CS6910
Lab 01

In the first iteration of this application, you'll do the following (see Suggested Readings in Moodle):

- Apply the DRY Principle ("Don't Repeat Yourself") by using *Extract Superclass* refactoring method to remove code duplication
- Complete an implementation of the *Observer Design Pattern* to let the application follow the *Model-View-Controller* design

Goals:

- Practice writing Java code
- Apply software development principles and design patterns to guide how you design and code
- Use version control software to track your work

What to do:

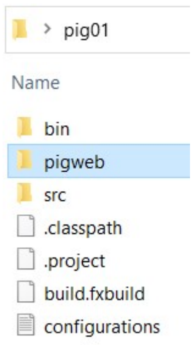
1. Download the runnablePig2105.jar file. This is an *executable JAR* file. JAR (Java ARchive) is a file format that allows us to collect a group of files together similar to Zipping. The difference is that we can set these things up to be executable and actually run an application (which you'll do next).

2. After downloading the JAR, go ahead and double-click it to run the sample Pig application you should be ending up with this week. It's probably worth running this one a few times to actually play the game with different strategies and see it working.

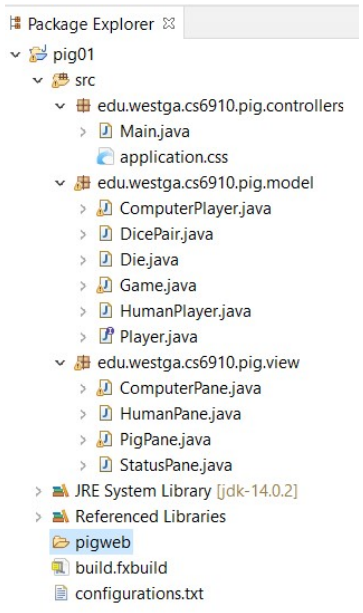
   Realize this is a 'stripped-down' version of Pig. Yes it plays; however there is a lot of room for improvement. Like I say, I'm just trying to get things up and running this week. There will be plenty of time for improvements later.

3. Once you have finished running the sample executable, you'll need to visit Moodle to download and extract the Starter files for this week's work. Note that this Starter does not contain a repository.

4. We want to keep the Web User Guide you created together with this Java application. So go ahead and just drag the folder with your HTML files into the root directory for your Eclipse project.



```
> pig01
Name
bin
pigweb
src
.classpath
.project
build.fxbuild
configurations
```



```
Package Explorer
pig01
  src
    edu.westga.cs6910.pig.controllers
      Main.java
      application.css
    edu.westga.cs6910.pig.model
      ComputerPlayer.java
      DicePair.java
      Die.java
      Game.java
      HumanPlayer.java
      Player.java
    edu.westga.cs6910.pig.view
      ComputerPane.java
      HumanPane.java
      PigPane.java
      StatusPane.java
  JRE System Library [jdk-14.0.2]
  Referenced Libraries
  pigweb
  build.fxbuild
  configurations.txt
```

5. Once you have your workspace structure set up this way make a commit to your repository (with a meaningful commit message)

6. Go ahead and run the `pig` application as it sits right now (the `main` method is inside the `Main.java` class). There isn't much there, but hey, it compiles and runs, that's always a good place to start.

Now that our Eclipse workspace is set up correctly and we have a working (but pretty lame) application, now it's time to dig in and start writing some code. Fortunately, the person who came before us left a bunch of `// TODO:` comments to help us out.

7. Show a list of all tasks that need to be completed. You can do this by viewing the Tasks view. Just use the Eclipse menu system and choose **Window > Show view > Tasks**. This should give you a list of items that need attention.

8. Work through the `TODO` items one by one. Be sure to:
    a. Leave in the comments <u>until the end</u> – they'll be useful if you make a mistake along the way and will help me to locate the areas where you were asked to write code. Be sure to remove these //TODO comments before submitting.
    b. Include your name in an `@author` comment at the top of the code files that you modify
    c. Make use of other comments and similar code inside other methods/classes as a model for some of this code you need to write. I'm not sure you'll need to read and understand *every* line of code, but you'll definitely need to read and understand some of it.
    d. When you are running the sample, take note of where the different Java components are located, what the computer does when it takes its turn, etc. You'll need this information when completing the different Java classes.
    e. Commit along the way with meaningful commit messages. Note, be specific about what was done, not just "Fixed TODO in the Game class" – we don't want to force someone else to have to cross-reference your messages with some other document.

    I suggest starting with those classes in the `view` package first (except the refactoring of `HumanPlayer` and `ComputerPlayer` classes – see below) and end by modifying the `Game` class last. Along the way, be sure to commit the changes (with meaningful commit messages) to your repository.

9. Once you've made all the necessary changes be sure to test the application one last time (just as you did with the sample executable file, which you might even use again at this stage to compare) and be sure it is working as expected and then make a commit (with a meaningful commit message) to the repository.

It's now time to apply the software design principles of refactoring the code (see the Suggested Readings in Moodle for more details).

10. Refactor the classes `HumanPlayer` and `ComputerPlayer` (Hint: Think about extracting an abstract superclass here)

11. Test again to be sure the application still works as expected and commit your change.

Style
Please continue to follow the Best Practices taught in previous courses for code style and format. Remember that the first thing I will do with each of your submissions is run it through Checkstyle. If *anything* shows in the Problems tab, I will not look at anything else and will simply mark it as Unsatisfactory. You will then need to correct all of these issues before submitting again. Only one submission will be considered each weekday.

Assumptions / ReadMe File
At any point in time you feel like there are assumptions that you need to make when doing this (or any) assignment, please create a text file named `ReadMe` and store this file in the top-level of your project (so when I unZip your project, I will see this file at the same level as the '`bin`' and '`src`' folders). Inside this file just list out what assumptions you felt that you needed to make when doing the exercise. Also explain any design decisions that you are concerned about and feel the need to explain. Note that I will *not* be grading this document – so if you don't include one, that's fine. I'm just giving you this opportunity to help explain what you are thinking ahead of time.

And of course, if your assumptions aren't completely fair ("I assumed that this is a stupid exercise and no one would ever want such a bad piece of software" or "I saw that there were just so many flaws in the Starter File, that I threw it away and created my own application from scratch"), then I'll be sure to contact you about that and give you the opportunity to fix your files before moving ahead. Also, it's probably worth reading through the information about Grading below.

Submitting
When you are finished, make one final commit to your repository, be sure that all of the class files are saved, close Eclipse, and Zip the folder holding your Eclipse project using 7-Zip. Give your file the name 6910*YourLastName*Week01.zip. Upload the Zip file to the appropriate link in Moodle.

Grading:
Again, this is a pass/fail course – which is nice because it will allow us to talk about smaller issues that may come up along the way without worrying about maintaining a perfect 100% score. Once things are 'graded', please visit the Moodle Gradebook to see the comments I've left. Those submissions that receive 'Unsatisfactory' will need to get the issues fixed and approved by me (by sending your work via email) before proceeding to the next unit.

Remember, all work must be marked as Satisfactory to pass this course.