JAVA -Junit

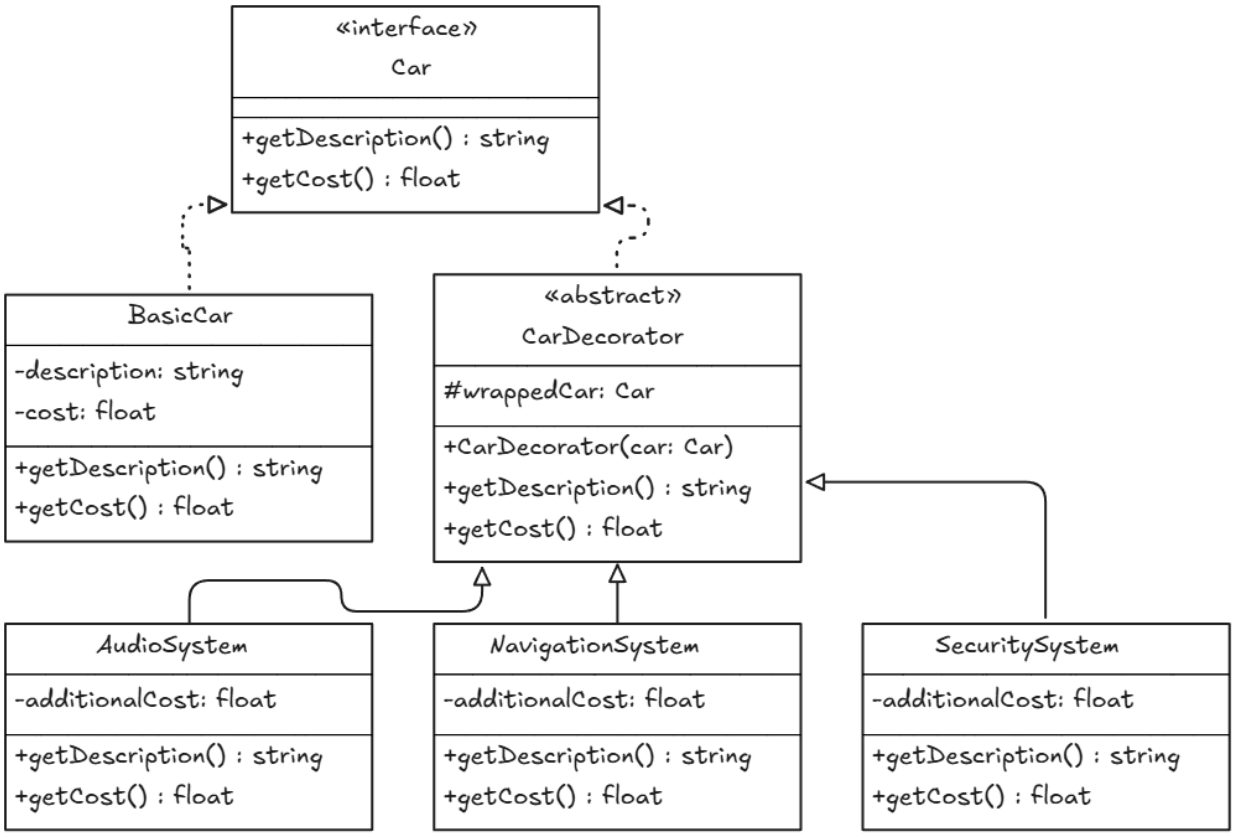
Prompt

-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



GEMINI 1.5pro

Code:

Decorator

รอบที่3

AudioSystemDecorator

```
// Concrete Decorator class for Audio System
class AudioSystem extends CarDecorator {
    public AudioSystem(Car decoratedCar) {
        super(decoratedCar);
    }

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

    @Override
    public double getCost() {
        return super.getCost() + 500;
    }
}
```

BasicCar

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

    @Override
    public double getCost() {
        return 20000;
```

```
    }
}
```

Car

```
// Interface for Car
interface Car {
    String getDescription();
    double getCost();
}
```

CarDecorator

```
// Abstract Decorator class implementing Car interface
abstract class CarDecorator implements Car {
    protected Car decoratedCar;

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

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

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

CarTest

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

class CarDecoratorTest {

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

    @Test
    void testCarWithAudioSystem() {
        Car car = new AudioSystem(new BasicCar());
        assertEquals("Basic Car, Audio System", car.getDescription());
        assertEquals(20500, car.getCost());
    }

    @Test
    void testCarWithNavigationSystem() {
        Car car = new NavigationSystem(new BasicCar());
        assertEquals("Basic Car, Navigation System", car.getDescription());
        assertEquals(21000, car.getCost());
    }

    @Test
    void testCarWithSecuritySystem() {
        Car car = new SecuritySystem(new BasicCar());
        assertEquals("Basic Car, Security System", car.getDescription());
        assertEquals(20800, car.getCost());
```

```
    }

    @Test
    void testCarWithAllFeatures() {
        Car car = new SecuritySystem(new NavigationSystem(new AudioSystem(new BasicCar())));
        assertEquals("Basic Car, Audio System, Navigation System, Security System", car.getDescription());
        assertEquals(22300, car.getCost());
    }
}
```

NavigationSystemDecorator

```
// Concrete Decorator class for Navigation System
class NavigationSystem extends CarDecorator {
    public NavigationSystem(Car decoratedCar) {
        super(decoratedCar);
    }

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

    @Override
    public double getCost() {
        return super.getCost() + 1000;
    }
}
```

SecuritySystemDecorator

```
// Concrete Decorator class for Security System
```

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

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

    @Override
    public double getCost() {
        return super.getCost() + 800;
    }
}
```

Diagram From Code: