

Basics of Apache Maven – project management tool

Welcome

[License](#)

ABOUT MAVEN

[What is Maven?](#)

[Features](#)

[Download](#)

[Use](#) >

[Release Notes](#)

DOCUMENTATION

[Maven Plugins](#)

[Maven Extensions](#)

[Index \(category\)](#)

[User Centre](#) >

[Plugin Developer Centre](#) >

[Maven Repository Centre](#) >

[Maven Developer Centre](#) >

[Books and Resources](#)

[Security](#)

COMMUNITY

[Community Overview](#)

Welcome to Apache Maven

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

If you think that Maven could help your project, you can find out more information in the "About Maven" section of the navigation. This includes an in-depth description of [what Maven is](#) and a [list of some of its main features](#).

This site is separated into the following sections, depending on how you'd like to use Maven:

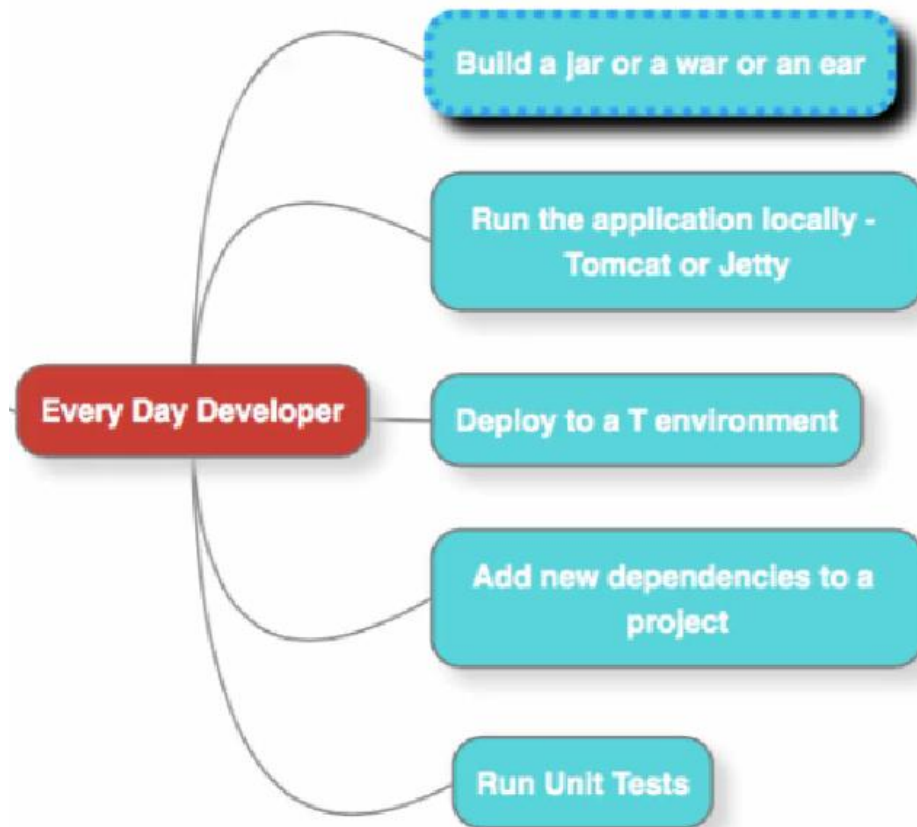
Use	Download, Install, Configure, Run Maven	Maven Plugins and Maven Extensions
	Information for those needing to build a project that uses Maven	Lists of plugins and extensions to help with your builds.
Extend	Write Maven Plugins	Improve the Maven Central Repository
	Information for developers writing Maven plugins.	Information for those who may or may not use Maven, but are interested in getting project metadata into the central repository .
Contribute	Help Maven	Develop Maven
	Information if you'd like to get involved. Maven is an open source community and welcomes contributions.	Information for those who are currently Maven developers, or who are interested in contributing to the Maven project itself.

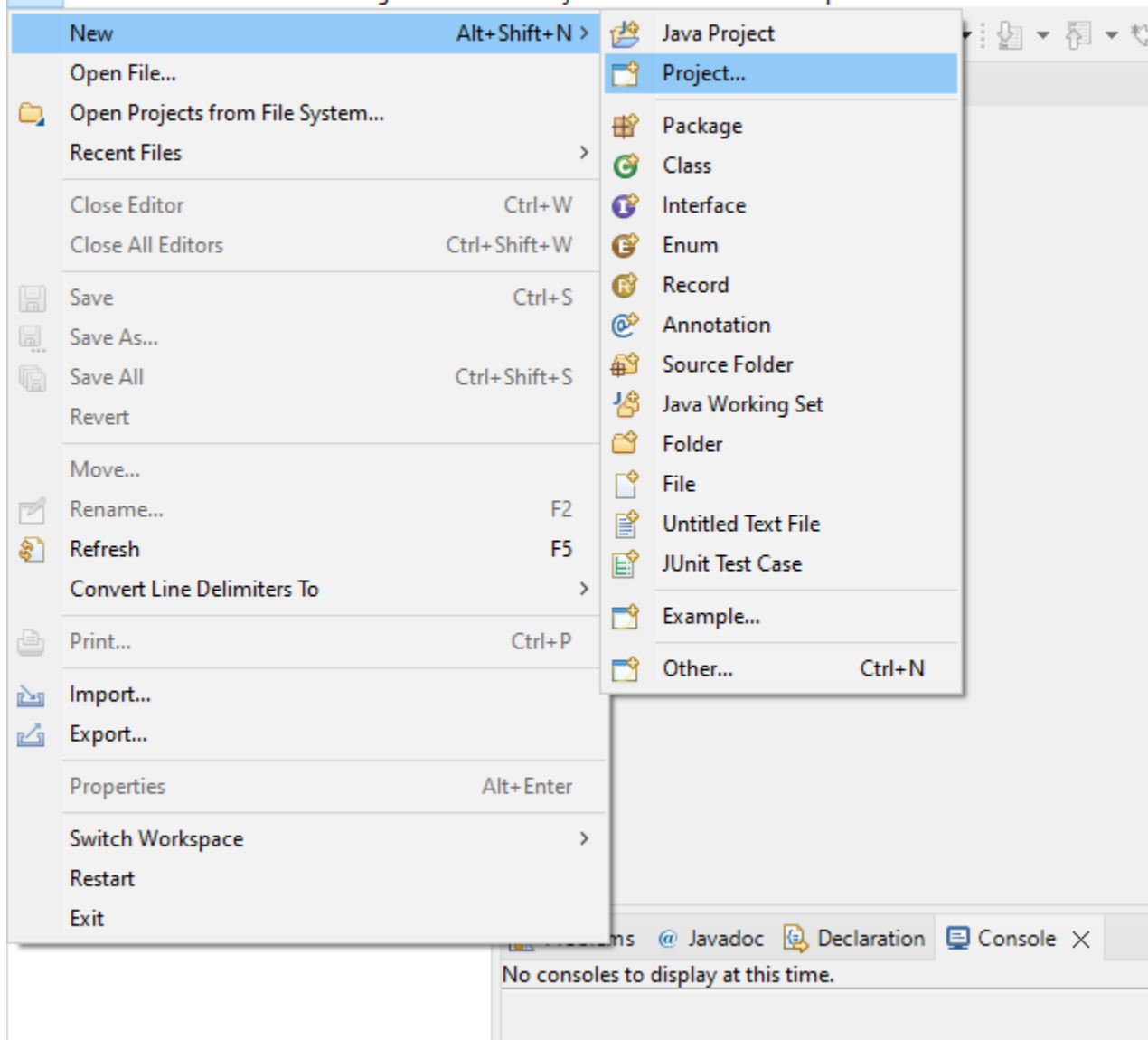
Each guide is divided into a number of trails to get you started on a particular topic, and includes a reference area and a "cookbook" of common examples.

You can access the guides at any time from the left navigation. If you are looking for a quick reference, you can use the [documentation index](#).

How to Get Support

Typical developer daily tasks on java project







Package Explorer

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

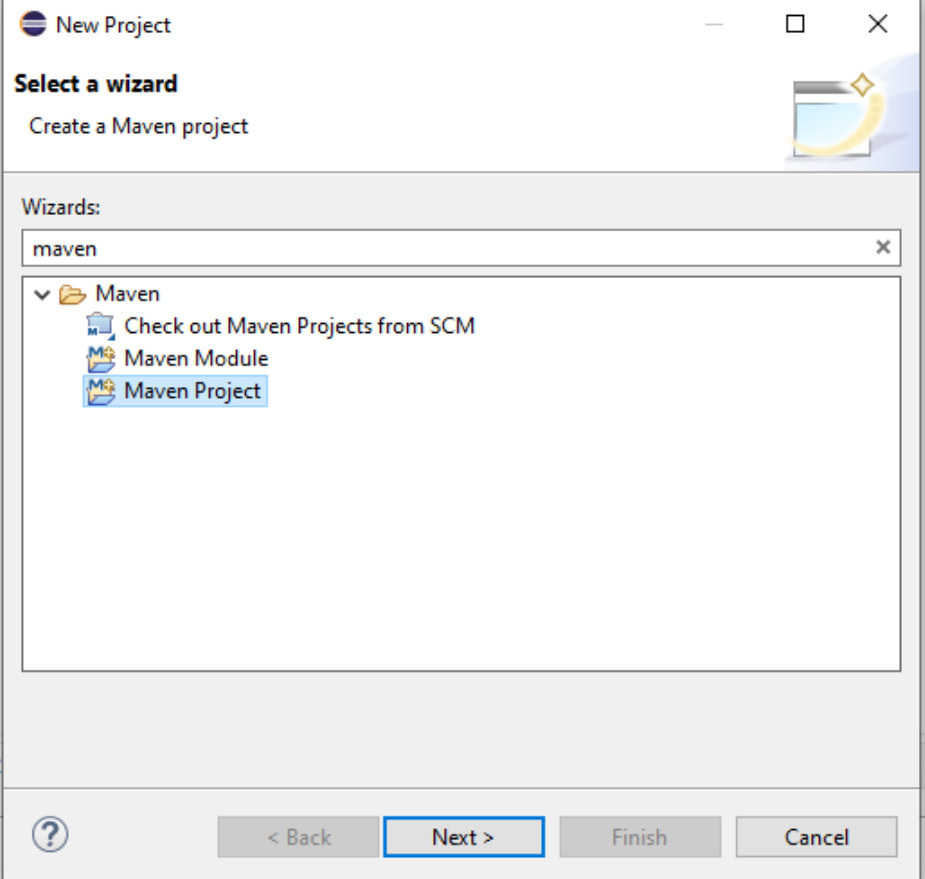
[Create a Java project](#)

[Create a project...](#)

[Import projects...](#)

Problems @ Javadoc Declaration

No consoles to display at this time.





Package Explorer

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

[Create a Java project](#)[Create a project...](#)[Import projects...](#)

Problems Javadoc Declaration

No consoles to display at this time.

