UNIT IV

Contents

Java annotations

- Java annotations are used to provide some kind of metadata to the Java compiler and JVM.
  - They are embedded within the source code
  - Tells the compiler about the behavior of the field, class, interface, or method.
- The annotation starts with the symbol @ followed by the name of the annotation.
- The following are few built-in annotations:
  - @Override:
    - It is used when the child class is overriding methods of its parent class
  - @Deprecated:
    - It is used to denote the class, method, or field that should no longer be referenced in the source code.
  - @SuppressWarnings:
    - It is used when the deprecated methods, classes, or fields are used and we don't want the compiler to generate a warning message.

@override: you can write your own example to override parent method in subclass

@Deprecated

```
public class A{
   @Deprecated
  // Method 1, Old method
 public void oldmethod()   {
    System.out.println("This is a deprecated method");
 }
  // Method 2, New method
 public void newmethod(String m1)   {
       System.out.println(m1);
 }
   public static void main(String a[])   {
    A obj = new A();
     // Now calling the old method
    obj.oldmethod();
 }}
```

@SuppressWarnings

```
public class Machine {
   private List versions;
   @SuppressWarnings("unchecked")
   public void addVersion(String version) {
      versions.add(version);
   }
}
```

Spring and Spring Boot Annotations

@Bean

- To declare a bean, simply annotate a method with the @Bean annotation
- @Bean annotation indicates that a method produces a bean to be managed by the Spring container.
- Use that method to register a bean definition within an **ApplicationContext** of the type specified as the method's return value.
- The @Bean annotation is usually declared in the Configuration class to create Spring Bean definitions.

```
public interface Dept
{
String getDept();
}
```

```
public class ITDept implements Dept{
@override
public String getDept(){
return "ITDepartment";
}
}
```

```
import org.springframework.context.annotation.Configuration;
@Configuration
public class AppConfig{
@Bean
public Dept getDept(){
return new ITDept();
}}
```

```
main()
{
ConfigurableApplicationContext context=SpringApplication.run(AnnotationsApplication.class,args);
ITDept ob=context.getBean(ITDept.class);
System.out.println(ob.getDept());}
```

@Bean

- By default bean name is the same as the method name @Bean(name="methodname")
- We can modify it using @Bean(name="your own name")

Some modifications in AppConfig class:
@Bean(name="ITDeptBean")
in the main method, we can call using the name
ItDept ob=(ITDept)context.getBean(name:ItDeptBean);

```
import org.springframework.context.annotation.Configuration;
@Configuration
public class AppConfig{
@Bean(name ="ITDeptBean")
public Dept getDept(){
return new ITDept();
}}
```

```
main()
{
ConfigurableApplicationContext context=SpringApplication.run(AnnotationsApplication.class,args);
ItDept ob=context.getBean(name:ItDeptBean);
System.out.println(ob.getDept());
}
```

@Component

- @Component annotation tells that an annotated class is a Spring bean or a spring component
- In this case, the spring container automatically create a spring bean (i.e. spring container creates an object to the class and manages it)

```
//create a class
@Component
public class DeptController{
public String getDept(){
return "its the Department controller";
}
}
```

```
main()
{
var context:ConfigurableApplicationContext=SpringApplication.run(SpringAnnotationsApplicatoin.class,args);
BookController ob=context.getBean(DeptController.class);
System.out.println(ob.getDept());
}
```

@Autowired

- **The @Autowired annotation is used to inject the bean automatically**
- **It is used in construction injection, setter injection and field injection**

Constructor Injection

```
@Component
public class ItDept
{
public String getDept()
{
return "IT Department";
}
}
```

```
main()
{
ConfigurableApplicationContext
context=SpringApplication.run(AnnotationsApplication.class,args);
DeptController ob1=(DeptController)context.getBean(name:deptController);
System.out.println(ob1.getDept());
}
```

```
@Component
public class DeptController
{
private ItDept ob;
@Autowired
public DeptController(ItDept ob)
{
this.ob=ob;
}
public String getDept()
{
return ob.getDept();
}
}
```

