Answers

1- IOC and DI means?

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.

In contrast with traditional programming, in which our custom code makes calls to a library, IoC enables a framework to take control of the flow of a program and make calls to our custom code. To enable this, frameworks use abstractions with additional behavior built in. If we want to add our own behavior, we need to extend the classes of the framework or plugin our own classes.

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?

Spring framework defines 6 types of scopes:

- singleton
- prototype
- request
- session
- application
- websocket

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?

AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does this by adding additional behavior to existing code without modifying the code itself. We can declare the new code and the new behaviors separately.

AOP is used in the Spring Framework to provide declarative enterprise services, especially as a replacement for EJB declarative services. The most important such service is declarative transaction management. AOP is also used to allow users to implement custom aspects, complementing their use of OOP with AOP.

5- What is Singleton and where to use it?

When we define a bean with the singleton scope, the container creates a single instance of that bean; all requests for that bean name will return the same object, which is cached. Any modifications to the object will be reflected in all references to the bean. This scope is the default value if no other scope is specified.

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

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. As with most Spring modules, we can easily configure or extend it in many ways.

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

Spring Boot is basically an extension of the Spring framework, which eliminates the boilerplate configurations required for setting up a Spring application.

It takes an opinionated view of the Spring platform, which paves the way for a faster and more efficient development ecosystem.

Here are just a few of the features in Spring Boot:

- Opinionated ‘starter' dependencies to simplify the build and application configuration
- Embedded server to avoid complexity in application deployment
- Metrics, Health check, and externalized configuration
- Automatic config for Spring functionality – whenever possible

8- Why to use VCS?

Version control, also known as source control, is the process of tracking and managing changes to files over time. VCS — version control systems — are software tools designed to help teams work in parallel.

You can use version control for versioning code, binary files, and digital assets.

Version control includes version control software, version control systems, or version control tools. It is a component of software configuration management, sometimes referred to as VCS programming.

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

SOLID principles are object-oriented design concepts relevant to software development. SOLID is an acronym for five other class-design principles: Single Responsibility Principle, Open-Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, and Dependency Inversion Principle.

Principle	Description
Single Responsibility Principle	Each class should be responsible for a single part or functionality of the system.
Open-Closed Principle	Software components should be open for extension, but not for modification.
Liskov Substitution Principle	Objects of a superclass should be replaceable with objects of its subclasses without breaking the system.
Interface Segregation Principle	No client should be forced to depend on methods that it does not use.
Dependency Inversion Principle	High-level modules should not depend on low-level modules, both should depend on abstractions.

Single Responsibility Principle:

```
public class Vehicle {
    public void printDetails() {}
    public double calculateValue() {}
    public void addVehicleToDB() {}
}
```

Open-closed principle:

```
public class Vehicle {
    public double calculateValue() {...}
}
public class Car extends Vehicle {
    public double calculateValue() {
        return this.getValue() * 0.8;
}
public class Truck extends Vehicle{
    public double calculateValue() {
        return this.getValue() * 0.9;
}
```

Liskov substitution principle:

```
public class Rectangle {
    private double height;
    private double width;
    public void setHeight(double h) { height = h; }
    public void setWidht(double w) { width = w; }
    ...
}
public class Square extends Rectangle {
    public void setHeight(double h) {
        super.setHeight(h);
        super.setWidth(h);
    }
    public void setWidth(double w) {
        super.setHeight(w);
        super.setWidth(w);
    }
}
```

Interface segregation principle:

```
public interface Vehicle {
    public void drive();
    public void stop();
    public void refuel();
    public void openDoors();
}
public class Bike implements Vehicle {

    // Can be implemented
    public void drive() {...}
    public void stop() {...}
    public void refuel() {...}

    // Can not be implemented
    public void openDoors() {...}
}
```

Dependency inversion principle:

```
public class Car {
    private Engine engine;
    public Car(Engine e) {
        engine = e;
    }
    public void start() {
        engine.start();
    }
}
public class Engine {
   public void start() {...}
}
```

Above - the car class is depending the engine. It doesn't let us make differences to engine. Here is what we can do with interface:

```
public class Car {
    private EngineInterface engine;
    public Car(EngineInterface e) {
        engine = e;
    }
    public void start() {
        engine.start();
    }
}
public class PetrolEngine implements EngineInterface {
   public void start() {...}
}
public class DieselEngine implements EngineInterface {
   public void start() {...}
}
```

10- What is RAD model?

The Rapid Application Development (or RAD) model is based on prototyping and iterative model with no (or less) specific planning. In general, RAD approach to software development means putting lesser emphasis on planning tasks and more emphasis on development and coming up with a prototype. In disparity to the waterfall model, which emphasizes meticulous specification and planning, the RAD approach means building on continuously evolving requirements, as more and more learnings are drawn as the development progresses.

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

Dependency management is a critical aspects of any complex project. And doing this manually is less than ideal; the more time you spent on it the less time you have on the other important aspects of the project.

Spring Boot starters were built to address exactly this problem. Starter POMs are a set of convenient dependency descriptors that you can include in your application. You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy-paste loads of dependency descriptors.

12- What is Caching? How can we achieve caching in Spring Boot?

Caching is a mechanism to enhance the performance of a system. It is a temporary memory that lies between the application and the persistent database. Cache memory stores recently used data items in order to reduce the number of database hits as much as possible.

Caching of frequently used data in application is a very popular technique to increase performance of application. With caching, we store such frequently accessed data in memory to avoid hitting the costly backends every time when user requests the data. Data access from memory is always faster in comparison to fetching from storage like database, file system or other service calls.

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.

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

The purpose of logging is to create an ongoing record of application events. Log files can be used to review any event within a system, including failures and state transformations. Consequently, log messages can provide valuable information to help pinpoint the cause of performance problems.

Spring Boot uses Apache Commons logging for all internal logging. Spring Boot's default configurations provides a support for the use of Java Util Logging, Log4j2, and Logback. Using these, we can configure the console logging as well as file logging.

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

Swagger is an Interface Description Language for describing RESTful APIs expressed using JSON. Swagger is used together with a set of open-source software tools to design, build, document, and use RESTful web services. Swagger includes automated documentation, code generation (into many programming languages), and test-case generation.