The Dependency Inversion principle states that our classes should depend upon interfaces or abstract classes instead of concrete classes and functions.

Let’s understand it with an example,

Suppose, there is a class called **Person** having dependency of **RedmiMobile** class

**RedmiMobile.java**

Text

Description automatically generated

**Person.java**

**Graphical user interface, text, application, website

Description automatically generated**

So when a client try to access the doCalling() method of **Person** class, it will execute perfectly.

Graphical user interface, text

Description automatically generated

**Output :**

Graphical user interface, text, application

Description automatically generated

But in terms of designing there is a big problem in the above code.

**Problem :**

A person don’t stick with the same mobile more than 2 years, so suppose if the person class want to change their mobile from Redmi to OnePlus. Then what we need to is

* Remove the existing dependency from **Person** class
* Add new mobile dependency in **Person** class

So this is again breaking our **Open/Closed Principle**.

As per the definition of **DIP,**  our **Person** class is not depend on Interface/Absract class rather it is depending on a Concrete class i.e. **Redmi**

So to avoid both the above problem, we should introduce a **Mobile** interface in our project and **Redmi** and **OnePlus** class should implements the **Mobile** interface.

Text

Description automatically generated

Text

Description automatically generated

Now the **Person** class should depend on **Mobile** interface instead of a **Redmi** or **OnePlus**

**Graphical user interface, text, application, email

Description automatically generated**

Now a valid question can come in your mind is that still if again wants to change our mobile from oneplus to iphone then we have to still modify our Person class as

new Iphone();

at 5th line.

But that is not the case when it comes to providing object to a class, the Spring framework do this job very well.

So, with the help of spring we can easily inject whatever dependency we have in our **Person** class.