JAVA -Junit

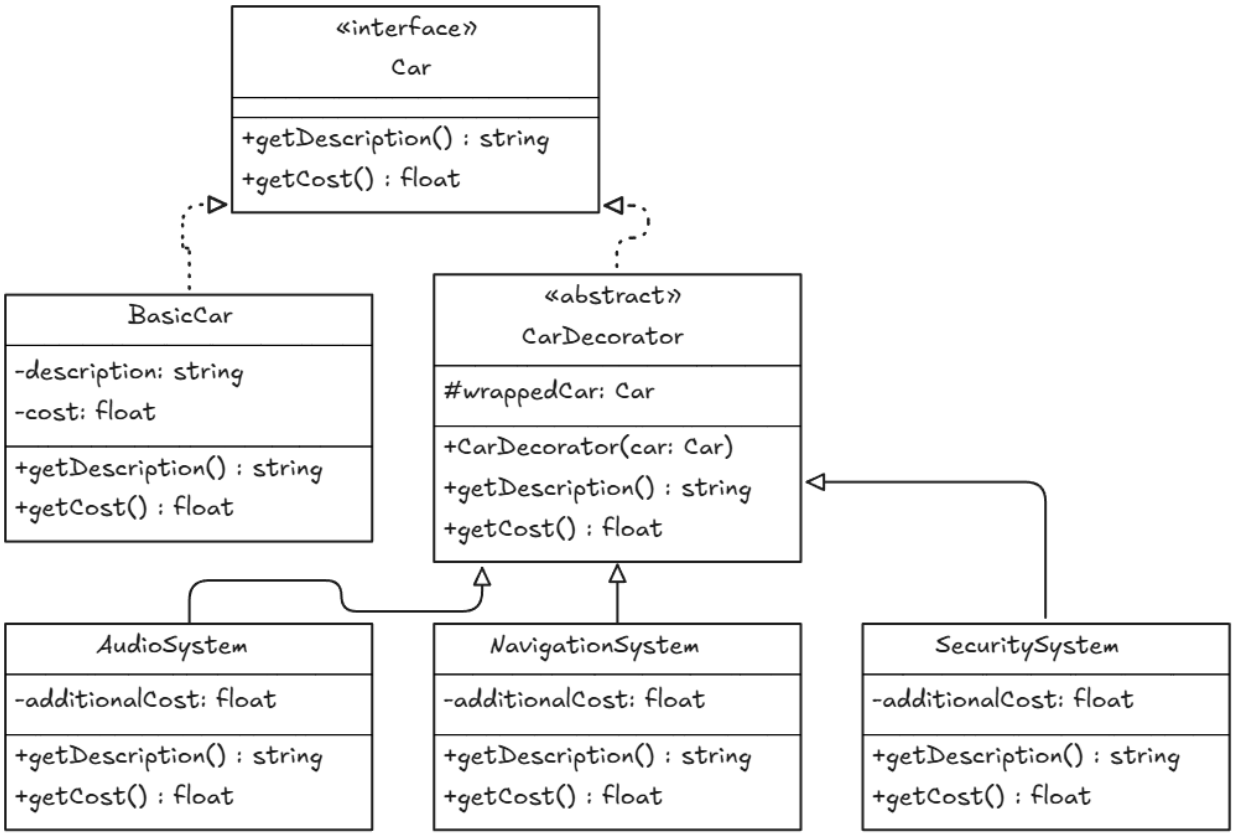
Prompt

-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 1.5flash

Code:

Decorator

รอบที่3

AudioSystemDecorator

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

    @Override
    public void assemble() {
        super.assemble();
        System.out.println("Audio System added.");
    }
}
```

BasicCar

```
class BasicCar implements Car {
    @Override
    public void assemble() {
        System.out.println("Basic Car assembled.");
    }
}
```

Car

```
interface Car {
    void assemble();
```

```
}
```

CarDecorator

```
abstract class CarDecorator implements Car {
    protected Car car;

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

    @Override
    public void assemble() {
        car.assemble();
    }
}
```

CarTest

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

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

class CarDecoratorTest {

    @Test
    void testBasicCar() {
        Car car = new BasicCar();
        car.assemble();
        // Expected output: Basic Car assembled.
    }

    @Test
```

```
    void testAudioSystemDecorator() {
        Car car = new AudioSystemDecorator(new BasicCar());
        car.assemble();
        // Expected output: Basic Car assembled.
        //               Audio System added.
    }

    @Test
    void testNavigationSystemDecorator() {
        Car car = new NavigationSystemDecorator(new BasicCar());
        car.assemble();
        // Expected output: Basic Car assembled.
        //               Navigation System added.
    }

    @Test
    void testSecuritySystemDecorator() {
        Car car = new SecuritySystemDecorator(new BasicCar());
        car.assemble();
        // Expected output: Basic Car assembled.
        //               Security System added.
    }

    @Test
    void testMultipleDecorators() {
        Car car = new SecuritySystemDecorator(new NavigationSystemDecorator(new
AudioSystemDecorator(new BasicCar())));
        car.assemble();
        // Expected output: Basic Car assembled.
        //               Audio System added.
        //               Navigation System added.
        //               Security System added.
    }
}
```

NavigationSystemDecorator

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

    @Override
    public void assemble() {
        super.assemble();
        System.out.println("Navigation System added.");
    }
}
```

SecuritySystemDecorator

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

    @Override
    public void assemble() {
        super.assemble();
        System.out.println("Security System added.");
    }
}
```

Diagram From Code: