Assignment 2: Builder Pattern

What is Builder Design Pattern?

Builder is a creational design pattern that lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.

Intend

- ✓ Separate the construction of a complex object from its representation so that the same construction process can create different representations.
- ✓ Parse a complex representation, create one of several targets.

Context: *Car*

In the context of a car, the builder design pattern can be used to create different cars with different customizations using the same construction process. The builder class would have methods for setting different car properties, such as the **Colour, Type of fuel, Transmission Control, Infotainment system, No. of woofers**. The client code would then use these methods to create the desired car object.

Implementation (Code)

Car.java

```
public class Car {
    private String color;
    private String fuel;
    private String control;
    private String assistant;
    private int woofers;


    public Car(String color, String fuel, String control, String assistant, int woofers) {
        super();
        this.color = color;
        this.fuel = fuel;
        this.control = control;
        this.assistant = assistant;
        this.woofers = woofers;
    }


    public String toString() {
        return "Car [Color: " + color + ", Fuel: " + fuel + ", Control: " + control + ", Assistant: " +
assistant + ", Woofers: " + woofers + "]";
    }
}
```

BuildCar.java

```
public class BuildCar {
    private String color;
    private String fuel;
    private String control;
    private String assistant;
    private int woofers;


    public BuildCar setColor(String color){
        this.color = color;
        return this;
    }
    public BuildCar setFuel(String fuel){
        this.fuel = fuel;
        return this;
    }
    public BuildCar setControl(String control) {
        this.control = control;
        return this;
    }
    public BuildCar setAssistent(String assistant){
        this.assistant = assistant;
        return this;
    }
    public BuildCar setWoofers(int woofers) {
        this.woofers = woofers;
        return this;
    }


    public Car getCar(){
        return new Car(color, fuel, control, assistant, woofers);
    }
}
```

Showroom.java

```
public class Showroom {
    public static void main(String[] args) {
        Car c = new BuildCar().setColor("Red").setFuel("Electric").setControl("Automatic")
                .setAssistent("Android Auto").setWoofers(8).getCar();
        System.out.println(c);
    }
}
```

Output

```
Car [Color: Red, Fuel: Electric, Control: Automatic,
 Assistant: Android Auto, Woofers: 8]
```

Applicability

1. Use the Builder pattern to get rid of a “telescoping constructor”.
2. Use the Builder pattern when you want your code to be able to create different representations of some product (for example sports and SUV car).
3. Use the Builder to construct Complex objects.