Aleyna ZENGİN

HW#2

1 – IOC and DI means ?

Inversion of control is a software design principle. With Ioc, it is aimed to minimize their dependencies by providing the management of object instances in the Application. It can also be explained as the framework doing the creation and management of the dependencies in your project, instead of the developer.

Dependency Injection is a programming technique/principle that aims to free a class/object from dependencies and makes that object as independent as possible. Basically, it is an approach that says that if you are going to use an object of another class within a class that you will create, you should not create it with the new keyword.

2 – Spring Bean Scopes ?

Spring Bean Scopes allows us to have more granular control of the bean instances creation. Sometimes we want to create bean instance as singleton but in some other cases we might want it to be created on every request or once in a session.

There are five types of spring bean scopes:

1. Singleton: Each Bean is a singleton by default, but is generated only once. We can think of it as in Singleton Design Pattern. Create once, use again and again.
2. Prototype: It is created every time a request is made for the bean in question. A different instance is generated on each creation.
3. Request : Based on its name, the request bean is created when an HTTP request arrives. An active shape is covered at the HTTP request level.
4. Session : It is created when HTTP request is received in Session Scope Web Applications. An approach similar to Request Scope.
5. Global Session: It covers the definition of a single Bean in the lifecycle of an HTTP. Valid only in a WEB responsive Spring.

3 – What does @SpringBootApplication do ?

The @SpringBootApplication annotation specifies the application's start method. The application starts with this method.

Spring Boot @SpringBootApplication annotation is used to mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning. It’s same as declaring a class with @Configuration, @EnableAutoConfiguration and @ComponentScan annotations.

@Configuration : It is an annotation that performs Java-based configuration.

@ComponentScan : It provides automatic scanning of the components included in the project.

@EnableAutoConfiguration : Allows default configurations to occur automatically.

4 – What is Spring AOP ? Where and How to use it ?

Aspect is a programming paradigm that handles the cross cutting concerns of our applications, and the paradigm's starting point is to find solutions to concerns. The motivation for using this structure is that it complies with principles such as single responsibility and don't repeat yourself.

Some usage examples from daily life:

- Logging → Logging of requests and responses coming to our service.
- Transaction Management → Performing the refund process after the error that will occur in the running code cycle from the receipt of the payment.
- Performance → Calculation of running times of methods.
- Validation → User e-mail permission control before the e-mail to be sent.

Adding Spring's AOP dependency to pom.xml:

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>5.2.7.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjrt</artifactId>
  <version>1.9.5</version>
</dependency>
```

Enable AOP configuration in Spring applications.

AppConfig.java

```
@Configuration
@EnableAspectJAutoProxy
public class AopConfig {

}
```

5 – What is Singleton and where to use it ?

The Singleton design pattern is used to get a single instance of a class. The goal is to provide a global access point to the created object. As long as the system is running, no second instance is created, this will ensure that the object is created only once. Singleton objects are created once the first time they are called and next requests are handled through this object.

We can easily use the singleton design pattern in places where new instances are created for each client such as config, connection (for example, db connection) and in the parts where we call helper functions.

6 – What is Spring Boot Actuator and Where to use it ?

Spring boot's actuator module allows us to monitor and manage application usages in production environment, without coding and configuration for any of them. These monitoring and management information is exposed via REST like endpoint URLs.

Actuator is mainly used to expose operational information about the running application — health, metrics, info, dump, env, etc.

The simplest way to enable the features is to add a dependency to the spring-boot-starter-actuator starter pom file.

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

7 - What is the primary difference between Spring and Spring Boot ?

Spring is an open-source lightweight framework widely used to develop enterprise applications.	Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs.
The most important feature of the Spring Framework is dependency injection.	The most important feature of the Spring Boot is Autoconfiguration.
Spring helps to create a loosely coupled application.	Spring Boot helps to create a stand-alone application.
To run the Spring application, we need to set the server explicitly.	Spring Boot provides embedded servers such as Tomcat and Jetty etc.
Spring doesn't provide support for the in-memory database.	Spring Boot provides support for the in-memory database such as H2.

8 – Why to use VCS ?

- Allowing developers to keep track of code changes.
- Allowing developers to see their code change history.
- Allowing developers to work on the same code files at the same time.
- Allowing developers to separate their code by branching.
- Merging code from different branches.
- Allowing developers to show their conflicts and resolve them.
- Allowing developers to revert their changes to a previous state.

9 – What are SOLID Principles ? Give sample usages in Java ?

SOLID is a popular set of design principles that are used in object-oriented software development. SOLID is an acronym that stands for five key design principles:

1. Single Responsibility Principle : Each software unit (class, object, method) should have a single responsibility.
2. Open/Closed Principle : Software units should be open to development and closed to change.
3. Liskov's Substitution Principle : Objects created from subclasses must exhibit the same behavior when they are replaced by objects of superclasses.
4. Interface Segregation Principle : Instead of interface classes that contain everything, interface classes that do a certain operation should be created.
5. Dependency Inversion Principle: Entities should depend only on abstractions but not on concretions.

- *SRP Principle violation example (bad):*

