Spring Framework

1. It is framework of frameworks. It is a combination of multiple frameworks like structs, EJB, hibernate, etc.
2. It is a set of programs which to develop web application, standalone application and enterprise level application.
3. It is mainly used to avoid Boiler Plate Coding in an application.
4. Boiler Plate Coding means repetitions of the code.
5. For example: While working with JDBC or servlets program, programmers are performing many operations repeatedly like Establish connection, creation of platform and etc.
6. Also, it is used to develop loosely coupled application.
    a. Dynamically changing method implementation for respective method signature is referred as loose coupling.
    b. Providing fixed method implementation for method signature is referred as tight coupling.
7. It implements Run-time polymorphism and HAS-A-Relationship.
8. It provides different modules to develop an application in easy and faster way.
    a. Spring core
    b. Spring web
    c. Spring date access
    d. Spring AOP (Aspect Oriented Programming)
    e. Spring Security
    f. Test

Spring Core

1. It is most important module of Spring framework, to develop an application.
2. Without having knowledge of Spring Core, programmer can’t develop any application by using Spring.
3. But Only Spring Core is not enough to develop entire application, its mandatory to use other modules with it.
4. It consists two important concepts are as follows.
    a. IoC (Inversion of Control)
    b. Dependency Injection

Note: Entire Spring Framework depends on Inversion of Control.

Inversion of Control

1. It is a software engineering principle which is implements in Spring Framework.
2. It is used to externalize object creation i.e.; external component will create object for programmer.
3. With the help of IoC programmer can focus on functionalities and business logic of an application.
4. IoC is used to achieve loose coupling in an application which are developed by using spring framework.
5. IoC reverse flow of Control of a program which means till now, entire object related work is performed by the program but because of IoC control got reversed, now everything will get performed by Spring framework itself.
6. To perform all operation related to Object, Developer introduced “IoC container” into Spring, it also known as “Spring container”.

IoC Container

1. It is an intelligent program present in Spring Framework.
2. It will create & manage the objects by reading configuration metadata present in XML file and annotations.
3. Internally, it can store multiple objects with the help of data structure.
4. To represent IoC container, we have two major interfaces.
    a. BeanFactory Interface present in org.springframework.core
    b. ApplicationContext Interface present in org.springframework.context
5. ApplicationContext extends properties of BeanFactory interface.
6. BeanFactory Supports only XML file Configuration metadata whereas ApplicationContext Supports XML file and annotations configuration metadata.
7. ApplicationContext has two implementation classes:
    a. ClassPathXMLApplicationContext used for XML file
    b. AnnotationConfigApplicationContext used for annotation
8. To create an object IoC container requires two information
    a. Description of an object.
    b. Configuration metadata
9. We can provide configuration metadata in two ways.
    a. By using XML file.
    b. By using annotations.
10. To provide Description of an object, create a class.

11. Configuration metadata helps to provide information of an object.

Steps to Create Project for Spring Core:

1. Create a Maven project.
2. Open pom.xml file and then provide “Spring context” dependency.
3. Update the project.
4. Create a package then create class to provide Description of Object.
5. Create XML file or Configuration Class, to provide Configuration metadata.
6. Create a Class which consists main method for execution of a code.
7. In main method, Launch IoC container and Access object from IoC container.

How to Launch IoC container?

1. For XML file configuration.

   ApplicationContext context =

   new ClassPathXmlApplicationContext(“pathOfXmlFile”);

2. For Annotation
   ApplicationContext context =
   new AnnotationConfigApplicationContext(ConfigurationClass);

How it works?

1. IoC reads XML file or Annotation Configuration.
2. As per programmer configuration, it creates an object by calling constructor of respective class.
3. Then, Its upcast object to Object class.

Note: We have two approaches for creation of an object.

1. Explicit Approach where Programmer writes code for creation of object, whereas IoC container manages the objects.
2. Implicit Approach where Programmers are not responsible to write code for creation of object, IoC container itself will create and manage the objects.

How to get object from IoC container?

1. Programmers are using getBean() method, to get object from IoC container, presents in BeanFactory Interface.
2. It is non-static overloaded method.
3. getBean(“id of object”) – needs to perform down casting.
4. getBean(ClassName.class) – avoids down casting but works only for one object.

5. getBean(“id”, ClassName.class) – avoids down casting and works for multiple object.

XML file Configuration Metadata

What is XML?

1. XML stands for Extensible Markup Language.
2. It represents by using tags.
3. It used to perform java configuration like servlet, hibernate, spring, etc.

Sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<! -- Definitions of Spring Framework -->
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <! -- Create an object -->
   <bean id="demo1" class="com.jsp.springcore.Demo">
   </bean>

<!-- throws an exception BeanDefinitionParsingException,
                if program provides same id for multiple beans -->
</beans>
```

1. In xml file <bean> tag used to create object which consists following attributes.
    a. Id = to provide name or ref variable for an object.
    b. Class = to specify fully qualified class name.
2. When programmer creates object by using <bean> then, internally default or constructor without argument will get called.

Scope of an Object

1. It is used to control object creation.
2. We have specifically two types of Scope.
    a. Singleton
    b. Prototype
3. By default, scope of an object is singleton, which means only one object will create for respective bean tag.
4. If scope is singleton then object gets create at the launch of IoC container, before calling getBean().
5. At every call of getBean() IoC container returns same object to a programmer.

6. Programmer can modify scope, by using scope attribute in <bean> tag.
7. If scope is prototype, then objects get create after calling getBean() method.
8. At every call of getBean(), IoC container return new object to a programmer.
9. For Example:

```
<bean id="s2" class="com.jsp.scope.Student" scope="prototype">
</bean>
```

Initialization of Primitive Data type & String type of Variable.

1. We have two ways
    a. Constructor injection
    b. Setter injection
2. Constructor injection:
    a. It is achieved by <constructor-arg> tag within <bean> tag.
    b. This tag represents argument of given constructor.
    c. If constructor have 3 arguments, then specify 2 <constructor-arg>.
    d. Data gets inject in variables by calling constructor with arguments.
    e. Also, object gets created by calling constructor with arguments.
    f. This tag consists following attributes:
        i. Value – to provide data.
        ii. Name – to specify argument name of given constructor
        iii. Index – to specify index of an argument.
    g. Programmer can use only “value”, but they have to maintain order of an argument.
    h. If programmer don’t want to maintain order, then, they can user “value” with “name” or “index” attribute.
3. For example:

```
//Constructor
    public Employee(int id, String ename) {
            super();
            this.id = id;
            this.ename = ename;
            System.out.println("Constructor of Employee with 2 arg");
     }
```

    a. Value attribute

```
<bean id="emp1" class="org.jsp.constructor_injection.Employee">
        <constructor-arg value="101"></constructor-arg>
        <constructor-arg value="QWERTY"></constructor-arg>
</bean>
```

    b. Value with name attribute

```
<bean id = "emp2" class="org.jsp.constructor_injection.Employee">
        <constructor-arg name="ename" value="Qwerty">
        </constructor-arg>
```

```
<constructor-arg name="id" value="102"></constructor-arg>
</bean>
```

c. Value with index attribute

```
<bean id="emp3" class="org.jsp.constructor_injection.Employee">
        <constructor-arg index="1" value="Demo"></constructor-arg>
        <constructor-arg index="0" value="103"></constructor-arg>
</bean>
```

4. Setter injection.
    a. It is achieved by <property> tag within <bean> tag.
    b. This tag represents setter method of respective variable.
    c. Data gets inject in variables by calling setter with arguments.
    d. Object gets created by calling constructor without arguments.
    e. This tag consists following attributes:
        i. Value – to provide data.
        ii. Name – to specify argument name.
    f. To call setter method, make use of <property> tag for that respective variable.
5. For example:

```
<bean id="order1" class="org.jsp.setter_injection.Order">
        <property name="orderId" value="1001"></property>
</bean>
```

Note: Constructor injection is used initialize all the data members of a given class. Setter injection prefers to initialize specific data members of a given class.

Dependency Injection:

1. It is used to inject object into Non-primitive Data type of variables.
2. When one object depends on another object, is referred as Dependency.
3. Injecting Dependency into a parent class object, is referred as Dependency injection.
4. It is one of the Design patterns and also a way to achieve IoC.
5. We can perform Dependency Injection in two ways:
    a. Constructor injection
    b. Setter injection
6. Injecting the dependency by using constructor with arguments, referred as constructor injection.
7. To perform it, make use of <constructor-arg> tag with “ref” attribute, Whereas “ref stands for reference”.
8. For example:
   Description of Bean

```
public class Author {
```

```
        private int id;
        private String name;
        public Author(int id, String name) {
                super();
                this.id = id;
                this.name = name;
                System.out.println("Author class constructor");
        }
}
public class Book {

        private int bookId;
        private String bookName;

        // Dependency
        private Author author;

        public Book(int bookId, String bookName, Author author) {
                super();
                this.bookId = bookId;
                this.bookName = bookName;
                this.author = author;
                System.out.println("Book class constructor");
        }

