Spring framework: It is a very popular framework used to build enterprise Java applications. Initially, it was a simpler, lightweight alternative to J2EE. It provides a large number of helper classes to make things easier.

Founded by Rod Johnson in 2004.

Core Spring 4 features:

* Inversion of Control
* Dependency Injection
* Spring AOP
* Spring MVC

Goals of Spring:

* Lightweight development with Java POJOs (Plain old java objects)
* Dependency Injection to promote loose coupling
* Declarative programming with Aspect Oriented Programming (AOP).
* Minimize boilerplate Java code (using helper classes)

Core Container: Main item of Spring. Manages how beans are creating. Has factory for creating beans and also manages bean dependencies.

* Beans
* Core
* Spel
* Context

Data Access Layer:

* JDBC (Helper classes to reduce code by 50%)
* ORM (Object to Relational Mapping)
* Transactions (Makes heavy use of AOP behind the scenes)
* OXM
* JMS (Java Message Service. Helps send async messages to a message broker. Spring provides helper classes for JMS)

Web Layer: All web related classes.

* Servlet
* WebSocket
* Web
* Portlet

Infrastructure:

* AOP
* Aspects
* Instrumentation
* Messaging

Test Layer:

* Unit
* Integration
* Mock

Spring Projects: Additional spring modules built on top of core Spring Framework.

* Spring Cloud, Spring Data
* Spring Batch, Spring Security
* Spring For Android, Spring Webflow
* Spring Web Services, Spring LDAP

**Inversion of Control**:

The design process of outsourcing the construction and management of objects. The outsourcing will be handled by an object factory.

The client application uses the object factory provided by Spring based on a configuration file. The configuration file contains the different classes which can be used.

Spring container can be configured in 3 ways:

* XML configuration file (legacy, but widely used by legacy apps)
* Java Annotations (modern)
* Java Source Code (modern)

Spring Development Process:

* Configure your Spring Beans
* Create Spring Container
* Retrieve beans from container

**Dependency Injection**:

Dependency Injection is **a fundamental aspect of the Spring framework**, through which the Spring container “injects” objects into other objects or “dependencies”. Simply put, this allows for loose coupling of components and moves the responsibility of managing components onto the container.

Injection types used are:

* Constructor Injection
* Setter Injection

Constructor Injection development process:

* Define the dependency interface and class
* Create a constructor in your class for injections
* Configure the dependency injection in spring config file

Setter injection: Inject dependencies by calling setter methods on the required class.

Development process:

* Create setter methods for injections
* Configure dependency in spring configuration file

Injecting literal values:

* Create setter methods in class for injections
* Configure in spring config file

Injecting values from a property file:

* Create properties file
* Load properties file into spring config file
* Reference the values from properties file

**Maven**

Provides a standard directory structure.

* Src/main/java – Java source code
* Src/main/resources – Properties/config files used by the app
* Src/main/webapp – JSP files and web config files along with other web assets (images,css, js, etc)
* Src/test – unit testing code and properties
* Target – destination directory for compiled code. Automatically created by Maven.

Maven key concepts:

POM.xml file -> Project Object Model file. It is the configuration file for the project. Basically a shopping list for Maven.

POM file structure:

* Project meta data
* Dependencies
* Plugins

Project co-ordinates uniquely identify a project. It has elements such as:

* groupID: Name of company/group/org. Convention is to use reverse domain name. Ex: com.test
* artifactID: Name of the project
* Version: A specific release version like 1.0, 2.0.. If project is under active development, then: 1.0-SNAPSHOT

Maven archetypes:

* Can be used to create new maven projects
* Contains template files for a given Maven project
* Can be thought of as a starter files collection for a project

Can create new Maven projects using Maven Archetypes.

* From command line with Maven
* From an DIE

Ex: maven-archetype-quickstart contains a sample maven project whereas maven-archetype-webapp contains a sample web app

How Maven works:

* Reads the config file (POM.xml)
* It checks local repo if the files are present
* If not, it goes to maven central repository to download the files and saves them in the repo.
* Then it builds and runs the application

Repository types:

* Local repository (Located on user’s computer)
* Central repository (Remote repository, needs internet connection)

Local Repository:

Path: C:\Users\<users-home-dir>\.m2\repository

Central repository path: repo.maven.apache.org/maven2

Additional repositories: Some teams may self-host their maven projects/APIs. They can be accessed by configuring additional repositories.

