

SWEN-601

Software Construction

Introduction to Java &
"Hello, World!"



Computer Programming

(with Java)



Fundamentals of Computer Programming

- A *computer program* is the encoding of an algorithm in a language that a computer can understand and execute.
- A *programming language* is:
 - Understandable by the computer*.
 - Readable and writable by humans.
 - Provides tools that can be used to express algorithms (as discrete steps executed in sequence).
- This semester, we will be using the Java programming language.

A *high level* programming language (like Java) that is understandable by humans usually must be *translated* into a **machine instructions** that the computer understands...

MyProgram.java

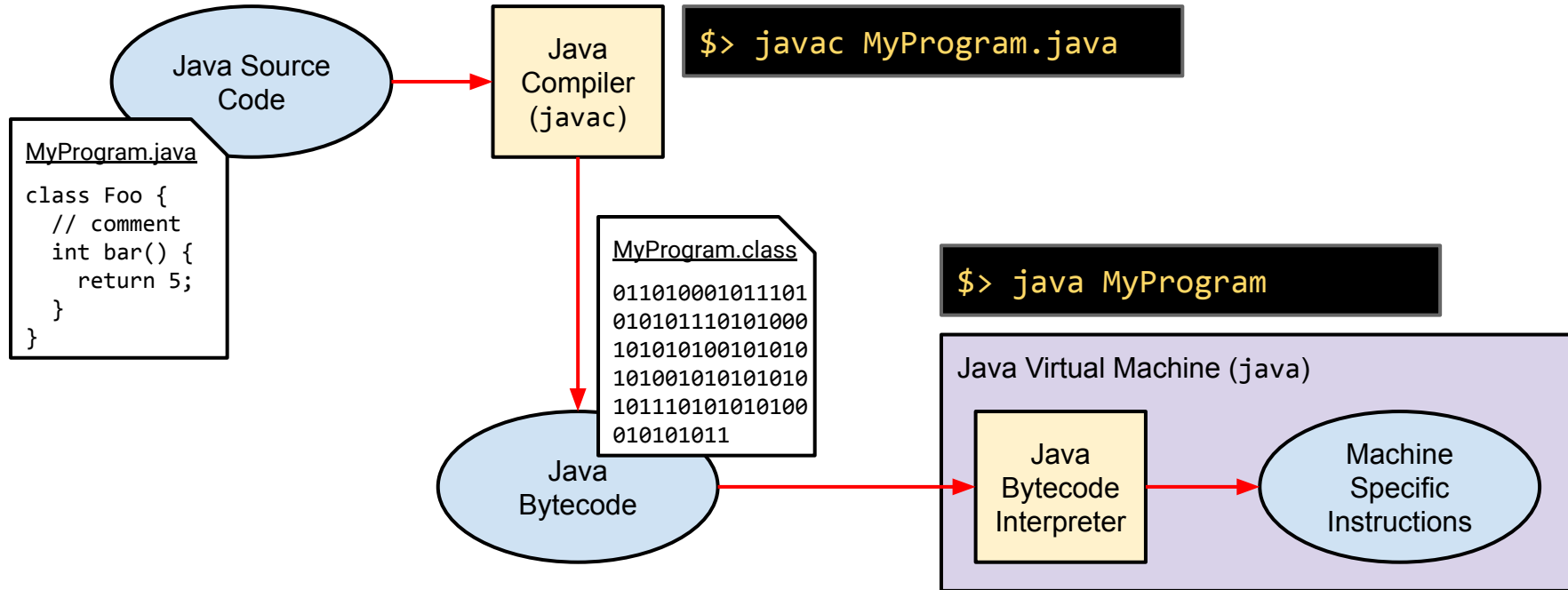
```
class Foo {  
    // comment  
    int bar() {  
        return 5;  
    }  
}
```

MyProgram.class

```
011010001011101  
010101110101000  
101010100101010  
101001010101010  
101110101010100  
010101011
```

This process of translating human-readable code into machine instructions is called **compiling**.

