The builder pattern is a design pattern in Object Oriented Programming that is used to construct complex objects step by step.
Let's say I create a public class car:

```java
public class Car {
    private String id;
    private String brand;
    private String model;
    private String engine;

    public Car(String id, String brand, String model, String engine) { //public constructor
        this.id = id;
        this.brand = brand;
        this.model = model;
        this.engine = engine;
    }
```

Here the public class name is car which has 4 private member variables for attributes like id,brand,model,engine. This class also provides a constructor which allows you to create a CAR object by specifying all values at once.

```java
public String getId() {
    return id;
}

public String getBrand() {
    return brand;
}

public String getModel() {
    return model;
}

public String getEngine() {
    return engine;
}
```

This is part of the getter methods. The car class provides getter methods like getId, getBrand, etc to access the attributes of a Car object. These are there so we can retrieve the attributes values after a Car object has been constructed.

```java
public static class CarBuilder {
    private String id;
    private String brand;
    private String model;
    private String engine;

    public CarBuilder setID(String id) {
        this.id = id;
        return this;
    }

    public CarBuilder setBrand(String brand) {
        this.brand = brand;
        return this;
    }

    public CarBuilder setModel(String model) {
        this.model = model;
        return this;
    }

    public CarBuilder setEngine(String engine) {
        this.engine = engine;
        return this;
    }

    public Car build() {
        return new Car(id, brand, model, engine);
    }
}
```

Here we have the CarBuilder Class. This class is a nested static class within the Car class. The CarBuilder class has its own set of member variables for the attributes of the Car being constructed.
This class also allows you to set the values for Attributes Individually. These methods follow a fluent interface style meaning that they return the CarBuilder instance by themselves allowing you to chain method calls. For example you can set attributes like id using the setid.
The CarBuild class also provides a build() method, when you call this method it will build the car with the attributes that I have set in the builder class and then return it.

There are a lot of benefits of using the builder Pattern because it eliminates the need of multiple overload constructors and improves code readability so the developer or client always knows what is in the class or builder class. It allows you to create immutable(unable to be changed) objects, as the Car class does not provide setter methods for modifying its attributes after construction. This is beneficial because nobody can change the state of this object which means that they can not be altered. They also can be cached, because once it's created and cached it can be reused without any concerts which could improve the performance of the application because the computer does not need to re-create them. If you don't use the builder pattern then you could face several issues like Complexity and maintenance. With Complexity I mean that , If I would create this code without a builder pattern and I would have so many attributes, then I would need to create more constructors, which could lead to complex class. Since I have more constructors , in case of maintaining the code or improving I would need to modify all constructors individually which could introduce errors. In summary, while using constructors to create objects is a common approach, the builder pattern provides a more elegant and flexible

solution, especially when dealing with complex objects with many attributes or optional parameters.