SM (M) 2018-2019 Lab 7: Design Patterns II – with inline SOLUTIONS

Revision: v2019a

Eclipse

Load in the Lab 7 workspace (available on Moodle) into your local workspace in Eclipse. Use the file -> import command to do this.

Java Generics

The Observer pattern uses and ArrayList, which is an example of generics in Java code. Understanding and using generics properly is an important part of Java programming.

Check this ArrayList API:
http://docs.oracle.com/javase/7/docs/api/index.html?java/util/ArrayList.html

And for reference, the Java Generics tutorial:
http://docs.oracle.com/javase/tutorial/extra/generics/index.html

SOLUTION:
The constructor in observer.Observable is empty. Using java generics, complete this constructor and test by running the main class.

```
this.observables = new ArrayList<Observer>();
```

Interface Polymorphism

Some of the patterns in the workspace use interface polymorphism. Identify which patterns use this, identify which lines of code this occurs on, and how it is crucial to the function of the pattern.

SOLUTION:
Observer Pattern – in Observable class line 7.
Proxy – in Main class line 48.
Read Only – in Main class line 24.

Related Patterns

This week we looked at the Proxy pattern. This pattern is closely related to the façade and adaptor patterns, but each pattern has a slightly different purpose. How are each of these patterns different, and how are they similar?

SOLUTION:
Differences: Proxy is for dealing with data stored using lightweight/heavyweight techniques. Façade is for providing an API that simplifies the use of a complex set of classes/packages. Adaptor is for incorporating an existing class into an existing inheritance hierarchy.

What is the difference between read only and immutable patterns? Why would you use one over the other?

SOLUTION:
Read-Only pattern provides an object that can't be edited, but the object's owner can still edit the object. Immutable objects cannot be edited once instantiated.

Anti Patterns

The OOSE Book describes antipatterns for Proxy, Read Only Interface, and the Observer patterns. Be sure you understand these and could identify an antipattern implementation. If you have time, test the antipattern implementation in your own workspace.

SOLUTION:
Observer should not be connected to observable, the reference should only be in the other direction. This anti-pattern does not correctly allow for different observers to be setup. Observer should not be a subclass of Observable.

Read Only Class should not be a subclass of Mutable object. This would not make it possible to prevent editing of the mutable object.

Without Proxy , code can be implemented with access to heavyweight object throughout code, which is bad.