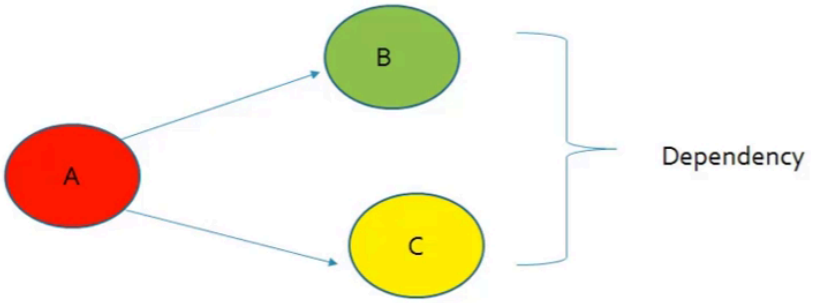
WAR file (Web Archive)
Spring Boot Framework
Dependecies Injection

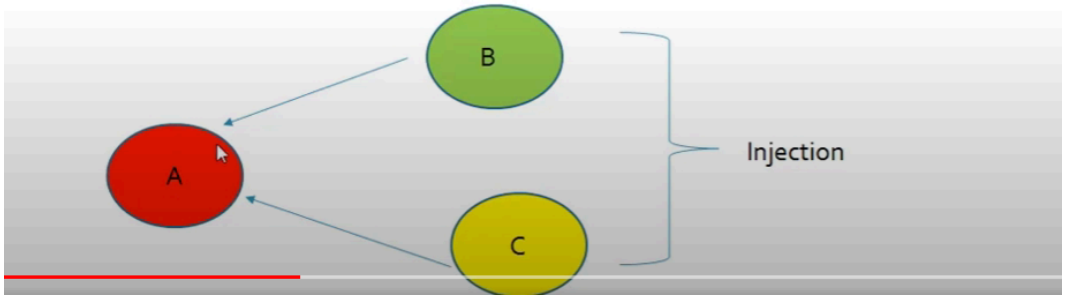
1 Inversion of Control
Giving control to frame work to do that spring allows dependency injection.

- 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

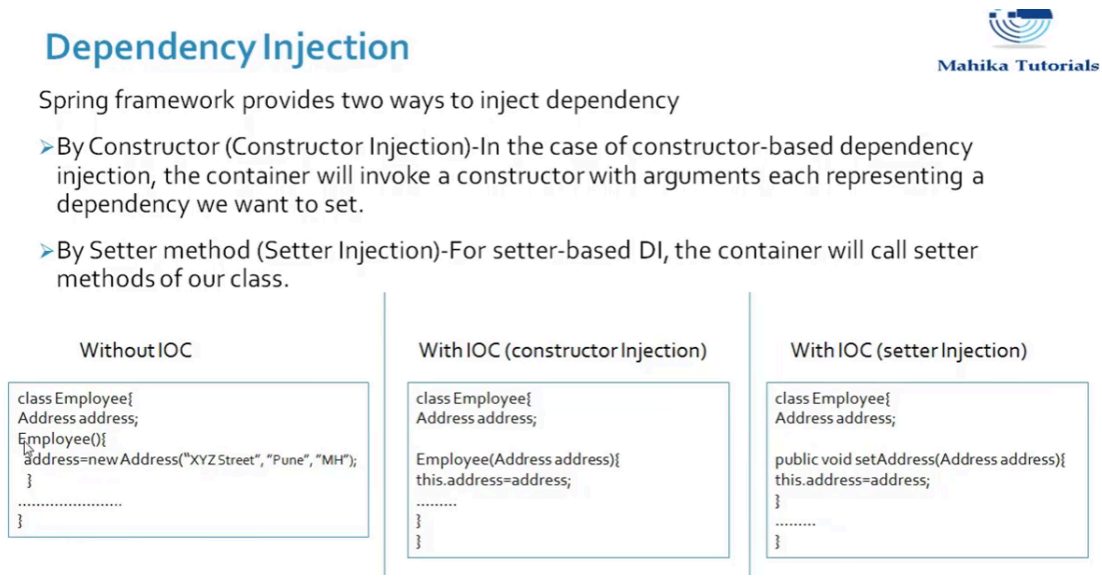




Dependency Injection and Inversion of Control

Mahika Tutorials

- 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.



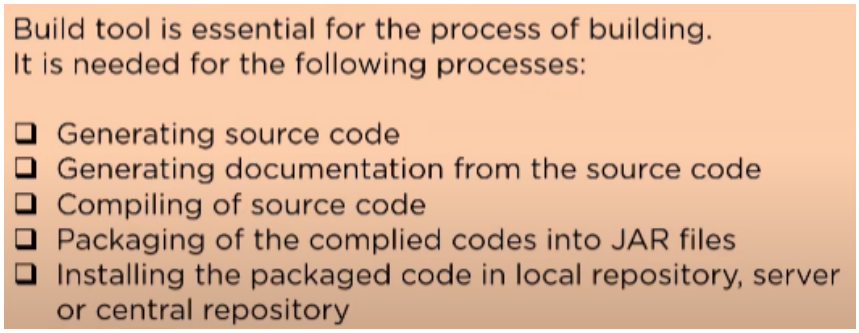
2 AOP(Aspect oriented programming)
Tackle down the cross cutting applications (repeatedly work)

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.



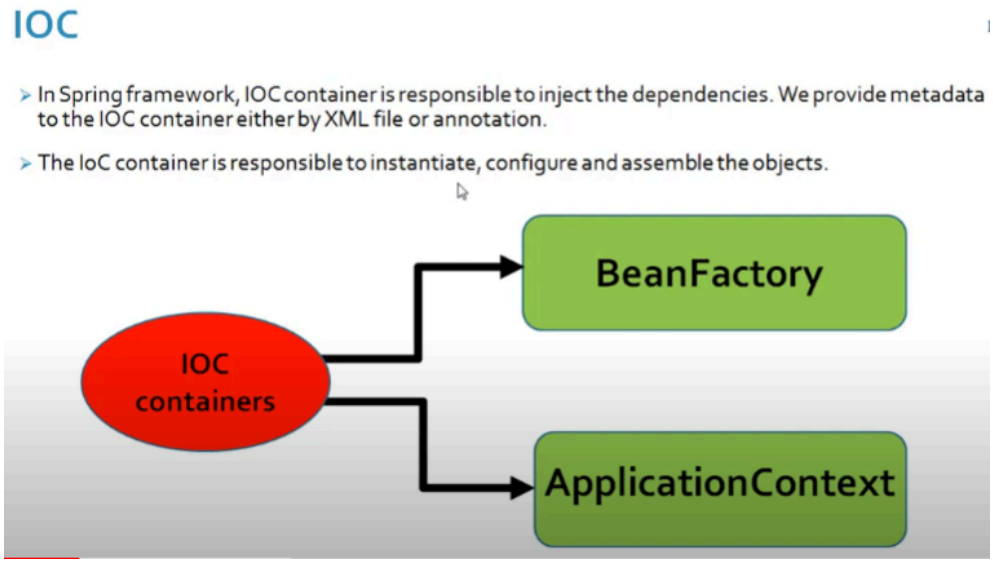
Maven mainly used for java-based projects, helps in downloading dependencieds, which refers to the libraries or **JAR files.**

