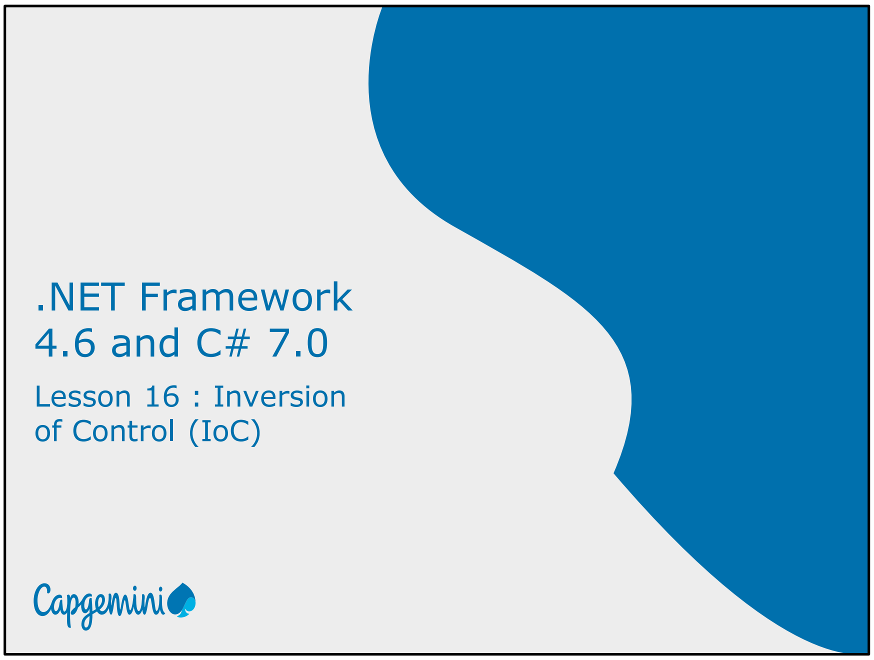
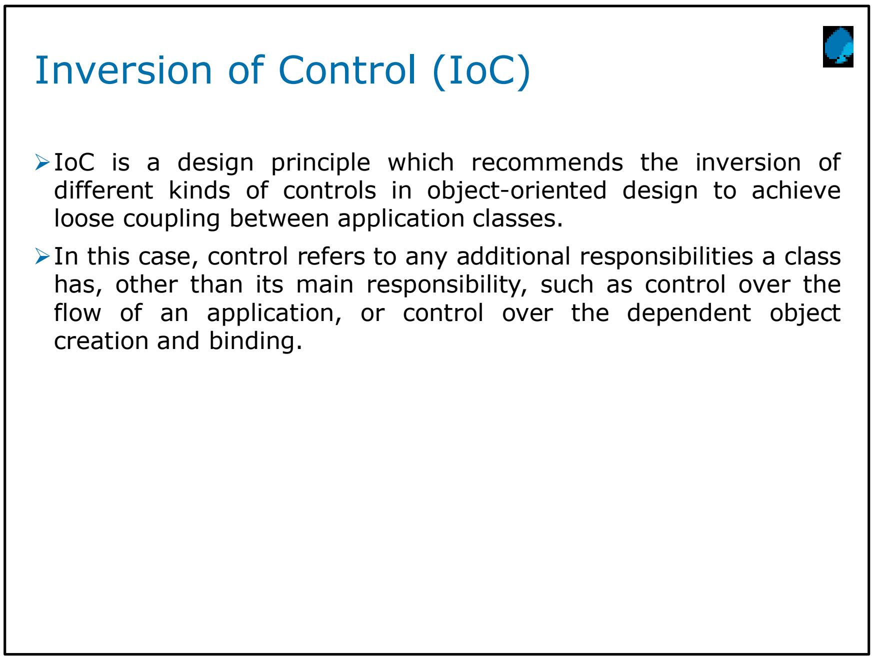


.NET Framework 4.6 and C# 7.0

Lesson 16 : Inversion of Control (IoC)

Lesson Objective

- In this lesson, you will learn:
  - Inversion of Control (IoC)
  - Dependency Inversion Principle (DIP)
  - Dependency Injection
  - Types of Dependency Injection
  - IoC Container



Inversion of Control (IoC)

- IoC is a design principle which recommends the inversion of different kinds of controls in object-oriented design to achieve loose coupling between application classes.
- In this case, control refers to any additional responsibilities a class has, other than its main responsibility, such as control over the flow of an application, or control over the dependent object creation and binding.

Inversion of Control (IoC) is a design principle (although, some people refer to it as a pattern). As the name suggests, it is used to invert different kinds of controls in object-oriented design to achieve loose coupling. Here, controls refer to any additional responsibilities a class has, other than its main responsibility. This include control over the flow of an application, and control over the flow of an object creation or dependent object creation and binding.

IoC is all about inverting the control. To explain this in layman's terms, suppose you drive a car to your work place. This means you control the car. The IoC principle suggests to invert the control, meaning that instead of driving the car yourself, you hire a cab, where another person will drive the car. Thus, this is called inversion of the control - from you to the cab driver. You don't have to drive a car yourself and you can let the driver do the driving so that you can focus on your main work.

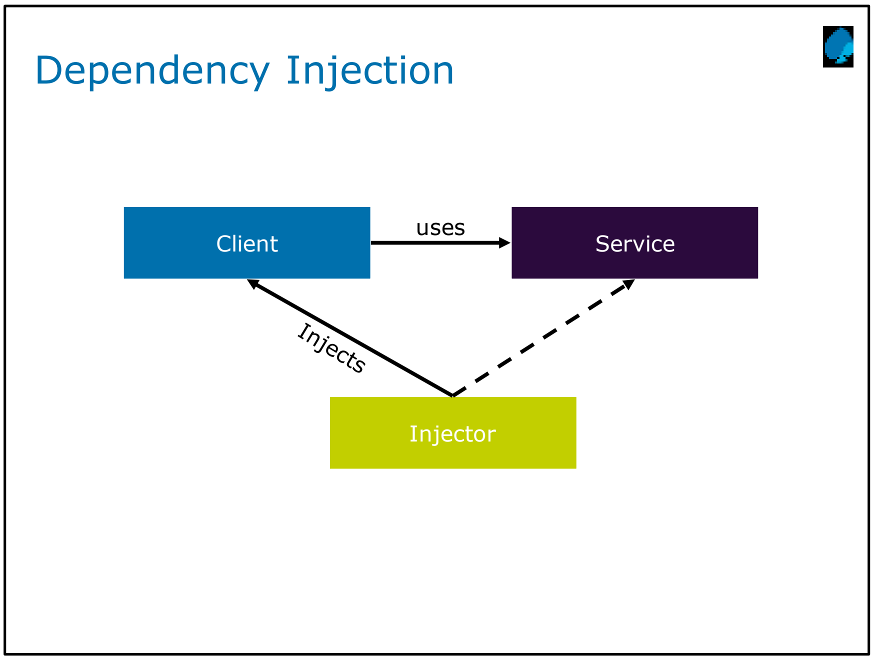
The IoC principle helps in designing loosely coupled classes which make them testable, maintainable and extensible.

Dependency Inversion Principle (DIP)

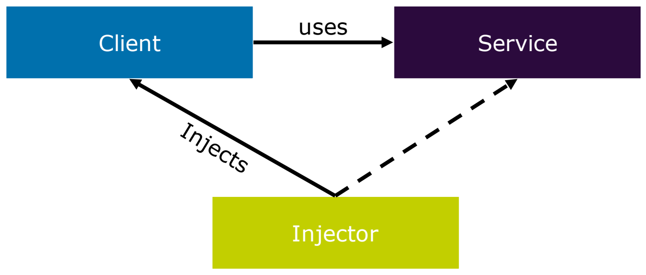
- The DIP principle also helps in achieving loose coupling between classes.
- It is highly recommended to use DIP and IoC together in order to achieve loose coupling.
- DIP suggests that high-level modules should not depend on low level modules. Both should depend on abstraction.
- Abstractions should not depend on details. Details should depend on abstractions.
- The DIP principle was invented by Robert Martin (a.k.a. Uncle Bob).

Dependency Injection

- Dependency Injection (DI) is a design pattern which implements the IoC principle to invert the creation of dependent objects.
- 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.
- The Dependency Injection pattern involves 3 types of classes.
  - Client Class: The client class (dependent class) is a class which depends on the service class
  - Service Class: The service class (dependency) is a class that provides service to the client class.
  - Injector Class: The injector class injects the service class object into the client class.



