**Design Pattern**

Design patterns provide, reusable solutions to common software design problems. However, when we are using these ideas in Java then you can make your code better, easier to update, and more flexible.

**Why Use Design Patterns?**

* Reusability**:** Using the same solution for common problems.
* Maintainability**:** Enhances code readability and structure.
* Flexibility**:** Adaptable to changing requirements.
* Communication**:** Provides a common vocabulary for developers.

**Types of Design Patterns**

Design patterns are categorized into three main types:

1. **Creational Design Patterns:** focus on how to create objects in a flexible way.

Examples: Singleton, Prototype, Factory, Builder

1. **Structural Design Patterns:** focus on how to combine classes and objects to build larger structures.

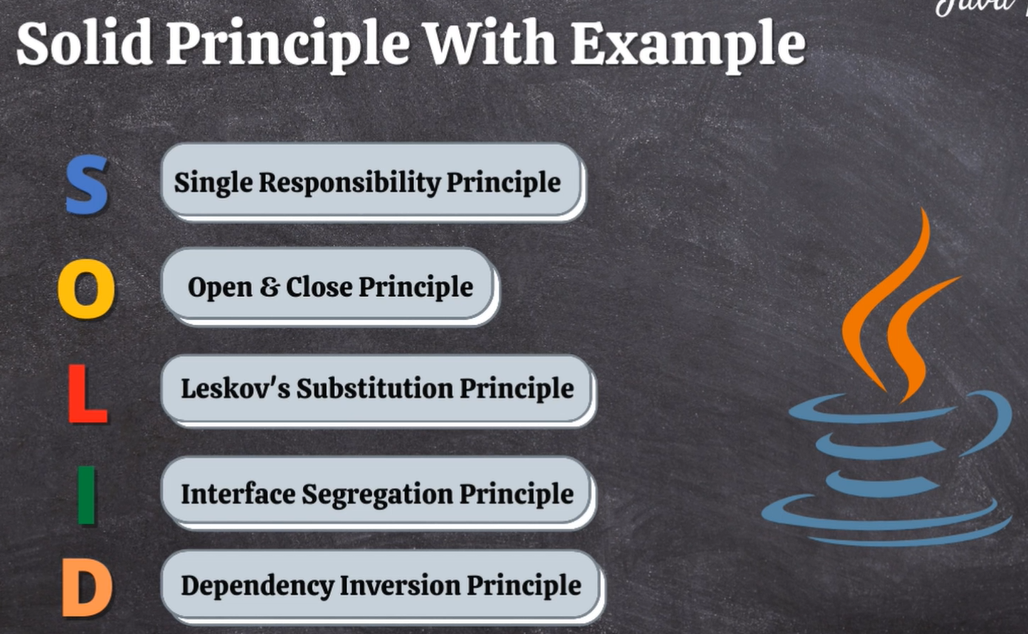
Examples: Adapter, Decorator, Proxy, Façade

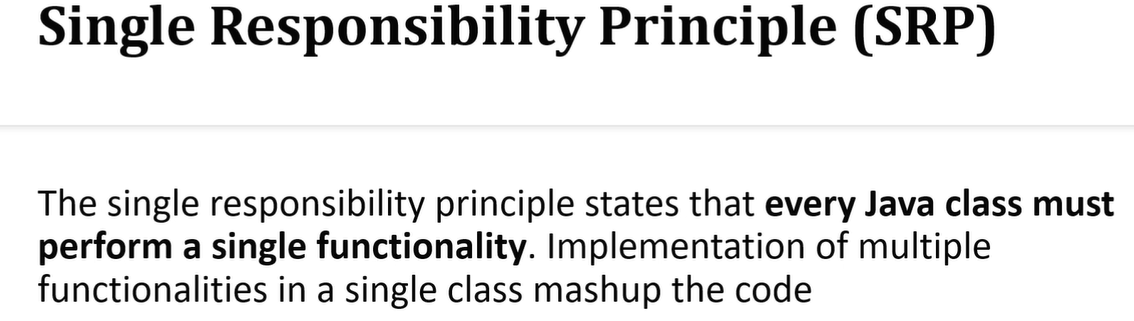
1. **Behavioral Design Patterns:** explain how to objects interact and communicate with each other.

