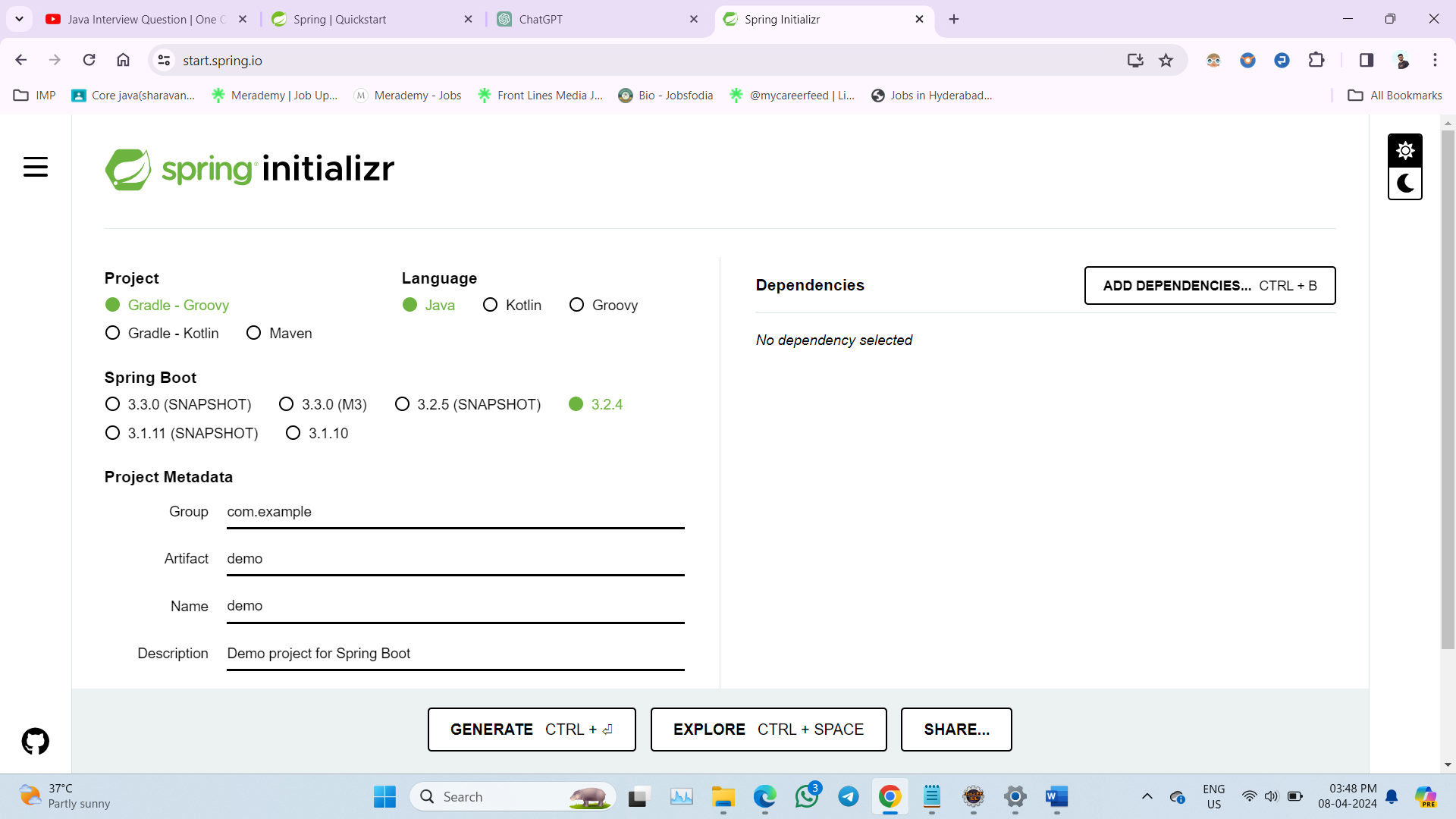
**SPRING Framework**



**Go to the spring init 🡪 initializer to create new spring project select the required configurations then download the zip file**

