# Day 26- Facilitation Guide

## Index

## For (1.5 hrs) ILT

**During the previous session about the Hibernate cache mechanism,Second level cache, we covered a variety of important concepts:**

Hibernate Caching:

Hibernate caching is a mechanism that allows Hibernate, an Object-Relational Mapping (ORM) framework, to store frequently accessed data in memory. Caching reduces the need to repeatedly query the database, enhancing application performance.

Hibernate provides two main levels of caching: the first-level cache (Session cache) and the second-level cache (SessionFactory cache). The first-level cache is associated with a specific session, while the second-level cache is shared across sessions and can store data at a global level.

Second-Level Cache:

The second-level cache in Hibernate is a shared, more global caching mechanism that enhances performance by storing data objects that are commonly accessed across multiple sessions and transactions. It's a crucial component of Hibernate's caching strategy. The second-level cache reduces the need to hit the database for data that can be cached in memory. Cache providers, such as Ehcache, Infinispan, and Redis, can be configured to handle the second-level cache.

**In this session, we'll investigate Spring framework, Dependency injection, and IOC container, along with the implementation of setter and getter dependency.**

## I. Spring framework

Spring is a lightweight and popular open-source Java-based framework developed by Rod Johnson in 2003. It is used to develop enterprise-level applications. It provides support to many other frameworks such as Hibernate, Tapestry, EJB, JSF, Struts, etc, so it is also called a framework of frameworks. It's an application framework and IOC (Inversion of Control) container for the Java platform. The spring contains several modules like IOC, AOP, DAO, Context, WEB MVC, etc.

Why use Spring?

Spring framework is a Java platform that is open source. Rod Johnson created it, and it was first released under the Apache 2.0 license in June 2003.
When it comes to size and transparency, Spring is a featherweight. Spring framework's basic version is about 2MB in size.

The Spring Framework's core capabilities can be used to create any Java program, however there are modifications for constructing web applications on top of the Java EE platform. By offering a POJO-based programming model, the Spring framework aims to make J2EE development easier to use and to promote good programming habits.

Applications of Spring

Following is the list of few of the great benefits of using Spring Framework −

**POJO Based** - Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.

**Modular** - Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.

**Integration with existing frameworks** - Spring does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.

**Testablity** - Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBeanstyle POJOs, it becomes easier to use dependency injection for injecting test data.

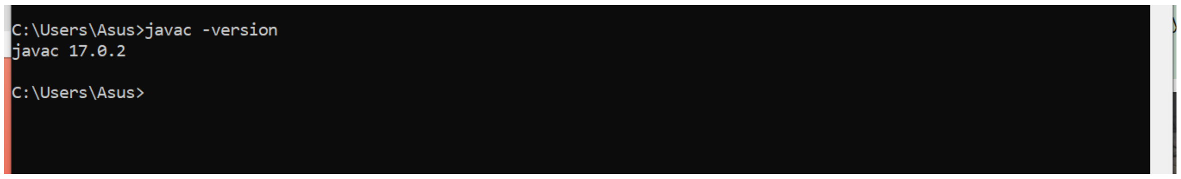
**Web MVC** - Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
**Central Exception Handling** - Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
**Lightweight** - Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
**Transaction management** - Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

## II. Software Setup and Configuration

Download and Install Spring Tool Suite (Spring Tools 4 for Eclipse) IDE:

Spring Tool Suite (STS) is a java IDE tailored for developing Spring-based enterprise applications. It is easier, faster, and more convenient. And most importantly it is based on Eclipse IDE. STS is free, open-source, and powered by VMware. Spring Tools 4 is the next generation of Spring tooling for the favorite coding environment. Largely rebuilt from scratch, it provides world-class support for developing Spring-based enterprise applications, whether you prefer Eclipse, Visual Studio Code, or Theia IDE.
Prerequisite: Make sure you have installed Java Development Kit (JDK) version 8 or newer. To check, simply go to the terminal and enter the below command to check if it is present or not.

javac -version



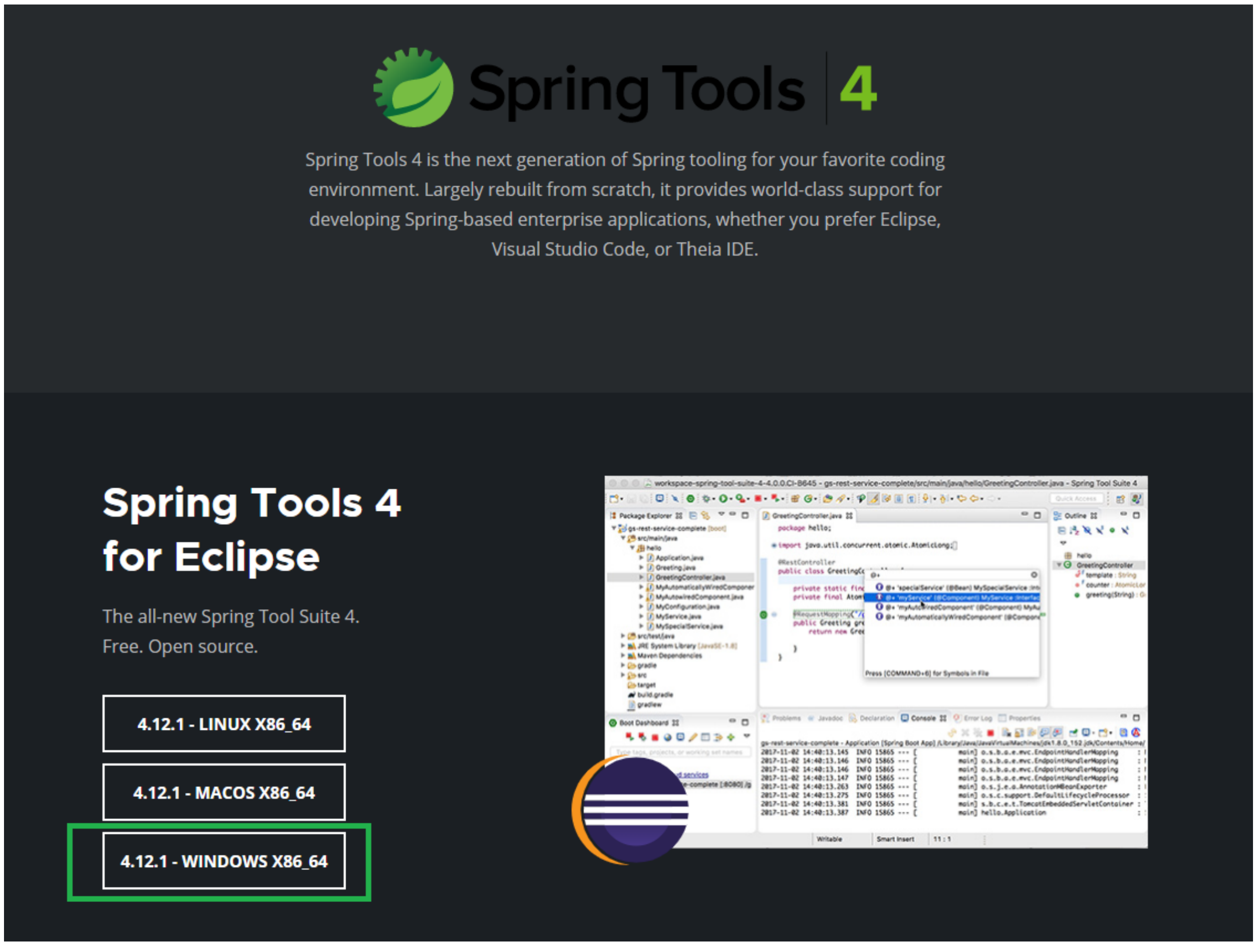
```
C:\Users\Asus>javac -version
javac 17.0.2

C:\Users\Asus>
```

Procedure:
1. Download SpringToolSuite as per the operating system to the local machine.
2. Move the downloaded JAR file to the corresponding folder.
3. Unzip this JAR file and open the corresponding folder.

4. Click on the SpringToolSuite4 Application file
5. Select the directory representing workspace and press the ‘LAUNCH’ button.

**Step 1: Go to their website and in Spring Tools 4 for the Eclipse section in order to download. choose your corresponding file according to your OS.**

***Note: Here we are going with Windows operating systems so we have chosen the Windows option as seen in the below image.***

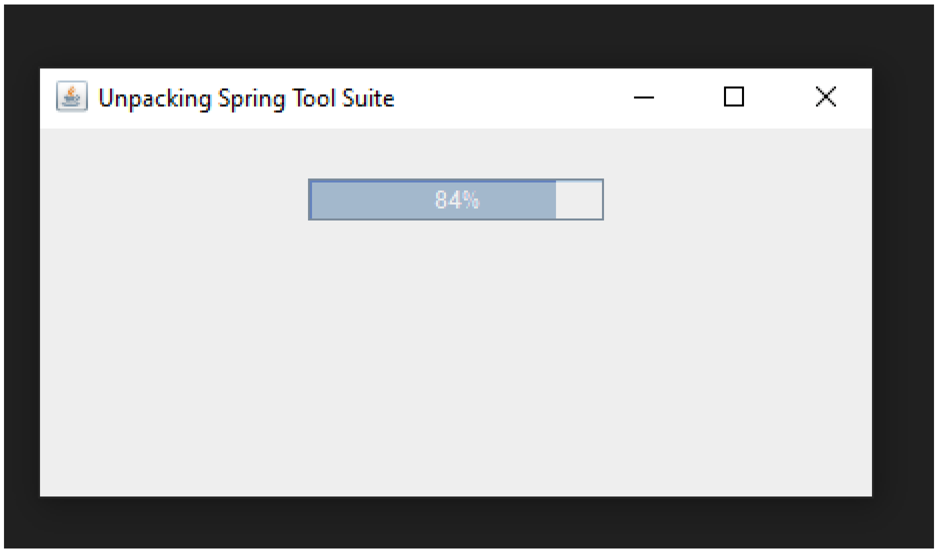

Spring Tools 4

Spring Tools 4 is the next generation of Spring tooling for your favorite coding environment. Largely rebuilt from scratch, it provides world-class support for developing Spring-based enterprise applications, whether you prefer Eclipse, Visual Studio Code, or Theia IDE.

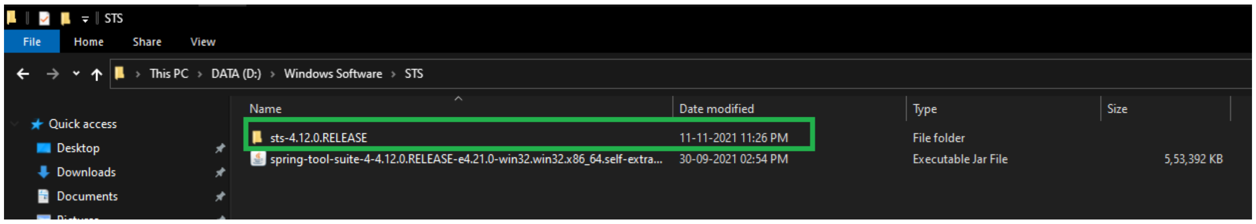
Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4.
Free. Open source.

Step 2: After clicking on the button a Jar file will be downloaded to your local system. Now create a folder and move this Jar file to that folder. And double-click on that Jar file. A pop-up window will appear like this.

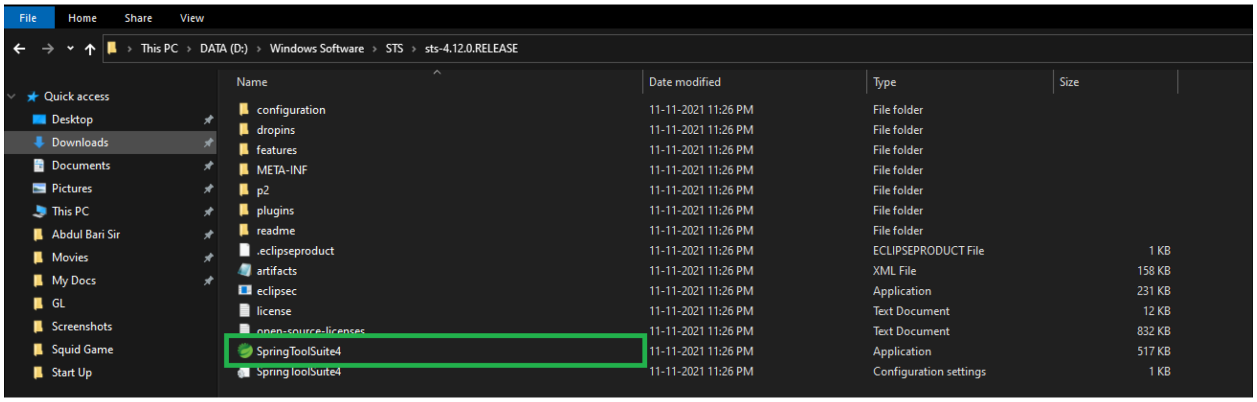


**Step 3:** After successfully Unpacking a new folder will create as shown in the below image. Now open the folder.



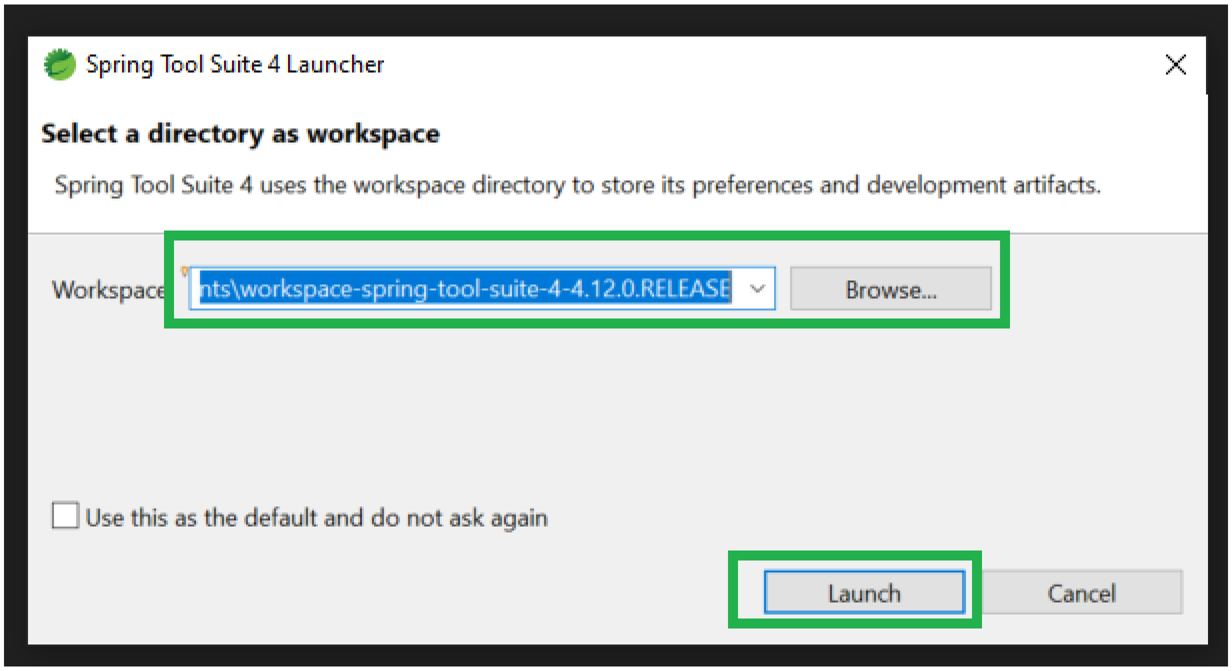
| Name | Date modified | Type | Size |
|---|---|---|---|
| sts-4.12.0.RELEASE | 11-11-2021 11:26 PM | File folder | |
| spring-tool-suite-4-4.12.0.RELEASE-e4.21.0-win32.win32.x86_64.self-extra... | 30-09-2021 02:54 PM | Executable Jar File | 5,53,392 KB |

**Step 4:** In this folder now click on the **SpringToolSuite4** Application file as shown in the below image.



| Name | Date modified | Type | Size |
|---|---|---|---|
| configuration | 11-11-2021 11:26 PM | File folder | |
| dropins | 11-11-2021 11:26 PM | File folder | |
| features | 11-11-2021 11:26 PM | File folder | |
| META-INF | 11-11-2021 11:26 PM | File folder | |
| p2 | 11-11-2021 11:26 PM | File folder | |
| plugins | 11-11-2021 11:26 PM | File folder | |
| readme | 11-11-2021 11:26 PM | File folder | |
| .eclipseproduct | 11-11-2021 11:26 PM | ECLIPSEPRODUCT File | 1 KB |
| artifacts | 11-11-2021 11:26 PM | XML File | 158 KB |
| eclipsec | 11-11-2021 11:26 PM | Application | 231 KB |
| license | 11-11-2021 11:26 PM | Text Document | 12 KB |
| open-source-licenses | 11-11-2021 11:26 PM | Text Document | 832 KB |
| SpringToolSuite4 | 11-11-2021 11:26 PM | Application | 517 KB |
| SpringToolSuite4 | 11-11-2021 11:26 PM | Configuration settings | 1 KB |

**Step 5:** Now select your directory as workspace by clicking on the **Browse** button and then click on the **Launch** button. And you are done.



Spring Tool Suite 4 Launcher

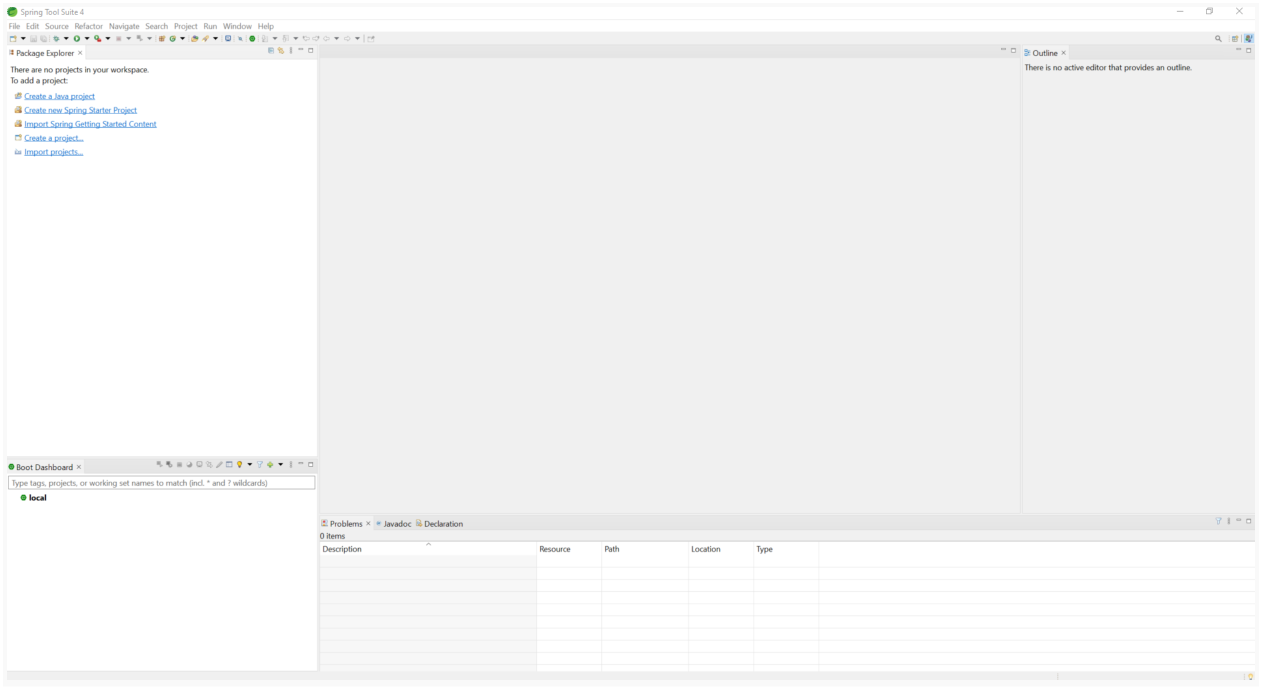
**Select a directory as workspace**

Spring Tool Suite 4 uses the workspace directory to store its preferences and development artifacts.

Workspace: nts\workspace-spring-tool-suite-4-4.12.0.RELEASE

☐ Use this as the default and do not ask again

*This is the **Home screen for Spring Tool Suite (Spring Tools 4 for Eclipse) IDE***



Spring Tool Suite 4

Package Explorer

There are no projects in your workspace.
To add a project:

- Create a Java project
- Create new Spring Starter Project
- Import Spring Getting Started Content
- Create a project...
- Import projects...

Outline

There is no active editor that provides an outline.

Boot Dashboard

Type tags, projects, or working set names to match (incl. * and ? wildcards)

local

Problems | Javadoc | Declaration

0 items

| Description | Resource | Path | Location | Type |
|---|---|---|---|---|

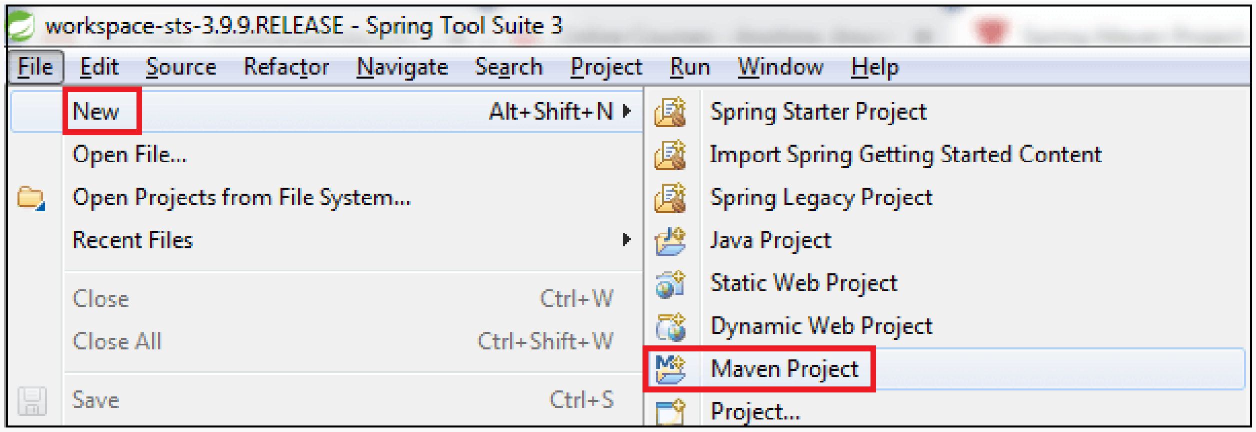
## Let's create project in Spring Tool Suite

Creating a Spring Boot Project Using STS:

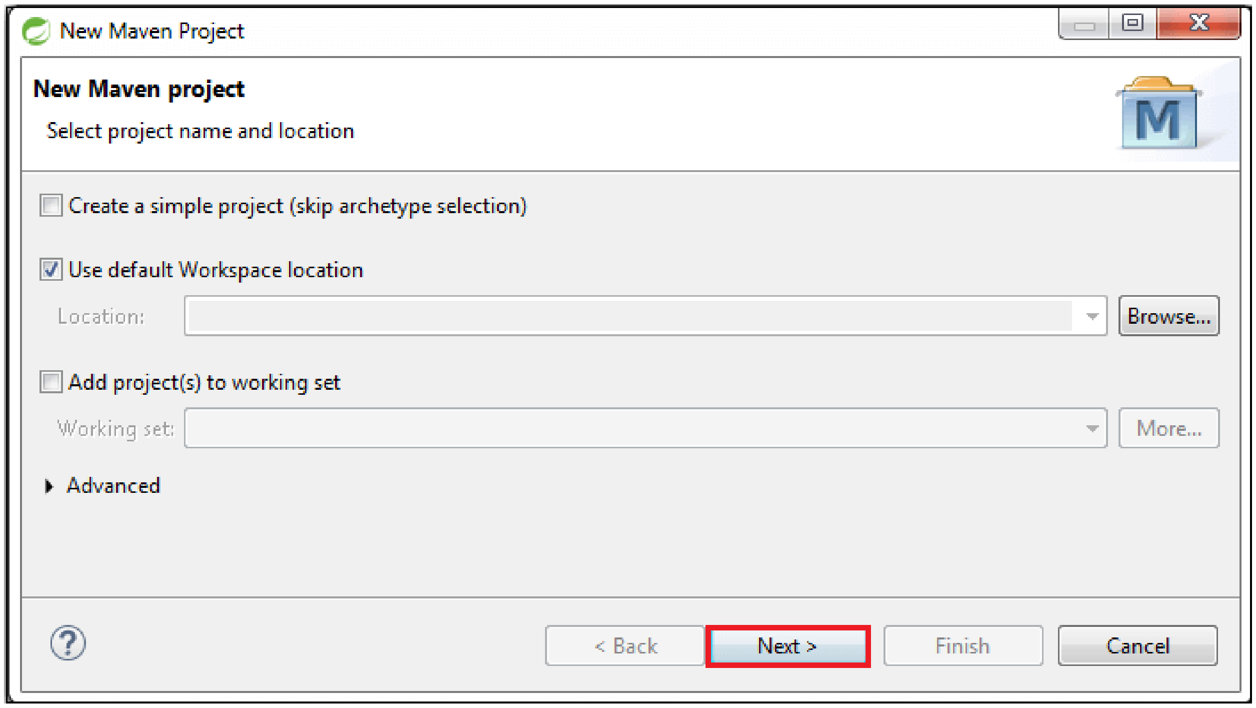
We can also use Spring Tool Suite to create a Spring project. In this section, we will create a **Maven Project** using **STS**.

**Step 1:** Open the Spring Tool Suite.

**Step 2:** Click on the File menu -> New -> Maven Project



It shows the New Maven Project wizard. Click on the **Next** button.


