Objectives:

1. Learn how to use your IDE to assist with Refactoring
2. Identify code smells and refactor to solve them
3. Refactor to a design pattern target

Instructions:

1. Pay attention to steps where it asks you to insert a comment in the code to identify your work. If I can't find it, I can't grade it!

Task 1: Eclipse Refactorings:

1. For this lab you will use the source code provided on Blackboard (almost the same code as you have been using). Please create a new project from the code, do not try to import it as an Eclipse archive.
2. Create a new package banking.interfaces (right-click on the project in Eclipse, select New → Package)
3. Refactor: move the source files *Asset.java*, *InterestBearing.java*, and *Account.java* to this new package using the Refactor utility in Eclipse (right-click on source files, select Refactor → Move…). Ensure the proper import statements were added by Eclipse. If not, use the right-click "Add Import" functionality.
4. Rename the class variable *balance* to *accountBalance* in Account.java using the context menu Refactor → Rename. Indicate your refactoring using the comment "CST316 ACTIVITY 1.4" in the code (exactly like that, in CAPS, so I can search). Ensure the project still builds correctly (you may need to make other changes along the way).
5. A common *lexical design pattern* (meaning, naming convention) is to prefix interface classes with an "I" and Abstract classes with an "A", as in "IMyInterface" or "AMyAbstractClass". Rename the interfaces and abstract classes in the project according to this convention, and make sure the changes propagate through the project and the project still builds and runs. Use the Refactor → Rename functionality in Eclipse.
6. Ensure your project still builds completely and correctly in Eclipse and the program runs properly (Run As…, you need the my.properties as a command-line argument as before).

CHECKPOINT 1: Zip up all of your *source tree only* and save it as labRefactoringDP.<asurite>_task1.zip

Task 2: Find Code Smells and Refactor

1. Find any code smell *within a class* as described in your notes and/or Fowler's book (remember you can access it online via ASU's library access to Safari). Identify the smell by making a comment prefixed by "ACTIVITY 2-1 SMELL WITHIN A CLASS <*name of smell*>". Then proceed to refactor the code to remove the smell. Summarize your refactoring changes in your above comment.
2. Find any code smell *between classes* as described in your notes and/or Fowler's book. Identify the smell by making comments in ALL RELEVANT CLASSES prefixed by "ACTIVITY 2-2 SMELL BETWEEN CLASSES <*name of smell*>". Then proceed to refactor the code to remove the smell. Summarize your refactoring changes in your code comments.

CHECKPOINT 2: Zip up *source tree only* and save it as labRefactoringDP.<asurite>_task2.zip

Task 3: Introduce Design Patterns and Check our Metrics

Let's target our refactorings to introduce some simple Design Patterns into this code.

1. The AccountServerFactory (itself a Factory pattern instance) is responsible for creating the AccountServer. However, there should only be one AccountServer object in the entire application, but AccountServerFactory does not implement this constraint right now. Refactor AccountServerFactory to implement a full *Singleton* pattern. Note: You must NOT modify any code outside of AccountServerFactory to get this pattern to work!
2. Implement the Observable pattern for ServerSolution, particularly observe when an Account is added to the List of accounts that ServerSolution maintains. When it does have the observer print a message to the console: "Account NNN has been added, there are now XXX accounts on the server" where NNN is the name of the Account object added, and XXX is the new number of Account objects in the list after the add was successful. Use Java's built-in Observer class and Observable interface to create your solution.

   Note 1: you need to instantiate your Observer somewhere in the code and tie it to the Observable through a call to addObserver. Leave a comment "CST316 TASK 3 ADDOBSERVER" in the line preceding where you decide to do this.
   Hint: the straightforward way to do this is to refactor the List object into its own object extending Observable

CHECKPOINT 3: Zip up *source tree only* and save it as labRefactoringDP.<asurite>_task3.zip