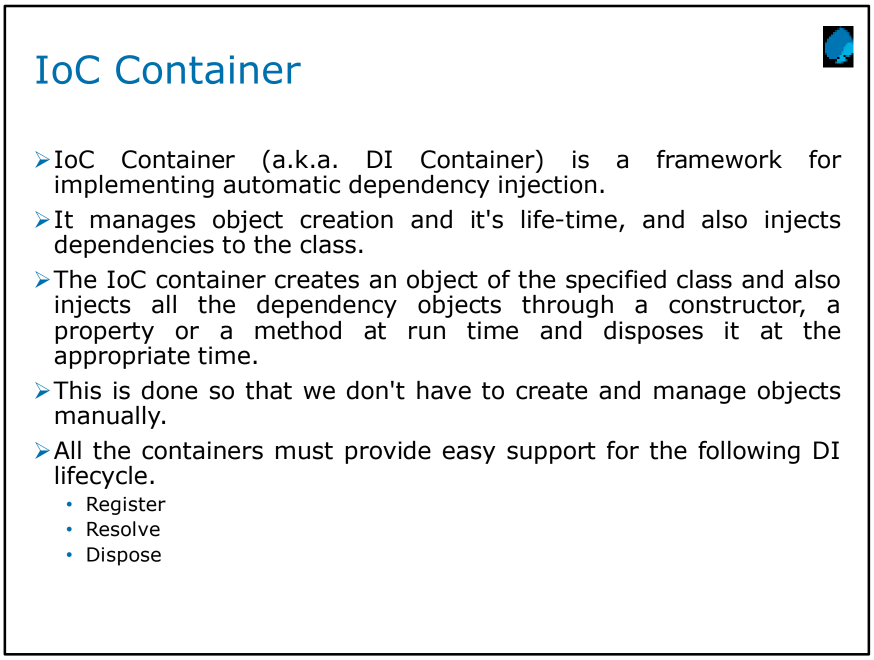
Dependency Injection



As you can see, the injector class creates an object of the service class, and injects that object to a client object. In this way, the DI pattern separates the responsibility of creating an object of the service class out of the client class.

Types of Dependency Injection

- 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.
- Constructor Injection
  - In the constructor injection, the injector supplies the service (dependency) through the client class constructor.
- Property Injection
  - In the property injection (aka the Setter Injection), the injector supplies the dependency through a public property of the client class.
- 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.



IoC Container

- IoC Container (a.k.a. DI Container) is a framework for implementing automatic dependency injection.
- It manages object creation and it's life-time, and also injects dependencies to the class.
- The IoC container creates an object of the specified class and also injects all the dependency objects through a constructor, a property or a method at run time and disposes it at the appropriate time.
- This is done so that we don't have to create and manage objects manually.
- All the containers must provide easy support for the following DI lifecycle.
  - Register
  - Resolve
  - Dispose

IoC Container (a.k.a. DI Container) is a framework for implementing automatic dependency injection. It manages object creation and it's life-time, and also injects dependencies to the class.

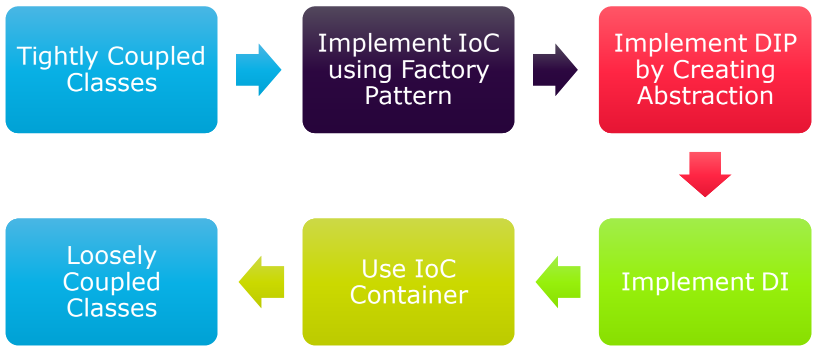
The IoC container creates an object of the specified class and also injects all the dependency objects through a constructor, a property or a method at run time and disposes it at the appropriate time. This is done so that we don't have to create and manage objects manually.

All the containers must provide easy support for the following DI lifecycle.

Register: The container must know which dependency to instantiate when it encounters a particular type. This process is called registration. Basically, it must include some way to register type-mapping.
Resolve: When using the IoC container, we don't need to create objects manually. The container does it for us. This is called resolution. The container must include some methods to resolve the specified type; the container creates an object of the specified type, injects the required

dependencies if any and returns the object.
Dispose: The container must manage the lifetime of the dependent objects. Most IoC containers include different lifetimemanagers to manage an object's lifecycle and dispose it.

Process


Tightly Coupled Classes
Implement IoC using Factory Pattern
Implement DIP by Creating Abstraction
Implement DI
Use IoC Container
Loosely Coupled Classes

Summary

- In this lesson, you have learnt:
  - IoC is a design principle which recommends the inversion of different kinds of controls in object-oriented design to achieve loose coupling between application classes.
  - The DIP principle also helps in achieving loose coupling between classes.
  - DI is a design pattern which implements the IoC principle to invert the creation of dependent objects.
  - IoC Container (a.k.a. DI Container) is a framework for implementing automatic dependency injection.