Applying the Decorator Design Pattern to Sensor Data Fusion

Frida Cano Falcón

Java Backend Academy MTY
August 2024

Week 3

August 30, 2024

Introduction

The Decorator design pattern is a structural pattern that allows additional functionality to be added to objects dynamically, without altering the objects themselves. This pattern is particularly useful in scenarios where objects need to be extended or modified at runtime. In the context of self-driving vehicles, sensor fusion is a key application where multiple sensor data streams are combined to provide enriched information for decision-making. This report illustrates how the Decorator pattern can be used to enhance sensor data and fuse information from various sensors, including cameras, LiDAR, and radar.

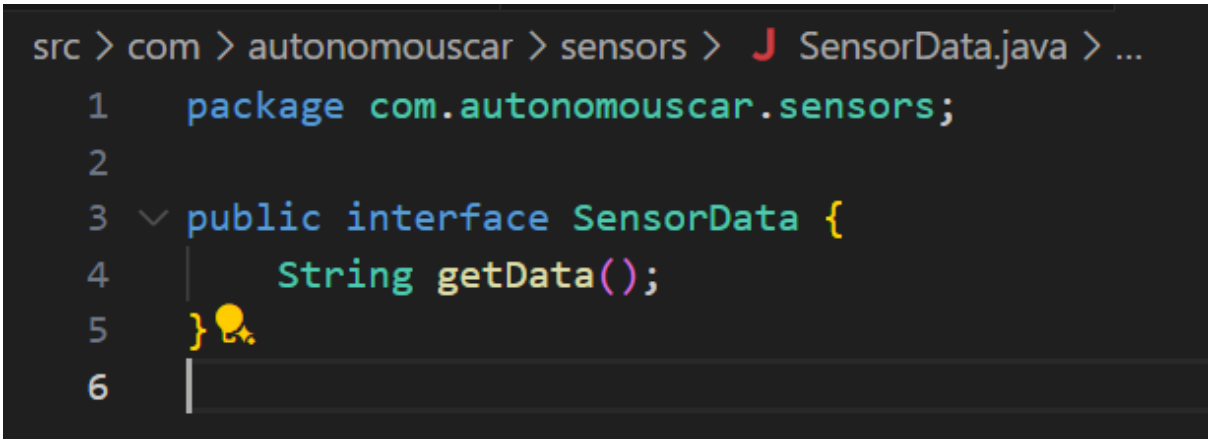
Project Overview

The project showcases the application of the Decorator pattern to sensor data fusion for a self-driving vehicle system. The key classes and their roles are:

1. *SensorData.java*: Defines the interface for sensor data.
2. *CameraData.java:* Concrete implementation of *SensorData* for camera data.
3. *LiDARData.java*: Concrete implementation of *SensorData* for LiDAR data.
4. *RadarData.java*: Concrete implementation of *SensorData* for radar data.
5. *SensorDataDecorator.java*: Abstract decorator class for *SensorData*.
6. *EnhancedCameraData.java*: Concrete decorator adding functionality to camera data.
7. *FusedData.java*: Concrete decorator for fusing sensor data.
8. *VisionSystem.java*: Demonstrates the use of the Decorator pattern in a sensor data fusion scenario.
9. *SensorDataTest.java*: JUnit 5 test cases to verify the functionality of the implemented classes.