**Extract the zip file and import the project folder to the IDE (eclipse)**

**Then work on the project🡪 this process helps to connect with database.**

1. XML Configuration
2. Java Class Configuration (Annotation Based Configuration)
   1. Annotation based @Autowired

We have two core Interfaces to manage the IOC container in spring

1. Bean Factory
2. Application Context

**BeanFactory:**

* **Definition**: The root interface for accessing the Spring container.
* **Purpose**: Provides the basic functionality for managing beans, including instantiation, wiring, and managing their lifecycle.
* **Usage**: Suitable for simple applications or lower-level Spring infrastructure.
* **Example**:

BeanFactory factory = new XmlBeanFactory(new ClassPathResource("beans.xml"));

MyBean myBean = (MyBean) factory.getBean("myBean");

**ApplicationContext**

* **Definition**: A sub-interface of BeanFactory that provides additional functionalities.
* **Purpose**: Enhances BeanFactory with features such as:
  + **Event propagation**: Handling events within the application.
  + **Internationalization (i18n)**: Supporting different languages and locales.
  + **Application-layer specific context**: Like WebApplicationContext for web applications.
* **Usage**: Suitable for most enterprise applications due to its rich feature set.
* **Example**:

ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");

MyBean myBean = (MyBean) context.getBean("myBean");

**Key Differences**

* **Scope**:
  + BeanFactory is more basic and fundamental.
  + ApplicationContext is more feature-rich and higher-level.
* **Features**:
  + BeanFactory only provides basic bean management.
  + ApplicationContext provides additional features such as event handling, internationalization, and application context-specific support.
* **Initialization**:
  + BeanFactory loads beans lazily (on demand).
  + ApplicationContext loads all singleton beans eagerly at startup.

**Dependency Injection (DI):**

1. Constructor Based
2. Setters and Getters Based
3. Annotation Based @Autowired for variables (field injection).

**XML Configuration:**

1. **Constructor Based:**

1st we need to create a file with extension .xml then we need to paste the beans definition from spring.io

<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*

[*https://www.springframework.org/schema/beans/spring-beans.xsd*](https://www.springframework.org/schema/beans/spring-beans.xsd)*”*>

</beans>

This above code has to paste in xml file for example we can name it as Config.xml

Now we need to add beans in it.

Before we add beans, we need to create an Interface which can we used to implement classes

**Ex: Vehicle: Interface**

**package** com.sathya.springex.Practice\_Demo;

**public** **interface** Vehicle {

**void** move();

}

Now we can create classes like Car and Bike

**Ex: Car: Class**

**package** com.sathya.springex.Practice\_Demo;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** Car **implements** Vehicle {

**private** **int** speed;

**private** String model;

**private** String color;

// All Argument Constructors using fields

// toString method() using fields

Or

Simply: use Lombok Annotations like

@Data

@AllArgsConstuctor

@NoArgsConstuctor

But we need to download configuration file which imports Lombok from spring as it is there is screenshot, and then we need to import that project in to ide – eclipse.

As we are implementing the interface it will takes the abstract method in class we can print or return the values to get those on console.

**XML Configuration File**

<beans …>

Now with in the beans tag we need to give the dependencies beans here like

<bean id = *“car”* class =*”com.sathya.springex.Practice\_Demo.Car”*>

<constructor-arg name = *“speed”* value = *“150”* ></constructor-arg>

<constructor-arg name = *“model”* value = *“Mustang GT”*></constructor-arg>

<constructor-arg name =*”color”* value = *“Monsteer Green”*></constructor-arg>

</beans>

Id: it’s a variable to represent the bean

Class: we need to mention the location of the class with packageName.ProjectName.Class

Constructor-arg Tag: its only for Constructor type Dependency Injection

name: variable from the given class

value: the value we need to assign to that variable

**App.Class**

This is the default class file which is automatically generated while we creating the project.