New Maven Project

Select project name and location

☒ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location: [Browse...](#)

☐ Add project(s) to working set

Working set: [More...](#)

► Advanced

[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the project structure for 'MyMavenApp', including 'src/main/java', 'src/main/resources', 'src/test/java', 'src/test/resources', 'JRE System Library [J2SE-1.5]', 'src', 'target', and 'pom.xml'. An arrow points from 'pom.xml' in the Package Explorer to the right-hand editor. The editor displays the contents of 'pom.xml', which is an XML file defining a Maven project. The XML content is as follows:

```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>com.firstmavenapp</groupId>
5   <artifactId>MyMavenApp</artifactId>
6   <version>0.0.1-SNAPSHOT</version>
7   <name>firstMavenApplication</name>
8   <description>Demo maven application</description>
9 </project>
```

eclipse-workspace - MyMavenApp/pom.xml - Eclipse IDE

File Edit Source Refactor Source Navigate Search Project Run Window Help



Package Explorer x pom.xml x

MyMavenApp
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library
src
target
pom.xml

https://maven.apache.org/xsd/maven-4.0.0.x
1 <project xmlns="http://maven.apache.org/POM

New

- Open in New Window
- Open Type Hierarchy F4
- Show In Alt+Shift+W
- Show in Local Terminal
- Copy Ctrl+C
- Copy Qualified Name
- Paste Ctrl+V
- Delete Delete
- Build Path
- Source Alt+Shift+S
- Refactor Alt+Shift+T
- Import...
- Export...
- Refresh F5
- Assign Working Sets...
- Coverage As
- Run As
- Debug As
- Profile As
- Validate
- Restore from Local History...
- Team
- Compare With
- Configure
- Properties Alt+Enter

Java Project
Project...
Package
Class
Interface
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Example...
Other... Ctrl+N

Create a Java class

Declaration Console x
at this time.

eclipse-workspace - MyMavenApp/pom.xml - Eclipse IDE

File Edit Source Refactor Source Navigate Search Project Run Window Help

Package Explorer

MyMavenApp
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [J2SE-1.5]
src
target
pom.xml

```
1 <project xmlns="http://maven.apache.org/xsd/maven-4.0.0" >  
2 <modelVersion>4.0.0</modelVersion>  
3 <groupId>com.firstmavenapp</groupId>  
4 <artifactId>MyMavenApp</artifactId>  
5 <version>0.0.1-SNAPSHOT</version>  
6 <name>firstMavenApplication</name>  
7 <description>Demo maven application</description>  
8 </project>
```

Problems Javadoc Declaration

No consoles to display at this time.

New Java Class

Java Class

Create a new Java class.

Source folder: MyMavenApp/src/main/java Browse...

Package: com.demomavenapp Browse...

Enclosing type: Browse...

Name: MymavenApp

Modifiers: ☒ public ☐ package ☐ private ☐ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...

Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments



Finish

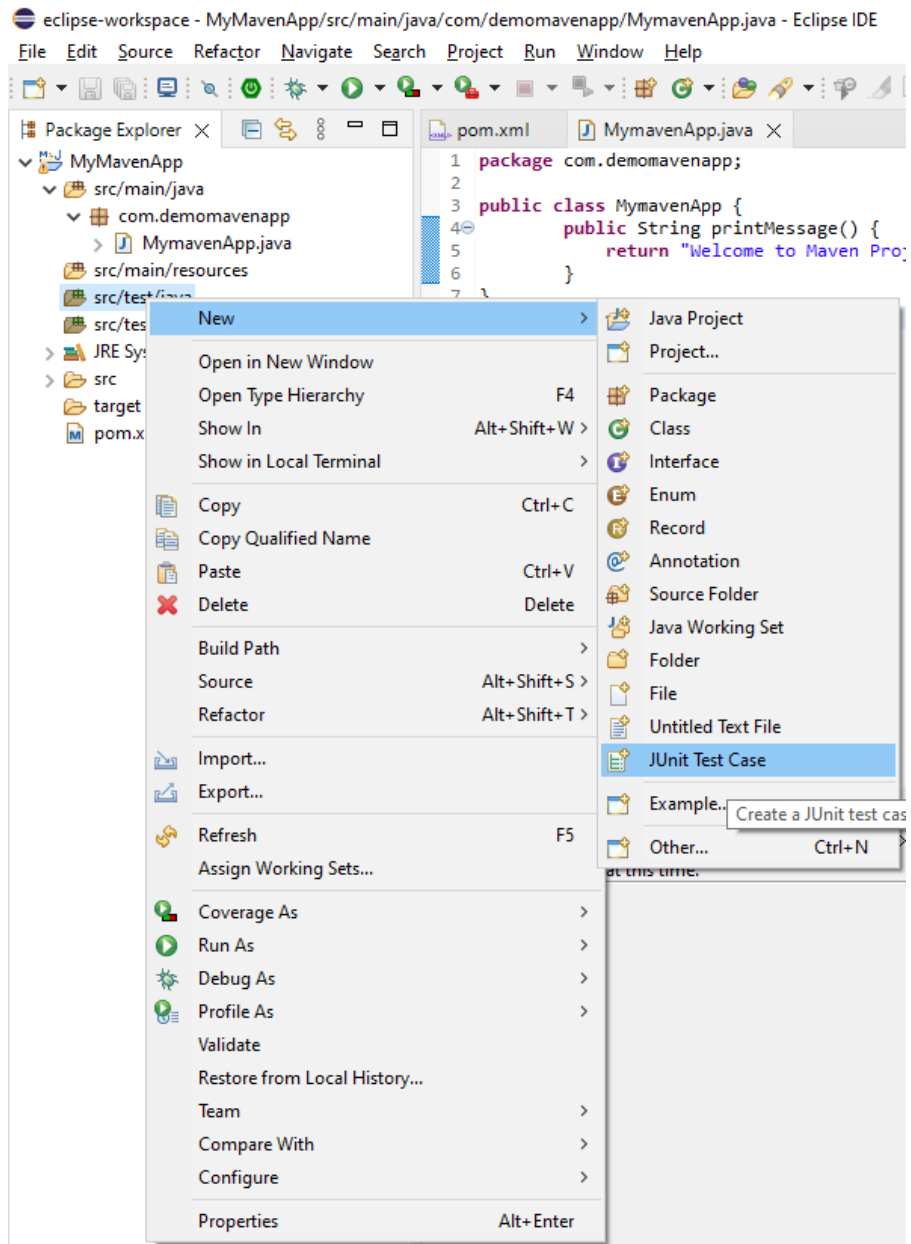
Cancel

eclipse-workspace - MyMavenApp/src/main/java/com/demomavenapp/MymavenApp.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer displays the project structure: MyMavenApp, src/main/java, com.demomavenapp, and MymavenApp.java. The main editor area shows the MymavenApp.java file with the following code:

```
1 package com.demomavenapp;
2
3 public class MymavenApp {
4     public String printMessage() {
5         return "Welcome to Maven Project";
6     }
7 }
8
```



New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. Specify the class under test to select methods to be tested on the next page.

☐ New JUnit 3 test

☒ New JUnit 4 test

☐ New JUnit Jupiter test

Source folder:

MyMavenApp/src/test/java

Browse...

Package:

com.demomavenapp

Browse...

Name:

MyMavaneTestApp

Superclass:

java.lang.Object

Browse...

Which method stubs would you like to create?

☐ @BeforeClass setUpBeforeClass()

☐ @AfterClass tearDownAfterClass()

☐ @Before setUp()

☐ @After tearDown()

☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Class under test:

Browse...

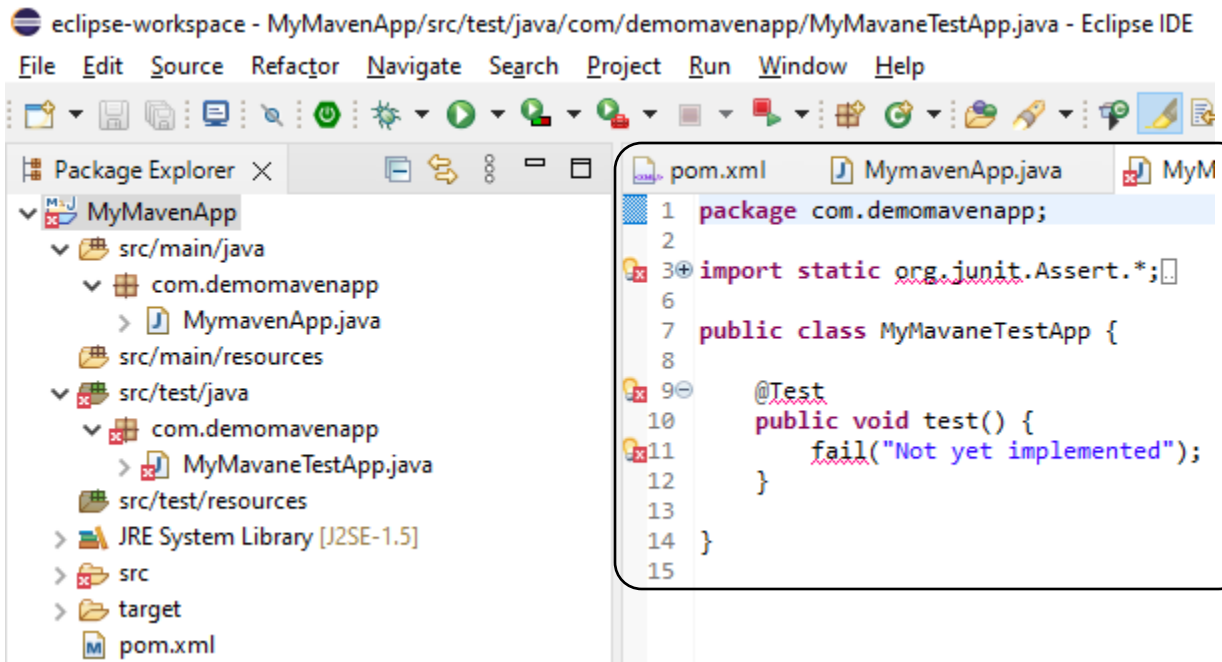
?

< Back

Next >

Finish

Cancel



junit dependency maven

Images

News

Videos

Maps

Shopping

About 22,10,000 results (0.35 seconds)

<https://mvnrepository.com> > artifact > junit > junit

JUnit - Maven Repository

JUnit. JUnit is a unit testing framework to write and run repeatable automated License ...

4.13.2

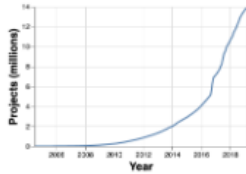
JUnit is a unit testing framework to write and run repeatable ...

4.12

MVN REPOSITORY

Search for groups, artifacts, categories

Indexed Artifacts (31.1M)



Popular Categories

- Testing Frameworks
- Android Packages
- Logging Frameworks
- Java Specifications
- JSON Libraries
- Core Utilities
- JVM Languages
- Mocking
- Language Runtime
- Web Assets
- Annotation Libraries
- Logging Bridges
- HTTP Clients
- Dependency Injection
- XML Processing
- Web Frameworks

Home » junit » junit » 4.13.2



JUnit » 4.13.2

JUnit is a unit testing framework to write and run repeatable automated tests on Java.

License	EPL 1.0
Categories	Testing Frameworks
Tags	testing junit
Organization	JUnit
HomePage	http://junit.org
Date	Feb 13, 2021
Files	jar (375 KB) View All
Repositories	Central HeavenArk Minebench Lutece Paris Xceptance
Ranking	#1 in MvnRepository (See Top Artifacts) #1 in Testing Frameworks
Used By	120,392 artifacts

