Builder

Luís Leal - 103511

April 24th, 2024

When should we use this pattern?

This creational design pattern should be used when:

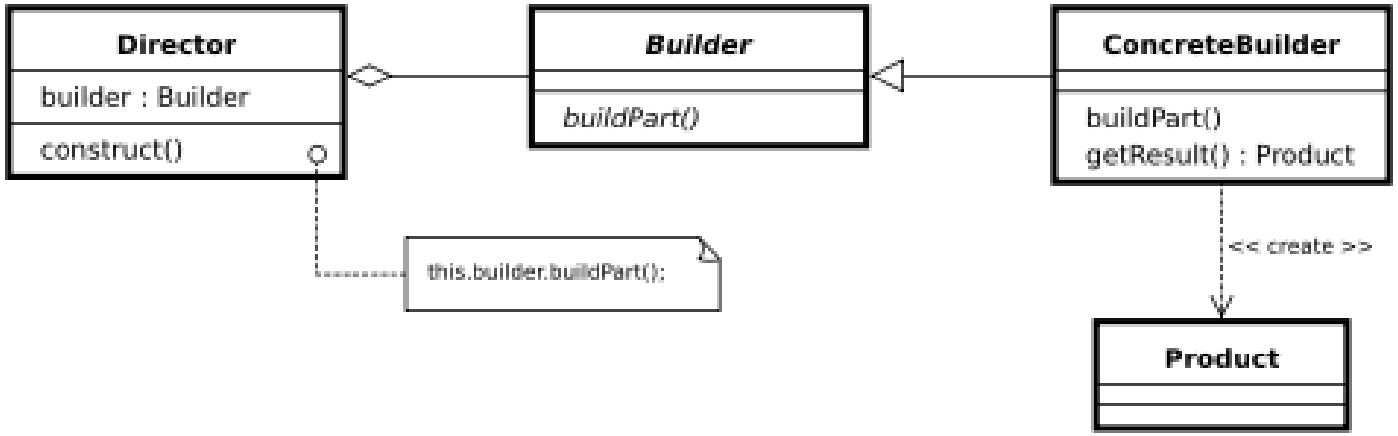
- Object creation has a **large number of optional parameters**
- Our goal is to create **immutable objects**
- The **object construction** is very **complex**
- Certain combinations of parameters are invalid (**consistency**)
- We want to improve **readability** and **maintainability**

How to implement this pattern?

1. Define common construction steps for building all available product representations
2. Declare the previous steps in the base builder interface
3. Create a concrete class for each of the product representations and implement their construction steps
4. Think about creating a director class, which may encapsulate various ways to construct a product using the same builder object
5. Client code creates both the builder and the director objects and, before construction starts, the client must pass a builder object to the director
6. The construction result can be obtained directly from the director only if all products follow the same interface

(Base) Class Structure

- **Builder** interface - declares product construction steps
- **Concrete Builders** - provide different implementations of the construction steps
- **Products –** resulting objects
- **Director** - defines the order in which to call construction steps
- **Client** - associate one of the builder objects with the director



Code Example(s)

```java
public interface CakeBuilder {
    public void setCakeShape(Shape shape);

    public void addCakeLayer();

    public void addCreamLayer();

    public void addTopLayer();

    public void addTopping();

    public void addMessage(String m);

    public void createCake();

    public Cake getCake();
}
```

```java
public class CakeMaster {
    CakeBuilder cakeBuilder;

    public void setCakeBuilder(CakeBuilder cakeBuilder) {
        this.cakeBuilder = cakeBuilder;
    }

    public CakeBuilder getCakeBuilder() {
        return cakeBuilder;
    }

    public void createCake(String message) {
        cakeBuilder.createCake();
        cakeBuilder.addMessage(message);
        cakeBuilder.addCakeLayer(); // Compose the perfect Dough
        cakeBuilder.addTopLayer(); // Give it a little sugar
        cakeBuilder.addTopping(); // End it with a topping
    }

    public void createCake(int num_layers, String message) {
        cakeBuilder.createCake();
        cakeBuilder.addMessage(message);
        for (int i = 0; i < num_layers; i++) {
            cakeBuilder.addCakeLayer(); // Compose the perfect Dough
        }
        if (num_layers > 1)
            cakeBuilder.addCreamLayer();
        cakeBuilder.addTopLayer(); // Give it a little sugar

    }

    public void createCake(Shape shape, int num_layers, String message) {
        cakeBuilder.createCake();
        cakeBuilder.addMessage(message);
        cakeBuilder.setCakeShape(shape);
        for (int i = 0; i < num_layers; i++) {
            cakeBuilder.addCakeLayer(); // Compose the perfect Dough
        }
        if (num_layers > 1)
            cakeBuilder.addCreamLayer();
        cakeBuilder.addTopLayer(); // Give it a little sugar
    }

    public Cake getCake() {
        return cakeBuilder.getCake();
    }
}
```

```java
public class ChocolateCakeBuilder implements CakeBuilder {

    private Cake chocolateCake;

    @Override
    public void setCakeShape(Shape shape) {
        chocolateCake.setShape(shape);
    }

    @Override
    public void addCakeLayer() {
        chocolateCake.addLayer();
    }

    @Override
    public void addCreamLayer() {
        chocolateCake.setMidLayerCream(Cream.Whipped_Cream);
    }

    @Override
    public void addTopLayer() {
        chocolateCake.setTopLayerCream(Cream.Chocolate);
    }

    @Override
    public void addTopping() {
        chocolateCake.setTopping(Topping.Fruit);
    }

    @Override
    public void addMessage(String m) {
        chocolateCake.setMessage(m);
    }

    @Override
    public void createCake() {
        chocolateCake = new Cake("Soft chocolate cake");
    }

    @Override
    public Cake getCake() {
        return chocolateCake;
    }
}
```

```java
public class Cake {
    public void addLayer() {
    }

    @Override
    public String toString() {
        if (numCakeLayers == 1 || this.midLayerCream == null) {
            if (this.topping == null)
                return this.cakeLayer + " cake with " + this.numCakeLayers + " layers, topped with "
                        + this.topLayerCream + " cream. Message says: \"" + this.message + "\".";
            else
                return this.cakeLayer + " cake with " + this.numCakeLayers + " layers, topped with "
                        + this.topLayerCream + " cream and " + this.topping + ". Message says: \"" + this.message
                        + "\".";
        }

        if (this.topping == null)
            return this.cakeLayer + " cake with " + this.numCakeLayers + " layers and " + this.midLayerCream
                    + " cream, topped with " + this.topLayerCream + " cream. Message says: \"" + this.message + "\".";
        else
            return this.cakeLayer + " cake with " + this.numCakeLayers + " layers and " + this.midLayerCream
                    + " cream, topped with " + this.topLayerCream + " cream and " + this.topping + ". Message says: \""
                    + this.message + "\".";
    }
}
```

```java
public class Lab06ex1 {
    Run | Debug
    public static void main(String[] args) {
        CakeMaster cakeMaster = new CakeMaster();

        CakeBuilder chocolate = new ChocolateCakeBuilder();
        cakeMaster.setCakeBuilder(chocolate);
        cakeMaster.createCake(message:"Congratulations"); // 1 cake layer
        Cake cake = cakeMaster.getCake();
        System.out.println("Your cake is ready: " + cake);
```