How Java Works



Scaffolding

- Some languages, like Python, allow you to type programming instructions directly into a file and execute them.
- Java is not like that. It requires **scaffolding** before a program will compile and run.
 - A **class** declaration.
 - A **main** method.
- We will fully explain what these things are in upcoming lectures.
- For today just follow the required pattern as shown here.



```
public class MyProgram {  
    public static void main(String[] args) {  
  
    }  
}
```

Activity: Your First Java Program

You may use one of the lab computers or your personal computer for this activity. Write, compile, and run your first Java program by following these instructions:

1. If you have not already, log into the computer.
2. Use **Notepad** (Windows) or **vi** (Mac/Linux) to create a new text file named "MyProgram.java". **Do not** use TextEdit on OSX.
3. Type the code in exactly as it is shown below and save the file.

```
public class MyProgram {  
    public static void main(String[] args) {  
  
    }  
}
```

4. Open a command line.
 - a. Windows: use the Start Menu to search for "command".
 - b. OSX: Use Spotlight Search (Command-Space) to search for "terminal".
5. Navigate to the directory to which you saved your program.
6. Compile the program using javac: **javac MyProgram.java**
7. List the files in the directory (Win: `dir`, Mac: `ls`). What do you see?
8. Run the program using java: **java MyProgram**

It is important to understand the steps involved in writing and running a Java program.

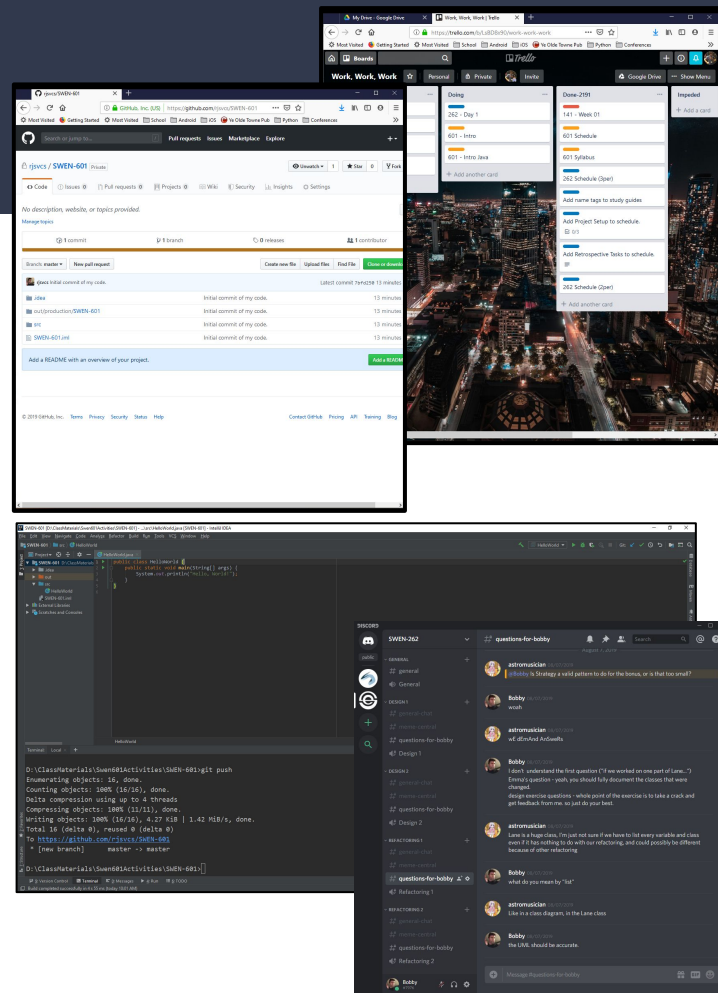
Software Development Tools

(Git, GitHub, & IntelliJ IDEA)



