```
==================================
Spring Boot & Microservices  ( 27 - SBMS )
==================================

Trainer : Mr. Ashok

Pre-Requisites : Core Java + ADV Java + SQL

Download Course Content : https://ashokitech.com/spring-boot-microservices-online-training

================
Course Content :
================
Spring Core
Spring Boot
Spring Data JPA
Spring Web MVC
RESTFul Services
Microservices
Spring Security
Spring Cloud
Apache Kafka
Redis Cache
Cloud Deployment

Tools : Maven + Junit 5 + Logging + Jenkins Deployement + Docker deployment


Course Duration : 3 - 3.5 Months

Class Timings : 7:30 PM - 9:00 PM (IST)   ( Mon - Sat )

Class Mode : Everyday Live Class in Zoom

Notes: Topic wise class notes will be provided in portal (www.ashokitech.com)


Plan-1 : Course Fee : 7000 INR (Live Classes + Materials)

Plan-2 : Course Fee : 10,000 INR (Live Classes + Materials + Backup Videos - 6 Months)


Note: One telegram / whatsapp group will be created for technical discussions

============================================
Programming Language  Vs Technology  Vs Framework
============================================

====================================
Q) What is a Programming Language ?
====================================

=> Programming Language contains set of Programs / Set of Instructions

                Ex:   C ,   C++ , Java, C#, Python etc....

=> Every Programming Language will follow set of Syntaxes / Rules

=> Using Programming Language we can develop

                        Applications / Tools / Softwares / Technologies /  Frameworks

====================
Q) What is Technology ?
```

```
====================

=> Technology is a software which is developed by using Programming Languages

                Ex: Servlets, JSP, JDBC

=> Technologies are used to simplify our task

=> By using Technologies we can develop several types of applications

                        Project = Business Logic  + Common Logics

======================
Q) What is a Framework ?
======================

=> To overcome the loops holes of Technologies Frameworks came into picture

=> Framework is a semi developed software

=> Framework provides common logics required for application development

                        Ex:

                                1) Capture form data

                                2) Validate Form Data

                                3) Create Connection Pool

                                4) DB CRUD Operations etc.....


=> Framework provides Re-Usable components

=> For example, every Java Developer should write below lines of code to perform DB operations

Class.forName("");
DM.getConn("");
createStatement ( )
executeQuery ( )
process ResultSet
close connection

Note: If we are writing same code for multiple times then it is called as Boiler Plate code /
Redundent Code.

=> To avoid duplicate code / commong logics in project  Frameworks came into picture.

                        Project  =   Business Logics + Common Logics

=> Frameworks provides Common Logics required for the project so that we can focus only on Business
Logic development.

=> Frameworks will improvie developers productivity (we can do more work in less time)

                        Ex:

                                JAVA =>  Hibernate,  Struts, Spring ...

                                .Net    =>  WCF

                                Python  =>  DJango, Flask

                                Salesforce  => Lightening etc
```

==========
Tools
==========

=> Tools are used to automate manual work

Ex:

1) Maven
2) JIRA
3) JENKINS
4 JMETER
5) POSTMAN
6) SONARQUBE etc.....

==

Note: Frameworks will be developed by using Programming Languages only.

Core Java ===> Programming Language

JDBC + Servlets + JSP ====> Technologies

Hibernate + Strtus + Spring + Spring Boot ===> Frameworks

==

==========
Application Architecture
==========

Project : Collection of programs

Project = Frontend + Backend + Database

Frontend : Presentation Logic / User Interface Logic

Frontend Techstack :

1) HTML & CSS
2) Java Script
3) Bootstrap
4) Angular (or) React JS (or) Vue JS

Backend : Web Layer (REST APIs) + Business Layer + Integration Layer + DAO Layer

Backend Tech Stack :

1) Java & J2EE
2) Spring Core
3) Spring Boot
4) JPA
5) REST APIs
6) JSON
7) Microservices
8) Security
9) Kafka + Redis
10) Reactive Programming

Database : Persistent Store

Ex: Oracle, MySQL, Postgres, Mongo DB etc...

Tools : Maven + Git Hub + Log4J + JUnit + Jenkins + SonarQube + JIRA + Docker + Kubernetes

Cloud Platforms : AWS / Azure / GCP

==========
Architecture Types
==========

1) Monolith Architecture

2) Microservices Architecture

==========
Framework
==========

=> Semi Developed software which provides common logics required for projects development

=> Frameworks will help the developers to implement more functionality in less time

=> When we use framework to develop the project, we can focus only on business logic.

==========
Types of Frameworks
==========

1) Frontend Frameworks : To develop user interface in the project.

Ex: Angular

2) Web Frameworks : To develop web layer in the project

Ex: Struts (Outdated)

3) ORM Frameworks : To develop persistence layer in the project.

Ex: Hibernate

=> By using Struts we can develop only Web Layer in the Project (Controllers)

=> By using Hibernate we can develop only Data Access Layer (Persistence Layer)

Note: To overcome the problems of Struts framework, Spring Framework came into market.

