CS355 Software Engineering Process

Lab Week 04

Design Pattern – Builder Pattern

In this lab, we are going to look at another software design pattern: Builder Pattern. In general, the details of object construction, such as instantiating and initializing the components that make up the object, are kept within the object, often as part of its constructor. This approach is suitable as long as the object under construction is simple and the object construction process is definite and always produces the same representation of the object.

However, this design may not be effective when the object being created is complex and the series of steps constituting the object creation process can be implemented in different ways, thus producing different representations of the object. Because the different implementations of the construction process are all kept within the object, the object can become bulky (construction bloat) and less modular. Subsequently, adding a new implementation or making changes to an existing implementation requires changes to the existing code.

To illustrate the use of the Builder Pattern, let's help a Car company which shows its different cars using a graphical model to its customers. The company has a graphical tool which displays the car on the screen. The requirement of the tool is to provide a car object to it. The car object should contain the car's specifications. The graphical tool uses these specifications to display the car. The company has classified its cars into different classifications like Sedan or Sports Car. There is only one car object, and our job is to create the car object according to the classification. For example, for a Sedan car, a car object according to the sedan specification should be built or, if a sports car is required, then a car object according to the sports car specification should be built. Currently, the Company wants only these two types of cars, but it may require other types of cars also in the future.

We will create two different builders, one of each classification, i.e., for sedan and sports cars. These two builders will help us inbuilding the car object according to its specification.

The Car class.

```
public class Car {
        private String bodyStyle;
        private String power;
        private String engine;
        private String breaks;
        private String seats;
        private String windows;
        private String fuelType;
        private String carType;

        public Car (String carType){this.carType = carType;}

       public String getBodyStyle() {return bodyStyle;}

        public void setBodyStyle(String bodyStyle) {this.bodyStyle = bodyStyle;}

        public String getPower() {return power;}

// see Car.java for the full source code.
```

The CarBuilder is the builder interface contains set of common methods used to build the car object and its components.

```
public interface CarBuilder {
        public void buildBodyStyle();
        public void buildPower();
        public void buildEngine();
        public void buildBreaks();
        public void buildSeats();
        public void buildWindows();
        public void buildFuelType();
        public Car getCar();

}
```

The getCar method is used to return the final car object to the client after its construction.

Let's see two implementations of the CarBuilder interface for sedan car.

```
public class SedanCarBuilder implements CarBuilder{
	private final Car car = new Car("SEDAN");

	@Override
	public void buildBodyStyle() {
		car.setBodyStyle("External dimensions: overall length (inches): 202.9, " +
				"overall width (inches): 76.2, overall height (inches): 60.7, wheelbase
(inches): 112.9," +
				" front track (inches): 65.3, rear track (inches): 65.5 and curb to curb
turning circle (feet): 39.5");
	}


	@Override
	public void buildPower(){
		car.setPower("285 hp @ 6,500 rpm; 253 ft lb of torque @ 4,000 rpm");
	}


	@Override
	public void buildEngine() {
		car.setEngine("3.5L Duramax V 6 DOHC");

	}

// see SedanCarBuilder.java for the full source code.
```

Now let's test the builder.

```
public class TestBuilderPattern {

	public static void main(String[] args) {
		CarBuilder carBuilder = new SedanCarBuilder();
		CarDirector director = new CarDirector(carBuilder);
		director.build();
		Car car = carBuilder.getCar();
		System.out.println(car);}
}
```

When to use the Builder Pattern

Use the Builder pattern when

- The algorithm for creating a complex object should be independent of the parts that make up the object and how they're assembled
- The construction process must allow different representations for the object that's constructed.

Exercise 1: Download the java files from Moodle and include them in a Java Project; compile and run the source code; try to understand the source code.

Exercise 2: Create a SportCarBuilder that implements CarBuilder interface. You may use SedanCarBuilder as a template.

Exercise 3: Use SportCarBuilder to construct a sport car object. You may use the code in TestBuilderPattern as a template.

Note that you need to override every single method defined in the CarBuilder interface.