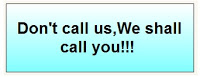
**Dependency Injection**

The basic concept of the dependency injection (also known as Inversion of Control pattern) is that you do not create your objects but describe how they should be created. You don't directly connect your components and services together in code but describe which services are needed by which components in a configuration file. A container (in the case of the Spring framework, the IOC container) is then responsible for hooking it all up.  
i.e., Applying IoC, objects are given their dependencies at creation time by some external entity that coordinates each object in the system. That is, dependencies are injected into objects. So, IoC means an inversion of responsibility with regard to how an object obtains references to collaborating objects.   
  


**Example:**

 Lets we have two classes-**Car**and **Engine.**Car has a object of Engine.

**Normal way:**

There are many ways to instantiate a object. A simple and common way is with new operator.

so here Car class contain object of Engine and we have it instantiated using new operator.

|  |
| --- |
| [<http://2.bp.blogspot.com/-CmdH3WuA99Y/UDUKlTBRSZI/AAAAAAAAAOM/co_ymbfhVtI/s1600/CarEngineClassDiagram.jpg>](http://2.bp.blogspot.com/-CmdH3WuA99Y/UDUKlTBRSZI/AAAAAAAAAOM/co_ymbfhVtI/s1600/CarEngineClassDiagram.jpg) |
| WITHOUT DI |

**With help of Dependency Injection:**

Now we outsource instantiation and supply job of instantiating to third party.**Car** needs object of **Engine** to operate but it outsources that job to some third party. The designated third party, decides the moment of instantiation and the type to use to create the instance. The dependency between class **Car** and class **Engine** is injected by a third party. Whole of this agreement involves some configuration information too. This whole process is called dependency injection.

|  |
| --- |
| [[http://2.bp.blogspot.com/-r0yOW6IvP30/UDUPcunS7UI/AAAAAAAAAOg/EV46c8RCw6w/s640/DIEngineCar.jpg](http://2.bp.blogspot.com/-r0yOW6IvP30/UDUPcunS7UI/AAAAAAAAAOg/EV46c8RCw6w/s1600/DIEngineCar.jpg)](http://2.bp.blogspot.com/-r0yOW6IvP30/UDUPcunS7UI/AAAAAAAAAOg/EV46c8RCw6w/s1600/DIEngineCar.jpg) |

## WITH DI

## BENIFITS OF DEPENDENCY INJECTION IN SPRING:

* Ensures configuration and uses of services are separate.
* Can switch implementations by just changing configuration.
* Enhances Testability as mock dependencies can be injected.
* Dependencies can be easily identified.
* No need to read code to see what dependencies your code talks to.

## TYPES OF DEPENDENCY INJECTION:

* [**Setter Injection**](http://javapostsforlearning.blogspot.in/2012/08/dependency-injection-via-setter-method.html):Setter-based DI is realized by calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.
* [**Constructor Injection**](http://javapostsforlearning.blogspot.in/2012/08/dependency-injection-via-constructor-in.html):Constructor-based DI is realized by invoking a constructor with a number of arguments, each representing a collaborator.
* **Inteface Injection**: In interface-based dependency injection, we will have an interface and on implementing it we will get the instance injected.