}

//XML file
<bean id="a1" class="com.jsp.dependency_injection.Author">
                <constructor-arg name="id" value="1"></constructor-arg>
                <constructor-arg name="name" value="Rod
Johnson"></constructor-arg>
</bean>

<bean id="book" class="com.jsp.dependency_injection.Book">
                <constructor-arg value="101"></constructor-arg>
                <constructor-arg value="Spring Framework"></constructor-
arg>
                <!-- to inject Dependency -->
                <constructor-arg ref="a1"></constructor-arg>
</bean>
```

9. Injecting the dependency by using setter method with arguments, referred as setter injection.
10. To perform it, make use of <property> tag with “ref” attribute, Whereas “ref stands for reference”.
11. For example:

    Description of Bean

```
public class Author {
        private int id;
```

```
        private String name;
        public Author(int id, String name) {
            super();
            this.id = id;
            this.name = name;
            System.out.println("Author class constructor");
        }
    }

public class Book {

    private int bookId;
    private String bookName;

    // Dependency
    private Author author;

    public Book() {
        System.out.println("Book Class constructor without argument");
    }

    public void displayBookInfo() {
        System.out.println("Id = " + bookId + ", bookName = " + bookName);
        author.displayAuthorInfo();
    }

    public int getBookId() {
        return bookId;
    }

    public void setBookId(int bookId) {
        this.bookId = bookId;
        System.out.println("setter method of bookId");
    }

    public String getBookName() {
        return bookName;
    }


    public void setBookName(String bookName) {
        this.bookName = bookName;
        System.out.println("setter method of bookName");
    }

    public Author getAuthor() {
        return author;
    }

    public void setAuthor(Author author) {
```

```
                this.author = author;
                System.out.println("setter method of author");
            }
        }
```

//XML file

```
<bean id="author1" class="com.jsp.dependency_injection.Author">
                <constructor-arg value="2"></constructor-arg>
                <constructor-arg value="QWERTY"></constructor-arg>
</bean>

<bean id="book" class="com.jsp.dependency_injection.Book">
                <property name="bookId" value="1"></property>
                <property name="bookName" value="Java"></property>
                <!-- dependency injection -->
                <property name="author" ref="author2"></property>
</bean>
```

Difference between Constructor injection and Setter injection.

Constructor injection	Setter injection
1. Data and dependency injected by using constructor with arguments.	1. Data and dependency injected by using setter methods.
2. Object created by using constructor with arguments.	3. Object created by using constructor without arguments.
4. <constructor-arg> tag is used in configuration file.	5. <property> tag used in configuration file.
6. <constructor-arg> tag with ref is used for dependency injection.	7. <property> tag with ref is used for dependency injection.
8. It prefers to initialize all the variables of a class.	9. It prefers to initialize specific variables of a class.

Autowiring

1. Implicitly injecting dependency into a parent object referred as “auto wiring”.
2. It is one way to achieve dependency injection.
3. To achieve auto wiring programmer will make use of “autowire attribute” with in <bean> tag.
4. It takes following values.
    a. No

    b. byType
    c. byName
    d. constructor
5. By default, for autowire attribute is “No”, which means no need to perform autowiring.
6. To achieve auto wiring “by Type”, provide “byType” for autowire attribute.
7. For this, IoC container will search dependency by using its type, and injects it by using setter injection.
8. byType works only for one <bean> tag of dependency. For multiple beans tag of dependency, it throws exception.
9. To achieve auto wiring “by Name”, assign “byName” for autowire attribute.
10. For this, IoC container will search dependency by using reference variable name specified in respective parent class as a global variable, and injects it by using setter injection.
11. Programmer have to provide id for dependency object same as reference variable specified in class.
12. It supports multiple <bean> tags of dependency.
13. For example:
14. //XML file

```
<bean id="author1" class="com.jsp.dependency_injection.Author">
                        <constructor-arg value="2"></constructor-arg>
                        <constructor-arg value="QWERTY"></constructor-arg>
</bean>

<bean id="book" class="com.jsp.dependency_injection.Book" autowire=”byType”>
                        <property name="bookId" value="1"></property>
                        <property name="bookName" value="Java"></property>
</bean>
```

15. If programmer assigned constructor value, IoC container firstly search dependency by using its type, if it unable to get then, it will search dependency by using name based on argument specified in constructor.

Note: To create object on dependency before parent object, make use of “depends-on” attribute with bean tag of parent object.

Annotation Configuration Metadata

What is annotation?

1. Annotation represents metadata of class, interfaces, methods, variables.
2. It provides information about all members to compiler and JVM.
3. Always it should be specified above class, interface, method and variable in Pascal casing format.

//Configuration class