```
class Employee {
public Pay calculatePay() {...}
public void save() {...}
public String describeEmployee() {...}
}
```

In this example we have pay calculation logic with database logic and reporting logic all mixed up within one class. If you have multiple responsibilities combined into one class, it might be difficult to change one part without breaking others.

- *Dependency Inversion Principle Example*

```
public interface Reader { char getchar(); }
public interface Writer { void putchar(char c)}

class CharCopier {

  void copy(Reader reader, Writer writer) {
    int c;
    while ((c = reader.getchar()) != EOF) {
      writer.putchar();
    }
  }
}

public Keyboard implements Reader {...}
public Printer implements Writer {…}
```

A program depends on Reader and Writer interfaces that are abstractions, and Keyboard and Printer are details that depend on those abstractions by implementing those interfaces. Here CharCopier is oblivious to the low-level details of Reader and Writer implementations and thus you can pass in any Device that implements the Reader and Writer interface and CharCopier would still work correctly.

10 - What is RAD model ?

RAD(Rapid Application Development) is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach. If the requirements are properly understood and described, and the project scope is a constraint, the RAD process allows a development team to create a fully functional system within a concise time period.

RAD (Rapid Application Development) is a concept that products can be developed faster and of higher quality through:

- Gathering requirements using workshops or focus groups
- Prototyping and early, reiterative user testing of designs
- The re-use of software components
- A rigidly paced schedule that refers design improvements to the next product version
- Less formality in reviews and other team communication

11 - What is Spring Boot starter ? How is it useful ?

Before Spring Boot was introduced, Spring Developers used to spend a lot of time on Dependency management. Spring Boot Starters were introduced to solve this problem so that developers can spend more time on the actual code than on the dependencies.

Spring Boot Starters are dependency descriptors that can be added in pom.xml section. There are about 50+ Spring Boot Starters for different Spring and related technologies. These starters provide all dependencies with a unique name. For example, if you want to use Spring Data JPA to access the database, you can include the spring-boot-starter-data-jpa dependency dependency.

The benefits of using Starters include:

- Increased productivity by reducing setup time for developers.
- POM management is simplified since the number of dependencies to add is reduced.
- Dependency configurations tested, production ready and supported.
- It is not necessary to remember the name and version of the dependencies.

12 – What is Caching ? How can we achive caching in Spring Boot ?

Caching is a part of temporary memory (RAM). It lies between the application and persistence database. It stores the recently used data that reduces the number of database hits as much as possible. In other words, caching is to store data for future reference.

First, to activate the cache feature in Spring Boot, the **@EnableCaching** annotation is added to the class where the main method is located.

There are a few annotations we need to know before we can do cache operations.

@Cacheable("cacheName") : If the relevant key value (cacheName) has not been written before, it will be cached first. Then the relevant values are returned to the user. If there is relevant information in the cache, it will be called from the cache.

@CacheEvict("cacheName") : If there is a value written with the corresponding key in the cache, it will delete it.

@CachePut("cacheName") : If we want a method to run every time, we use the method with this annotation. This way, the cache will always have the most up-to-date data.

@CacheConfig: It is a class-level annotation that provides a common cache-related setting. It tells the Spring where to store cache for the class. When we annotate a class with the annotation, it provides a set of default settings for any cache operation defined in that class. Using the annotation, we need not to declare things multiple times.

13 – What & How & Where & Why to logging ?

Java provides the ability to capture the log files.

The need for Log capture

There are multiple reasons why we may need to capture the application activity.

- Recording unusual circumstances or errors that may be happening in the program
- Getting the info about whats going in the application

We can use logging when we simply want to log our system's state or user actions to a file, so our operations staff has a way of knowing what's happening.

```
logger.info("Application successfully started on port 8080");
```

We can use logging when we need to record error messages whenever an exception happens and then send an e-mail or text message to a human for urgent intervention.

```
logger.error("Database connection is down", exception);
```

Or one of our batch jobs might want to record and send warnings to a central, GUI-based log server, whenever it fails to import some records of a csv file.

```
logger.warn("Invalid bank account number for record=[{}]", 53);
```

14 - What is Swagger? Have you implemented it using Spring Boot?

One of the most important needs in Web API development is the need for documentation. Because what the API methods do and how they are used should be clear in the documentation.

The purpose of Swagger is to provide an interface for RestApis. This allows both people and computers to see, examine and understand the features of RestApis without accessing the source code. Yes, I have implemented Swagger in my previous Spring Boot project.

In order to include Swagger in the project in Maven, we must add it as a dependency in the pom.xml file.

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>


<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

After including Swagger in our project, we need to perform some configuration operations.

After specifying **@Configuration**, we need to activate the Swagger 2.0 specification with **@EnableSwagger2**.