Spring Core Assignment

Name – Rabindra Kumar

ID – asterdm2212_13

Q1- What is Spring Bean and Spring Container?

The bean is an important concept of the Spring framework and as such, if you want to use Spring, it is fundamental to understand what it is and how it works. By definition, a Spring bean is an object that form the backbone of your application and that is managed by the Spring IoC container. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. Otherwise, a bean is simply one of many objects in your application. Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.

The Spring container is the core of Spring Framework. The container, use for creating the objects and configuring them. Also, Spring IoC Containers use for managing the complete lifecycle from creation to its destruction. It uses *Dependency Injection* (DI) to manage components and these objects are called Spring Beans. The container uses configuration metadata which represent by Java code, annotations or XML along with Java POJO classes as seen below.

Steps to activate Spring container in our application:

- Create a project with a name SpringExample and a package packagecom.example. These should under src folder of the created project.
- Add the needed Spring libraries using Add External JARs.
- Create Java classes HelloWorld and MainApp under the package packagecom .example.
- Create Beans config file Beans.xml under src folder.
- At last, create content of all Java files and Beans configuration file and run the file.

Q2 – Explain about IOC and Dependency Injection.

Inversion of Control (IOC) Injection

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.

The advantages of this architecture are:

- decoupling the execution of a task from its implementation
- making it easier to switch between different implementations
- greater modularity of a program
- greater ease in testing a program by isolating a component or mocking its dependencies, and allowing components to communicate through contracts

Dependency Injection

Dependency Injection (DI) is a design pattern used to implement IoC. It allows the creation of dependent objects outside of a class and provides those objects to a class through different ways. Using DI, we move the creation and binding of the dependent objects outside of the class that depends on them.

The Dependency Injection pattern involves 3 types of classes.

1. **Client Class:** The client class (dependent class) is a class which depends on the service class
2. **Service Class:** The service class (dependency) is a class that provides service to the client class.
3. **Injector Class:** The injector class injects the service class object into the client class.

Types of Dependency Injection:

1. Constructor Injection
2. Property Injection
3. Method Injection