HOMEWORK2

RÜMEYSA TÜRKER

1 – IOC and DI means?

IoC (Inversion of Control): It’s a generic term and implemented in several ways (events, delegates etc).

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. We most often use it in the context of object-oriented programming.

DI (Dependency Injection): DI is a sub-type of IoC and is implemented by constructor injection, setter injection or Interface injection.

Dependency injection is a pattern we can use to implement IoC, where the control being inverted is setting an object's dependencies.

Connecting objects with other objects, or “injecting” objects into other objects, is done by an assembler rather than by the objects themselves.

2 – Spring Bean Scopes?

In Spring, bean scope is used to decide which type of bean instance should be returned from Spring container back to the caller. 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** – only one instance of the spring bean will be created for the spring container. This is the default spring bean scope. While using this scope, make sure bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues.
2. **prototype** – A new instance will be created every time the bean is requested from the spring container.
3. **request** – This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request.
4. **session** – A new bean will be created for each HTTP session by the container.
5. **global-session** – This is used to create global session beans for Portlet applications.

3 – What does @SpringBootApplication do?

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.

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

Aspect Oriented Programming (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class.

AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by cross-cutting concerns.

A cross-cutting concern is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.

AOP provides the pluggable way to dynamically add the additional concern before, after or around the actual logic.

5 – What is Singleton and where to use it?

It is used where only a single instance of a class is required to control the action throughout the execution. A singleton class shouldn't have multiple instances in any case and at any cost. Singleton classes are used for logging, driver objects, caching and thread pool, database connections.

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

Spring Boot Actuator is a sub-project of Spring Boot. It adds several production grade services to your application with little effort on your part. Actuator brings production-ready features to our application.

Monitoring our app, gathering metrics, understanding traffic, or the state of our database become trivial with this dependency. The main benefit of this library is that we can get production-grade tools without having to actually implement these features ourselves.

Actuator is mainly used to expose operational information about the running application — health, metrics, info, dump, env, etc. It uses HTTP endpoints or JMX beans to enable us to interact with it.

Once this dependency is on the classpath, several endpoints are available for us out of the box.

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

Spring	Spring Boot
Spring Framework is a widely used Java EE framework for building applications.	**Spring Boot Framework** is widely used to develop **REST APIs**.
It aims to simplify Java EE development that makes developers more productive.	It aims to shorten the code length and provide the easiest way to develop **Web Applications**.
The primary feature of the Spring Framework is **dependency injection**.	The primary feature of Spring Boot is **Autoconfiguration**. It automatically configures the classes based on the requirement.
It helps to make things simpler by allowing us to develop **loosely coupled** applications.	It helps to create a **stand-alone** application with less configuration.

The developer writes a lot of code (**boilerplate code**) to do the minimal task.	It **reduces** boilerplate code.
To test the Spring project, we need to set up the sever explicitly.	Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc.
It does not provide support for an in-memory database.	It offers several plugins for working with an embedded and **in-memory** database such as **H2**.
Developers manually define dependencies for the Spring project in **pom.xml**.	Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care of downloading the dependencies **JARs** based on Spring Boot Requirement.

8 – Why to use VCS?

Version control systems(VCS) are a category of software tools that helps in recording changes made to files by keeping a track of modifications done to the code.

Version control is important to keep track of changes — and keep every team member working on the right version. You should use version control software for all code, files, and assets that multiple team members will collaborate on.

As we know that a software product is developed in collaboration by a group of developers they might be located at different locations and each one of them contributes in some specific kind of functionality/features. So in order to contribute to the product, they made modifications in the source code(either by adding or removing). A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what change has been made. A separate branch is created for every contributor who made the changes and the changes aren't merged into the original source code unless all are analyzed as soon as the changes are green signalled they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.

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

SOLID principles are class-level, object-oriented design concepts that, in conjunction with an extensive test suite, help you avoid code rot.

SOLID design is an acronym for the following five principles:

1. **S**ingle Responsibility Principle
2. **O**pen-Closed Principle
3. **L**iskov Substitution Principle
4. **I**nterface Segregation Principle
5. **D**ependency Inversion Principle

These principles provide a valuable standard for guiding developers away from such “code rot," and instead towards building applications that provide lasting value for customers and sanity for future developers working on your project.

Single Responsibility Principle

The single responsibility principle states that every Java class must perform a single functionality. Implementation of multiple functionalities in a single class mashup the code and if any modification is required may affect the whole class. It precise the code and the code can be easily maintained.

```
public class Person
{
    private Long personId;
    private String firstName;
    private String lastName;
    private String age;
    private List<Account> accounts;
}
```

erson.java

```
public class Account
{
    private Long guid;
    private String accountNumber;
    private String accountName;
    private String status;
    private String type;
}
```

ccount.java

Open-Closed Principle

The application or module entities the methods, functions, variables, etc. The open-closed principle states that according to new requirements the module should be open for extension but closed for modification. The extension allows us to implement new functionality to the module.

```
public class HelloWorldAction extends Action
{
    @Override
    public ActionForward execute(ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response)
            throws Exception
    {
        //Process the request
    }
}
```

xtending Struts Action

Liskov Substitution Principle

The Liskov Substitution Principle (LSP) applies to inheritance in such a way that the derived classes must be completely substitutable for their base classes. In other words, if class A is a subtype of class B, then we should be able to replace B with A without interrupting the behavior of the program.

