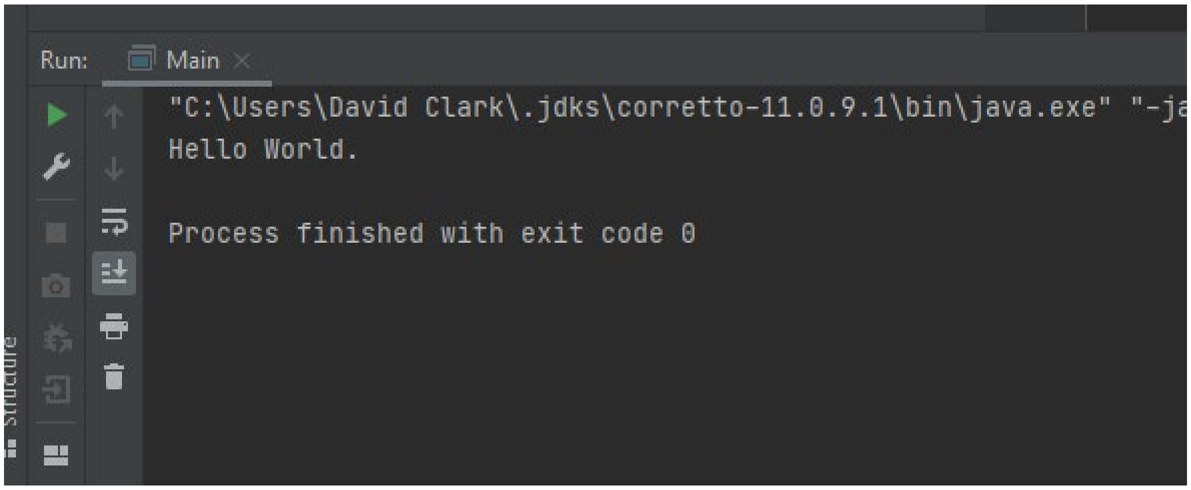


Project name: untitled

Project location: C:\Users\David Clark\IdeaProjects\untitled

Base package: com.company

5. IntelliJ will show you `**Main.java**`, and will start downloading some things in the background. Go ahead and add the following to the main function: `**System.out.println("Hello world.");**`
6. Run the program by clicking the green play icon in the upper right corner, or by choosing `Run -> Run 'Main'` from the menu, or by typing Shift-F10. Confirm that `Hello World.` is printed in the output window at the bottom before continuing.

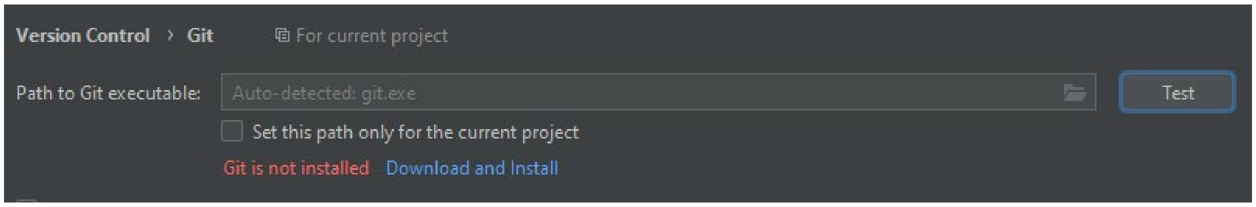


```
"C:\Users\David Clark\.jdks\corretto-11.0.9.1\bin\java.exe" "-ja
Hello World.

Process finished with exit code 0
```

Git

You may already have Git installed on your computer. Go to `**File -> Settings -> Version Control -> Git**`, and click the Test button on the right. If something like '**Git is not installed**' appears, click the '**Download and install**' text beside it, and install a version of git for your machine.

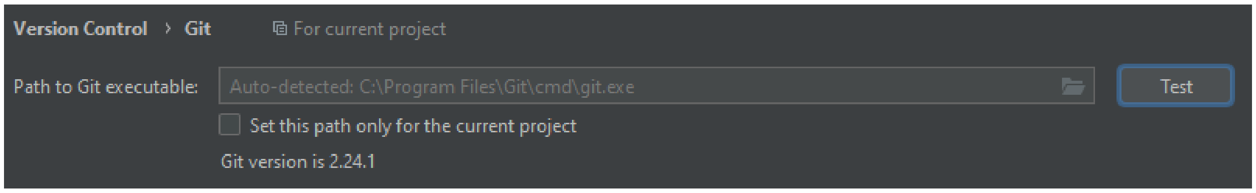


Path to Git executable: Auto-detected: git.exe

Set this path only for the current project

Git is not installed

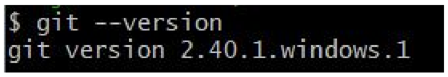
Test that Git works properly : type `**git --version**` at the command line or in a terminal. You should see something like `**git version is 2.24.1**` - the exact version isn't critical, as long as you're reasonably up-to-date.



