An Assignment of Observer Pattern

***Course Title:* Design**

***Course Code*: SE 3109**

Submitted By:

Md. Mynuddin

Roll No: ASH1825007M

Year-03, Term-01

Session: 2017-18

Submitted To:

Falguni Roy

Assistant Professor

Software Engineering, IIT

Noakhali Science and Technology University

Date Of Submission: 19th February, 2020

“Software Engineering”

NOAKHALI SCIENCE AND TECHNOLOGY UNIVERSITY

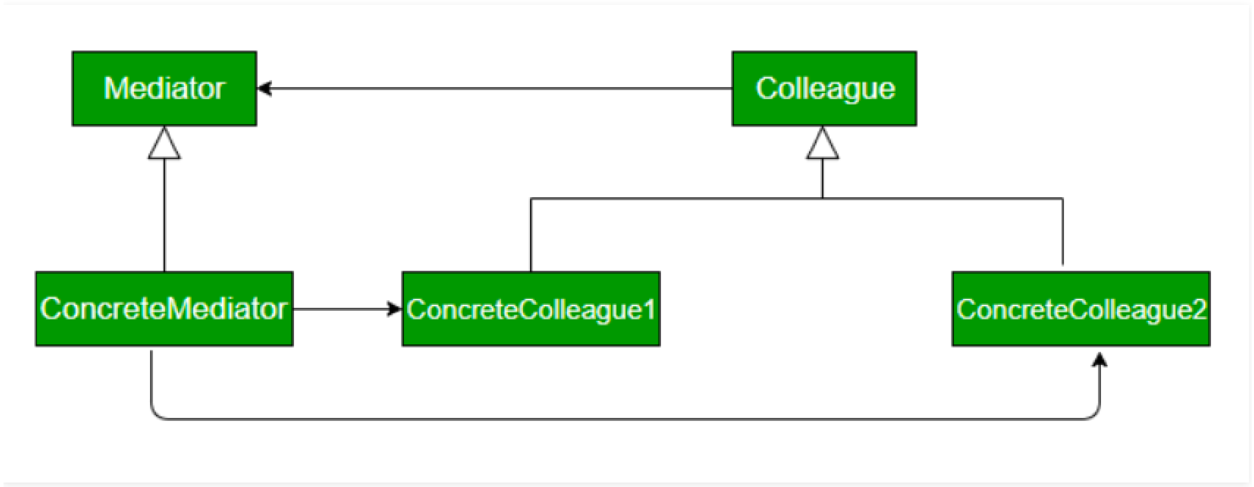
What is Mediator Pattern?

Mediator pattern is used to reduce communication complexity between multiple objects or classes. This pattern provides a mediator class which normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling.

Mediator enables decoupling of objects by introducing a layer in between so that the interaction between objects happen via the layer. If the objects interact with each other directly, the system components are tightly-coupled with each other that makes higher maintainability cost and not hard to extend. Mediator pattern focuses on providing a mediator between objects for communication and help in implementing lose-coupling between objects.

Air traffic controller is a great example of mediator pattern where the airport control room works as a mediator for communication between different flights. Mediator works as a router between objects and it can have it’s own logic to provide way of communication.

UML Diagram Mediator design pattern:



Design components:

Mediator : It defines the interface for communication between colleague objects.

Concrete Mediator: It implements the mediator interface and coordinates communication between colleague objects.

Colleague: It defines the interface for communication with other colleagues

Concrete Colleague: It implements the colleague interface and communicates with other colleagues through its mediator

Which Pattern Category It Belongs?

Mediator design pattern is one of the **behavioral design patterns**, so it deals with the behaviors of objects. Mediator design pattern is used to provide a centralized communication medium between different objects in a system.

Problem Statement:

There are few users who can chat with each other's. It's not a good idea to connect each participant to all the others because the number of connections would be really high. The best solution is to have a hub where all participants will connect; this hub is just the mediator class.

Solution:

Step 1: Mediator Pattern Interface

First of all we will create Mediator interface that will define the contract for concrete mediators.

Step 1

Create mediator class.

ChatRoom.java

```
import java.util.Date;

public class ChatRoom {
   public static void showMessage(User user, String message){
      System.out.println(new Date().toString() + " [" +
user.getName() + "] : " + message);
   }
}
```

Create user class

User.java

```
public class User {
   private String name;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public User(String name){
      this.name  = name;
   }

   public void sendMessage(String message){
      ChatRoom.showMessage(this,message);
   }
}
```

Use the *User* object to show communications between them.

MediatorPatternDemo.java

```
public class MediatorPatternDemo {
   public static void main(String[] args) {
      User robert = new User("Robert");
      User john = new User("John");

      robert.sendMessag("Hi! John!");
      john.sendMessage("Hello! Robert!");
   }
}
```

Verify the output.

```
Thu Jan 31 16:05:46 IST 2013 [Robert] : Hi! John!
Thu Jan 31 16:05:46 IST 2013 [John] : Hello! Robert!
```

One Solid principle that is covers by mediator pattern:

Mediator Pattern covers **single responsibilities principle**. Single-Responsibility Principle(SRP) states that every class must exactly do just one thing. In other words, there should not be more than one reason for us to modify a class.

It covers (SRP) because it provides

Less coupling: Since every class would be doing just one thing, there'll be far fewer dependencies

Easier to test: the code will more likely be easier to test with far fewer test cases covering the system in entirety