**Package** com.sathya.springex.Practice\_Demo;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** App

{

**public** **static** **void** main( String[] args )

{

ApplicationContext context = **new** ClassPathXmlApplicationContext(“Config.xml”);

Car c = context.getBean(“car”, Car.**class**);

c.move();

}

}

ApplicationContext context = **new** ClassPathXmlApplicationContext(“Config.xml”);

* **ApplicationContext:** This is an interface in the Spring Framework that represents the context in which beans can be accessed, loaded, and managed.
* context: This is the name of the variable being declared. It will hold the instance of the application context.
* **new:** This keyword is used to create a new instance of a class or interface.
* **ClassPathXmlApplicationContext:** This is a class provided by Spring Framework for creating an application context by loading the configuration file from the classpath.
* **(“Config.xml”):** This is the parameter passed to the constructor of ClassPathXmlApplicationContext. It specifies the name of the XML configuration file that contains the bean definitions for the application context.
* **“Config.xml”:** This is a string literal representing the name of the XML configuration file. It is enclosed in double quotes.

Car c = context.getBean(“car”, Car.**class**);

Car: Class name used to create the object or bean

c: it’s a reference variable to store the instance/object.

**Context.getBean(“car”, Car.class):** This calls the getBean method on the context object to retrieve a bean named “car” from the application context.

**“car”:** This is the name of the bean to retrieve from the context. Beans in the Spring application context are identified by their unique names. It is an **id** in the xml file bean tag.

Id: “car” in bean tag .xml file is same in getBean(“car”.Car.class)

**Car.class:** This specifies the class type of the bean to retrieve. It ensures that the retrieved bean is cast to the Car class.

(Class\_name of the bean(dependency) . class keyword)

c.move();

c: it is an reference variable stored the object of Car class

move(): its an abstract method from vehicle interface override in Car class and it is used to print something on console.

* **Setters and Getters Based**

Every this is same in class file and App file except xml file

In xml file we need to give the values in property tag instead of constructer-arg tag

<bean id=*”bike”* class =*”com.sathya.springex.Practice\_Demo.Bike”*>

<property name=*”speed”* value = *“100”*></property>

<property name=*”model”* value = *“Ninja H2R GT”*></property>

<property name=*”cost”* value = *“10000”*></property>

</bean>

Above bean should be within the beans tag.

Name: variable from the Car class

Value: value we need to assign to the variable.

**Annotation Based @Autowired:**

@Autowired annotation wont work for the primitive data variables includes String type in Setters and Getters method (), Constructor and also for individually assigning the variables

So, for Primitive data type variables we need to assign the values manually to inject the beans

Using, xml file or java config.class file.

In xml file same as above for @Autowired annotation for constructor we have to use constructor-arg and for @Autowired annotation for setters’ method we have to use property tags to mention values to the variables.

This Annotation Based (Field Based) Configuration mostly used for class type data type variables where we don’t have to give values to it, as it holds the instance(object) of the class we don’t have to assign any values to it.

**primitive type variables:**

@Component

**public** **class** Car **implements** Vehicle {

@Autowired

**private** **int** speed;

@Autowired

**private** String model;

@Autowired

**private** String color;

**class type variables:**

@Component

**public** **class** Person {

@Autowired

**private** HomeLoanService homeLoanService;

@Autowired

**private** RealEstateService realEstateService;

**Reference:**

* @Practice\_Demo
* @personal\_Loan

**Java Class Configuration**

1. Construction based:

@Configuration

**public** **class** JavaConfig {

@Bean

**public** Car car()

{

Car car = **new** Car(190,"Bugatti");

**return** car;

}

}

1. Setters Based:

@Configuration

**public** **class** JavaConfig {

@Bean

**public** Car car()

{

Car car = **new** Car();

car.setSpeed(120);

car.setBrand("ducatti");

**return** car;

}

}