Path to Git executable: Auto-detected: C:\Program Files\Git\cmd\git.exe

Set this path only for the current project

Git version is 2.24.1



```
$ git --version
git version 2.40.1.windows.1
```

Your course repository is located at gitlab.csc.uvic.ca. You can find it by selecting '**Projects -> Your Projects**' from the upper left corner of the screen.

This course won't be covering the use of Git, which was one of the tools covered in SENG265. If you're having trouble remembering how to clone/push/pull a repository, don't worry; some of the commands are a little obscure. Check the hands-on tutorial that GitLab provides. A more advanced understanding of Git will be quite helpful during the class; refer to the online book on the topic if you wish.

IntelliJ has robust gui support for most of the basic Git functions. That said, you are encouraged to become familiar with the command line functionality provided by git. You should also feel free to try some of the visual gui git utilities that are out there - they may help you better visualize changes you make to your repository. Your teaching instructors will be using the command line exclusively, and may be unable to support you if you rely on an unfamiliar tool.

A few notes on Git best practices for this course:

- Make sure you commit frequently - at least once per exercise.
- Write clear commit messages - when writing software with others, these messages help communicate your intent to your co-workers. If in doubt, check out a guide on how to write clear commit messages.
- Push frequently to the GitLab remote, so that any automated builds and tests are triggered.

Gradle (optional)

The currently available version of Gradle is 7.6, and is available from gradle.org. You can complete everything in the course without installing gradle manually, and you will not be tested on your knowledge of gradle at the command line, but it is a commonly-used build management tool for Java, and familiarity with it may help you in your future career. As the course goes on, you may decide that you like the way the software projects we test are set up (in terms of its directory structure, etc) and want to reproduce that structure for your own projects. You can do so using either the command-line version of gradle, or the one built-in to IntelliJ.

If you choose to investigate further, try running some of the built-in tests with gradle from the command line. From the project's main directory, try running `**gradlew check**`, or `**gradlew test**`, or `**gradlew staticAnalysis**`. Note the output, both on the console and in the reports generated in `**build/reports**`. Now check the `**build.gradle**` file, which is where these three commands are defined - can you see which tests were run for each?

Lab 1 Exercise – write some basic unit tests

Create your labs with lab01 repository in assignments – you'll find it at https://gitlab.csc.uvic.ca/

You can do this from the command line, or you can do so from within IntelliJ (Use **File -> New -> Project From Version Control**). IntelliJ has extensive Git support, so it's usually easier to do everything from within the Git interface.

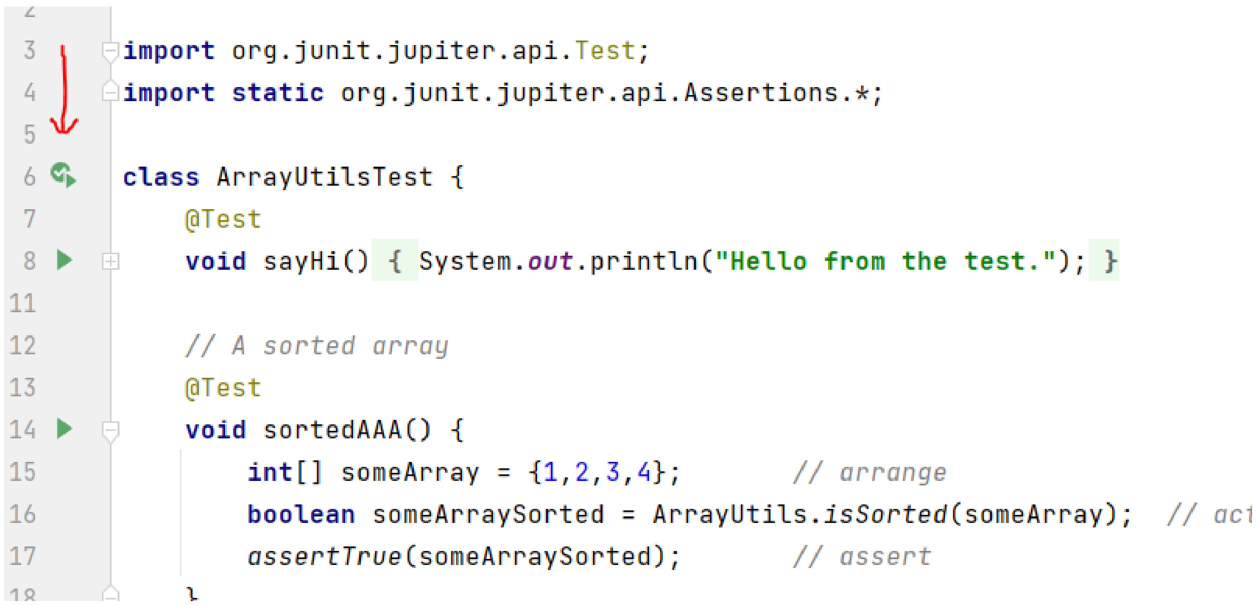
The lab01 project is a very simple one – it contains only one application class called ArrayUtils, and that class has only one static method – isSorted(). **ArrayUtils.java is available in Brightspace**. isSorted takes an int array as input, and returns a boolean – true if the array is sorted smallest to largest, and false otherwise. Empty arrays or arrays of one integer are treated as sorted, as are consequtive identical values.

