**DEPENDENCY INJECTION:**

* In software design, dependency Injection is a design pattern that aims to decouple objects from their dependencies.
* Instead of creating their own dependencies internally, objects receive them from an external source.

**INVERSION OF CONTROL:**

* Spring IoC (Inversion of Control) Container is the core of Spring Framework.
* It creates the objects, configures and assembles their dependencies, manages their entire life cycle.
* The Container uses Dependency Injection(DI) to manage the components that make up the application.
* It gets the information about the objects from a configuration file(XML) or Java Code or Java Annotations and Java POJO class.
* These objects are called Beans.
* Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion Of Control.

**There are 2 types of IoC containers:**

* + - Bean Factory
    - Application context
* That means if you want to use an IoC container in spring whether we need to use a BeanFactory or ApplicationContext.
* The BeanFactory is the most basic version of IoC containers, and the ApplicationContext extends the features of BeanFactory.
* The followings are some of the main features of Spring IoC,
  + - Object Creation
    - Configuring Objects
    - Managing Lifecycle

**BEANS:**

* In Spring, the objects are the backbone of your application.
* The Objects that are managed by the Spring IoC Container are called Beans.
* A bean is instantiated, assembled, and managed by Spring IoC Container.

There are two types of Spring Dependency Injection.

1. Setter Dependency Injection (SDI)
2. Constructor Dependency Injection (CDI)

**Setter Dependency Injection (SDI)**

* In this, the Dependency Injection will be injected with the help of setter and/or getter methods.
* We initialize the beans using setter methods with the help of an annotation, i.e., @Autowired.
* That annotation will be used at the start of the setter method.

**Constructor Dependency Injection (CDI)**

* In Constructor Injection, the Dependency Injection will be injected with the help of constructors.
* We initialise the beans using constructors methods with the help of an annotation, i.e., @Autowired at the start of the constructor.

**Field Dependency Injection (FDI):**

* In Field Injection, the Dependency Injection will be injected with the help of fields.
* We initialise the beans using fields with the help of an annotation, i.e., @Autowired at the declaration of the field.

**AUTOWIRED ANNOTATION:**

* The @Autowired annotation in Spring is used for dependency injection.
* It allows Spring to automatically resolve and inject collaborating beans or dependencies into your application.
* When you use @Autowired, Spring will look for a bean that matches the type of the dependency and inject it where the annotation is placed.
* @Autowired helps Spring automatically manage the dependencies of your application, allowing you to focus on business logic without worrying about manual bean creation and injection.

**getBean():**

* In Spring, the getBean() function is used to retrieve beans or objects that are managed by the Spring IoC container by their name or type.
* This function is part of the ApplicationContext interface, which represents the Spring container responsible for managing the lifecycle of beans.
* The getBean() function allows you to access Spring beans from the IoC container.
* Beans are created and wired by the container, and through this function, you can fetch them and use them in your application.

**Overloaded getBean() Methods:**

Spring provides several overloaded versions of getBean() to give flexibility in how you fetch beans:

* getBean(String name): Retrieves the bean by its name.
* getBean(String name, Class<T> requiredType): Retrieves the bean by its name and type.
* getBean(Class<T> requiredType): Retrieves the bean by its type.
* getBean(String name, Object... args): Retrieves the bean by its name and provides constructor arguments to create a new bean instance.

**APPLICATION CONTEXT:**

* In Spring, the ApplicationContext is the central interface for providing configuration information to an application.
* It extends the BeanFactory interface, which is the core container in the Spring framework, and adds more enterprise-specific functionality.

**DIFFERENCE BETWEEN @Autowired and getBean():**

In Spring, both @Autowired and getBean() are used for dependency injection, but they work in slightly different ways:

* **@Autowired:** It is an annotation that tells Spring to automatically inject (or "wire") a required bean into a class field, constructor, or method. Spring does this automatically when it scans and creates the application context. You don’t need to manually retrieve the bean.
* **getBean():** This is a method you call manually to retrieve a bean from the Spring application context. You use it when you need to programmatically access a bean.