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**

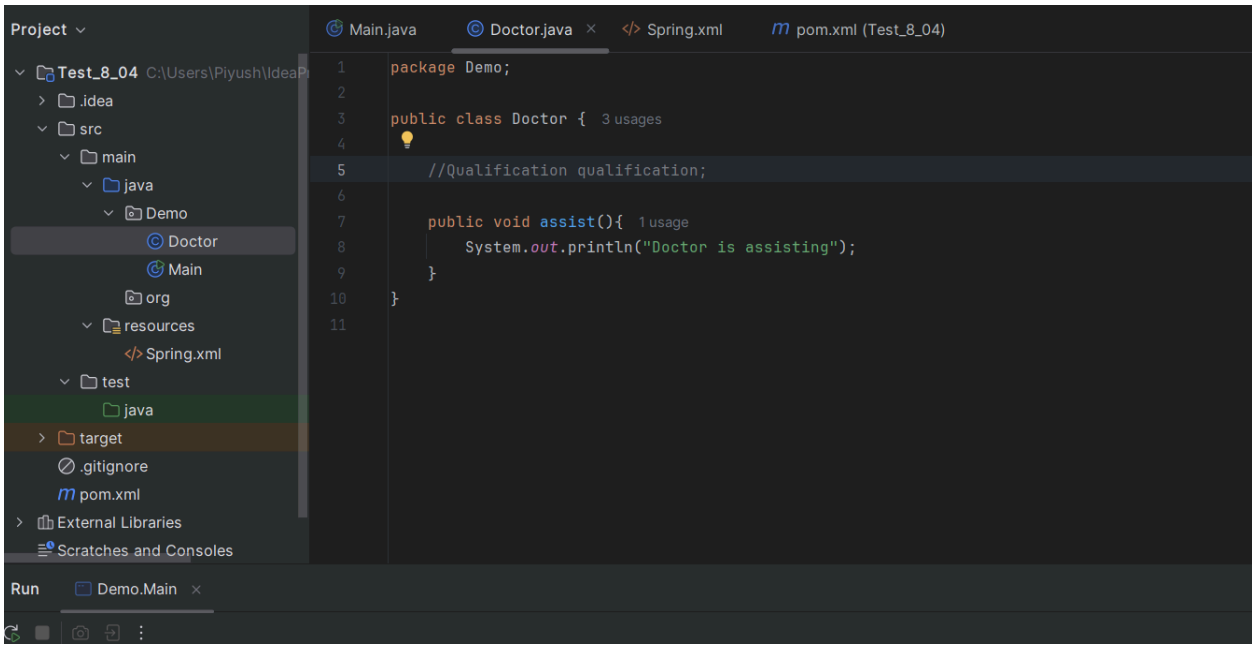
Dependency Injection

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

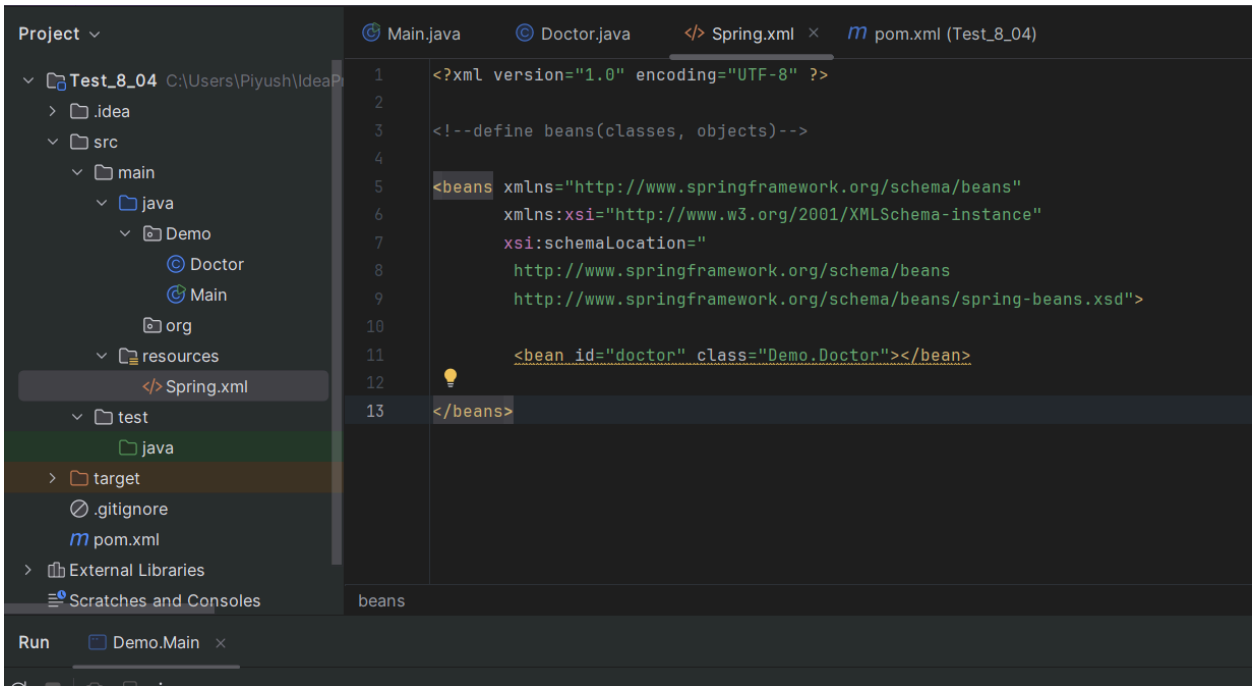


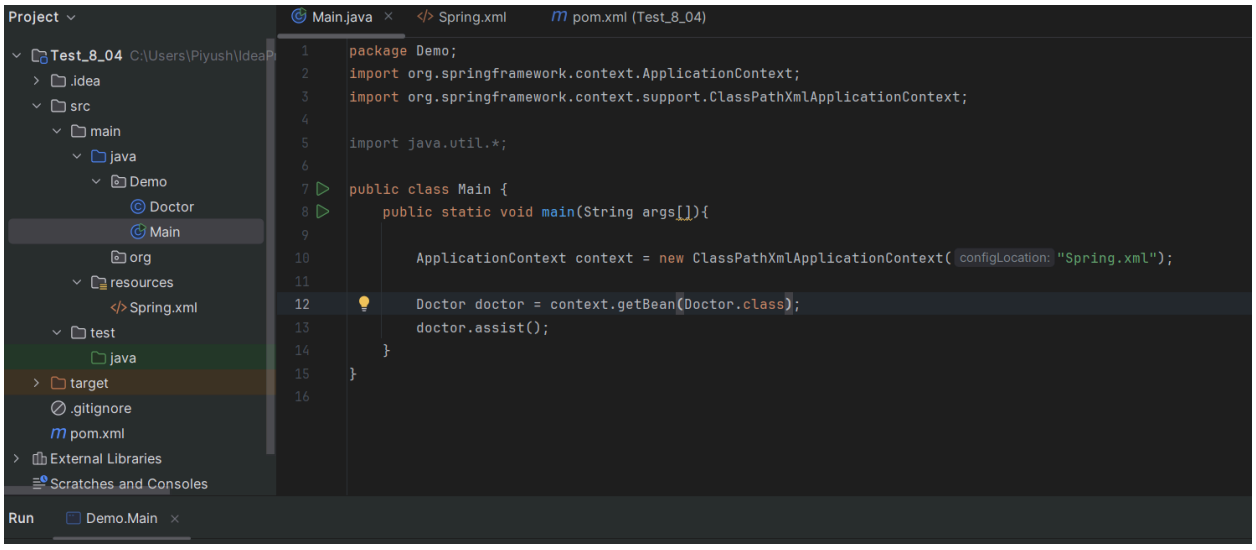
When ever spring starts we get beans from container
IOC(Inversion of control) has 2 interface **Beanfactory** and **ApplicationContext**
The BeanFactory provides the configuration framework and basic functionality, while the ApplicationContext adds enhanced capabilities to it.
clas
ApplicationContext extends beans factory . so there are lot more features in ApplicationContext
This try to get all beans

Class **ClassPathXMLApplicationContex**t implements the ApplicationContext interface

