ECS 160 Assignment 1

Due by Thursday, Oct 17 at 11:59 pm

There are 3 parts to the assignment. Credit as noted. You should zip the hw1 package that includes all the Java programs, with classes and methods named as indicated below and submit it to Canvas.

In every program, We will check the output with a test case, and also check the implementation, to ensure that it conforms to the required design pattern structure. You will be provided with an example JUnit test file and example expected output files to compare your class/method names and output with. Your method and class names should exactly match those in the example test file, and your output should match the expected output as well.

You should do the homework by yourself. Make sure you understand the concepts. The assignment grade will be the geometric mean of the score on the homework, and the (possible) related quiz, which will be given in the lecture. If you do the homework yourself, you will find the related quiz easy, otherwise, it will be well-nigh impossible.

At the start of each question, we list the required package and new classes needed to answer the question. Note that in Question 3, you will also need to modify some of the classes from Question 2.

1. (package hw1, PrintManager.java) Implement a class PrintManager as a SINGLETON pattern that allows at most one instance to be created. It should include a method called ThePrintManager() which is used to create the one instance. Its behavior should be logged to a file named "*Q1Log.txt*" in the working directory. The first call should write the string "Instance Created" to the log file. Every other call should write "Previously Created instance returned" to the log file. 5 points.

2. (package hw1, LibraryBook.java, LBState.java, Borrowed.java, OnShelf.java, GotBack.java, BadOperationException.java) Implement a LibraryBook as a STATE design pattern. The Basic LibraryBook should use an associated State. Implement each State as a SINGLETON: log "<state_name> Instance Created" when the state is created the first time and nothing on future references. Use a public static method getInst() to get the state's instance. Both LibraryBook and the states should support the methods shelf(), issue(), extend(), and returnIt(). Use Figure 1 below to model the states and applicable methods and effects. The initial state of every book should be OnShelf. Methods not shown transitioning *from* a state should throw a BadOperationException if called on that state (you should create that Exception class) with a message "Can't use <method_name> in <current_state_name> state". You need to log every successful state transition "Leaving State xx for State yy". For example, extend() should log "Leaving State Borrowed for State Borrowed". Finally, each concrete state should

support a toString() method that returns the state class name (e.g. “Borrowed”). Don’t worry about the actual details (updating the library database), we just want to see if you understand the STATE pattern. All the messages should be logged in a file called “*Q2&3Log.txt*” in the working directory. 10 points.

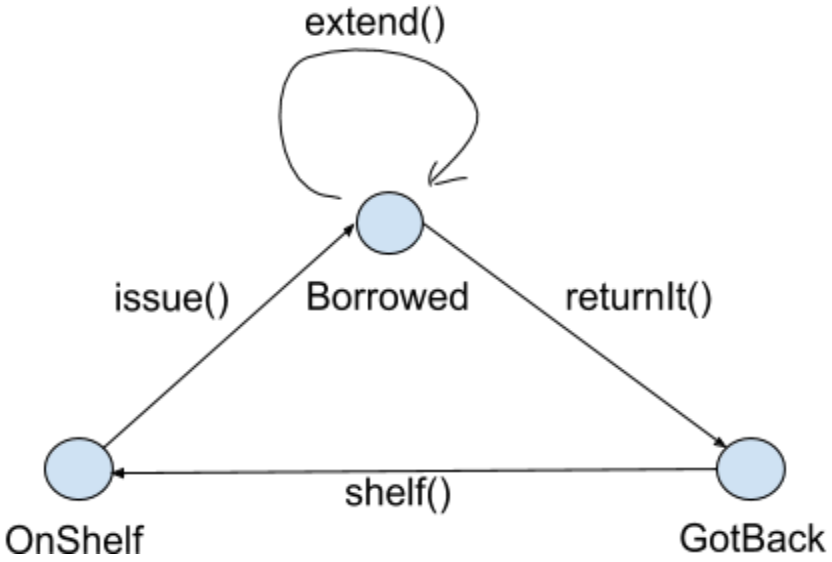

Figure 1

3. (package hw1, Observer.java, SourceObserver.java, DestObserver.java, Subject.java) Incorporate within the above code an OBSERVER pattern instance. The LibraryBook class from the above pattern should also be a SUBJECT role in the OBSERVER pattern, and support attach()/detach() methods to add and remove observers, and a Notify() method to call Update(this) on the observers ONLY when the library book changes to a different state. Create two ConcreteObservers implementing the Observer interface: SourceObserver and DestObserver. The subject and the observers should take a string called name on construction, which should be used as the basis of the observer’s equality/hash code and be printed in the observer’s toString() method. The SourceObserver should track the *source* state, and for each state transition logs “<observer_name> OBSERVED <subject_name> LEAVING STATE: <state_name>”. If SourceObserver has not previously seen the subject’s state, use UNOBSERVED for <state_name>. The DestObserver should track the *destination* state, and for each state transition logs “<observer_name> OBSERVED <subject_name> REACHING STATE: <state_name>”. When you call attach(), you should log “<observer_name> is now watching <subject_name>”, and when detach() successfully removes a subscriber, you should log “<observer_name> is no longer watching <subject_name>”. Note that the observer classes should be able to track multiple subjects, so you may need to store the tracked subjects in the observer. All the messages should be logged in a file named “*Q2&3Log.txt*” in the working directory (Please **DON’T** cover the logs you recorded in Question 2. Rather, you are supposed to append new messages). Be Careful! 10 points.