@Autowired

- **The @Autowired annotation is used to inject the bean automatically**
- **It is used in construction injection, setter injection and field injection**

Setter Injection

```
@Component
public class ItDept
{
public String getDept()
{
return "IT Department";
}
}
```

```
@Component
public class DeptController
{
private ItDept ob;
@Autowired
public String setDept(ItDept ob)
{
this.ob=ob;
}
public String getDept()
{
return ob.getDept();
}
}
```

```
main()
{
ConfigurableApplicationContext
context=SpringApplication.run(AnnotationsApplication.class,args);
DeptController ob1=(DeptController)context.getBean(name:deptController);
System.out.println(ob1.getDept());
}
```

@Autowired

- **The @Autowired annotation is used to inject the bean automatically**
- **It is used in construction injection, setter injection and field injection**

Field(variable) Injection

```
@Component
public class ItDept
{
public String getDept()
{
return "IT Department";
}
}
```

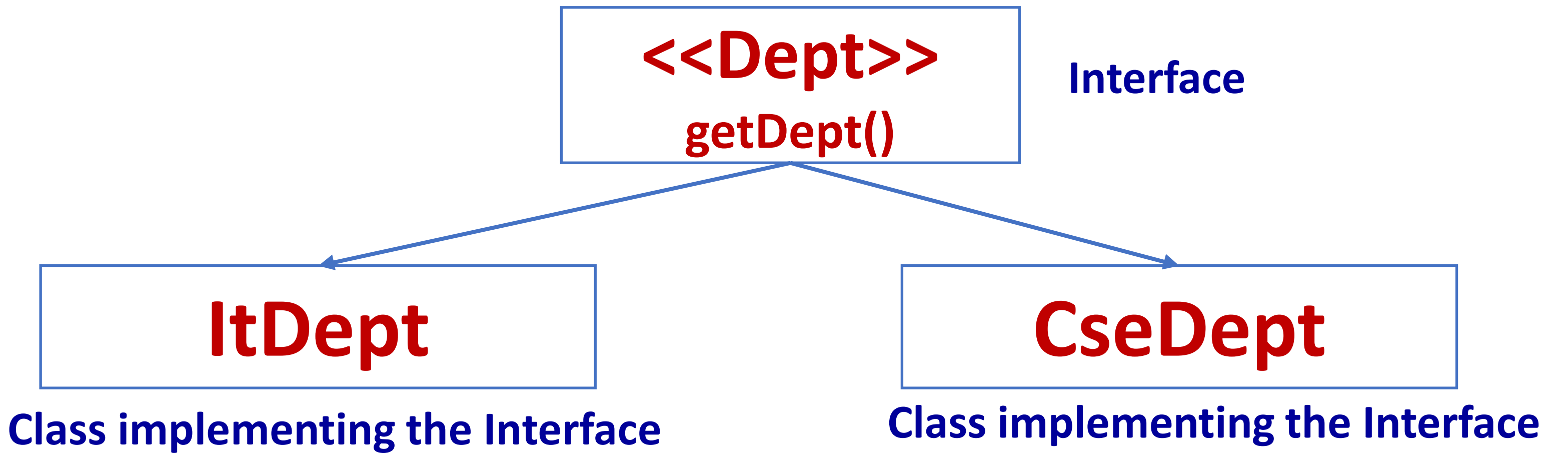
```
@Component
public class DeptController
{
@Autowired
private ItDept ob;
public String getDept()
{
return ob.getDept();
}
}
```

```
main()
{
ConfigurableApplicationContext
context=SpringApplication.run(AnnotationsApplication.class,args);
DeptController ob1=(DeptController)context.getBean(name:deptController);
System.out.println(ob1.getDept());
}
```

