Beaconfire Inc, Home Work, Week3 Day12.
By (Ping) Nalongsone Danddank.
ndanddank@gmail.com
wechatID: ndanddank
Short Answer:

1. What is Maven and why it is used?

-> is a tool that help us to do some tasks like downloading dependencies, putting additional jars on a classpath, compiling source code into binary code, running tests, packaging compiled code into deployable artifacts such as JAR, WAR, and ZIP files, and deploying these artifacts to an application server or repository.

-> Because: Maven automates , minimizing the risk of humans making errors while building the software manually and separating the work of compiling and packaging our code from that of code construction.

2. What is POM? And what information does POM contain?

-> Project Object Model, The configuration of a Maven project is done via a Project Object Model (POM), represented by a pom.xml file.

-> The POM describes the project, manages dependencies, and configures plugins for building the software. The POM also defines the relationships among modules of multi-module projects.

3. What is Maven Repository? What is Maven Central Repository? What is local Maven Repository?

-> is a location, generally on a filesystem (either remote or local), where maven artifacts are stored and managed. Once artifacts have been stored in a maven repository, they are available for retrieval and inclusion in other maven projects.

-> Central repository system — project dependencies can be loaded from the local file system or public repositories.

-> is located in the .m2/ repository folder under the home directory of the user.

4. What is JAR and WAR? What are the differences?

-> While compressed files are typically given a .zip extension, the Java community instead uses the .ear extension for Java EE-based enterprise applications, .war for web applications, and .jar for stand-alone Java applications and linkable libraries.

-> A WAR file only requires a Java EE Web Profile-compliant application server to run. A JAR(Java Archive) file only requires a Java installation

Differences between WAR and EAR files

	WAR	EAR
Full name	Web Application Archive	Enterprise Application Archive
Supported API	Servlet and JSP API	Java EE API
Contains	Web-based resources such as images, HTML, property files and compiled Java code	Other Java EE archives such as WAR, RAR, EJB-JAR and JAR files
Target runtime	Java Web Profile-compliant application server such as Liberty or Tomcat	Java EE-compliant application server such as JBoss or WebSphere
Deployment descriptor name	web.xml	application.xml

5. What is Dependency Injection?

-> Separating the usage from the creation of the object. Make more easy to test.

Dependency injection is basically providing the objects that an object needs (its dependencies) instead of having it construct them itself. It's a very useful technique for testing, since it allows dependencies to be mocked or stubbed out.

6. What is the difference between BeanFactory and ApplicationContext in Spring?

(BeanFactory and ApplicationContext both are Java interfaces and ApplicationContext extends BeanFactory. Both of them are configuration using XML configuration files. In short BeanFactory provides basic Inversion of control(IoC) and Dependency Injection (DI) features while ApplicationContext provides advanced features.)

-> BeanFactory loads beans on-demand, while ApplicationContext loads all beans at startup. Thus, BeanFactory is lightweight as compared to ApplicationContext.

-> The BeanFactory interface provides an advanced configuration mechanism capable of managing any type of object. The ApplicationContext is a sub-interface of BeanFactory with additional features:

• Easier integration with Spring's AOP features

• Message resource handling (for use in internationalization)

• Application-layer specific contexts such as the WebApplicationContext for use in web applications.

7. How Can We Inject Beans in Spring?

-> Dependency Inject in Spring can be done in three ways:

• Constructor: In the case of constructor-based dependency injection, the container will invoke a constructor with arguments each representing a dependency we want to set . (For mandatory dependencies or when aiming for immutability, use constructor injection)

• Setter: Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or a noargument static factory method to instantiate your bean. (For optional or changeable dependencies, "mutability", use setter injection)

• Field: Through reflection, directly into fields (Avoid field injection in most cases)

8. Which Is the Best Way of Injecting Beans and Why?

-> Using constructor.

-> we can use constructor based dependency injection for mandatory dependencies and setter based injection for optional dependencies.

SETTER DI	CONSTRUCTOR DI
The bean must include getter and setter methods for the properties.	The bean class must declare a matching constructor with arguments. Otherwise, BeanCreationException will be thrown.
Circular dependencies or partial dependencies result with Setter DI because object creation happens before the injections.	No scope for circular or partial dependency because dependencies are resolved before object creation itself.
Preferred option when properties are less and mutable objects can be created.	Preferred option when properties on the bean are more and immutable objects (eg: financial processes) are important for application.