isSorted() is located at **app>src>main>java>lab01>ArrayUtils.java**. Our goal is to test this method.

Running Tests

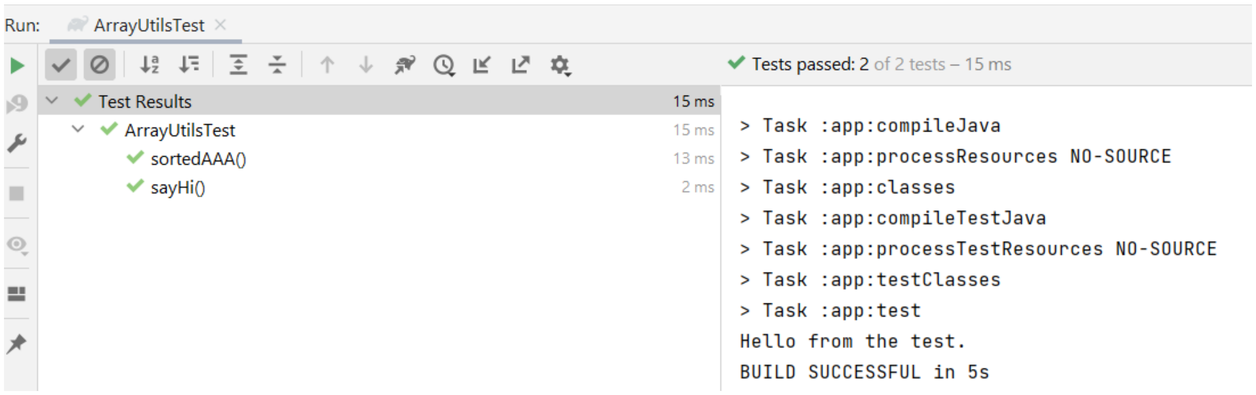
Our test class should be at **app>src>test>java>lab01>ArrayUtilsTest.java. ArrayUtilsTest.java is available in Brightspace.** When we write unit tests against a given class, we always write one test class, and we name it so that it's easy to remember which class it tests.

IntelliJ can run entire test suites or individual tests due its close integration with Gradle. We have two tests that have already been written – **sayHi()** and **sortedAAA().** Try running these tests by clicking the arrow-and-checkmark icon in the 'gutter' – the column between the line numbers and the source code.



```
3   import org.junit.jupiter.api.Test;
4   import static org.junit.jupiter.api.Assertions.*;
5
6   class ArrayUtilsTest {
7       @Test
8       void sayHi() { System.out.println("Hello from the test."); }
11
12      // A sorted array
13      @Test
14      void sortedAAA() {
15          int[] someArray = {1,2,3,4};        // arrange
16          boolean someArraySorted = ArrayUtils.isSorted(someArray);   // act
17          assertTrue(someArraySorted);        // assert
```

You'll see the results of the tests in the lower-left corner:



```
Run:  ArrayUtilsTest
Tests passed: 2 of 2 tests – 15 ms

Test Results            15 ms
  ArrayUtilsTest        15 ms
    sortedAAA()         13 ms
    sayHi()              2 ms
```

```
> Task :app:compileJava
> Task :app:processResources NO-SOURCE
> Task :app:classes
> Task :app:compileTestJava
> Task :app:processTestResources NO-SOURCE
> Task :app:testClasses
> Task :app:test
Hello from the test.
BUILD SUCCESSFUL in 5s
```

It is a very desireable feature of test that they run very quickly. That said, sometimes we would prefer to run an individual test, rather than the whole suite. Do so now for the sortedAAA() method, by clocking on the green right-arrow-and-circle icon in the line number gutter on line 14.

