15. Implementing complex conditional logic

So, you need to implement either a switch statement with several cases or an if-else logic with several conditions.

A traditional way of dealing with this is just to implement specific code under each condition. However, if you are dealing with complex conditional logic, your code will become harder to read and maintain.

If you have this conditional logic inside a single method, it will probably become quite difficult to write unit tests for this method. Your code will probably violate SOLID principles, as the method will have more than one responsibility. It will be responsible for both making conditional decisions and executing different types of logic based on those decisions. You may end up having as many responsibilities inside this method as there are conditions in your statement!

Suitable design patterns

Strategy

Strategy is a design pattern where you have a container object, usually referred to as Context, that contains a specific interface inside of it. Strategy is any class that implements this interface.

The exact implementation of a Strategy object inside the Context object is assigned dynamically. So, the Context object allows you

to replace the exact implementation of the Strategy object at any time.

Strategy object would usually have one core action method that is defined on the interface and the Context object will have a wrapper method with a similar signature that will call this method on the current Strategy implementation.

So, what you would normally do is write several Strategy implementations, each of which would contain its own version of the method that executes the action. Then, when you need to have any multi-condition logic, all you will do under each condition is pass a particular implementation of the Strategy into the Context class. In the end, you just execute the action method on the Context class.

If you do this, all that your method with the conditional logic will be responsible for is deciding which Strategy to pass into the Context class. And it will be trivial to validate this behavior with unit tests. Then, each Strategy implementation will be solely responsible for its own specific version of the logic that is to be executed, which is also easy to read and write unit tests against.

Why would you use Strategy

- A really easy way of isolating specific conditional behavior into its own method.
- Helps to enforce the single responsibility principle by making the code with the conditional statements solely responsible for outlining the conditions.
- It becomes easy to write unit tests against any code with complex conditional logic.

Factory Method

Factory method, which we already had a look at in chapter **chapter 6**, is also designed to be used in a complex conditional logic, but its usage is different from the usage of Strategy.

Factory Method allows you to generate any object of a particular type and, just like with Strategy, you can decide on the exact implementation of the Factory Method based on conditions. However, there is one subtle, but significant, difference.

Strategy is used for executing a particular behavior or returning a short-lived simple data object, while the Factory Method is used for generating a long-lived object that can execute its own logic outside of the Factory Method.

For example, let's imagine that you have two databases – the main database and the archive database. They both store data in exactly the same format, while the data itself is different.

If you would need to decide which database to return the data from at the request time, you would use Strategy. For example, a Boolean value that determines whether or not to get the data from the archive is defined as a part of the request. Then you would select a Strategy implementation depending on whether that value is set to true or false.

If true, you would use the implementation that gets the data from the archive database. If false, you would get the implementation that retrieves the data from the main database. But whichever database the data comes from, it's the same data structure in the code that represents the data.

Now, imagine that you need to decide whether or not to use the archive database at config time. This way, you would have an abstraction that represents a database connection. When you launch the application, a Factory Method will assign the representation of either the main or the archive database to this abstraction depending on the config value. After that, any request will retrieve the data from whichever database was set up by this logic when the application was first launched.

The differences between Factory Method and Strategy

- Strategy is used for conditional execution of a specific action that may involve the retrieval of relatively simple data objects.
- Factory Method is used for conditionally creating long-lived objects that have their own logic.

Abstract factory

Because abstract factory is nothing other than a collection of related Factory Methods, it is applicable in the same way as the Factory Method is, but only if you need to obtain a group of related objects rather than a single object.