What is a circular dependency

-> Circular dependency in Spring happens when two or more beans require instance of each other through constructor dependency injections.

For example: There is a ClassA that requires an instance of ClassB through constructor injection and ClassB requires an instance of class A through constructor injection. In that sort of configuration where beans for both classes need to be injected into each other, Spring container can't decide which bean should be created first. The Spring IoC container will detect this circular reference at runtime while trying to inject dependencies and throw a BeanCurrentlyInCreationException.

9. What is the Scope of a Bean?

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

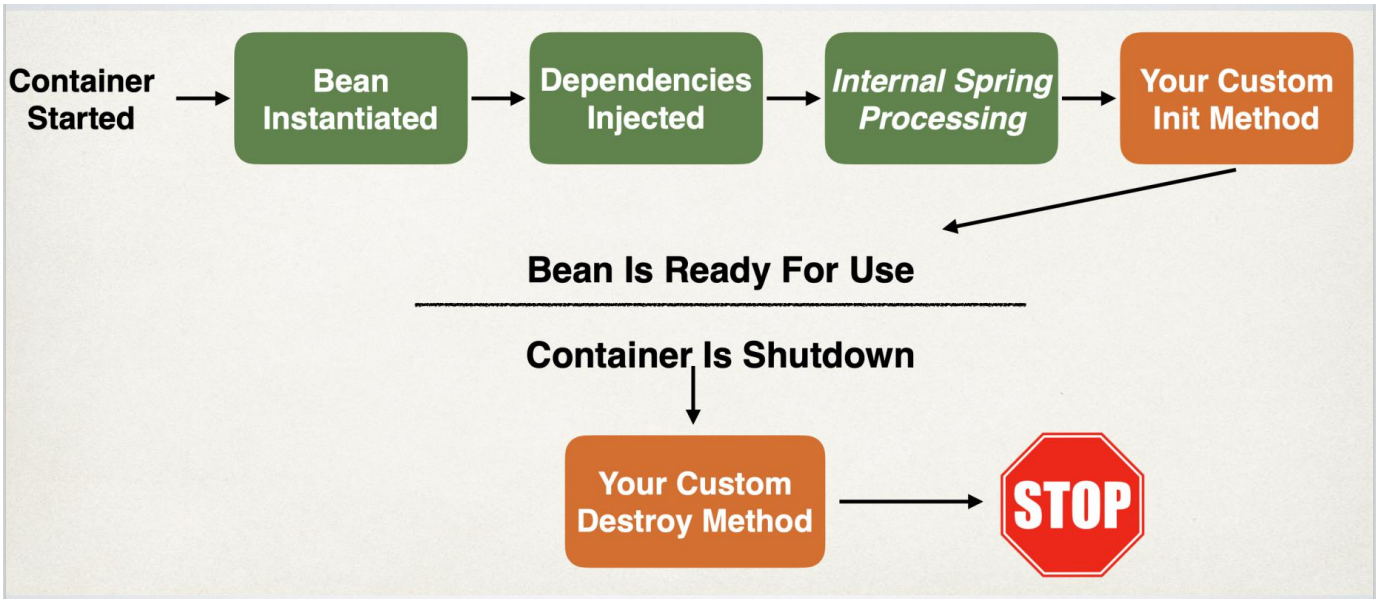
When defining a <bean> you have the option of declaring a scope for that bean. For example, to force Spring to produce a new bean instance each time one is needed, you should declare the bean's scope attribute to be prototype. Similarly, if you want Spring to return the same bean instance each time one is needed, you should declare the bean's scope attribute to be singleton.

10. How Does the Scope Prototype Work?

-> Scopes a single bean definition to any number of object instances. Or will return a different instance every time it is requested from the container.

11. What Does the Spring Bean Lifecycle Look Like?

->Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed.


Container Started
Bean Instantiated
Dependencies Injected
Internal Spring Processing
Your Custom Init Method
Bean Is Ready For Use
Container Is Shutdown
Your Custom Destroy Method
STOP