```
=======================================
Day-01 : Spring Boot & Microservices
=======================================

1) What is Spring Boot ?

2) Why Spring Boot ?

3) Advantages with Spring Boot

4) What type of apps we can build using boot ?


======================
What is Spring Boot ?
======================

=> Spring Boot is an approach to develop spring framework based applications with less
configurations.

=> Spring Boot is an extension for existing spring framework.

=> Spring Boot internally uses spring framework only.

=> Spring Boot supports rapid application development.

        Spring Boot = (Spring + Auto Config) - xml config

=========================================================
What type of applications we can create using boot ?
=========================================================

=> By using this spring boot we can develop several types of applications

                        a) stand-alone app
                        b) web application ( C2B )
                        c) distributed application (B2B)

                          (webservices / rest apis)

========================
Spring Boot Advantages
========================

1) POM starters

2) Dependency Version Conflicts solution

3) Embedded Servers

4) Auto Configuration

5) Actuators....

============================
What is POM starter ?
============================

=> pom starters are used to simplify maven or gradle configurations

=> We have several pom starters

                        a) spring-boot-starter-web

                        b) spring-boot-starter-data-jpa

                        c) spring-boot-starter-mail
```

d) spring-boot-starter-security

Note: when we add pom starter boot will take care of depenency version.

==========
Embedded Servers
==========

=> Spring Boot will provide servers to run our application

a) tomcat (default)
b) jetty
c) netty

==========
Auto Configuration
==========

=> It is one of the most imp feature in spring boot

=> Boot will identify configurations required for the application based on pom starters and it will provide that configuration in runtme.

==========
Actuators
==========

=> To monitor and manage our application

=> Production Ready Features

/health
/beans
/mappings
/configProps
/heapdump
/threaddemp

==========
Day-02 : Developing Spring Boot Apps
==========

=> We can develop in 2 ways

a) Spring Initializr website (start.spring.io)

-> Create Project
-> Download as zip file
-> Extract zip file
-> Import into IDE as Maven project

b) IDE (STS/IntelliJ)

==========
Spring Boot Application Folder Structure
==========

src/main/java => To write application source code

- Application.java (Start class & Entry Point)

src/main/resources => To keep application config file

- application.properties

src/test/java => To write junit test cases

- ApplicationTest.java

```
src/test/resources      => To keep unit test config files

Maven Dependencies  => libraries downloaded

target  => byte code (.class files)

pom.xml => Maven config file


====================================
What is start class in Spring Boot ?
====================================

=> Start class is entry point for boot application execution

=> It is also called as main class in spring boot

=> When we create boot application, start class will be created by default.

@SpringBootApplication
public class Application {

        public static void main(String[] args) {
                SpringApplication.run(Application.class, args);
        }

}

========================================
What is @SpringBootApplication annotation?
========================================

=> This is used at start class of spring boot...

=> This is equal to 3 annotations

        a) @SpringBootConfiguration
        b) @EnableAutoConfiguration
        c) @ComponentScan


====================================
What is @ComponentScan annotation ?
====================================

=> It is used to scan the project and identify spring beans available in the application.

=> Component Scanning works based on base-package of application.

Note: The package which contains start class of spring boot is called as base package.

=> The packages which are starting with base package name are called as sub packages.


                        in.ashokit
                                - Application.java

                        in.ashokit.service
                        in.ashokit.dao
                        in.ashokit.controller
                        in.ashokit.util

                        com.ashokit ---------- will not be scanned

============================================================
Q) Can we configure more than one base package in the app ?
============================================================
```

```
@SpringBootApplication
@ComponentScan(basePackages = { "in.ashokit", "com.tcs" })
public class Application {

	public static void main(String[] args) {
		SpringApplication.run(Application.class, args);
	}

}
```

==========
Q) How to represent java class as Spring Bean ?
==========

@Component : To represent java class as spring bean

@Service : To represent java class as spring bean

@Repository : To represent java class as spring bean

@Configuration : To represent java class as configuration class

@Bean : To customize objs creation process (method level)

@Controller : To handle http requests in web app (c2b)

@RestController : To handle http requests in rest apis

@ComponentScan : To scan spring beans in our applications

==========
Q) What is run ( ) method in spring boot start class ?
==========

=> It is entrypoint for boot application execution

- Bootstrap context
- starting listeners
- prepare Environment
- printing banner
- create context (IOC)
- prepare context
- call runners
- return ioc

==========
What is banner in Spring Boot ?
==========

=> Logo which is printing on console is called banner

=> By default spring logo will be printed on console

=> We can customize banner text by creating "banner.txt"

File location : src/main/resources

=> Banner works based on modes

a) console (default)

2) log (print in log file)

