JAVA -Junit

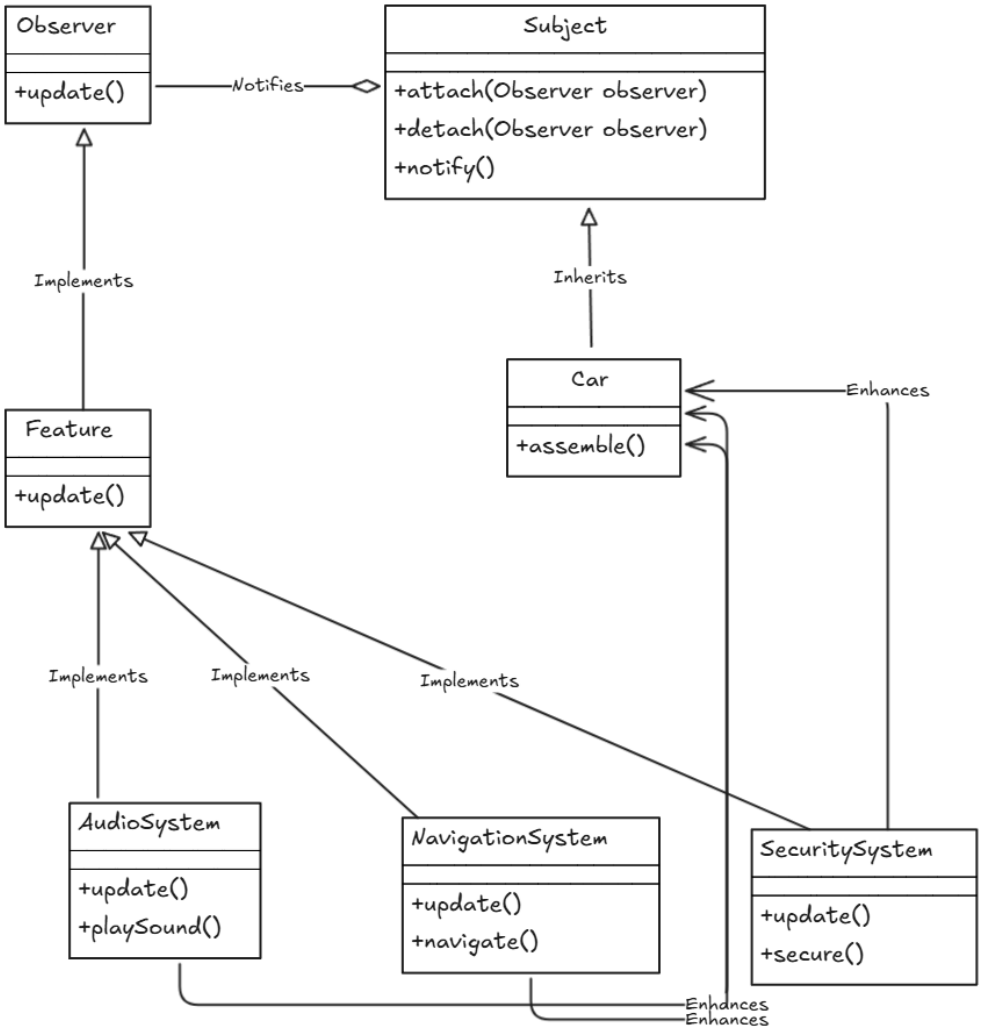
Prompt

-Observer

Use the Observer pattern to write code in Java programming language and write JUnit to test the given code that have 100% statement coverage from the requirements below. Use Observer pattern to design automotive system development by adding features to cars, such as audio system, navigation system or security systems without changing the main structure of the car.

Diagram

Observer



Observer

รอบที่2

AudioSystem

```
public class AudioSystem implements Observer {
    private String featureUpdate;

    @Override
    public void update(String featureUpdate) {
        this.featureUpdate = "Audio System: " + featureUpdate;
    }

    public String getFeatureUpdate() {
        return featureUpdate;
    }
}
```

AutomotiveSystemTest

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CarTest {

    @Test
    public void testAddAndNotifyObservers() {
        Car car = new Car();

        AudioSystem audioSystem = new AudioSystem();
        NavigationSystem navigationSystem = new NavigationSystem();
        SecuritySystem securitySystem = new SecuritySystem();

        car.addObserver(audioSystem);
```

```
        car.addObserver(navigationSystem);
        car.addObserver(securitySystem);

        car.setFeatureUpdate("Bluetooth enabled");

        assertEquals("Audio System: Bluetooth enabled", audioSystem.getFeatureUpdate());
        assertEquals("Navigation System: Bluetooth enabled", navigationSystem.getFeatureUpdate());
        assertEquals("Security System: Bluetooth enabled", securitySystem.getFeatureUpdate());
    }

    @Test
    public void testRemoveObserver() {
        Car car = new Car();

        AudioSystem audioSystem = new AudioSystem();
        NavigationSystem navigationSystem = new NavigationSystem();
        SecuritySystem securitySystem = new SecuritySystem();

        car.addObserver(audioSystem);
        car.addObserver(navigationSystem);
        car.addObserver(securitySystem);

        car.setFeatureUpdate("GPS tracking enabled");

        assertEquals("Audio System: GPS tracking enabled", audioSystem.getFeatureUpdate());
        assertEquals("Navigation System: GPS tracking enabled", navigationSystem.getFeatureUpdate());
        assertEquals("Security System: GPS tracking enabled", securitySystem.getFeatureUpdate());

        // Remove the Navigation System and update again
        car.removeObserver(navigationSystem);
        car.setFeatureUpdate("Anti-theft system enabled");

        assertEquals("Audio System: Anti-theft system enabled", audioSystem.getFeatureUpdate());
        assertEquals("Security System: Anti-theft system enabled", securitySystem.getFeatureUpdate());
```

```
        assertEquals("Navigation System: GPS tracking enabled", navigationSystem.getFeatureUpdate());  // No
update after removal
    }
}
```

NavigationSystem

```
public class NavigationSystem implements Observer {
    private String featureUpdate;

    @Override
    public void update(String featureUpdate) {
        this.featureUpdate = "Navigation System: " + featureUpdate;
    }

    public String getFeatureUpdate() {
        return featureUpdate;
    }
}
```

Observer

```
// Observer.java
public interface Observer {
    void update(String featureUpdate);
}
```

SecuritySystem

```
public class SecuritySystem implements Observer {
    private String featureUpdate;
```

```
    @Override
    public void update(String featureUpdate) {
        this.featureUpdate = "Security System: " + featureUpdate;
    }

    public String getFeatureUpdate() {
        return featureUpdate;
    }
}
```

Subject

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

public class Car {
    private List<Observer> observers = new ArrayList<>();
    private String featureUpdate;

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

    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(featureUpdate);
        }
    }
```

```
    public void setFeatureUpdate(String featureUpdate) {
        this.featureUpdate = featureUpdate;
        notifyObservers();
    }
}
```

Diagram From Code: