Spring - TOC

14 November 2025 16:39

https://docs.spring.io/spring-framework/docs/3.0.x/spring-framework-reference/html/overview.html

Module 1 -
Part-1.1 Framework & IOC basic (Inversion of Control)
Part-1.2 Java Object Creation vs Spring Beans and how the container manages objects
Part-1.3 Spring Architecture and Spring Modules
Part-1.4 Dependency Injection -- Constructor , Setter and Field Injection
Part-1.5 Spring Configuration styles -- XML , Javaconfig and Annotations
Part-1.6 Bean Scope , Lifecycle callbacks , the application context
Part-1.7 Types of containers -- Bean Factory , Application Context

Maven

15 November 2025 09:13

Maven is a build tool or build automation and dependency management tool for java projects .

Maved Tool : A tool that builds your java project + downloads some required JAR files automatically + manages project structure

What it does :

1)**Download Jar files manually** --- from maven central repository -- select the dependency with the correct version from this maven central repository copy the dependency (under maven tab) and add it in your project in pom.xml

For eg:

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
   <groupId>org.springframework</groupId>   ---- is your package name
   <artifactId>spring-context</artifactId>        ---- is your module name
   <version>6.2.11</version>
</dependency>
```

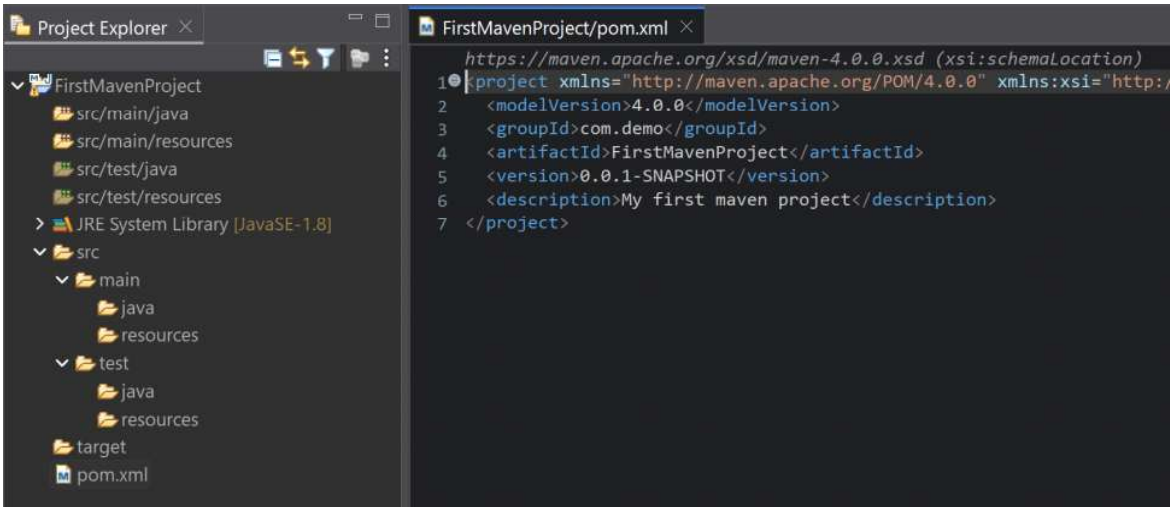
Now Maven will download the above dependency you defined in POM.xml and set in your class path --- you will see a **maven repository folder (In this folder you would be able to see your jar files)**

2) **Set Build path Manually**
3) **Managing Project structure**
4) **No need to worry about different versions of dependencies**

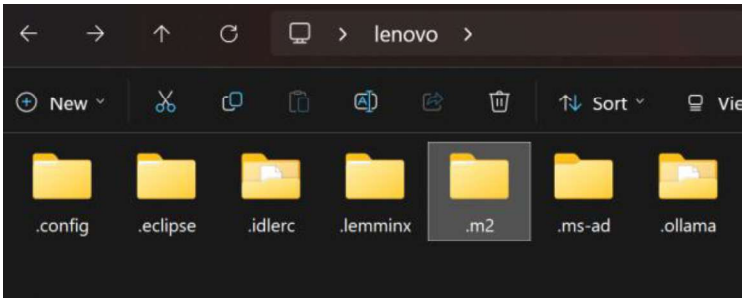
In real scenarios :

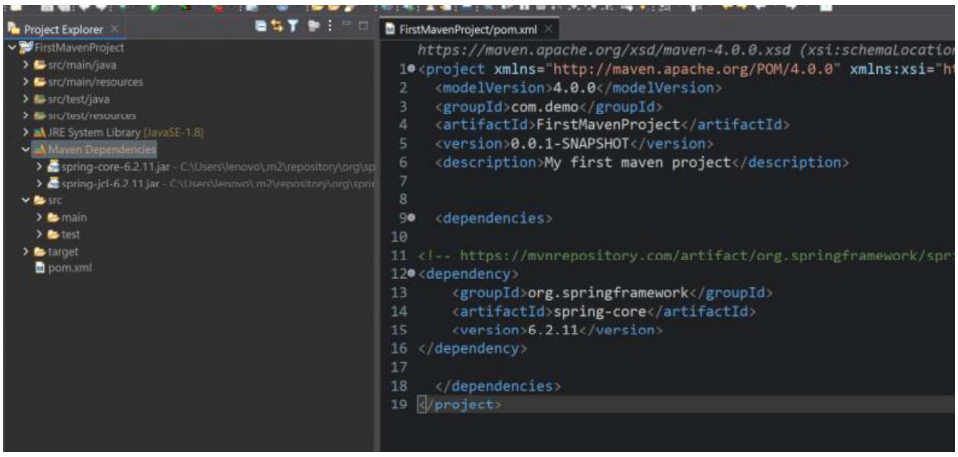
a) Adding multiple jar files (20 -30) in pom.xml **, where maven downloads automatically**
b) Earlier developer a uses a jar v1.0 , developer b uses 1.2 **but here maven ensures same versions for everyone**
c) Earlier project build manually with commands **but maven builds with mvn clean install**
d) Earlier it was hard to manage multi- module projects **- Maven manages modules with parent POM**

Maven Project Structure --- When you created manually



2) Addded the spring core dependency **once you save** this it will be downloaded locally in your system in
.m2\repository folder and it will show the jar file under maven repository folder



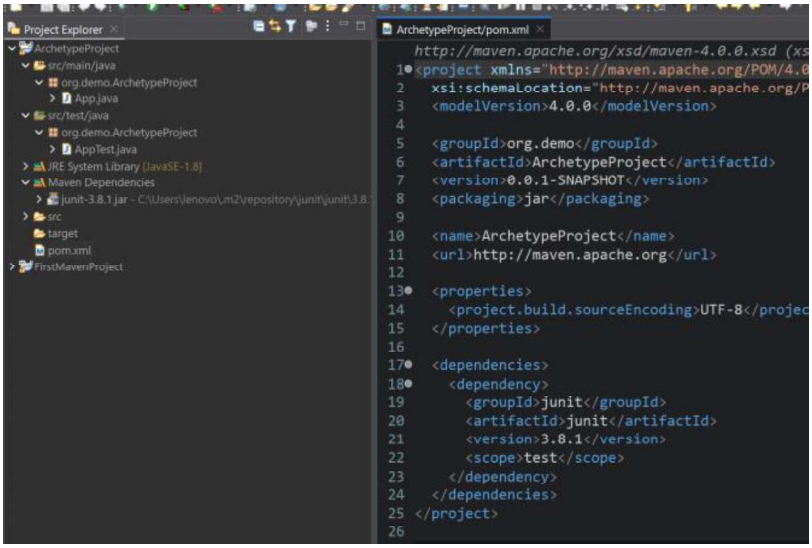


POM.xml is the main file of your project which is Project Object Model where you define all the dependencies