It extends the open-close principle and also focuses on the behavior of a superclass and its subtypes. We should design the classes to preserve the property unless we have a strong reason to do otherwise.

```java
import java.beans.PropertyEditorSupport;
import org.springframework.util.StringUtils;
import com.howtodoinjava.app.model.Isbn;

public class IsbnEditor extends PropertyEditorSupport {
    @Override
    public void setAsText(String text) throws IllegalArgumentException {
        if (StringUtils.hasText(text)) {
            setValue(new Isbn(text.trim()));
        } else {
            setValue(null);
        }
    }

    @Override
    public String getAsText() {
        Isbn isbn = (Isbn) getValue();
        if (isbn != null) {
            return isbn.getIsbn();
        } else {
            return "";
        }
    }
}
```
IsbnEditor.java

Interface Segregation Principle

The principle states that the larger interfaces split into smaller ones. Because the implementation classes use only the methods that are required. We should not force the client to use the methods that they do not want to use.

The goal of the interface segregation principle is similar to the single responsibility principle.

```
public class MouseMotionListenerImpl implements MouseMotionListener
{
    @Override
    public void mouseDragged(MouseEvent e) {
        //handler code
    }

    @Override
    public void mouseMoved(MouseEvent e) {
        //handler code
    }
}
```

MouseMotionListenerImpl.java

Dependency Inversion Principle

The principle states that we must use abstraction (abstract classes and interfaces) instead of concrete implementations. High-level modules should not depend on the low-level module but both should depend on the abstraction. Because the abstraction does not depend on detail but the detail depends on abstraction. It decouples the software.

```
public class Windows98Machine {

private final StandardKeyboard keyboard;

private final Monitor monitor;

public Windows98Machine() {

 monitor = new Monitor(); keyboard = new StandardKeyboard();

 } }
```

10 - What is RAD model ?

RAD Model or Rapid Application Development model is a software development process based on prototyping without any specific planning. In RAD model, there is less attention paid to the planning and more priority is given to the development tasks. It targets at developing software in a short span of time.

In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype.

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

Spring Boot provides a number of starters that allow us to add jars in the classpath. Spring Boot built-in starters make development easier and rapid. Spring Boot Starters are the dependency descriptors.

In the Spring Boot Framework, all the starters follow a similar naming pattern: spring-boot-starter-*, where * denotes a particular type of application.

The starters provide the required dependencies needed for the build according to the type of project chosen. For example, a Web project has certain dependencies that may be included simply by stating a "Web project."

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

Caching is the process of storing copies of files in a cache, or temporary storage location, so that they can be accessed more quickly.

Spring framework provides cache abstraction api for different cache providers. The usage of the API is very simple, yet very powerful. Today we will see the annotation based Java configuration on caching. Note that we can achieve similar functionality through XML configuration as well.

@EnableCaching

It enables Spring's annotation-driven cache management capability. In spring boot project, we need to add it to the boot application class annotated with @SpringBootApplication. Spring provides one concurrent hashmap as default cache, but we can override CacheManager to register external cache providers as well easily.

@Cacheable

It is used on the method level to let spring know that the response of the method are cacheable. Spring manages the request/response of this method to the cache specified in annotation attribute.

@CachePut

Sometimes we need to manipulate the cacheing manually to put (update) cache before method call. This will allow us to update the cache and will also allow the method to be executed. The method will always be executed and its result placed into the cache (according to the @CachePut options).

@CacheEvict

It is used when we need to evict (remove) the cache previously loaded of master data. When CacheEvict annotated methods will be executed, it will clear the cache.

We can specify key here to remove cache, if we need to remove all the entries of the cache then we need to use allEntries=true. This option comes in handy when an entire cache region needs to be cleared out – rather then evicting each entry (which would take a long time since it is inefficient), all the entries are removed in one operation.

@Caching

This annotation is required when we need both CachePut and CacheEvict at the same time

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

Logging is a powerful aid for understanding and debugging program's run-time behavior. Logs capture and persist the important data and make it available for analysis at any point in time.

Java takes a customizable and extensible approach to logging. While Java provides a basic logging API through the java.util.logging package, you can easily use one or more alternative logging solutions instead. These solutions provide different methods for creating log data, but share the same basic structure.

The Java logging API consists of three core components:

1. Loggers are responsible for capturing events (called LogRecords) and passing them to the appropriate Appender.
2. Appenders (also called Handlers in some logging frameworks) are responsible for recording log events to a destination. Appenders use Layouts to format events before sending them to an output.
3. Layouts (also called Formatters in some logging frameworks) are responsible for converting and formatting the data in a log event. Layouts determine how the data looks when it appears in a log entry.

When your application makes a logging call, the Logger records the event in a LogRecord and forwards it to the appropriate Appender. The Appender then formats the record using a Layout before sending it a destination such as the console, a file, or another application. Additionally, you can use one or more Filters to specify which Appenders should be used for which events. Filters aren't required, but they give you greater control over the flow of your log messages.

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

Swagger is a set of open-source tools built around the OpenAPI Specification that can help you design, build, document and consume REST APIs. The major Swagger tools include:

- Swagger Editor – browser-based editor where you can write OpenAPI specs.
- Swagger UI – renders OpenAPI specs as interactive API documentation.
- Swagger Codegen – generates server stubs and client libraries from an OpenAPI spec.

To enable the Swagger2 in Spring Boot application, you need to add the following dependencies in our build configurations file.

```
<dependency>
   <groupId>io.springfox</groupId>
   <artifactId>springfox-swagger2</artifactId>
   <version>2.7.0</version>
</dependency>
<dependency>
   <groupId>io.springfox</groupId>
   <artifactId>springfox-swagger-ui</artifactId>
   <version>2.7.0</version>
</dependency>
```