```
=============
Spring Boot
=============

=> Spring framework available in market from 2004 onwards...

=> Spring Boot came into market in the year of 2015...

=> Spring Boot is an extension for spring framework.

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

=> Using springboot we can develop several types of applications.

                1) stand-alone apps (cli)

                2) web apps (C 2 B)

                3) distributed apps (B 2 B) (webservices)

=> If we develop project using spring then we need to manage configurations on our own.

=> If we develop project using springboot then we will get Auto Configuration.

=> Using springboot we can achieve rapid application development.

===========================
Advantages with springboot
===========================

1) POM starters

2) Auto Configuration

3) Embedded Servers

4) Actuators (monitoring and management)


#### pom starters (maven dependencies) : Simplifying maven dependencies in pom.xml file.

        a) web-starter

        b) data-jpa-starter

        c) mail-starter

        d) security-starter

        e) actuator-starter

Note: pom starters are used to enable auto configurations in boot application.


#### Auto Configuration

=> Based on pom starters boot will identify configurations required for the project and boot will
manage those configurations in runtime of our application.

                web-starter ====> tomcat server

                security-starter ===> default login page for authentication
                jpa-starter ===> db connection pool

                actuator-starter ===> monitoring support

#### Embedded Servers

=> Springboot will take care of servers to run our web applications
  (we no need to download & install server)

=> Springboot will support 3 embedded containers

                1) tomcat (default)

                2) jetty

                3) netty


#### Actuators (production ready features)

=> Actuators are used to monitor and manage our applications.

                - check health
                - classes loaded
                - check url patterns
                - thread pool
                - heap memory


==========================
How to install STS IDE
==========================

1) download sts ide jar file

STS Download Link : https://cdn.spring.io/spring-tools/release/STS4/4.24.0.RELEASE/dist/e4.32/spring-
tool-suite-4-4.24.0.RELEASE-e4.32.0-win32.win32.x86_64.self-extracting.jar

2) run that jar file from cmd

syntax : java -jar <file-name>

3) Go to sts folder and open "SpringToolSuite4.exe" file

=============================
How to install IntelliJ IDE
=============================

1) Download IntelliJ community version (free of cost)

Download Link : https://www.jetbrains.com/idea/download/download-thanks.html?
platform=windows&code=IIC

2) Once .exe file is downloaded double click and install it.


=====================================
How to create springboot project ?
=====================================

## Approach-1 : Create boot application using "spring intializer website" then download it and
extract it and import into "Eclipse / STS / intelliJ" IDE.

                URL : start.spring.io
```

Approach-2 : Use STS ide to create springboot application directley.

======================================
Springboot project folder structure
======================================

```
sb-app

        - src/main/java ===========> project source code

                        - Application.java ----> start/main class of spring boot app (entry point)

        - src/main/resources ===========> config files (xml,yml,props)

                        - application.properties ----> config props

        - src/test/java =========> unit test classes (junits)

                        - ApplicationTests.java -----> Junit class

        - Maven dependencies =====> jars downloaded

        - target (.class files goes here)

        - pom.xml ======> maven config file
```

======================================
What is spring-boot-starter-parent ?
======================================

=> The "spring-boot-starter-parent" is a special starter in Spring Boot projects.

=> "spring-boot-starter-parent" acts as parent project for our springboot projects.

=> parent-starter will provide below functionalities

1) default configurations

2) manages dependency versions

3) Reduces boiler plate code

==
How to change embedded server port number ?
==

=> SpringBoot embedded server runs by default on 8080 port.

=> By adding "server.port" property in application.properties file we can change port number

```
server.port=9090
```

==================================
What is dev-tools in springboot ?
==================================

=> Devtools is a maven dependency that we can add in springboot app pom.xml file..

=> It is used to auto restart our embedded servers when there is a code change in our application.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

```
=====================================
What is start class in springboot ?
=====================================
```

```
@SpringBootApplication
public class Application {

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

=> This start class will be created by default when boot app got created.

=> Start class is also called as main class of springboot application.

=> It is entrypoint for boot application execution. Execution will start from here only.