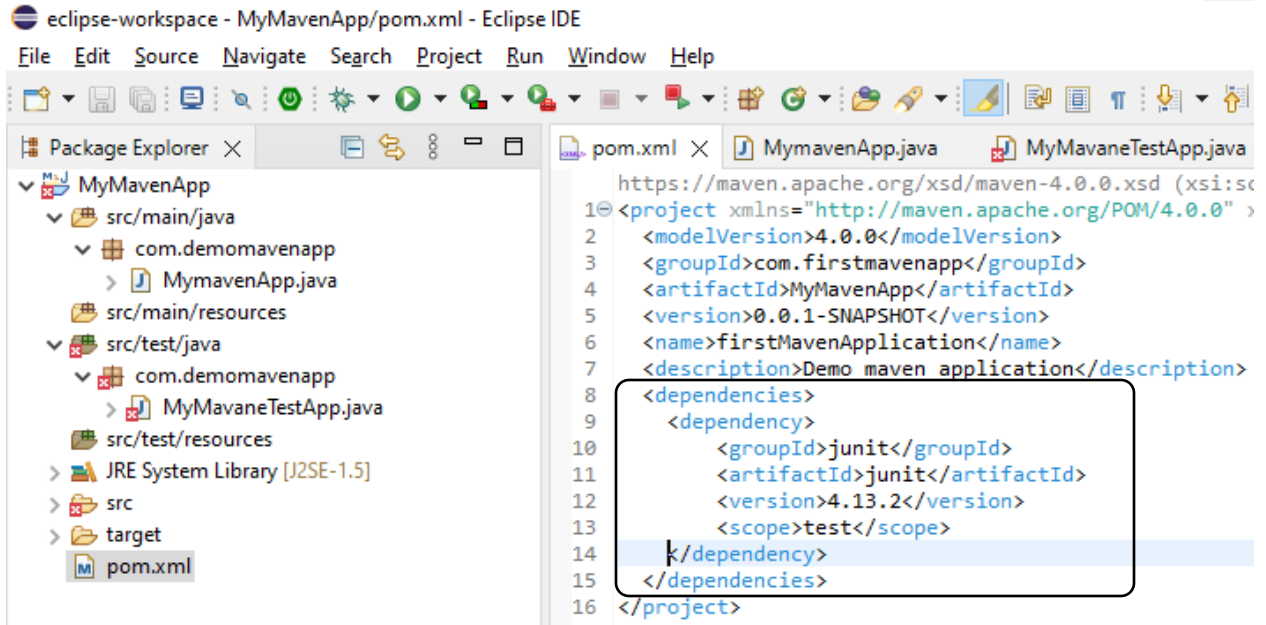
Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13.2</version>
  <scope>test</scope>
</dependency>
```

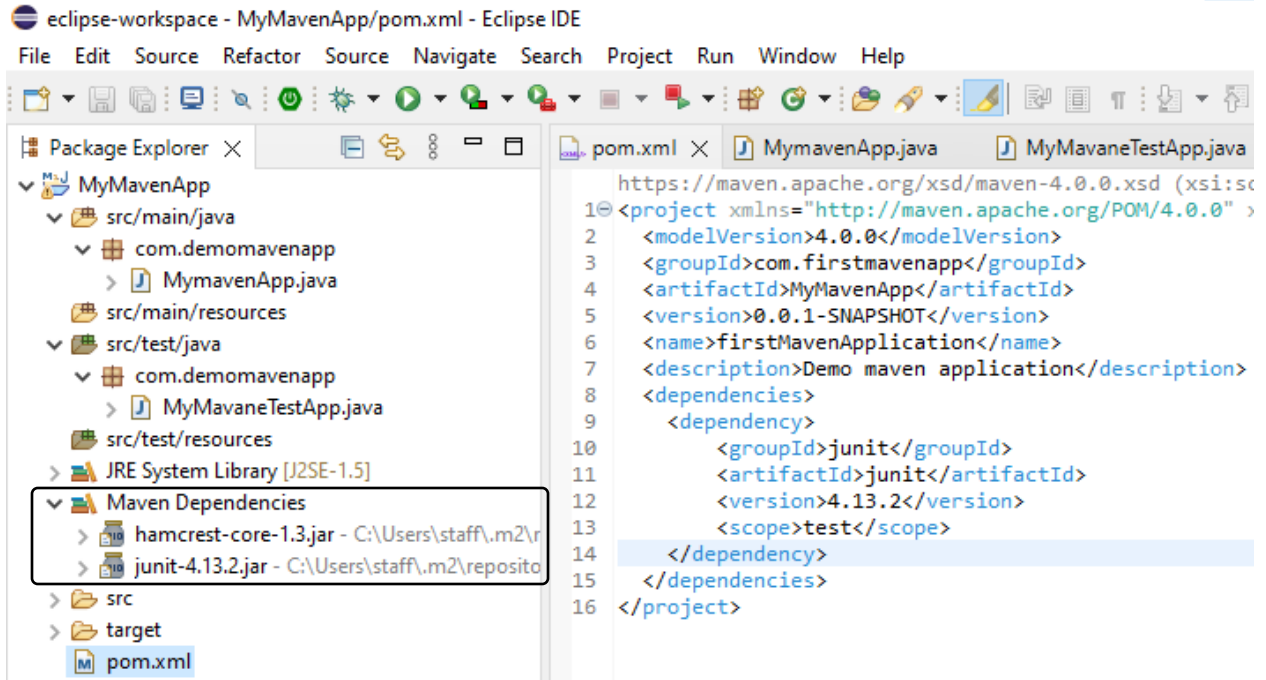
☒ Include comment with link to declaration

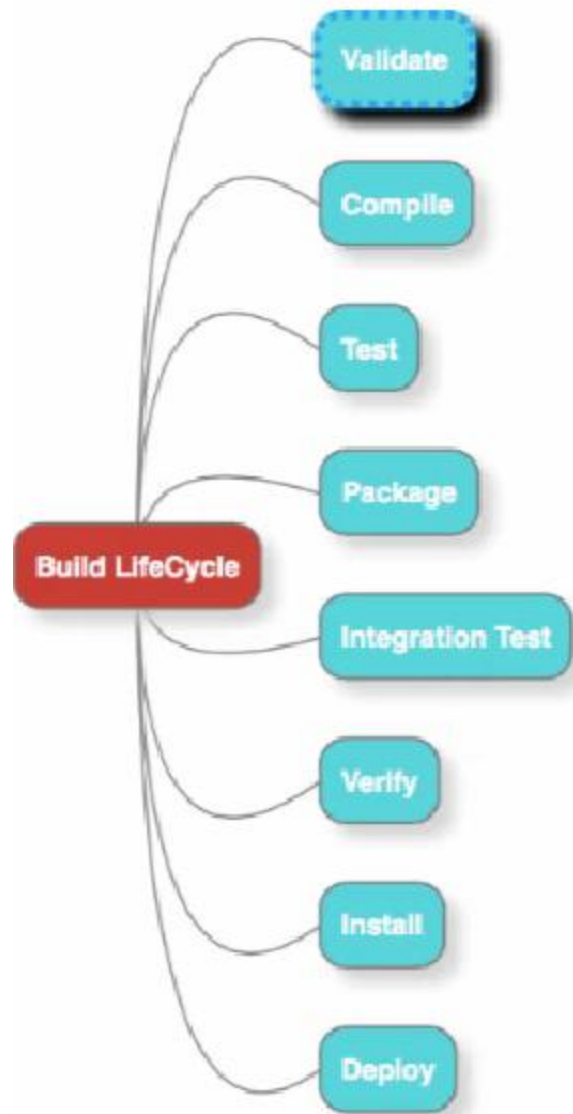
Copied to clipboard!

Copy dependency and paste in pom.xml

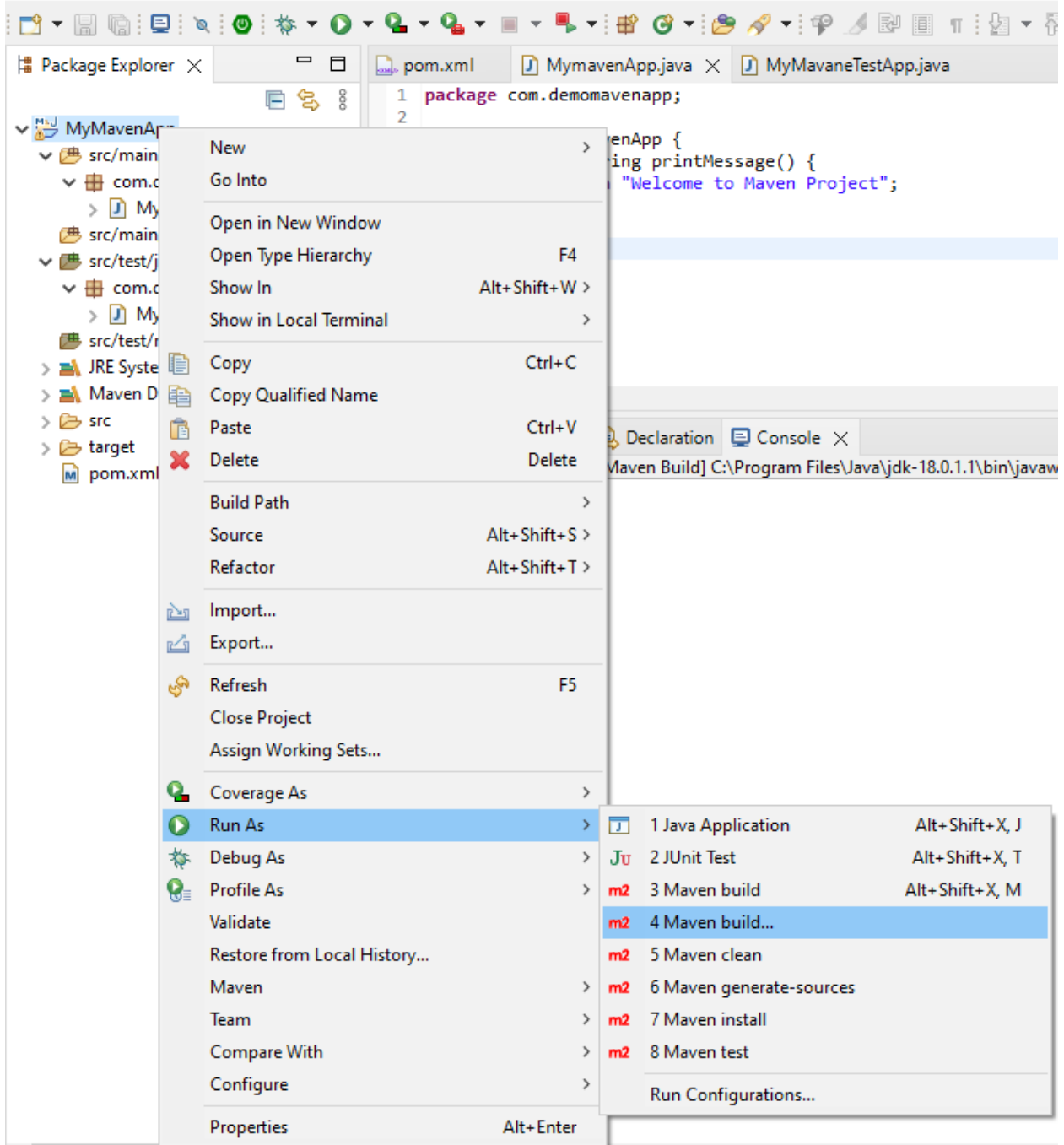


Just Save pom.xml file, after few seconds, Maven Dependencies adds Junit & related dependencies.





Apache Maven Build Life Cycle



Edit Configuration

Edit configuration and launch.

Name: MyMavenApp (1)

Main

JRE

Refresh

Source

Environment

Common

Base directory:

{project_loc:MyMavenApp}

Workspace...

File System...

Variables...

Goals:

clean

Profiles:

User settings:

C:\Users\staff\.m2\settings.xml

Workspace...

File System...

Variables...

☐ Offline

☐ Update Snapshots

☐ Debug Output

☐ Skip Tests

☐ Non-recursive

☐ Resolve Workspace artifacts

Threads:

1

Color Output:

Auto

Parameter Name	Value	
		<div>Add...</div>
		<div>Edit...</div>
		<div>Remove</div>

Revert

Apply

?

Run

Close

eclipse-workspace - MyMavenApp/src/main/java/com/demomavenapp/MymavenApp.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help



Package Explorer

- MyMavenApp
 - src/main/java
 - com.demomavenapp
 - MymavenApp.java
 - src/main/resources
 - src/test/java
 - com.demomavenapp
 - MyMavaneTestApp.java
 - src/test/resources
 - JRE System Library [J2SE-1.5]
 - Maven Dependencies
 - src
 - target
 - pom.xml

pom.xml MymavenApp.java MyMavaneTestApp.java

```
1 package com.demomavenapp;
2
3 public class MymavenApp {
4     public String printMessage() {
5         return "Welcome to Maven Project";
6     }
7 }
8
```

Problems Javadoc Declaration Console

<terminated> MyMavenApp (1) [Maven Build] C:\Program Files\Java\jdk-18.0.1\bin\javaw.exe (11-Dec-2022, 8:10:14 pm – 8:10:24 pm) [pid: 10348]

