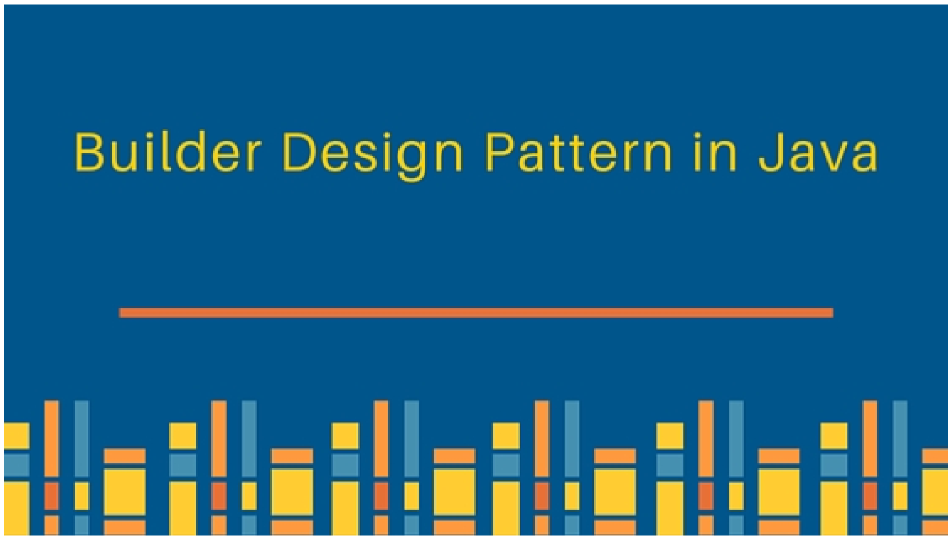
Builder Design Pattern in Java

MAY 11, 2018 BY PANKAJ — 38 COMMENTS

Today we will look into Builder pattern in java. Builder design pattern is a **creational design pattern** like **Factory Pattern** and **Abstract Factory Pattern**.

Table of Contents [hide]

Builder Design Pattern



Builder pattern was introduced to solve some of the problems with Factory and Abstract Factory design patterns when the Object contains a lot of attributes.

There are three major issues with Factory and Abstract Factory design patterns when the Object contains a lot of attributes.

1. Too Many arguments to pass from client program to the Factory class that can be error prone because most of the time, the type of arguments are same and from client side its hard to maintain the order of the argument.
2. Some of the parameters might be optional but in Factory pattern, we are forced to send all the parameters and optional parameters need to send as NULL.
3. If the object is heavy and its creation is complex, then all that complexity will be part of Factory classes that is confusing.

We can solve the issues with large number of parameters by providing a constructor with required parameters and then different setter methods to set the optional parameters. The problem with this approach is that the Object state will be **inconsistent** until unless all the attributes are set explicitly.

Builder pattern solves the issue with large number of optional parameters and inconsistent state by providing a way to build the object step-by-step and provide a method that will actually return the final Object.

Builder Design Pattern in Java

Let's see how we can implement builder design pattern in java.

1. First of all you need to create a static nested class and then copy all the arguments from the outer class to the Builder class. We should follow the naming convention and if the class name is `Computer` then builder class should be named as `ComputerBuilder`.
2. Java Builder class should have a public constructor with all the required attributes as parameters.

3. Java Builder class should have methods to set the optional parameters and it should return the same Builder object after setting the optional attribute.
4. The final step is to provide a `build()` method in the builder class that will return the Object needed by client program. For this we need to have a private constructor in the Class with Builder class as argument.

Here is the sample builder pattern example code where we have a Computer class and ComputerBuilder class to build it.

```
package com.journaldev.design.builder;

public class Computer {

        //required parameters
        private String HDD;
        private String RAM;

        //optional parameters
        private boolean isGraphicsCardEnabled;
        private boolean isBluetoothEnabled;


        public String getHDD() {
                return HDD;
        }

        public String getRAM() {
                return RAM;
        }

        public boolean isGraphicsCardEnabled() {
```

Notice that Computer class has only getter methods and no public constructor. So the only way to get a Computer object is through the ComputerBuilder class.

