

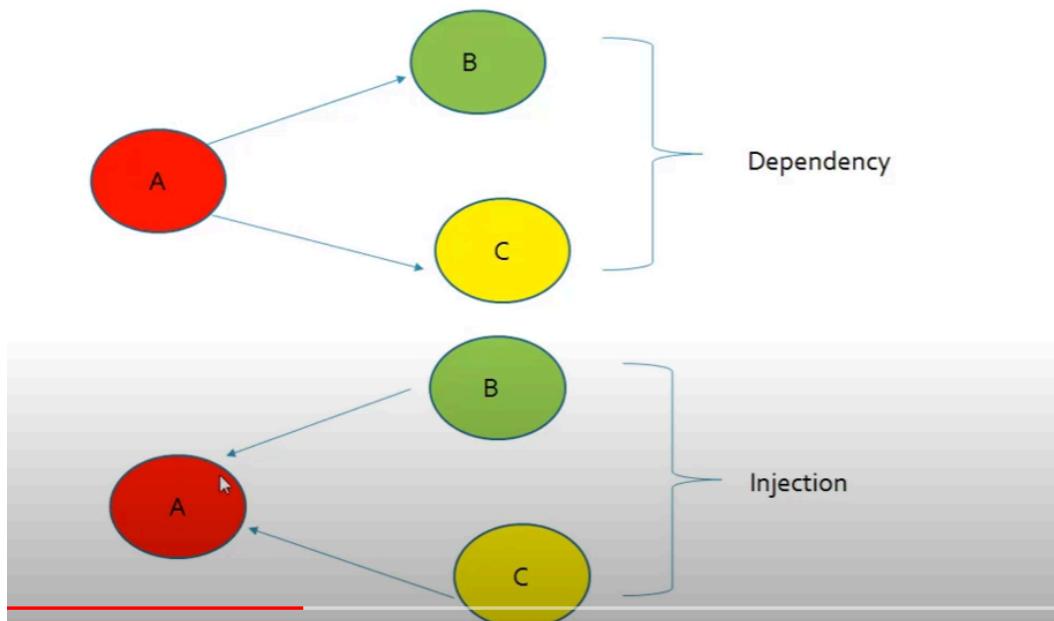
1 Inversion of Control

Giving control to frame work to do that spring allows dependency injection.

- A class X has a dependency to class Y, if class X uses class Y as a variable.

```
class X
{
    Y y;
    .....
}
```

Dependency Injection and Inversion of Control



Dependency Injection and Inversion of Control

- Inversion of Control is a principle by which the control of objects is transferred to a container or framework.
- Dependency injection is a pattern through which IoC is implemented, where the control being inverted is the setting of object's dependencies.
- The act of connecting objects with other objects, or “injecting” objects into other objects, is done by container rather than by the objects themselves.
- A class should not configure itself but should be configured from outside.

Dependency Injection

Spring framework provides two ways to inject dependency

- By Constructor (Constructor Injection)-In the case of constructor-based dependency injection, the container will invoke a constructor with arguments each representing a dependency we want to set.
- By Setter method (Setter Injection)-For setter-based DI, the container will call setter methods of our class.

Without IOC	With IOC (constructor Injection)	With IOC (setter Injection)
<pre>class Employee{ Address address; Employee(){ address=new Address("XYZ Street", "Pune", "MH"); }</pre>	<pre>class Employee{ Address address; Employee(Address address){ this.address=address; }</pre>	<pre>class Employee{ Address address; public void setAddress(Address address){ this.address=address; }</pre>

2 AOP(Aspect oriented programming)

Tackle down the cross cutting applications (repeatedly work)

3 Web Application

Spring data libraries to connect with database easily (not tedious jdbc)

Ease of implementing other frameworks so its called **frameworks of frameworks**

Spring Bot is a layer on top of the java to build our applications

Maven is build tool helps in project management.

Build tool is essential for the process of building.

It is needed for the following processes:

- Generating source code
- Generating documentation from the source code
- Compiling of source code
- Packaging of the complied codes into JAR files
- Installing the packaged code in local repository, server or central repository

Maven mainly used for java-based projects, helps in downloading dependencies, which refers to the libraries or **JAR files**.

Feature	Maven	Gradle
Build Script	XML (pom.xml)	Groovy
Project Structure	Strict conventions	More flexible
Dependency Management	Transitive	Non-transitive
Build Performance	Slower	Faster (due to build cache and incremental builds)
Multi-language Support	Java only	Java, C++, Python, and more
Customization	Limited	More control over the build process and can customize it to their needs

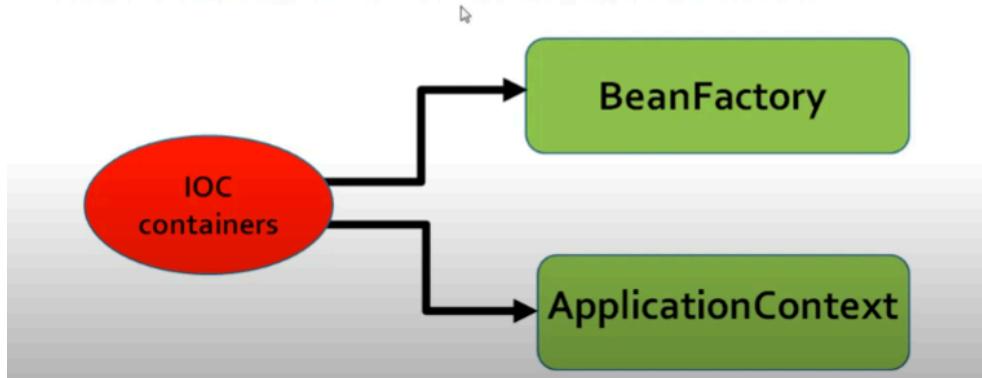
- **groupId** – a unique base name of the company or group that created the project (unique)
- **artifactId** – a unique name of the project

Dependency Injection

the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans

IOC

- In Spring framework, IOC container is responsible to inject the dependencies. We provide metadata to the IOC container either by XML file or annotation.
- The IoC container is responsible to instantiate, configure and assemble the objects.