What it includes :

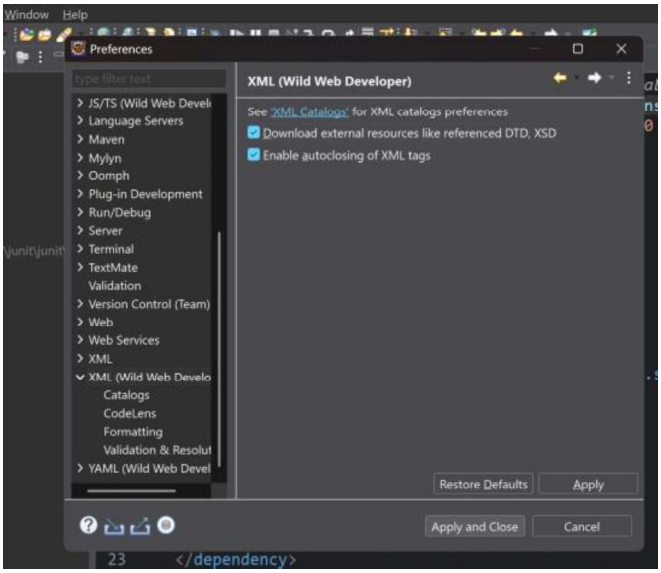
- Project name
- Version
- Dependencies(jars)
- Packaging type(jar/war)
- Plugins
- Build configuration

Under src/main/java ---- java class files will be there
Src/main/resources --- config files (like application.properties or xml file)
Src/main/webapp --- you can create a folder to add jsp , html files

Maven Project Structure --- When it is created as archetype



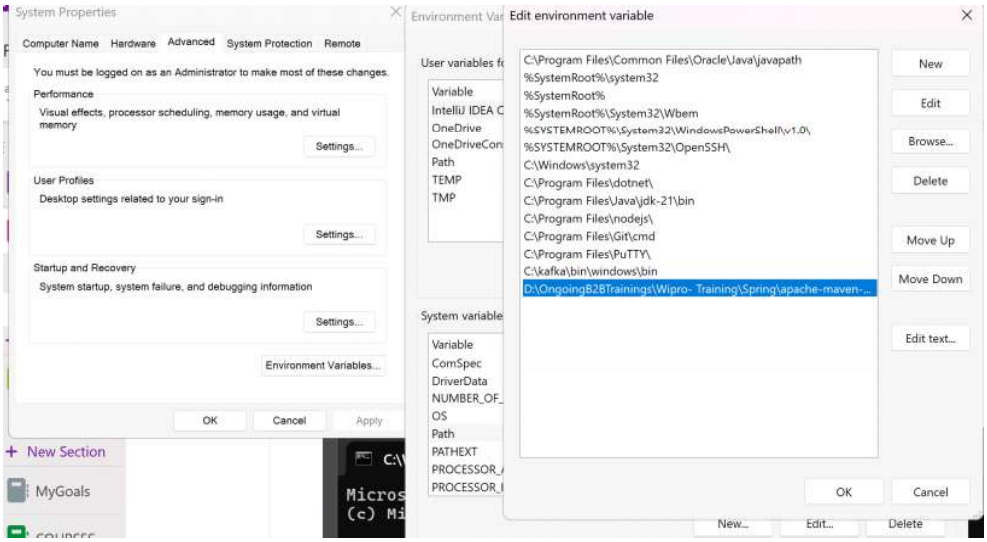
Do the settings

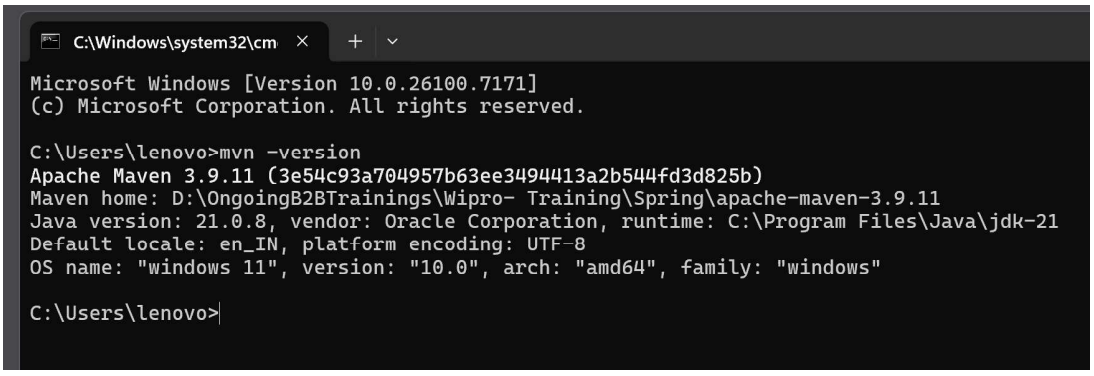


Download the Maven and set in environment variables

https://maven.apache.org/download.cgi

Once you download and set the path in env. Variables
D:\OngoingB2BTrainings\Wipro- Training\Spring\apache-maven-3.9.11\bin





Maven Build LifeCycle and Goals

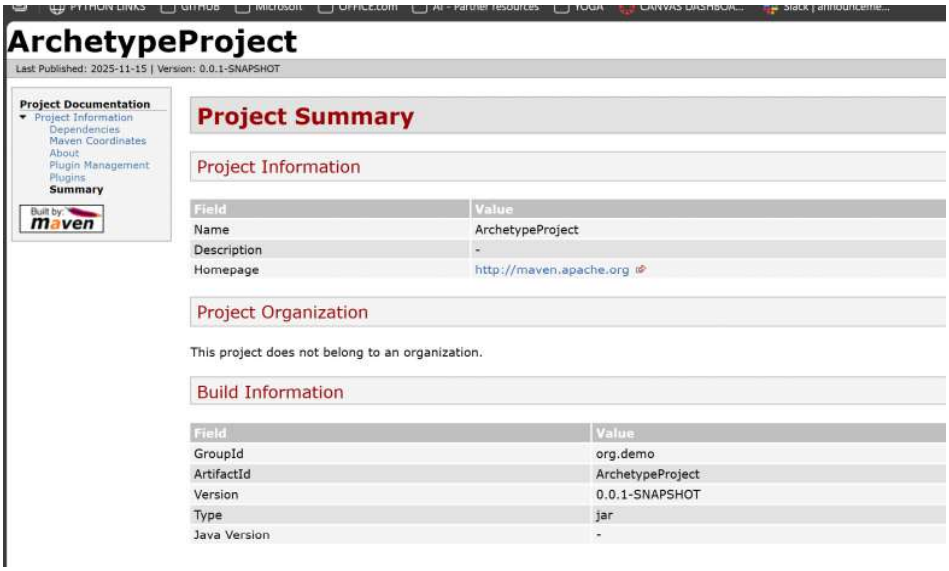
Clean -- mvn clean -- Deletes target/ compiled files
Default (Build lifecycle) -- what it includes

a. Validate
b. Compile -- mvn compile -- Compiles java code
c. Test -- mvn test -- runs junit tests
d. Package -- mvn package -- create jar or war file
e. Verify
f. Install -- mvn install --- adds jar/war to local repository
g. Deploy

To see the hierarchy use the command as mvn dependency:tree
Site

a. Documentation -- mvn site

```
D:\OngoingB2BTrainings\Wipro- Training\Spring\ArchetypeProject>cd target

D:\OngoingB2BTrainings\Wipro- Training\Spring\ArchetypeProject\target>cd site

D:\OngoingB2BTrainings\Wipro- Training\Spring\ArchetypeProject\target\site>dir
 Volume in drive D is New Volume
 Volume Serial Number is 26A1-80A8

 Directory of D:\OngoingB2BTrainings\Wipro- Training\Spring\ArchetypeProject\target\site

15-11-2025  11:01    <DIR>          .
15-11-2025  11:01    <DIR>          ..
15-11-2025  11:01    <DIR>          css
15-11-2025  11:01             7,387 dependencies.html
15-11-2025  11:01             2,841 dependency-info.html
15-11-2025  11:01    <DIR>          images
15-11-2025  11:01             2,852 index.html
15-11-2025  11:01             3,705 plugin-management.html
15-11-2025  11:01             4,913 plugins.html
15-11-2025  11:01             4,325 project-info.html
15-11-2025  11:01             3,918 summary.html
               7 File(s)         29,941 bytes
               4 Dir(s)  81,472,704,512 bytes free

D:\OngoingB2BTrainings\Wipro- Training\Spring\ArchetypeProject\target\site>index.html

D:\OngoingB2BTrainings\Wipro- Training\Spring\ArchetypeProject\target\site>
```



- **Examples of Common Goals:**
  - clean:clean: Removes the target directory and all generated build artifacts.
  - compiler:compile: Compiles the project's source code.
  - surefire:test: Executes unit tests.
  - jar:jar or war:war: Packages the compiled code into a JAR or WAR file.
  - install:install: Installs the project's artifact into the local Maven repository.

- deploy:deploy: Deploys the project's artifact to a remote Maven repository.

Module 1 - Intro -- Spring Fundamentals

14 November 2025 16:39

Part-1.1

Framework : (Framework is collection of libraries that we have collections in Core Java) so we have powerful frameworks like **struts , spring , hibernate , spring boot** for the developers to create enterprise level of application covering all real live aspects

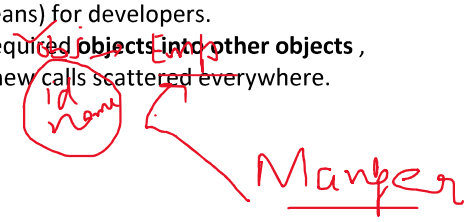
In Spring, the term "framework" refers to a comprehensive and integrated platform that provides a foundational structure and extensive support for developing Java-based enterprise applications. It is not merely a collection of libraries, but rather a cohesive ecosystem designed to simplify and streamline various aspects of application development.

IOC : Inversion of control "who controls object creation"
DI : Dependency Injection " How dependencies provided"

In core java , developers create objects with new keyword manually

Employee e = new Employee();

But in Spring , **a container(IOC)** creates and manages objects(beans) for developers.
Dependency Injection means Spring automatically injects the required **objects into other objects** ,
So your code depends on interfaces and configuration , not on new calls scattered everywhere.



Tight Coupling in Core Java