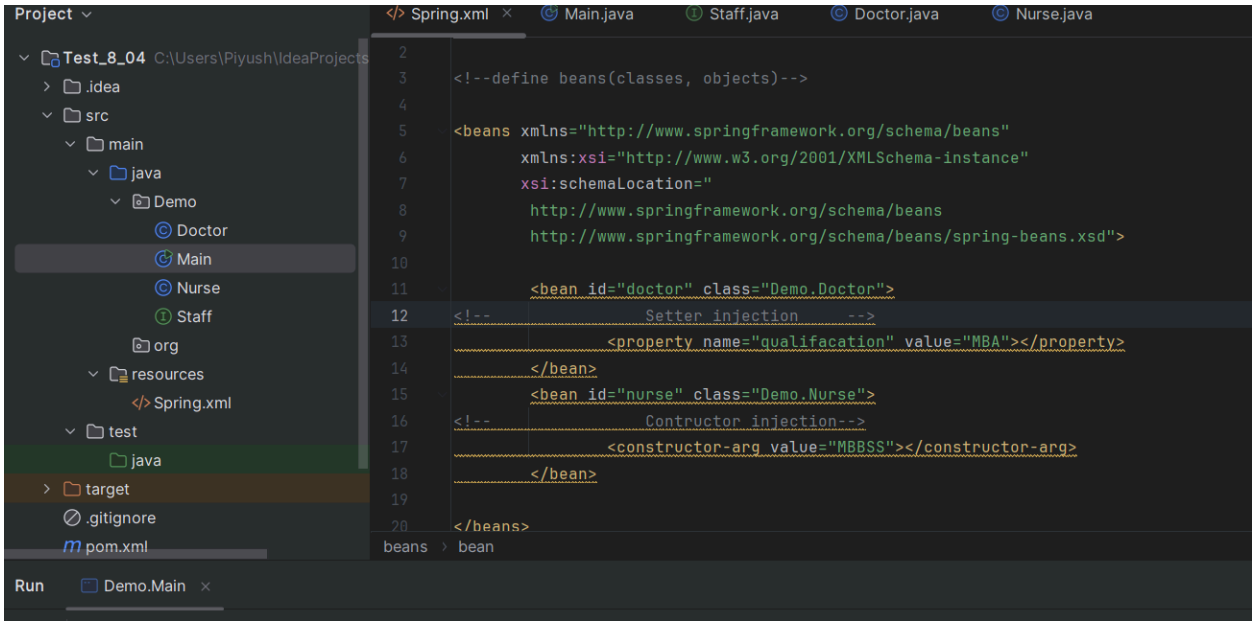


Defining xml file for Bean

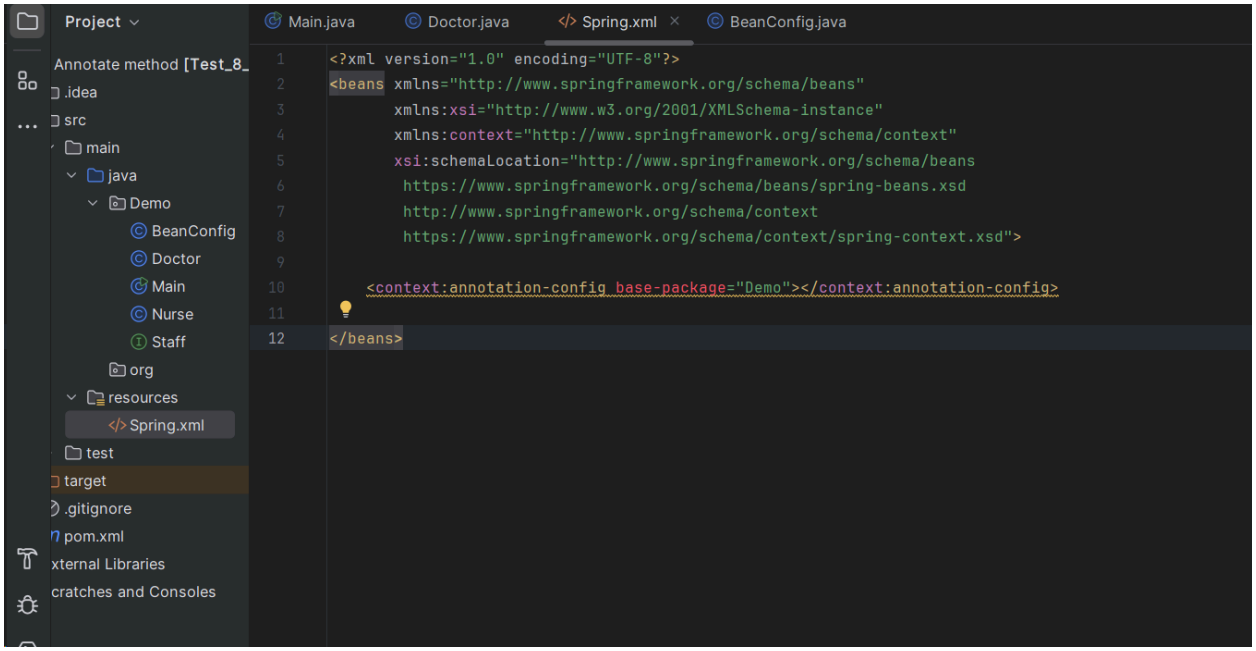


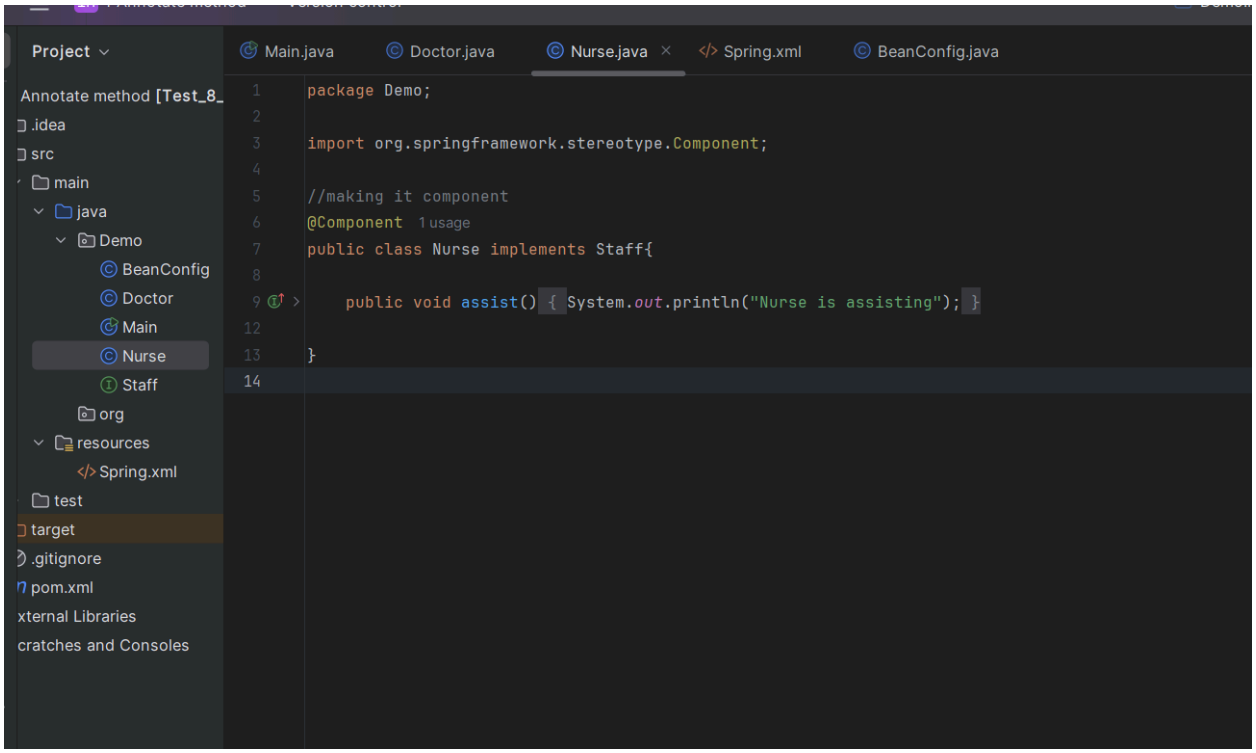


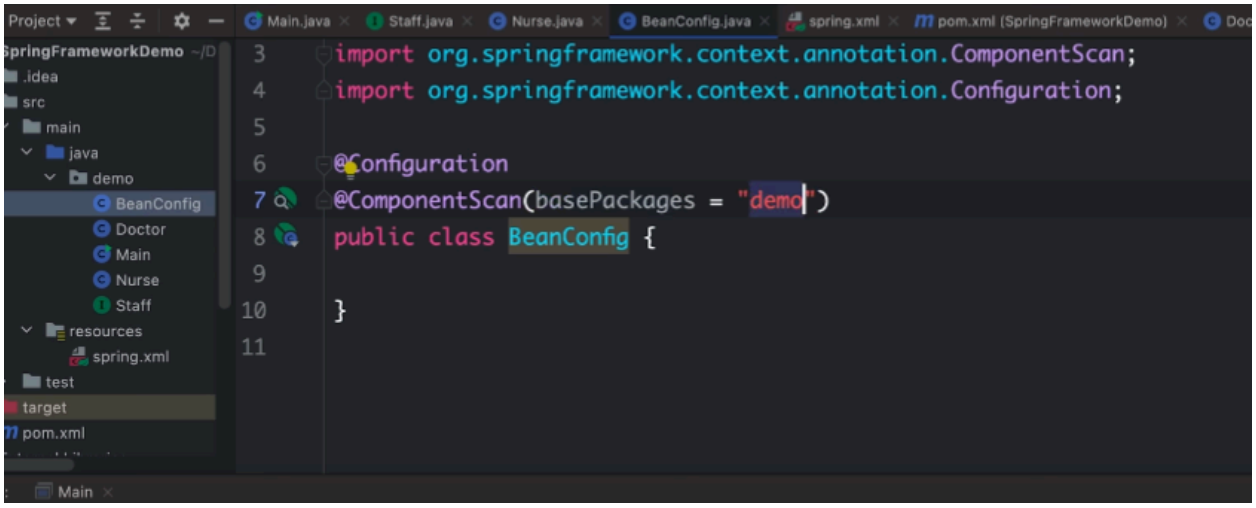
Setter Injection and Constructor Injection into beans