When ever spring starts we get beans from container

IOC(Inversion of control) has 2 interface **Beanfactory** and **ApplicationContext**

The BeanFactory provides the configuration framework and basic functionality, while the ApplicationContext adds enhanced capabilities to it.

class

ApplicationContext extends beans factory . so there are lot more features in ApplicationContext

This try to get all beans

Class **ClassPathXMLApplicationContext** implements the ApplicationContext interface

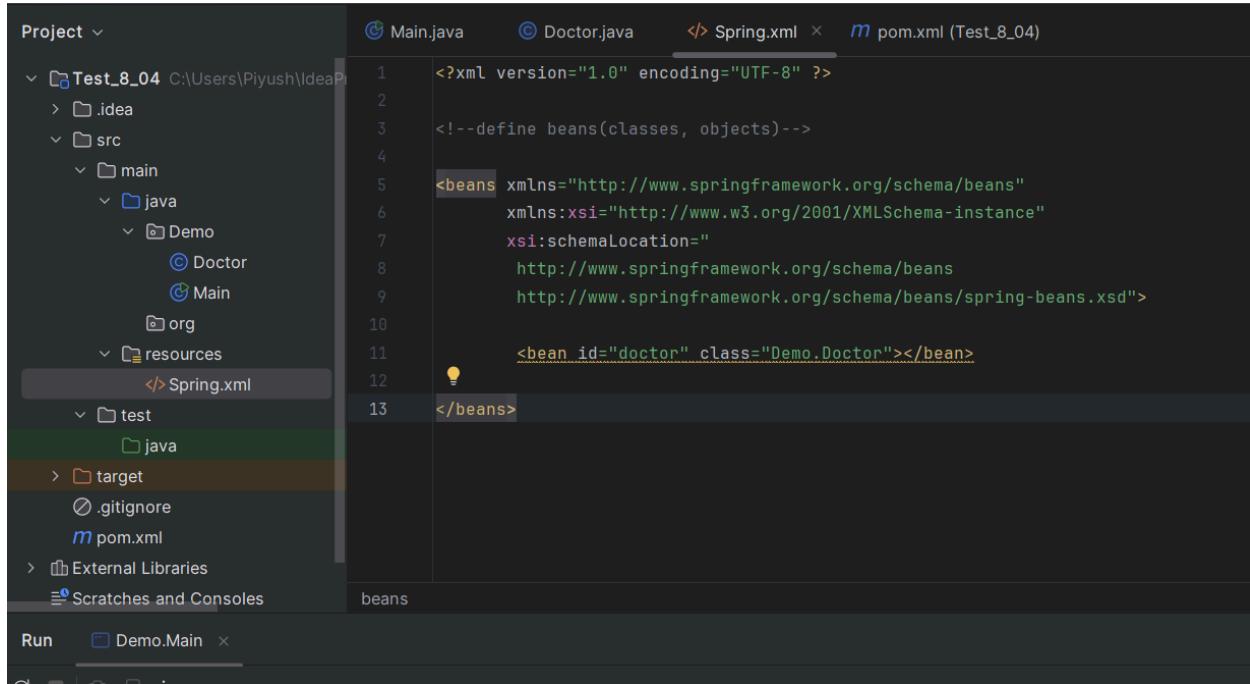
The screenshot shows an IDE interface with a project tree on the left and a code editor on the right. The project tree for 'Test_8_04' includes a 'src' folder with 'main' and 'test' subfolders, and an 'External Libraries' section. The code editor displays 'Doctor.java' with the following content:

```

package Demo;
public class Doctor {
    //Qualification qualification;
    public void assist(){
        System.out.println("Doctor is assisting");
    }
}
  
```

The 'Doctor' class is selected in the code editor. The status bar at the bottom shows 'Run' and 'Demo.Main'.

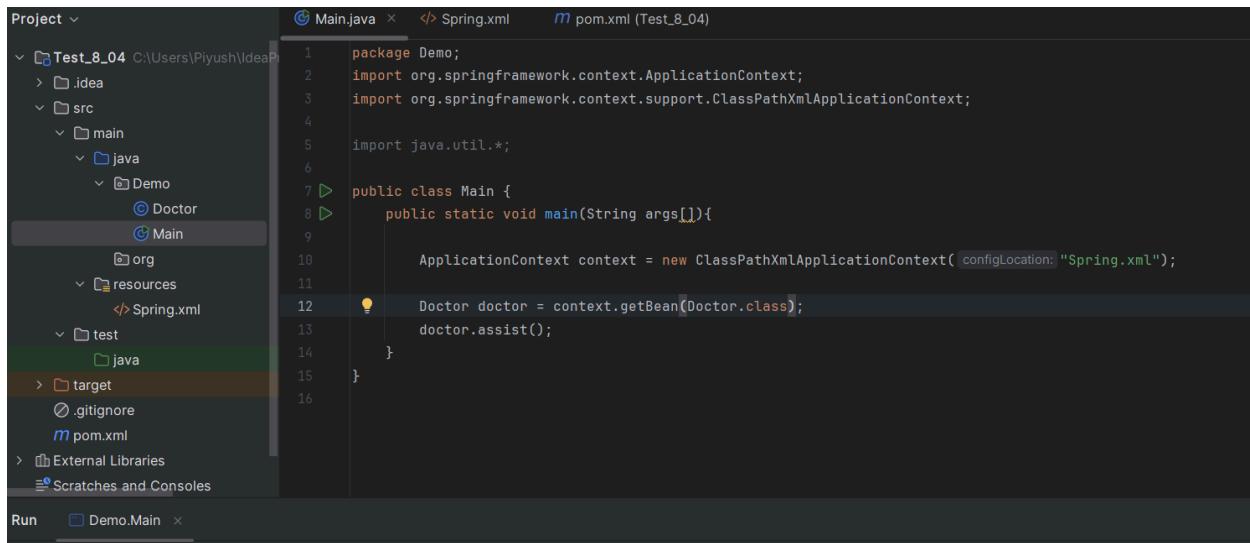
Defining xml file for Bean



The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `Spring.xml` file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<! --define beans(classes, objects)-->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="doctor" class="Demo.Doctor"></bean>
</beans>
```

The `beans` element is highlighted in yellow, indicating it's selected.



The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `Main.java` file:

```
package Demo;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import java.util.*;
public class Main {
    public static void main(String args[]){
        ApplicationContext context = new ClassPathXmlApplicationContext("Spring.xml");
        Doctor doctor = context.getBean(Doctor.class);
        doctor.assist();
    }
}
```

The `Doctor` bean is injected into the `Main` class using the `getBean` method of the `ApplicationContext`.

Setter Injection and Constructor Injection into beans

Project ▾

- Test_8_04 C:\Users\Piyush\ideaProjects
- .idea
- src
 - main
 - java
 - Demo
 - Doctor
 - Main
 - Nurse
 - Staff
 - resources
 - </> Spring.xml
 - test
 - java
- target
- .gitignore
- pom.xml

Run Demo.Main

```

<!--define beans(classes, objects)-->

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="doctor" class="Demo.Doctor">
        <!-- Setter injection -->
        <property name="qualification" value="MBA"/>
    </bean>
    <bean id="nurse" class="Demo.Nurse">
        <!-- Contructor injection-->
        <constructor-arg value="MBBS"/>
    </bean>
</beans>
```

Using Component annotate

Project ▾

- Annotate method [Test_8_04]
- .idea
- src
 - main
 - java
 - Demo
 - BeanConfig
 - Doctor
 - Main
 - Nurse
 - Staff
 - resources
 - </> Spring.xml
 - test
 - target
 - .gitignore
 - pom.xml

External Libraries

Scratches and Consoles

Main.java Doctor.java Spring.xml BeanConfig.java

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           https://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/context
                           https://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config base-package="Demo"/>
</beans>
```

```
package Demo;
import org.springframework.stereotype.Component;
//making it component
@Component 1 usage
public class Nurse implements Staff{
    public void assist() { System.out.println("Nurse is assisting"); }
}
```

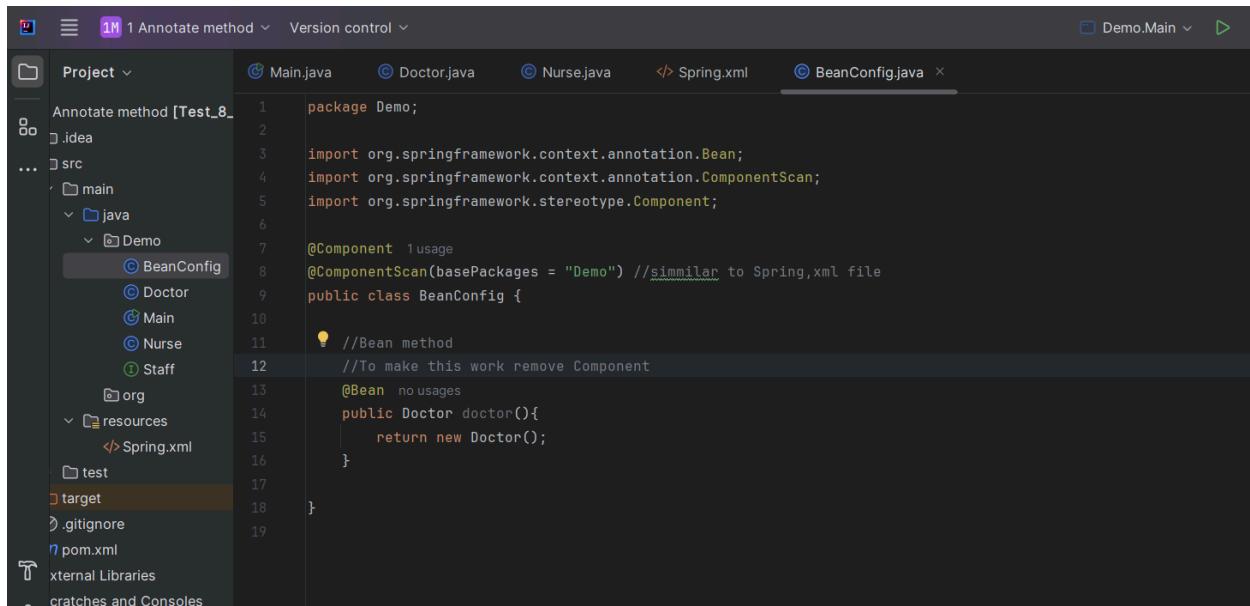
```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "demo")
public class BeanConfig {
```

```
package Demo;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import java.util.*;
public class Main {
    public static void main(String args[]){
        //using ClassPathXmlApplicationContext class
        ApplicationContext context = new ClassPathXmlApplicationContext( configLocation: "Spring.xml");
        Doctor doctor = context.getBean(Doctor.class);
        doctor.assist();
        Staff staff = context.getBean(Nurse.class);
        staff.assist();
    }
}
```

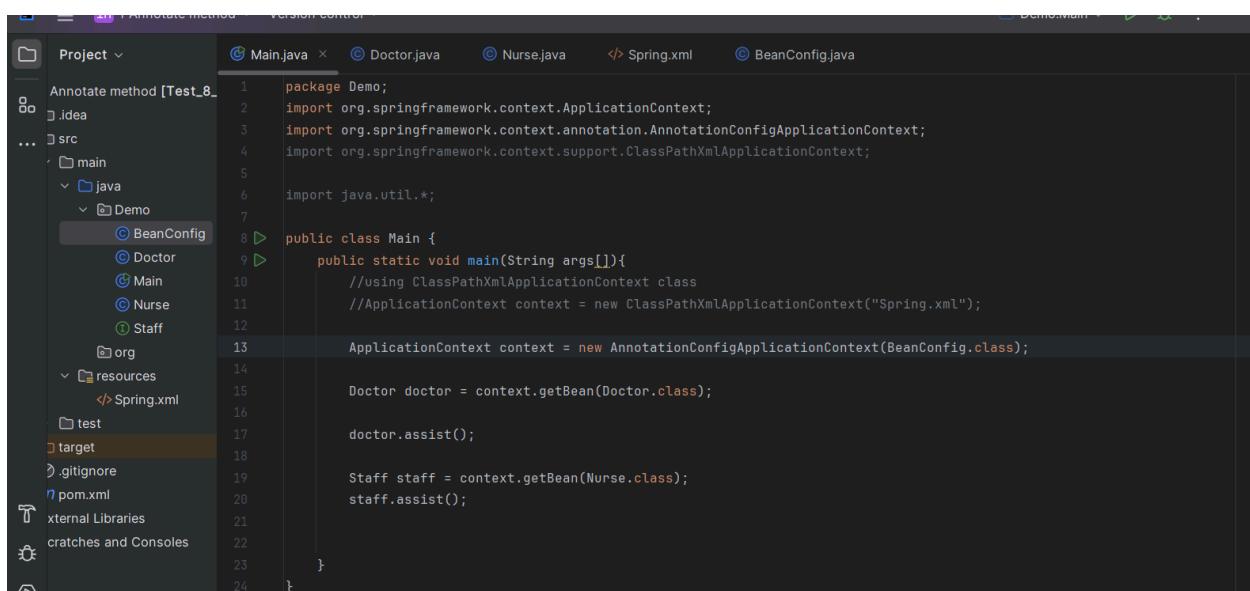
Here it will go to BeanConfig class, check what all package to be load where @Component is defined.

Defining in bean Class



IntelliJ IDEA interface showing the code for BeanConfig.java:

```
package Demo;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.stereotype.Component;
@Component
@Usage
@ComponentScan(basePackages = "Demo") //similar to Spring.xml file
public class BeanConfig {
    //Bean method
    //To make this work remove Component
    @Bean
    public Doctor doctor(){
        return new Doctor();
    }
}
```



IntelliJ IDEA interface showing the code for Main.java:

```
package Demo;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import java.util.*;
public class Main {
    public static void main(String args[]){
        //using ClassPathXmlApplicationContext class
        //ApplicationContext context = new ClassPathXmlApplicationContext("Spring.xml");
        ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);
        Doctor doctor = context.getBean(Doctor.class);
        doctor.assist();
        Staff staff = context.getBean(Nurse.class);
        staff.assist();
    }
}
```

Types of scope

Singleton (default scope)(All access only one object)

Prototype (Different objects)

The screenshot shows two code editors and a terminal window in IntelliJ IDEA.

Top Editor (Nurse.java):

```
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

//making it component
@Component 4 usages
@Scope(scopeName = "singleton") default
@Scope(scopeName = "prototype")
public class Nurse implements Staff{

    @Override
    public String toString() {
        return "Doctor{" + "qualification=" + Qualification + "}";
    }

    private String Qualification; 3 usages

    public void assist(){ 1 usage
        System.out.println("Nurse is assisting");
    }
}
```

Bottom Editor (Main.java):

```
ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);

Doctor doctor = context.getBean(Doctor.class);

doctor.assist();

Nurse nurse = context.getBean(Nurse.class);

nurse.setQualification("MBBS");
System.out.println(nurse.getQualification());

Nurse nurse1 = context.getBean(Nurse.class);
System.out.println(nurse1.getQualification());
```

Terminal:

```
C:\Users\Piyush\.jdks\corretto-11.0.22\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.3\lib\idea_rt.jar" -Dfile.encoding=UTF-8
Doctor is assisting
MBBS
null
```

request
session
application
Websocket

Bean Life Cycle

Life Cycle Methods

**Spring provide two important methods
to every bean**

public void init()

public void destroy()