```
[INFO] Scanning for projects...
[INFO] -----< com.firstmavenapp:MyMavenApp >-----
[INFO] Building firstMavenApplication 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ MyMavenApp ---
[INFO] Deleting C:\Users\staff\eclipse-workspace\MyMavenApp\target
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 0.869 s
[INFO] Finished at: 2022-12-11T20:10:24+05:30
[INFO]
```

Edit configuration and launch.



Name: MyMavenApp (2)

Main JRE Refresh Source Environment Common

Base directory:

`${project_loc:MyMavenApp}`

Workspace...

File System...

Variables...

Goals: compile

Profiles:

User settings: C:\Users\staff\.m2\settings.xml

Workspace...

File System...

Variables...

☐ Offline☐ Update Snapshots☐ Debug Output☐ Skip Tests☐ Non-recursive☐ Resolve Workspace artifacts

Threads: 1

Color Output: Auto

Parameter Name

Value

Add...

Edit...

Remove

Revert

Apply



Run

Close

eclipse-workspace - MyMavenApp/src/main/java/com/demomavenapp/MyMavenApp.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- MyMavenApp
 - src/main/java
 - com.demomavenapp
 - MyMavenApp.java
 - src/main/resources
 - src/test/java
 - com.demomavenapp
 - MyMavenTestApp.java
 - src/test/resources
 - JRE System Library [J2SE-1.5]
 - Maven Dependencies
 - src
 - target
 - pom.xml

pom.xml

```
1 package com.demomavenapp;
2
3 public class MymavenApp {
4     public String printMessage() {
5         return "Welcome to Maven Project";
6     }
7 }
8
```

MyMavenApp.java

MyMavenTestApp.java

Problems Javadoc Declaration Console

<terminated> MyMavenApp (2) [Maven Build] C:\Program Files\Java\jdk-18.0.1\bin\javaw.exe (11-Dec-2022, 8:11:23 pm - 8:11:30 pm) [pid: 9456]

```
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ MyMavenApp ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file to C:\Users\staff\eclipse-workspace\MyMavenApp\target\classes
[INFO] -----
[ERROR] COMPILATION ERROR :
[INFO] -----
[ERROR] Source option 5 is no longer supported. Use 7 or later.
[ERROR] Target option 5 is no longer supported. Use 7 or later.
[INFO] 2 errors
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 5.249 s
[INFO] Finished at: 2022-12-11T20:11:30+05:30
[INFO] -----
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-compiler-plugin:3.1:compile (default-compile) on project MyMavenApp: Compilation failure: Compilation failure:
[ERROR] Source option 5 is no longer supported. Use 7 or later.
[ERROR] Target option 5 is no longer supported. Use 7 or later.
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
```

pom.xml × MymavenApp.java MyMavaneTestApp.java

```
6 <name>firstMavenApplication</name>
7 <description>Demo maven application</description>
8 <dependencies>
9   <dependency>
10     <groupId>junit</groupId>
11     <artifactId>junit</artifactId>
12     <version>4.13.2</version>
13     <scope>test</scope>
14   </dependency>
15 </dependencies>
16 <properties>
17   <maven.compiler.source>1.8</maven.compiler.source>
18   <maven.compiler.target>1.8</maven.compiler.target>
19 </properties>
20 </project>
```

eclipse-workspace - MyMavenApp/pom.xml - Eclipse IDE

File Edit Source Navigate Search Project Run Window Help

Package Explorer x

- MyMavenApp
 - src/main/java
 - com.demomavenapp
 - MymavenApp.java
 - src/main/resources
 - src/test/java
 - com.demomavenapp
 - MyMavaneTestApp.java
 - src/test/resources
 - JRE System Library [J2SE-1.5]
 - Maven Dependencies
 - src
 - target
 - pom.xml

pom.xml x MymavenApp.java MyMavaneTestApp.java

```
6 <name>firstMavenApplication</name>
7 <description>Demo maven application</description>
8 <dependencies>
9   <dependency>
10     <groupId>junit</groupId>
11     <artifactId>junit</artifactId>
12     <version>4.13.2</version>
13     <scope>test</scope>
14   </dependency>
15 </dependencies>
16 <properties>
17   <maven.compiler.source>1.8</maven.compiler.source>
18   <maven.compiler.target>1.8</maven.compiler.target>
19 </properties>
20 </project>
```

Problems @ Javadoc Declaration Console x

<terminated> MyMavenApp (3) [Maven Build] C:\Program Files\Java\jdk-18.0.1\bin\javaw.exe (11-Dec-2022, 8:13:44 pm – 8:13:50 pm) [pid: 11008]

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.firstmavenapp:MyMavenApp >-----
[INFO] Building firstMavenApplication 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ MyMavenApp ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ MyMavenApp ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file to C:\Users\staff\eclipse-workspace\MyMavenApp\target\classes
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 2.074 s
[INFO] Finished at: 2022-12-11T20:13:49+05:30
[INFO]
```

Edit configuration and launch.



Name:

☒ Main
 ☐ JRE
 ☐ Refresh
 ☐ Source
 ☐ Environment
 ☐ Common

Base directory:

Goals:

Profiles:

User settings:

☐ Offline
 ☐ Update Snapshots
 ☐ Debug Output
 ☐ Skip Tests
 ☐ Non-recursive
 ☐ Resolve Workspace artifacts

Threads: Color Output:

Parameter Name	Value

eclipse-workspace - MyMavenApp/src/main/java/com/demomavenapp/MymavenApp.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help



Package Explorer

MyMavenApp
src/main/java
com.demomavenapp
MymavenApp.java
src/main/resources
src/test/java
com.demomavenapp
MyMavaneTestApp.java
src/test/resources
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml

pom.xml MymavenApp.java MyMavaneTestApp.java

```
1 package com.demomavenapp;  
2  
3 public class MymavenApp {  
4     public static void main(String[] args) {  
5         System.out.println("Welcome to Maven Project");  
6     }  
7 }  
8
```

Problems Javadoc Declaration Console

```
<terminated> MyMavenApp (5) [Maven Build] C:\Program Files\Java\jdk-18.0.1\bin\javaw.exe (11-Dec-2022, 8:18:01 pm - 8:18:08 pm) [pid: 8404]  
[INFO] Scanning for projects...  
[INFO] -----< com.firstmavenapp:MyMavenApp >-----  
[INFO] Building firstMavenApplication 0.0.1-SNAPSHOT  
[INFO] -----[ jar ]-----  
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ MyMavenApp ---  
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!  
[INFO] Copying 0 resource  
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ MyMavenApp ---  
[INFO] Nothing to compile - all classes are up to date  
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ MyMavenApp ---  
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!  
[INFO] Copying 0 resource  
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ MyMavenApp ---  
[INFO] Changes detected - recompiling the module!  
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!  
[INFO] Compiling 1 source file to C:\Users\staff\eclipse-workspace\MyMavenApp\target\test-classes  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 5.216 s  
[INFO] Finished at: 2022-12-11T20:18:08+05:30  
[INFO] -----
```


Edit configuration and launch.



Name: MyMavenApp (6)

Main JRE Refresh Source Environment Common

Base directory:

\${project_loc:MyMavenApp}

Workspace...

File System...

Variables...

Goals: test

Profiles:

User settings: C:\Users\staff\.m2\settings.xml

Workspace...

File System...

Variables...

☐ Offline

☐ Update Snapshots

☐ Debug Output

☐ Skip Tests

☐ Non-recursive

☐ Resolve Workspace artifacts

Threads: 1

Color Output: Auto

Parameter Name	Value	
		Add...
		Edit...
		Remove

Revert

Apply

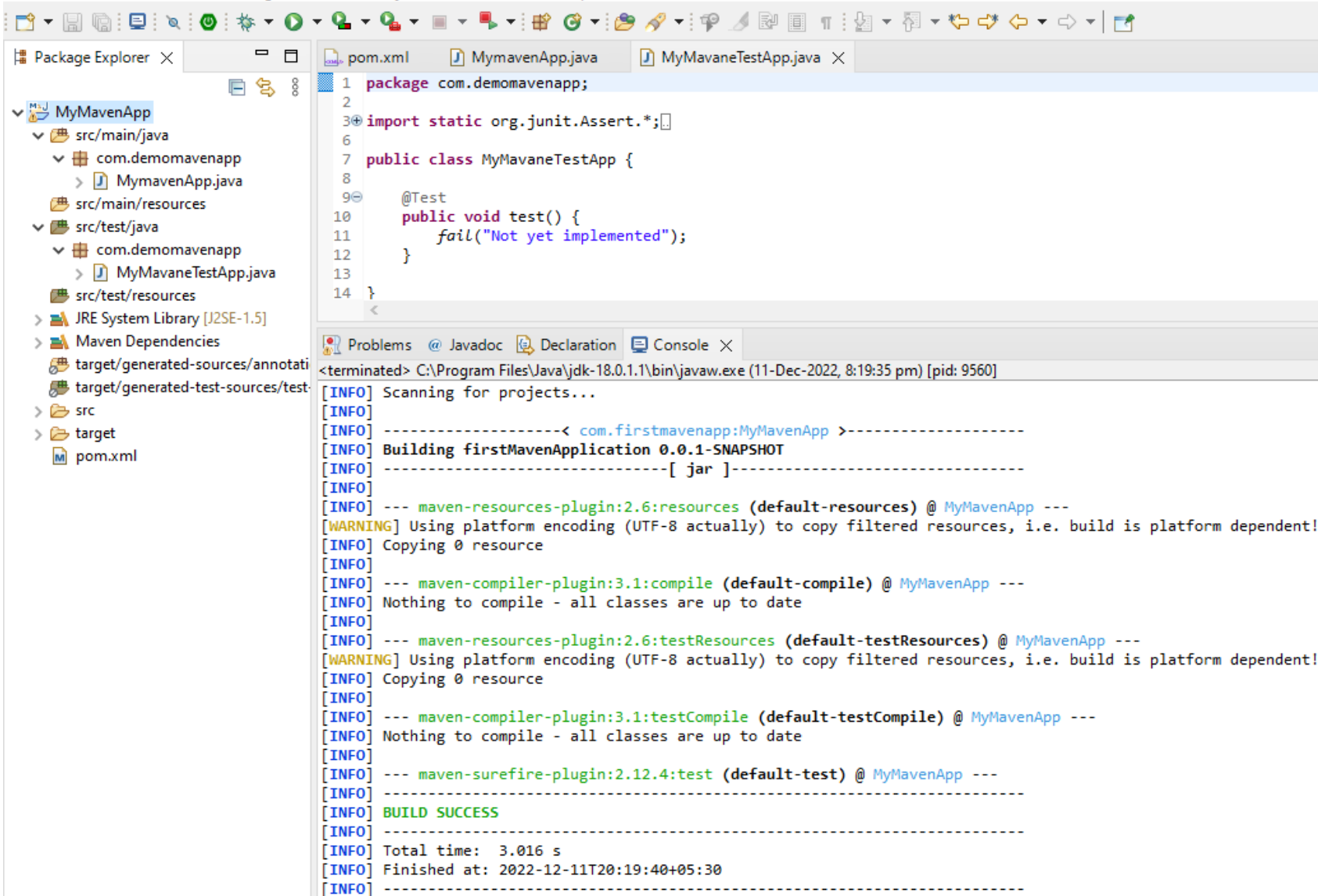


Run

Close

eclipse-workspace - MyMavenApp/src/test/java/com/demomavenapp/MyMavaneTestApp.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help



The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays the project structure: MyMavenApp, src/main/java, com.demomavenapp, MyMavenApp.java, src/main/resources, src/test/java, com.demomavenapp, MyMavaneTestApp.java, src/test/resources, JRE System Library [J2SE-1.5], Maven Dependencies, target/generated-sources/annotati, target/generated-test-sources/test, src, target, and pom.xml. The Editor on the right shows the MyMavaneTestApp.java file with the following code:

```
1 package com.demomavenapp;
2
3 import static org.junit.Assert.*;
4
5
6
7 public class MyMavaneTestApp {
8
9     @Test
10     public void test() {
11         fail("Not yet implemented");
12     }
13 }
14
```

The Console at the bottom shows the build output:

```
<terminated> C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe (11-Dec-2022, 8:19:35 pm) [pid: 9560]
[INFO] Scanning for projects...
[INFO] -----< com.firstmavenapp:MyMavenApp >-----
[INFO] Building firstMavenApplication 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ MyMavenApp ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ MyMavenApp ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ MyMavenApp ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ MyMavenApp ---
[INFO] Nothing to compile - all classes are up to date
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ MyMavenApp ---
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.016 s
[INFO] Finished at: 2022-12-11T20:19:40+05:30
[INFO] -----
```

Edit configuration and launch.