Here is a builder pattern example test program showing how to use Builder class to get the object.

```
package com.journaldev.design.test;

import com.journaldev.design.builder.Computer;

public class TestBuilderPattern {
	public static void main(String[] args) {
		//Using builder to get the object in a single line of code and
		//without any inconsistent state or arguments management issues
		Computer comp = new Computer.ComputerBuilder(
				"500 GB", "2 GB").setBluetoothEnabled(true)
				.setGraphicsCardEnabled(true).build();
	}

}
```

Builder Design Pattern Video Tutorial

Recently I uploaded a YouTube video for Builder Design Pattern. I have also explained why I think the builder pattern defined on WikiPedia using Director classes is not a very good Object Oriented approach, and how we can achieve the same level of abstraction using different approach and with one class.

Note that this is my point of view, I feel design patterns are to guide us, but ultimately we have to decide if it's really beneficial to implement it in our project or not. I am a firm believer of KISS principle.

Subscribe to my YouTube Channel YouTube 6K

If you like the video, please do share it, like it and subscribe to my channel. If you think I am mistaken or you have any comments or feedback so that I can improve my videos in future, please let me know through comments here or on YouTube video page.

Builder Design Pattern Example in JDK

Some of the builder pattern example in Java classes are;

- java.lang.StringBuilder#append() (unsynchronized)
- java.lang.StringBuffer#append() (synchronized)

That's all for builder design pattern in java.

You can download the example code from my GitHub Repository.

« PREVIOUS
Abstract Factory Design Pattern in Java

NEXT »
Prototype Design Pattern in Java

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DESIGN PATTERNS

Comments

Mehdi Ahmed says

JULY 20, 2018 AT 2:41 AM

Hey man thanks for this nice video/Article

I have a question.

What if you are working with POJO that you need to persist with Jpa/Hibernate.

Do we need the setters for that?

Reply

Pankaj says

JULY 20, 2018 AT 7:41 AM

Hibernate requires No-Args constructor as well as getter-setter for the fields. So yes, they will be required to work with Hibernate.

Reply

Bismeet says

JULY 11, 2018 AT 6:40 PM

Since we have to invoke the constructors and setter methods explicitly anyway, cant we do it directly on the computer class via a public constructor? Why did we even need the inner class ??

Reply

Kamil says

JULY 26, 2018 AT 3:03 AM

Once you build the object it is immutable. We can achieve that only by providing setters methods on the nested class. Otherwise, we would expose public setters on the class causing mutability of the object.

Reply

Ujjawal Besra says
JULY 1, 2018 AT 3:33 AM
This is a different version of Builder pattern which I have come to know but definitely its a way of implementing things. Thanks for sharing this with the code.
Reply

satya says
MAY 29, 2018 AT 6:30 PM
If you are copying same data from compute calss to computer builder class and pass those parameters from client to builder . . Why are we doing this , why cant we directly instantiate object of computer from client . What is the use of builder pattern . Because you are any way having constructer in builder class for which client needs to send parameters , in the same way we can declare constructer in main class itself and parameters from client. You example is not clear why to use builder pattern.
Reply

Bismeet says
JULY 11, 2018 AT 6:41 PM
I have the same question.
Reply

Kamil says
JULY 26, 2018 AT 3:52 AM
Immutability is the key. Once the object is built, it cannot be changed since there are no setters
Reply

yaccob says
MARCH 8, 2018 AT 2:30 PM
A question regarding "We should follow the naming convention and if the class name is Computer then builder class should be named as ComputerBuilder":
Since our builder is a nested class, it will usually be addressed by something like ContainingClass.NestedClass, right? For example ClassToBeBuilt.Builder. Why does the convention suggest to use ClassToBeBuilt.ClassToBeBuiltBuilder instead?
Reply

Pankaj says
MARCH 8, 2018 AT 11:33 PM
It's not a requirement to have Builder class as a nested class. Just for simplicity of coding, I have create it as nested class. If you have a lot of POJO and their builder classes, then it's best strategy to have a package itself for builder classes. For example `com.journaldev.java.pojo` for POJOs and `com.journaldev.java.builders` for Builder classes. However note that in this case, POJO classes constructor can't be private and can be instantiate directly.
Reply

kathiravan says
NOVEMBER 1, 2017 AT 6:26 AM
Font is too big to Read.
pls make one unit small to boost Readability.
Reply

ravi says
JUNE 15, 2017 AT 2:21 AM
Builder pattern requires a Director, I don't see it here.
just because you are calling a build() in the end will not make it a builder pattern.
Reply

ravi says
JUNE 15, 2017 AT 2:27 AM
continuing Builder pattern must build a complete object in parts. i.e. there must be a abstract builder and then concrete builders must be responsible for building one part each.
StringBuilder in java is not a right example of BuilderPattern neither the HttpSecurityBuilder of spring.
Reply