=> Spring Framework is called as Application Development Framework

=> By using Spring framework we can develop end to end application

=> Spring is free & open source framework

=> Spring Framework developed in Modular Fashion

Note: Spring framework means collection of modules

==========
Spring Modules
==========

1) Spring Core
2) Spring Context
3) Spring JDBC
4) Spring ORM
5) Spring AOP
6) Spring Web MVC
7) Spring Security
8) Spring Social
9) Spring Batch
10) Spring Data JPA
11) Spring REST
12) Spring Cloud

Note: Spring is very flexible framework. It will not force to use all modules. Based on requirement we can pickup particular module and we can use it.

=> Spring is versatile framework (Easily it can be integrated with other frameworks)

=> The current version of Spring framework is 6.0

=> Spring framework is under license of VM Ware Tanza...

URL : www.spring.io

==========

1) Spring Core : It is base module in the spring framework.

=> Spring Core Module providing fundamental concepts of Spring Framework

1) IOC Container (Inversion Of Control)

2) Dependency Injection

3) Bean Life cycle

4) Bean Scopes

5) Autowiring etc...

2) Spring Context : It will deal with configurations required for our Spring Applications.

3) Spring AOP : Aspect Oriented Programming

=> AOP is used to seperate business logics & Secondary logics in the project

Ex: Security, Logging, Tx, Auditing, Exception Handling...

Note: If we combine business logics & secondary logics then we will face maintence issues of our project.

4) Spring JDBC : Spring JDBC is used to simplify Database Communication logic

=> In java jdbc we need to write boiler plate code (repeated code) like below in several classes

=> Load driver

=> Get connection
=> Create Statement
=> Execute Query
=> Close connection

=> Using Spring JDBC we can directley execute query the remaining part Spring JDBC will take care

5) Spring Web MVC : It is used to develop both Web Applications & Distributed Applications

=> Web Applications (C 2 B)

Ex: Gmail.com, facebook.com etc...

=> Distributed Applications (B 2 B) / Web Services or RESTFul Service

Ex:

IRCTC ---- MakeMyTrip

Passport --- AADHAR app

6) Spring ORM (Object Relational Mapping)

=> Spring Framework having integration with ORM frameworks

Ex: Spring ORM , Spring Data JPA etc....

Note: JDBC will represent data in text format where as Hibernate ORM will represent data in Objects format.

7) Spring Security

=> Security is very crucial for every application

=> Using Spring Security We can implement Authentication & Authorization

=> Spring Security with OAuth2.0

=> Spring Security with JWT (JSON Web Tokens)

8) Spring Batch : Batch means bulk operation

=> Reading data from Excel and store it into database table

=> Sending Monthly statements to customers in email

=> Sending Reminders to customers as Bulk SMS

9) Spring Cloud : It provides some common tools to quickly build distributed systems.

=> It provides service registry to register all our microservices at one place

=> It provides API Gateway to have single entry point for all our apis

=> Load Balancer

=> Monitoring

=> Circutit Breaker (Fault Tolerant Systems / Resillence)

=> Distributed Messaging

=> Routing

10) Spring Test : It provides Unit Test framework

==========

==========
Spring Core : It is all about Managing dependencies among the classes with loosely coupling
==========

=> In project we will develop several classes. All those classes we can categorize into 3 types

1) POJO

2) Java Bean

3) Component

==========
What is Pojo (Plain Old Java Object)
==========

=> Any Java class which can be compiled by using only JDK software is called as POJO class.

Ex-1 : Below class is valid POJO

```
class Demo1 {

        int id;
        String name;
}
```

Ex-2 : Below class is valid POJO

```
class Demo2 extends Thread {

        int id;
        String name;
}
```

Ex-3 : Below class is valid POJO

```
class Demo3 implements Runnable {

                // run method
}
```

Ex-4 : Below class is not POJO because Servlet is part of JEE

```
class Demo4 implements Servlet {

                // run method
}
```

==========
What is Java Bean ?
==========

=> Any java class which follows bean specification rules is called as Java Bean.

1) Class should implement serializable interface

2) Class should have private data members (variables)

3) Every private variable should have public setter & public getter method

4) Class should have zero-param constructor

Note : Bean classes are used to write business logic and to store and retrieve data

==========
What is Component ?
==========

=> The java classes which contains business logic is called as Component classes

Ex: Controllers, Services , Dao classes

=> Controller classes will have logic to deal with Request & Response

=> Service classes will have business logic of our project

Ex: Generate OTP, Send OTP, Send Email, Encrypt & Decrypt PWD etc...

=> DAO classes will contain the logic to communicate with Database

==========

=> In a project we will develop multiple classes and those classes will be dependent on other classes.

Ex:

=> Controller class will call service class methods

=> Service class will call Dao class methods

=> In Java one class can talk to another class in 2 ways

1) Inheritence (IS - A)

2) Composition (HAS - A)

==========
Car & Engine Example
==========

```
class Engine {

        void start ( ) {
                // logic
        }

}


class Car {

        void drive( ) {
                 // star the engine
                 // drive the car
```

```
	}
}
```

=> If we want to drive the car then we need to start the Engine that means Car class drive ( ) method should call Engine class start ( ) method.