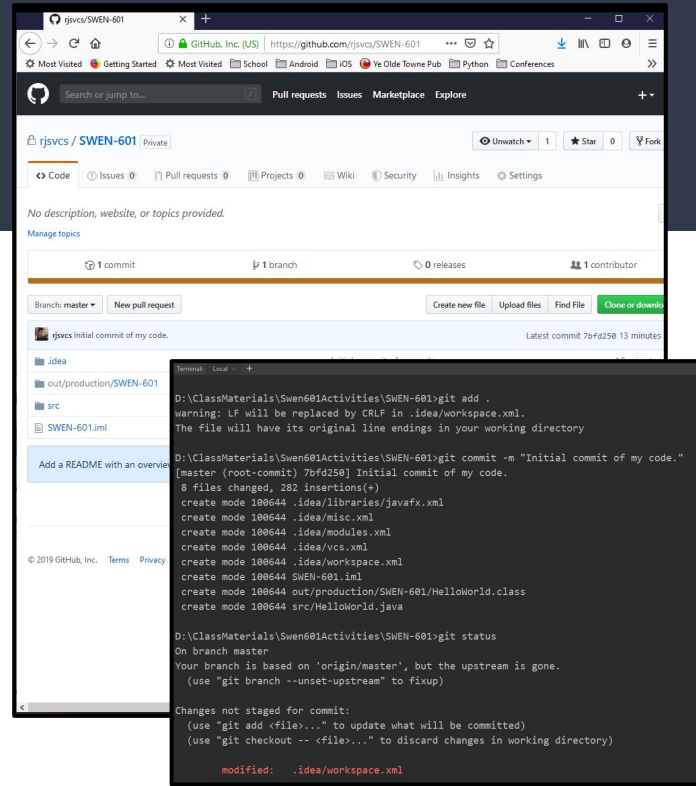
Tools of the Trade

- Most software developers do not write code in plain text editors and compile it from the command line.
- Most developers use a suite of tools to enhance development.
 - **Source Control Management (SCM)**
 - **Integrated Development Environment (IDE)**
 - **Task Tracking, e.g. Trello, Jira**
 - **Communication, e.g. Slack, Discord**
- This semester we will focus mainly on the first two.
 - **Source Control Management with Git/GitHub**
 - **The IntelliJ IDEA Integrated Development Environment.**



Source Control Management

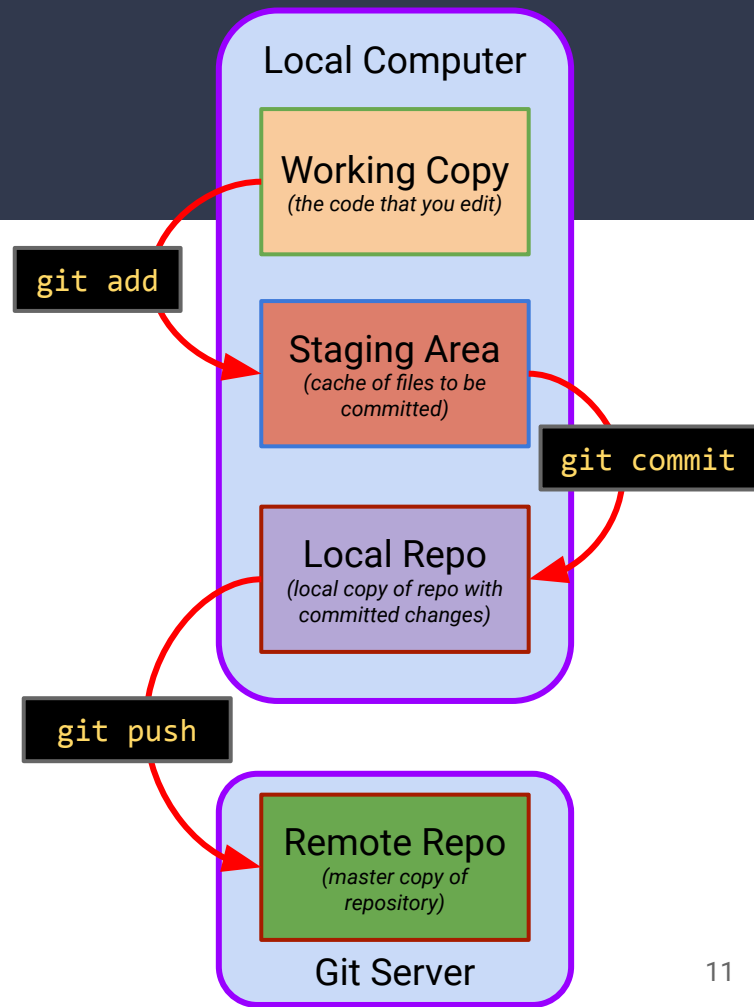
- Source Control Management (SCM) provides a mechanism for you to backup your code.
- It does a lot more than that, too.
 - Allows teams of developers to work on the same code and keep their code up to date.
 - Keeps track of version history, so code can be reverted to an older version.
 - Can be connected to Continuous Integration (CI) to automatically build and test every time code is changed.
- This semester, we will be using Git and GitHub for SCM.



We'll use [GitHub](https://github.com) as a secure, cloud-based Git server and `git` from the command line.

