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
* In spring there Inversion of control and dependency Injections and bean creation which made spring framework strong.

Spring framework will has different features like light weighted, MVC, IOC, dependency Injection.

* 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.
* Spring provides integration between data access layer and service layer by using JDBC, ORM tools.
* 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.

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.
* @Bean is a **method level annotation** and ,by default, name of the method serves as the bean name.

@Bean(“hello”)

**public** CalculatorImpl multiply() {

**return** **new** CalculatorImpl();

}

@Bean(“Hi”)

**public** CalculatorImpl multiply() {

**return** **new** CalculatorImpl();

}

@RestController

@Autowired

@Qualifier(“Hi”)

Private CalculatorImpl CalculatorImpl;

@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.
* In spring Application context its supports Beans.
* @Component used to auto-detect and auto-configure beans using classpath scanning.
* @Component is a class level annotation

@Controller:

* This says that class is a Controller class and beans marked to it will be directly injected through the dependency injections. All the urls mapping will be given in the controller.

@Service = @component

* It is used for writing all the business logics and it is 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.

@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.

http:localhost:8080/api/hello/hi/data

context path:/api

@RequestMapping(value= “/hello”) 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 Dispatcher Servlet checks the entry of view resolver in the configuration file and calls the specified view component.

POST – Database saving

PUT – updation

DELETE -- delete

GET -- fetch

@RequestMapping(value=”/hello”)

@GetMapping(“/hi)

logic

@PostMapping(“/hi/data”)

{

“name”:”pranith”,

}

@RequestBody: Json to JAVA object conversion.

@ResponseBody: Java object to Json and directly send to HttpResponse.

Maven:

* It is a Build tool, used to build the application and generate the JAR/WAR file.

**Commands:**

* Mvn clean
* Mvn install
* Mvn build

Tomcat server: It is a deployment server. Used to deploy the application.

By default port no:8080

**Commands:**

Startup.bat

Shutdown.bat

# [**What will happen if we interchange @service and @repository annotation in the spring MVC**](https://stackoverflow.com/questions/35449808/what-will-happen-if-we-interchange-service-and-repository-annotation-in-the-sp)

Ans:

That would probably screw up your transactions. A service needs to pass transactions that need propagation to the DAO. Both are components so creation of a bean won’t be a problem

What are stereotype Anotations:

**@Component**

**@Service**

**@Repository**

**@Controller**

**@RestController**

Annotations:

@EnableAutoConfiguration

@SpringBootConfiguration

@ComponentScan

@Springbootapplication

@Controller 🡺

@ResponseBody 🡺 java object response converts to json object 🡺 httpresponse

@RequestBody 🡺 HttPRequest Data json object to java object converting

@RestController == @Controller + @ResponseBody

@RequestMapping (“/student”)🡺 urls

@Service

@Repository

@Transactional 🡺 To handle the transactional mangement

@Bean

@Component

@Configuration

@Autowired

@Qualifier

@PostMapping🡺 save

@GetMapping 🡺 get

@PutMapping 🡺 update

@DeleteMapping 🡺 delete

@PathVariable 🡺 reading data from url

@RequestParam 🡺 reading data from url with key value pair

**Actuators:**

* Before the spring framework, if we had to introduce this type of monitoring functionality in our applications then we had to manually develop all those components and that too was very specific to our need. But with spring boot we have Actuator module which makes it very easy.
* all the management and monitoring related information is easily available ith Actuators.
* Spring boot’s module Actuator allows you to monitor and manage application usages in production environment, without coding and configuration for any of them. These monitoring and management information is exposed via [REST](https://restfulapi.net/) like endpoint URLs.
* By default, only /health and /info are exposed via Web APIs. Rest are exposed via JMX. Use management.endpoints.web.exposure.include=\* to expose all endpoints through the Web APIs.

|  |
| --- |
| **application.properties** |
| management.endpoints.web.exposure.include=\*    # To expose only selected endpoints  #management.endpoints.jmx.exposure.include=health,info,env,beans |

Actuattos Urls:

|  |  |
| --- | --- |
| /mappings | Displays a collated list of all @RequestMapping paths. |
| /env | Returns list of properties in current environment |
| /health | Returns application health information. |
| /caches | It exposes available caches. |
| /loggers | The configuration of loggers in the application.. |
| /scheduledtasks | Displays the scheduled tasks in the application. |

|  |  |
| --- | --- |
| /metrics | It shows several useful metrics information like JVM memory used, system CPU usage, open files, and much more. |

* By default, [spring security](https://howtodoinjava.com/spring-security-tutorial/) is enabled for all actuator endpoints if it available in the classpath.
* If you wish to configure custom security for HTTP endpoints, for example, only allow users with a certain role to access then configure WebSecurityConfigurerAdapter
* By default, all endpoints (except /shutdown) are enabled. To disable all endpoints, by default, use property:

**management.endpoints.enabled-by-default=false**

* To use the only required endpoints which the application need to expose using the pattern management.endpoint.<id>.enabled.

Ex: management.endpoint.health.enabled=true

management.endpoint.loggers.enabled=true

* To disable Actuators endpoint security we use **management.security.enabled=false**

entry to the **application.properties** file to disable actuator security.

* **http://localhost:8080/actuator/beans:** This will give all the spring beans loaded in the context.
* **http://localhost:8080/actuator/env:** This will give all the environmental configuration about the server.
* **http://localhost:8080/actuator/threaddump:** This will give the current server thread dump