Pankaj says
MAY 1, 2018 AT 2:29 AM
The Builder pattern I have explained here is the same as defined by Joshua Bloch in Effective Java. I don't like the Builder pattern implementation provided in Wikipedia. It defeats the purpose of having variables. According to Wikipedia implementation, every Director class will be able to create only one variance of an Object. So if we have Director classes for "Car" and we

want to have 5 different colours, it will result in classes like BlackCarDirector, RedCarDirector, GreenCarDirector.
I hope you understood where I am going with this. Again it's my point of view, design patterns provide us a way to do things properly. But it doesn't mean we have to follow them strictly as is, we can make certain modifications based on our project requirement to make it even better with code simplicity.
Reply

Vinoth says
MARCH 14, 2017 AT 8:16 AM
http://stackoverflow.com/questions/5238007/stringbuilder-and-builder-pattern
he Append method of StringBuilder simply adds more characters to the existing string. There are no new objects created (basic function of the builder pattern) and there is no design pattern involved. It's a single method call to a single object instance.
Please correct it.
Reply

PK says
JANUARY 6, 2017 AT 5:40 AM
Well explained.
Reply

Venu says
DECEMBER 16, 2016 AT 6:46 AM
I see Builder Pattern being used in Spring Security (Something like this) . This is reference for Pattern being used ?
public final class HttpSecurity extends
AbstractConfiguredSecurityBuilder
implements SecurityBuilder,
HttpSecurityBuilder {
its too complex to me to grasp easily in spring custom class . However this tutorial example is clear to understand the basic idea of using this Builder Pattern and implement.
@Override
public void configure(HttpSecurity http) throws Exception {
http.

anonymous().disable()
.requestMatchers().antMatchers("/user/**")
.and().authorizeRequests()
.antMatchers("/user/**").access("hasRole('ADMIN')")
.and().exceptionHandling().accessDeniedHandler(new OAuth2AccessDeniedHandler());
}
Thank you Pankaj /Jounaldev
Reply

Saurabh Tiwari says
NOVEMBER 23, 2016 AT 5:46 AM
Please correct this syntax which is using in Builder factory pattern example.
Your Code:
Computer comp = new Computer.ComputerBuilder(
"500 GB", "2 GB").setBluetoothEnabled(true)
.setGraphicsCardEnabled(true).build();
call statement is wrong
"new Computer.ComputerBuilder"
Reply

R says
DECEMBER 4, 2016 AT 4:25 AM
Please see the full call the build method is returning us the Computer Object.
Reply

Hari says
OCTOBER 14, 2016 AT 2:22 AM
Very nice article for Design Patterns!! SPOC blog for all design patterns for beginners.
Reply

naval jain says
SEPTEMBER 19, 2016 AT 2:47 AM
It seems the only problem addressed here is when there are lot of input arguments to the constructor and all of them are not necessary.
The above specified solution is just one of the use case of builder pattern.

Don't know if it is correct.
Please refer to effective Java for the above specified problem. Chapter 2 Item 2.
Reply

Maliox says
SEPTEMBER 9, 2016 AT 7:40 PM
This is not the definition of Builder pattern described in GoF. However, it's a good solution for the problem mentioned at the beginning of the article which was addressed in Effective Java book.
Reply

Kishan says
JULY 8, 2016 AT 9:54 AM
I have one question that we are returning this in set() I know it is very important to return this in Builder pattern. But, Can you tell me what is the actual meaning when we return this and how it is actually executed. I have some confusions that how complier deal when we have used set() method of different fields for multiple times in same line of code while creating object and who will pick up this references of object before build().
Reply

Prabhakaran says
JULY 8, 2016 AT 2:46 AM
Builder pattern described here is completely different from what other websites implementations. Here it shows with static inner class but others its not. Not sure which one is correct.
Reply

Arijit Medya says
APRIL 14, 2016 AT 5:18 AM
This blog was really nicely articulated and helpful. Thanks …
Reply

vijay says
MAY 5, 2015 AT 11:25 AM

By this way, we can also create the object. Can you please explain the difference in using return this and the way below :

```
public class TestBuilderPattern {
public static void main(String[] args) {
// Using builder to get the object in a single line of code and
// without any inconsistent state or arguments management issues
Computer.ComputerBuilder comp = new Computer.ComputerBuilder("500 GB",
"2 GB");
comp.setBluetoothEnabled(true);
comp.setGraphicsCardEnabled(true);
comp.build();
System.out.println(comp);
}
public void setGraphicsCardEnabled(boolean isGraphicsCardEnabled) {
this.isGraphicsCardEnabled = isGraphicsCardEnabled;
// return this;
}
public void setBluetoothEnabled(boolean isBluetoothEnabled) {
this.isBluetoothEnabled = isBluetoothEnabled;
// return this;
}
}
```

Reply

Rishi Naik says

JANUARY 5, 2015 AT 10:50 PM

What is a need of getter and setter methods for outer class in above example if we are providing private constructor and no outer world can access the outer class?

