**Why Do We Need the Dependency Injection Design Pattern in C#?**

The Dependency Injection Design Pattern in C# allows us to develop Loosely Coupled Software Components. In other words, we can say that Dependency Injection Design Pattern is used to reduce the Tight Coupling between the Software Components. As a result, we can easily manage future changes and other complexities in our application. In this case, if we change one component, then it will not impact the other components.

##### ****What is Tight Coupling in Software Design?****

**Tight Coupling means two objects are dependent on each other.** That means when a class is dependent on another class, then it is said to be a tight coupling between these two classes. In that case, if we change the Dependent Object, then we also need to change the classes where this dependent object is being used. If your application is a small one, then it is not that difficult to handle the changes but if you have a big Enterprise-Level application, then it’s really very difficult to handle these changes.

##### ****What is Loose Coupling in Software Design?****

**Loosely Coupling means two objects are independent of each other.** That means if we change one object then it will not affect another object. The loosely coupled nature of software development allows us to manage future changes easily and also allows us to manage the complexity of the application.

##### ****What is Dependency Injection Design Pattern in C#?****

The Dependency Injection Design Pattern in C# is a process in which we are injecting the dependent object of a class into a class that depends on that object. The Dependency Injection Design Pattern is the most commonly used design pattern nowadays to remove the dependencies between the objects.

 The Dependency Injection Design Pattern involves 3 types of classes:

* + 1. **Client Class:** The Client Class (dependent class) is a class that depends on the Service Class. That means the Client Class wants to use the Services (Methods) of the Service Class.
    2. **Service Class:** The Service Class (dependency) is a class that provides the actual services to the client class.
    3. **Injector Class:** The Injector Class is a class that injects the Service Class object into the Client Class.

##### ****Different Types of Dependency Injection in C#:****

1. **Constructor Injection:** When the Injector Injects the Dependency Object (i.e. Service Object) into the Client Class through the Client Class Constructor, then it is called Constructor Dependency Injection.
2. **Property Injection:** When the Injector Injects the Dependency Object (i.e. Service Object) into the Client Class through the public Property of the Client Class, then it is called Property Dependency Injection. This is also called the Setter Injection.
3. **Method Injection:** When the Injector Injects the Dependency Object (i.e. Service Object) into the Client Class through a public Method of the Client Class, then it is called Method Dependency Injection.

**Example to Understand Dependency Injection Design Pattern in C#:**

Let us understand the Dependency Injection Design Pattern in C# with an Example. Let us first create a Console Application with the name DependencyInjectionExample. Once you create the Console Application, next we are going to create 3 classes **Employee, EmployeeDAL,**and **EmployeeBL.**

**Employee.cs**

namespace DependencyInjectionDesignPattern

{

//This is going to be our Model class which holds the Model data

//This class is going to be used by both EmployeeDAL and EmployeeBL

public class Employee

{

public int ID { get; set; }

public string Name { get; set; }

public string Department { get; set; }

}

}

**EmployeeDAL.cs (Service)**

using System.Collections.Generic;

namespace DependencyInjectionDesignPattern

{

//Service Class or Dependency Object

//This is the class that is responsible for Interacting with the Database

//This class is going to be used by the EmpoloyeeBL class

//That means it is going to be the Dependency Object

public class EmployeeDAL

{

public List<Employee> SelectAllEmployees()

{

List<Employee> ListEmployees = new List<Employee>

{

//Get the Employees from the Database

//for now we are hard coded the employees

new Employee() { ID = 1, Name = "Pranaya", Department = "IT" },

new Employee() { ID = 2, Name = "Kumar", Department = "HR" },

new Employee() { ID = 3, Name = "Rout", Department = "Payroll" }

};

return ListEmployees;

}

}

}

**EmployeeBL.cs (Client)**

using System.Collections.Generic;

namespace DependencyInjectionDesignPattern

{

//Client Class or Dependent Object

//This is the Class that is going to consume the services provided by the EmployeeDAL Class

