==========
What is Spring
==========

=> Spring is free & open source java based framework.

=> Using spring we can develop entire application.

=> Spring is called as "application development framework"

=> Spring framework provides common logics required for application development.

=> The author of spring framework is "Rod Johnson".

=> Now spring framework is under license of VMWare company.

=> First version of spring released in the year of 2003.

=> The current version of spring is 6.x version

########## Note: SpringBoot is an extension for Spring Framework. ############

=> Spring Framework developed in modular fashion

1) Spring Core

2) Spring Context

3) Spring AOP

4) Spring DAO / JDBC

5) Spring ORM

6) Spring Web MVC

7) Spring Cloud

8) Spring Security

9) Spring Batch

10) Spring Data

=> Spring Framework is loosely coupled.

Note: It is not mandatory to use all modules of spring framework in one project.

==========
Spring Core Module
==========

=> It is base module of spring framework eco system.

=> Spring core is providing fundamental concepts of spring framework

1) IOC Container

2) Dependency Injection (DI)

3) Auto Wiring

```
======================
Spring Context Module
======================

=> It provides configuration support for spring application development.

=> Configurations we can do in 2 ways

        1) XML (out dated)

        2) Annotations (trending)

============
Spring AOP
============

=> AOP stands for Aspect Oriented Programming.

=> AOP is used to seperate cross-cutting logics of our application

        ex: security, tx, logging, exception handling..


=========================
Spring JDBC / DAO module
=========================

=> It is used to simplify Database connectivity in java applications.

Note: In Java JDBC, we should write so many lines of boiler plate code to perform
DB operations in the project.

                                // load driver
                                // get conn
                                // create stmt
                                // execute query
                                // close conn

=> Spring JDBC provided predefined classes to execute SQL queries directley.
                                JdbcTemplate.execute(query);

=============
Spring ORM
=============

=> ORM means Object Relational Mapping.

=> ORM is used to map Java Objects with Relational Database tables.

=> It is used to simplify Persistence layer development with ORM principles.

=> We can represent DB table data in the form of objects.

=> Spring ORM provided predefined methods to perform curd operations by using
objects.

                hibernateTemplate.save(empObj);

                List<Emp> list = hibernateTemplate.getAll();

Note: Spring ORM internally using Hibernate and Hibernate internally uses JDBC
api.
```

App --> Spring ORM --> Hibernate --> JDBC API --> Database

==========
Spring WEB MVC
==========

=> It is used to develop web applications (C 2 B).

Note: If we develop a web application using servlets, we need to write lot of boiler plate code like below

1) capture form fields data (req.getParameter('key'))
2) validate form data and
3) convert form fields data into object

==========
Spring Cloud
==========

=> It is used to develop Microservices based applications.

=> It provides several services required for microservices management

1) Eureka Server
2) API Gateway
3) Config Server
4) Feign Client

==========
Spring Security
==========

=> It is used to implement security logics in our applications.

=> By using Spring Security module we can implement Authentication and Authorization.

Authentication => who can login into our application

Authorization => after login, which functionality user can access

==========
Spring Batch
==========

=> Spring Batch module is used to implement bulk operations in our applications.

1) generate bank acc stmt and send to customers emails

2) generate credit card bill stmts and send to customers emails

4) Read data from excel file + process it + store into database

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

1) SBMS Overview

2) Core Java vs Adv Java Vs Frameworks

3) What is Framework & Why

4) Java Related Frameworks

5) Hibernate vs Struts vs Spring

6) Spring Introduction

7) Spring Architecture

8) Spring Modules Overview

==========
Spring Core module
==========

=> Base module of Spring Framework

=> Providing fundamental concepts of spring framework

1) IOC
2) DI
3) Auto Wiring

Spring Core module is used to manage our classes in the project.

=> In a project we will have several classes

1) Controller Classes (handle request & response)

2) Service Classes (handle business logic)

3) DAO classes (handle DB ops)

=> In project execution process, One java class method should call another java class method

Ex:
1) Controller class method should call service class method

2) Service class method should call DAO class method

=> We have 2 options to access one java class method in another java class

1) Inheritence (IS-A)

2) Composition (HAS-A)

==========
IS-A Relation
==========

=> Extend the properties from one class to another class.