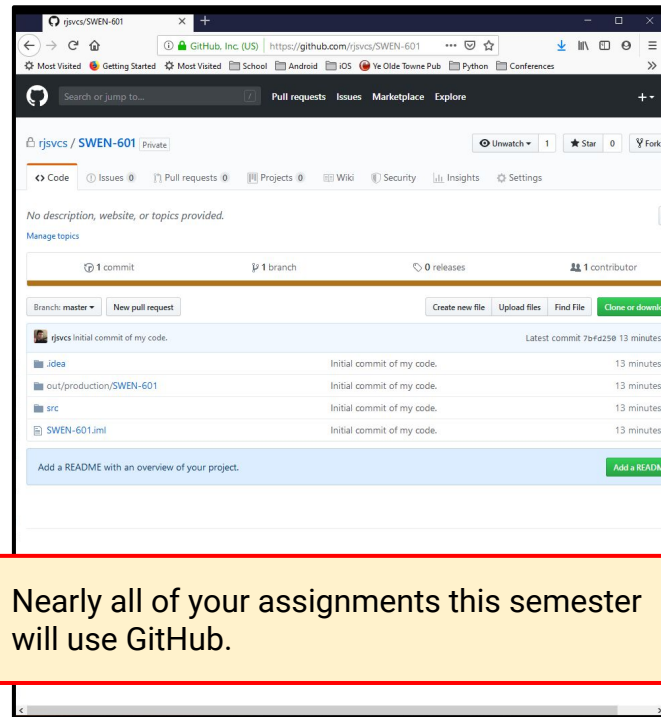
Git in a Nutshell

- Most SCM systems store a **working copy** of code on the local computer.
 - When the programmer changes the code in their working copy, they can **commit** the changes to a server.
 - Other developers can **check out** those changes to download them to their own working copy.
- Git works a little differently. The local computer stores up to *three* different versions of the code.
 - A **working copy** for the developer to change.
 - A **staging area** to which changes are temporarily **added**.
 - A **local repository** into which the developer has **committed** their changes.
- A Git server (GitHub) stores the **remote repository** that is a copy of the code shared by the team.
 - Changes must be **pushed** to move them from the local repository to the remote repository.



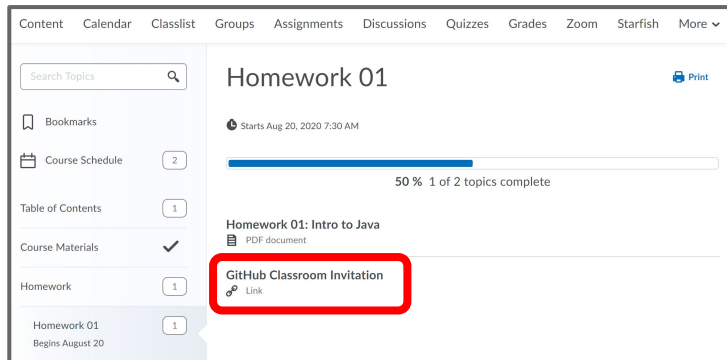
GitHub

- A Git server is a remote machine, accessed over a network, and to which code is pushed.
- The server keeps a master copy of all of the changes ever made, committed, and pushed.
- GitHub is a Git server “in the cloud.”
 - It can be accessed through a browser.
 - You can create unlimited repositories.
 - Repositories can be **public**, i.e. visible to everyone on the Internet, or **private**, i.e. visible only to you and people that you invite.
- While you can create your own Git server any time, we’ll use GitHub for this class.



Activity: Accepting an Assignment

1 Your instructor will provide you with a GitHub classroom link. Find it on MyCourses under “Homework” in the Content section



The screenshot shows the MyCourses interface. The top navigation bar includes links for Content, Calendar, Classlist, Groups, Assignments, Discussions, Quizzes, Grades, Zoom, Starfish, and More. The main content area is titled 'Homework 01' and shows a progress bar indicating '50 % 1 of 2 topics complete'. Below the progress bar, there is a section for 'Homework 01: Intro to Java' with a PDF document icon. In the left sidebar, under the 'Homework' section, the 'GitHub Classroom Invitation' link is highlighted with a red box.

2 The very first time you accept an assignment, you will need to pick your name in the roster and link it to your GitHub Account.

Identifiers	
Anderson, George	>
Carpenter, Joe	>
Doe, Jane	>
Johnson, Jerry	>
Smith, John	>
St. Jacques, Bobby	>
Zimmerli, Daniel	>

3 After linking your account, you will be prompted to accept the assignment. Do so, and a new, empty, **private** repository will be created for you. Copy the URL.

You're ready to go!

You accepted the assignment, **Homework 01**. Your assignment repository has been created:

<https://github.com/SWEN-601/homework-01-rjsvcs>

Note: You may receive an email invitation to join **SWEN-601** on your behalf. No further action is necessary.



Activity: Cloning Your Repository

1 Copy the URL to your repository from your browser (if you need to, you can click the name of the repository in the list on the left).



2 Open a command line. On Windows use the Start Menu to search for "command." On Mac, use spotlight search to search for "terminal."

