Zunaira Abdul Aziz
BSSE23058

6/7

Quiz-2 (CLO-2)

BSSE-2023 (A&B) Dr. Muhammad Asif

Software Requirement Engineering

SE203-T (Spring-25)

Information Technology University (ITU) - Lahore

27-Feburary-2024 Time: 15min Marks=7

Instructions:

- Each question contains **incomplete/missing/wrong code** related to the **Observer Pattern**.
- You need to **identify and correct** the errors or complete the missing parts.
- Write the correct implementation in Java.

Question 1: Fix the missing code to ensure that observers receive updates.

```
import java.util.ArrayList;
import java.util.List;

interface Observer {
    void update(String message);
}

class ConcreteObserver implements Observer {
    private String name;

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

    @Override
    public void update(String message) {
        System.out.println(name + " received update: " + message);
    }
}

class Subject {
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        // Missing Code: Add observer to the list
    }

    public void notifyObservers(String message) {
        for (Observer observer : observers) {
            // Missing Code: Notify each observer
        }
```

observers.add(observer);

~~observers.update();~~
observers.update(message);

Question 2: The Observer interface is implemented incorrectly in ConcreteObserver. Fix the errors.

```
interface Observer {
    void update(); // Incorrect method signature
}

class ConcreteObserver implements Observer {
    private String observerName;

    public ConcreteObserver(String name) {
        this.observerName = name;
    }

    @Override
    public void update(String message) {
        System.out.println(observerName + " received: " + message);
    }

}
```

void update (String message);

Question 3: Modify the **Subject** class to allow observers to unsubscri from updates.

```
import java.util.*;

interface Observer {
    void update(String message);
}

class Subject {
    private List<Observer> observers = new ArrayList<>();
```

```
    }
}

public class ObserverPatternDemo {
    public static void main(String[] args) {
        Subject subject = new Subject();

        Observer observer1 = new ConcreteObserver("Observer1");
        Observer observer2 = new ConcreteObserver("Observer2");
        // Missing Code: Register observers
        subject.notifyObservers("Hello, Observers!");
    }

}
```

```
    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void removeObserver(Observer observer) {
        // Missing Code: Implement the remove functionality
    }

    public void notifyObservers(String message) {
        for (Observer observer : observers) {
            observer.update(message);
        }
    }

}
```

Question 4: The following code notifies observers in **reverse order**. Modify it so that observers are notified in the correct sequence.

```
class Subject {
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void notifyObservers(String message) {
        for (int i = observers.size() - 1; i >= 0; i--) {  // Incorrect notification order
            observers.get(i).update(message);
        }
    }
}
```

Question 5: The following code violates the **Observer Pattern** principles. Identify and correct the issue.

```
class Subject {
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void updateAllObservers(String message) {  // Incorrect: Method name and approach
        for (Observer observer : observers) {
            observer.update(message);
        }
    }
}
```