//That means it is the Dependent Class which is Depending on the EmployeeDAL Class

public class EmployeeBL

{

public EmployeeDAL employeeDAL;

public List<Employee> GetAllEmployees()

{

//Creating an Instance of Dependency Class means it is a Tight Coupling

employeeDAL = new EmployeeDAL();

return employeeDAL.SelectAllEmployees();

}

}

}

This is a Tight Coupling because the EmployeeDAL is tightly coupled with the EmployeeBL class. Every time the EmployeeDAL class changes, the EmployeeBL class also needs to change. This is the problem. Let us see how to use the Constructor Dependency Injection to make these classes Loosely coupled.

##### ****Using Constructor Dependency Injection Design Pattern in C#****

##### 

Let us see how we can use the Constructor Dependency Injection Design Pattern in C# to make these classes loosely coupled. So, first Modify the **EmployeeDAL.cs** class file as shown below. As you can see in the below code, first we create one interface i.e **IEmployeeDAL** with one abstract method i.e. **SelectAllEmployees**. Then that interface is implemented by the EmployeeDAL class and provides implementations for the abstract SelectAllEmployees method.

**EmployeeDAL.cs (Service)**

using System.Collections.Generic;

namespace DependencyInjectionDesignPattern

{

//Service Class or Dependency Object

//Dependency Object should be Interface-Based

public interface IEmployeeDAL

{

List<Employee> SelectAllEmployees();

}

//This is the class that is responsible for Interacting with the Database

//This class is going to be used by the EmpoloyeeBL class

//That means it is going to be the Dependency Object

public class EmployeeDAL : IEmployeeDAL

{

public List<Employee> SelectAllEmployees()

{

List<Employee> ListEmployees = new List<Employee>

{

//Get the Employees from the Database

//for now we are hard coded the employees

new Employee() { ID = 1, Name = "Pranaya", Department = "IT" },

new Employee() { ID = 2, Name = "Kumar", Department = "HR" },

new Employee() { ID = 3, Name = "Rout", Department = "Payroll" }

};

return ListEmployees;

}

}

}

**EmployeeBL.cs (Client)**

using System.Collections.Generic;

namespace DependencyInjectionDesignPattern

{

//Client Class or Dependent Object

//This is the Class that is going to consume the services provided by the IEmployeeDAL Class

//That means it is the Dependent Class which Depending on the IEmployeeDAL Class

public class EmployeeBL

{

public IEmployeeDAL employeeDAL;

//Injecting the Dependency Object using Constructor means it is a Loose Coupling

public EmployeeBL(IEmployeeDAL employeeDAL)

{

this.employeeDAL = employeeDAL;

}

public List<Employee> GetAllEmployees()

{

return employeeDAL.SelectAllEmployees();

}

}

}

So here in the EmployeeBL class, we are not creating the object of the EmployeeDAL class. Instead, we are passing it as a parameter to the constructor of the EmployeeBL class. As we are Injecting the Dependency Object through the constructor, it is called Constructor Dependency Injection in C#.

**Injector Class**

The Injector class will Inject the Dependency Object (EmplpyeeDAL Object) into the Client Class (EmplpyeeBL Class). In our example, the Main method of the Program class is going to be the Injector Class. So, the Injector class will decide which EmployeeDAL instance to be used by the EmployeeBL class, and then the Injector will Inject that instance into the EmployeeBL class through the EmployeeBL class constructor as we are using Constructor Dependency Injection.

using System;

using System.Collections.Generic;

namespace DependencyInjectionDesignPattern

{

class Program

{

static void Main(string[] args)

{

//Create an Instance of EmployeeBL and Inject the Dependency Object as an Argument to the Constructor

EmployeeBL employeeBL = new EmployeeBL(new EmployeeDAL());

List<Employee> ListEmployee = employeeBL.GetAllEmployees();

foreach (Employee emp in ListEmployee)

{

Console.WriteLine($"ID = {emp.ID}, Name = {emp.Name}, Department = {emp.Department}");

}

Console.ReadKey();

}

}

}