Private repositories: Used privately by companies to share modules between teams. Can use maven repo manager products such as Archiva (Apache), Artifactory (JFrog) or Nexus (Sonatype). Cloud hosted solutions can also be used.

**Spring MVC**

It is a framework for building web applications in Java.

Based on Model View Controller design pattern

Components of a Spring MVC application:

* A set of web pages to layout UI components
* A collection of spring beans (Controllers, services, etc)
* Spring annotations (XML, Java or Annotations)

How does Spring MVC work behind the scene:

* FrontController known as dispatcher servlet (Part of spring framework). It delegates requests to other objects or items in our system.
* As dev, we create MVC. Model contains data, view contains jsp page to render data, controller contains business logic

Controller:

* FrontController delegates request to controller
* Contains business logic (Handle request, store/retrieve data (db, web service…), place data in model
* Send to appropriate view template

Model:

* Contains data
* Store/retrieve data via backend systems (database, web service, etc.. use spring bean if u like)
* Place data in model (data can be any java object/collection)

View template:

* Spring supports many view templates but the most common is JSP+JSTL
* Developer creates a page and displays data
* Can also use other templates such as Thymeleaf, Groovy, etc.

Other concepts covered:

* Creating controllers and views
* Request params and Request mappings
* Form tags and data binding
* Form validation (built-in validation rules, ranges and regular expressions, custom validation rules)

**Bean Lifecycle**

Container started -> Bean instantiated -> Dependencies injected -> Internal spring processing -> Your custom init method -> bean is ready for use

Once container is shut down, you can use a custom destroy method.

For initialization method, use init-method. Similar for destroy method, use destroy-method.

**Java Annotations**

* Special labels/markers added to Java classes
* Provide meta-data about the class
* Processed at compile time or runtime for special processing

@Override:

* Tell compiler that we are overriding a method
* Compiler will check/verify code at compilation time

Spring beans can be configured with annotations. Annotations minimize XML configuration. Once we add an annotation to a class, spring will scan the classes for special annotations. If a class has special annotations, it will automatically register in the spring container.

Default annotations -> When we do not provide any name for the Component, the default bean name provided will be the name of the class with the first letter being small.

Ex: FootballCoach -> footballCoach

**Spring Dependency Injection with Annotations and Autowiring:**

* For DI, spring can use autowiring
* Spring will look for a class that matches the property (matches by type: class or interface)
* Spring will inject it automatically, hence it’s autowired

Injection types:

* Constructor injection
* Setter injection
* Field injection

Use what is comfortable and stick to one type.

**Qualifiers**:

* If there are multiple implementations, we need to use qualifiers.

Other topics covered:

* Scope annotations and bean lifecycle
* Spring config with Java code (no xml) and injecting values from properties files

Spring config with Java code:

* We define a class called configuration and add componentscan annotation to it so that all the classes are added to the spring factory. Otherwise, we can use other ways such as using method to get the object
* For injecting values from properties file, we need to use @PropertySource(“filename”)

**Hibernate**

* Framework for saving/persisting Java objects into a database.

Benefits of Hibernate:

* Handles all low level SQL code
* Minimizes the amount of JDBC code to be developed
* Provides Object to Relational Mapping

Hibernate and JDBC:

* Hibernate uses JDBC for all database communications

Types of mappings:

* One to one
* One to many (bi)
* One to many (uni)
* Many to many

Entity lifecycle:

* Detach -> If entity is detached, its not associated with hibernate session
* Merge -> if entity detached, merge will reattach to session
* Persist -> transitions new instances to managed state. Next flush/commit will save in db
* Remove -> transitions managed entities to be removed. Next flush/commit will delete from db
* Refresh -> reload/sync object with data from db. Prevents stale data

One to one mapping:

* Instructor can have instructor detail
* Cascade: apply same operation to related entities
* By default, no operations are cascaded
* We can list all at once (ALL) or individually

One-to-one bidirectional:

* We have to use mappedBy to help find the associated row. Ex:
* Look at instructorDetail property in Instructor class
* Use info from Instructor class @JoinColumn
* **the value of mappedBy is the name of the association-mapping attribute on the owning side**