Open/Closed Principle

This **principle of Open/Closed** is also part of SOLID principles of software development which falls under the concepts of Object Oriented Programming.

Open/Closed principle suggests that the class's capability can be extended but it should not modify as it can be susceptible to bugs. If I want to give exact formal definition for this principle, it would be as per below:

'The class is open for extension but it is close for modification'

We will understand the whole concept of this principle as per below:

Let us take an example. We had created the InvoiceSave class which had the logic to save the invoice.

```
2  // Use this editor to write, compile and run your Java code online
3
4  class InvoiceSave
5  {
6      //class of invoice is inherited from the parent class
7      Invoice invoice;
8
9      //creating the public constructor with the parameter passed
10     public InvoiceSave(Invoice invoice)
11     {
12         this.Invoice = invoice;
13     }
14
15     public void SaveToDB()
16     {
17         //logic for saving the invoice in the database
18     }
19 }
```

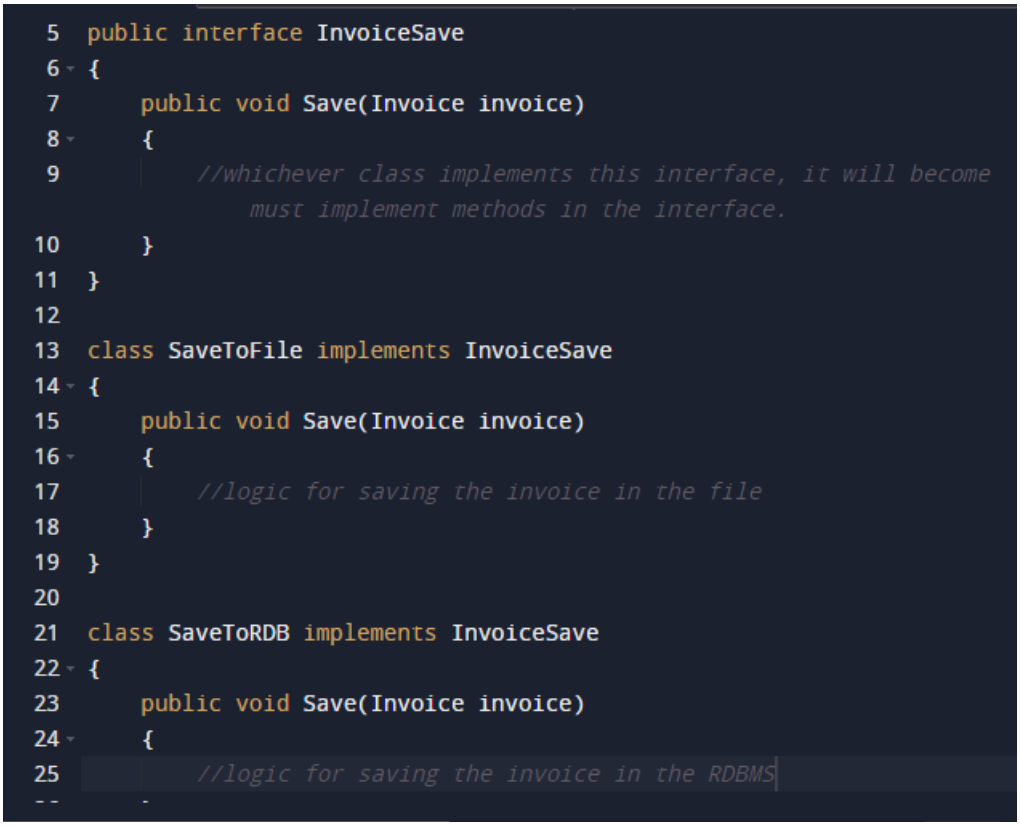
Now suppose I want to implement the feature of saving the invoice in the file in this class. It will require me to modify the class as I need to create the new method and write the method which is definitely a modification of the class.

```
2  // use this editor to write, compile and run your Java code online
3
4  class InvoiceSave
5  {
6      //class of invoice is inherited from the parent class
7      Invoice invoice;
8
9      //creating the public constructor with the parameter passed
10     public InvoiceSave(Invoice invoice)
11     {
12         this.Invoice = invoice;
13     }
14
15     public void SaveToDB()
16     {
17         //logic for saving the invoice in the database
18     }
19
20     public void SaveToFile()
21     {
22         //logic for saving the invoice in the database_
23     }
24 }
```

There is a possibility that by doing this method of implementation, the software can have bugs.This example is violation of **Single Responsibility** and **Open/Closed principle**.

Now to reduce the risk and prevent the bugs, we can implement the below solution:

We can implement the interface which will contain the common save method and we can implement different classes for different functionality using the interface.



```
5  public interface InvoiceSave
6  {
7      public void Save(Invoice invoice)
8      {
9          //whichever class implements this interface, it will become
               must implement methods in the interface.
10     }
11 }
12
13 class SaveToFile implements InvoiceSave
14 {
15     public void Save(Invoice invoice)
16     {
17         //logic for saving the invoice in the file
18     }
19 }
20
21 class SaveToRDB implements InvoiceSave
22 {
23     public void Save(Invoice invoice)
24     {
25         //logic for saving the invoice in the RDBMS
```

This is the way we can solve the problem and have continuity in the implementation of **SOLID Principles.**

Now here if I want to save the invoice in the database which is NoSQL which has totally different logic so I can implement the new class which will implement the interface which is good practice for maintenance and understanding of the codebase. Here we are extending the capability of InvoiceSave class.

So this is how we can understand the **Open/Closed Principle**.