@Qualifier

- The annotation is used to avoid confusion when 2 or more beans are configured for same type

- Used in combination with @Autowired



<<Dept>>
getDept()
Interface
ItDept
CseDept
Class implementing the Interface
Class implementing the Interface

The class invoking the IT and CSE classes

- Which class to be invoked by the Spring IOC
- It depends on the @Qualifier annotation
- Use Constructor injection and specify with @Qualifier

Example @ Qualifier

```
public interface Dept
{
String getDept();
}
```

```
@Component
public class ItDept implements Dept
{@override
public String getDept()
{
return "This is IT Department;
}}
```

```
@Component
public class CseDept implements Dept
{@override
public String getDept()
{
return "This is CSE Department;
}}
```

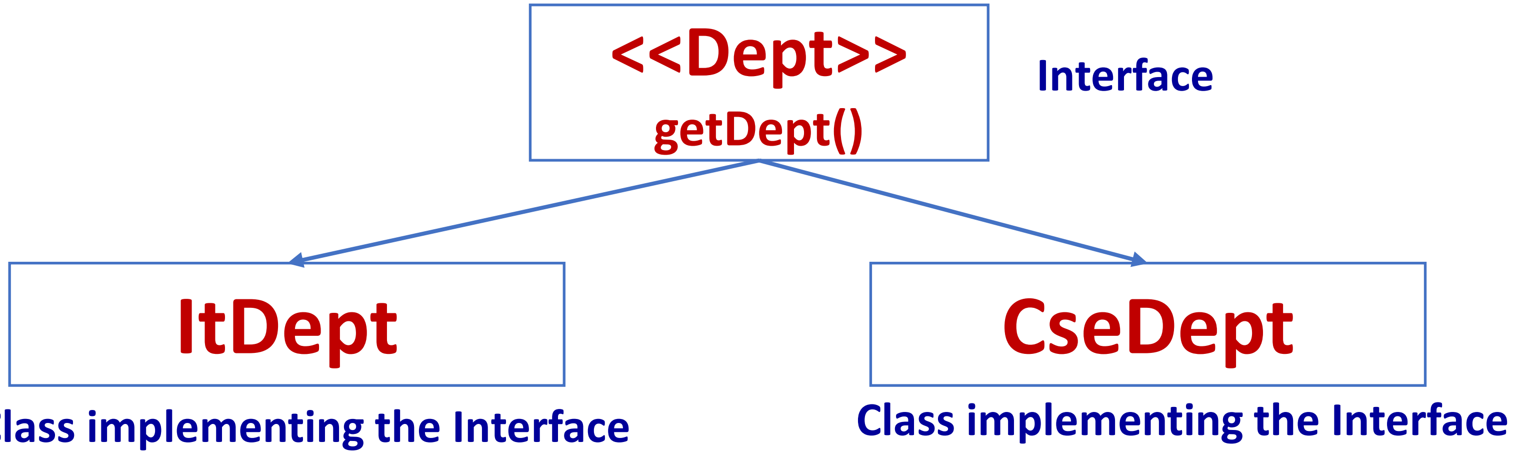
```
@Component
public class DeptController{
private Dept ob;
@Autowired
public DeptController(@Qualifier("itDept") Dept ob)
{
this.ob=ob;
}
public String getDept()
{
return ob.getDept();
}}
```

```
main()
{
ConfigurableApplicationContext context=SpringApplication.run(AnnotationsApplication.class,args);
DeptController ob1=(DeptController)context.getBean(name:deptController);
System.out.println(ob1.getDept());
}
```

@Primary

- The annotation is used to avoid confusion when 2 or more beans are configured for same type and identify to which bean needs higher priority

- Used in combination with @Autowired


<<Dept>>
getDept()
Interface
ItDept
CseDept
Class implementing the Interface
Class implementing the Interface

The class invoking the IT and CSE classes

- Which class to be given higher priority and invoked by the Spring IOC
- It depends on the @Primary annotation