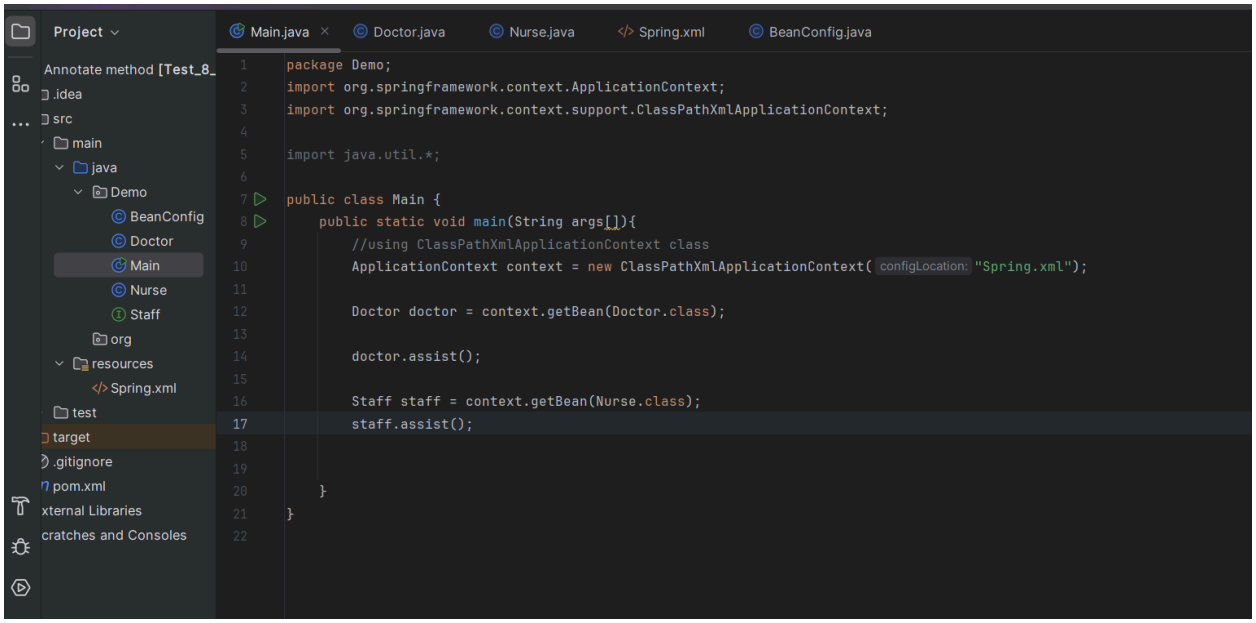


Using Component annotate



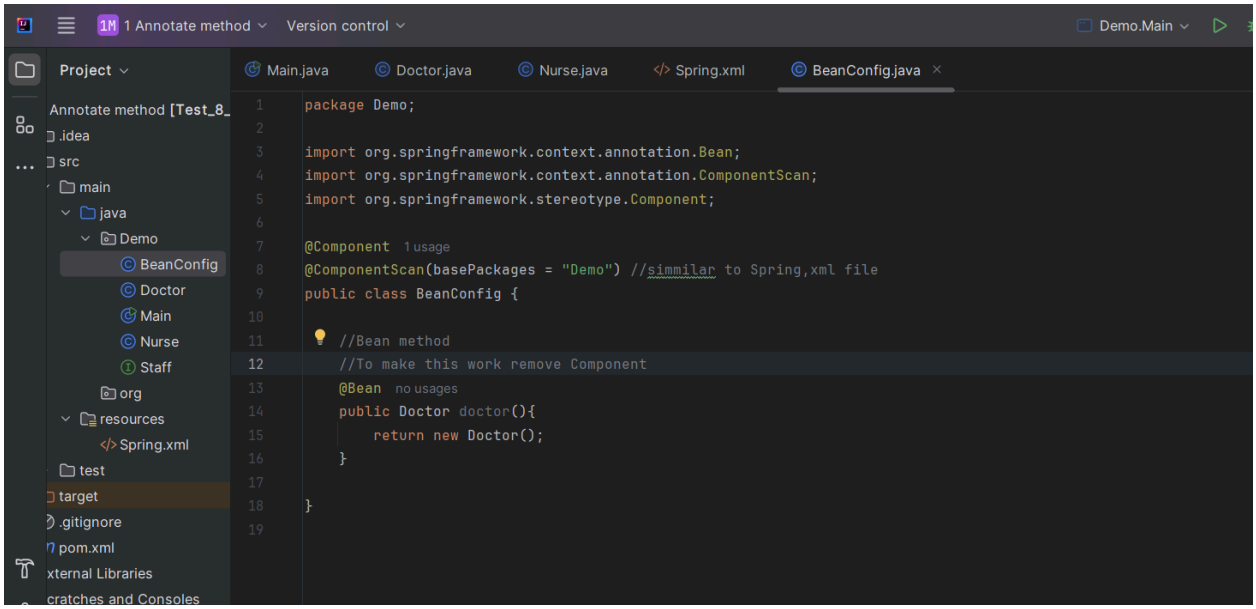


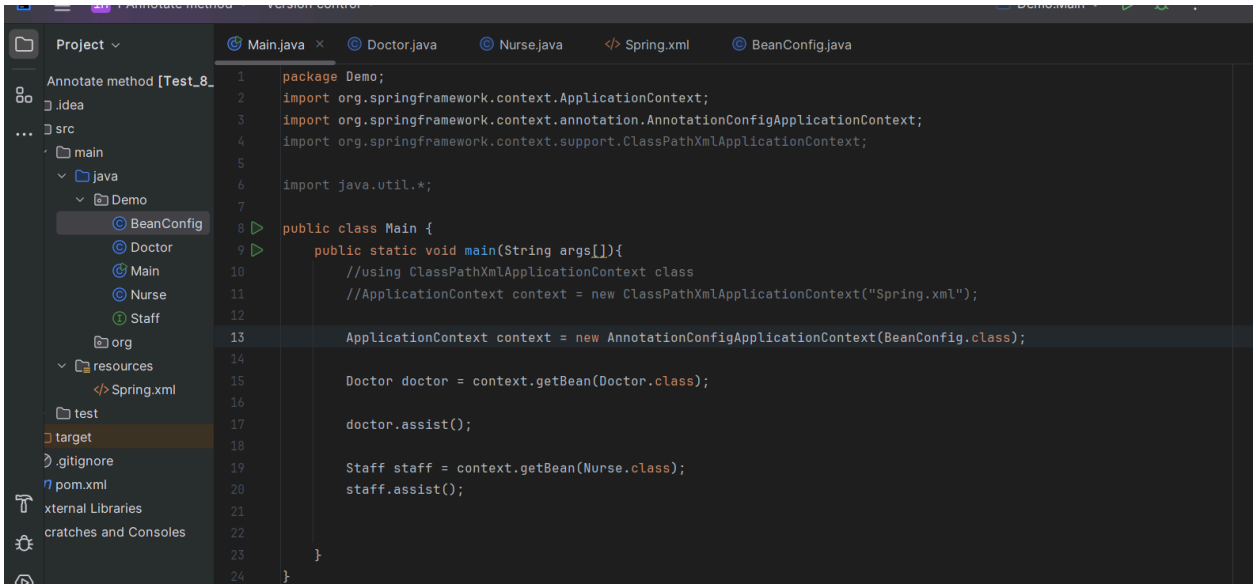




Here it will go to BeanConfig class, check what all package to be load where @Component is defined.

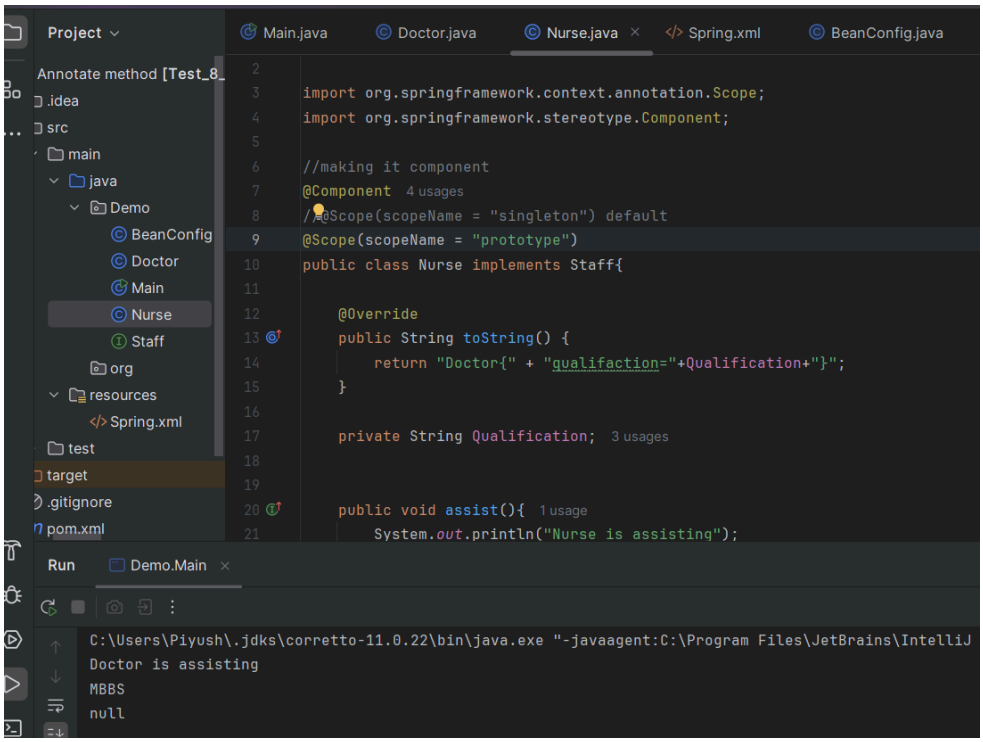
Defining in bean Class

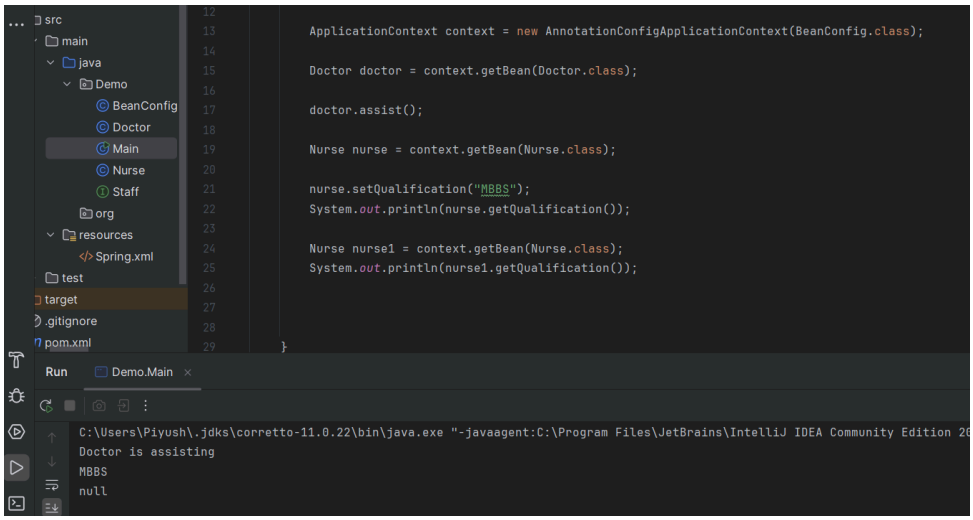




Types of scope

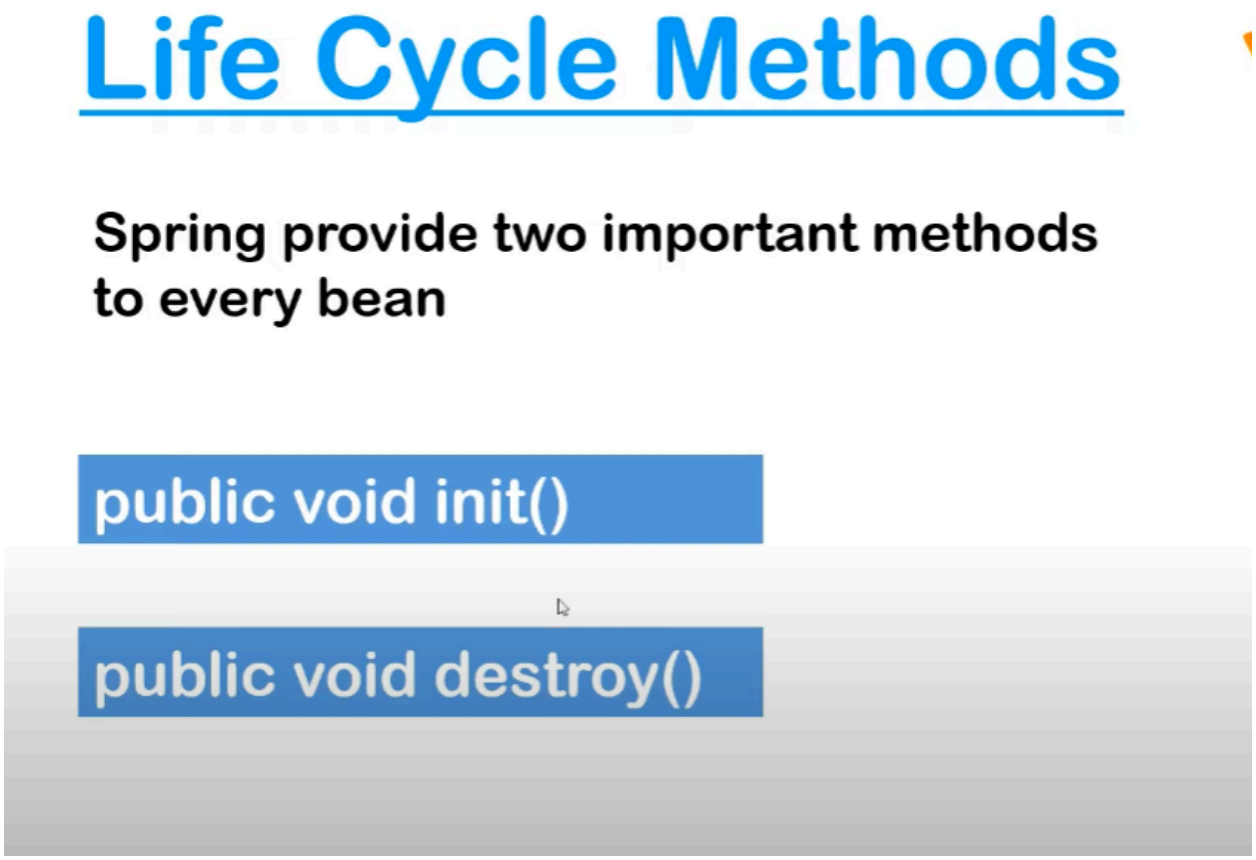
Singleton (default scope)(All access only one object)
Prototype (Different objects)





