**Builder Design Pattern**

In our day to day programming we create lots of objects using “constructor” but problem persists when we require lot of constructor say:

public class Message{

Message(String from,String to,String message){}

Message(String from,List<String> to,String message){}

Message(String from,String to,String message,Acknowledge ack){}

Message(String from,List<String> to,String message,Acknowledge ack){}

Message(String from,List<String> to,String message,Acknowledge ack,boolean retryOnFailure){}

.

.

}

So in above example we have one Message class and we have different - different constructor to make our Message object.

Some times we have one message and want to send it to one person only hence we will create object using 1st constructor

Message(String from,String to,String message){}

Some times we have one message and want to send it to multiple person hence we will create object using 2st constructor

Message(String from,List<String> to,String message){}

Some times we have one message and want to send it to one person only but want delivery acknowledge hence we will create object using 3st constructor

Message(String from,List<String> to,String message){}

And so on..

So there is better way to creating complex object using Builder pattern.

**### More about Builder pattern**

One of major advantage of using builder is it creates immutable objects and which is not possible by using getter and setter because using this approach because using it we use immutability and its also not as good readable as builder created object is.

We can create our object using constructor also but do you think in above example if we are asked to add some more parameters to constructor, is it really manageable code ??

Now here the problem comes when we manage the code with so many constructors.

One simple solution to all of these problem is Builder pattern.

**### Builder example**

The problem we have discussed earlier will now be solved using Builder pattern:

``` java

public class Message{

private String to;

private String from;

private List<String> toAl=new ArrayList<String>();

private Boolean retry;

Message(MessageBuilder mb){

this.to=mb.to;

this.from=mb.from;

this.toAl=mb.toAl;

this.retry=mb.retry;

}

public String getTo(){

return to;

}

public String getFrom(){

return from;

}

public String getToAl(){

return toA1;

}

public String getRetry(){

return retry;

}

public class MessageBuilder{

private String to;

private String from;

private List<String> toAl=new ArrayList<String>();

private Boolean retry;

public static MessageBuilder To(String to)

{

this.to=to;

return this;

}

public MessageBuilder To(List<String> toAl)

{

this.toAl=toAl;

return this;

}

public MessageBuilder From(String from)

{

this.from=from;

return this;

}

public MessageBuilder To(String to)

{

this.to=to;

return this;

}

public Message build(){

Message message = new Message(this);

validateUserObject(user);

return message;

}

private void validateUserObject(User user) {

//Do some basic validations to check

//if user object does not break any assumption of system

}

}

private void validateUserObject(User user) {

//Do some basic validations to check

//if user object does not break any assumption of system

}

}

```

``` java

public class Main{

public static void main(String a[])

{

List<String> al=new ArrayList<String>();

al.add(“909-090-9090”);

al.add(“121-212-1212”);

al.add(“232-323-2323”);

Message message=new Message.MessageBuilder()

.To(“123-123-1234”)

.From(“123-123-1234”)

.Message(“hello world”)

.build();

Message message=new Message.MessageBuilder()

.To(“123-123-1234”)

.From(al)

.Message(“hello world”)

.build();

Message message=new Message.MessageBuilder()

.To(“123-123-1234”)

.From(al)

.Message(“hello world”)

.Retry(true)

.build();

}

}

```

In above program we can clearly see that we don’t have setter method to set values to our object.

All data members are private and can’t be accessed outside and this is what which makes it immutable class.

**### Some classes in already exists in Java**

We have StringBuilder and StringBuffer already available which uses Builder pattern already.

Let’s see below example for reference:

``` java

StringBuilder builder = new StringBuilder("Temp");

String data = builder.append(1)

.append(true)

.append("friend")

.toString();

System.out.println(data);

```

```

Output:

Temp1truefriend

```

**### Benefits of using Builder**

Undoubtedly, the **number of lines of code increase** at least to double in builder pattern, but the effort pays off in terms of **design flexibility** and much more **readable code**. The **parameters to the constructor are reduced** and are provided in **highly readable method calls**.

Builder pattern also helps minimizing the number of parameters in constructor and thus there is **no need to pass in null for optional parameters** to the constructor. It’s really attracts me.

Another advantage is that Object is always instantiated in a complete state rather than sitting in an incomplete state until the developer calls (if ever calls) the appropriate “setter” method to set additional fields.

And I finally I can build **immutable objects** without much complex logic in object building process.