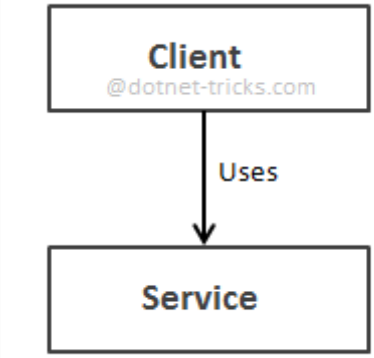
Dependency Injection Pattern in C#

Dpendency Injection (DI) is **a software design pattern** that allow us to **develop loosely coupled** code. DI is a great way to reduce tight coupling between software components. DI also enables us to better **manage future changes** and other complexity in our software. The purpose of DI is to make code **maintainable**.

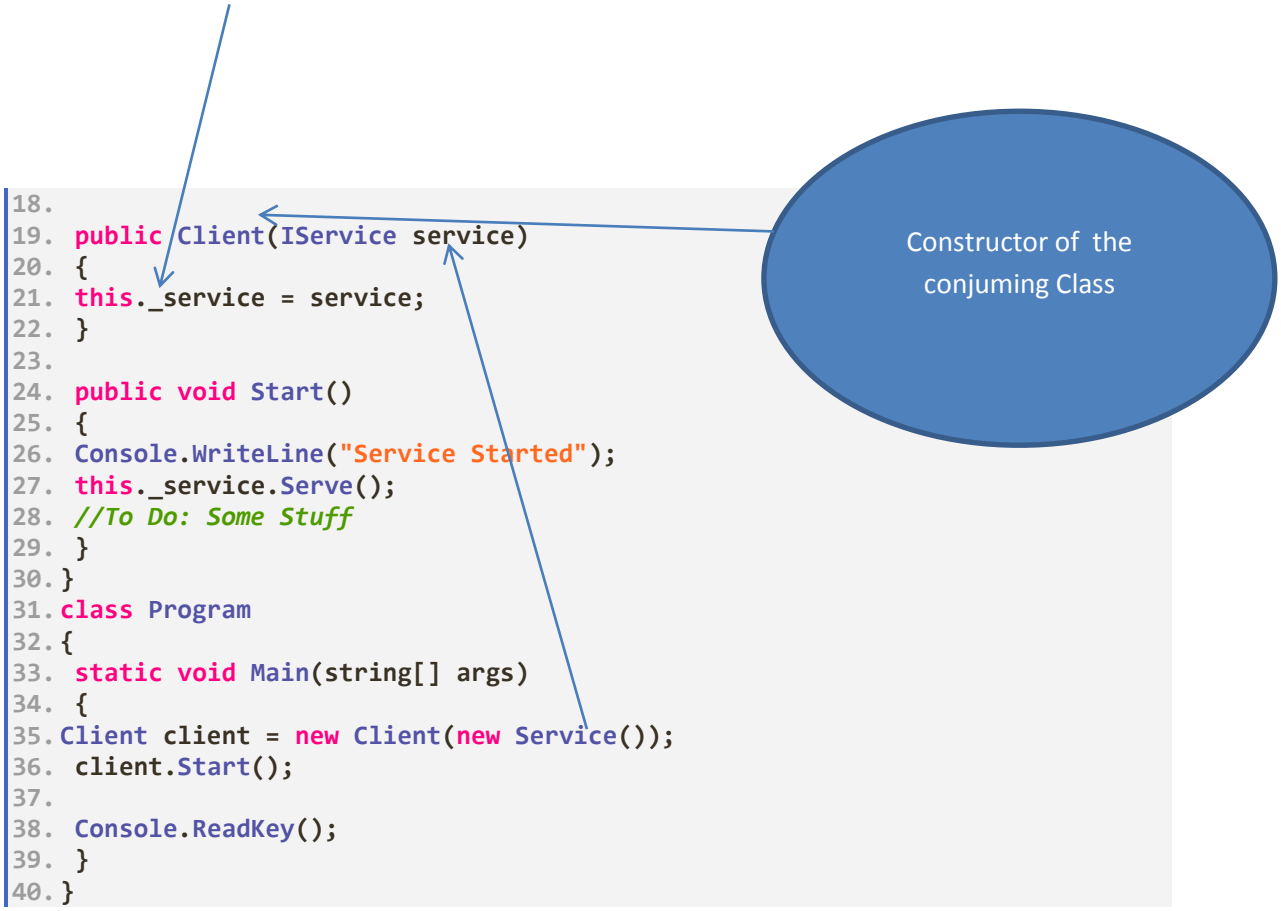
For example, Suppose your `Client` class needs to use a `Service` class component, then the best you can do is to make your `Client` class aware of an `IService` interface rather than a `Service` class. In this way, you can change the implementation of the `Service` class at any time (and for how many times you want) without breaking the host code.



Constructor Injection

1. This is the most common DI.
2. Dependency Injection is done by supplying the DEPENDENCY through the class's constructor when instantiating that class.
3. Injected component can be used anywhere within the class.
4. Should be used when the injected dependency is required for the class to function.
5. It addresses the most common scenario where a class requires one or more dependencies.

```
1. public interface IService
2. {
3.  void Serve();
4. }
5.
6. public class Service : IService
7. {
8.  public void Serve()
9.  {
10. Console.WriteLine("Service Called");
11. //To Do: Some Stuff
12. }
13.}
14.
15.public class Client
16.{
17. private IService _service;
```




```
18.
19.  public Client(IService service)
20.  {
21.  this._service = service;
22.  }
23.
24.  public void Start()
25.  {
26.  Console.WriteLine("Service Started");
27.  this._service.Serve();
28.  //To Do: Some Stuff
29.  }
30.}
31.class Program
32.{
33.  static void Main(string[] args)
34.  {
35.Client client = new Client(new Service());
36.  client.Start();
37.
38.  Console.ReadKey();
39.  }
40.}
```

The Injection happens in the constructor, by passing the Service that implements the IService-Interface. The dependencies are assembled by a "Builder" and Builder responsibilities are as follows:

1. knowing the types of each IService
2. according to the request, feed the abstract IService to the Client

Key points about DI

1. Reduces class coupling
2. Increases code reusing
3. Improves code maintainability
4. Improves application testing