

Design Patterns - Creational - Builder Pattern

Saturday, May 21, 2016

4:46 PM

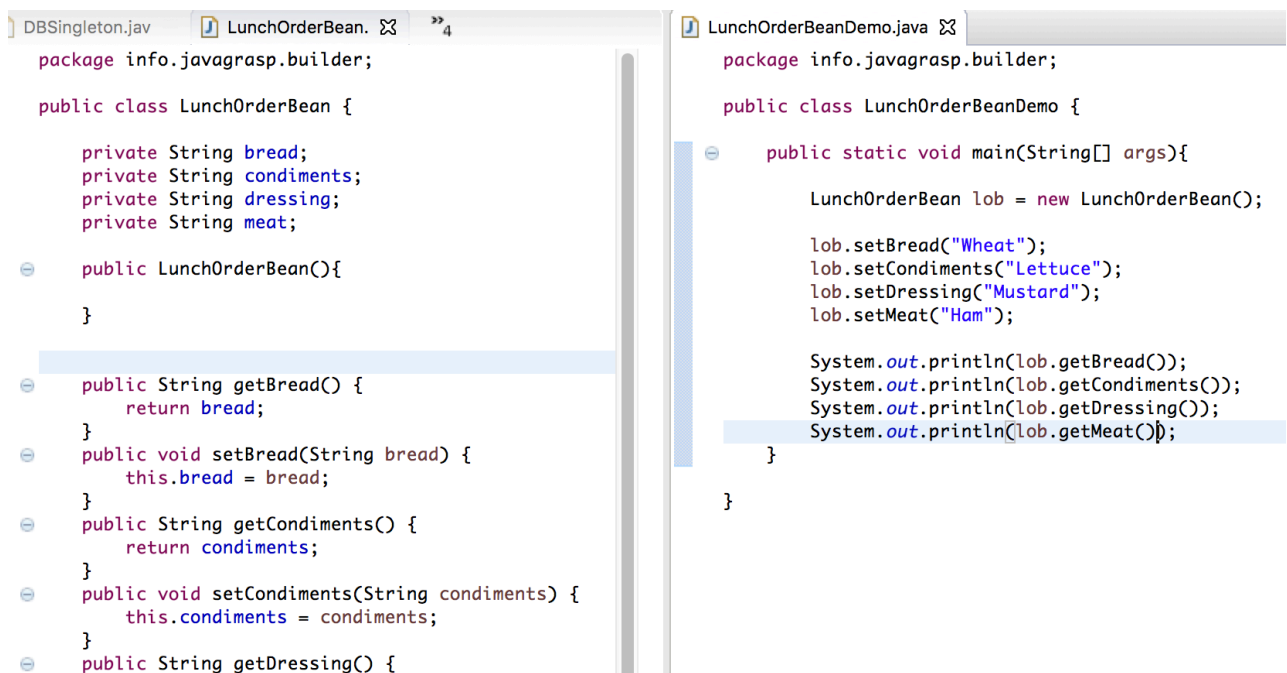
When to use:

- Lot of constructors (ways) to create the objects
- Lots of parameters involved
- Lots of getters and setters
- You want immutable objects (When all you can do is just get something off a object and NO setters)
 - o So when there is lot of setters are there you give a chance that after the object creation they will be mutable
- So when there are lot of different constructors for creating an instance of an objects the concept is called **Telescoping Constructors**

Example : StringBuilder

Problem Statement:

Till now I don't have the idea of what the problem here is



```
DBSingleton.java  LunchOrderBean.java  LunchOrderBeanDemo.java

package info.javagrasp.builder;

public class LunchOrderBean {

    private String bread;
    private String condiments;
    private String dressing;
    private String meat;

    public LunchOrderBean(){

    }

    public String getBread() {
        return bread;
    }
    public void setBread(String bread) {
        this.bread = bread;
    }
    public String getCondiments() {
        return condiments;
    }
    public void setCondiments(String condiments) {
        this.condiments = condiments;
    }
    public String getDressing() {

    }
}
```

```
LunchOrderBeanDemo.java

package info.javagrasp.builder;

public class LunchOrderBeanDemo {

    public static void main(String[] args){

        LunchOrderBean lob = new LunchOrderBean();

        lob.setBread("Wheat");
        lob.setCondiments("Lettuce");
        lob.setDressing("Mustard");
        lob.setMeat("Ham");

        System.out.println(lob.getBread());
        System.out.println(lob.getCondiments());
        System.out.println(lob.getDressing());
        System.out.println(lob.getMeat());

    }
}
```

```

        return dressing;
    }
    public void setDressing(String dressing) {
        this.dressing = dressing;
    }
    public String getMeat() {
        return meat;
    }
    public void setMeat(String meat) {
        this.meat = meat;
    }
}