New Maven Project

**New Maven project**

Select project name and location

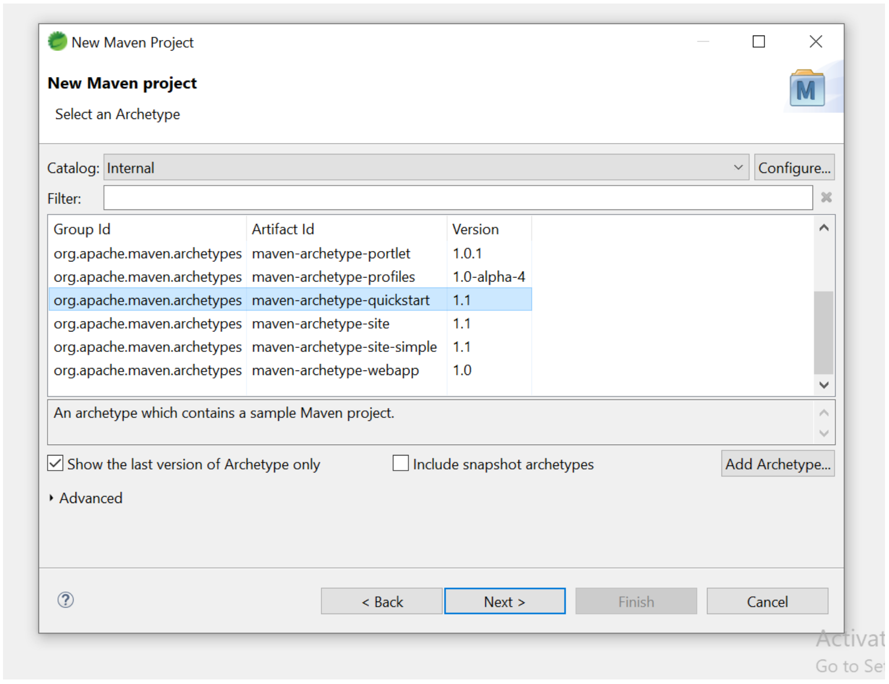
- [ ] Create a simple project (skip archetype selection)
- [✓] Use default Workspace location

Location:

- [ ] Add project(s) to working set

Working set:

**Step 3:** Select the **maven-archetype-quickstart** and click on the **Next** button.



**New Maven project**

Select an Archetype

Catalog: Internal

Filter:

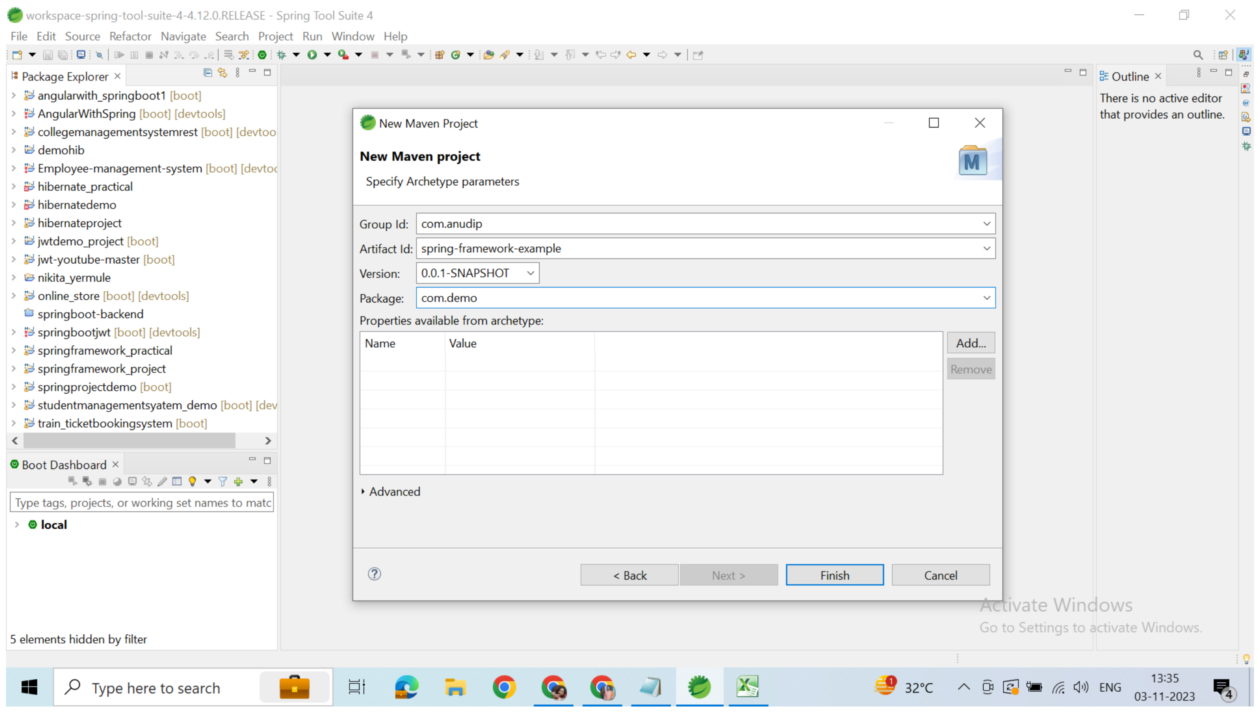
| Group Id | Artifact Id | Version |
|---|---|---|
| org.apache.maven.archetypes | maven-archetype-portlet | 1.0.1 |
| org.apache.maven.archetypes | maven-archetype-profiles | 1.0-alpha-4 |
| org.apache.maven.archetypes | maven-archetype-quickstart | 1.1 |
| org.apache.maven.archetypes | maven-archetype-site | 1.1 |
| org.apache.maven.archetypes | maven-archetype-site-simple | 1.1 |
| org.apache.maven.archetypes | maven-archetype-webapp | 1.0 |

An archetype which contains a sample Maven project.

[✓] Show the last version of Archetype only   [ ] Include snapshot archetypes

Advanced

**Step 4:** Provide the **Group Id** and **Artifact Id**. We have provided Group Id **com.anudip** and Artifact Id **spring-framework-example**. Now click on the **Finish** button.


workspace-spring-tool-suite-4-4.12.0.RELEASE - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
angularwith_springboot1 [boot]
AngularWithSpring [boot] [devtools]
collegemanagementsystemrest [boot] [devtoo
demohib
Employee-management-system [boot] [devtoo
hibernate_practical
hibernatedemo
hibernateproject
jwtdemo_project [boot]
jwt-youtube-master [boot]
nikita_yermule
online_store [boot] [devtools]
springboot-backend
springbootjwt [boot] [devtools]
springframework_practical
springframework_project
springprojectdemo [boot]
studentmanagementsyatem_demo [boot] [dev
train_ticketbookingsystem [boot]
Boot Dashboard
Type tags, projects, or working set names to matc
local
5 elements hidden by filter
Outline
There is no active editor that provides an outline.
New Maven Project
Add...
Remove
< Back
Next >
Finish
Cancel
Activate Windows
Go to Settings to activate Windows.
Type here to search
32°C
ENG
13:35
03-11-2023

