Builder Design Pattern

20 November 2021 23:21

- The builder pattern is a design pattern designed to provide a flexible solution to various object creation problems in object-oriented programming. The intent of the Builder design pattern is to separate the construction of a complex object from its representation.
- It is used to construct a complex object step by step and the final step will return the object. The process of constructing an object should be generic so that it can be used to create different representations of the same object.

- Thus, there are two specific problems that we need to solve:
  - Too many constructor arguments.
  - Incorrect object state.

Advantages of Builder Design Pattern

- The parameters to the constructor are reduced and are provided in highly readable method calls.
- Builder design pattern also helps in minimizing the number of parameters in the constructor and thus there is no need to pass in null for optional parameters to the constructor.
- Object is always instantiated in a complete state
- Immutable objects can be built without much complex logic in the object building process.

- All implementations of java.lang.Appendable are in fact good example of use of Builder pattern in java. e.g. java.lang.StringBuilder#append() [Unsynchronized class]

  java.lang.StringBuffer#append() [Synchronized class]

  java.nio.ByteBuffer#put() (also on CharBuffer, ShortBuffer, IntBuffer, LongBuffer, FloatBuffer and DoubleBuffer)

- Look how similar these implementations look to what we discussed above.
  StringBuilder builder = new StringBuilder("Temp");

```
String data = builder.append(1)
      .append(true)
      .append("friend")
      .toString();
```

- Separate the construction of a complex object from its representation so that the same construction process can create different representations
- The Builder pattern allows us to write readable, understandable code to set up complex objects

When to use Builder Pattern:

- If you find yourself in a situation where you keep on adding new parameters to a constructor, resulting in code that becomes error-prone and hard to read, perhaps it's a good time to take a step back and consider refactoring your code to use a Builder.

- Look at the User class and User Builder class which we created in the class to implement builder design pattern. User class can have many user attributes, like first name, last name, email, mobile, address
  And in future we can have more. If we create the constructor with these fields if the field is optional then we have to pass null and every time we should be conscious while creating the object, we may pass address in place of email if we mistaken.

- To avoid all these issues instead of calling the constructor, we can create the method and call these methods to create the object. So complex object can be easily developed with this approach
- We created User Builder inner class and added all the attributes using the methods and created the User object