**Dependency Injection**

Dependency Injection (DI) is a design pattern used to implement IoC (Inversion of control). It allows the creation of dependent objects outside of a class and provides those objects to a class through different ways. Using DI, we move the creation and binding of the dependent objects outside of the class that depends on them.

Also, dependency injection is a technique in which an object receives other objects that it depends on. These objects are called dependencies. The injection refers to the passing of a dependency (a service) into the object (a client) that would use it.

The Dependency Injection pattern involves 3 types of classes.

1. **Client Class:** The client class (dependent class) is a class which depends on the service class
2. **Service Class:** The service class (dependency) is a class that provides service to the client class.
3. **Injector Class:** The injector class injects the service class object into the client class.

**Types of Dependency Injection**

As you have seen above, the injector class injects the service (dependency) to the client (dependent). The injector class injects dependencies broadly in three ways: through a constructor, through a property, or through a method.

1. **Constructor Injection:** In the constructor injection, the injector supplies the service (dependency) through the client class constructor.
2. **Property Injection:** In the property injection (aka the Setter Injection), the injector supplies the dependency through a public property of the client class.
3. **Method Injection:** In this type of injection, the client class implements an interface which declares the method(s) to supply the dependency and the injector uses this interface to supply the dependency to the client class.

Tim Corey began the concept of dependency injection by explaining the program he has written which contains loading data, processing data, saving processed info and logging finished processing of the data.

In Dependency Injection principle, it is better to make use of an interface in order to have a multiple inheritance. The underscore such **ILogger \_Logger;** is used when we don’t need to instantiate.

Tim Corey explained the AutoFac Nugget which is used to manage the dependencies between classes so that applications stay easy to change as they grow in size and complexity.

Dependency Injection is a way of having a specific task which performs a certain function inside a containment so when we need to make adjustment or further implementations we know exactly where to go, in dependency injection, one change can have effect on other applications because it is dependent on an object.

Tim Corey explained the configure method() which is used to configure the container. **The builder.Build()** is used to build the container.

The container used in the implementation is used to store all of the information of the classes. The keyword **RegisterType and As** is used to register a class to an interface so whenever we see that interface the class should respond with the instance of the class.

For instance:

**Builder.RegisterType<BusinesLogic>().As<IBusinessLogic>();**

In dependency injection, one application can switch the other applications instead of having to change all of them one after the other which is time consuming.