```
@Configuration
public class AppConfig {
	public AppConfig() {
		System.out.println("AppConfig constructor");
	}

	@Bean(name = "demo")
	@Scope("prototype")
	public Demo getDemoObject() {
		System.out.println("getDemoObject method from Appconfig");
		Demo d1 = new Demo();
		return d1;
	}
}
```

1. @Configuration – It is used to represent Configuration class.
2. Configuration class object created by IoC container implicitly.
3. To create object of other classes in Configuration class, create non-static method with return type as respective class.
4. Then create object of class, and return to the same method.
5. @Bean – It is used above method, which creates object of class.
6. Any method denoted with @Bean, called by IoC container implicitly.
7. By default, object Id will be same as method name.
8. Programmer can modify it, by using name attribute.
9. If method is not denoted with @Bean, IoC container fails to call that method.
10. The object which is created by method calling, programmer can use it, by using “getBean()”.
11. @Scope – It is used to modify scope of an object to “prototype”.

Initialization of Primitive Data type, String type of Variable and Dependency Injection.

1. By using Constructor Injection

```
@Configuration
public class ApplicationConfig {

	//creates object of dependency
	@Bean(name = "addr")
	public Address getAddress() {
		return new Address(1502, "Ramtekdi");
	}

	@Bean
	public Person getPersonObject() {
		//dependency injection
```

```
            return new Person("QWERTY", getAddress());
    }
}
```

2. By using Setter Injection

```
@Configuration
public class AppConfig {

    @Bean
    public Address getAddressObject() {
            Address addr = new Address();
            addr.setArea("Hadapsar");
            addr.setPlotNo(1);
            return addr;
    }

    @Bean
    public Person getPerson() {
            Person person = new Person();
            //dependency injection
            person.setAddress(getAddressObject());
            person.setName("Sample");
            return person;
    }
}
```

Implicit Approach - Object create and manage by IoC container.

```
@Component("t1")
@Scope("prototype")
public class Test {

        @Value("10")
        private int a;

        public Test() {
                System.out.println("Test constructor");
        }
}
```

Configuration Class

```
@Configuration
@ComponentScan(basePackages = "com.jsp.autowiring")
public class Config {
}
```

1. @Component – used to denote class, which object should get create by IoC container.
2. By default, object id will be same as class name.
3. Programmer can modify, object id by using @Component.
4. @Scope – used above the class, to modify scope of object to prototype.
5. @ComponentScan is used to scan the package, which consist classes denoted with @Component annotation.
6. For this, programmer have specify “basePackages” attribute, to provide package name.
7. @Value – used to initialize primitive and String data type variable.
8. This annotation can be used above variable name, with constructor arguments and above setter methods.
9.
```
public Book(@Value("2") int id,
                    @Value("SQL") String name,
                    @Value("456.90") double price) {
            super();
            this.id = id;
            this.name = name;
            this.price = price;
}
```
10.
```
@Value("3")
public void setId(int id) {
            this.id = id;
            System.out.println("Setter of id");
}
```
11. **To inject dependency implicitly, programmers are using @Autowire, above variable, with constructor argument and above setter methods.**
12. Programmers, can initialize and inject dependencies by using annotations in three ways
    a. By using variables.
    b. By using constructor with arguments
    c. By using setters
13. **Always, firstly IoC search dependency by its type and then it will search by its name.**
14. **For example:**

    1.
```
@Autowired
private Product product;
```
    2.
```
public Order(@Value("1") int id,
                    @Value("20") int noOfProducts,
                    @Autowired Product product) {
            super();
```

```
                this.id = id;
                this.noOfProducts = noOfProducts;
                this.product = product;
                System.out.println("Order class consturctor");
        }
3.      @Autowired
        public void setId(int id) {
                this.id = id;
        }
```

15. @Qualifier – used with @Autowired, if multiple objects are there of same type, to avoid confusion, which object should get inject in dependency.
16. @Primary – used with @Component above class, to avoid confusion between multiple objects of same type, it provides priority for respective class object.

Note: To achieve implicit approach by using XML file, instead of writing <bean> just specify <context:component-scan base-package = “”>, to provide package name, which consists “classes denoted with @Component”.

For example:

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context=http://www.springframework.org/schema/context
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd>

        <context:component-scan base-
package="com.jsp.java_based_approach"></context:component-scan>
</beans>
```