3) off

=> We can banner mode like below in application.properties file

```
spring.main.banner-mode=off
```

```
==============================================
How IoC will be started in boot application ?
==============================================

=> run ( ) method is responsible to start IOC in boot app

=> run ( ) method will use predefined class to start IOC based on pom starter

default app (standalone) : spring-boot-starter

                Class : AnnotationConfigApplicationContext

web app : spring-boot-starter-web (tomcat)

                Class:  AnnotationConfigServletWebServerApplicationContext

reactive app : spring-boot-starter-webflux (netty)

        Class :  AnnotationConfigReactiveWebServerApplicationContext


================================
What is Runner in Spring Boot ?
================================

=> Runner is used to execute logic only once when application started.

=> We have 2 types of runners in spring boot

                        a) ApplicationRunner
                        b) CommandLineRunner

@Component
public class MyAppRunner implements ApplicationRunner{

        @Override
        public void run(ApplicationArguments args) throws Exception {
                System.out.println("my runner executed....");
                // logic
        }
}

Usecases : insert data in db, delete staging tables data, setup cache.



====================
Day-5 : Autowiring
====================

=> It is used to perform dependency injection

=> Injecting one bean object into another bean object is called as dependency injection.

=> We can perform dependency injection in 3 ways

                                1) setter injection
                                2) constructor injection
                                3) field injection

Note: IoC container is responsible to perform dependency injection.

=> IOC will manage and colloborate bean objects
```

```
======================
What is Spring Bean ?
======================

=> The java class which is managed by IoC is called as spring bean.


==================================
What is Constructor injection ?
==================================

=> Injecting dependent bean object into target bean object using target class constructor is
called as CI.

=> To enable this we will write @Autowired at constructor level.

Note: When we have only one parameterized constructor in target class then @Autowired is optional.


@Service
public class UserService {

	private UserDao userDao;

	public UserService() {
		System.out.println("UserService:: default - constructor");
	}

	@Autowired
	public UserService(UserDao userDao) {
		System.out.println("UserService:: param - Constructor");
		this.userDao = userDao;
	}

	public void getName(int id) {
		String findName = userDao.findName(id);
		System.out.println("Name ::" + findName);
	}

}


==================================
What is setter injection ?
==================================

=> Injecting dependent bean object into target bean object using target class setter method is
called as SI.

=> To enable this we will write @Autowired at setter method level.

@Service
public class UserService {

	private UserDao userDao;

	public UserService() {
		System.out.println("UserService:: default - constructor");
	}

	@Autowired
	public void setUserDao(UserDao userDao) {
		this.userDao = userDao;
	}

	public void getName(int id) {
		String findName = userDao.findName(id);
		System.out.println("Name ::" + findName);
	}
```

```
}

==========================
What is field injection ?
==========================

=> Injecting dependent bean object into target bean object using target bean variable is called as
field injection.

Note: Internally IOC will use reflection api to perform field injection.

@Service
public class UserService {

        @Autowired
        private UserDao userDao;

        public void getName(int id) {
                String findName = userDao.findName(id);
                System.out.println("Name ::" + findName);
        }

}

Note: Using reflection api we can access private variables outside of the class.


public class User {

        private int age = 20;

        public int getAge() {
                return age;
        }
}

public class Test {

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

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

                Object object = clz.newInstance();


                Field ageField = clz.getDeclaredField("age"); // loading variable
                ageField.setAccessible(true); // making variable accessible
                ageField.set(object, 50); // setting value to variable


                User u = (User) object;
                int age = u.getAge();
                System.out.println("User Age :: " + age);

        }

}

==================
Autowiring modes
==================

=> Autowiring works based on modes

1) byName  : Based on variable name it will identify dependent bean

2) byType : Based on variable datatype it will identify dependent bean.
```

```
==================================
What is @Qualifier annotation ?
==================================

=> It is used to identify dependent bean based on bean name

@Component
public class Robot {

        @Autowired
        @Qualifier("chip64")
        private IChip chip;

}

Note: IChip is an interface which is having 32bit & 64bit implementation classes.

=> Based on above configuration it will inject the bean whose name is "chip64"


==============================================
Q) What is Ambiguity problem in Autowiring ?
==============================================

=> If one interface having more than one impl class then we will get ambiguity problem in "byType"
mode.

=> To resolve byType ambiguity problem we can use @Primary annotation.

=> The bean which is having @Primary annotation will get highest priority for autowiring.


====================
What is Bean scope
====================

=> Bean scope represents how many objects should be created for spring bean.

=> By default every spring bean is singleton (only one object)

=> We can customize bean scope using @Scope annotation

=> prototype scope represents every time new object.


Note: For singleton beans  objects will be created when ioc container starting.


@Component
@Scope("prototype")
public class Robot {
                //logic
}

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

@SpringBootApplication

        - @SpringBootConfiguration
        - @EnableAutoConfiguration
        - @ComponentScan

@Configuration
@Bean
@Component
@Service
@Repository
@Autowired
```

```
@Qualifier
@Primary
@Scope

==================================
CI ---> target class constructor
==================================

=> dependent bean is mandatory to create target bean

=> if dependent bean not available then target bean can't created.

=> first dependent bean obj will be created then only target bean object will be created.

==================================
SI --> target class setter method
==================================

=> First target bean object will be created then dependent object obj will be created.

=> After target bean creation using target bean obj setter method will be called (optional)

==================================
FI --> target class variable
==================================

=> IOC will use reflection api to perform FI.

=> Field Inections is simple to write.

=> IOC violating oops principles.

=> It is giving chance to violate Single Responsibility principle.


=========
Summary
==========

1) What is Spring Boot

2) Spring Boot Advantages

                        - pom starters
                        - auto configuration
                        - embedded servers
                        - actuators

3) First Boot Application

4) Boot Application Folder Structure (Maven)

5) What is start class in Spring Boot

                        - @SpringBootApplication
                        - SpringApplication.run ()

6) What is Component Scanning

7) How run ( ) method works internally

                        - bootstrap our app
                        - start listeners
                        - prepare env
                        - print banner
                        - create IOC
                        - call runners
                        - return IOC
```

8) How to customize banner in boot app

9) How ioc will start in boot application ?

10) Runners in Spring Boot

- Application Runner
- Commandline Runner

11) Spring Boot Annotations

- @SpringBootApplication
- @Configuration
- @Bean
- @Component
- @Service
- @Repository

12) Bean Scopes (@Scope)

- singleton
- prototype
- request
- session

13) Autowiring

- byName
- byType
- @Autowired
- @Qualifier
- @Primary

14) What is Dependency Injection

- Constructor Injection
- Setter Injection
- Field Injection

15) What is IOC container