```
========================================
How run () method works internally ?
========================================
```

- start the timer

- create bootstrap context

- load listner classes

- prepare environment (read props/yml file)

- print banner

- create application-context (ioc)

- prepare and refresh IOC (DI)

- stop the timer

- calculate time taken to start application and print on console

- call runners

- return IOC obj

```
====================================================================
Q) How IOC container will be started in Springboot application ?
====================================================================
```

=> run ( ) method will take care of starting IOC container.

=> run ( ) will use below classes to start IOC container based on starter available in pom.xml

spring-boot-starter :: AnnotationConfigApplicationContext

spring-boot-starter-web :: AnnotationConfigServletWebServerApplicationContext

spring-boot-starter-webflux :: AnnotationConfigReactiveWebServerApplicationContext

Note-1: when we use "web-starter" boot will give "tomcat" as default embedded container.

Note-2:When we use "webflux-starter" boot will give "netty" as default embedded container.

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

What is banner in springboot ?

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

=> When we run boot application by default spring logo will be printed on console that is called as banner in springboot.

Note: run ( ) method contains logic to print the banner.

=> We can change banner text according to our requirement by creating "banner.txt" file in "src/main/resources" folder.

=> springboot banner works based on modes. We have 3 types of modes here

1) console (default)
2) log
3) off

===

What is return type of run ( ) method ?

===

=> run ( ) method returns ioc container obj using ConfigurableApplicationContext (interface)

```
ConfigurableApplicationContext context =
                                SpringApplication.run(Application.class, args)
```

Note: in the above code, context variable is holding interface impl class obj.

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

What is runner in springboot ?

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

-> runners are used to execute the logic only one time when boot application started.

ex-1: load data from db table to cache memory when app started

ex-2: clean up temp tables data when app started

ex-3: Send email when our application got started.

Note: run() method will call runners available in springboot app.

==

Q) What is @SpringBootApplication annotation ?

==

=> This is used at start class of the springboot.

=> This annotation is equal to below 3 annotations

- @SpringBootConfiguration
- @EnableAutoConfiguration
- @ComponentScan

@SpringBootConfiguration : Indicates that this class provides Spring Boot application configuration.

@EnableAutoConfiguration : Tells Spring Boot to automatically configure your application based on the dependencies you have in your classpath.

@ComponentScan : Automatically discovers and registers spring beans in the specified packages.

```
==============================================================
Q) What is component scanning and how it works internally ?
==============================================================

=> It is the process of identifying spring beans available in the project.

=> Component Scanning will start from base pacakge (the package which contains start class).

=> Once base package scanning completed, then it will go for sub packages of base package.

Note: Any package name which is starting with base package name is called as sub package.

                        in.ashokit  --------------------- (base package)

                        in.ashokit.beans ------------- will be scanned

                        in.ashokit.dao --------------- will be scanned

                        in.ashokit.service ----------- will be scanned

                        com.tcs.beans ------------------- will not be scanned


Note: We can configure more than one base package using @ComponentScan annotation like below.

@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 ?
==================================================

=> We have several annotations to represent our java class as Spring bean.

@Component

@Service

@Repository

@Configuration

@Bean

@Controller

@RestController

##### Note: Springbean classes will be managed by IOC container.


==============================================
Q) @Component Vs @Service Vs @Repository
==============================================

=> These 3 annotations are part of spring framework and we can use in springboot also.

=> These annotations are also called as stereotype annotations.
```

=> By Using these 3 annotations we can represent java classes as spring beans.

Note: These 3 annotations are class level annotations.

Note: If we represent java class as spring bean then IOC will manage our class (obj creation, dependency injection).

=================
1. @Component
=================

-> General-purpose stereotype.

-> Indicates that the class is a Spring-managed component.

-> Spring will autodetect this class through classpath scanning and register it as a bean.

```
@Component
public class Engine {

}
```

=================
2. @Service
=================

-> Specialization of @Component.

-> It is used to annotate service layer classes.

-> Semantically tells the developer and Spring that this class contains business logic.