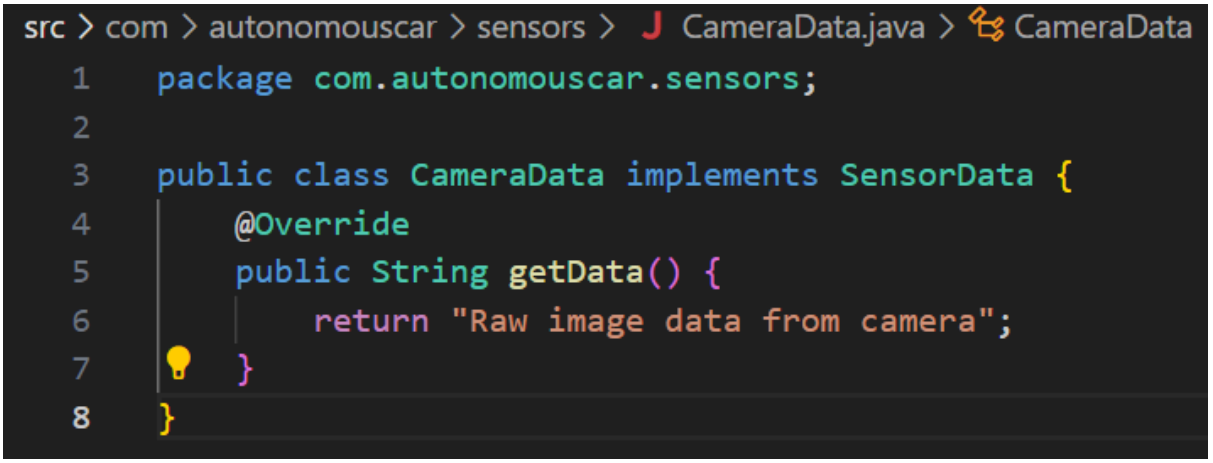
1. SensorData.java

The *SensorData* interface defines a common method for retrieving sensor data.



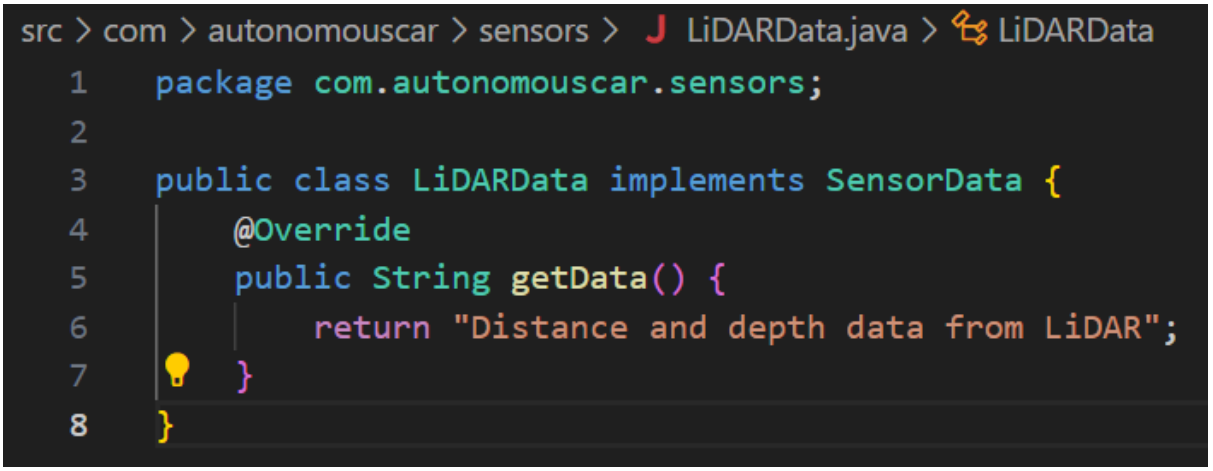
2. CameraData.java

CameraData implements the *SensorData* interface, providing raw data from a camera sensor.



3. LiDARData.java

LiDARData implements the *SensorData* interface, providing data from a LiDAR sensor.



4. RadarData.java

RadarData implements the *SensorData* interface, providing data from a radar sensor.

src > com > autonomouscar > sensors > RadarData.java > RadarData

```java
package com.autonomouscar.sensors;

public class RadarData implements SensorData {
    @Override
    public String getData() {
        return "Speed and distance data from radar";
    }
}
```

5. SensorDataDecorator.java

The *SensorDataDecorator* abstract class serves as the base for all decorators, adding additional behavior to *SensorData*.

src > com > autonomouscar > sensors > SensorDataDecorator.java > ...

```java
package com.autonomouscar.sensors;

public abstract class SensorDataDecorator implements SensorData {
    protected SensorData decoratedSensorData;

    public SensorDataDecorator(SensorData decoratedSensorData) {
        this.decoratedSensorData = decoratedSensorData;
    }
}
```

6. EnhancedCameraData.java

EnhancedCameraData extends *SensorDataDecorator* to add additional processing to camera data.

src > com > autonomouscar > sensors > J EnhancedCameraData.java > ...

```java
package com.autonomouscar.sensors;

public class EnhancedCameraData extends SensorDataDecorator {

    public EnhancedCameraData(SensorData decoratedSensorData) {
        super(decoratedSensorData);
    }

    @Override
    public String getData() {
        return decoratedSensorData.getData() + " with object recognition";
    }
}
```

7. FusedData.java

FusedData extends *SensorDataDecorator* to combine data from multiple sensors.