Examples: Observer, Strategy, Template Method, Command

**Benefitsof Using Design Patterns**

* **Improved code quality:** Adhering to established patterns leads to better-structured and more efficient code.
* **Faster development:** Reusable patterns save time and effort.
* **Enhanced collaboration:** A common language for discussing design solutions.
* **Easier maintenance:** Well-structured code is easier to modify and extend.

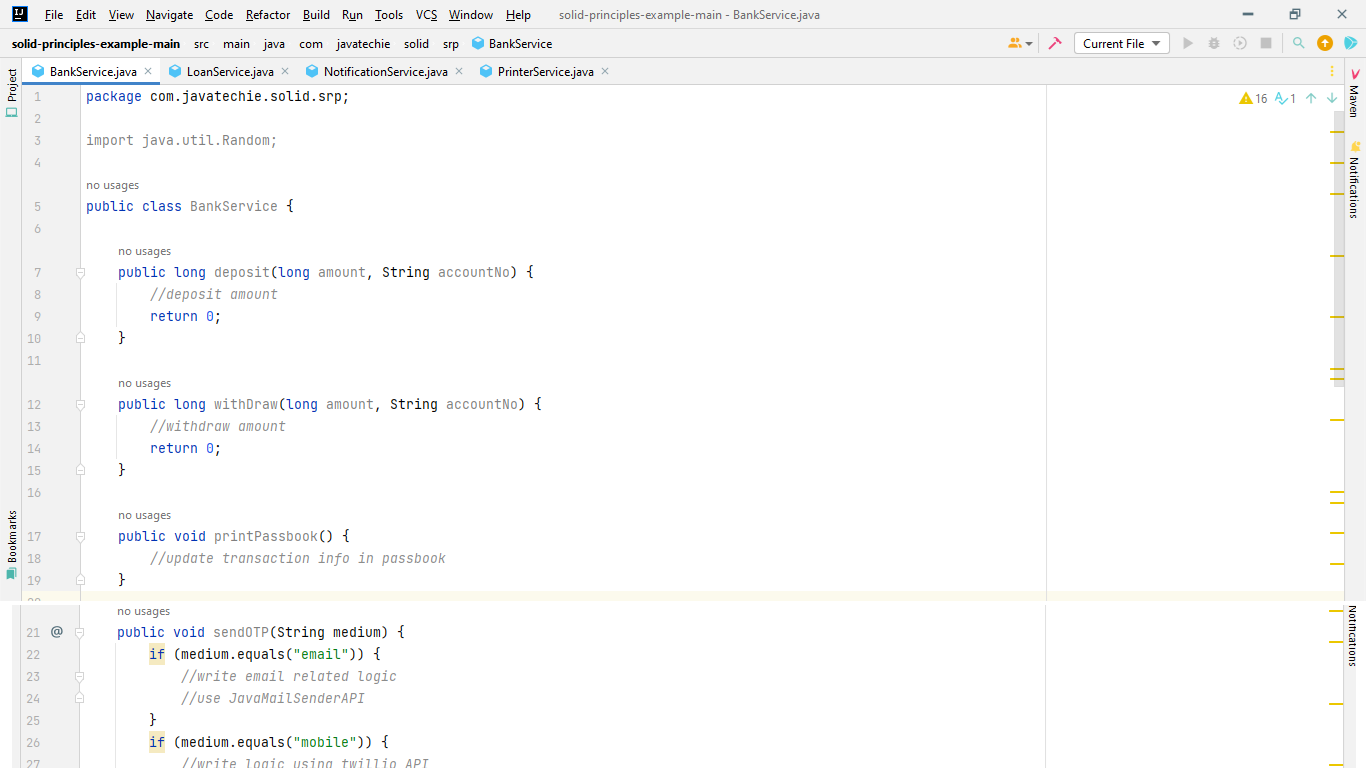




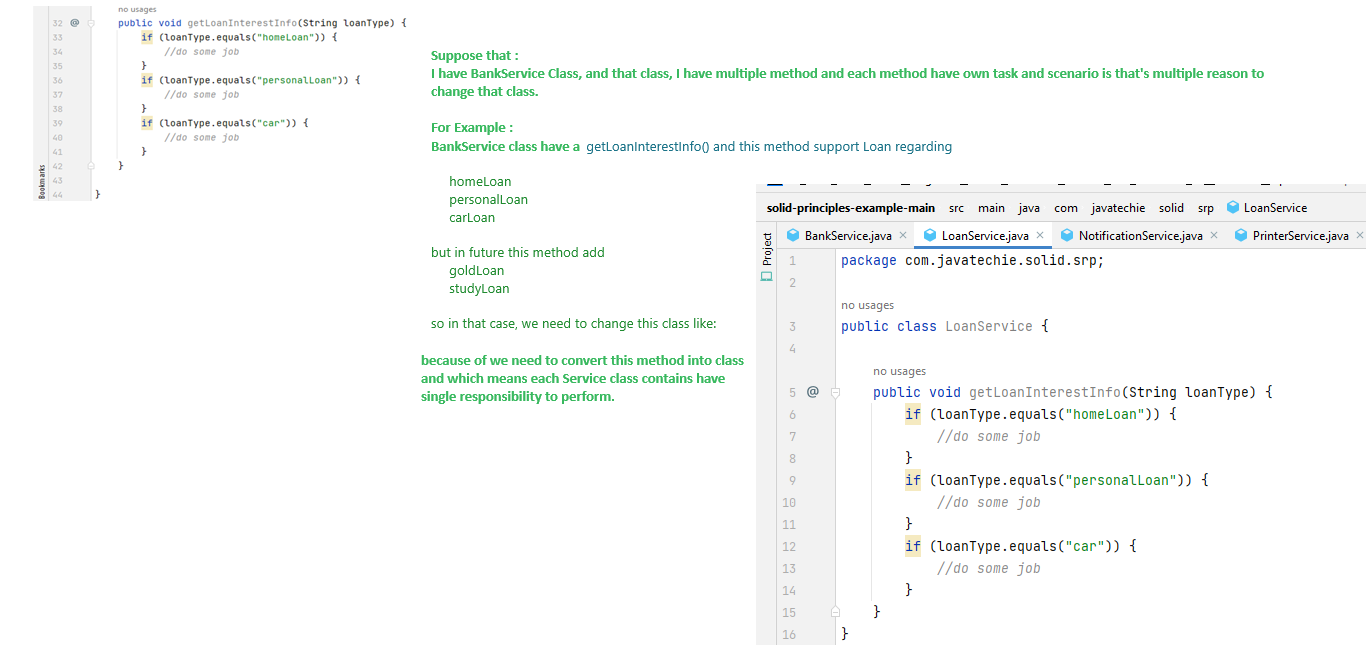
**Means in simple words**

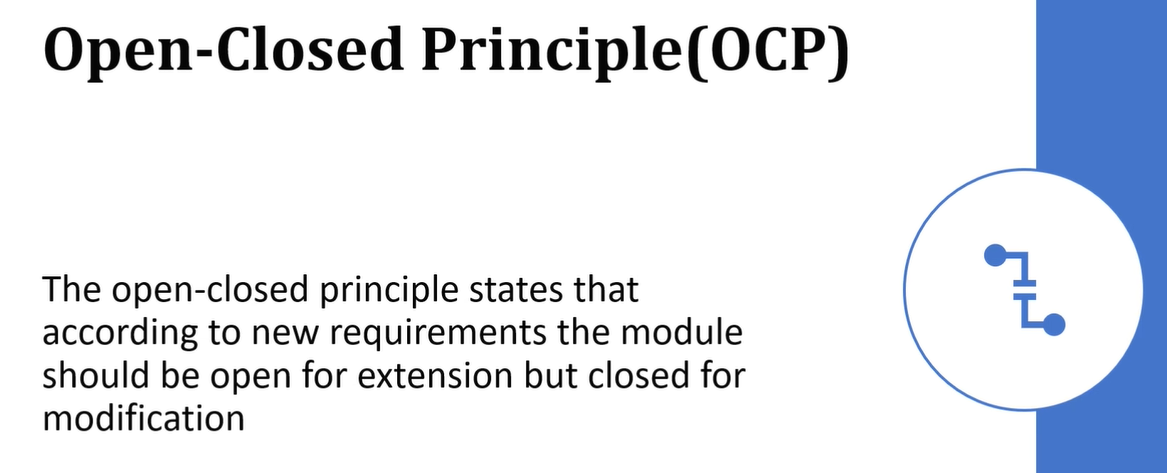
A class have only one reason to change which means every class should have a responsibility or a single job or single purpose to perform.