```
@Service
public class BookService {

}
```

=================
3. @Repostiory
=================

-> Another specialization of @Component.

=> It is Used to annotate DAO (Data Access Object) classes.

=> It provides additional benefits like automatic exception translation from persistence-specific exceptions (like JDBC exceptions) into Springâ€™s DataAccessException.

```
@Repository
public class UserDao {

}
```

============================
Q) What is @Bean annotation
============================

=> It is method level annotation.

=> It is used when we want to customize bean obj creation.

```
@Bean
public AppSecurity createInstance() {
        // logic
        return new AppSecurity("SHA-256");
}
```

=======================================
Q) What is @Configuration annotation ?
=======================================

=> It is used to represent java class as configuration class.

=> This configuration class is used as replacement for xml configuration.

```
----------------------------------------------------------------------
<bean id="myService" class="com.example.MyService"/>

<bean id="myRepository" class="com.example.MyRepository"/>
---------------------------------------------------------------------
@Configuration
public class AppConfig {

    @Bean
    public MyService myService() {
        return new MyService();
    }

    @Bean
    public MyRepository myRepository() {
        return new MyRepository();
    }
}
-----------------------------------------------------------------------
```

=========================
Runners in Springboot
=========================

-> Runners are used to execute any logic only once when the boot application got started.

-> Runners will be called by SpringApplication.run () method.

use cases:

1) To delete temporary tables data when app started

2) load static tables data into cache memory when app started.

=> We have 2 types of runners in springboot

```
                1) Application Runner (FI)  ==> run (..)

                2) CommandLine Runner (FI) ==> run (..)
```

```
-------------------------------------------------------------------
@Component
public class MyAppRunner implements ApplicationRunner {

        @Override
        public void run(ApplicationArguments args) throws Exception {
                System.out.println("AppRunner executed...");
        }
}
-------------------------------------------------------------------
@Component
public class MyCmdRunner implements CommandLineRunner {

        @Override
        public void run(String... args) throws Exception {
                System.out.println("MyCmdRunner executed...");
        }
}
```

=======================
Project Architecture
=======================

=> In one project we will create Multiple classes like below

        ex : Controller classes

                 Service classes

                 Dao classes

                 Model or DTO classes

                 Entity classes

=> Controllers are used to handle user request and response.

=> Service classes are used to handle business logic.

=> Dao classes are used to perform DB operations.

=> model and dto classes are used to represent data in the form of object.

=> Entity classes are used to represent table structure for ORM operations.

=> By using all these classes we will create Layered Architecture for our project.

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

=> One class method should call another class method for request processing/execution.

        controller method () ---> service method () --------> dao method ( )

=> To call one class method from another class method we need to perform Dependency Injection.

=> The process of injecting one class obj into another class obj is called as dependency injection (DI).

Ex:
                controller class method should call service class method
                (inject service obj into controller)

                Service class method should call dao class method
                (inject dao obj into service)

=> Dependency Injection we can perform in 3 ways

                        1) Setter Injection

                        2) Constructor Injection

                        3) Field Injection

=> IOC container is responsible to perform dependency injection in our applications.

```
=> By using Autowiring we will tell IOC to perform Dependency Injection.

=> To perform DI with Autowiring we will use @Autowired annoation

=> @Autowired annotation we can use at 3 places

                         - setter method level (SI)

                         - constructor level (CI)

                         - field/variable level (FI)

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

=> Injecting dependent obj into target obj using target class setter method is called as setter
injection (S.I).

=> To perform setter injection we will use @Autowired annotation at setter method level.


@Component
public class UserService {

        private UserDao userDao;

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

        public void printName(int id) {
                String nameById = userDao.findNameById(id);
                System.out.println(nameById);
        }
}

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

=> Injecting dependent bean object into target bean object by using target class parameterized
constructor is called as constructor injection.

=> To perform constructor injection we will use @Autowired annotation at constructor level.

@Component
public class UserService {

        private UserDao userDao;

        //@Autowired
        public UserService(UserDao userDao) {
                System.out.println("param constructor");
                this.userDao = userDao;
        }

        public void printName(int id) {
                String nameById = userDao.findNameById(id);
                System.out.println(nameById);
        }
}
```

Note: When we are having single parameterized constructor in target class then writing @Autowired annotation is optional.

==================================
What is Field injection ?
==================================

=> Injecting dependent bean object into target bean object by using target class variable is called as Field injection.

=> To perform field injection we will use @Autowired annotation at variable level.