Name:

☒ Main
 ☐ JRE
 ☐ Refresh
 ☐ Source
 ☐ Environment
 ☐ Common

Base directory:

Goals:

Profiles:

User settings:

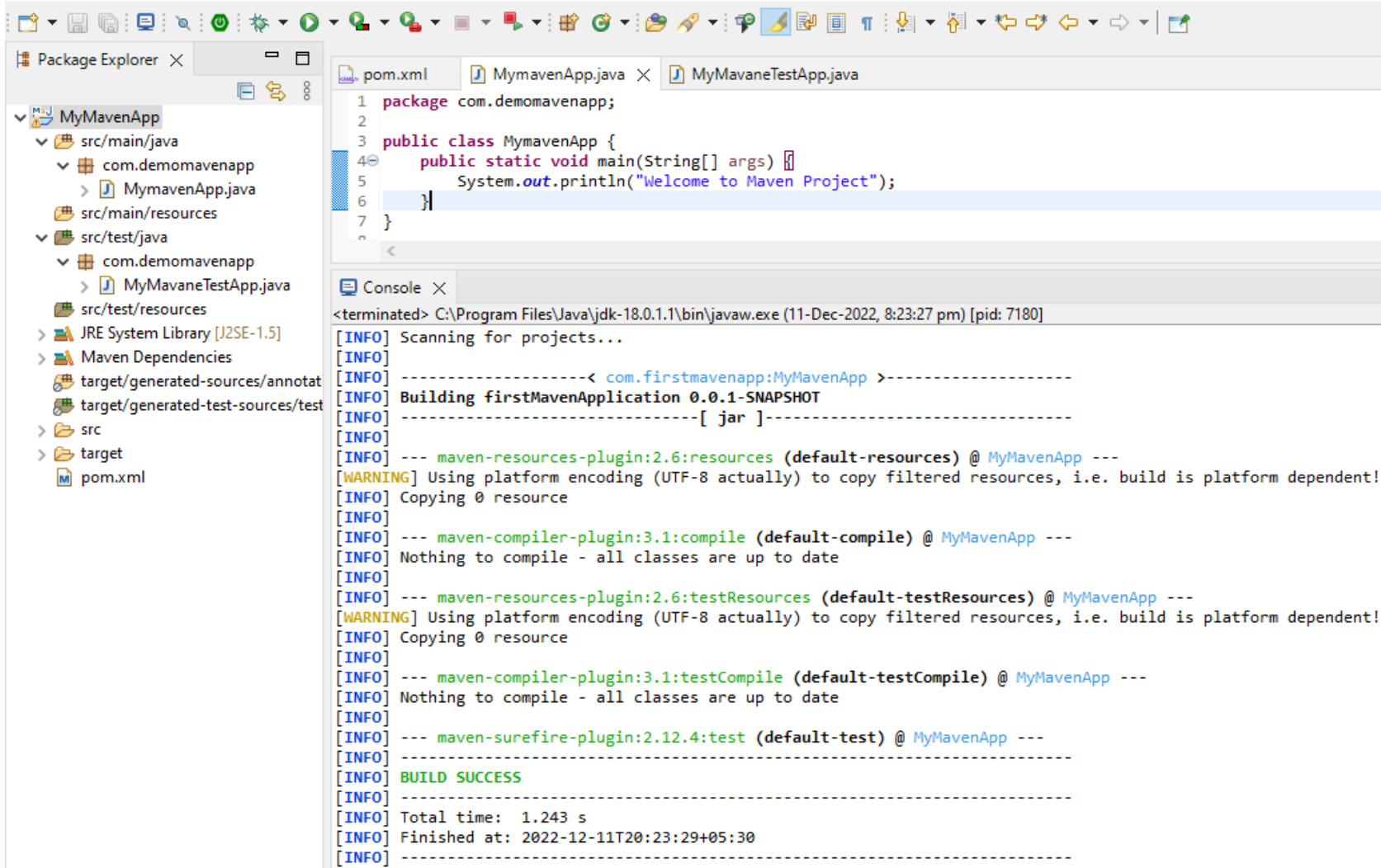
☐ Offline
 ☐ Update Snapshots
 ☐ Debug Output
 ☐ Skip Tests
 ☐ Non-recursive
 ☐ Resolve Workspace artifacts

Threads: Color Output:

Parameter Name	Value

eclipse-workspace - MyMavenApp/src/main/java/com/demomavenapp/MymavenApp.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help



The screenshot displays the Eclipse IDE interface for a Maven project named 'MyMavenApp'. The Package Explorer on the left shows the project structure, including the source code and resources. The Editor on the right shows the 'pom.xml' and 'MymavenApp.java' files. The Console at the bottom shows the build output, including warnings about platform encoding and a successful build message.

```
1 package com.demomavenapp;
2
3 public class MymavenApp {
4     public static void main(String[] args) {
5         System.out.println("Welcome to Maven Project");
6     }
7 }
```

<terminated> C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe (11-Dec-2022, 8:23:27 pm) [pid: 7180]

[INFO] Scanning for projects...

[INFO] -----< com.firstmavenapp:MyMavenApp >-----

[INFO] Building firstMavenApplication 0.0.1-SNAPSHOT

[INFO] -----[jar]-----

[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ MyMavenApp ---

[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ MyMavenApp ---

[INFO] Nothing to compile - all classes are up to date

[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ MyMavenApp ---

[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] Copying 0 resource

[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ MyMavenApp ---

[INFO] Nothing to compile - all classes are up to date

[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ MyMavenApp ---

[INFO] -----

[INFO] BUILD SUCCESS

[INFO] -----

[INFO] Total time: 1.243 s

[INFO] Finished at: 2022-12-11T20:23:29+05:30

[INFO] -----

Introduction to Spring Framework

Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.

Spring enables you to build applications from “plain old Java objects” (POJOs) and to apply enterprise services non-invasively to POJOs. This capability applies to the Java SE programming model and to full and partial Java EE.

Examples of how you, as an application developer, can use the Spring platform advantage:

- Make a Java method execute in a database transaction without having to deal with transaction APIs.
- Make a local Java method a remote procedure without having to deal with remote APIs.
- Make a local Java method a management operation without having to deal with JMX APIs.
- Make a local Java method a message handler without having to deal with JMS APIs.

What Is Spring?

The Spring framework provides comprehensive infrastructure support for developing Java applications.

It's packed with some nice features like Dependency Injection, and out of the box modules like:

- Spring JDBC
- Spring MVC
- Spring Security
- Spring AOP
- Spring ORM
- Spring Test

These modules can drastically reduce the development time of an application.

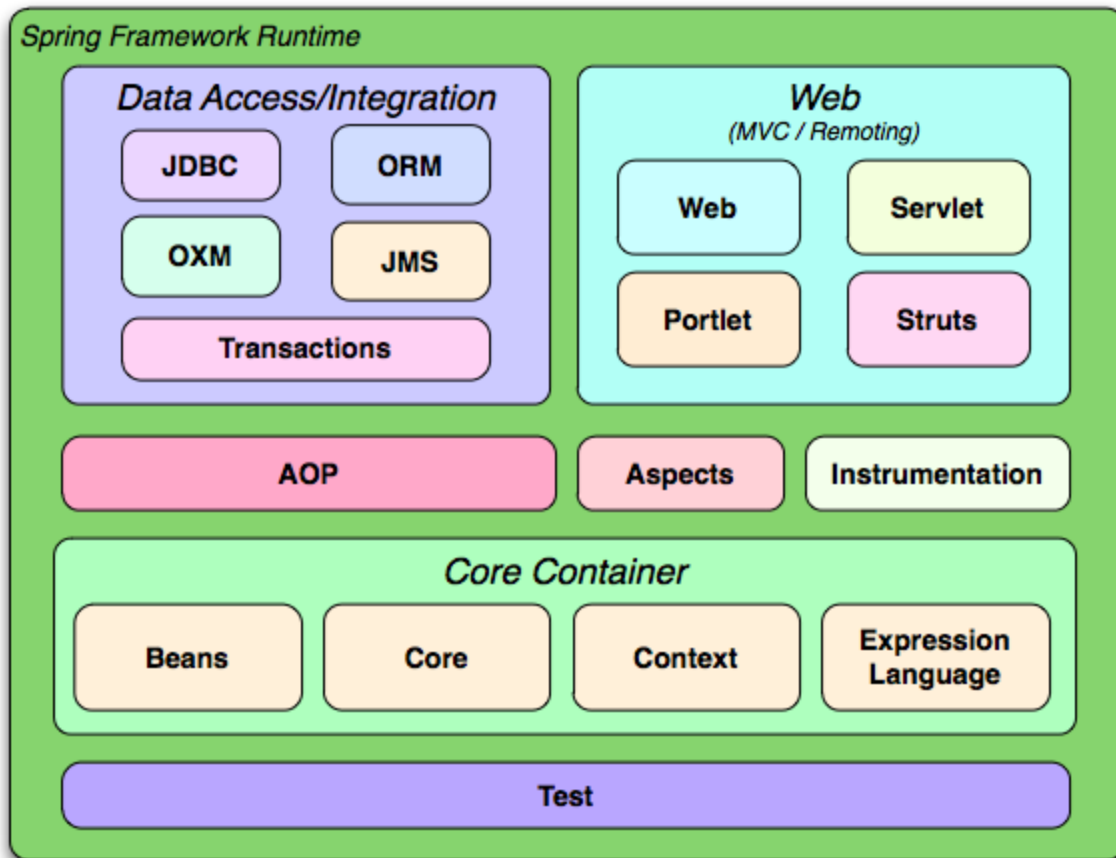
Why Spring Framework?

A **general purpose programming language like Java is capable of supporting a wide variety of applications**. Not to mention that Java is actively being worked upon and improving every day.

Moreover, there are countless open source and proprietary libraries to support Java in this regard. So why do we need a framework after all? Honestly, it isn't absolutely necessary to use a framework to accomplish a task. But, it's often advisable to use one for several reasons:

- Helps us **focus on the core task rather than non core IT** tasks associated with it
- Brings together years of wisdom in the form of design patterns
- Helps us adhere to the industry and regulatory standards
- Brings down the total cost of ownership for the application
- Forces us to **write an application in a specific manner**
- Binds to a specific version of language and libraries
- Adds to the resource footprint of the application

Spring Framework Architecture



1. Core Container

The Core Container in Spring Architecture contains the Core, Beans, Context, Expression Language modules.

- The **Core and Beans** modules provide the fundamental parts of the framework, including the IoC and Dependency Injection features. The **BeanFactory** is a sophisticated implementation of the factory pattern. It removes the need for programmatic singletons and allows you to decouple the configuration and specification of dependencies from your actual program logic.
- The **Context** module uses the base provided by the Core and Beans modules. It has access rights for any objects that define and configure. In Context module, the major point is **ApplicationContext** interface.
- A powerful expression language tool for querying & manipulation of object graph at runtime is given by **Expression Language Module**.

2. Data Access/Integration

The *Data Access/Integration* layer consists of the JDBC, ORM, OXM, JMS and Transaction modules.

- The **JDBC** module provides a JDBC-abstraction layer that removes the need to do tedious JDBC coding and parsing of database-vendor specific error codes.
- The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis. Using the ORM package you can use all of these O/R-mapping frameworks in combination with all of the other features Spring offers, such as the simple declarative transaction management feature mentioned previously.
- The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
- The Java Messaging Service (**JMS**) module contains features for producing and consuming messages.
- The Transaction module supports programmatic and declarative transaction management for classes that implement special interfaces and for *all your POJOs (plain old Java objects)*.

3. The Web Container

The *Web* layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules.

- Spring's *Web* module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context. It also contains the web-related parts of Spring's remoting support.
- The **Web-Servlet** module contains Spring's model-view-controller (*MVC*) implementation for web applications. Spring's MVC framework provides a clean separation between domain model code and web forms, and integrates with all the other features of the Spring Framework.
- The **Web-Struts** module contains the support classes for integrating a classic Struts web tier within a Spring application. Note that this support is now deprecated as of Spring 3.0.

Consider migrating your application to Struts 2.0 and its Spring integration or to a Spring MVC solution.

- The ***Web-Portlet*** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

4. AOP and Instrumentation

Spring's ***AOP*** module provides an *AOP Alliance*-compliant aspect-oriented programming implementation allowing you to define, for example, method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated. Using source-level metadata functionality, you can also incorporate behavioral information into your code, in a manner similar to that of .NET attributes.