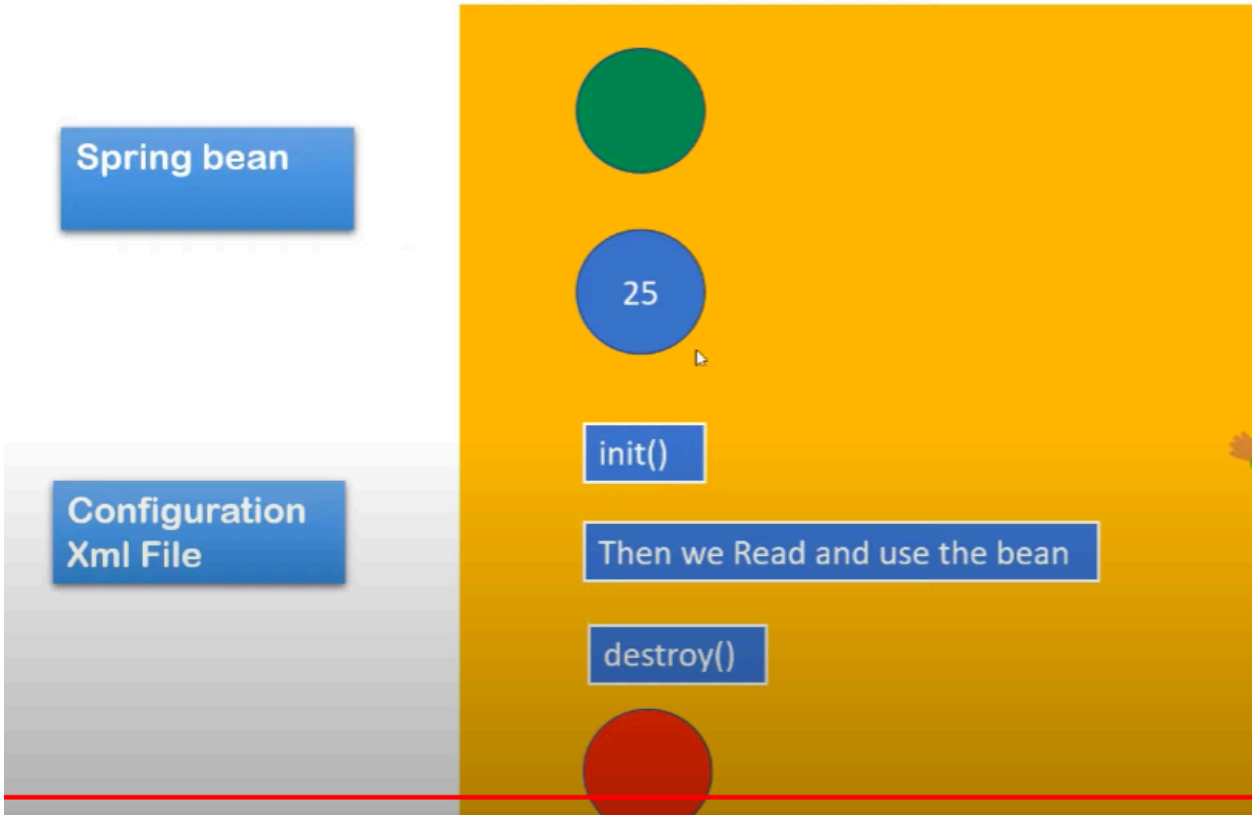
request
session
application
Websocket

Bean Life Cycle



Bean accepts xml file

Life Cycle



Configure Technique

- Xml
- Spring Interface
- Annotation

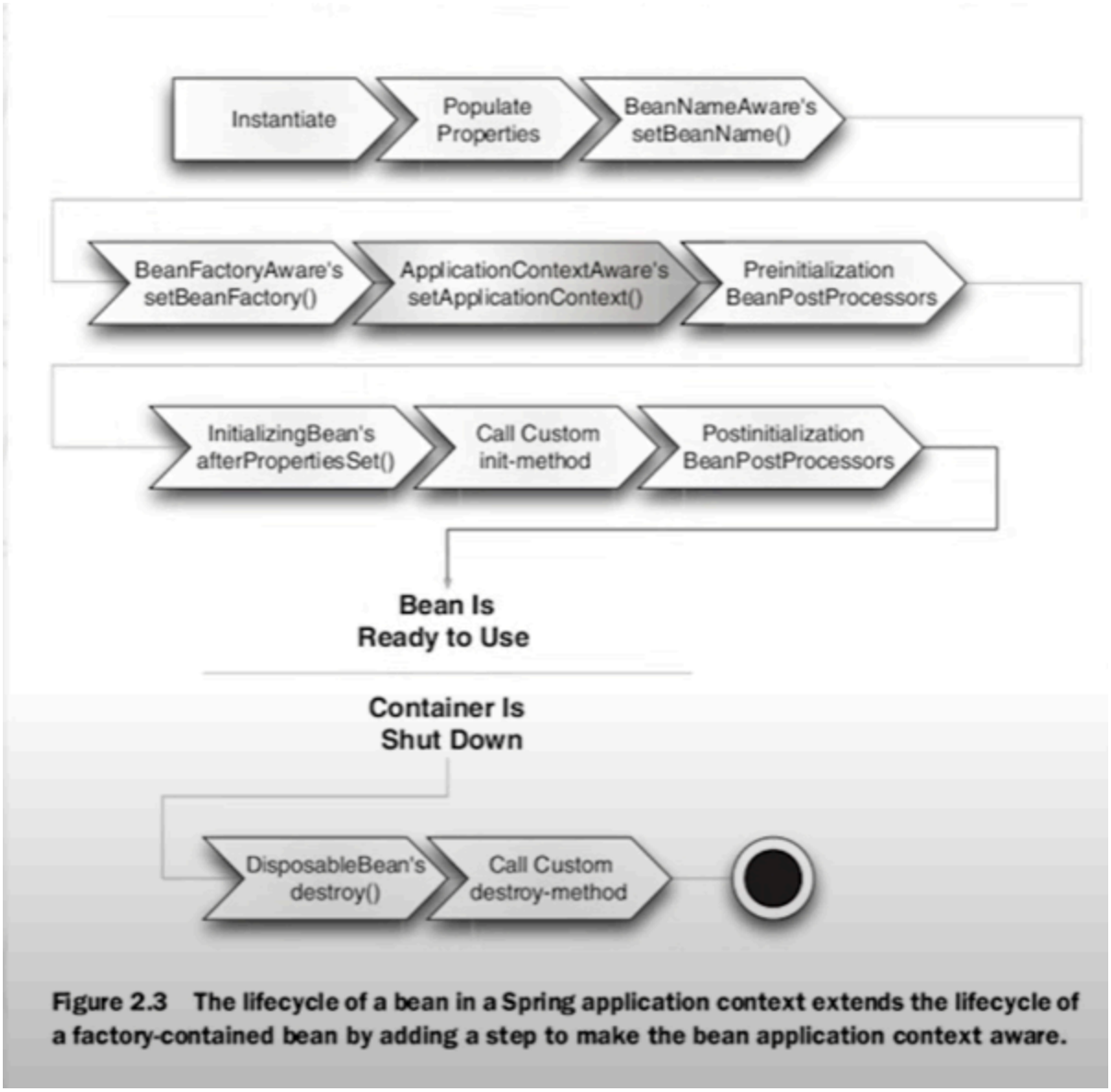
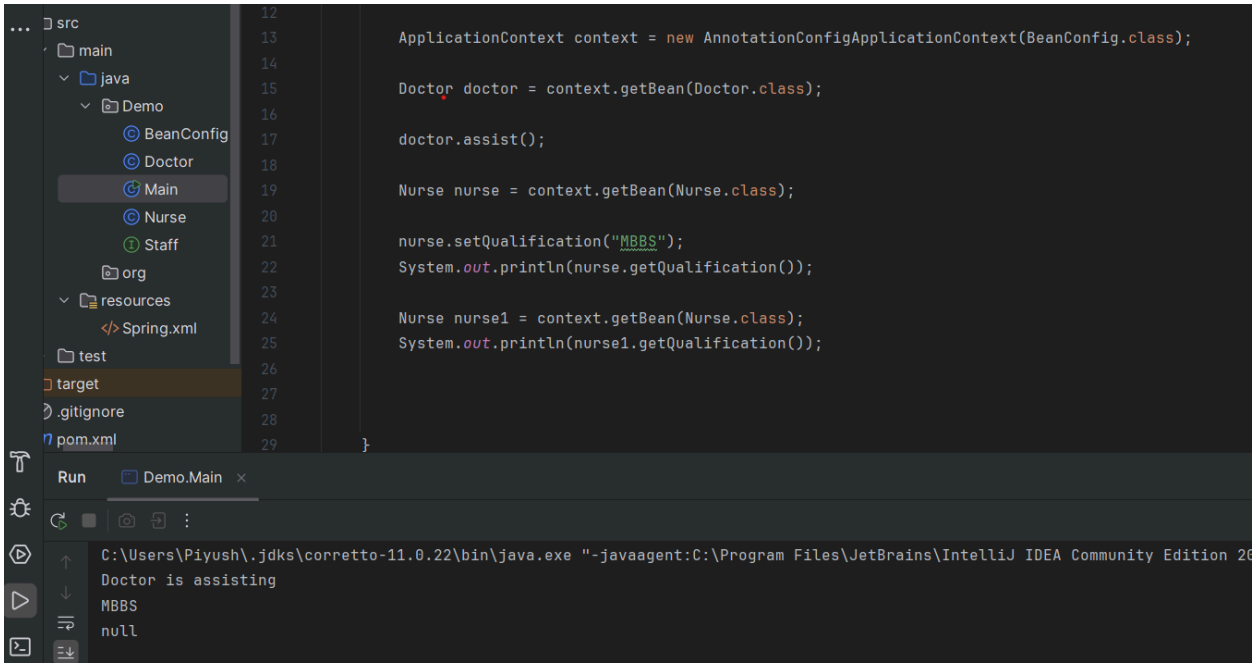
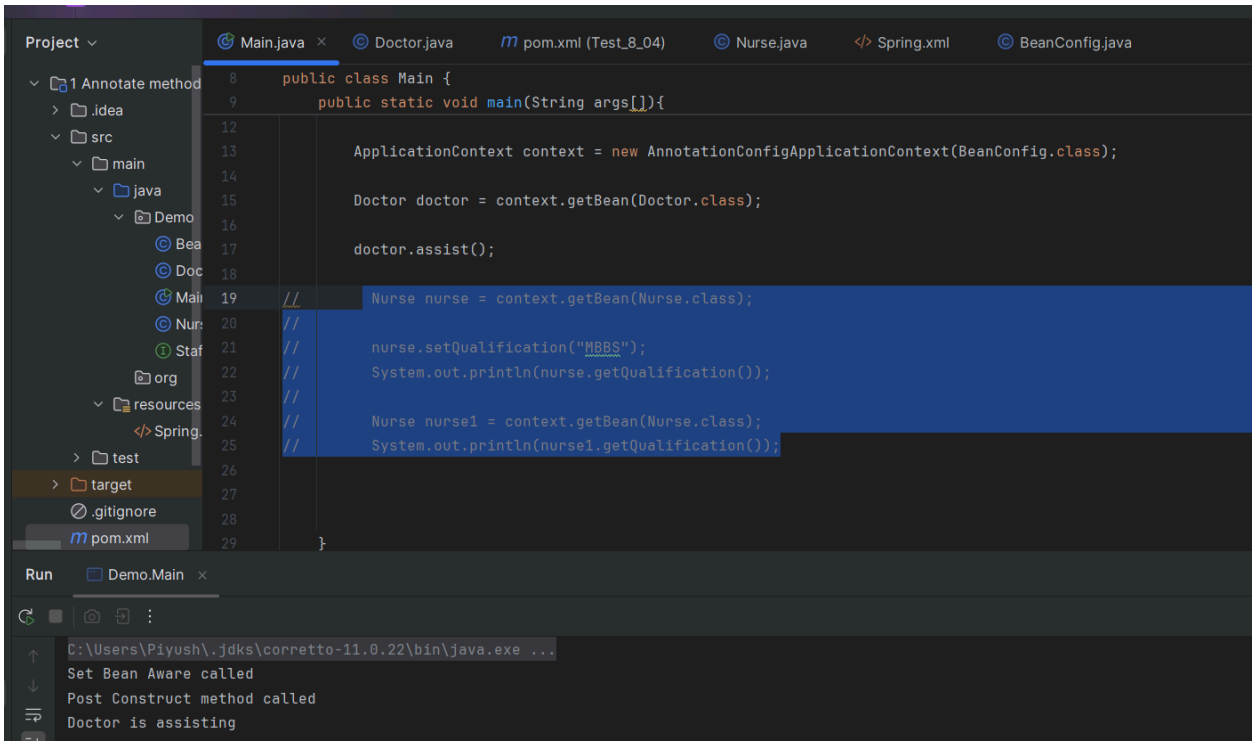


