Design Patterns : Dependency Injection

Dependency injection solves the problem of having classes that are tightly coupling by moving the creation of objects and the binding of dependencies outside of the class that depends on them.

Without dependency injection, classes can only use hard-coded dependencies that are instantiated within the class itself. This makes it difficult to replace dependencies or to reuse a class for different purposes. Dependency injection allows us to inject dependencies into a class from the outside, making the class more flexible and reusable.

Instead of classes building the objects they depend on, they are passed in as parameters (usually in the constructor), at instantiation time.

Why use dependency injection?

This makes the code more testable, as the dependencies can be mocked and passed in. This also helps with decoupling, as the class does not need to know how to build the objects it depends on.

Decoupling means reducing the interdependence between different parts of the code. When two components are tightly coupled, they are highly dependent on each other, and changes made to one component can cause unexpected behavior or even break the other component. On the other hand, when components are decoupled, they are less dependent on each other, and changes made to one component do not affect the behavior of the other component as much. Making the dependencies easier replaced or modified without affecting the behavior of other objects in the systems.

The reason it makes testing easier is say you are testing a class, without dependency injection, it would create an instance of something it is relying on. In the example below, the class is relying on the database class. If you are testing the class, you are also testing the database class. This is not ideal, as you want to test the class in isolation. With dependency injection, you can pass in a mock database class, which will allow you to test the class in isolation.

Another reason to use dependency injection is that you can make one instance of a class and pass it to multiple classes. This is useful if you want to share data between classes. This also makes the code more efficient, as you are not creating multiple instances of the same class. When you make multiple instances of the same class, you are using more memory, and it is more work for the garbage collector to clean up the memory.

When might you use dependency injection?

An example where you would use dependency injection is if you are developing a email app. This email app needs to have services such as gmail, outlook, yahoo, etc, which are the dependencies that get injected into the email app.

Example without dependency injection

```
public class OurSystem{
    private Database database;

    public OurSystem(){
        this.database = new Database();
    }

}
```

Example with dependency injection

```
public class OurSystem{
    private Database database;

    public OurSystem(Database database){
        this.database = database;
    }

}
```

```
public class Main{
    public static void main(String[] args){
        // dependencies
        Database database = new Database();

        // inject dependencies
        OurSystem system = new OurSystem(database);
    }
}
```

Frameworks

As the application gets bigger, it becomes more difficult to manage dependencies. To solve this problem, we can use a dependency injection framework, such as Spring or Guice that can help manage dependencies. We leave the dependency injection to the framework, and we just need to tell the framework what dependencies we need.