The separate ***Aspects*** module provides integration with AspectJ.

The ***Instrumentation*** module provides class instrumentation support and classloader implementations to be used in certain application servers.

5. Test

The ***Test*** module supports the testing of Spring components with JUnit or TestNG. It provides consistent loading of Spring ApplicationContexts and caching of those contexts. It also provides mock objects that you can use to test your code in isolation.

Key Components of Spring Boot Framework

Spring Boot Framework has mainly four major Components.

1. Spring Boot Starters
2. Spring Boot AutoConfigurator
3. Spring Boot CLI
4. Spring Boot Actuator
5. Spring Boot Starter

Spring Boot Starters is one of the major key features or components of Spring Boot Framework. The main responsibility of Spring Boot Starter is to combine a group of common or related dependencies into single dependencies.

When we add “spring-boot-starter-web” jar file dependency to our build file, then Spring Boot Framework will automatically download all required jars and add to our project classpath.

Spring Boot AutoConfigurator

Traditional Spring Framework requires lots of configurations and for this it uses Either

- **XML Configuration or**
- **Annotation Configuration.**

Spring Boot Starter reduces build’s dependencies and Spring Boot AutoConfigurator reduces the Spring Configuration.

Spring Boot CLI

Spring Boot CLI (Command Line Interface) is Spring Boot software to run and test Spring Boot applications from command prompt.

When we run Spring Boot applications using CLI, then it internally uses Spring Boot Starter and Spring Boot AutoConfigure components to resolve all dependencies and execute the application.

Spring Boot Actuator

When we run our Spring Boot Web Application, Spring Boot Actuator automatically provides hostname as “localhost” and default port number as “8080”. We can access this application using “http://localhost:8080/” end point.

We actually use HTTP Request methods like GET and POST to represent Management Endpoints using Spring Boot Actuator.

In simple words it connects our application to some port and it is easily accessible we need not to deploy our application explicitly.

What is Spring Boot?

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

Why Spring Boot?

- It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
- It provides a powerful batch processing and manages REST endpoints.
- In Spring Boot, everything is auto configured; no manual configurations are needed.
- It offers annotation-based spring application
- Eases dependency management
- It includes Embedded Servlet Container

Features of spring Boot:

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

Compare Spring & Spring Boot

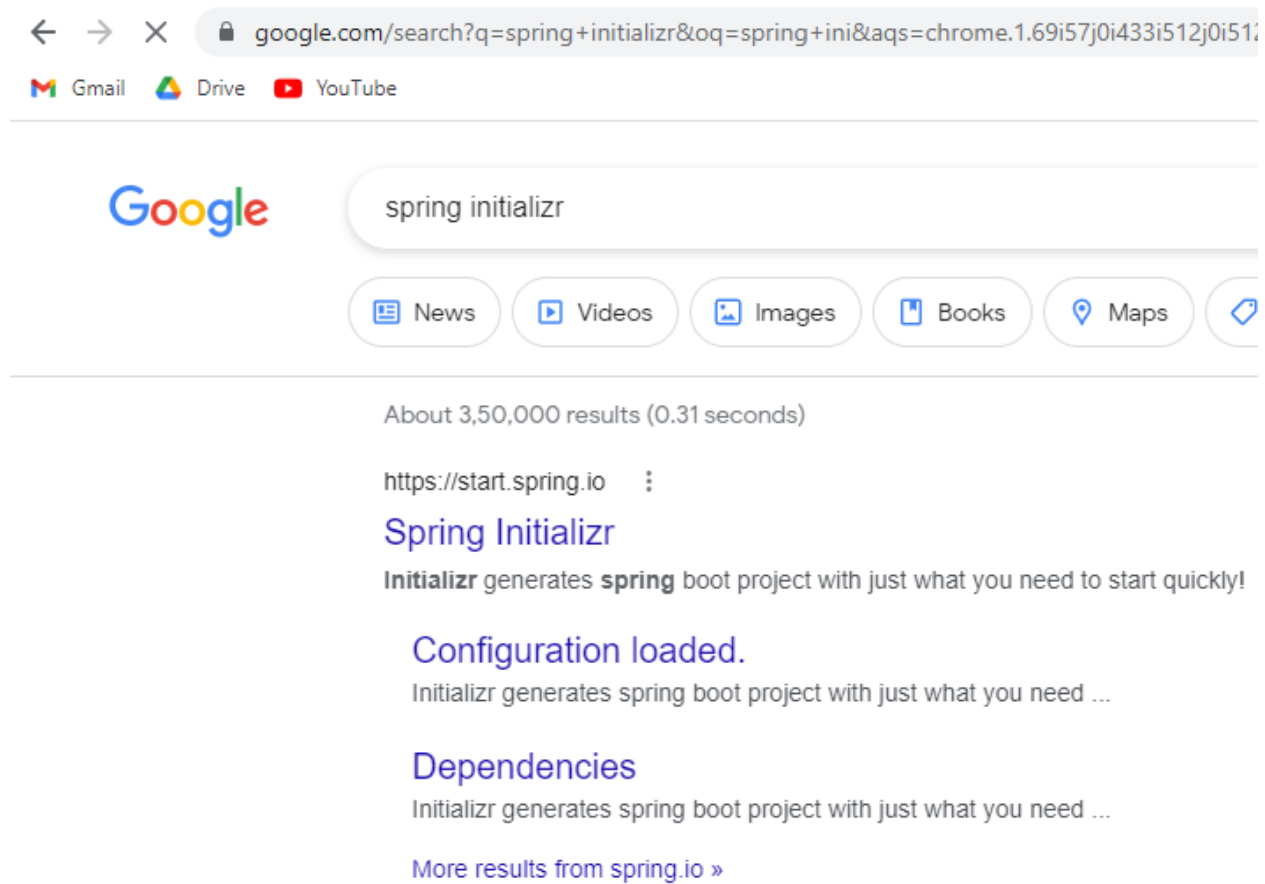
Sl no	Spring	Spring Boot
1	Widely used for building enterprise Java applications	Widely used for building REST APIs

2	Aims to simplify enterprise Java development	Aims to shorten code length and easily build web applications
3	Allows building loosely coupled applications	Allows building standalone applications
4	Main feature is dependency injection	Main feature is auto-configuration
5	Involves writing lots of boilerplate code	Reduces boilerplate code
6	Needs dependencies to be defined manually	Starters take care of dependencies
7	Involves setting up server manually	Includes embedded server like Tomcat and Jetty

Basis	Spring	Spring Boot
Where it's used?	Spring framework is a java EE framework that is used to build applications.	Spring Boot framework is mainly used to develop REST API's
Key feature	The primary or most important feature of the Spring framework is dependency injection(Dependency Injection (DI) is a design technique that removes dependencies from computer code, making the application easier to maintain and test).	The main or primary feature of the Spring Boot is Autoconfiguration(Simply described, Spring Boot autoconfiguration is a method of automatically configuring a Spring application based on the dependencies found on the classpath.) Autoconfiguration can speed up and simplify development by removing the need to define some beans that are part of the auto-configuration classes.
Why it's used	Its goal is to make Java EE (Enterprise Edition) development easier, allowing developers to be more productive.	Spring Boot provides the RAD(Rapid Application Development) feature to the Spring framework for faster application development.
Type of Application Development	Spring framework helps to create a loosely coupled application.	Spring Boot helps to create a stand-alone application.
Servers dependency	In the Spring framework to test the Spring Project, we need to set up the servers explicitly.	Spring Boot offers built-in or embedded servers such as Tomcat and jetty.
Deployment descriptor	To run a Spring application a deployment descriptor is required.	In Spring Boot there is no need for the Deployment descriptor.
In-memory database	Spring framework does not provide support for the in-memory database.	Spring Boot provides support for the in-memory database such as

support		H2.
Boilerplate code	Spring framework requires too many lines of code (boilerplate code) even for minimal tasks.	You avoid boilerplate code which reduces time and increases productivity.
Configurations	In the Spring framework, you have to build configurations manually.	In Spring Boot there are default configurations that allow faster bootstrapping.
Dependencies	Spring Framework requires a number of dependencies to create a web app.	Spring Boot, on the other hand, can get an application working with just one dependency. There are several more dependencies required during build time that is added to the final archive by default.
HTTP Authentication	HTTP Basic Authentication is for enabling security confirmations, it indicates that several dependencies and configurations need to be enabled to enable security. Spring requires both the standard spring-security-web and spring-security-config dependencies to set up security in an application. Next, we need to add a class that extends the WebSecurityConfigurerAdapter and makes use of the @EnableWebSecurity annotation.	Spring Boot also requires these dependencies to make it work, but we only need to define the dependency of spring-boot-starter-security as this will automatically add all the relevant dependencies to the classpath.
Testing	Testing in Spring Boot is difficult in comparison to Spring Boot due to a large amount of source code.	Testing in Spring Boot is easier due to the reduced amount of source code.
XML Configuration	In the Spring framework, XML Configuration is required.	No need for XML configuration in Spring Boot.
CLI Tools	Spring framework does not provide any CLI tool for developing and testing applications.	Spring Boot provides a CLI tool for developing and testing Spring Boot applications.
Plugins	Spring framework does not provide any plugin for maven, Gradle, etc. like Spring Boot.	Spring Boot provides build tool plugins for Maven and Gradle. The Plugins offer a variety of features, including the packaging of executable jars.

Understanding the spring initializer interface:





Project

☒ Gradle - Groovy ☐ Gradle - Kotlin ☐ Maven

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.0.1 (SNAPSHOT) ☒ 3.0.0 ☐ 2.7.7 (SNAPSHOT) ☐ 2.7.6

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 19 ☒ 17 ☐ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

No dependency selected



GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

DEVELOPER TOOLS

GraalVM Native Support

Support for compiling Spring applications to native executables using the GraalVM native-image compiler.



Spring Boot DevTools

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok

Java annotation library which helps to reduce boilerplate code.

Spring Configuration Processor

Generate metadata for developers to offer contextual help and "code completion" when working with custom configuration keys (ex.application.properties/.yml files).

WEB

Spring Web

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Reactive Web

Build reactive web applications with Spring WebFlux and Netty.

Spring for GraphQL

Build GraphQL applications with Spring for GraphQL and GraphQL Java.

Rest Repositories



Project

☐ Gradle - Groovy
 ☐ Gradle - Kotlin
 ☒ Java
 ☐ Kotlin
 ☐ Groovy

☒ Maven

Spring Boot

☐ 3.0.1 (SNAPSHOT)
 ☒ 3.0.0
 ☐ 2.7.7 (SNAPSHOT)
 ☐ 2.7.6

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 19 ☒ 17 ☐ 11 ☐ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

OAuth2 Client SECURITY

Spring Boot integration for Spring Security's OAuth2/OpenID Connect client features.

Spring Data MongoDB NOSQL

Store data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.

Testcontainers TESTING

Provide lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a Docker container.



GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

Downloads

File

Home

Share

View

This PC > Downloads >

Downloads

Documents

Pictures

789notes

converted

Name

Date modified

Type

Size

Today (2)

demo.zip

11-12-2022 10:13 PM

ZIP File

63 KB

demo

11-12-2022 10:15 PM

File folder

Yesterday (1)

demo

File

Home

Share

View

C:\Users\staff\Downloads\demo

Downloads

Documents

Pictures

789notes

converted

reduced

Videos

OneDrive

Name

Date modified

Type

Size

.mvn

11-12-2022 04:43 PM

File folder

src

11-12-2022 04:43 PM

File folder

.gitignore

11-12-2022 04:43 PM

Text Document

1 KB

HELP.md

11-12-2022 04:43 PM

Markdown Source...

2 KB

mvnw

11-12-2022 04:43 PM

File

11 KB

mvnw.cmd

11-12-2022 04:43 PM

Windows Comma...

7 KB

pom.xml

11-12-2022 04:43 PM

XML Document

3 KB

eclipse-workspace - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help



Package Explorer ×



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

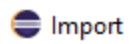
 [Create a Java project](#)

 [Create a project...](#)

 [Import projects...](#)

Console ×

No consoles to display at this time.