Writing Tests

Let's look at the isSortedAAA() test method.

```
// A sorted array
@Test
void sortedAAA() {
    int[] someArray = {1,2,3,4};        // arrange
    boolean someArraySorted = ArrayUtils.isSorted(someArray);  // act
    assertTrue(someArraySorted);        // assert
}
```

We're using the Junit testing framework, which recognizes test methods and allows them to run tests and see their results as shown on the previous page. To use this framework, we import from **org.junit.jupiter.api.Test**. In order to mark a method as a test method, we write **@Test** before the return type. The return type of any Junit test MUST be void.

The textbook (in "Software testing automation") also discusses the "**triple-A**" test structure – *Arrange, Action, Assert*. First we 'arrange' – we set up the system to permit us to run our test. This may be done in one line, as in this test, or it may be an extensive process involving database or network access and the instantiation of hundreds of classes. In our case, we're just creating a simple array of integers.

When we 'act', we call the method we're testing. In general, we should only have one 'action' per test – if we interact with the code we're testing multiple times, and the test fails, it can be hard to tell which action caused the fault.

Finally, we 'assert' – we claim something, and the test passes if that thing is true. In this case, we know that the array is sorted, so the boolean we get back from isSorted() should be **true**. If it isn't, we know something's wrong with isSorted().

Assertions

Java has a built-in assert statement, but we won't be using it.

```
assert x == y;
```

is just shorthand for writing:

```
if (x != y) {
     throw new RuntimeException();
}
```

Assert statements like these are DISABLED by default – they have no effect at all. They must be enabled by passing the -ea parameter to the java command on the command line in order to have effect. They throw unchecked AssertionError exceptions, so do not need to be declared with the **throws** statement, and should never be caught or handled in any way – their only role is to end the program. Note the lack of parentheses; assert is a statement, not a function (although putting in parentheses won't hurt).

This form of assert is not used to test software – it is used by the developers of software to enforce invariants. The problem with the built-in assert statement is that it simply ends the program unless caught, and prints difficult-to-understand information to the console. That's not the behaviour we want – we may be running hundreds or thousands of tests, and we need simple yes-or-no output that can be confirmed at a glance.

We will therefore not use this type of assertion in this course – this section is here to explain its omission.

Junit Assertions

Junit provides an assertion class alongside its other testing framework capabilities. Junit also permits us to use other, third-party assertion libraries (such as AssertJ), but Junit is pretty straightforward, and so we'll start with it.

To use these assertions, we'll need to add import statements to the top of our file:

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

These assertions take the following forms:

assertEquals(*expected_value, actual_value*);
assertTrue(*some_expression*);
assertFalse(*some_expression*);
assertSame(*expected_object_reference, actual_object_reference*);

There are more; see the Junit 5 documentation for the full list. People have a lot of trouble remembering the "expected goes first, actual goes second" order, and getting them in the wrong order can produce some very confusing error messages. IntelliJ annotates the expected and actual arguments, so that helps, and Junit also provides useful information in its exception messages.

We'll get into much more detail about Junit and its assertions as the course goes on.

Your task:

Complete the remaining tests and confirm that they pass.

Commit and Push to Gitlab

Ensure that your tests pass, and then commit and push your repository to the Gitlab remote. Choose Git -> Commit, and select the files in the default changelist that you've altered. Type a meaningful commit message in the "Commit Message" box, and click "Commit and Push". If you have not yet run git config, you may be asked for your name and e-mail address during this first commit.

Go to Gitlab and ensure that your changes are present in the remote repository.

We cannot mark your work if you do not push your changes back to gitlab.csc.uvic.ca.

Things to think about:

You don't need to answer these questions, but it would be good to think about them as the course goes on. Each would make a good midterm question.

1. Why can't we exhaustively test our entire software project? What should we do instead?
2. What is the pesticide paradox about and what does it imply to software testers?
3. Why should we automate, as much as possible, the test execution?
4. Why are the test classes named the way they are?
5. How many test methods should there be in each class?
6. There are assertions, exceptions, and invariants – three related, but different concepts. Explain in your own words the differences between these concepts, and how they are related.
7. What is Gradle? What is it used for?