=> Super class methods we can access directley in sub class.

Ex : Car and Engine classes

Car class ----> drive ( ) method

```
                Engine class ----> start ( ) method
```

Note: If we want to drive the car then we need to start the Engine first. That means Car class functionality is depending on Engine class functionality.

=> Car class drive ( ) method should call Engine class start( ) method.

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

public class Engine {

        public boolean start() {
                // logic
                System.out.println("Engine started...");
                return true;
        }

}
---------------------------------------
package in.ashokit;

public class Car extends Engine {

        public void drive() {
                boolean status = super.start();

                if (status) {
                        System.out.println("Journey started...");
                } else {
                        System.out.println("Engine having trouble...");
                }
        }
}
-------------------------------------------------------------
```

=> In the above approach Car is extending properties from Engine class.

=> In future Car can't extend props from other classes bcz java doesn't support for multiple inheritence.

=> With IS-A relationship our classes will become tightly coupled.

=> To overcome problems of IS-A relation we can use HAS-A relation.

```
================
HAS-A relation
================
```

=> Create the object and call the method

=> Inside Car class, create object for Engine class and call eng class start ( ) method.

```
-----------------------------------------------------
public class Car {

        public void drive() {

                Engine eng = new Engine();
```

```
                boolean status = eng.start();
                if (status) {
                        System.out.println("Engine started...");
                        System.out.println("Journey started...");
                } else {
                        System.out.println("Engine having trouble...");
                }
        }
}
```

=> If someone modify Engine class constructor then Car class will fail...

=> with HAS-A relation also our java classes becoming tightly coupled.

Note: Always we need to develop our classes with loosely coupling.

=> To make our classes loosely coupled, we should not extend properties and we should not create object directley.

=> To make our classes loosely coupled we can use Spring Core Module concepts

1) IOC Container
2) Dependency Injection

==========
What is IOC Container
==========

=> IOC stands for Inversion of control.

=> IOC is used to manage & colloborate the classes and objects available in the application.

=> IOC will perform Dependency Injection in our application.

=> Injecting Dependent class object into target class object is called as Dependency Injection.

=> By using IOC and DI we can achieve Loosely coupling among the classes in our application.

Note: We need to provide input for IOC regarding our target classes and dependent classes to perform Dependency Injection.

Note: We can do configuration in 2 ways

1) XML Based (outdated -> springboot will not support)

2) Annotations

=> IOC will take our normal java classes as input and it provides Spring Beans as output.

==========
What is Spring Bean
==========

=> The java class which is managed by IOC is called as Spring bean.

```
================================================
First App development using Spring framework
================================================

## Step-1 : Create maven project using IDE (Eclipse/ STS / IntelliJ)

        - select simple project (standalone)
        - groupId : in.ashokit
        - artifactId : 01-Spring-App


## Step-2 : Configure Spring dependency in project pom.xml file to download
required libraries.

                URL : https://mvnrepository.com/

-------------------------------------
        <dependencies>
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context</artifactId>
                        <version>6.2.5</version>
                </dependency>
        </dependencies>
-----------------------------------------

## Step-3 :: Create Required java classes

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

public class Engine {

        public Engine() {
                System.out.println("Engine Constructor :: Executed");
        }
}

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

## Step-4 :: Create Spring Bean Configuration file and configure java classes as
spring beans.

                File Location : src/main/resources/spring-beans.xml

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

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="e" class="in.ashokit.Engine"/>

</beans>
-------------------------------------------------------

## Step-5 :: Create Main class to test our application.

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

public class MyApp {
```

```
        public static void main(String[] args) {

                // Start IOC container by giving xml file as input
                ApplicationContext ctxt = new
ClassPathXmlApplicationContext("spring-beans.xml");

                System.out.println("========== IOC Started ==========");

                // getting bean obj from IOC
                Engine e = ctxt.getBean(Engine.class);

                e.start();

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

```
============================
What is Dependency Injection
============================
```

=> The process of injecting one class object into another class object is called as dependency injection.

