JAVA -Junit

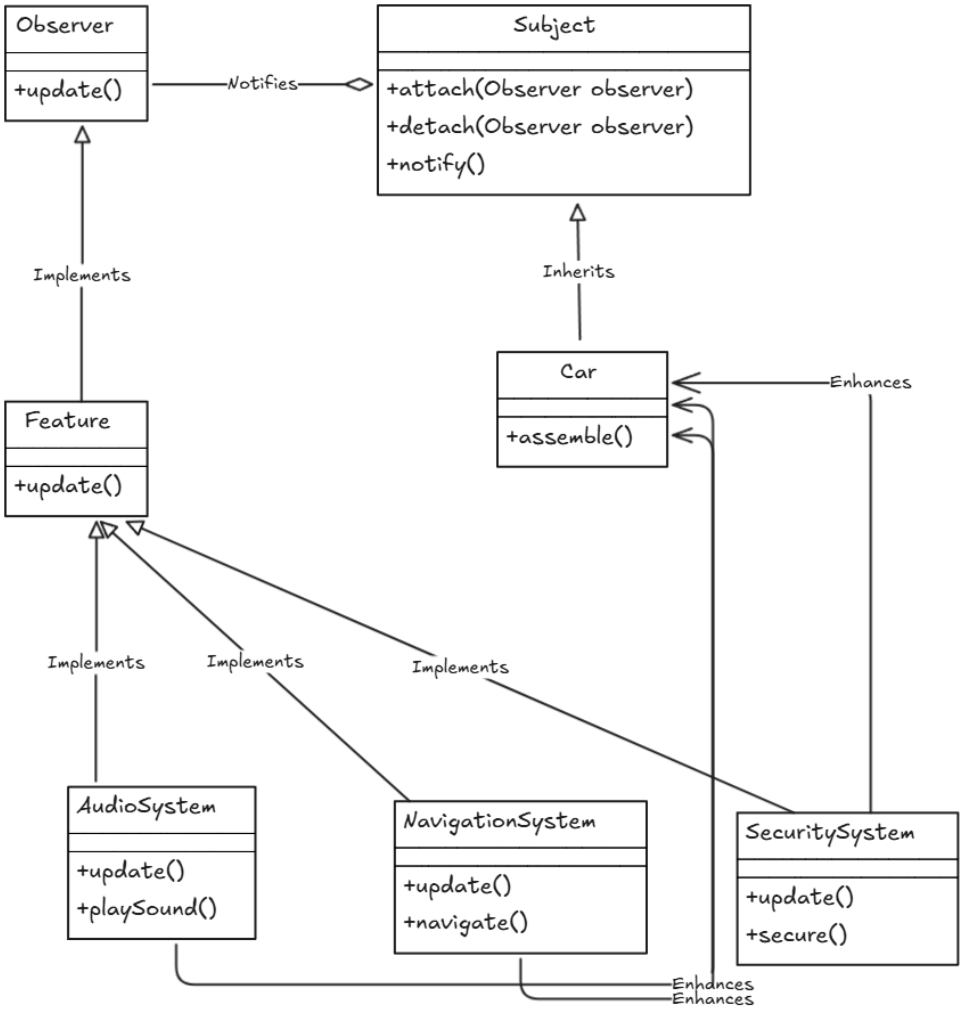
Prompt

-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

รอบที่1

AudioSystem

```
// Concrete Observer (AudioSystem)
class AudioSystem implements Feature {
    @Override
    public void update(Car car) {
        System.out.println("Audio System: Car features updated.");
    }
}
```

AutomotiveSystemTest

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTrue;

public class CarTest {

    @Test
    public void testAddFeature() {
        Car car = new CarImpl();
        AudioSystem audioSystem = new AudioSystem();
        car.addFeature(audioSystem);

        // Verify audioSystem is added
        assertTrue(car.features.contains(audioSystem));
    }

    @Test
    public void testRemoveFeature() {
        Car car = new CarImpl();
```

```
        AudioSystem audioSystem = new AudioSystem();
        car.addFeature(audioSystem);
        car.removeFeature(audioSystem);

