JAVA -Junit

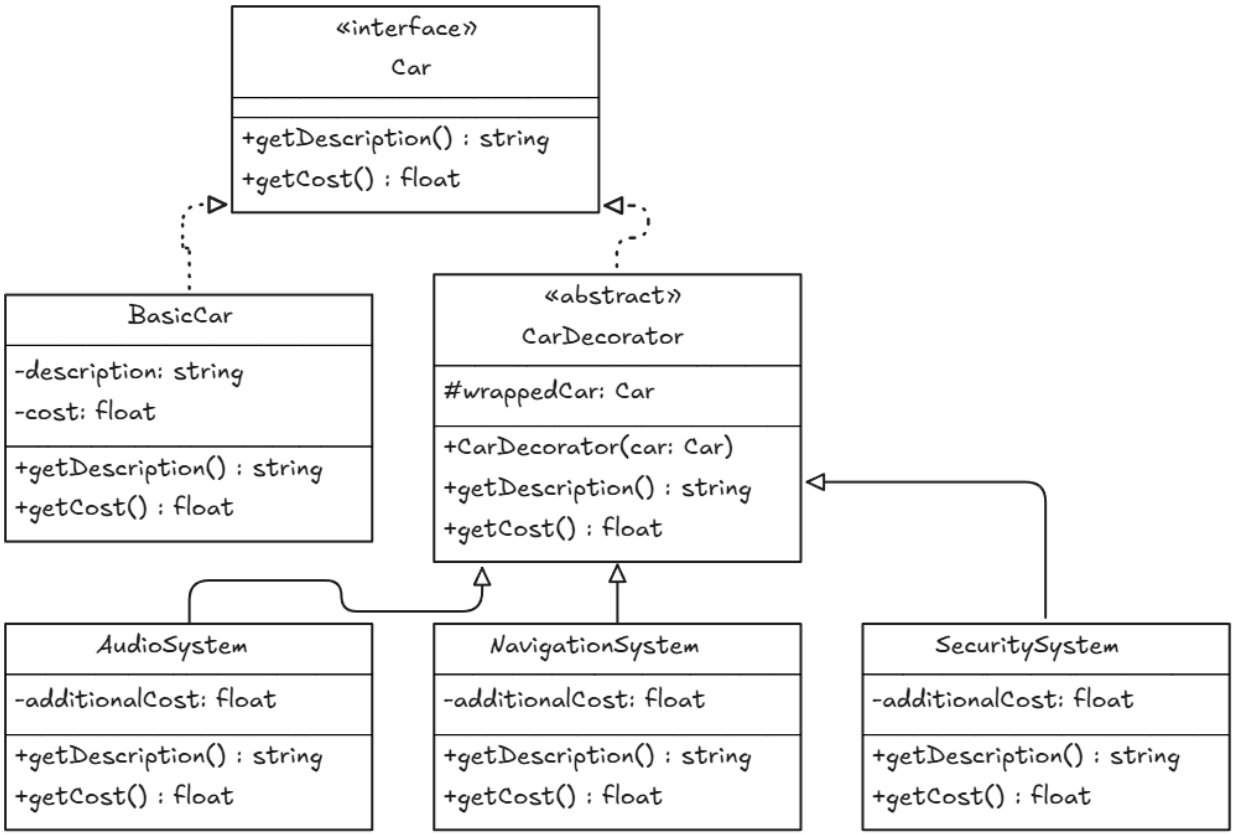
Prompt

-Decorator

Use the Decorator 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 Decorator 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

Decorator



Github Copilot

Code:

Decorator

รอบที่1

AudioSystemDecorator

```
public class AudioSystem extends CarDecorator {
    public AudioSystem(Car car) {
        super(car);
    }

    @Override
    public String getDescription() {
        return decoratedCar.getDescription() + ", Audio System";
    }

    @Override
    public double getCost() {
        return decoratedCar.getCost() + 1500;
    }
}
```

BasicCar

```
public class BasicCar implements Car {
    @Override
    public String getDescription() {
        return "Basic Car";
    }

    @Override
    public double getCost() {
```

```
        return 20000;
    }
}
```

Car

```
public interface Car {
    String getDescription();
    double getCost();
}
```

CarDecorator

```
public abstract class CarDecorator implements Car {
    protected Car decoratedCar;

    public CarDecorator(Car car) {
        this.decoratedCar = car;
    }

    @Override
    public String getDescription() {
        return decoratedCar.getDescription();
    }

    @Override
    public double getCost() {
        return decoratedCar.getCost();
    }
}
```

CarTest

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

public class CarDecoratorTest {

    @Test
    public void testBasicCar() {
        Car car = new BasicCar();
        assertEquals("Basic Car", car.getDescription());
        assertEquals(20000, car.getCost());
    }

    @Test
    public void testCarWithAudioSystem() {
        Car car = new AudioSystem(new BasicCar());
        assertEquals("Basic Car, Audio System", car.getDescription());
        assertEquals(21500, car.getCost());
    }

    @Test
    public void testCarWithNavigationSystem() {
        Car car = new NavigationSystem(new BasicCar());
        assertEquals("Basic Car, Navigation System", car.getDescription());
        assertEquals(22000, car.getCost());
    }

    @Test
    public void testCarWithSecuritySystem() {
        Car car = new SecuritySystem(new BasicCar());
        assertEquals("Basic Car, Security System", car.getDescription());
        assertEquals(22500, car.getCost());
    }
```

```
    @Test
    public void testCarWithAllFeatures() {
        Car car = new SecuritySystem(new NavigationSystem(new AudioSystem(new BasicCar())));
        assertEquals("Basic Car, Audio System, Navigation System, Security System", car.getDescription());
        assertEquals(26000, car.getCost());
    }
}
```

NavigationSystemDecorator

```
public class NavigationSystem extends CarDecorator {
    public NavigationSystem(Car car) {
        super(car);
    }

    @Override
    public String getDescription() {
        return decoratedCar.getDescription() + ", Navigation System";
    }

    @Override
    public double getCost() {
        return decoratedCar.getCost() + 2000;
    }
}
```

SecuritySystemDecorator

```
public class SecuritySystem extends CarDecorator {
    public SecuritySystem(Car car) {
        super(car);
    }
```

```
    @Override
    public String getDescription() {
        return decoratedCar.getDescription() + ", Security System";
    }

    @Override
    public double getCost() {
        return decoratedCar.getCost() + 2500;
    }
}
```

Diagram From Code: