**Dependency Inversion Principle**

* **Introduction:**

High level modules should not depend on low level modules. Both should depend on abstractions.

Abstractions should not depend on details. Details should depend on abstractions.

A close up of a map

Description automatically generated

A close up of text on a white background

Description automatically generated

Let’s take the Product Catalog for further investigation.

A screenshot of a social media post

Description automatically generated

In ProductCatalog analysis ProductCatalog is directly depends on SQLProductRepository. Which violates the princinple.

Principle says both has to depend on abstraction.

Let’s create an interface named ProductRepository.

SQLProductRepository has to implement ProductRepository.

A screenshot of a social media post

Description automatically generated

Now we have to inject this in ProductCatalog, so let’s introduce a factory class which returns a new instance of SQLProductRepository.

A screenshot of a social media post

Description automatically generated

Finally the dependency looks like below.

A screenshot of a social media post

Description automatically generated

Now the dependency direction has changed as below.

A close up of a sign

Description automatically generated

This inversion in the dependency arrow is the reason behind the principle name, dependency inversion.

* **Dependency Injection:**

As per the above implementation ProductCatalog is depend on ProductFactory to instantiate the SQLProductRepository.

We don’t want ProductCatalog class to worry about when and how to instantiate the ProductRepository.

We can provide an instantiated ProductRepository to the ProductCatalog

Without even asking.

Finally the dependency looks like below.

A screenshot of a social media post

Description automatically generated

In the above design we are injecting dependency of ProductCatalog instead of ProductCatalog instantiating the dependency.