Report on Observer Pattern Assignment

Submitted To:

Kishan Kumar Ganguly
Assistant Professor

Submitted by:

Mehzabin Haque (BSSE 1233)

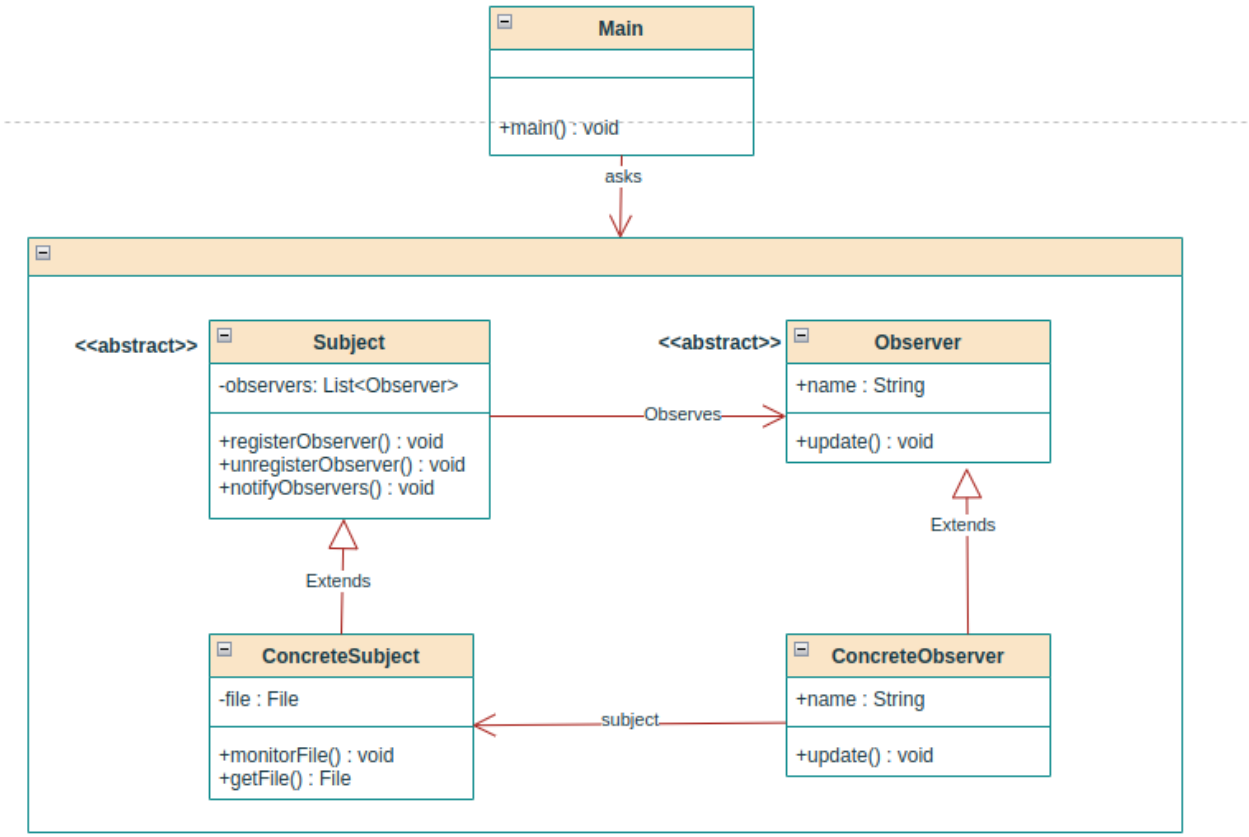
Date: Feb 15, 2023

Introduction:

The observer pattern is used when there is a one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under the behavioral pattern category. The observer pattern uses three actor classes. Subject, Observer, and Client. Subject is an object having methods to attach and detach observers to a client object. We have created an abstract class *Observer* and a concrete class *Subject* that is extending class Observer.

Here we have to create a system for file monitoring system using Oberver Design Pattern. Below are provided the UML diagram, the system's advantages and disadvantages, and a demonstration example.

UML Diagram:



Benefits:

1. **Loose coupling:** The Subject class and the Observer class are decoupled, which means that changes in one class will not affect the other classes.
2. **Real-time monitoring:** Observers will get notifications for any update in the monitored file which is actually real-time monitoring.
3. **Extensibility:** New observers can be added to the system with minimal changes to the existing code.

Limitations:

1. Only monitors a single file. To monitor multiple files, we would need to multiple create multiple instances of the ConcreteSubject class.
2. The current implementation may not be suitable for large files.

Demonstration:

Let's say we have a file on our PC called "test.txt." We want to continuously check this file for modifications. We make a fresh instance of the **ConcreteSubject** class and supply the constructor with the file path. Then, we make observer1 and observer2 as two instances of the **ConcreteObserver** class. The registerObserver() method is then used to register both observers with the **ConcreteSubject**. Both observers are now registered and will be informed whenever the file is changed.

After that, we edit "test.txt". Using the notifyObservers() function, the **ConcreteSubject** class recognizes the change and alerts observer1 and observer2. Both observers get the notification and show the information about the change, such as the file name, the kind of change that was made, and the time of the change.
This procedure can be repeated numerous times to ensure that the observers are always informed when a file is changed. Using the unregisterObserver() method, we can also unregister one of the observers and confirm that the unregistered observer is no longer receiving notifications.

This approach may prove useful in a number of circumstances, such as real-time monitoring of log files or configuration files for modifications. It makes it possible for numerous observers to receive notifications of changes as soon as they take place, which can be useful for promptly identifying and resolving problems.