JAVA -Junit

Prompt

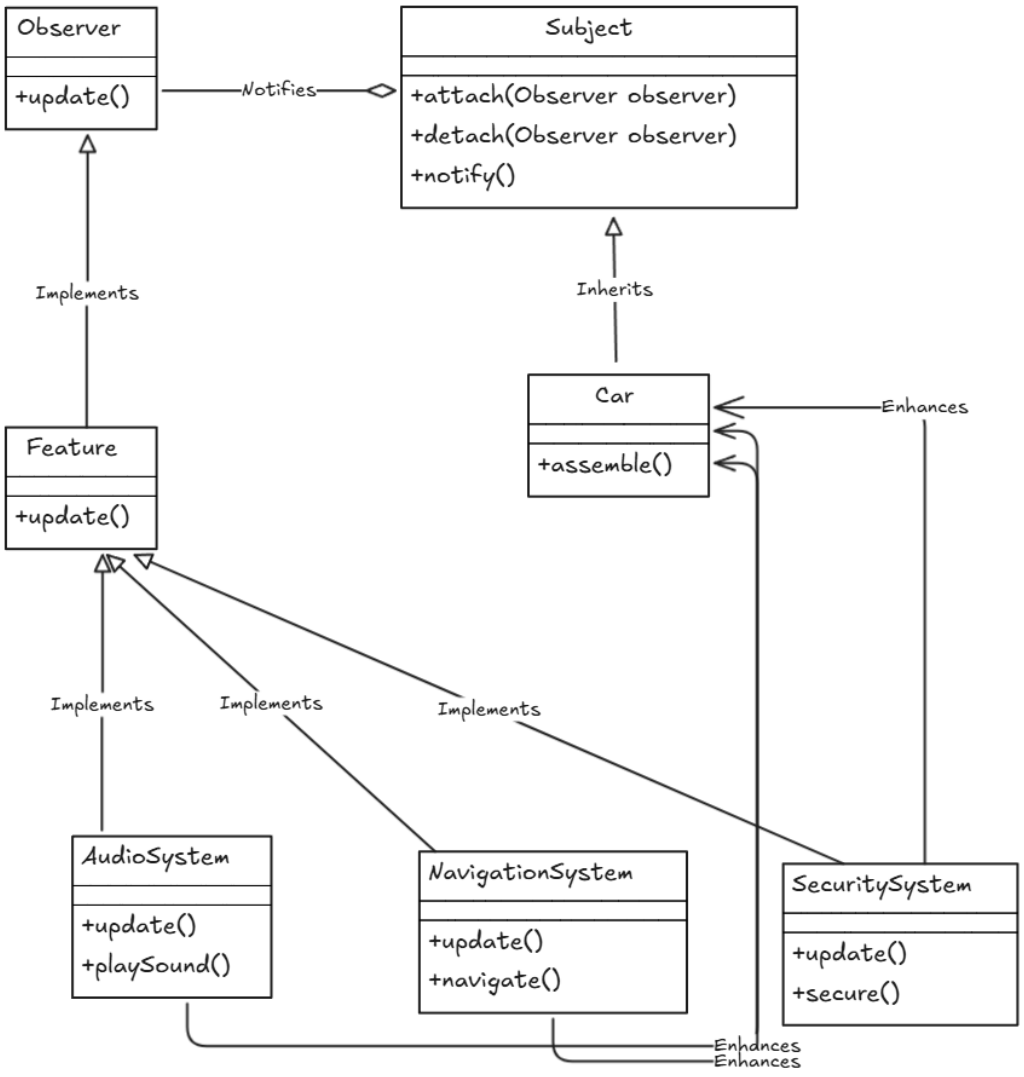
-Observer

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

รอบที่3

AudioSystem

```
// Concrete Observer (ConcreteObserver) - AudioSystem
class AudioSystem implements CarObserver {
    private boolean isInstalled = false;

    @Override
    public void update(Car car, String feature) {
        if (feature.equalsIgnoreCase("Audio System") && car instanceof BasicCar) {
            isInstalled = true;
            System.out.println("Audio System installed and ready.");
        }
    }

    // Method to check if the Audio System is installed
    public boolean isInstalled() {
        return isInstalled;
    }
}
```

AutomotiveSystemTest

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

public class CarFeatureTest {

    @Test
    void testCarFeatureAddition() {
        // Create a basic car
        BasicCar car = new BasicCar();
        // Create observers
        AudioSystem audioSystem = new AudioSystem();
        NavigationSystem navigationSystem = new NavigationSystem();
        SecuritySystem securitySystem = new SecuritySystem();

        // Attach observers to the car
        car.attach(audioSystem);
        car.attach(navigationSystem);
        car.attach(securitySystem);

        // Add features to the car
        car.addFeature("Audio System");
        car.addFeature("Navigation System");
        car.addFeature("Security System");

        // Assertions to check if features are installed
        assertTrue(car.hasFeature("Audio System"));
        assertTrue(audioSystem.isInstalled());

        assertTrue(car.hasFeature("Navigation System"));
        assertTrue(navigationSystem.isInstalled());

        assertTrue(car.hasFeature("Security System"));
        assertTrue(securitySystem.isActivated());
    }

    // Test for when no observers are attached
    @Test
    void testNoObservers() {
        BasicCar car = new BasicCar();
        car.addFeature("Audio System"); // Should not throw any exceptions
        assertTrue(true); // Placeholder, the test passes if no exception is thrown
    }
```

```
    // Test for detaching an observer
    @Test
    void testDetachObserver() {
        BasicCar car = new BasicCar();
        AudioSystem audioSystem = new AudioSystem();
        car.attach(audioSystem);
        car.detach(audioSystem);
        car.addFeature("Audio System");
        assertFalse(audioSystem.isInstalled()); // Audio system should not be installed
    }
}
```

BasicCar

