The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly.

There are two types of IoC containers. They are:

1. **BeanFactory**
2. **ApplicationContext**

Difference between BeanFactory and the ApplicationContext

The org.springframework.beans.factory.**BeanFactory** and the org.springframework.context.**ApplicationContext** interfaces acts as the IoC container. The ApplicationContext interface is built on top of the BeanFactory interface. It adds some extra functionality than BeanFactory such as simple integration with Spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. WebApplicationContext) for web application. So it is better to use ApplicationContext than BeanFactory.

The objects that form the backbone of your application and that are managed by the Spring IoC container are called **beans**.

The **BeanPostProcessor** interface defines callback methods that you can implement to provide your own instantiation logic, dependency-resolution logic, etc.

A bean definition can contain a lot of configuration information, including constructor arguments, property values, and container-specific information such as initialization method, static factory method name, and so on.

A child bean definition inherits configuration data from a parent definition. The child definition can override some values, or add others, as needed.

DI exists in two major variants and the following two sub-chapters will cover both of them with examples −

|  |  |
| --- | --- |
| **Sr.No.** | **Dependency Injection Type & Description** |
| 1 | [**Constructor-based dependency injection**](https://www.tutorialspoint.com/spring/constructor_based_dependency_injection.htm)  Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on the other class. |
| 2 | [**Setter-based dependency injection**](https://www.tutorialspoint.com/spring/setter_based_dependency_injection.htm)  Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean. |

You can mix both, Constructor-based and Setter-based DI but it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies.

The code is cleaner with the DI principle and decoupling is more effective when objects are provided with their dependencies. The object does not look up its dependencies and does not know the location or class of the dependencies, rather everything is taken care by the Spring Framework.

The Spring container can **autowire** relationships between collaborating beans without using <constructor-arg> and <property> elements, which helps cut down on the amount of XML configuration you write for a big Spring-based application.

|  |  |
| --- | --- |
| **Sr.No.** | **Limitations & Description** |
| 1 | **Overriding possibility**  You can still specify dependencies using <constructor-arg> and <property> settings which will always override autowiring. |
| 2 | **Primitive data types**  You cannot autowire so-called simple properties such as primitives, Strings, and Classes. |
| 3 | **Confusing nature**  Autowiring is less exact than explicit wiring, so if possible prefer using explict wiring. |

# **Difference between constructor and setter injection**

There are many key differences between constructor injection and setter injection.

1. **Partial dependency**: can be injected using setter injection but it is not possible by constructor. Suppose there are 3 properties in a class, having 3 arg constructor and setters methods. In such case, if you want to pass information for only one property, it is possible by setter method only.
2. **Overriding**: Setter injection overrides the constructor injection. If we use both constructor and setter injection, IOC container will use the setter injection.
3. **Changes**: We can easily change the value by setter injection. It doesn't create a new bean instance always like constructor. So setter injection is flexible than constructor injection.

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.