Name_____ CSC 210
Due: 09 November 2025 Fall 2025

Project Management and Existing Codebases

- **Objective**: Learn about generating documentation, unit testing, and correct project structure
- **Key Concepts**:
  - Javadoc Comments
  - Unit Test / Junit
  - packages
  - Maven
  - Style Rules
  - Jar Files
  - Code Refactoring
  - Project Directory Structure
  - Object-Oriented Design

Creating Your Project

- Enter your local copy of your git repository on your lab or workstation and create a lab-06 directory.
- Download the zip file of lab-06 from Brightspace and copy the examples folder into your repository in your lab-06 directory

Methods

A. Add in and generate javadoc comments

1. Inside the examples directory, take a moment to review the `README` file and project structure and content of the project named `SimpleJavadocDemo`.
2. Make sure you are able to use the code in a test program and that you understand how the program works.
3. Using the command suggested in the `README` file, generate documentation for the project. Verify that the command was successful by confirming that the relevant html files have been generated.
4. Take some time to load some of the files in a browser and navigate around.
5. Do the same as the previous 4 steps for the project named `Documentation`. Note that this project will involve a bit more setup and has instructions specific for a Unix-like system such as MacOS or Linux. It is okay if you don't get the full project working, but do take some time to appreciate what is set up within it.
6. Go to the project titled, `Object-Oriented-Tic-Tac-Toe`, compile it, use it, come to an understanding of how the code works.
7. Write `javadoc` comments for every class and method in the `Object-Oriented-Tic-Tac-Toe` project.
8. Use the command line to generate the web pages for the `javadoc` comments you wrote and browse them by opening them with your web browser.

Name_____ CSC 210
Due: 09 November 2025 Fall 2025

B. Write tests for a project

1. Inside the examples directory, take a moment to review the `README` file and project structure and content of the project named `Testing`.

2. Using the commands provided, compile the code.

3. Run the `myprog Main` program that demonstrates usage of the packages

4. Run the tests for the library code and read the results.

5. Add an additional class in the `com.main.math` package named `BooleanLogic`. Give the class 4 static functions. There function names should be: `And`, `Or`, `Not`, and `Xor`. They should each take two truth values, except for `Not` which will take one, as parameters. Each function should return the result of performing the corresponding logical operator. So for instance, `And` should return `arg1 && arg2` and `Or` should return `arg1 || arg2`.

6. Write unit tests for each of the four functions in the class. Rely on the structure of the other packages and their tests as examples for how to structure everything.

C. Split up a project that is all in one file to separate files

1. Compile and use the single file program named `Aquarium-Sim.java` in the examples directory.

2. Study the code and understand its structure

3. Create a new project directory named `AquariumSim`. Inside this directory, break up the single file program code for `AquariumSim` so that each class from the original program has its own file.

4. Once the program is broken up, compile and run the code to make sure that it works.

D. Split up a project that is all in one directory into separate packages

1. Make a copy of the project `Object-Oriented-Tic-Tac-Toe` and name it `PackagedTicTacToe`.

2. Divide the project up into 3 separate packages. There names should be: `gamelogic`, `userinterface`, and `main`. Create directories for each package, move files relevant to the packages in their respective directories, and place the package name at the top of each file.

3. Be sure to import your packages inside code files that reference code from other packages.

4. Attempt to compile and run your program to prove that you have made all of the necessary code changes. Remember that you will need to reference class paths with the `-cp` flag for everything to work. Consult the `Testing` project for help with the structure of the packages and with building the commands for compiling and running the project.

Name_____ CSC 210
Due: 09 November 2025 Fall 2025

E. Study How Maven Projects Are Organized

1. Explore the projects in `examples/Maven`. Maven is a standard build tool for Java. It simplifies the work of configuring and managing a project as it grows in complexity.

2. Start with `simple-maven-project`. Read the `README.md` file. Execute the suggested commands.

3. Explore the directory structure. Read the `App.java` source code and note its simplicity. Also look at the file, `pom.xml` and note how it specifies the project structure and its dependencies. Maven relies on `pom.xml` for determining how to build, clean, test, and execute the project.

4. Do the same for the project named `multi-file-maven`. Notice how the structure differs with multiple source files. Make note of what remains the same and what differs with the commands in the `README.md` and the structure of the `pom.xml` file.

5. Now move to the project, `library-catalog`. In addition to having multiple files and packages, this project contains unit tests. Note the extended structure of the project and the additional commands in `README.md`. Also make note of the import of `junit` in `pom.xml`.

Run the commands. Make sure you you understand how this project extends the functionality we've been looking at up to this point. Also compare and contrast this project with the `Testing` project we looked at earlier which had unit tests but did not use maven. Instead, it had custom scripts for achieving the same effect. Which method is easier to manage?

6. Next, look at the project named `doc-demo`. This project is a library package without a main program. It contains the packages with library code and unit tests. It also has commands in the `README.md` file for generating and hosting documentation pages using `javadoc`. Note the extensive structure in pom.xml for specifying dependencies and commands.

Try to run the commands to compile, run unit tests, and generate and host the documentation pages. Try to understand how the commands in the `README.md` file relate to what is in `pom.xml`. Also, compare and contrast this maven-based project with our earlier projects demonstrating documentation, `SimpleJavadocDemo` and `Documentation`.

7. Finally, open up the project, calculator client. Notice that this project has a main program and imports the package from doc-demo. Note how this achieved within the project's structure. Compile and run the project. Remember this pattern as it is a common pattern for code reuse in maven that you will see again and again.

Name_____ CSC 210
Due: 09 November 2025 Fall 2025

F. Convert a program from procedural to object-oriented code with packages, unit testing, documentation, and maven.

1. Compile and run the program in the source file, `LemonadeStand.java` at the root of the examples directory.

2. Play the game and understand how it works.

3. Analyze the code

4. Create a new directory named `ObjectOrientedLemonade`

5. Break the code up into class files so that the logic of the program is built around an object-oriented design

6. Compile and run your new program

7. Adjust the program so that it can build using Maven

8. Break everything up into packages

9. Add javadoc comments

10. Add unit tests

11. Practice building, executing unit tests, generating and serving documentation, and executing the program to confirm that everything is working.

12. Make a `README.md` file specifying how to perform all these activities in your project. If you need help getting things working, refer to all of the other projects, especially in the Maven directory, as examples.

Conclusion

Congratulations! You now have a firm understanding of what it takes to manage a Java project that is ready to be developed by a group of people and potentially deployed to users! This knowledge will server you well in future labs where projects will take on this level of complexity and refinement. Not only that, you are ever closer to being prepared to work with code bases out in the wild and in a professional work setting. Perhaps you will even begin contributing to a project sooner than you may realize!