```
// Concrete Observable (ConcreteSubject)
class BasicCar implements Car {
    private List<CarObserver> observers = new ArrayList<>();
    private List<String> installedFeatures = new ArrayList<>();

    @Override
    public void attach(CarObserver observer) {
        observers.add(observer);
    }

    @Override
    public void detach(CarObserver observer) {
        observers.remove(observer);
    }

    @Override
    public void notifyObservers(String feature) {
        for (CarObserver observer : observers) {
            observer.update(this, feature); // Pass the car object for context
        }
    }

    public void addFeature(String feature) {
        System.out.println("Adding feature: " + feature);
        installedFeatures.add(feature);
        notifyObservers(feature);
    }

    // Method to check if a feature is installed
    public boolean hasFeature(String feature) {
        return installedFeatures.contains(feature);
    }
}
```

NavigationSystem

```
// Concrete Observer (ConcreteObserver) - NavigationSystem
class NavigationSystem implements CarObserver {
    private boolean isInstalled = false;

    @Override
    public void update(Car car, String feature) {
        if (feature.equalsIgnoreCase("Navigation System") && car instanceof BasicCar) {
            isInstalled = true;
            System.out.println("Navigation System installed and ready.");
        }
    }

    // Method to check if the Navigation System is installed
    public boolean isInstalled() {
        return isInstalled;
    }
}
```

Observer

```
// Observer interface
interface CarObserver {
    void update(Car car, String feature); // Update method with Car object
}
```

SecuritySystem

```
// Concrete Observer (ConcreteObserver) - SecuritySystem
class SecuritySystem implements CarObserver {
    private boolean isActivated = false;

    @Override
    public void update(Car car, String feature) {
        if (feature.equalsIgnoreCase("Security System") && car instanceof BasicCar) {
            isActivated = true;
            System.out.println("Security System installed and activated.");
        }
    }

    // Method to check if the Security System is activated
    public boolean isActivated() {
        return isActivated;
    }
}
```

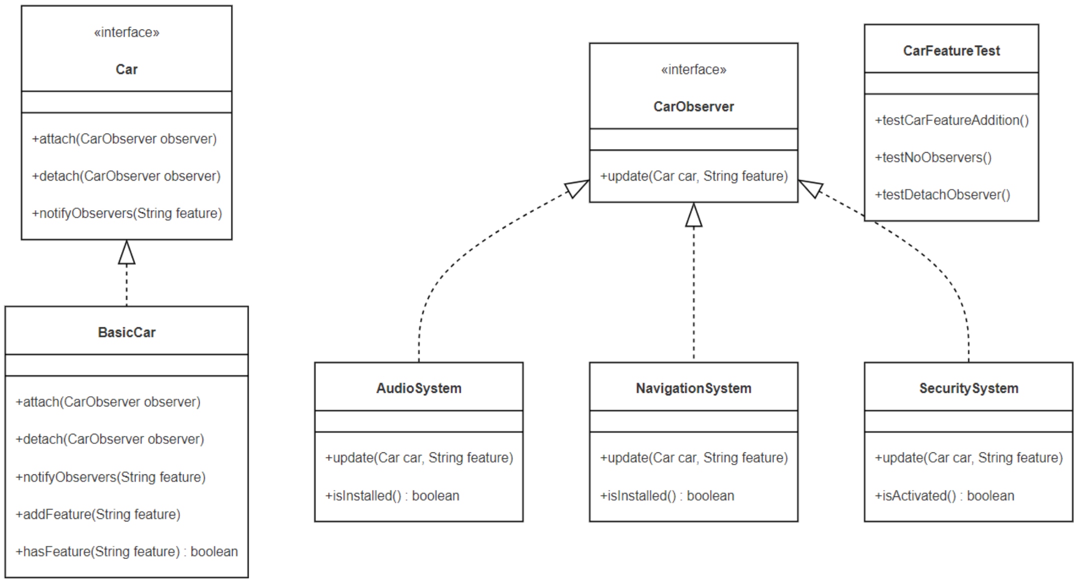
SecuritySystemDecorator

```
// Concrete Observer (ConcreteObserver) - SecuritySystem
class SecuritySystem implements CarObserver {
    private boolean isActivated = false;

    @Override
    public void update(Car car, String feature) {
        if (feature.equalsIgnoreCase("Security System") && car instanceof BasicCar) {
            isActivated = true;
            System.out.println("Security System installed and activated.");
        }
    }

    // Method to check if the Security System is activated
    public boolean isActivated() {
        return isActivated;
    }
}
```

Subject

```
// Observable interface (Subject)
interface Car {
    void attach(CarObserver observer);
    void detach(CarObserver observer);
    void notifyObservers(String feature);
}
```

Diagram From Code:


«interface»
Car
+attach(CarObserver observer)
+detach(CarObserver observer)
+notifyObservers(String feature)
BasicCar
+attach(CarObserver observer)
+detach(CarObserver observer)
+notifyObservers(String feature)
+addFeature(String feature)
+hasFeature(String feature) : boolean
«interface»
CarObserver
+update(Car car, String feature)
CarFeatureTest
+testCarFeatureAddition()
+testNoObservers()
+testDetachObserver()
AudioSystem
+update(Car car, String feature)
+isInstalled() : boolean
NavigationSystem
+update(Car car, String feature)
+isInstalled() : boolean
SecuritySystem
+update(Car car, String feature)
+isActivated() : boolean

โค้ดที่แก้ไข:

1. BasicCar.java (add)

private List<CarObserver> observers = new ArrayList<>(); --> private final List<CarObserver> observers = new ArrayList<>();

private List<String> installedFeatures = new ArrayList<>(); --> private final List<String> installedFeatures = new ArrayList<>();