1.Why should we use Spring Boot Framework?

**Rapid development**: Spring Boot allows developers to quickly build applications with minimal configuration

**Easy to learn**: Spring Boot has a low learning curve and is easy to use.

**Microservices**: Spring Boot is well-suited for building microservices-based architectures due to its lightweight nature and ease of integration with other microservices frameworks.

**2.Spring Container:**

In Spring Framework, the container is responsible for managing the application's components and providing them with the necessary dependencies. The container is sometimes also referred to as the Inversion of Control (IoC) container or the Dependency Injection (DI) container.

**key features of Spring Container**

* Dependency Injection- The container injects the necessary dependencies into the beans at runtime, based on the dependencies declared in the application context.
* Lifecycle management- The container manages the lifecycle of the beans, including creating, initializing, and destroying them.
* Configuration management- The container manages the configuration of the application, including property values and other settings.

**Two types of containers in Spring:**

1.Bean Factory and

2.Application Context.

**Bean Factory** is a lightweight container that provides the basic functionality of managing the beans.

**Application Context** is a more advanced container that provides additional features such as support for internationalization, event handling, and more.

How to use spring container?

* To use the Spring container, you need to define the beans and their dependencies in the application context file. The application context file is an XML file that defines the beans and their dependencies. You can also use Java annotations or Java-based configuration to define the beans and their dependencies.
* When the application starts up, the container reads the application context file and creates the beans defined in it. It then injects the dependencies into the beans based on the configuration specified in the application context file. Finally, it manages the lifecycle of the beans and provides them to the application as needed.

**IoC (Inversion of control)**

* Inversion of control (IOC):is a design pattern in software development that promotes loose coupling and modular design.
* It is also known as the Dependency Injection (DI) pattern, which refers to the process of injecting dependencies into an object, rather than creating them internally.

**Process of IoC (Inversion of Control)**

The process of IoC works internally by using a container to manage dependencies. The container maintains a registry of available services and the dependencies required by each service. When an object requests a service, the container provides the service and its dependencies.

**AoP(Aspect Oriented Programming)**

In Java, Spring AOP provides a way to modularize cross-cutting concerns in your application by using aspects. Aspects are defined separately from the core logic of the application and can be applied to multiple classes or methods.

To use Spring AOP in your Java application, you need to configure the AOP framework in your Spring application context. This can be done through XML configuration files or using annotations.

A **cross-cutting concern** is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

Layered architecture of a spring boot?

* Spring Boot is a framework that is based on the principles of modularization and separation of concerns, which are typically implemented using a layered architecture.

The layered architecture of a Spring Boot application typically includes the following layers:

Presentation layer-->Service layer-->Data access layer-->Persistence Layer-->Infrastructure layer

**1.Presentation layer:**

* This layer is responsible for handling the presentation logic of the application, such as handling user input.
* Presentation layer implemented using spring mvc.

**2.Service layer:**

* This layer contains the business logic of the application, and is responsible for processing data.
* Implemented using Spring's service-oriented architecture (SOA) features.

**3.Data Access layer:**

* This layer is responsible for interacting with the database
* implemented using Spring Data, which provides a API for accessing various types of data sources.

**4.Persistence Layer:**

* This layer is responsible for managing the persistence of data in the database
* Implemented using an Object Relational Mapping (ORM) framework such as Hibernate.

**5.Infrastructure layer:**

* This layer contains infrastructure components that provide common functionality such as logging, security, and caching.
* implemented using various Spring libraries

The layered architecture of a Spring Boot application provides a clear separation of concerns, making it easier to manage and maintain complex applications.

**Spring Cloud -**

Spring Cloud provides a set of annotations that can be used to enable service discovery and registration in a Spring Boot application.

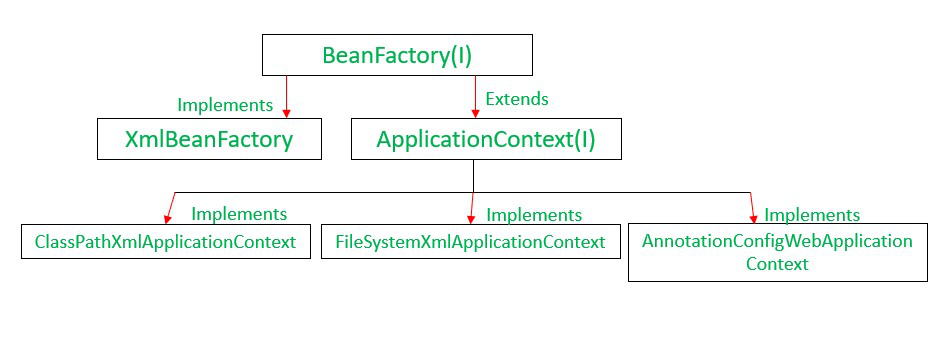
**Spring MVC (Model-View-Controller):**

A Spring MVC is a Java framework which is used to build web applications.

* **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format.

**Different types of spring IoC container**

* **Bean Factory--** The most common implementation class used for this BeanFactory is **XmlBeanFactory** available in **org.springframework.beans.factory.xml package.**
* **Application Context--**There are so many implementation classes that can be used such as **ClassPathXmlApplicationContext**, **FileSystemXmlApplicationContext**, **AnnotationConfigWebApplicationContext** etc.



| **BeanFactory** | **ApplicationContext** |
| --- | --- |
| It is a fundamental container that provides the basic functionality for managing beans.  It is the actual container that instantiates, configures, and manages a number of beans. | It is an advanced container that extends the BeanFactory that provides all basic functionality and adds some advanced features.  The ApplicationContext interface is the advanced container that enhances BeanFactory functionality in a more framework-oriented style.  While the BeanFactory provides basic functionality for managing and manipulating beans, often in a programmatic way, the ApplicationContext provides extra functionality like MessageSource, Access to resources, Event propagation to beans, Loading of multiple (hierarchical) contexts etc |

**Creation of SpringBoot Application**

### Step 1. Launch Spring Initializr using [https://start.spring.io](https://start.spring.io/) link

### Step 2. Specify Project Details

Look at the above diagram, we have specified following details:

* Generate: Maven Project
* Language: Java
* Java Version: 1.8 (Default)
* Spring Boot: 2.1.4
* Group: com.customercare
* Artifact: customercare
* Name:Customercare
* Description: Demo project for Spring Boot
* Package Name : com.customercare
* Packaging: jar (This is the default value)
* Dependencies: Web, JPA, postgres,spring data

Once, all the details are entered, click on Generate Project button will generate a spring boot project and downloads it. Next, Unzip the downloaded zip file and import it into Eclipse IDE.

### 3. Import project in Eclipse

In Eclipse, Click File -> Import -> Existing Maven Project

type in the path of the folder where you extracted the zip file to in the next screen.

Once you click Finish, Maven would take some time to download all the dependencies and initialize the project.