src > com > autonomouscar > sensors > J FusedData.java > ...

```java
package com.autonomouscar.sensors;

public class FusedData extends SensorDataDecorator {

    public FusedData(SensorData decoratedSensorData) {
        super(decoratedSensorData);
    }

    @Override
    public String getData() {
        // Assume this method combines data from multiple sources if needed.
        return "Fused Data: " + decoratedSensorData.getData();
    }
}
```

8. VisionSystem.java

The *VisionSystem* class demonstrates how to use the decorators to process and fuse sensor data.

src > com > autonomouscar > sensors > VisionSystem.java > VisionSystem > main(String[])

```java
package com.autonomouscar.sensors;

public class VisionSystem {
    Run | Debug
    public static void main(String[] args) {
        // Create individual sensor data
        SensorData cameraData = new CameraData();
        SensorData lidarData = new LiDARData();
        SensorData radarData = new RadarData();

        // Wrap the camera data with enhanced features
        SensorData enhancedCameraData = new EnhancedCameraData(cameraData);

        // Fuse different sensor data
        SensorData fusedCameraData = new FusedData(enhancedCameraData);
        SensorData fusedLidarData = new FusedData(lidarData);
        SensorData fusedRadarData = new FusedData(radarData);

        // Print combined data
        System.out.println("Camera Data: " + fusedCameraData.getData());
        System.out.println("LiDAR Data: " + fusedLidarData.getData());
        System.out.println("Radar Data: " + fusedRadarData.getData());
    }
}
```

9. SensorDataTest.java

JUnit 5 test cases to verify the functionality of the implemented classes.

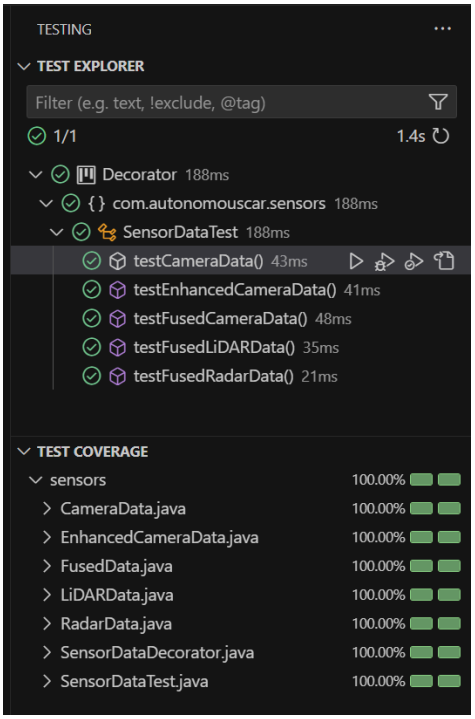
src > com > autonomouscar > sensors > SensorDataTest.java > ...

```java
package com.autonomouscar.sensors;
import org.junit.jupiter.api.Test;

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

public class SensorDataTest {

    @Test
    public void testCameraData() {
        SensorData cameraData = new CameraData();
        assertEquals("Raw image data from camera", cameraData.getData());
    }

    @Test
    public void testEnhancedCameraData() {
        SensorData cameraData = new CameraData();
        SensorData enhancedCameraData = new EnhancedCameraData(cameraData);
        assertEquals("Raw image data from camera with object recognition", enhancedCameraData.getData());
    }

    @Test
    public void testFusedCameraData() {
        SensorData cameraData = new CameraData();
        SensorData enhancedCameraData = new EnhancedCameraData(cameraData);
        SensorData fusedCameraData = new FusedData(enhancedCameraData);

        assertEquals("Fused Data: Raw image data from camera with object recognition", fusedCameraData.getData());
    }
```

```java
    @Test
    public void testFusedLiDARData() {
        SensorData lidarData = new LiDARData();
        SensorData fusedLidarData = new FusedData(lidarData);

        assertEquals("Fused Data: Distance and depth data from LiDAR", fusedLidarData.getData());
    }

    @Test
    public void testFusedRadarData() {
        SensorData radarData = new RadarData();
        SensorData fusedRadarData = new FusedData(radarData);

        assertEquals("Fused Data: Speed and distance data from radar", fusedRadarData.getData());
    }
}
```

JUnit5

Several tests were implemented to verify the functionality. Here is the proof of the tests with coverage.



Conclusion

The project effectively demonstrates the application of the Decorator design pattern to sensor data fusion. By using decorators, we were able to enhance individual sensor data and combine data from multiple sources to provide a richer and more informative output. This approach illustrates how the Decorator pattern can be used to extend the functionality of objects dynamically while keeping the codebase clean and manageable. The provided implementation and tests validate the correct application of the pattern and the expected behavior of the sensor data fusion system.