Import



Select

Import existing Maven projects



Select an import wizard:

- > General
- > Git
- > Gradle
- > Install
- ▼ Maven
 - Check out Maven Projects from SCM
 - Existing Maven Projects
 - Install or deploy an artifact to a Maven repository
 - Materialize Maven Binary Project
 - Materialize Maven Projects from SCM
- > Oomph
- > Run/Debug
- > Team
- > TextMate
- > XMI



< Back

Next >

Finish

Cancel

Maven Projects

Select Maven projects

Root Directory:

Projects:

☒ /pom.xml com.example:demo:0.0.1-SNAPSHOT:jar

Select All

Deselect All

Select Tree

Deselect Tree

Refresh

☐ Add project(s) to working set

► Advanced

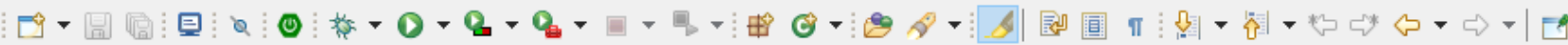


< Back

Next >

Finish

Cancel



Package Explorer

demo [boot]

- > src/main/java
- > src/main/resources
- > src/test/java
- > JRE System Library [JavaSE-17]
- > Maven Dependencies
- > src
 - target
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.0.0</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.example</groupId>
12  <artifactId>demo</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>demo</name>
15  <description>Demo project for Spring Boot</description>
16  <properties>
17    <java.version>17</java.version>
18    <testcontainers.version>1.17.6</testcontainers.version>
19  </properties>
20  <dependencies>
21    <dependency>
22      <groupId>org.springframework.boot</groupId>
23      <artifactId>spring-boot-starter-data-jpa</artifactId>
24    </dependency>
25    <dependency>
26      <groupId>org.springframework.boot</groupId>
27      <artifactId>spring-boot-starter-data-mongodb</artifactId>
28    </dependency>
29    <dependency>
30      <groupId>org.springframework.boot</groupId>
31      <artifactId>spring-boot-starter-oauth2-client</artifactId>
32    </dependency>
33    <dependency>
34      <groupId>org.springframework.boot</groupId>
35      <artifactId>spring-boot-starter-web</artifactId>
36    </dependency>
37
38    <dependency>
39      <groupId>org.springframework.boot</groupId>
40      <artifactId>spring-boot-starter-test</artifactId>
41      <scope>test</scope>
42    </dependency>
43    <dependency>
44      <groupId>org.testcontainers</groupId>
45      <artifactId>junit-jupiter</artifactId>
46      <scope>test</scope>
47    </dependency>
48  </dependencies>
```

eclipse-workspace - demo/src/main/java/com/example/demo/DemoApplication.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer X Run As... DemoApplication.java X

demo [boot]

- src/main/java
 - com.example.demo
 - DemoApplication.java
 - src/main/resources
 - src/test/java
 - JRE System Library [JavaSE-17]
 - Maven Dependencies
 - src
 - target
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

```
1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class DemoApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(DemoApplication.class, args);
11     }
12
13 }
14
```

eclipse-workspace - Eclipse IDE

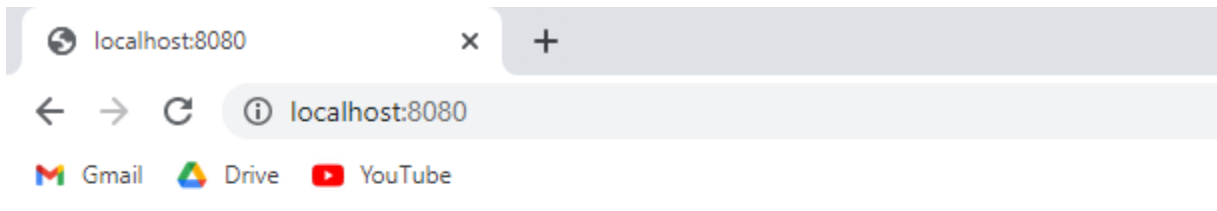
File Edit Navigate Search Project Run Window Help

Package Explorer X Console X pom.xml DemoApplication.java

New configuration [Java Application] C:\Program Files\Java\jdk-18.0.1\bin\javaw.exe (11-Dec-2022, 10:40:33 pm) [pid: 3024]

Spring Boot (v3.0.0)

```
2022-12-11T22:40:36.010+05:30 INFO 3024 --- [main] com.example.demo.DemoApplication : Starting DemoApplication using Java 18.0.1.1 with PID 3024 (C:\Users\staff\Downloads\
2022-12-11T22:40:36.013+05:30 INFO 3024 --- [main] com.example.demo.DemoApplication : No active profile set, falling back to 1 default profile: "default"
2022-12-11T22:40:37.478+05:30 INFO 3024 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-12-11T22:40:37.494+05:30 INFO 3024 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-12-11T22:40:37.494+05:30 INFO 3024 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.1]
2022-12-11T22:40:37.628+05:30 INFO 3024 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-12-11T22:40:37.633+05:30 INFO 3024 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1462 ms
2022-12-11T22:40:38.101+05:30 INFO 3024 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-12-11T22:40:38.112+05:30 INFO 3024 --- [main] com.example.demo.DemoApplication : Started DemoApplication in 2.668 seconds (process running for 3.254)
```



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Dec 11 22:42:36 IST 2022

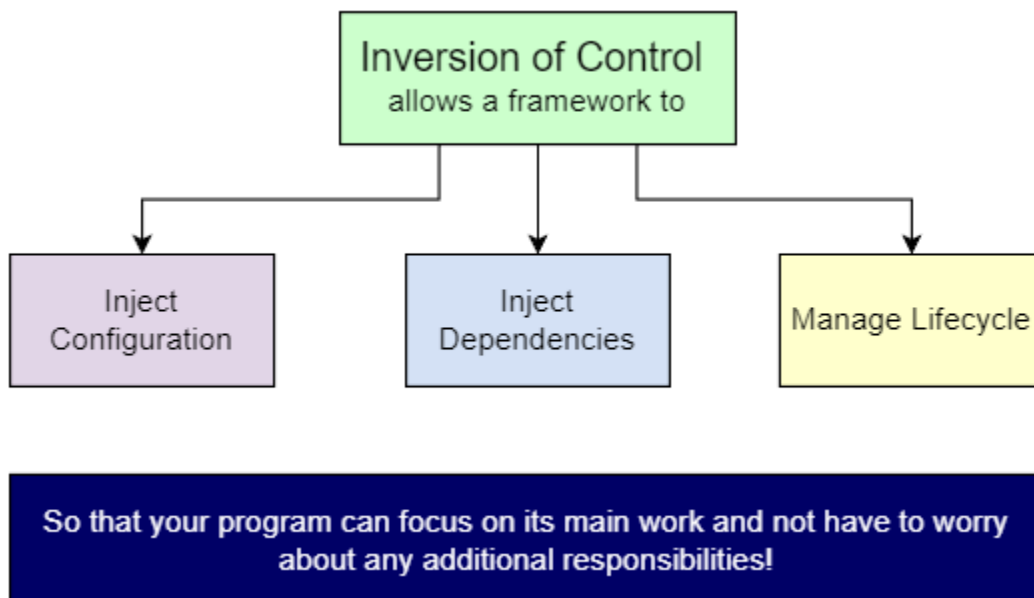
There was an unexpected error (type=Not Found, status=404).

Inversion of Control and Dependency Injection

What is inversion of control?

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. We most often use it in the context of object-oriented programming.

In contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code. To enable this, frameworks use abstractions with additional behavior built in. **If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes.**



The advantages of this architecture are:

- decoupling the execution of a task from its implementation
- making it easier to switch between different implementations
- greater modularity of a program
- greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts

We can achieve Inversion of Control through various mechanisms such as:

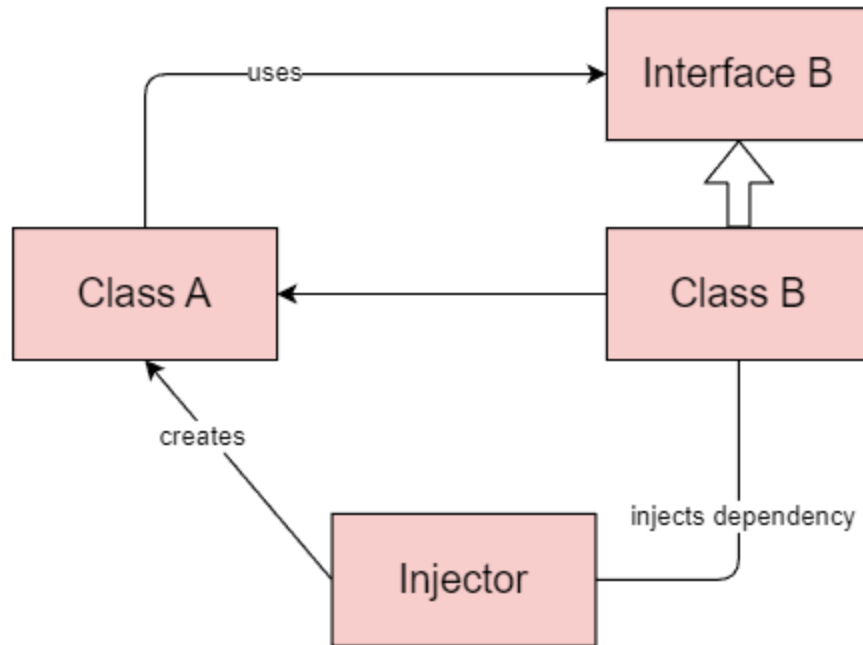
- Strategy design pattern
- Service Locator pattern
- Factory pattern
- Dependency Injection (DI)

What is dependency injection?

- **Dependency injection** is a pattern we can use to implement IoC, where the control being inverted is setting an object's dependencies.
- **Connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves.**
- **Dependency injection** is a technique that allows objects to be separated from the objects they depend upon.

For example, suppose object A requires a method of object B to complete its functionality. Well, dependency injection suggests that instead of creating an instance of class B in class A using the new operator, the object of class B should be injected in class A using one of the following methods:

- Constructor injection
- Setter injection
- Interface injection



Here's how we would create an object dependency in traditional programming:

```
public class Store {
    private Item item;
    public Store() {
        item = new ItemImpl1();
    }
}
```

In the example above, we need to instantiate an implementation of the *Item* interface within the *Store* class itself.

By using DI, we can rewrite the example without specifying the implementation of the *Item* that we want:

```
public class Store {
    private Item item;
    public Store(Item item) {
        this.item = item;
    }
}
```

The Spring IoC Container

An IoC container is a common characteristic of frameworks that implement IoC.

In the Spring framework, the interface *ApplicationContext* represents the IoC container. The Spring container is responsible for instantiating, configuring and assembling objects known as *beans*, as well as managing their life cycles.

The Spring framework provides several implementations of the *ApplicationContext* interface: *ClassPathXmlApplicationContext* and *FileSystemXmlApplicationContext* for standalone applications, and *WebApplicationContext* for web applications.

In order to assemble beans, the container uses configuration metadata, which can be in the form of XML configuration or annotations.

Here's one way to manually instantiate a container:

```
ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
```

To set the *item* attribute in the example above, we can use metadata. Then the container will read this metadata and use it to assemble beans at runtime.

Dependency Injection in Spring can be done through constructors, setters or fields.

1. Constructor-Based Dependency Injection

In the case of *constructor-based dependency injection*, the container will invoke a constructor with arguments each representing a dependency we want to set.

i. Annotations:

Spring resolves each argument primarily by type, followed by name of the attribute, and index for disambiguation. Let's see the configuration of a bean and its dependencies using **annotations**:

```
@Configuration
public class AppConfig {

    @Bean
    public Item item1() {
        return new ItemImpl1();
    }

    @Bean
    public Store store() {
        return new Store(item1());
    }
}
```

The *@Configuration* annotation indicates that the class is a source of bean definitions. We can also add it to multiple configuration classes.

We use the *@Bean* annotation on a method to define a bean. If we don't specify a custom name, then the bean name will default to the method name.

For a bean with the default *singleton* scope, Spring first checks if a cached instance of the bean already exists, and only creates a new one if it doesn't. If we're using the *prototype* scope, the container returns a new bean instance for each method call.

ii. XML configurations:

Another way to create the configuration of the beans is through XML configuration:

```
<bean id="item1" class="org.store.ItemImpl1" />
<bean id="store" class="org.store.Store">
    <constructor-arg type="ItemImpl1" index="0" name="item"
    ref="item1" />
</bean>
```

2. Setter-Based Dependency Injection

For setter-based DI, the container will call setter methods of our class after invoking a no-argument constructor or no-argument static factory method to instantiate the bean. Let's create this configuration using annotations:

```
@Bean
public Store store() {
    Store store = new Store();
    store.setItem(item1());
    return store;
}
```

We can also use XML for the same configuration of beans:

```
<bean id="store" class="org.store.Store">
    <property name="item" ref="item1" />
</bean>
```

We can combine constructor-based and setter-based types of injection for the same bean. The Spring documentation recommends using constructor-based injection for mandatory dependencies, and setter-based injection for optional ones.

3. Field-Based Dependency Injection

In case of Field-Based DI, we can inject the dependencies by marking them with an `@Autowired` annotation:

```
public class Store {
    @Autowired
    private Item item;
}
```

While constructing the *Store* object, if there's no constructor or setter method to inject the *Item* bean, the container will use reflection to inject *Item* into *Store*. We can also achieve this using [XML configuration](#).

Autowiring Dependencies

Wiring allows the Spring container to automatically resolve dependencies between collaborating beans by inspecting the beans that have been defined.

There are four modes of autowiring a bean using an XML configuration:

- **no**: the default value – this means no autowiring is used for the bean and we have to explicitly name the dependencies.
- **byName**: autowiring is done based on the name of the property, therefore Spring will look for a bean with the same name as the property that needs to be set.
- **byType**: similar to the *byName* autowiring, only based on the type of the property. This means Spring will look for a bean with the same type of the property to set. If there's more than one bean of that type, the framework throws an exception.
- **constructor**: autowiring is done based on constructor arguments, meaning Spring will look for beans with the same type as the constructor arguments.

For example, let's autowire the *item1* bean defined above by type into the *store* bean:

```
@Bean(autowire = Autowire.BY_TYPE)
public class Store {

    private Item item;

    public setItem(Item item) {
        this.item = item;
    }
}
```

We can also inject beans using the `@Autowired` annotation for autowiring by type:

```
public class Store {

    @Autowired
    private Item item;
}
```

@Qualifier annotation:

If there's more than one bean of the same type, we can use the **@Qualifier** annotation to reference a bean by name:

```
public class Store {

    @Autowired
    @Qualifier("item1")
    private Item item;
}
```

Now let's autowire beans by type through XML configuration:

```
<bean id="store" class="org.store.Store" autowire="byType">
</bean>
```

Next, let's inject a bean named *item* into the *item* property of *store* bean by name through XML:

```
<bean id="item" class="org.store.ItemImpl1" />

<bean id="store" class="org.store.Store" autowire="byName">
</bean>
```

We can also override the autowiring by defining dependencies explicitly through constructor arguments or setters.

Bean Scope(Object Scope):

The scope of a bean defines the life cycle and visibility of that bean in the contexts we use it.

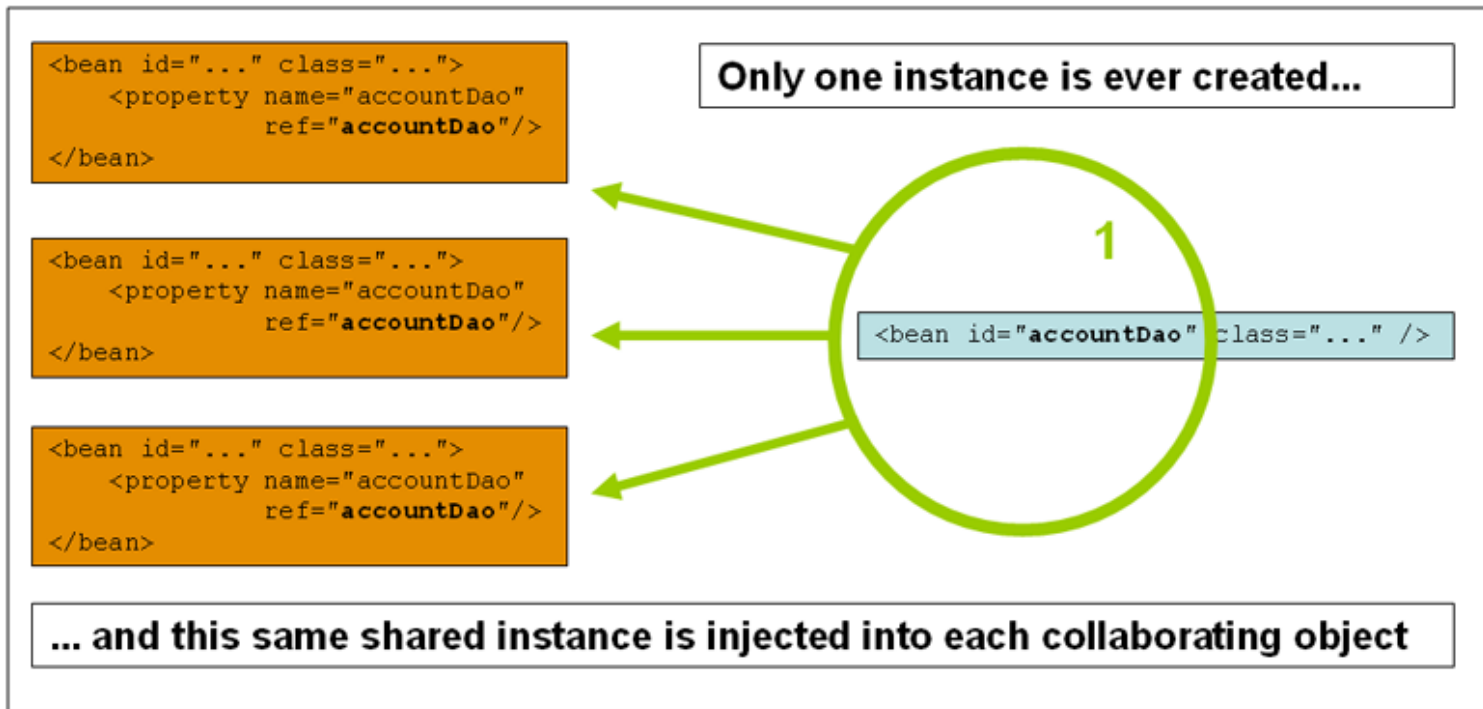
The latest version of the Spring framework defines 6 types of scopes:

- singleton
- prototype
- request
- session
- application
- websocket

The last four scopes mentioned, *request*, *session*, *application* and *websocket*, are only available in a web-aware application.

1.Singleton Scope

When we define a bean with the *singleton* scope, the container creates a single instance of that bean; all requests for that bean name will return the same object, which is cached. Any modifications to the object will be reflected in all references to the bean. This scope is the default value if no other scope is specified.



Here is the content of **HelloWorld.java** file –

Annotation Configuration:

```
@Bean
@Scope("singleton")
public class HelloWorld {
    // class elements, properties and methods..
}
```

Or XML config:

```
package com.HelloWorld;

public class HelloWorld {
    private String message;

    public void setMessage(String message){
        this.message = message;
    }
    public void getMessage(){
        System.out.println("Your Message : " + message);
    }
}
```

Following is the content of the MainApp.java file –

```
package com.HelloWorld;
```

```

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContex
t;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new
        ClassPathXmlApplicationContext("Beans.xml");
        HelloWorld objA = (HelloWorld)
        context.getBean("helloWorld");

        objA.setMessage("I'm object A");
        objA.getMessage();

        HelloWorld objB = (HelloWorld)
        context.getBean("helloWorld");
        objB.getMessage();
    }
}

```

Following is the configuration file Beans.xml required for singleton scope –

```

<?xml version = "1.0" encoding = "UTF-8"?>
<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
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
    <bean id = "helloWorld" class = "com.HelloWorld" scope =
    "singleton">
        </bean>

    </beans>

```

Once you are done creating the source and bean configuration files, let us run the application. If everything is fine with your application, it will print the following message –

```

Your Message : I'm object A
Your Message : I'm object A

```

2. Prototype Scope

If the scope is set to prototype, the Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made. As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans.

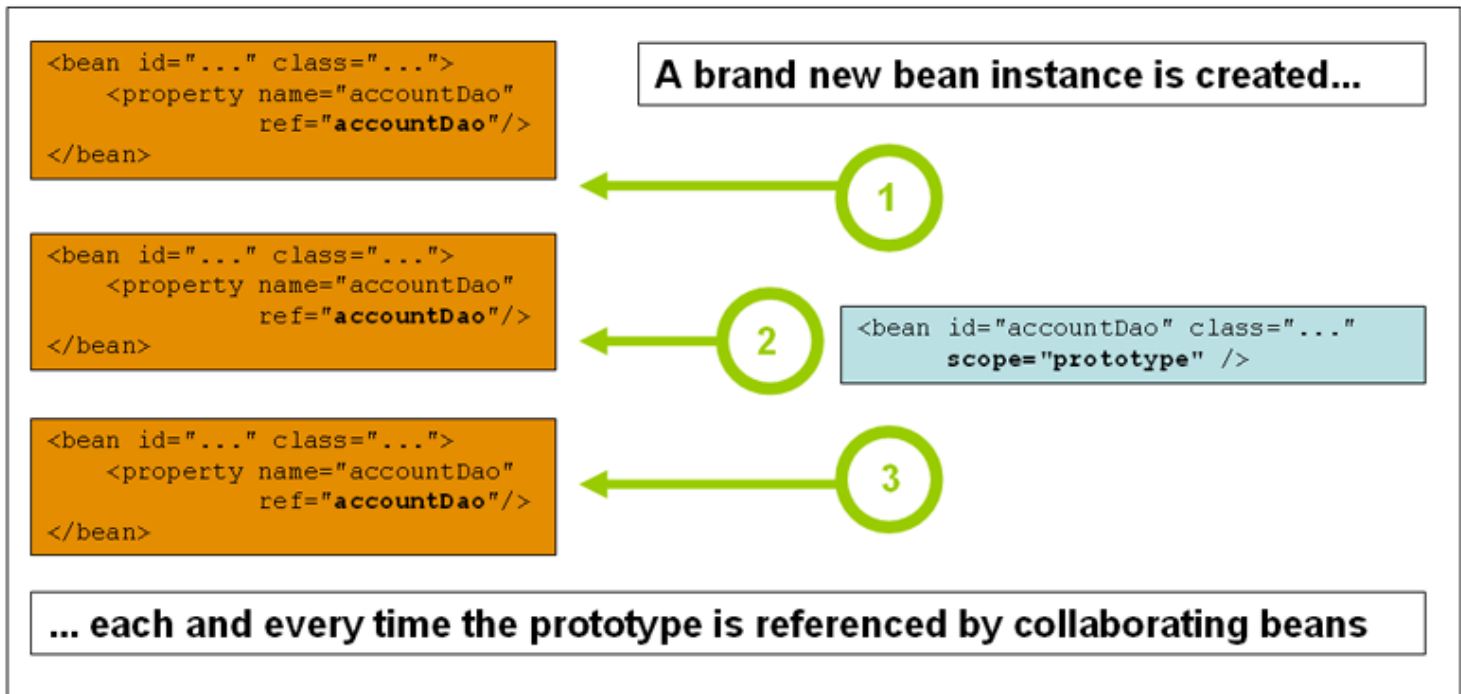
```

@Bean
@Scope("prototype")

```



```
public class HelloWorld {
    // class elements, properties and methods..
}
```



For the same HelloWorld.java, MainApp.java,

Following is the configuration file Beans.xml required for prototype scope –

```
<?xml version = "1.0" encoding = "UTF-8"?>

<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
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id = "helloWorld" class = "com.HelloWorld" scope =
        "prototype">
        </bean>
</beans>
```

Once you are done creating the source and bean configuration files, let us run the application. If everything is fine with your application, it will print the following message –

Your Message : I'm object A

Your Message : null