Reply

Ketki says

DECEMBER 8, 2015 AT 3:34 AM

That's correct as per my knowledge as well. There is no need to have getter and setter methods in the outside class.

Reply

Pratap Shinde says

JANUARY 23, 2016 AT 3:37 AM

Rishi: There are no setters in the outer class only getters.

Secondly you do need those getter methods, otherwise how are you going to utilize the

properties set in the Outer class
By having just private constructor doesn't mean you cannot access the outer class
Computer comp = new Computer.ComputerBuilder("500 GB", "2 GB").setBluetoothEnabled(true)
.setGraphicsCardEnabled(true).build(); //This is creating the outer object
System.out.println(comp.isBluetoothEnabled()) //This is use of getter method
Reply

Ashakant says
OCTOBER 5, 2014 AT 2:08 AM
Can you please share/post UML design for the same as explain above
Thanks
Ashakant
Reply

Ashish says
SEPTEMBER 9, 2014 AT 5:46 AM
Hi Pankaj,
There is still an issue with this approach. We are still expecting the client code to invoke the setters explicitly to ensure that the object state is consistent before the finally created object is returned upon invocation of build() method. We could have invoked the build method without even calling the setters and the boolean variables would have been initialized to false by default and still the object creation would have been successful with the consistent state. I think the another approach could be to provide a series of overloaded constructors with various combinations of optional parameters which invoke them in sequence until all the parameters are set and then the final object is created. In this case, we would not require to have even a build method.
Please comment.
Reply

Pankaj says
SEPTEMBER 9, 2014 AT 5:52 AM
You can easily avoid that by using Boolean rather than primitive type boolean. Object default value is NULL. Also you should have a constructor with all the mandatory parameters so that client is forced to provide values for them.
Creating multiple constructor with optional parameters will not help and you will not be using Builder pattern then.
Reply

Amritpal Singh says
AUGUST 13, 2014 AT 4:35 AM
Hello Pankaj i have Question Why u have used only static nested class why not normal/regular inner class…..i knw differences but in this example why we have used static nested clas only….plz reply..!
Reply

Surya says
AUGUST 25, 2014 AT 4:26 PM
The primary usage of the **Static** Nested Inner class here is to create an inner class object without instantiating the enclosing outer class.
Instead of creating multiple overload constructors and/or various setters for the Computer class and try to construct an object in multiple steps (Object created this might be inconsistent till all the needy fields are set!), we try to construct the computer object by using the static nested inner class **ComputerBuilder**.
Reply

M says
JULY 8, 2014 AT 4:21 AM
Good article.
Why does below functions needs to return "this" when setting parameter? can you please give pointer on this?
Reply

Pankaj says
JULY 8, 2014 AT 7:37 AM
`return this;` is used to return the current object, this is how Builder pattern works and it's key for that.
Otherwise `new Computer.ComputerBuilder("500 GB", "2 GB").setBluetoothEnabled(true).setGraphicsCardEnabled(true).build();` will not work.
Reply

HIMANSU NAYAK says

FEBRUARY 2, 2014 AT 3:50 AM

Good article. Simple, lucid & very specific.
Brings all the initialization complexity to inner class, which keeps you outer class clean, love this pattern.
Is it like builder pattern cannot be implemented on those class whose attributes keep shifting from mandatory to optional and vice-versa, unless its designed to take every attributes thru the setter method of inner class.

Reply

Pankaj says

FEBRUARY 2, 2014 AT 4:34 AM

Yes if the attributes keep on changing from mandatory to optional and vice versa, we will have to change the inner class that will break the pattern. So implementation should be done based on clear requirements.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

DESIGN PATTERNS TUTORIAL

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial

Factory Design Pattern

Builder Design Pattern

Java Programming Courses

Learn Programming Online

Java Course Online

Free Programming Tutorials

Learn Java in 7 Days

Create Your Own Website