```
public class Employee
{
private Manager manager = new Manager();  // manual new // hard code dependency
}

// Here the drawback are :
```

1. You cannot replace the manager with ProductOwner without editing Employee
2. Unit testing is harder - you cannot mock Manager easily
3. The object lifecycle (new) is controlled by developer not a framework controlled

```
Class Manager
{

public void taskAllocation(String user)
{
System.out.println("Task is allocated by : Manager " + user);
}

}

Class Delegate{

private Manager m = new Manager(); // hard -code dependecy

Public notifyUser()
{
m.taskAllocation("Niti");
}

}
```

```
Public class App
{
Public static void main(String[] args)
{
Delegate d = new Delegate();
d.notifyUser();
}
}
```

Refer the Example Code **Part 1.1**

Part -1.2

A spring bean is an object created , managed and injected by the Spring IOC Container , not be using new keyword.

Let's say in core java time we have **team leaders** who hires developers themselves (manual creation)
But now in spring the hiring of developers is managed by HR (which is a spring container) :

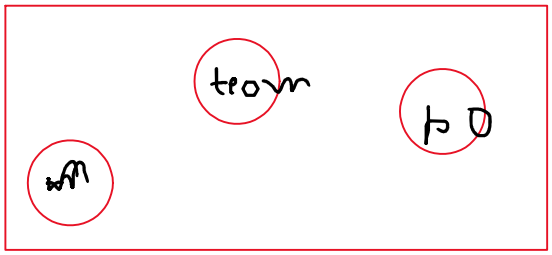
- Hiring
- Offer letter
- Maintain employees list
- Assign roles
- Also ensuring some replacement of employee who leave in between

In core java

```
Manager m= new Manager();
m.notifyUser("Niti");
```

In Spring

```
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);

          // ask the container or spring to get the delegate bean  or
object
          Manager m = context.getBean(Manager.class); // Spring will
create an object (new Manager())
       m.notifyUser("Niti");

Earlier Developer manages and controls the object life cycle but now
spring manages the life cycle of object (IOC) Inversion of control

Earlier no central registry  -- But now we have Bean Container
registry
```



Note: Class Name will be your Bean Name

Spring Life Cycle :

1. Class Scanned through AppConfig or any other configuration file you have defined or created
2. Beans will be instantiated
3. Dependencies Injected
4. It will be ready to used

In lifecycle there are call back added
@PostConstruct : Initialized
@PreDestroy : Cleanup before shut down

- **@PostConstruct** is standard and widely supported for methods that need to run after bean initialization.
- **@PreConstruct** is not part of standard Spring or Java EE specifications and is used in more specialized contexts or custom frameworks.
  In general, you would use @PostConstruct for initialization tasks in Spring-managed beans.

From <https://medium.com/must-read-articles-in-a-minute/preconstruct-and-postconstruct-annotation-12ff7c868f8b>

If you have tomcat 9 or below then use

```
<!--
https://mvnrepository.com/artifact/javax.annotation/javax.annotation-api -->
<dependency>
    <groupId>javax.annotation</groupId>
     <artifactId>javax.annotation-api</artifactId>
     <version>1.3.2</version>
</dependency>
```

And if it tomcat 10 or above then use below one:

```
<!--
https://mvnrepository.com/artifact/jakarta.annotation/jakarta.annotation-api -->
<dependency>
    <groupId>jakarta.annotation</groupId>
    <artifactId>jakarta.annotation-api</artifactId>
    <version>2.1.1</version>
</dependency>
```

Spring BeanScope :

Singleton(default) : One instance per Spring container
Prototype : New instance will be created per every request

For eg:

```
Manager m1 = context.getBean(Manager.class);
Manager m2 = context.getBean(Manager.class);
m1.hashCode();
m2.hashCode();
```

Which will be giving same instance or hashCode in spring because of singleton