Spring Boot Assignment/Task:

1. Inversion of Control & Dependency Injection (DI)

Task:
Create a small Spring Boot application (NOT Spring Boot with auto-configuration) with the following:

Note:

❓ Why do they mention "NOT Spring Boot with auto-configuration"?

Because:

✅ 1. They want you to understand core Spring concepts

If auto-configuration is allowed, Spring Boot will do everything for you, and you won't learn how Spring actually works internally.

For example:

Feature	Spring Boot (auto-config)	Pure Spring / Manual config
DataSource	Auto-configured	You create DataSource bean manually
MVC	Auto-configured	You configure DispatcherServlet manually
JPA	Auto-configured	You define EntityManagerFactory manually
Component Scan	Auto	You specify packages manually

So, **they want you to learn the core Spring (not Boot) fundamentals**.

1. **Create two classes**:
    - **MessageService** → has a method getMessage() that returns "Hello from MessageService!".
    - **UserController** → depends on **MessageService**.
2. **Inject the dependency in 3 ways**:
    - **Constructor Injection**
    - **Setter Injection**
    - **Field Injection**
3. **Use ApplicationContext to get the bean**:
    - ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
    - UserController user = context.getBean(UserController.class);
    - System.out.println(user.printMessage());

Goal:
Observe that you never used new to create objects, which helps clarify how IoC and DI work.

---

2. Spring Bean Lifecycle

Task:
Create a Spring class DatabaseConnection and perform the following:

1. **Use AnnotationConfigApplicationContext** to load the bean and manually close the context:
2. context.close();

Goal:
Check the console to observe lifecycle events before and after destruction.

---

3. Spring Boot Starter + Auto-Configuration + Properties

Task:
Create a new Spring Boot app with the following steps:

1. **Create a simple class**:

    @Component

```
public class GreetingService {
  public String greet() {
    return "Welcome to Spring Boot!";
   }
}
```

2. **Autowire it in the main application**:

   ```
   @Autowired
   GreetingService service;

   public void init() {
     System.out.println(service.greet());
   }
   ```

3. **Configure application.properties**:

   ```
   server.port=9090
   spring.main.banner-mode=off
   app.message=Hello from properties!
   ```

4. **Read app.message inside GreetingService**:

   ```
   @Value("${app.message}")
   private String msg;
   ```

Goal:
Understand auto-configuration, value injection, starter dependencies, and application.properties configuration.

Console Level

1. Console-based Application (Non-Boot)

Task:
Create a Spring (non-Boot) console-based application that displays a message using IoC + DI.

1. **Create a class GreetingService** with a method getMessage() that returns "Hello from Spring Console App!".
2. **Create a class GreetingController** that depends on GreetingService.
3. Use **Constructor Injection** to inject the dependency.
4. **Create beans.xml** and declare the beans.
5. In the **main()** method:
    - ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");
    - GreetingController controller = ctx.getBean(GreetingController.class);
    - System.out.println(controller.showMessage());

Goal:
The message should print on the console without using new for object creation, proving IoC works.

Spring + Hibernate

1. Hibernate Integration with Spring

Task:
Create a Spring application that stores Employee data in a database using Hibernate.

1. **Create an Employee entity** with fields:
    - id, name, salary.
2. **Configure Hibernate properties** in hibernate.cfg.xml:
    - DB URL, username, password, dialect, hbm2ddl=update.
3. **Create a DAO class EmployeeDAO** with methods:
    - saveEmployee(Employee e)
    - getEmployeeById(int id)
    - getAllEmployees().
4. **Configure Spring beans** (applicationContext.xml):
    - Configure DataSource, SessionFactory, EmployeeDAO.
5. In the main class, save and fetch employees:
6. EmployeeDAO employeeDAO = context.getBean(EmployeeDAO.class);

7. employeeDAO.saveEmployee(new Employee(...));
8. List<Employee> employees = employeeDAO.getAllEmployees();

Goal:
You should be able to insert data, fetch data, and see Hibernate SQL logs in the console.

Spring + Database (Spring JDBC / JPA)

1. Spring JDBC with JdbcTemplate

Task:
Create a Spring project that uses **JdbcTemplate** to insert and fetch student records.

1. **Table**: Student(id, name, course).
2. **Create a DAO class StudentDAO** with methods:
    - addStudent(Student s)
    - getStudents().
3. **Configure**:
    - DataSource (MySQL/Oracle)
    - JdbcTemplate Bean.
4. Use JdbcTemplate queries:
    - jdbcTemplate.update("INSERT INTO student VALUES(?,?,?)", ...);
    - jdbcTemplate.query("SELECT * FROM student", ...);

Goal:
You should see data stored in the database without writing SQL manually.