**New Maven project**

Specify Archetype parameters

Group Id: com.anudip

Artifact Id: spring-framework-example

Version: 0.0.1-SNAPSHOT

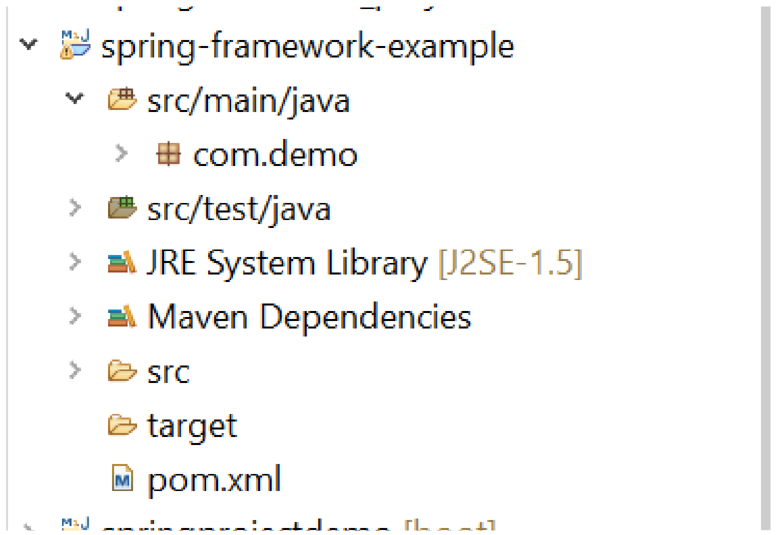
Package: com.demo

Properties available from archetype:

| Name | Value |
|---|---|

Advanced

When we click on the Finish button, it creates the project directory, as shown in the following image.



**Step 5:** Open the **App.java** file. We found the following code that is by default.

**App.java**

```
package com.javatpoint;
public class App
{
public static void main( String[] args )
{
System.out.println( "Hello World!" );
}
}
```

The Maven project has a **pom.xml** file which contains the following default configuration.

**pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.anudip</groupId>
  <artifactId>spring-framework-example</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>spring-framework-example</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

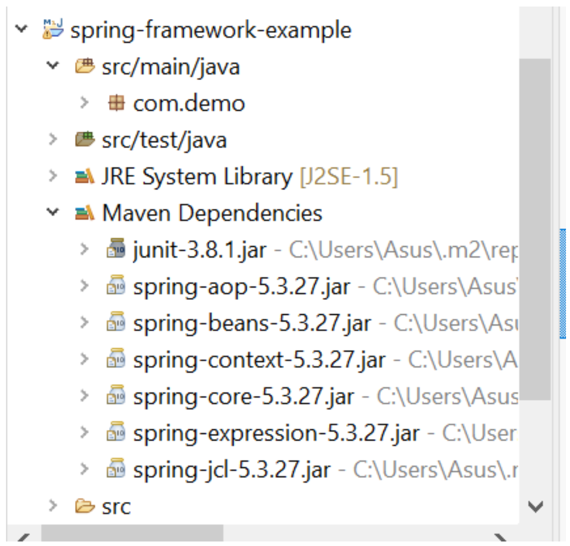
**Step 6:** Add Java version inside the <properties> tag.

```
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
```

**Step 7:** In order to make a Spring Project, we need to configure it. So, we are adding spring-core and spring-context dependency in the pom.xml file.

```
  <dependencies>
  <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.27</version>
</dependency>

 <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.27</version>
</dependency>

</dependencies>
```

**Note:** When we add the dependencies in the pom file, it downloads the related jar file. We can see the downloaded jar files in the Maven Dependencies folder of the project directory.



```
spring-framework-example
  src/main/java
    com.demo
  src/test/java
  JRE System Library [J2SE-1.5]
  Maven Dependencies
    junit-3.8.1.jar - C:\Users\Asus\.m2\rep
    spring-aop-5.3.27.jar - C:\Users\Asus
    spring-beans-5.3.27.jar - C:\Users\Asu
    spring-context-5.3.27.jar - C:\Users\A
    spring-core-5.3.27.jar - C:\Users\Asus
    spring-expression-5.3.27.jar - C:\User
    spring-jcl-5.3.27.jar - C:\Users\Asus\.r
  src
```

After adding all the dependencies, the pom.xml file looks like the following:

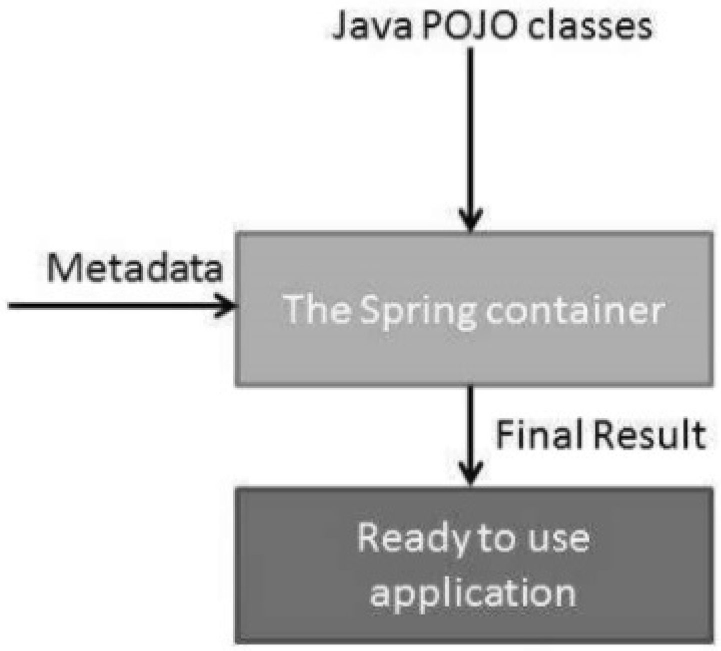
**pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.anudip</groupId>
  <artifactId>spring-framework-example</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>spring-framework-example</name>
  <url>http://maven.apache.org</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
     <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>

  </properties>

  <dependencies>


  <!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>5.3.27</version>
</dependency>


 <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.27</version>
</dependency>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

**III. IOC container**

The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses DI to manage the components that make up an application. These objects are called Spring Beans, which we will discuss in the next Session.

The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code. The following diagram represents a high-level view of how Spring works. The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.



