Spring Core Tutorial in one video: Learn code with Durgesh

**Sprint Architecture**

**### Softwares:**

1. Eclipse/NetBeans/IntelliJ,

2. Tomcat Server,

3. MySQL for Database,

4. SQLyog / Workbench / phpMyAdmin for MySQL GUI

**### Steps for Spring Framework Setup:**

1. Create Maven Project,

2. Add Dependencies (Spring Core, Spring Context),

3. Create Beans (Java POJOs),

4. Create Configuration File (`config.xml`),

5. Setter Injection,

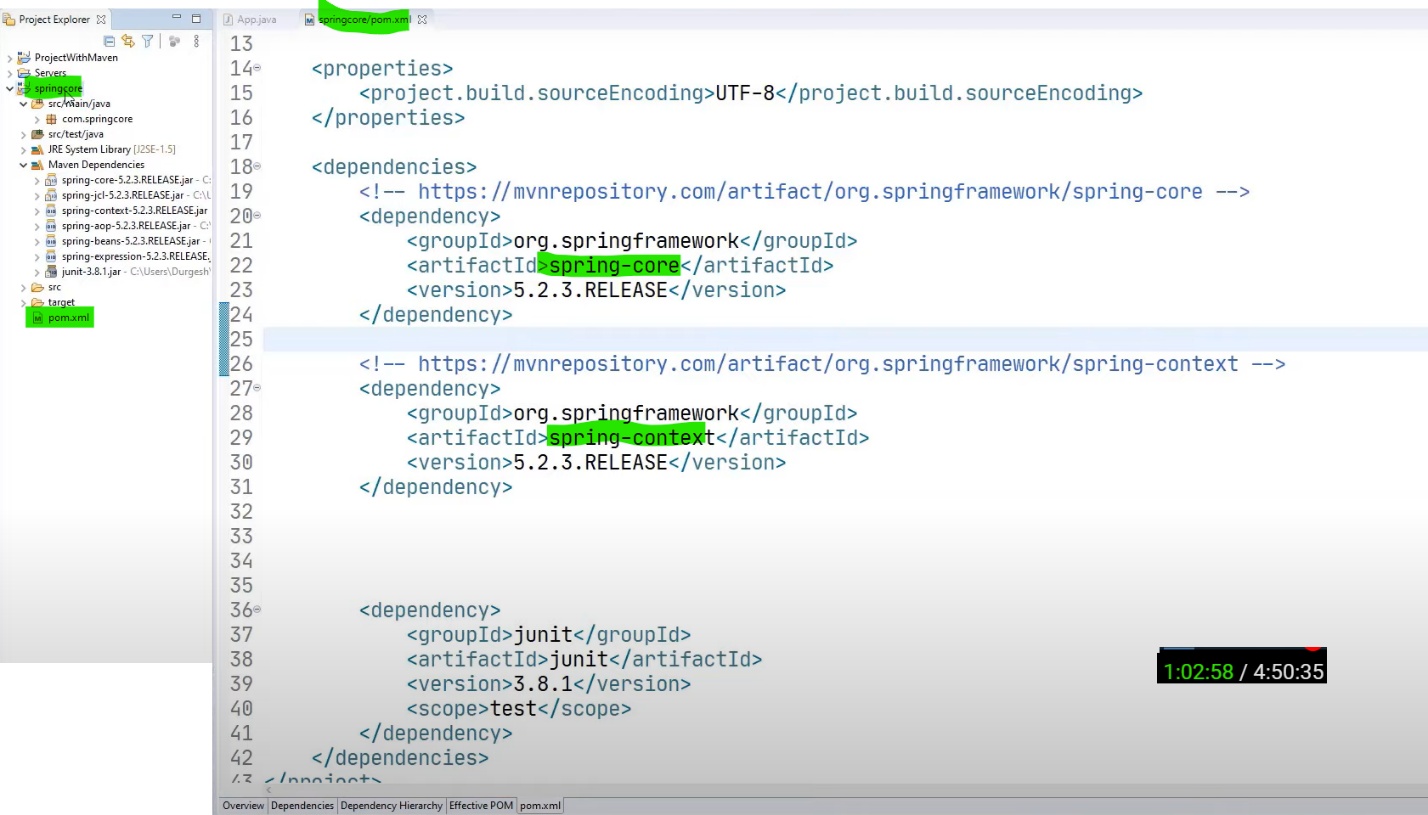
6. Main Class (Pull the object and use it).