        // Verify audioSystem is removed
        assertTrue(car.features.isEmpty());
    }

    @Test
    public void testNotifyObservers() {
        Car car = new CarImpl();
        AudioSystem audioSystem = new AudioSystem();
        NavigationSystem navigationSystem = new NavigationSystem();
        SecuritySystem securitySystem = new SecuritySystem();

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

        // Trigger notification
        car.notifyObservers();
    }
}
```

CarImpl

```
// Concrete Subject (CarImpl)
class CarImpl implements Car {
    private List<Feature> features = new ArrayList<>();

    @Override
    public void addFeature(Feature feature) {
        features.add(feature);
```

```
        notifyObservers();
    }

    @Override
    public void removeFeature(Feature feature) {
        features.remove(feature);
        notifyObservers();
    }

    @Override
    public void notifyObservers() {
        for (Feature feature : features) {
            feature.update(this);
        }
    }
}
```

NavigationSystem

```
// Concrete Observer (NavigationSystem)
class NavigationSystem implements Feature {
    @Override
    public void update(Car car) {
        System.out.println("Navigation System: Car features updated.");
    }
}
```

Observer

```
// Observer (Feature)
interface Feature {
    void update(Car car);
```

```
}
```

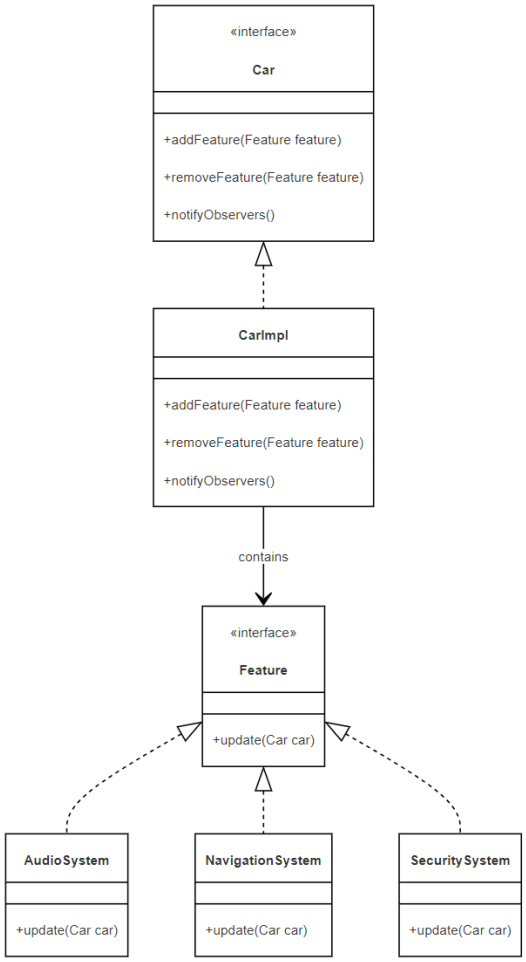
SecuritySystem

```
// Concrete Observer (SecuritySystem)
class SecuritySystem implements Feature {
    @Override
    public void update(Car car) {
        System.out.println("Security System: Car features updated.");
    }
}
```

Subject

```
// Subject (Car)
interface Car {
    void addFeature(Feature feature);
    void removeFeature(Feature feature);
    void notifyObservers();
}
```

Diagram From Code:



โค้ดที่แก้ไข:

1.Car (add)

```
List<Feature> getFeatures();
```

2.CatImp (add)

```
private List<Feature> features = new ArrayList<>(); --> private final List<Feature> features = new
ArrayList<>();
@Override
    public List<Feature> getFeatures() {
        return features;
    }
```

3.CarTest (Edit)

```
assertTrue(car.features.contains(audioSystem)); --> assertTrue(car.getFeatures().contains(audioSystem));

assertTrue(car.features.isEmpty()); --> assertTrue(car.getFeatures().contains(audioSystem));
```