```
@Component
public class UserService {

    @Autowired
        private UserDao userDao;

        public void printName(int id) {
                String nameById = userDao.findNameById(id);
                System.out.println(nameById);
        }
}
```

Note: Field Injection works based on Reflection API.

=> Using Reflection API we can access private variables outside of the class.

```
-------------------------------------------------------------
package in.ashokit;

public class User {

        private int age = 0;

        public void printAge() {
                System.out.println("Age :: " + age);
        }
}
------------------------------------------------------------
package in.ashokit;

import java.lang.reflect.Field;

public class Test {

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

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

                Object obj = clz.getDeclaredConstructor().newInstance();

                User u = (User) obj;

                u.printAge(); // before setting value for private variable

                Field field = clz.getDeclaredField("age");
                field.setAccessible(true);

                field.set(u, 25); // setting value for private var

                u.printAge(); // after setting value for private variable

        }
}

===============================================
Which Dependency Injection is better to use ?
===============================================

## CI : Dependencies are injected through the class constructor.

=> First dependent object will be created.

=> Promotes immutability : dependencies can't be changed after object creation.

=> Ideal for mandatory dependencies : If dependent obj is available, then only target obj will be
created.

Best for: Mandatory dependencies and making code easier to test and maintain.


## SI : Dependencies are injected through public setter methods.

=> First target object will be created.

=> If we write @Autowired at setter method then only it will be called.

=> If setter method is not called dependent obj will not be injected then there is a chance of
getting NullPointerExceptions.

=> Allows optional dependencies.

=> Supports re-injection or modification post-construction.

Best for: Optional dependencies or when you need to change dependencies dynamically.


## FI : Dependencies are injected directly into class fields using Reflection API.

=> Least boilerplate â€” quick and clean-looking code.

=> Makes testing and understanding the class harder.

=> Cannot be used with final fields.

=> Difficult to test with pure unit tests (requires reflection or framework support).

Note: Generally not recommended


=================
Bean life cycle
=================

Q) What is spring bean ?

-> The java class which is managed by ioc container is called as spring bean.

-> IOC container will take care of bean life cycle

                - creating bean object
                - manage bean object
                - destroy bean object

=> When iOC container managing bean life cycle we can execute life cycle methods using below
annotations

                1) @PostConstruct (after obj creation)
                2) @PreDestroy (before obj deletion)


------------------------------------------------------
@Component
public class Motor {

        @PostConstruct
        public void start() {
                System.out.println("Motor getting started....");
        }

        public void doWork() {
                System.out.println("Motor is running...");
        }

        @PreDestroy
        public void stop() {
                System.out.println("Motor stopped...");
        }

}
---------------------------------------------------------------
```

===

1) Spring vs Spring Boot

2) what is springboot ?

3) What are the advantages with SpringBoot ?

4) How to create a springboot project ?

5) What is folder structure of springboot app ?

6) What is start class in springboot ?

7) How run () method works internally in springboot ?

8) How IOC container will be started in Springboot application ?

9) What is banner in springboot and how to customize it ?

10) What is Runner in springboot ?

11) What is the return type of run () method in springboot start class ?

12) What is @SpringBootApplication annotation ?

13) What is Auto Configuration in SpringBoot & how it works ?

14) What is Component Scanning and how it works in background ?

15) Can we configure more than one base package and how to do it?

16) How to represent java class as a Spring Bean ?

17) @Component Vs @Service Vs @Repository annotations

18) What is @Configuration and @Bean annotation.

19) What is IOC Container

```
20) What is Dependency Injection

21) What is setter injection

22) What is constructor injection

23) What is Field Injection

24) Can we access private variables outside of the class ?

25) Which dependency injection is recommended to use ?

26) What is Spring Bean life cycle ?

27) What are @PostConstruct and @PreDestory annotations ?
```