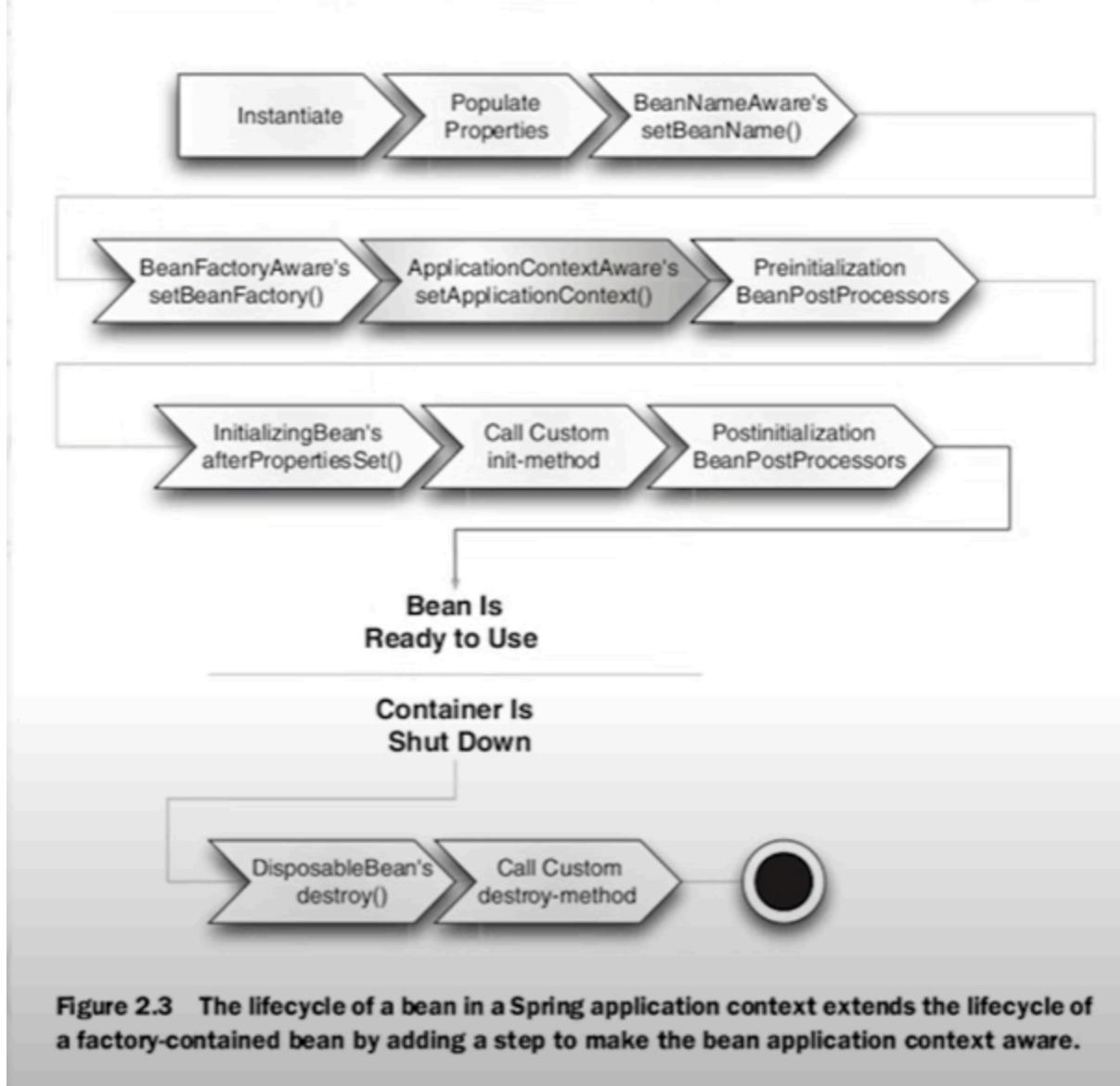
Bean accepts xml file

Life Cycle



Configure Technique





SetBeanMethod

Post construct

File structure:

```

... src
  - main
    - Java
      - Demo
        - BeanConfig
        - Doctor
        - Main (selected)
        - Nurse
        - Staff
      - org
    - resources
      - Spring.xml
  - test
  - target
  - .gitignore
  - pom.xml

```

Code content (Main.java):

```

12
13     ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);
14
15     Doctor doctor = context.getBean(Doctor.class);
16
17     doctor.assist();
18
19     Nurse nurse = context.getBean(Nurse.class);
20
21     nurse.setQualification("MBBS");
22     System.out.println(nurse.getQualification());
23
24     Nurse nurse1 = context.getBean(Nurse.class);
25     System.out.println(nurse1.getQualification());
26
27
28 }
29 }
```

Run configuration: Demo.Main

Output window:

```

C:\Users\Piyush\.jdks\corretto-11.0.22\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.3\lib\idea_rt.jar" -Dfile.encoding=UTF-8 -jar "C:\Users\Piyush\IdeaProjects\Spring\target\Demo-0.0.1-SNAPSHOT.jar"
Doctor is assisting
MBBS
null

```

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "1 Annotate method". It contains a "src" directory with a "main" package containing "Main.java", "Doctor.java", and "Nurse.java". It also includes "resources" and "test" directories.
- Main.java:** The code defines a main class that creates a Spring context and retrieves beans by type.
- POM Configuration:** The "pom.xml" file is present in the project root.
- Run Output:** The terminal shows the execution of the application, outputting messages about bean creation and qualification.

```
public class Main {
    public static void main(String args[]){
        ApplicationContext context = new AnnotationConfigApplicationContext(BeanConfig.class);
        Doctor doctor = context.getBean(Doctor.class);
        doctor.assist();
        // Nurse nurse = context.getBean(Nurse.class);
        // ...
        // ...
        // ...
    }
}
```

```
C:\Users\Piyush\.jdks\corretto-11.0.22\bin\java.exe ...
Set Bean Aware called
Post Construct method called
Doctor is assisting
```

Bean then no Components

If Components then no need to define Bean

AOP

Remove cross cutting concern(loggers,authentication,sanitizing the data)

Loggers

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "3". It contains a "src" directory with a "main" package containing "Main.java", "ShoppingCart.java", "AuthenticationAspect.java", and "LoggingAspect.java". It also includes "resources" and "test" directories.
- AuthenticationAspect.java:** The code defines a component that logs a message when the "checkout" method is called.
- POM Configuration:** The "pom.xml" file is present in the project root.

```
package demo;

import org.springframework.stereotype.Component;

@Component 2 usages
public class ShoppingCart {
    public void checkout(String status){ 1 usage
        System.out.println("Checkout method called!!!");
    }
}
```

Project ▾

C:\Users\Piyush\IdeaProjects\3... Main.java ShoppingCart.java AuthenticationAspect.java LoggingAspect.java BeanConfig.java

```
1 package demo;
2
3 import org.springframework.context.annotation.ComponentScan;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.context.annotation.EnableAspectJAutoProxy;
6
7 @Configuration
8 @ComponentScan(basePackages = "demo")
9 @EnableAspectJAutoProxy
10 public class BeanConfig {
11
12 }
13
```

Run demo.Main

Project ▾

C:\Users\Piyush\IdeaProjects\3... Main.java ShoppingCart.java AuthenticationAspect.java LoggingAspect.java BeanConfig.java

```
1 package demo;
2
3 import org.aspectj.lang.annotation.After;
4 import org.aspectj.lang.annotation.Aspect;
5 import org.aspectj.lang.annotation.Before;
6 import org.springframework.stereotype.Component;
7
8 @Component
9 @Aspect
10 public class LoggingAspect {
11
12     @Before("execution(* demo.ShoppingCart.checkout(..))")
13     public void beforeLogger(){
14         System.out.println("Before Logger");
15     }
16
17     @After("execution(* *.checkout(..))")
18     public void afterLogger(){
19         System.out.println("After Logger");
20     }
21 }
```

Run demo.Main

Point cuts for authorization

Project ▾

C:\Users\Piyush\IdeaProjects\3... Main.java ShoppingCart.java AuthenticationAspect.java LoggingAspect.java BeanConfig.java

```
7
8     @Component
9     @Aspect
10    public class AuthenticationAspect {
11
12        @Pointcut("within(demo..*)")
13        public void authenticatingPointCut(){}
14
15    }
16
17    @Pointcut("within(demo..*)")
18    public void authorizationPointCut(){}
19
20
21    @Before("authenticatingPointCut() && authorizationPointCut()")
22    public void authenticate(){
23        System.out.println("Authenticating the Request");
24    }
25
26 }
```

"C:\Program Files\Java\jdk-17.0.5\bin\java.exe" ...

Authenticating the Request
Before Logger
Checkout method called!!

Spring Boot is extension of Spring framework (Rapid Application Development)

Starter Template For all(jdbc, test)

Auto Configuration

Embedded Servers (JAR File)(Always production ready)

Micro Services (smaller, independently deployable services)

Tomcat is an open-source web server and servlet. The Apache Software Foundation has developed it. It is used widely for hosting Java-based applications on the web.

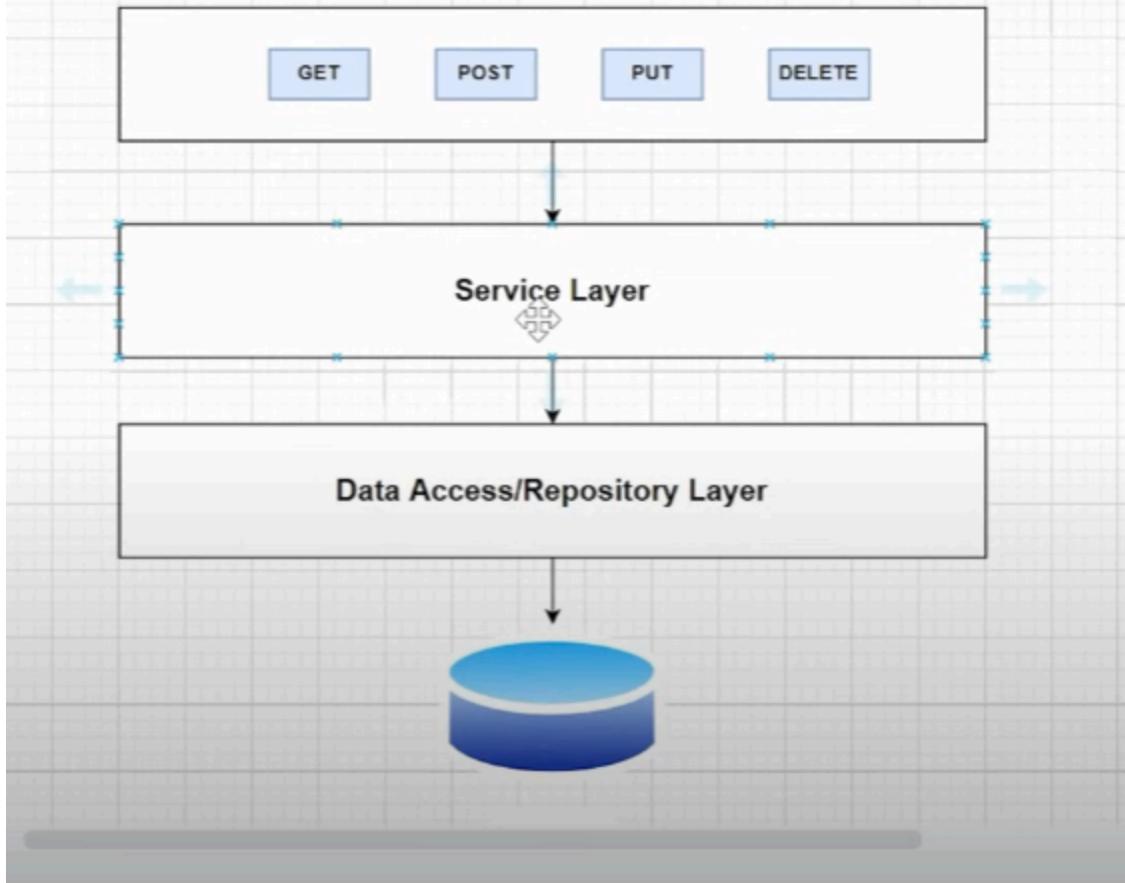
The screenshot shows the IntelliJ IDEA interface with a Spring Boot project named "Spring-Boot-REST-Test". The project structure on the left includes folders for .idea, .mvn, src (with main and java subfolders), resources, static, templates, application.properties, test, target, .gitignore, and HELP.md. The code editor on the right displays "HelloController.java" with the following content:

```
1 package com.LearnSpringBoot.SpringBootRESTTest.controller;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RequestMethod;
6 import org.springframework.web.bind.annotation.RestController;
7
8 @RestController no usages
9 public class HelloController {
10
11     // @RequestMapping(value = "/", method = RequestMethod.GET)
12     @GetMapping("/")
13     public String helloWorld(){
14         return "Hello World!!!";
15     }
16 }
```

