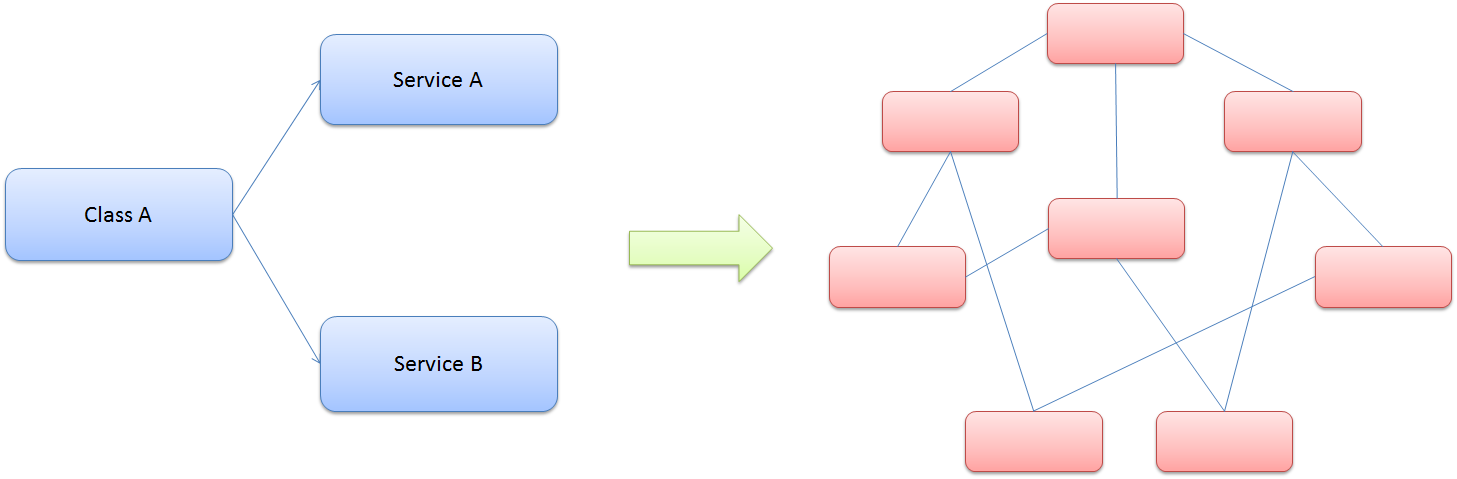
IoC is a principle, or an architectural pattern. There are quite a few definitions of IoC, but the basis behind it is always the same - it helps towards a loosely coupled architecture. DI is one way in which IoC can be applied, according to one school of thought.  DI is IoC according to another.

In **Object Oriented Programming** paradigm, objects work together in a collaboration model where there are contributors and consumers. Naturally, this communication model generates dependencies between objects and components, becoming difficult to manage when complexity increases.



*Class dependencies and model complexity*

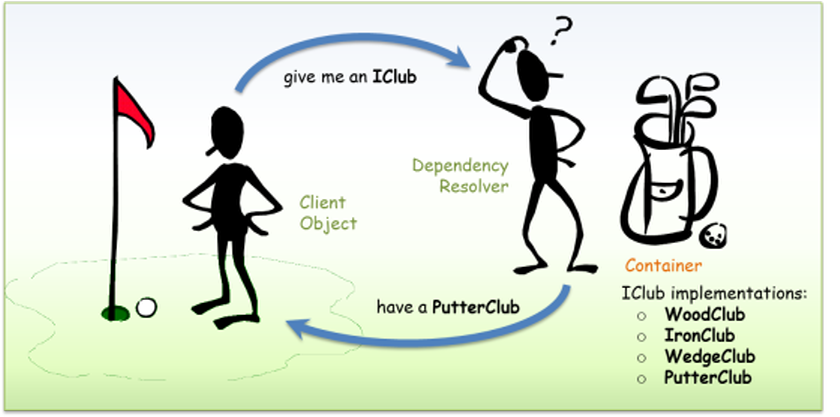
You have probably heard about the **Factory Pattern** and the separation between the interface and the implementation using services, where the client objects are often responsible for service location.

The Dependency Injection pattern is a particular implementation of Inversion of Control. **Inversion of Control (IoC)** means that objects do not create other objects on which they rely to do their work. Instead, they get the objects that they need from an outside source (for example, an xml configuration file).

**Dependency Injection (DI)** means that this is done without the object intervention, usually by a framework component that passes constructor parameters and set properties.

The Dependency Injection (DI) Design Pattern

At a high level, the goal of Dependency Injection is that a client class (e.g. *the golfer*) needs something that satisfies an interface (e.g. *IClub*). It doesn't care what the concrete type is (e.g. *WoodClub, IronClub, WedgeClub* or *PutterClub*), it wants someone else to handle that (e.g. a good *caddy*). The Dependency Resolver in ASP.NET MVC can allow you to register your dependency logic somewhere else (e.g. a container or a *bag of clubs*).



*Dependency Injection - Golf analogy*

The advantages of using Dependency Injection pattern and Inversion of Control are the following:

* Reduces class coupling
* Increases code reusing
* Improves code maintainability
* Improves application testing

**Note:** Dependency Injection is sometimes compared with Abstract Factory Design Pattern, but there is a slight difference between both approaches. DI has a Framework working behind to solve dependencies by calling the factories and the registered services.

Now that you understand the Dependency Injection Pattern, you will learn throughout this lab how to apply it in ASP.NET MVC 4. You will start using Dependency Injection in the **Controllers** to include a database access service. Next, you will apply Dependency Injection to the **Views** to consume a service and show information. Finally, you will extend the DI to ASP.NET MVC 4 Filters, injecting a custom action filter in the solution.