Figure 2.3 The lifecycle of a bean in a Spring application context extends the lifecycle of a factory-contained bean by adding a step to make the bean application context aware.

SetBeanMethod
Post construct





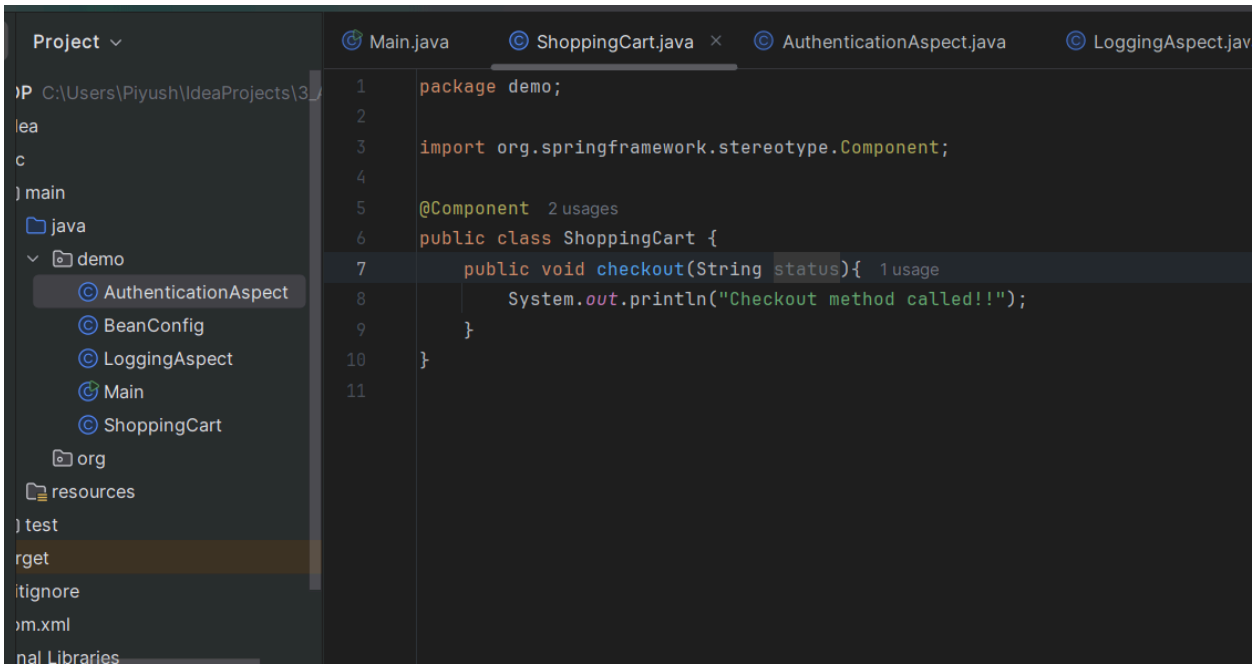
Bean then no Components

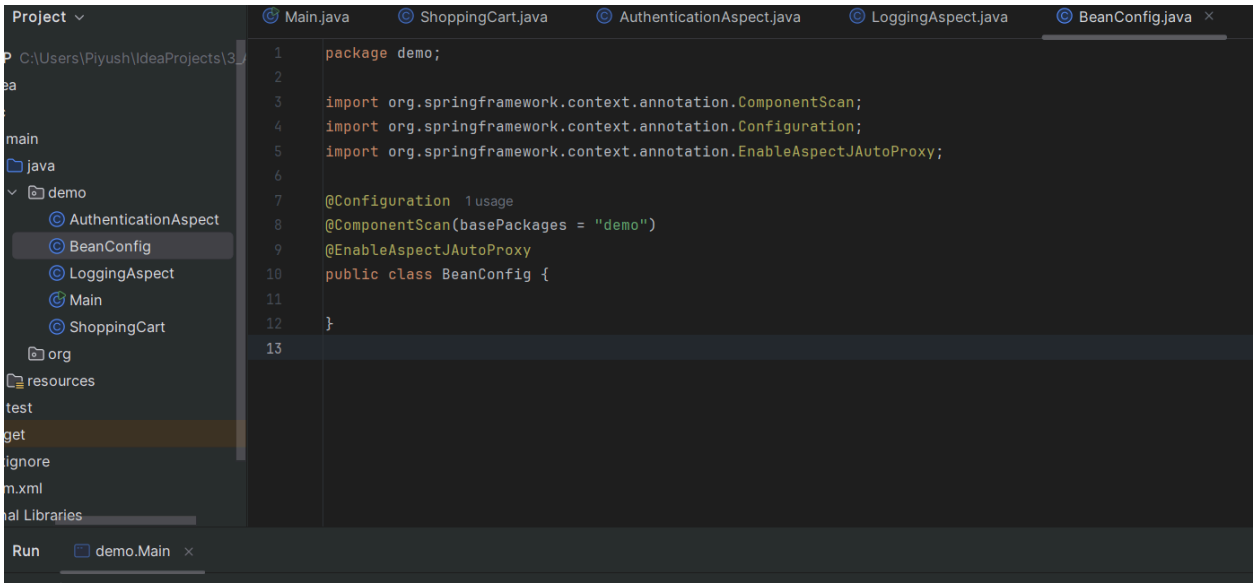
If Components then no need to define Bean