3 Make a new directory for your SWEN-601 class activities, homework, and practica, e.g. /Users/bobby/Swen601Code, and then change into that directory.

```
bb8:~>mkdir 601Code
bb8:~>cd 601Code
bb8:~/601Code>_
```

4 Use the **git clone** command to clone your repository (paste the URL that you copied in step 1). You will see a warning message that your repository is empty.

```
bb8:~/601Code>git clone https://github.com/SWEN-601/hmwk-01-rjs
Cloning into 'SWEN-601'...
warning: You appear to have cloned an empty repository.
```

```
bb8:~/601Code>cd SWEN-601
bb8:~/601Code>ls
bb8:~/601Code>
```

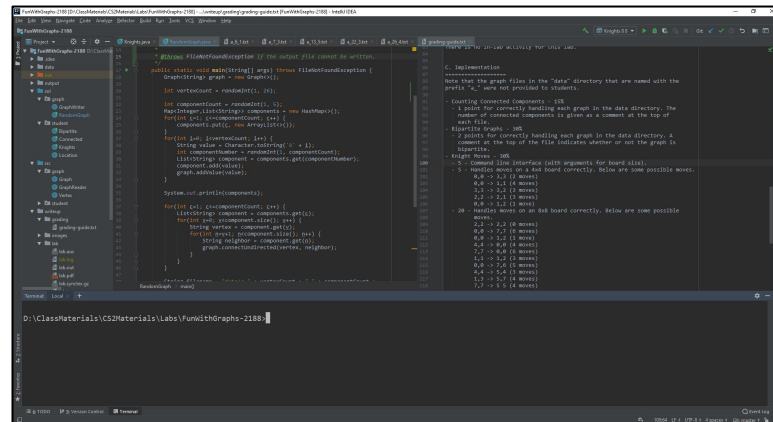
5 Cloning the repository created a new directory. If you change into the new directory and list the contents, you will see that it is empty. Next step: *let's add some code!*

Integrated Development Environment (IntelliJ IDEA)

- An Integrated Development Environment (IDE) brings *almost* everything that you need into one user interface.
 - A text editor with syntax highlighting
 - An integrated compiler
 - A launcher to run programs
 - A terminal window
 - Integrated SCM*
 - And much more (including plugins)

- IDEs are **very** feature rich, and can therefore sometimes have very intimidating user interfaces. *Don't panic!*

- This semester we will use [IntelliJ IDEA](#).



Now that you have gone through the process of using a text editor and the command line to write, compile, and run a program you should be able to appreciate an IDE that provides a “one stop shop” for your development.

** IntelliJ IDEA does have integrated support for Git, but it's flaky. We'll use the terminal.*

A Quick Tour of IntelliJ IDEA

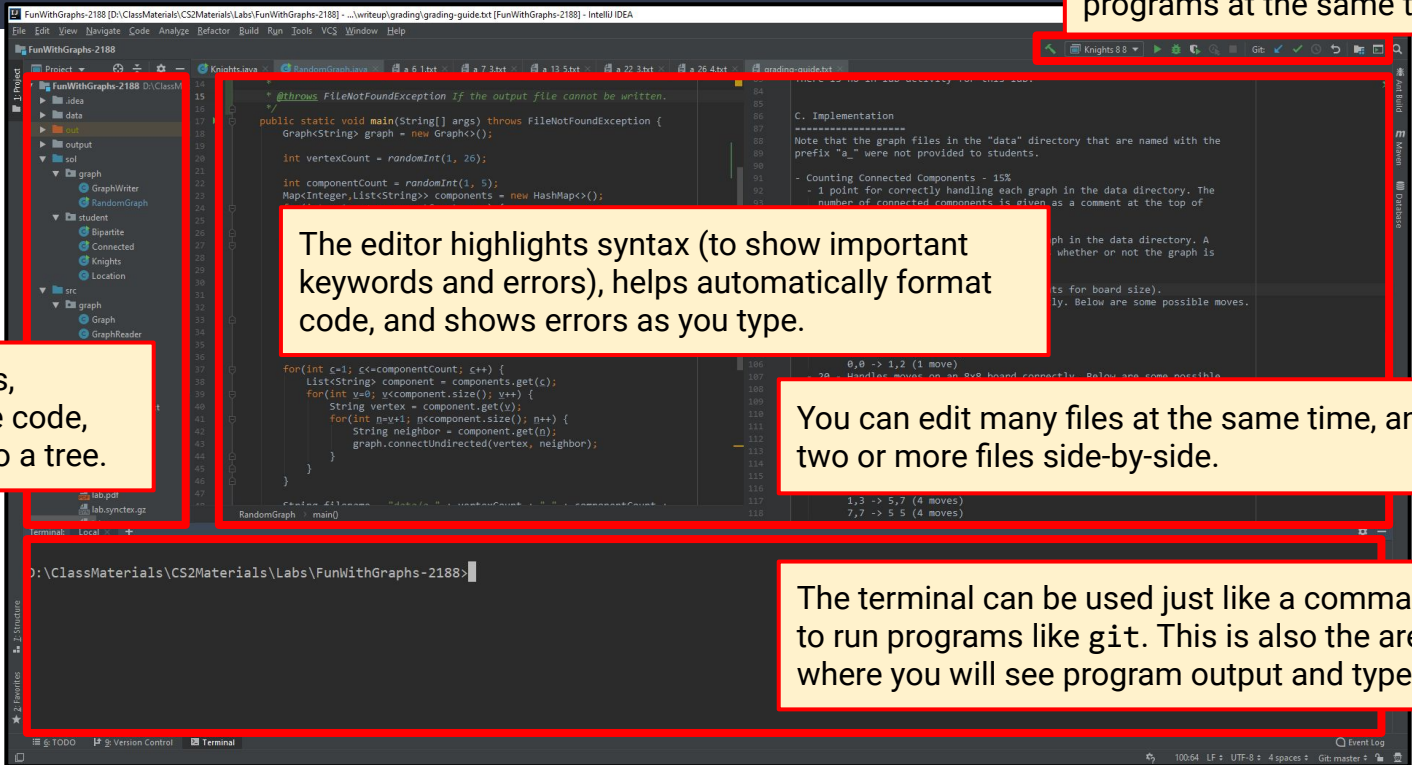
Run controls let you start, stop, restart, and debug multiple programs at the same time.

