What is Spring Framework?

In Java, you must create object instance by yourself. Sometimes you don't have enough information to do that.

Spring "DI" can supply whatever object instance you want. They can be configured and wired externally without changing any business logic.

Spring frameworks contain following concepts:

- DI (Dependency Injection) : Instance of objects are injected into a target Object's field (where field should be ideally of an interface type) via Constructors/Setters instead of target Object creating the instances themselves. Hence, this approach enabled application objects being POJO which can be used in different environment and with different implementations.

- IOC (Inversion of Control) container: Framework code invokes application code during an operation and ask for application specific information instead of application calling the Framework code directly, hence control is inverted. An example of IOC is Template pattern via sub-classing. Spring IOC provides annotations based IOC as well.

- AOP (Aspect-Oriented Programming) : This allows separation of cross-cutting concerns by adding behaviors (aspects) to the application code instead of application involving into those concerns itself. This enables application to be modular instead of mixing different concerns to a single place. The examples are Transaction management, logging etc.

- Lightweight Alternative to Java EE : Spring is lightweight solution for building enterprise application using POJO. It can be used in servlet container (e.g. Tomcat server) and doesn't require an Application server.

What is Spring bean?

A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container, for example, in the form of XML definitions.

Example

We are going to give a quick step by step example with concise explanations to demonstrate how a particular implementation of an interface can be injected to a client application using Spring Framework.

Creating Beans

```
public interface GreetingService {
    void sayHi(String name);
}

public class GreetingServiceImpl implements GreetingService {
    public void sayHi(String message) {
        System.out.println(message);
    }
}
```

```
public class GreetingServiceClient {

    @Autowired
    private GreetingService greeting;

    public void showMessage() {
        greeting.sayHi("Hello world!");
    }
}
```

What is @Autowired?

In above code, we used annotation @Autowired in class GreetingServiceClient. This annotation tells the Spring container where to perform dependency injection. We have to register instance of GreetingServiceClient as bean to make it happen. We also have to register GreetingServiceImpl as bean. Next we will see how we can do that by using @Configuration on Java config class.

Spring Configuration and Starting the Container

In this example, class AppRunner, is responsible for defining and wiring dependencies. In other words it's a Spring specific configuration class. Our example demonstrates Java-based Configuration which is the alternative to XML based configuration. The same class is also responsible for starting the Spring container from the main method.

```
@Configuration
public class AppRunner {
    @Bean
    public GreetingService createGreetingService() {
    }
    @Bean
    public GreetingServiceClient createClient() {
        return new GreetingServiceClient();
    }

    public static void main(String... strings) {
        AnnotationConfigApplicationContext context =
                                new AnnotationConfigApplicationContext(AppRunner.class);
        GreetingServiceClient bean = context.getBean(GreetingServiceClient.class);
        bean.showMessage();
    }

}
```

The annotations @Configuration and @Bean

We used annotation @Configuration to tell the container that beans are defined in AppRunner. The methods which returns bean instances are annotated with @Bean, which is another directive for the container so that it will know what objects should be registered as beans.

What is AnnotationConfigApplicationContext?

This class implements ApplicationContext. The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata.

In our example we passed our config java class to the AnnotationConfigApplicationContext constructor. This class implements AnnotationConfigRegistry as well, so it accepts annotated classes as input.

Once instantiated, a spring application context object represents runtime reference of the spring container. Using the context object we can access registered beans. It's read-only but can be refreshed/reloaded whenever needed.

Running app

Running AppRunner class will generate following output:

```
Hello world!
```