The run tab at the bottom shows the application's log output:

```
↑ 2024-04-09T12:05:19.719+05:30 INFO 1836 --- [main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
↓ 2024-04-09T12:05:19.720+05:30 INFO 1836 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 843 ms
→ 2024-04-09T12:05:20.032+05:30 INFO 1836 --- [main] o.s.w.e.tomcat.TomcatWebServer : Tomcat started on port 8082 (http) with context path ''
→ 2024-04-09T12:05:20.041+05:30 INFO 1836 --- [main] c.l.S.SpringBootTest : Started SpringBootRestTestApplication in 1.549 seconds (process time)
→ 2024-04-09T12:05:22.096+05:30 INFO 1836 --- [nio-8082-exec-1] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring DispatcherServlet 'dispatcherServlet'
→ 2024-04-09T12:05:22.096+05:30 INFO 1836 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
→ 2024-04-09T12:05:22.096+05:30 INFO 1836 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 0 ms
```

REST API



Controller Layer for (RestAPI's)
Service Layer(Business Logics)
Repository Layer(Interaction with Database)(Reference to DB)(SpringDataJPA Dependency)
DataBaseInstantiate DataBase Connection)(Database configuration should be added)

Spring Data JPA, part of the larger **Spring Data** family, makes it easy to easily implement JPA-based (Java Persistence API) repositories. It makes it easier to build Spring-powered applications that use data access technologies.

Setting configuration.

The screenshot shows a Java project structure in a code editor. The project includes a .mvn folder, a src directory containing main, java, resources, static, and templates, and a test, target, .gitignore, HELP.md, mvnw, mvnw.cmd, and pom.xml file. The application.properties file is selected in the resources folder. The pom.xml file is open in the top right, showing Spring Boot configuration:

```
server.port=8082
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

1 entity model with getters and setters

The screenshot shows the same Java project structure. The Department.java file is selected in the editor. It contains the following code:

```
package com.LearnSpringBoot.SpringBootRESTTest.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
@Entity
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long departmentId;
    private String departmentName;
    private String departmentAddress;
    private String departmentCode;
    public Department() {
    }
    @Override
    public String toString() {
        return "Department{" + "departmentId=" + departmentId + ", departmentName=" +
               departmentName + ", departmentAddress=" + departmentAddress + ", departmentCode=" + departmentCode;
    }
    public Long getDepartmentId() {
        return departmentId;
    }
    public void setDepartmentId(Long departmentId) {
    }
}
```

2 Repository Layer(Interface) (JPA module extension)

Project ▾

- > .mvn
- > src
 - └ main
 - └ java
 - └ com.LearnSpringBoot.SpringBootRESTTest
 - └ controller
 - DepartmentController
 - HelloController
 - └ entity
 - Department
 - └ repository
 - DepartmentRepository
- > service
 - └ SpringBootRestTestApplication
- > resources
 - └ static
 - └ templates
 - application.properties
- > test
- > target
- ∅ .gitignore
- M HELP.md
- mvnw
- mvnw.cmd

DepartmentRepository.java x DepartmentServiceImpl.java pom.xml (Spring-Boot-REST-Test) application

```
1 package com.LearnSpringBoot.SpringBootRESTTest.repository;
2
3 import com.LearnSpringBoot.SpringBootRESTTest.entity.Department;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository no usages
8 public interface DepartmentRepository extends JpaRepository<Department, Long> {
9 }
```

3 Controller Layer (Define End points and service layers function calls)

Project ▾

- > .mvn
- > src
 - └ main
 - └ java
 - └ com.LearnSpringBoot.SpringBootRESTTest
 - └ controller
 - DepartmentController
 - HelloController
 - └ entity
 - Department
 - └ repository
 - DepartmentRepository
- > service
 - └ SpringBootRestTestApplication
- > resources
 - └ static
 - └ templates
 - application.properties
- > test
- > target
- ∅ .gitignore
- M HELP.md
- mvnw
- mvnw.cmd
- M pom.xml
- > target
- > External Libraries
- Scatches and Consoles

DepartmentController.java x DepartmentService.java DepartmentRepository.java DepartmentServiceImpl.java pom.xml application

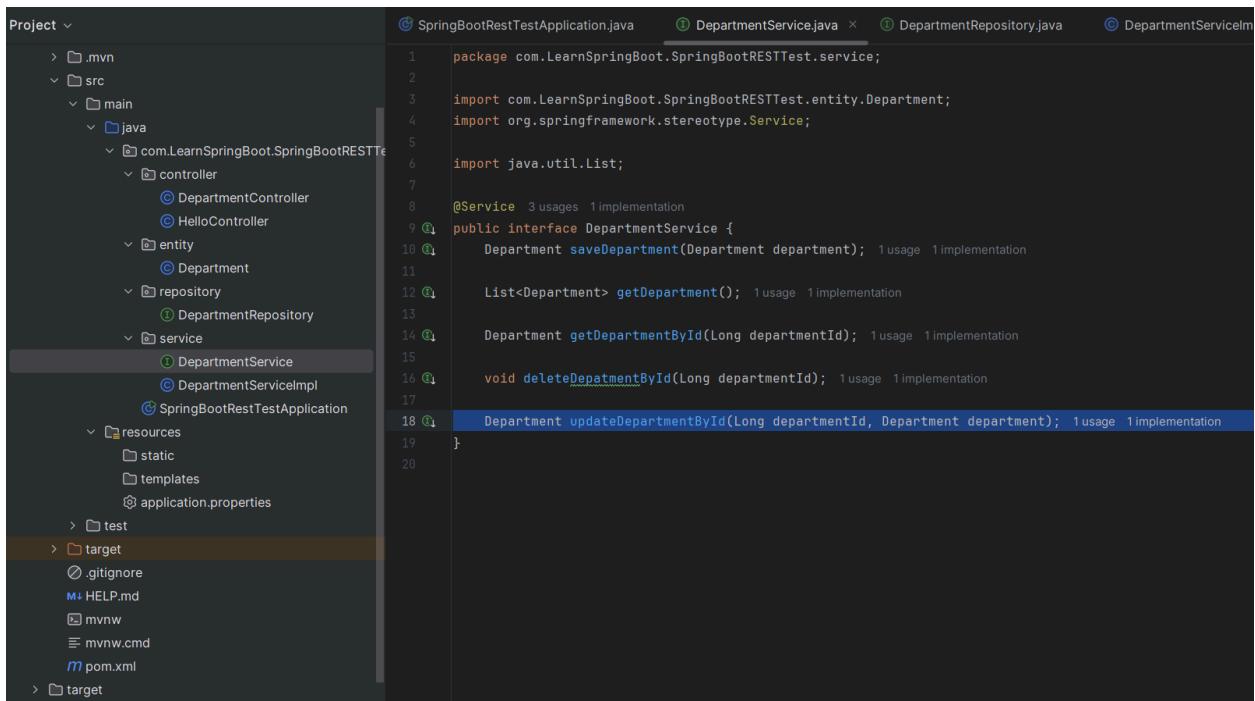
```
11 @RestController no usages
12 public class DepartmentController {
13
14     @Autowired 5 usages
15     private DepartmentService departmentService;
16
17     @PostMapping("/departments") no usages
18     public Department saveDepartment(@RequestBody Department department){
19         return departmentService.saveDepartment(department);
20     }
21
22     @GetMapping("/getdepartments") no usages
23     public List<Department> getDepartment(){
24         return departmentService.getDepartment();
25     }
26
27     @GetMapping("/getdepartment/{id}") no usages
28     public Department getDepartmentById(@PathVariable("id") Long departmentId){
29         return departmentService.getDepartmentById(departmentId);
30     }
31
32     @DeleteMapping("/deletedepartment/{Id}") no usages
33     public String deleteDepartmentById(@PathVariable("Id") Long departmentId){
34         departmentService.deleteDepartmentById(departmentId);
35         return "Department deleted successfully!";
36     }
37
38     @PutMapping("/departments/{Id}") no usages
39     public Department updateDepartment(@PathVariable("Id") Long departmentId,@RequestBody Department department){
40         return departmentService.updateDepartmentById(departmentId, department);
41     }
42 }
```