Example :





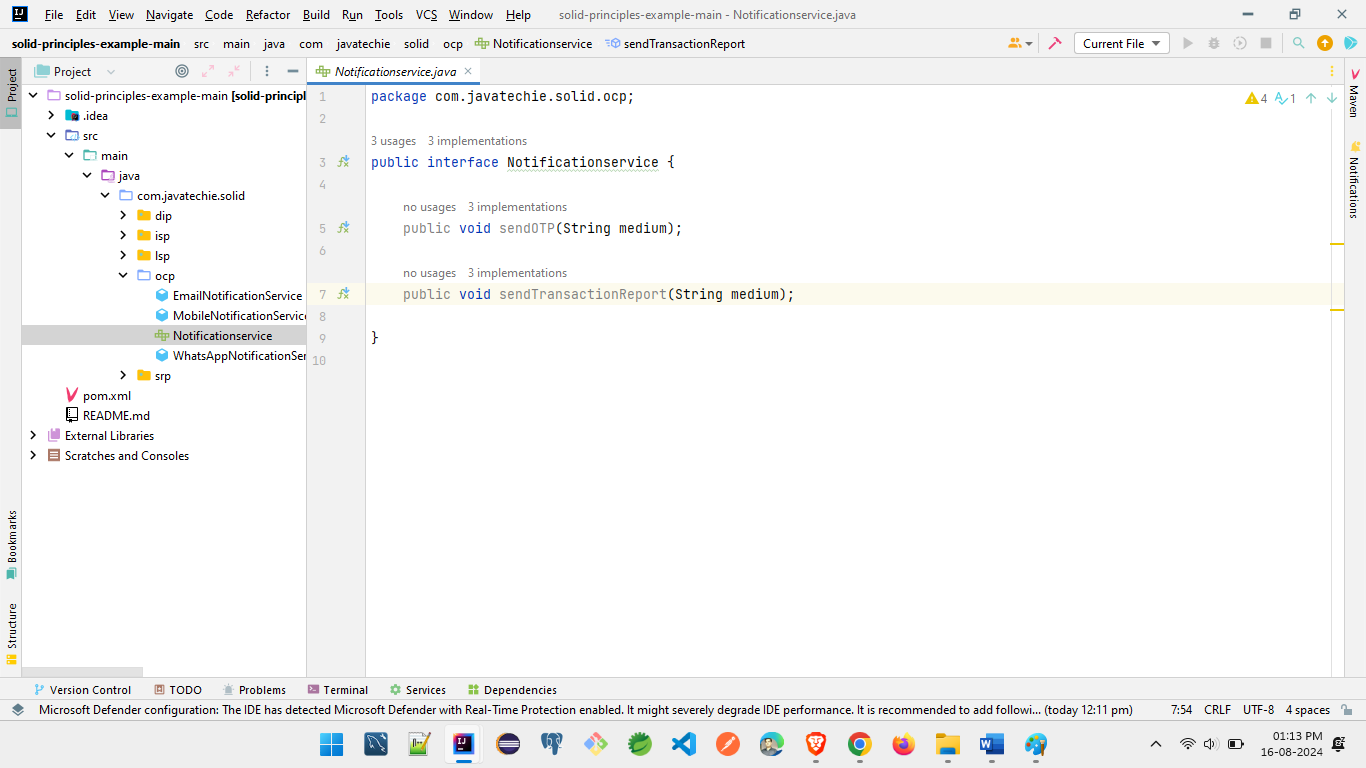


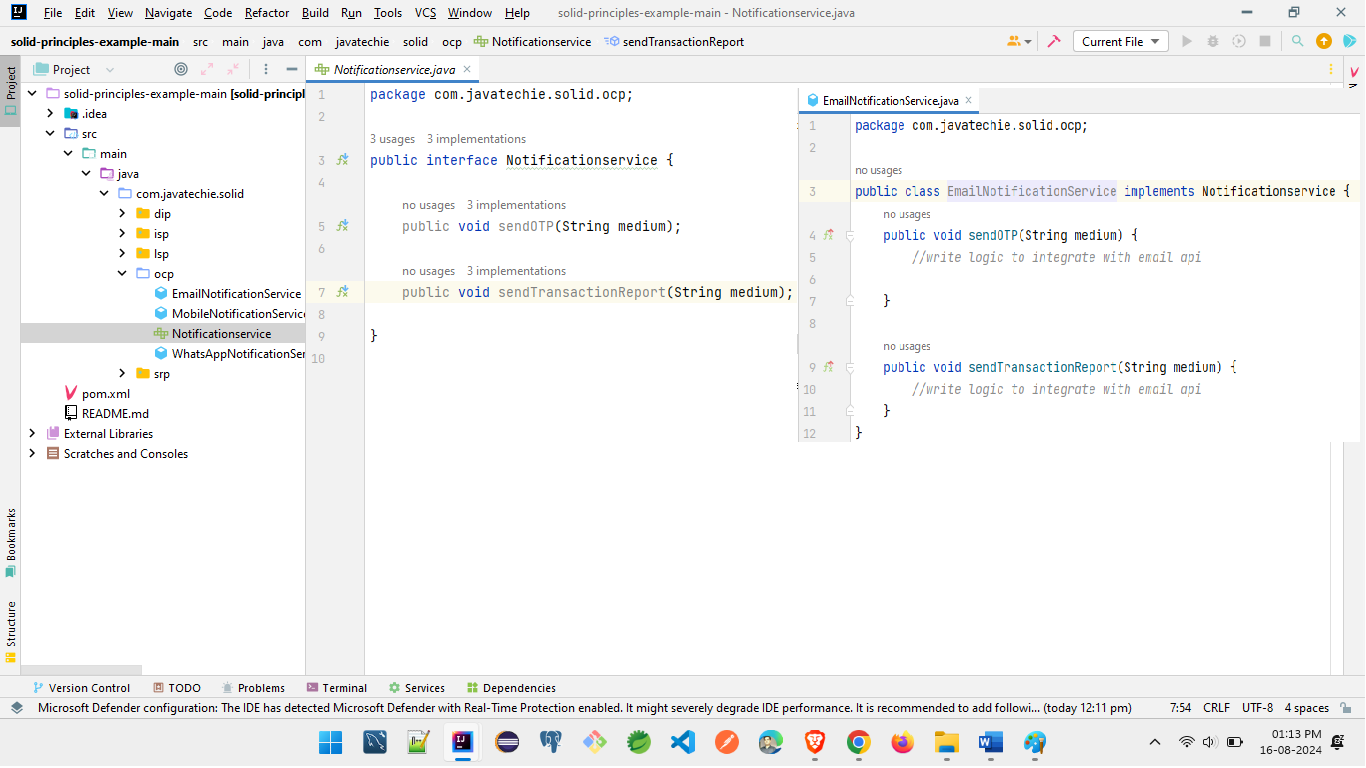


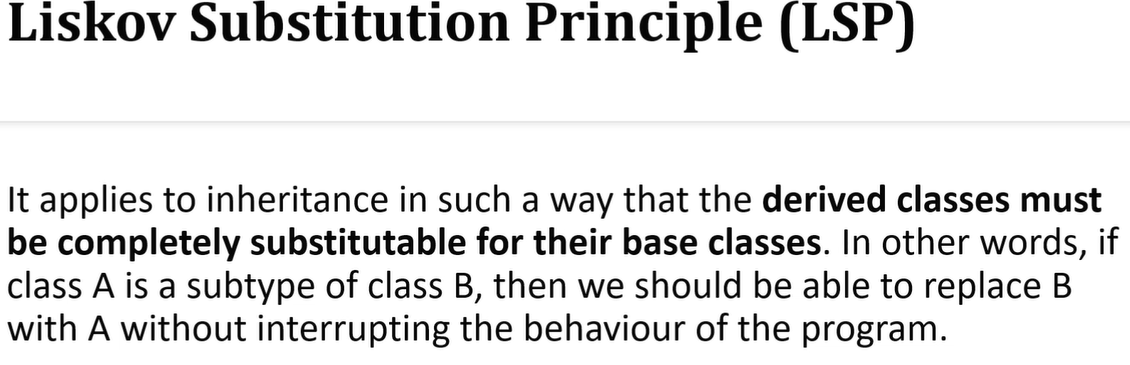
Which means in simple words,

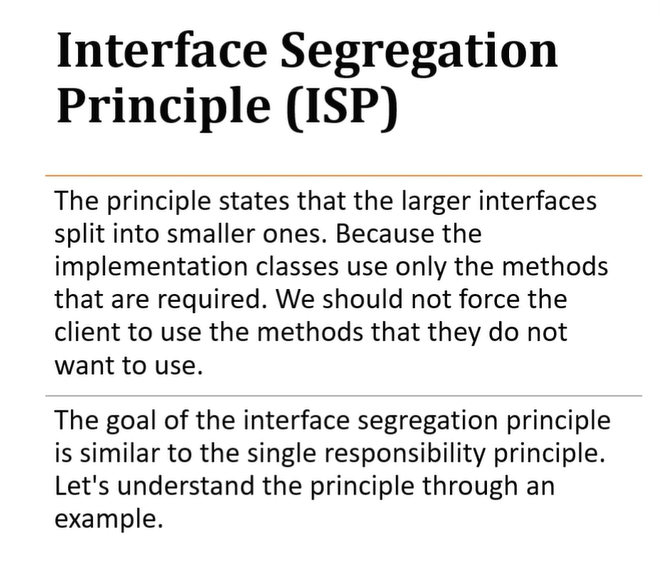
We should be able to extend the class behavior without modifying it.