Q) In how many ways Car class can call Engine class method ?

=> in 2 ways

1) Inheritence

2) Composition

```
============================== IS-A approach ========================
package in.ashokit;

public class Engine {

	public int start ( ) {
		// logic
		return 1;
	}
}

package in.ashokit;

public class Car extends Engine {

	public void drive() {

		// start the engine

		int start = super.start();

		if (start >= 1) {
			System.out.println("Journey Started");
		}

		// start the journey
	}
}
========================== HAS - A Relation ================================


public class Car {

	public void drive() {

		Engine eng = new Engine ( );
		int start = eng.start();

		if (start >= 1) {
			System.out.println("Journey Started");
		}

		// start the journey
	}
}
```

Note: Tomorrow, if Engine class constructor modified then our Car class will be effected.

```
=============================================================================================

Note: If we use any approach from above then Car class will become tightley coupled with Engine
class. That is not recommended.


#################### Always we need to develop our classes with Loosely Coupling
######################


=> Loosely coupling means without creating Object and without Inheriting properties we should be able
to access one class method in  another class.

=> If we make any changes in Engine class then Car class shouldn't be effected then we can say our
classes are loosely coupled.

######################### To develop classes with loosely coupling we need to use Interfaces
############


package in.ashokit;

public class Car {

	private IEngine eng;

	public Car(IEngine eng) {
		this.eng = eng;
	}

	public void drive() {
		int start = eng.start();

		if (start >= 1) {
			System.out.println("Journey Started...");
		} else {
			System.out.println("Engine in trouble...");
		}
	}
}
-------------------------------------------------------------
package in.ashokit;

public class Main {

	public static void main(String[] args) {

		Car car = new Car (new PetrolEngine());
		car.drive();
	}
}

-----------------------------------------------------------------------------



============================
What is Dependency Injection ?
============================

=> The process of injecting one class object into another class is called as 'Dependency Injection'.

=> We can perform Dependency Injection in 3 ways

		1) Setter Injection
```

```
		2) Constructor Injection

		3) Field Injection

------------------------------- Example to Understand Dependency Injection ------------------------
-------
public class Car {

	private IEngine eng ;

	public void setEng(IEngine eng) {
		this.eng = eng;
	}

	public void drive() {
		int start = eng.start();

		if (start >= 1) {
			System.out.println("Journey Started...");
		} else {
			System.out.println("Engine in trouble...");
		}
	}
}
-----------------------------------------------------------------------------


=> In the above program 'Car' class is dependent on 'Engine' object that means 'Engine' class object
should be injected into 'Car' class.


		Note: Car is dependent on Engine


==============================
Setter Injection (SI)
==============================

=> Setter Injection means, Injecting dependent object into target object using target class setter
method.


public class Car {

	private IEngine eng;

	public void setEng(IEngine eng) {
		this.eng = eng;
	}

	public void drive() {
		int start = eng.start();
		// logic
	}
}

public class Main {

	public static void main(String[] args) {

		// creating target obj
		Car car = new Car();

		// injecting dependent obj into target thru setter method (Setter Injection - SI)
```

```
                car.setEng(new PetrolEngine());

                car.drive();
        }
}

=========================
Constructor Injection  ( CI )
=========================

=> Constructor Injection means, Injecting dependent object into target object using target class
constructor.


public class Car {

        private IEngine eng;

        public Car (IEngine eng) {
                this.eng = eng;
        }

        public void drive() {
                int start = eng.start();
                // logic
        }
}

public class Main {

        public static void main(String[] args) {

                // creating target obj  ( Constructor Injection )
                Car car = new Car(new DieselEngine());

                car.drive();
        }
}



================================================================

Q) Can we perform both SI & CI for single variable ?

Yes, but Setter Injection will override Constructor Injection value.


public class Main {

        public static void main(String[] args) {

                // creating target obj ( Constructor Injection - CI )
                Car car = new Car(new DieselEngine());

                // Setter Injection - SI
                car.setEng(new PetrolEngine());

                car.drive();
        }
}

===================
Field Injection - FI
===================
```

=> Field Injection means, injecting depending object into target class using target class variable is called as Field Injection.

```
public class Car {

	private IEngine eng;

	public void drive() {
		int start = eng.start();

		if (start >= 1) {
			System.out.println("Journey Started...");
		} else {
			System.out.println("Engine in trouble...");
		}
	}
}
```

Note: We can access private variables outside of the class using Reflection API like below

```
public class Main {

	public static void main(String[] args) throws Exception {

		Class<?> clz = Class.forName("in.ashokit.Car");

		Object object = clz.newInstance();
		Car carObj = (Car) object;

		Field engField = clz.getDeclaredField("eng");
		engField.setAccessible(true);

		// Injecting value to variable
		engField.set(carObj, new PetrolEngine());

		carObj.drive();
	}
}
```

==========

```
List l = new ArrayList ( ) ;  // valid

List l = new LinkedList ( ) ;  // valid

List l = new Vector ( ) ;  // valid


IEngine eng = new DieselEngine ( ) ; // valid

IEngine eng = new PetrolEngine ( ) ; // valid
```