Prepared By : Dr. Ramiro Liscano
Maintained By : Dr. Hani Sami

Lab#1 – Software Design Patterns.

Objective

In this lab you will be exposed to some design patterns by reading the pattern and trying to implement then in the Java language.

Submission Items

The following items should be submitted for this lab:

- A lab report that includes the following sections:
  - Cover sheet that includes lab title, lab members, and date.
  - Lab Deliverables as per the lab instructions
  - Observations on the lab (challenges and improvements.)

Pre-lab

There are many design patterns available for software developers. A good tutorial for design patterns can be found at https://www.tutorialspoint.com/design_pattern/ (last accessed Sept. 2025) though there are many sources in the Internet and in books such as the infamous Gang of Four book "Design Patterns – Elements of Reusable Object-Oriented Software." Please read through some of these patterns to become familiar with their structure, problem domain, and solution. Another good source of information is Wikipedia https://en.wikipedia.org/wiki/Design_Patterns (last accessed Sept. 2025.)

Background

A design pattern is:

- a standard solution to a common programming problem
- a technique for making code more flexible by making it meet certain criteria
- a design or implementation structure that achieves a particular purpose
- a high-level programming idiom
- shorthand for describing certain aspects of program organization
- connections among program components
- the shape of an object diagram or object model

They are very useful for software developers since they offer programming solutions to typical styles of programming challenges. These patterns follow a typical structure as shown below:

Pattern Name and Classification:
- Name: conveys the essence of the pattern, a good name is vital (good for design vocabulary)

Intent:
- Short statement that answers questions such as what does the design pattern do? What is its rationale? What particular design issue or problem does it address?

Also Known As:
- Other well-known names for the pattern, if any

Motivation (Forces):
- A scenario that illustrates a design problem and how class and object structures in the pattern solve the problem

Applicability:

- What are the situation in which the design pattern can be applied? What are examples of poor designs that the pattern can address?

Structure:

- A graphical representation of the classes in the pattern using a notation based on UML

Participants:

- Classes and/or objects participating in the design pattern and their responsibilities

Collaborations:

- How the participants collaborate to carry out their responsibilities

Consequences:

- How does the pattern support objectives? What are the trade-offs and results of using the pattern?

Implementation:

- What pitfalls, hints or techniques should you be aware when implementing the pattern? Are there language-specific issues?

Sample Code:

- Code fragments that illustrate how you might implement the pattern

Known Uses:

- Examples of the pattern found in real systems

Related Patterns:

- What design patterns are closely related to this one? What are the important differences?

In this lab we will use a condensed description of a pattern and propose a problem that needs to be solved using the pattern.

A. The Adapter Pattern

Adapters change the interface of a class without changing its basic functionality. For instance, they might permit interoperability between a geometry package that requires angles to be specified in radians and a client that expects to pass angles in degrees.

A garage door manufacturer has built a controller for standard doors that raise and lower at a fixed speed. They now have manufactured a premium garage door that can raise and lower at different speeds but they want to still use their pre-existing APIs for the standard garage door. **The code for the standard and premium garage doors is included with the lab.**

- Design a *GarageDoorAdapter* class that will adapt the standard garage door *open()* and *close()* operators to those that are defined by the premium garage door.
- Write a *TestGarageDoors* class to test the *GarageDoorAdapter* class.

Deliverable

- Submit the application Java files as well as screen dumps of the test results.

B. The Observer Pattern

The Observer pattern is a very common pattern in software code and it is used to define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. It is the foundation for many other more complex patterns like the publish-subscribe architecture pattern.

In this section of the lab you will use the Observer pattern to notify a set of customers registered for notifications from a particular store. **The *Store* and *Subject* classes are given to you and you need to**

- Create a an abstract *Observer* class with an abstract *update(float discount)* method that will be used by a *Customer* class.
- Create the *Customer* class that extends the *Observer* abstract class. This class should:

  - Implement the *update* method so that it will change the *Customers* discount state value and printout that customer's name and discount.
  - The parameters of the constructor of the *Customer* class should allow for the specification of the favorite *Store* and name of the customer.
  - Contains *register* and *unregister* methods that registers the customer with a particular store.
- Create a test class that instantiates a couple of *Customers*, registers the customers for notifications, unregisters them with different stores, and changes the discount for the store.

Deliverable

- Submit the application Java files as well as screen dumps of the test results.