**Step 1: Create App.java main class**

A screenshot of a computer

Description automatically generated

Step 2: Add dependencies Spring Core, Spring Context) in pom.xml file



Step 3: Create Bean (java *Plain Old Java Objects*.) **Student.java**

* Bean is a java class which holds data/ object.
* Main class will use the path of **config.xml** and will get this object and use this object.
* It *has getter and setter method, parameter constructor, default constructor.*

**package** com.springcore;

**public** **class** Student {

**private** **int** studentId;

**private** String studentName;

**private** String sudentAddress;

**public** **int** getStudentId() {

**return** studentId;

}

//getter and setter method

**public** **void** setStudentId(**int** studentId) {

**this**.studentId = studentId;

}

**public** String getStudentName() {

**return** studentName;

}

**public** **void** setStudentName(String studentName) {

**this**.studentName = studentName;

}

**public** String getSudentAddress() {

**return** sudentAddress;

}

**public** **void** setSudentAddress(String sudentAddress) {

**this**.sudentAddress = sudentAddress;

}

//parameter constructor

**public** Student(**int** studentId, String studentName, String sudentAddress) {

**super**();

**this**.studentId = studentId;

**this**.studentName = studentName;

**this**.sudentAddress = sudentAddress;

}

//default constructor

**public** Student() {

**super**();

// **TODO** Auto-generated constructor stub

}

@Override

**public** String toString() {

**return** "Student [studentId=" + studentId + ", studentName=" + studentName + ", sudentAddress=" + sudentAddress

+ "]";

}

}

Step 4: Create configuration file

* We will give this path to **spring container**, Spring container will create object, manage lifecycle, then we can get object.
* Creating configuration file, create any file name with .xml extension. Here file is config.xml
* Import spring bean schema from spring documentation.

A screenshot of a computer

Description automatically generated

**Step-by-Step Explanation for Spring Bean Configuration**

1. **Define the Bean Tag**:

In Spring XML configuration, you use the <bean> tag to define a Spring bean. It requires two main attributes:

* + - **class**: This specifies the fully qualified class name (including the package).
    - **name**: This is the name by which the bean can be accessed in the Spring container.

<bean class="com.springcore.Student" name="student1">

1. **Define Property for studentId**:
   * We use the <property> tag to inject values into the bean's properties.
   * Each <property> tag specifies the name of the property to be set and the value to inject.
   * The **name** attribute of the <property> tag must match the property name in the Student class (e.g., studentId).
   * The **value** tag specifies the value that will be injected into that property.

<property name="studentId">

<value>22344</value>

</property>

1. **Define Property for studentName**:
   * Similarly, for the studentName property, we use a <property> tag with the name attribute matching the property name in the Student class (studentName), and a <value> tag to specify the name value.

<property name="studentName">

<value>Durgesh Tiwari</value>

</property>

1. **Define Property for studentAddress**:
   * Similarly, you define the studentAddress property using the <property> tag. The value tag will contain the address as a string.

<property name="studentAddress">

<value>Delhi</value>

</property>

1. **Closing the Bean Tag**:
   * After defining all the properties, close the <bean> tag to complete the bean definition.

</bean>

**Complete XML Configuration:**

xml

<bean class="com.springcore.Student" name="**student1**">

<property name="studentId">

<value>22344</value>

</property>

<property name="studentName">

<value>Durgesh Tiwari</value>

</property>

<property name="studentAddress">

<value>Delhi</value>

</property>

</bean>

**For another object/ class**

<bean class="com.springcore.Student2" name="**student2**">

<property name="studentId">

<value>999</value>

*… in similar way we will keep on adding objects properties here*

</property>

</bean>

**Notes:**

* **Setter Injection**: In the Student class, the properties (studentId, studentName, studentAddress) should have corresponding setter methods (setStudentId(), setStudentName(), and setStudentAddress()).
* The Spring container will automatically call these setter methods to inject the values (22344, Durgesh Tiwari, and Delhi) when it creates the **student1** bean.
* You will fetch this bean by the name **student1** in the Spring context.
* We can write in same line also.like: <property name="studentAddress” value=”Delhi” />
* We are creating beans here for **student1** object only, we can keep on adding other beans for another objects.

Step 6: Main Class pull the object and use it here. **App.java**

1. **Configuration File Setup**:
   * Ensure that this configuration file (config.xml) is in the application’s classpath.
2. **Create ApplicationContext**:
   * Instantiate the **ApplicationContext** object, specifying the location of the **config.xml** file.

ApplicationContext context = new ClassPathXmlApplicationContext("**config.xml**");

1. **Dependency Injection**:
   * Use the **getBean** method to get the *Student* *object (student1)* from the Spring container.

Student student1 = (Student) context.getBean("student1");

1. **Print the Student Object**:
   * Output the Student object to the console.

System.out.println(student1);

A computer screen with text and green text

Description automatically generated