The editor highlights syntax (to show important keywords and errors), helps automatically format code, and shows errors as you type.

Your project files, including source code, are arranged into a tree.

You can edit many files at the same time, and see two or more files side-by-side.

The terminal can be used just like a command line to run programs like git. This is also the area where you will see program output and type input.



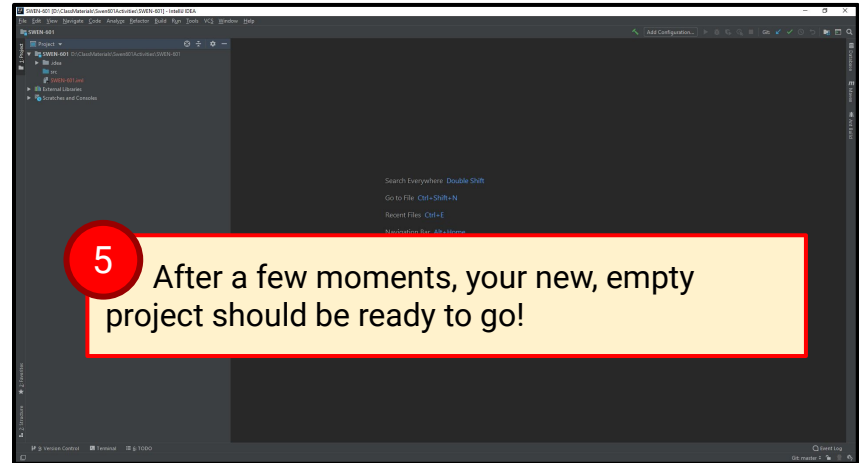
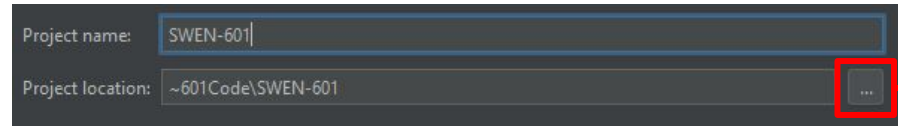
Activity: Creating a new Project

1 Start IntelliJ IDEA. If you are prompted for a license, log into your JetBrains account (you should have an educational license).

2 Use the **+ Create New Project** button in the splash or the *File* → *New* → *Project...* menu to start a new project.


3 Make sure to select **Java** from the options on the left before hitting next. Accept the defaults on the next screen.

4 Type the path to your empty Git repository or use the **...** to browse to the location. Name your project the same. Then click *Finish*.



5 After a few moments, your new, empty project should be ready to go!

“Hello, World!” (Standard Output)



The Development Lifecycle

- As you write code, you will repeat the same series of steps over and over until your code is finished.
 - Some code is never really “finished,” and so you will repeat these steps as long as you work on the code.

1. Pull the latest code from GitHub using git pull.
2. Add some new code.
3. Compile, run, and test your code. *
4. Fix bugs until your tests pass.
5. Use git to commit your changes to your local repository.
6. Push your changes to GitHub.