Spring provides the following two distinct types of containers:

**Spring BeanFactory Container:**

This is the simplest container providing the basic support for DI and is defined by the org.springframework.beans.factory.BeanFactory interface. The BeanFactory and related interfaces, such as BeanFactoryAware, InitializingBean, DisposableBean, are still present in Spring for the purpose of backward compatibility with a large number of third-party frameworks that integrate with Spring.

Resource resource=**new** ClassPathResource("applicationContext.xml");

BeanFactory factory=**new** XmlBeanFactory(resource);

The constructor of XmlBeanFactory class receives the Resource object so we need to pass the resource object to create the object of BeanFactory.

**Spring ApplicationContext Container:**

This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by the org.springframework.context.ApplicationContext interface.

ApplicationContext context =

**new** ClassPathXmlApplicationContext("applicationContext.xml");

The constructor of ClassPathXmlApplicationContext class receives string, so we can pass the name of the xml file to create the instance of ApplicationContext.

**Note:** The ApplicationContext container includes all functionality of the BeanFactorycontainer, so it is generally recommended over BeanFactory. BeanFactory can still be used for lightweight applications like mobile devices or applet-based applications where data volume and speed is significant.

In the same session, we'll delve into practical implementations and real-world applications.

***Trainer will ask questions to students to increase the curiosity:***

***How does Spring handle circular dependencies between beans in the IoC container?***

**IV. Dependency Injection**

Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The design principle of Inversion of Control emphasizes keeping the Java classes independent of each other and the container frees them from object creation and maintenance. These classes, managed by Spring, must adhere to the standard definition of Java-Bean. Dependency Injection in Spring also ensures loose-coupling between the classes.

Need for Dependency Injection:

Suppose class One needs the object of class Two to instantiate or operate a method, then class One is said to be dependent on class Two. Now though it might appear okay to depend a module on the other but, in the real world, this could lead to a lot of problems, including system failure. Hence such dependencies need to be avoided. Spring IOC resolves such dependencies with Dependency Injection, which makes the code easier to test and reuse. Loose coupling between classes can be possible by defining interfaces for common functionality and the injector will instantiate the objects of required implementation. The task of instantiating objects is done by the container according to the configurations specified by the developer.

Types of Spring Dependency Injection:

There are two types of Spring Dependency Injection. They are:

1.Setter Dependency Injection
2.Constructor Dependency Injection

**Knowledge check…**

***Now that we've covered the basic understanding of IOC container and Dependency Injection, it's time to test your understanding. The Trainer will conduct a short poll quiz to assess your knowledge on these topics.***

**1.What does the IoC container in Spring stand for?**
a) Inversion of Classes
b) Inversion of Configuration
c) Inversion of Control
d) Inversion of Components

2. Which type of dependency injection is considered a best practice because it ensures that the object is in a valid state immediately after construction?
a) Constructor Injection

b) Setter Injection
c) Field Injection
d) Property Injection

**V. Setter and Constructor Injection**

1. **Setter Dependency Injection (SDI):** This is the simpler of the two DI methods. In this, the DI will be injected with the help of setter and/or getter methods. Now to set the DI as SDI in the bean, it is done through the bean-configuration file For this, the property to be set with the SDI is declared under the <property> tag in the bean-config file.

Example: Let us say there is a class Student that uses SDI and sets the properties of Student class. The code for it is given below.

**Student Bean Class:**

```
package com.demo;


import java.util.Date;

public class Student {

        private String studentId;
        private String firstName;
        private String lastName;
        private Date dateOfBirth;
        private String gender;
        private String email;
        private String phone;

//defining Setter and Getter Method
        public String getStudentId() {
                return studentId;
        }
}



public void setStudentId(String studentId) {
        this.studentId = studentId;
}

public String getFirstName() {
        return firstName;
}

public void setFirstName(String firstName) {
        this.firstName = firstName;
}

public String getLastName() {
        return lastName;
}

public void setLastName(String lastName) {
        this.lastName = lastName;
}

public Date getDateOfBirth() {
        return dateOfBirth;
}

public void setDateOfBirth(Date dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
}

public String getGender() {
        return gender;
}

public void setGender(String gender) {
        this.gender = gender;
}

public String getEmail() {
        return email;
}
```

```
	public void setEmail(String email) {
		this.email = email;
	}

	public String getPhone() {
		return phone;
	}

	public void setPhone(String phone) {
		this.phone = phone;
	}

//defining toString()
	@Override
	public String toString() {
		return "Student [studentId=" + studentId + ", firstName=" + firstName + ", 
lastName=" + lastName
				+ ", dateOfBirth=" + dateOfBirth + ", gender=" + gender + ", 
email=" + email + ", phone=" + phone + "]";
	}

}
```

**Setting the SDI in the bean-config file:**

Create a bean-config.xml file under src/main/java folder and add document type definition(DTD) in the file.

```
<?xml version="1.0" encoding="UTF-8"?>
<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-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">
```

```
  <!-- Bean Configuration -->
  <bean id="dateFormater" class="java.text.SimpleDateFormat">
      <constructor-arg value="dd-MM-yyyy" />
   </bean>

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

  <property name="studentId" value="S101"/>
  <property name="firstName" value="John"/>
  <property name="lastName" value="Doe"/>

     <property name="dateOfBirth">
          <bean factory-bean="dateFormater" factory-method="parse">
           <constructor-arg value="1996-08-12" />
          </bean>
      </property>
  <property name="gender" value="M"/>
  <property name="email" value="john@gamil.com"/>
  <property name="phone" value="8890678456"/>
  </bean>

  </beans>
```

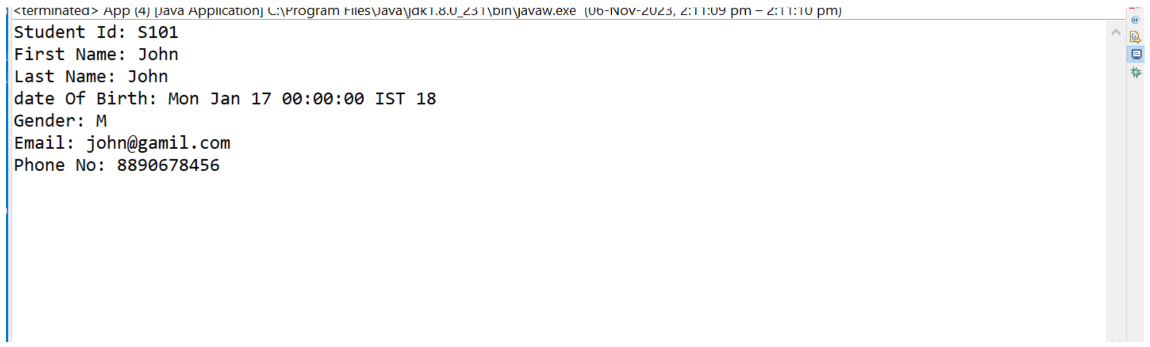
In this context, we are populating the properties of the Student object with values through injection.

