**Spring Framework:**

* Spring is an application framework which is open source and loosely coupled and it reduce the complexity of Enterprise application development.
* Spring supports various frameworks like Hibernate, EJB and struts.
* Spring tells us that use the Association relationship instead of Inheritance relation, It means that you no need to implement or extend the interface or class to get those functionality, just use the reference for that by creating HAS-A relationship (mainly used in POJO classes). So that our class will become light weight.
* In spring there Inversion of control and dependency Injections and bean creation which made spring framework strong.
* Spring framework will have different features like light weighted, MVC, Aspect oriented programming, IOC, dependency Injection.
* Spring core container is a core of spring framework where it contains Spring Core, Spring Bean, Spring Context.
* Spring core is a part of Dependency Injection, where **dependency Injection** will avoid the problems like where one Objects injects the data to another objects which is dependent.
* This is avoided by loosely coupling between the classes can be possible by using interface where the common functionality and implementation will be different but functionality will be remains same. So what ever we need we can use directly not depending upon one. This configurations will be done by developers.
* This **Injection of dependence** is done by either Setters or Constructor’s.
* This is mainly done because here java classes will be independent, and this configurations and dependencies will be independent.
* Spring provides integration between data access layer and service layer by using JDBC,ORM tools,JSM(Java messaging services).
* Spring configuration file is a where classes are configured and introduced to each other,if this is not maintained correctly then it will become tightly coupled and it wil be difficult for big projects.
* In spring framework , **Spring Inversion of control(IOC)/spring container** is the main part, it creates the objects and configure them and manages it, it uses dependency injection for this,

Container will get the instructions for which object to instantiate and configure based upon on java annotations or XML based.

* Difference ways of Dependency injections is like setters, constructors, Interface Injections. In spring framework only we use Setters and constructors based .
* Setters injections is like we create a setter code and it can be override the constructor property, it wont create any instantaces if we do any modifications done.
* Contructors doesn’t override setters property and it will create new instances if any modificatiosn is done.

**Benefits of IOC**:

* It will minimize the amount of code in your application.
* It will make your application easy to test because it doesn’t require any singletons
* It promotes loose coupling

Beans:

* Beans are nothing but Objects and this is created by the IOC and it plays a crucial role.
* Beans can be created by the XML configuration and by Annotations Based configurations  
  @Bean, @Configuration : both are used annotation based.
* Bean is like singleton object once we create and we use it where ever it requires, Beanfactory is like combined of beans.
* @Component: This makes the Java class as Bean, so the component-scanning mechanism of spring can pick it up and pull it into the application context
* Application context is an interface which provides the configurations and it is advanced of the IOC container.
* @Component used to auto-detect and auto-configure beans using classpath scanning. There’s an implicit one-to-one mapping between the annotated class and the bean (i.e. one bean per class).
* @Bean is used to explicitly declare a single bean, rather than letting Spring do it automatically for us.
* Another big difference is that @Component is a **class level annotation** where as @Bean is a **method level annotation** and ,by default, name of the method serves as the bean name.
* @Controller:
* This says that class is a Controller class and beans marked to it will be directly injected through the dependency injections.
* @Service: @component
* It is used for writing all the business logics and it it part of components annotation
* the basic difference is that @component is generic type and object will create for any use but on the other hand @service is a type of component and its object is also created by IOC container but its purpose is to define the bussiness logic .
* @repository:
* It is used in Database Layer , which says that it is DAO and integrates with Service layer.
* **It also makes the unchecked exceptions (thrown from DAO methods) eligible for translation** into Spring DataAccessException.
* @Autowired annotation:
* It is like without creating the objects we will inject the beans, it used at setters and constructors and Methods above.

@Qualifier:

* When you create more than one bean of the same type and want to wire only one of them with a property  you can use the **@Qualifier** annotation along with **@Autowired** to remove the ambiguity by specifying which exact bean should be wired.
* noSuchBeanDefinitionException will through if there is same beans.
* @RequestMapping annotation is used for mapping a particular HTTP request method to a specific class/ method in controller that will be handling the respective request. This annotation can be applied at both levels:
* The **DispatcherServlet** is the core of Spring Web MVC framework. It handles all the HTTP requests and responses. The DispatcherServlet receives the entry of handler mapping from the configuration file and forwards the request to the controller. The controller then returns an object of Model And View. The DispatcherServlet checks the entry of view resolver in the configuration file and calls the specified view component.
* The DispatcherServlet mapping will be created in WEB.XML

**Flow**:

HTTP Request 🡺 Controller 🡺 Service 🡺 Repository 🡺 Database

Database 🡺 Repository 🡺 Service 🡺 Controller 🡺 HTTP Response

A 🡺 B 🡺 C 🡺 D 🡺 E

Dependency Injection 🡺 Setter Injection, Constructor

Class A{

B b;

A(B b){

This.b = b;

}

}

IOc Container 🡺

@Autowoired

A a = new A();

a.methodName();

@Autowired

A a;

a.methodName():

@Bean

Public Hello hello(){

Return new Hello();