JAVA -Junit

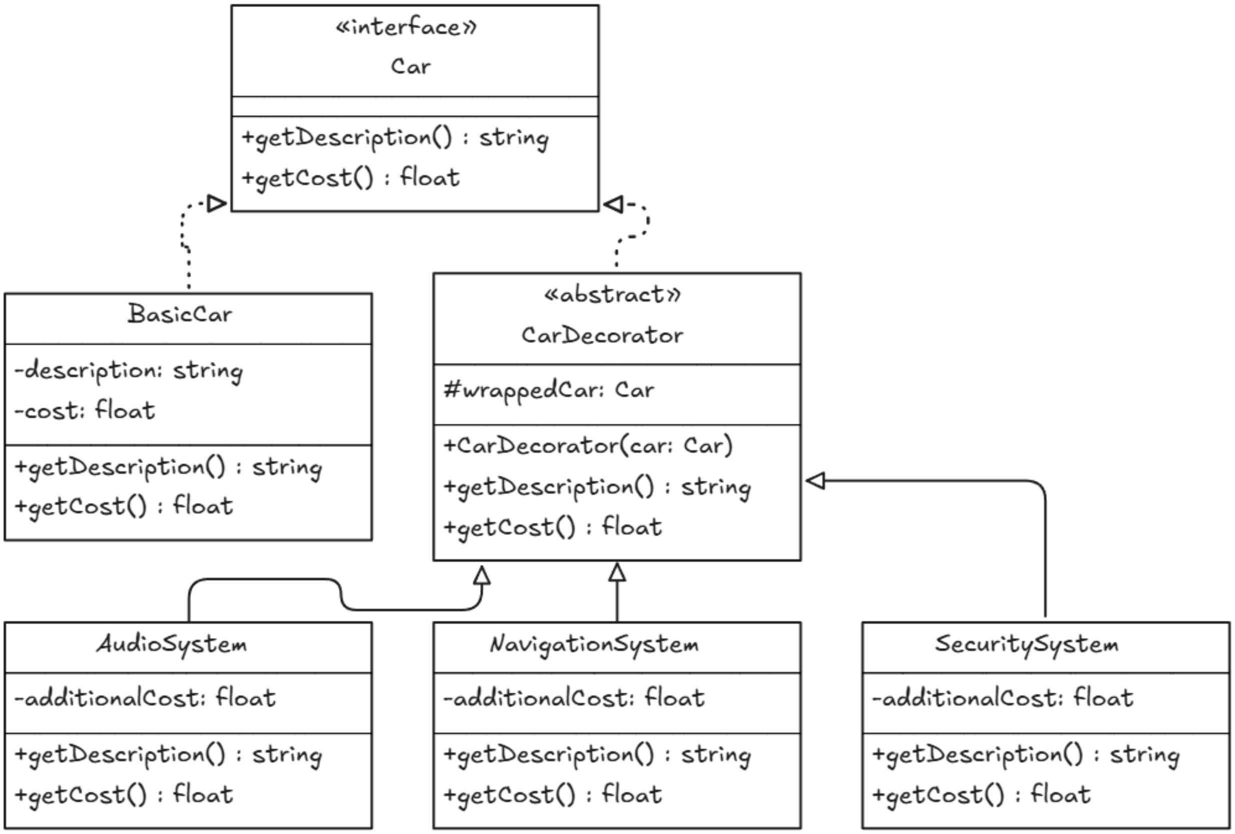
Prompt

-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



ChatGPT4o

Code:

Decorator

รอบที่2

AudioSystemDecorator

```
public class AudioSystemDecorator extends CarDecorator {

    public AudioSystemDecorator(Car decoratedCar) {
        super(decoratedCar);
    }

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

    @Override
    public double cost() {
        return super.cost() + 1500.0;
    }
}
```

BasicCar

```
public class BasicCar implements Car {

    @Override
    public String assemble() {
        return "Basic Car";
    }
    @Override
    public double cost() {
        return 20000.0; // Base price of the car
    }
}
```

Car

```
public interface Car {

    String assemble();

    double cost();

}
```

CarDecorator

```
public abstract class CarDecorator implements Car {

    protected Car decoratedCar;



    public CarDecorator(Car decoratedCar) {

        this.decoratedCar = decoratedCar;

    }



    @Override

    public String assemble() {
        return decoratedCar.assemble();
    }

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

CarTest

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CarDecoratorTest {

    @Test
    public void testBasicCar() {
        Car car = new BasicCar();
        assertEquals("Basic Car", car.assemble());
        assertEquals(20000.0, car.cost(), 0.01);
    }

```

```
@Test
public void testCarWithAudioSystem() {
    Car car = new AudioSystemDecorator(new BasicCar());
    assertEquals("Basic Car, Audio System", car.assemble());
    assertEquals(21500.0, car.cost(), 0.01);
}

@Test
public void testCarWithNavigationSystem() {
    Car car = new NavigationSystemDecorator(new BasicCar());
    assertEquals("Basic Car, Navigation System", car.assemble());
    assertEquals(22500.0, car.cost(), 0.01);
}

@Test
public void testCarWithSecuritySystem() {
    Car car = new SecuritySystemDecorator(new BasicCar());
    assertEquals("Basic Car, Security System", car.assemble());
    assertEquals(23000.0, car.cost(), 0.01);
}

@Test
public void testCarWithAllFeatures() {
        Car car = new SecuritySystemDecorator(
                    new NavigationSystemDecorator(
                    new AudioSystemDecorator(
                    new BasicCar())));
        assertEquals("Basic Car, Audio System, Navigation System, Security System", car.assemble());
        assertEquals(27000.0, car.cost(), 0.01);
    }
}
    public void testCarWithNavigationSystem() {
        Car navigationCar = new NavigationSystemDecorator(new BasicCar());
        assertEquals("Basic Car, Navigation System", navigationCar.assemble());
        assertEquals(22500.0, navigationCar.cost(), 0.01);
    }

    @Test
    public void testCarWithSecuritySystem() {
        Car securityCar = new SecuritySystemDecorator(new BasicCar());
        assertEquals("Basic Car, Security System", securityCar.assemble());
        assertEquals(23000.0, securityCar.cost(), 0.01);
    }

    @Test
    public void testCarWithAllFeatures() {
        Car fullyLoadedCar = new SecuritySystemDecorator(
                                new NavigationSystemDecorator(
                                new AudioSystemDecorator(
                                new BasicCar())));
        assertEquals("Basic Car, Audio System, Navigation System, Security System", fullyLoadedCar.assemble());
        assertEquals(27000.0, fullyLoadedCar.cost(), 0.01);
    }
}
```

NavigationSystemDecorator

```
public class NavigationSystemDecorator extends CarDecorator {

    public NavigationSystemDecorator(Car decoratedCar) {
        super(decoratedCar);
    }

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

    public double cost() {

        return super.cost() + 2500.0;

    }

}
```

SecuritySystemDecorator

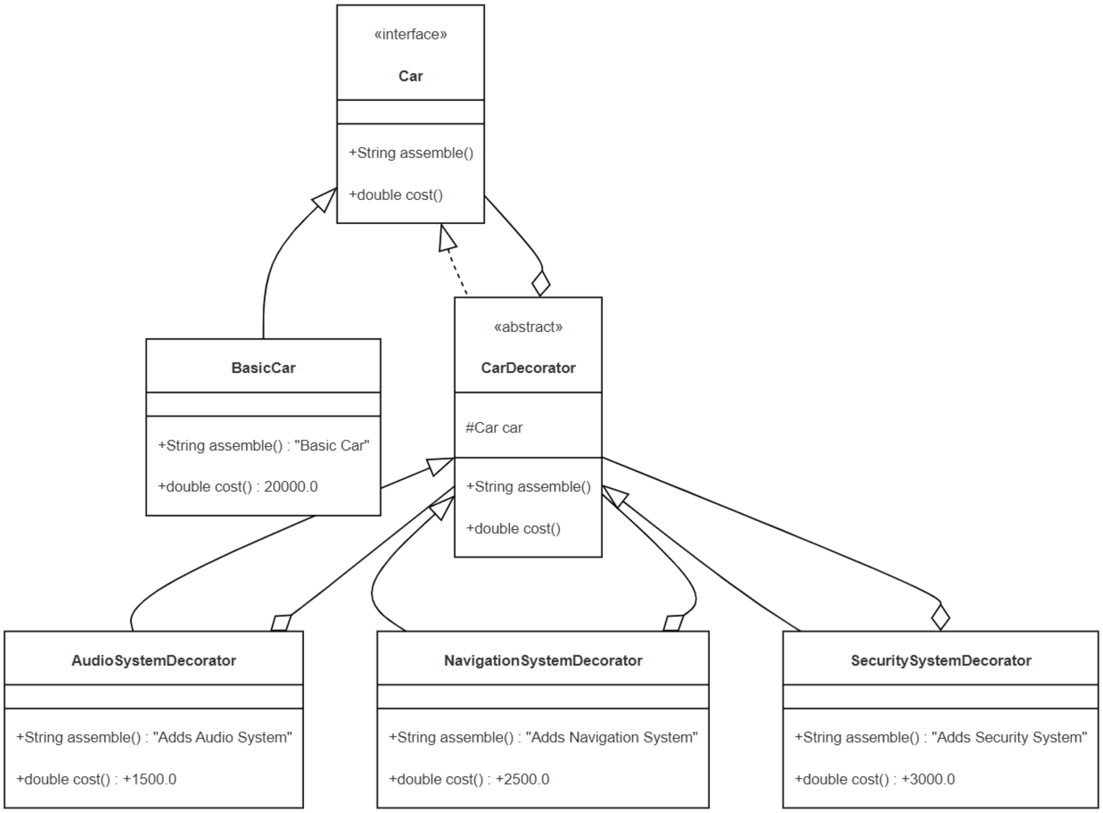
```
public class SecuritySystemDecorator extends CarDecorator {


    public SecuritySystemDecorator(Car decoratedCar) {
        super(decoratedCar);
    }


    @Override
    public String assemble() {
        return super.assemble() + ", Security System";
    }


    @Override
    public double cost() {
        return super.cost() + 3000.0;
    }
}
```

Diagram From Code:



«interface»
Car
+String assemble()
+double cost()
«abstract»
CarDecorator
#Car car
+String assemble()
+double cost()
BasicCar
+String assemble() : "Basic Car"
+double cost() : 20000.0
AudioSystemDecorator
+String assemble() : "Adds Audio System"
+double cost() : +1500.0
NavigationSystemDecorator
+String assemble() : "Adds Navigation System"
+double cost() : +2500.0
SecuritySystemDecorator
+String assemble() : "Adds Security System"
+double cost() : +3000.0