```

36
37     @Override 1 usage
38     public Department updateDepartmentById(Long departmentId, Department department) {
39         Department depDB = departmentRepository.findById(departmentId).get();
40
41         if(Objects.nonNull(department.getDepartmentName()) &&
42             !"".equalsIgnoreCase(department.getDepartmentName())){
43             depDB.setDepartmentName(department.getDepartmentName());
44         }
45
46         if(Objects.nonNull(department.getDepartmentAddress()) &&
47             !"".equalsIgnoreCase(department.getDepartmentAddress())){
48             depDB.setDepartmentAddress(department.getDepartmentAddress());
49         }
50
51         if(Objects.nonNull(department.getDepartmentCode()) &&
52             !"".equalsIgnoreCase(department.getDepartmentCode())){
53             depDB.setDepartmentCode(department.getDepartmentCode());
54         }
55
56         return departmentRepository.save(depDB);
57     }
58

```

4 Service Layer all the logic and implementation



The screenshot shows the Eclipse IDE interface with the following details:

- Project View:** On the left, it shows the project structure under "Project". The "src" folder contains "main" which has "java", "resources", and "test". "java" contains "com.LearnSpringBoot.SpringBootRESTTest" which has "controller", "entity", "repository", "service", and "SpringBootRestTestApplication". "repository" contains "DepartmentRepository". "service" contains "DepartmentService" and "DepartmentServiceImpl". "resources" contains "static", "templates", and "application.properties".
- Editor View:** On the right, the code editor displays the `DepartmentService.java` file. The code is as follows:

```

1 package com.LearnSpringBoot.SpringBootRESTTest.service;
2
3 import com.LearnSpringBoot.SpringBootRESTTest.entity.Department;
4 import org.springframework.stereotype.Service;
5
6 import java.util.List;
7
8 @Service 3 usages 1 implementation
9 public interface DepartmentService {
10     Department saveDepartment(Department department); 1 usage 1 implementation
11
12     List<Department> getDepartment(); 1 usage 1 implementation
13
14     Department getDepartmentById(Long departmentId); 1 usage 1 implementation
15
16     void deleteDepartmentById(Long departmentId); 1 usage 1 implementation
17
18     Department updateDepartmentById(Long departmentId, Department department); 1 usage 1 implementation
19
20 }

```

The screenshot shows a Java project structure on the left and a code editor on the right. The project structure includes .mvn, src (with main, java, resources, static, templates), test, target, .gitignore, HELP.md, mvnw, mvnw.cmd, pom.xml, and External Libraries. The code editor displays `DepartmentServiceImpl.java` with the following content:

```
11  @Service 1 usage
12  public class DepartmentServiceImpl implements DepartmentService{
13
14      @Autowired 6 usages
15      private DepartmentRepository departmentRepository;
16
17      @Override 1 usage
18      public Department saveDepartment(Department department) {
19          return departmentRepository.save(department);
20      }
21
22      @Override 1 usage
23      public List<Department> getDepartment() {
24          return departmentRepository.findAll();
25      }
26
27      @Override 1 usage
28      public Department getDepartmentById(Long departmentId) {
29          return departmentRepository.findById(departmentId).get();
30      }
31
32      @Override 1 usage
33      public void deleteDepartmentById(Long departmentId) {
34          departmentRepository.deleteById(departmentId);
35      }
36
37      @Override 1 usage
38      public Department updateDepartmentById(Long departmentId, Department department) {
39          Department depDB = departmentRepository.findById(departmentId).get();
40
41          if(Objects.nonNull(department.getDepartmentName()) &&
42              !"".equalsIgnoreCase(department.getDepartmentName())){
43
44      }
45
46      }
47
48      }
49
50      }
51
52      }
53
54      }
55
56      }
57
58      }
59
59  }
```