```

So now I will build objects using constructors instead of setting all the beans. So getting rid of all setter methods.

```

// LunchOrderTeloBeanDemo.java
package info.javagrasp.builder;

public class LunchOrderTeloBeanDemo {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        LunchOrderTeloBean lunchOrderTelo =
            new LunchOrderTeloBean("Wheat", "Lettuce", "Mustard", "Ham");
        System.out.println(lunchOrderTelo.getBread());
        System.out.println(lunchOrderTelo.getCondiments());
        System.out.println(lunchOrderTelo.getDressing());
        System.out.println(lunchOrderTelo.getMeat());
    }
}

// LunchOrderTeloBean.java
private String condiments;
private String dressing;
private String meat;

public LunchOrderTeloBean(String bread){
    this.bread = bread;
}

public LunchOrderTeloBean(String bread, String condiments){
    this(bread);
    this.condiments = condiments;
}

public LunchOrderTeloBean(String bread, String condiments, String dressing){
    this(bread, condiments);
    this.dressing = dressing;
}

public LunchOrderTeloBean(String bread,
    String condiments, String dressing,
    String meat) {
    // TODO Auto-generated constructor stub
    this(bread, condiments, dressing);
    this.meat = meat;
}

public String getBread() {
    return bread;
}

public String getCondiments() {
    return condiments;
}

public String getDressing() {
    return dressing;
}

public String getMeat() {
    return meat;
}

```

Now what if we need to create a sandwich with just bread and meat. We cannot do it using this way. So we will have to maintain the old way of getters and setters.

So HERE comes the **BUILDER PATTERN**. (Excited ?!! I'm)

- 1) Create the Lunch Order Bean with all member variables as **PRIVATE FINAL**
- 2) Create the GETTERS for all the member variables.
- 3) Create a inner class or even a separate class , for this example we name it as BUILDER.
- 4) Create a constructor that takes this BUILDER as input.
- 5) BUILDER class has the same member variables with **PRIVATE** signature.
- 6) Create SETTER methods that will take in these variables , BUT UNLIKE generic

setters instead of RETURNING VOID, they RETURN BUILDER itself using THIS.

- 7) Create a method in our BUILDER class, here we call it as build(). That will return LunchOrder where LunchOrder was created using point 4.

Inner(Builder) Class:

```
public static class Builder{
    private String bread;
    private String condiments;
    private String dressing;
    private String meat;

    public LunchOrder build(){
        return new LunchOrder(this);
    }

    public Builder setBread(String bread) {
        this.bread = bread;
        return this;
    }

    public Builder setCondiments(String condiments) {
        this.condiments = condiments;
        return this;
    }

    public Builder setDressing(String dressing) {
        this.dressing = dressing;
        return this;
    }

    public Builder setMeat(String meat) {
        this.meat = meat;
        return this;
    }
}
```

LunchOrder Class:

```
private final String bread;
private final String condiments;
private final String dressing;
private final String meat;

public LunchOrder(Builder builder) {
    // TODO Auto-generated constructor stub
    this.bread = builder.bread;
    this.condiments = builder.condiments;
    this.dressing = builder.dressing;
    this.meat = builder.meat;
}

public String getBread() {
    return bread;
}

public String getCondiments() {
    return condiments;
}

public String getDressing() {
    return dressing;
}

public String getMeat() {
    return meat;
}
```

Demo Class:

```
package info.javagrasp.builder;

public class BuilderLunchOrderDemo {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        LunchOrder.Builder builder = new LunchOrder.Builder();
```

```

        builder.setBread("Wheat");
        builder.setCondiments("Lettuce");
        builder.setDressing("Mustard");
        builder.setMeat("Ham");

        LunchOrder lo = builder.build();

        System.out.println(lo.getBread());
        System.out.println(lo.getCondiments());
        System.out.println(lo.getDressing());
        System.out.println(lo.getMeat());
    }
}

```

Problem we solved here:

- 1) Instead of having multiple telescopic constructors we made it possible with a simple builder.
- 2) So you have 100 member variables , you are allowed to have 100 getters but not setters(we are trying to build a IMMUTABLE object).
- 3) Every variable can have multiple choices so you cant figure out a default value , except that you can have them as NULL
 - a. In our case if the user didn't choose MEAT and you have HAM as default value for meat then the sandwich will always have HAM in the meat unless you set it to something else. But we don't have any setters. Also we don't have a constructor that will help you in this case.
- 4) So what builder pattern does is , when you have the condition to build a IMMUTABLE object , instead of writing TELESCOPIC constructors use a BUILDER pattern.