Note: When we want to call one java class method from another java class method then we need Dependency Injection.

Note: IOC is responsible to perform dependency injection.

=> We can perform Dependency Injection in 3 ways

1) Constructor Injection

2) Setter Injection

3) Field Injection

```
==================================
What is Constructor Injection ?
==================================
```

=> Injecting dependent obj into target obj using target class parameterized constructor is called Constructor injection (C.I).

```
        // constructor injection
        public Car(Engine eng) {
                this.eng = eng;
        }
```

Note: To represent constructor injection we will use below syntax

Syntax : <constructor-arg name="" ref=""/>

Ex :

```
<bean id="c" class="in.ashokit.Car">
        <constructor-arg name="eng" ref="e"/>
</bean>

<bean id="e" class="in.ashokit.Engine" />
```

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

```
What is Setter Injection ?
==================================

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

// SETTER METHOD with dependent obj as parameter
public void setEng(Engine eng) {
        this.eng = eng;
}

Note: To represent setter injection we will use below syntax

Syntax : <property name="" ref=""/>

Ex:

<bean id="c" class="in.ashokit.Car">
        <property name="eng" ref="e"/>
</bean>

<bean id="e" class="in.ashokit.Engine" />

=============
Bean Scopes
=============

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

=> We have below bean scopes in spring

        1) Singleton (default)

        2) Prototype

        3) Request

        4) Session

-------------
Singleton
-------------

=> singleton is default scope.

=> Only one instance will be created for spring bean.

=> Singleton scoped beans objects will be created when IOC container started.

=> Singleton beans will follow Eager Loading.

-------------
Prototype
-------------

=> Every time new object will be created for spring bean on demand basis.

=> When we call getBean() method then only obj will be created.

=> Prototype beans will follow lazy loading.

------------------
```

request & session

=> These 2 scopes are belongs to spring web mvc module.

==========
Spring Core Annotations
==========

1) @Component

2) @Service

3) @Repository

4) @Configuration

5) @Bean ------------------ method level

6) @ComponentScan

7) @Autowired

8) @Qualifier

9) @Primary

10) @Scope

==========
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 Spring
DataAccessException.


@Repository
public class UserDao {

}

====================================
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.

@Configuration
public class AppConfig {

        public AppConfig() {
                System.out.println("AppConfig :: Constructor");
        }
}

============================
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 component scanning and how it works internally ?
============================================================

=> It is the process of identifying spring beans available in the project by
scanning application packages.

=> To specify component scanning we will use @ComponentScan annotation.

------------------------------------------------------------
@Configuration
@ComponentScan(basePackages = "in.ashokit")
public class AppConfig {
```

```
        public AppConfig() {
                System.out.println("AppConfig :: Constructor");
        }
}
```

=> Component Scanning will start from base pacakge

=> 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.

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

==========
Q) What is @Autowired ?
==========

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

=> controller class method should call service class method
(inject service obj into controller class)

=> 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 to 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)

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

CI : Dependencies are injected through the target 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 code, quick and clean looking code.

=> 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...");
	}

}
```

Note: The above 2 annotations are not part of spring framework, to use them in our application we should add below dependency in pom.xml file.

```
<dependency>
    <groupId>javax.annotation</groupId>
    <artifactId>javax.annotation-api</artifactId>
    <version>1.3.2</version>
</dependency>
```

==========
Spring Core Summary
==========

1) What is Framework & Why

2) Struts Vs Hibernate Vs Spring

3) Spring Introduction

4) Spring Architecture

5) Spring Modules Overview

6) Spring Core Module & Why

7) IOC Container

8) Dependency Injection

9) Constructor Injection

10) Setter Injection

11) Field Injection

12) Bean Scopes (singleton & prototype)

13) Spring Core Annotations

14) @Component Vs @Service Vs @Repository

15) @Configuration & @Bean

16) @ComponentScan

17) @Autowired + @Qualifier + @Primary

18) Bean Lifecycle

- @PostConstruct
- @PreDestroy

==========