# Strategy Design Pattern

16 November 2021 08:51

- Strategy pattern (also known as the policy pattern) is a behavioural software design pattern that enables selecting an algorithm at runtime. Instead of implementing a single algorithm directly, code receives run-time instructions as to which in a family of algorithms to use
- The application can switch strategies at run-time.
- Strategy enables the clients to choose the required algorithm, without using a “switch” statement or a series of “if-else” statements.

- When to Use Strategy Design Pattern ?

- Use the Strategy pattern when you want to use different variants of an algorithm within an object and be able to switch from one algorithm to another during runtime. Use the Strategy when you have a lot of similar classes that only differ in the way they execute some behaviour.

- Here some examples of Strategy in core Java libraries:
  - java.util.Comparator#compare() called from Collections#sort().
  - javax.servlet.http.HttpServlet: service() method, plus all of the doXXX() methods that accept HttpServletRequest and HttpServletResponse objects as arguments.
  - javax.servlet.Filter#doFilter()

In order to explain the Strategy Design Pattern, consider the below example.
Example of Bank, which has Privileged customer and Normal customer and interest rates also will be different for each type of customer.
So In future this functionality may be extended for different customers or for example Employee of bank.
If we implement this functionality using if and else conditions then we may spoil the Open closed and single responsibility principle so in order to implement this we go for Strategy, such that at run time it will be decided which algorithm can be implemented.

1. Create the interface and make 1 abstract method
2. Create Privileged customer and Un Privileged customer class and extend the interface and override the abstract method for the interest rate
3. Create a separate class and create the constructor for this class which accepts interface reference  mentioned in point 1 as parameter

4. Create 1 method and on this object call the abstract method overridden in the Privileged  and Un Privileged  classes
5. From the Main class whenever u pass the Privileged Object then Privileged class interest methods gets invoked
6. And if we want to extend this functionality for Employee in future just create New class called Employee and implement the interface written in point1.
7. From Main method pass this Employee reference and call the method.
8. In this approach we are not modifying the code but extending the functionality.