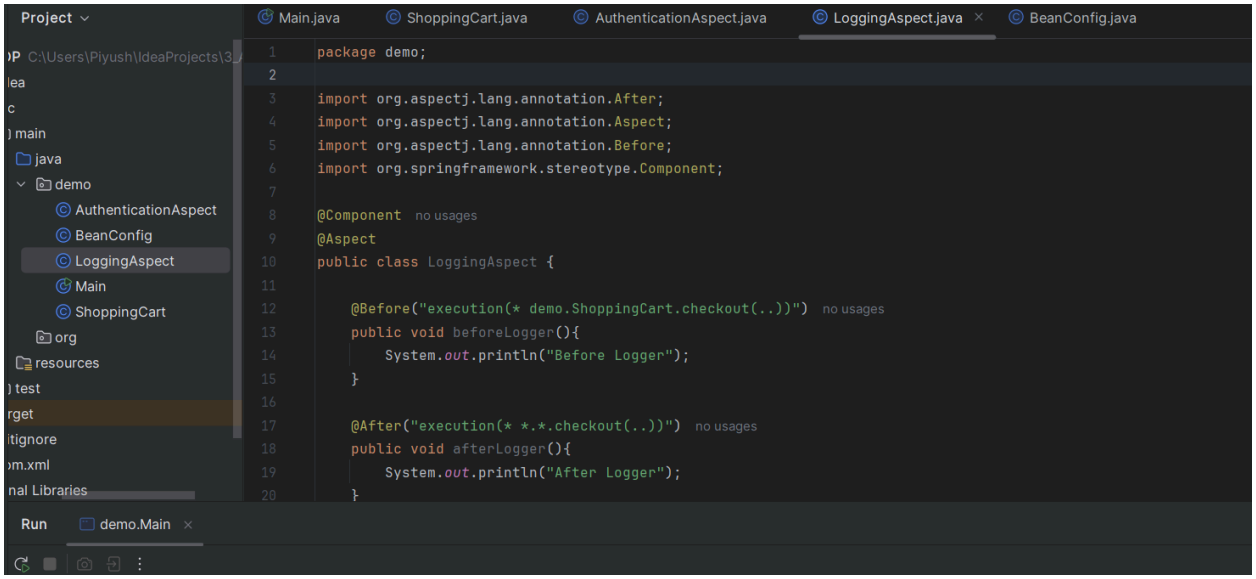
AOP

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

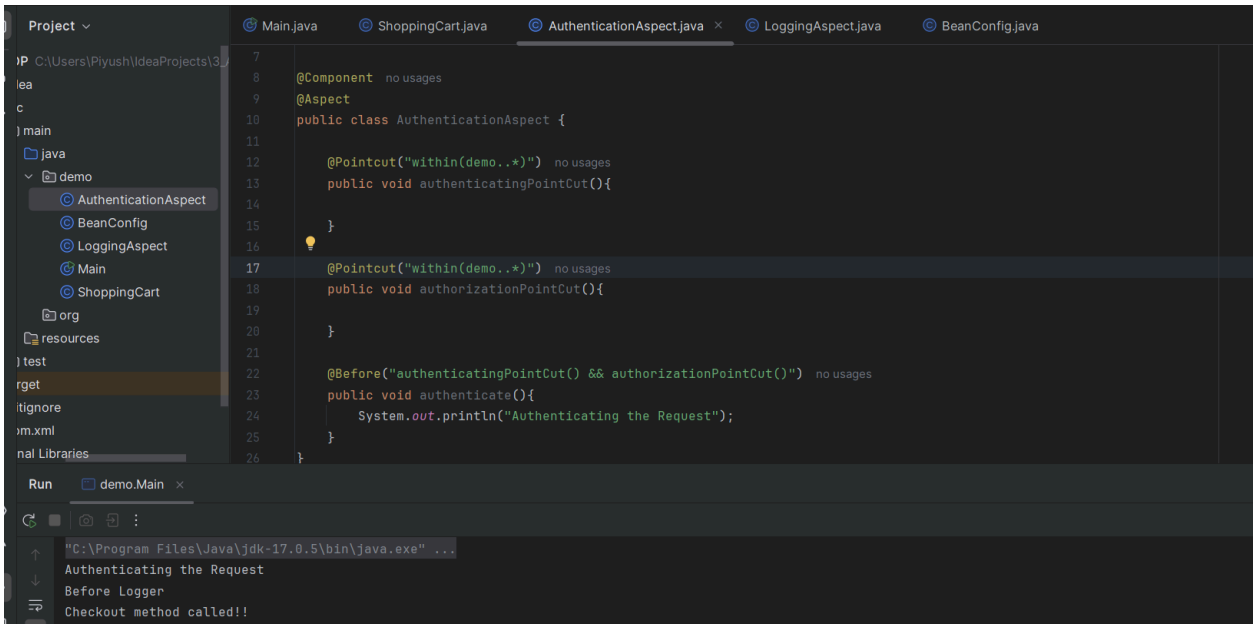
Loggers







Point cuts for authorization



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

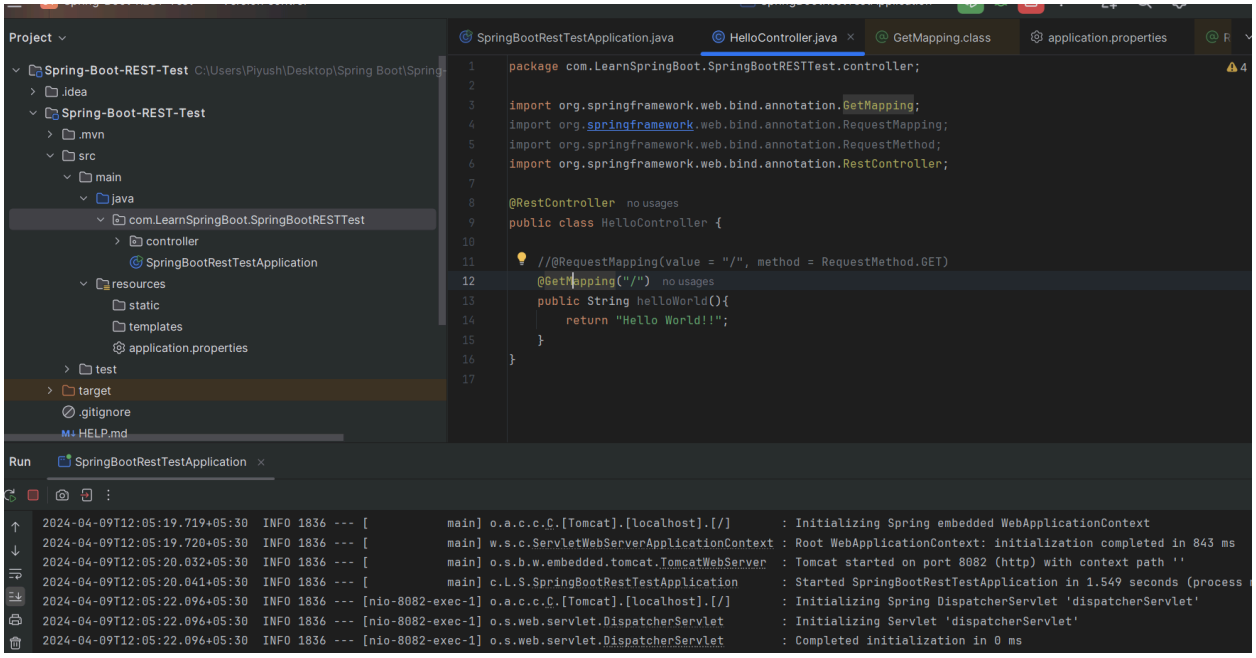
Starter Template For all(jdbc, test)

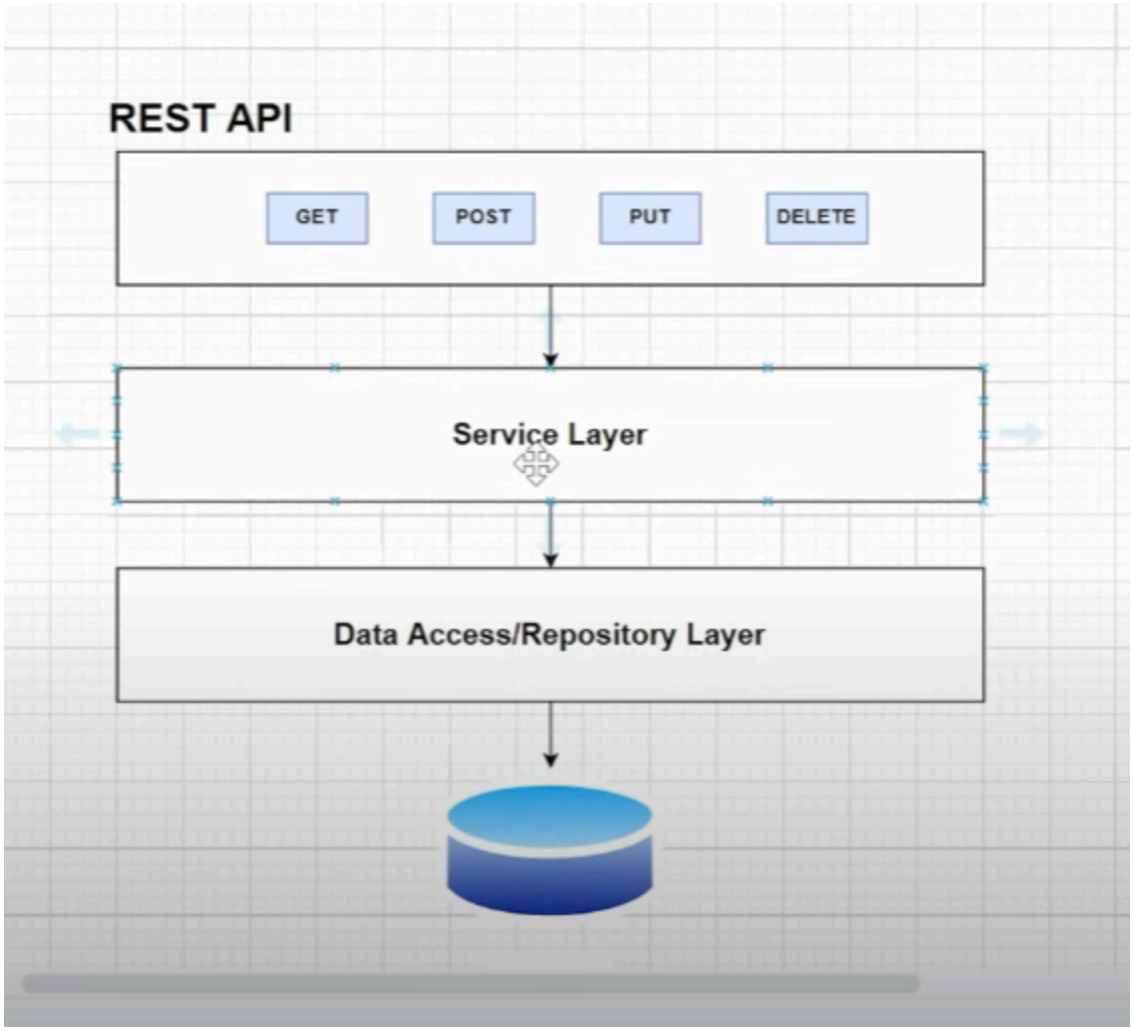
Auto Configuration

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.

