**Submission Requirements**

1. Always create a directory at the root of your repo with the name of the assignment (or lab). example, "**assignment3**".

2. Put all materials for that assignment in that directory.

3. Within that directory, you should have a separate package for every problem.

a. The package names should follow this naming convention: assignmentN.problemM where you replace N with the assignment number, and M with the problem number, e.g., all your code for problem 1 for this assignment must be in a package named: assignment1.problem1. (There is only one problem in Assignment 3)

4. Make sure to include the build.gradle and other files.

a. It's easiest to just work within the directory on your machine.

b. We must be able to load the project, build it, and run it (and your tests)

5. Make sure your programs have a main method and that it is able to run.

6. Please include a **readme.md** file that specifies how to run your code. Especially if parameters are used.

Additional expectations for every package:

* One .java file per Java class
* One .java file per Java test class
* One pdf or image file for each UML Class Diagram that you create
* All methods must have tests
* All non-test classes and non-test methods must have valid Javadoc

**Lastly, a few more rules that you should follow in this assignment:**

* Your classes should have a public modifier
* Instance fields should have a private modifier
* Use this to access instance methods and fields inside a class

**Code Criteria**

* **Gradle built:** Your project should successfully build using the provided **build.gradle** file, and it should generate all the default reports.
* **Javadoc generation:** Your Javadoc generation should complete with no errors or warnings.
* **Checkstyle report:** Your Checkstyle report must have no violations.
* **Code coverage report:** Your JaCoCo report must indicate 70% or more code coverage per package for "Branches" **and** "Instructions".
* **Methods hashCode(), equals(), toString():** all of your classes *have to* provide appropriate implementations. (appropriate means that it is sufficient to autogenerate these methods, as long as autogenerated methods suffice for your specific implementation). Please don’t forget to autogenerate your methods in an appropriate order - starting from the ancestor classes, towards the concrete classes.
* **Javadoc:** please include a short description of your class/method, as well as tags from @params and @returns in your Javadoc documentations (code comments). Additionally, if your method throws an exception, please also include a tag @throws to indicate that.
* **UML diagrams:** Please include UML diagrams for the final versions of your designs for every problem. In doing so, please note that you do not have to hand-draw your UML diagrams anymore. Auto-generating them from your code will be sufficient.