Example :



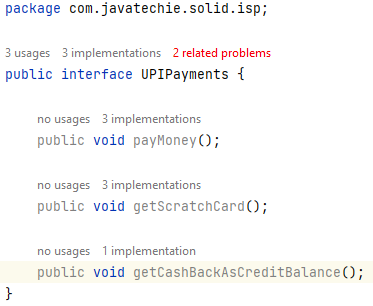


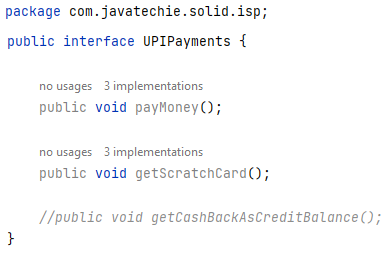




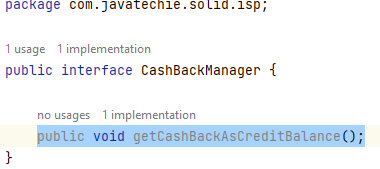
For below example:

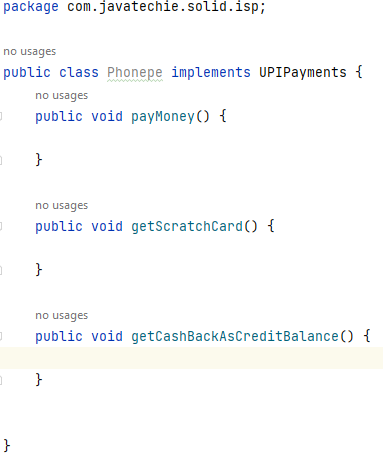
I have created one Interface UPIPayments.



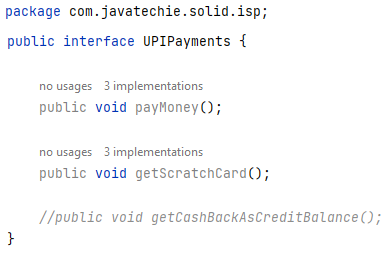


And I have created class **Phonepe** and implements **UPIPayments** then all method should be overriding force fully in this class but avoid this problem we have to create another **interface CashBackManager** Interface and inside this interface create a method below.



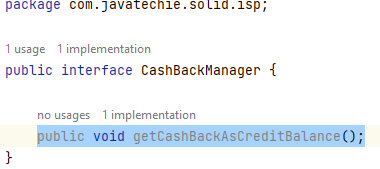


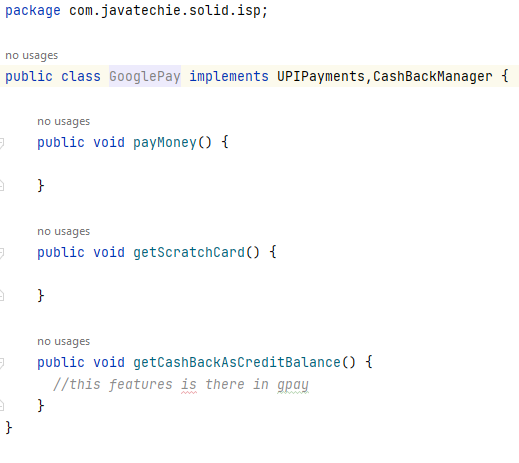
When we are removing a one method from UPIPayment Interface then after implements in PhonePe class.

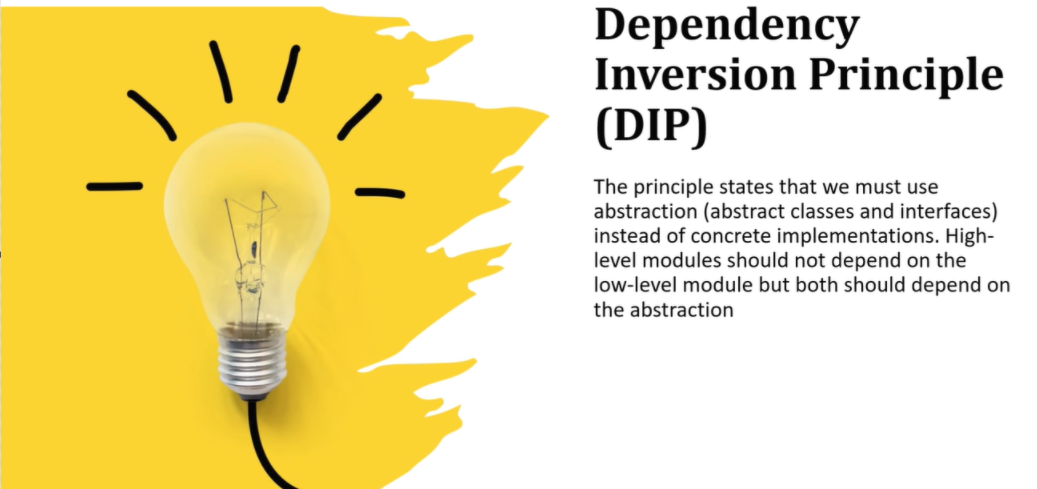




And below I have create another class GooglePay and we have need to Implement interface CashBackManager which means if you don’t’ want force fullly implement the behavior of interface then you have create another interface like below and create some method in that and then after implement it in your class just like GooglePay class.

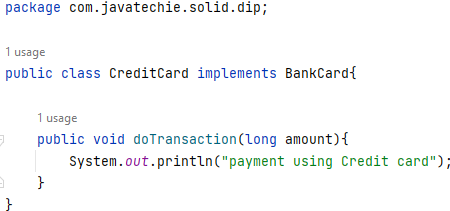






Step1.

I have create two classes :



Step2.



Step3,

I have create another class ShoppingMall and I want to call DebitCard and CreditCard class here and Some time I am getting problem in DebitCard class then goes to CreditCard then we have to change code level means it’s title coupled so resolved this problem we have to create one interface BankCard and implement in each classes and then after used this Interface in ShopingMall class and it’s being loosely coupled. And it’s work as excepted as above designed pattern.

