**Lab 4**

**Task 1. Create a Spring Boot web application, by following the step-by-step tutorial provided for you, in the attached document file - eLibrary - My First Spring Boot Web Application**

A screenshot of a computer

Description automatically generated

**2. Write short answers to the following questions.**

Note: For each question, show it or give an example to support your answer.

1. **What is Spring?**

Spring is an open-source framework for building enterprise-level applications in Java. It provides a comprehensive programming and configuration model that makes developing Java applications easier, more modular, and scalable.

Spring provides many modules for different needs and purposes:

* Spring Core: manage DI and IoC
* Spring Boot
* Spring Data
* Spring Security
* Spring Batch

1. **What is Spring Boot?**

Spring Boot is an extension of the Spring Framework that simplifies the setup, development, and deployment of Spring applications. It uses convention over configuration and provides embedded servers.

1. **What is the relation between Spring platform and Spring Boot?**

Spring Boot is built on top of the Spring Framework and leverages Spring Framework’s features but aims to reduce the complexity involved in configuring a Spring application from scratch. Spring Boot applications use the Spring Framework under the hood.

1. **What is the relation between Spring platform and Spring framework?**

**Spring Framework** is the core component of the Spring ecosystem, providing foundational tools and features to build Java applications such as DI, IoC, Web MVC, AOP, Data Access, Security, …

**Spring Platform** is a broader concept that includes many tools, projects, and frameworks offered by Spring, such as Spring framework, Spring Boot, Spring Cloud, Spring Data, Spring Security, …

1. **What is Dependency Injection and how is it done in the Spring platform/framework?**

Dependency injection (DI) is a technique in which the dependencies of a component (such as a class or a service) are provided from outside rather than being created using “new” keyword inside the component itself. DI helps in decoupling components, making them more flexible, and easier to test.

**Example**

|  |  |
| --- | --- |
| **Without DI** | **With DI** |
| Suppose we have 2 classes  -LogToFile: write messages to file  -LogToDb: write messages to database | Create an interface ILogger and 2 classes that implement ILogger |
| class LogToFile {  void log(String message) {  //log to file  }  }  class LogToDb {  void log(String message) {  //log to database  }  } | interface ILogger {  void log(String message);  }  class LogToFile implements ILogger {  void log(String message) {  //log to file  }  }  class LogToDb implements ILogger {  void log(String message) {  //log to database  }  } |
|  | // register ILogger as DI  @Configuration  public class AppConfig {  @Bean  public ILogger logger() {  return new LogToFile();  }  } |
| Usage in Controller  public class HomeController {  private final ILogger logger;  public HomeController() {  this.logger = new LogToFile();  }  } | Usage in Controller  public class HomeController {  private final ILogger logger;  public HomeController(ILogger logger) {  this.logger = logger;  }  } |
| In case we want to change from LogToFile to LogToDb, we need to change all the codes that new LogToFile() | By using ILogger as DI, in case we want to change from LogToFile to LogToDb, we just need to change inside AppConfig class, making it flexible. |

**DI is implemented in Spring in many ways**

|  |
| --- |
| Using XML configuration |
| Using annotations such as @Autowired, @Component, @Service, @Repository, @Controller |
| Using @Configuration classes and @Bean annotated methods  // register ILogger as DI  @Configuration  public class AppConfig {  @Bean  public ILogger logger() {  return new LogToFile();  }  } |

1. **What is Inversion of Control (IoC) and how is it related to Spring?**

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. IoC is implemented using Dependency Injection (DI).

In Spring, IoC is one of the core features of the framework, and It is implemented through the **Spring IoC Container**.

The Spring IoC Container is responsible for:

* Creating the objects
* Configuring the objects
* Managing dependency between objects
* Wiring the objects together
* Managing the complete lifecycle of objects

**Task 3. Create software solution architecture diagram for the ADS Dental Surgery system**

A diagram of a software application

Description automatically generated