Step 1 is particularly important when you are working with other programmers on the same project at the same time *or* if you are switching between computers (e.g. a lab computer and your laptop).

If you are working alone, on a single computer, you will cycle through steps 2 through 6 over and over.

A typical *agile* developer may work through this entire list once every 5 minutes or so.

* We will talk about what it really means to test your code in a few weeks.

“Hello, World!”

- “Hello, World!” is often the first program that a developer writes when learning a new programming language.
- The program simply prints the message “Hello, World!” to standard output.
- The programmer learns:
 - The basic syntax of the language.
 - How to produce textual output.
 - How to compile their program.
 - How to run the program.
- That’s a lot for just a few lines of code!

We’ve already learned that Java requires a minimum amount of **scaffolding** for any executable program.

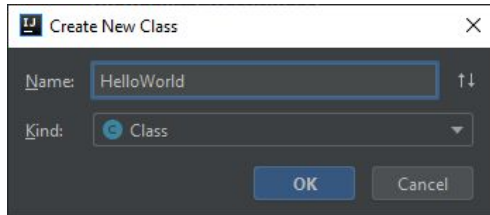
This includes a **class declaration** and a main method.

```
public class MyProgram {  
    public static void main(String[] args) {  
    }  
}
```

Now we will learn how to use IntelliJ to create a new Java project, add code, compile it, and run it all from inside the IDE.

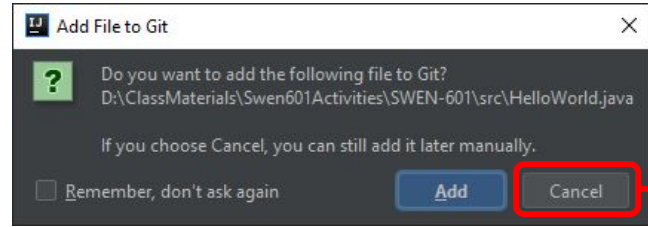
Activity: Add Some New Code

1 In your project, right click on the src folder and select *New* → *Java Class* from the popup menu.

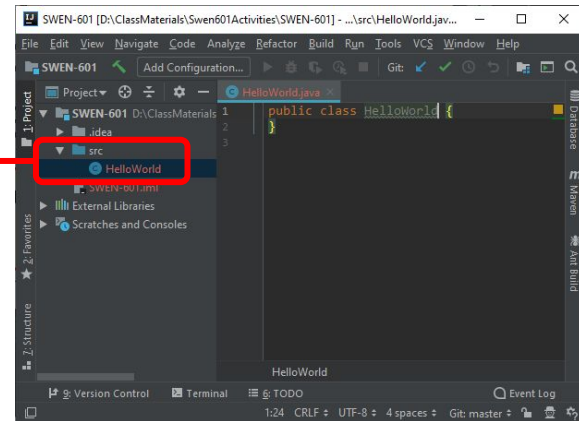


2 In the dialog, name your new class HelloWorld and click OK.

3 You will be asked whether or not you want to add your new file to Git. Click “Cancel” for now.



4 You will notice that the new class has appeared in your src tree and is open in the editor. Use the editor to add a main method now (refer to your notes).



Standard Output

- **Standard Output** in most cases refers to sending textual output from inside your program to that default output destination for the environment in which your program is running.
 - This is usually the terminal window.
- In Java, `System.out` is a special feature that is used to access standard output. It provides lots of different features, including:
 - `System.out.println` - prints a line of text ending with a newline.
 - `System.out.print` - prints text without a newline.

In Java, a **string** of text is enclosed in double quotes, e.g. `"Buttercup"`.

```
System.out.println("Buttercup");  
System.out.print("123");  
System.out.print("456");  
System.out.println("789");
```

If executed, the code above would produce the output below in the terminal window.

```
abc  
123456789
```

Activity: Compile, Run, & Test Your Code

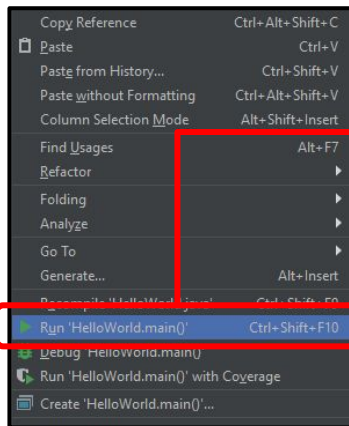
1 Modify your HelloWorld main function so that it prints a "Hello, World!" message to standard output.