Example @Primary

```
public interface Dept
{
String getDept();
}
```

```
@Component
@Primary
public class ItDept implements Dept
{@override
public String getDept()
{
return "This is IT Department;
}}
```

```
@Component
public class CseDept implements Dept
{@override
public String getDept()
{
return "This is CSE Department;
}}
```

```
@Component
public class DeptController{
private Dept ob;
@Autowired
public DeptController(Dept ob)
{
this.ob=ob;
}
public String getDept()
{
return ob.getDept();
}}
```

```
main()
{
ConfigurableApplicationContext context=SpringApplication.run(AnnotationsApplication.class,args);
DeptController ob1=(DeptController)context.getBean(name:deptController);
System.out.println(ob1.getDept());
}
```

@Lazy Annotation

- Spring creates all singleton beans eagerly during the startup
- We can load the beans lazily(on-demand) using @Lazy annotation
- We may combine with @Component, @Configuration and @Bean

```
@Component
class EarlyLoader
{
public EarlyLoader()
{
return "This is loaded early";
}
}
```

```
@Component
@Lazy
class LazyLoader
{
public LazyLoader()
{
return "This is loaded lazily";
}
}`
```

```
main()
{
ConfigurableApplicationContext context=SpringApplication.run(AnnotationsApplication.class,args);
//usage of lazy loading
LazyLoader ob=context.getBean(LazyLoader.class);
}
```

@ConfigurationProperties

- The annotation allows to map the entire property file into an object
- This annotation helps to load a group of related properties
- Suppose, application.properties file contain information related to an employee
    - emp.name=kumar
    - emp.sal=25000
- These properties are accessed in a class using @ConfigurationProperties by specifying @ConfigurationProperties("emp");

```
@Configuration
@ConfigurationProperties("emp");
public class Employee{
private String name;
private int sal;
public void setName(String name){
this.name=name;}
public String getName(){
return name;
}
public void setSal(int sal){
this.sal=sal;
}
public String getSal(){
return sal;
}
@override
public String toString(){
return "Emp details are"+name+""+sal;
}}
```

```
main(){
@SpringBootApplication
public class Application{
@Autowired
private Employee employee;
public static void main(String [] args)
{
SpringApplication.run(Application.class,args);
}
@PostConstruct
public void init()
{
System.out.println(employee.toString());
}
}
```

Spring framework stereotype annotations

- Stereotype annotations are the annotations that denote the roles of types or methods in the overall architecture
- These annotations are used at the class level
  1. **@Component**
  2. **@Controller**
  3. **@Repository**
  4. **@Service**

1. @Component
   - This indicates that an annotated class is a **component**.
   - The classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.

```
@Component
public class ItDept
public String getDept()
{
return "This is IT Department;
}
}
```

Spring framework stereotype annotations contd..

2. @Controller

- This indicates that an annotated class is a **controller**.
- This annotation serves as a specialization of **`@Component`** that allows you to implement classes to be autodetected through classpath scanning.
- It is used with **`@RequestMapping`** and **`@ResponseBody`** annotations for developing web APIs.

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
@Controller
public class MyController {
@RequestMapping(method = RequestMethod.GET, value = "/")
@ResponseBody
public String doSomething() {
return "Hello";
}
}
```

When requested for **`http://localhost:8080/`** on the browser, it returns **`Hello`**.

Spring framework stereotype annotations contd..

3. Repository

- This indicates that an annotated class is a **Repository**
- This is used when the application involves retrieval, storage, or search on the database or collection of objects.

```
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;
import com.author.kickstart.interfaces.impl.Car;
@Repository
public interface MyRepository extends CrudRepository<Car, String> {
}
```

We can see our Spring with database access example too.

4. @Service

- This indicates that an annotated class is a **Service**

```
package com.author.kickstart.service;
import org.springframework.stereotype.Service;
@Service
public class MyService {
}
```