Finally, you can create a Spring application to test the setter injection:
**App.java**

```
package com.demo;

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

/**
 * Hello world!
 *
 */
public class App
```

```
{
   public static void main( String[] args )
   {
           // Load the Spring configuration
          ApplicationContext context=new
ClassPathXmlApplicationContext("bean-config.xml");

          // Get the Student bean from the Spring container
          Student student=(Student)context.getBean("student1");

          // Display the Student object
          System.out.println("Student Id: "+student.getStudentId());
          System.out.println("First Name: "+student.getFirstName());
          System.out.println("Last Name: "+student.getFirstName());
          System.out.println("date Of Birth: "+student.getDateOfBirth());
          System.out.println("Gender: "+student.getGender());
          System.out.println("Email: "+student.getEmail());
          System.out.println("Phone No: "+student.getPhone());
   }
}
```

**Output:**



```
<terminated> App (4) [Java Application] C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe (06-Nov-2023, 2:11:09 pm – 2:11:10 pm)
Student Id: S101
First Name: John
Last Name: John
date Of Birth: Mon Jan 17 00:00:00 IST 18
Gender: M
Email: john@gamil.com
Phone No: 8890678456
```

**2. Constructor Dependency Injection (CDI):** In this, the DA will be injected with the help of contructors. Now to set the DI as CDI in bean, it is done through the bean-configuration file For this, the property to be set with the CDI is declared under the <constructor-arg> tag in the bean-config file.

Example: Let us say there is a class Student that uses CDI and sets the properties of Student class. The code for it is given below.

**Student Bean Class:**

```
package com.demo;


import java.util.Date;

public class Student {

        private String studentId;
        private String firstName;
        private String lastName;
        private Date dateOfBirth;
        private String gender;
        private String email;
        private String phone;

//defining Getter Method
        public String getStudentId() {
                return studentId;
        }


        public String getFirstName() {
                return firstName;
        }




        public String getLastName() {
                return lastName;
        }


        public Date getDateOfBirth() {
                return dateOfBirth;
        }
```

```
	public String getGender() {
		return gender;
	}



	public String getEmail() {
		return email;
	}



	public String getPhone() {
		return phone;
	}

	//defining Parameterized Constructor
	public Student(String studentId, String firstName, String lastName, Date
dateOfBirth, String gender, String email,
			String phone) {
		super();
		this.studentId = studentId;
		this.firstName = firstName;
		this.lastName = lastName;
		this.dateOfBirth = dateOfBirth;
		this.gender = gender;
		this.email = email;
		this.phone = phone;
	}

	//defining Default Constructor

	public Student() {
		super();
		// TODO Auto-generated constructor stub
	}

//defining toString()
	@Override
	public String toString() {
		return "Student [studentId=" + studentId + ", firstName=" + firstName + ",
lastName=" + lastName
				+ ", dateOfBirth=" + dateOfBirth + ", gender=" + gender + ",
email=" + email + ", phone=" + phone + "]";
	}

}
```

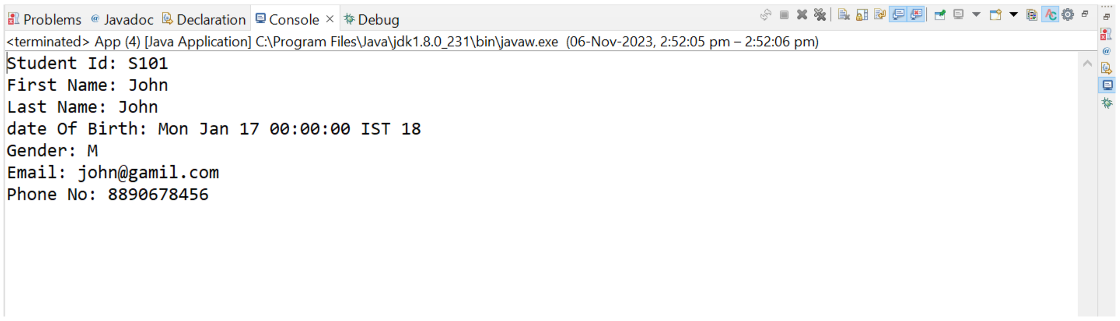
**Setting the CDI in the bean-config file:**

```
<?xml version="1.0" encoding="UTF-8"?>
<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-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <!-- Bean Configuration -->
   <bean id="dateFormater" class="java.text.SimpleDateFormat">
        <constructor-arg value="dd-MM-yyyy" />
    </bean>

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

   <constructor-arg value="S101"/>
   <constructor-arg value="John"/>
   <constructor-arg value="Doe"/>

      <constructor-arg name="dateOfBirth">
            <bean factory-bean="dateFormater" factory-method="parse">
             <constructor-arg value="1996-08-12" />
            </bean>
        </constructor-arg>
   <constructor-arg value="M"/>
   <constructor-arg value="john@gamil.com"/>
   <constructor-arg value="8890678456"/>
   </bean>
   </beans>
```

Finally, you can create a Spring application to test the setter injection:

```
package com.demo;

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

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
          // Load the Spring configuration
         ApplicationContext context=new
ClassPathXmlApplicationContext("bean-config.xml");

         // Get the Student bean from the Spring container
         Student student=(Student)context.getBean("student1");

         // Display the Student object
         System.out.println("Student Id: "+student.getStudentId());
         System.out.println("First Name: "+student.getFirstName());
         System.out.println("Last Name: "+student.getFirstName());
         System.out.println("date Of Birth: "+student.getDateOfBirth());
         System.out.println("Gender: "+student.getGender());
         System.out.println("Email: "+student.getEmail());
         System.out.println("Phone No: "+student.getPhone());
    }
}
```

**Output**



```
<terminated> App (4) [Java Application] C:\Program Files\Java\jdk1.8.0_231\bin\javaw.exe  (06-Nov-2023, 2:52:05 pm – 2:52:06 pm)
Student Id: S101
First Name: John
Last Name: John
date Of Birth: Mon Jan 17 00:00:00 IST 18
Gender: M
Email: john@gamil.com
Phone No: 8890678456
```

**Food for thought..**

The trainer can ask the students the following question to engage them in a discussion:

How does dependency injection promote the concept of "Inversion of Control" (IoC) in software design?

## Exercise:

**Use ChatGPT to explore more about Dependency Injection:**

**Put the below problem statement in the message box and see what ChatGPT says.**

Which type of dependency injection is considered a best practice because it ensures that the object is in a valid state immediately after construction?