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

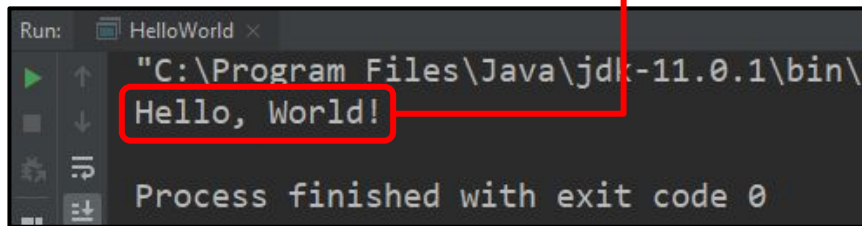
2 There is no need to save your code; IntelliJ continuously saves as you type. If you make a mistake, it will be highlighted or underlined in red.

```
System.out.println("Hello, World!");
```

3 Once your code appears to be error free (no red), IntelliJ can compile and run it in one step. Right click anywhere in your code and choose *Run* in the menu.



4 If there are no bugs or errors, you should see output in the Run window at the bottom.



A Closer Look

This is a class declaration. The name of the class must exactly match the name of the file that contains it (including the case).

The **body** of the class is enclosed in curly braces ({}). All code in Java must be inside the body of a class.

```
public class HelloWorld {
```

```
    public static void main(String[] args) {
```

The main method must be declared exactly as shown here.

```
        System.out.println("Hello, World!");
```

The **body** of the main method is enclosed in curly braces ({}). The code inside the body of the main method is executed when the class is run.

```
    }
```

```
}
```



Activity: Use git to Commit Your Changes

1

Remember that backing your code up to GitHub is a three step process: *staging*, *committing*, and *pushing*. First, let's open a terminal and use `git status` to see what has changed.

```
Untracked files:
```

```
...
```

```
.idea/  
SWEN-601.iml  
out/  
src/
```

2

We have untracked files! Time to stage them so that they can be committed to the local copy of the repository (you may see a warning).

```
bb8:~/601Code/SWEN-601>git add .
```

3

Next, create a local backup of your changes on your computer by committing the staged changes to your local repository.

```
bb8:~/601Code/SWEN-601>git commit -m "My first commit."  
[master (root-commit) 7bfd250] Initial commit of my code.  
8 files changed, 282 insertions(+)  
create mode 100644 .idea/libraries/javafx.xml  
create mode 100644 .idea/misc.xml  
create mode 100644 .idea/modules.xml  
create mode 100644 .idea/vcs.xml  
create mode 100644 SWEN-601.iml  
create mode 100644 out/production/SWEN-601/HelloWorld.class  
create mode 100644 src/HelloWorld.java
```




Activity: Push Your Changes to GitHub

You can commit as many times as you'd like, but code that has been committed is still only on your local computer.

To move changes to your Git server, you need to **push** the changes.

1

Open a terminal and use `git push` to push your local changes up to the Git server.

```
bb8:~/601Code/SWEN-601>git push
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 4 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (16/16), 4.27 KiB | 1.42 MiB/s, done.
Total 16 (delta 0), reused 0 (delta 0)
To https://github.com/rjsvcs/SWEN-601
 * [new branch]   master -> master
```

2

If you try it again immediately, you will see that there is nothing to push.

```
bb8:~/601Code/SWEN-601>git push
Everything up-to-date
```

Comments

- It is considered good practice to **comment** your code as you write it.
 - Comments allow you to use plain language to describe the intent of your code.
- Java supports three kinds of comments:
 - Single-line comments that begin with `//` and continue to the end of the line.
 - Block or multi-line comments that begin with `/*` and continue until `*/`.
 - Javadoc comments, which are a special kind of comment that can be parsed and translated into a web page that describes your class and code. Javadoc comments begin with `/**` and end with `*/`.

```
// this is an example of a single line comment

public class Foo { // they can follow other code

// you may sometimes choose to use single line
// comments on more than one line.

/*
 * But you could choose to use a multi-line
 * comment to write a comment that takes up more
 * than one line, too.
 */

/**
 * Javadoc comments are special, and we will talk
 * about them in much more detail in the coming
 * weeks.
 *
 * For now you can focus on the other two kinds.
 */
```



Activity: Comment Your Code

1 Add a few comments to your HelloWorld program to document it. Don't worry too much about what they say.

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

2 When you are finished, use `git status` in the terminal to see what has changed.

```
bb8:~/601Code/SWEN-601>git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git checkout -- <file>..." to discard changes in working  
  directory)  
  
        modified:   .idea/workspace.xml  
        modified:   src/HelloWorld.java  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

3 Stage (add), commit, and push your changes to GitHub.

4 Open your repository in a browser and look around. What do you see?

QUESTIONS?!