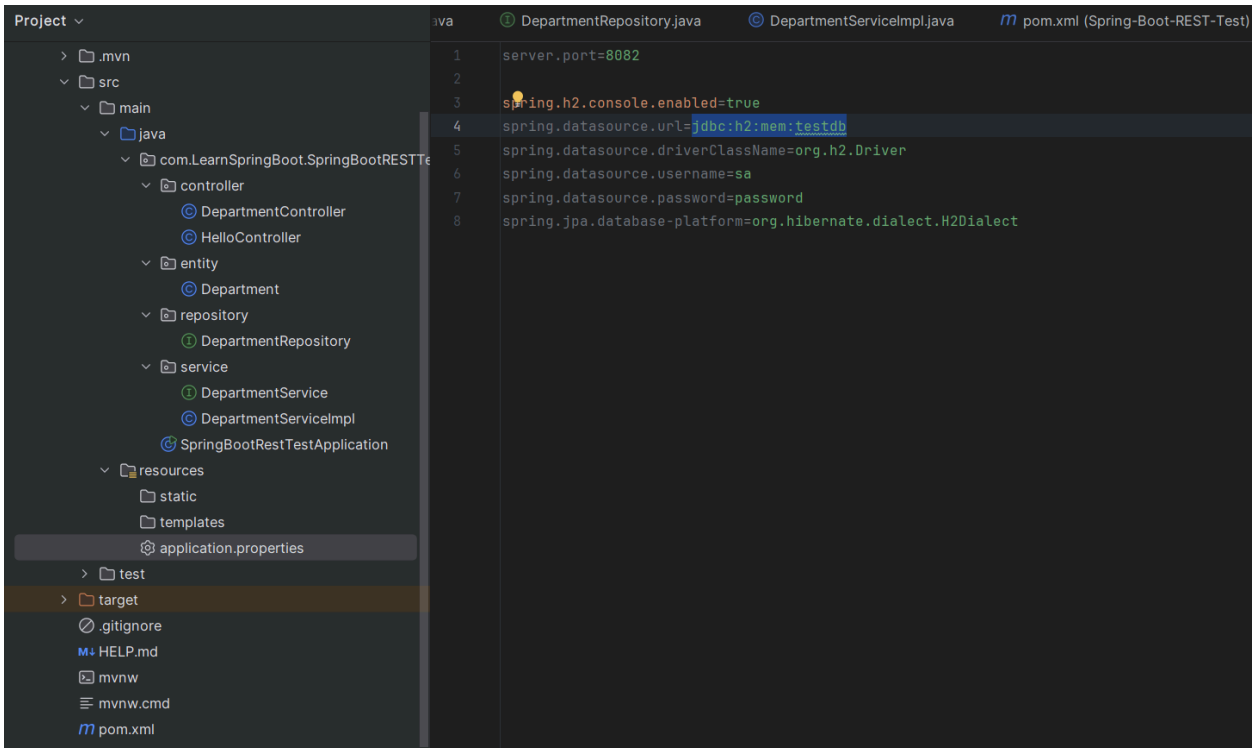




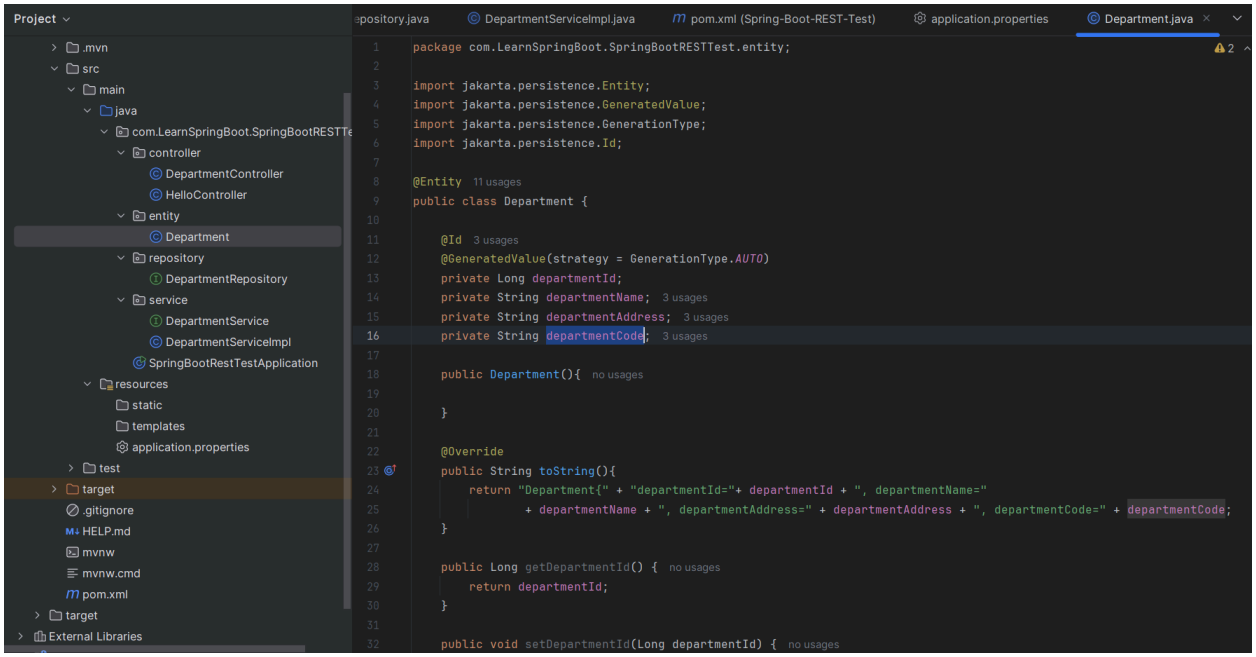
Controller Layer for (RestAPI's)
Service Layer(Business Logics)
Repository Layer(Interaction with Database)(Reference to DB)(SprigDataJPA Dependency)
DataBase(Instantiate DataBase Connection)(Database configuration should be added)

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.

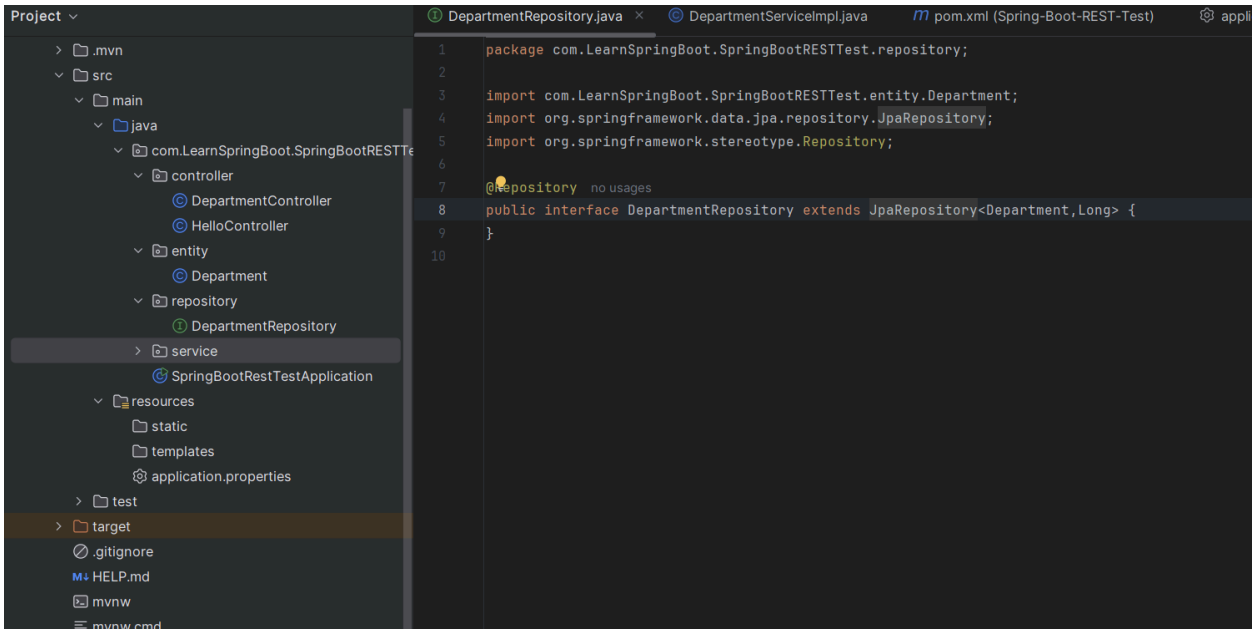
Setting configuration.



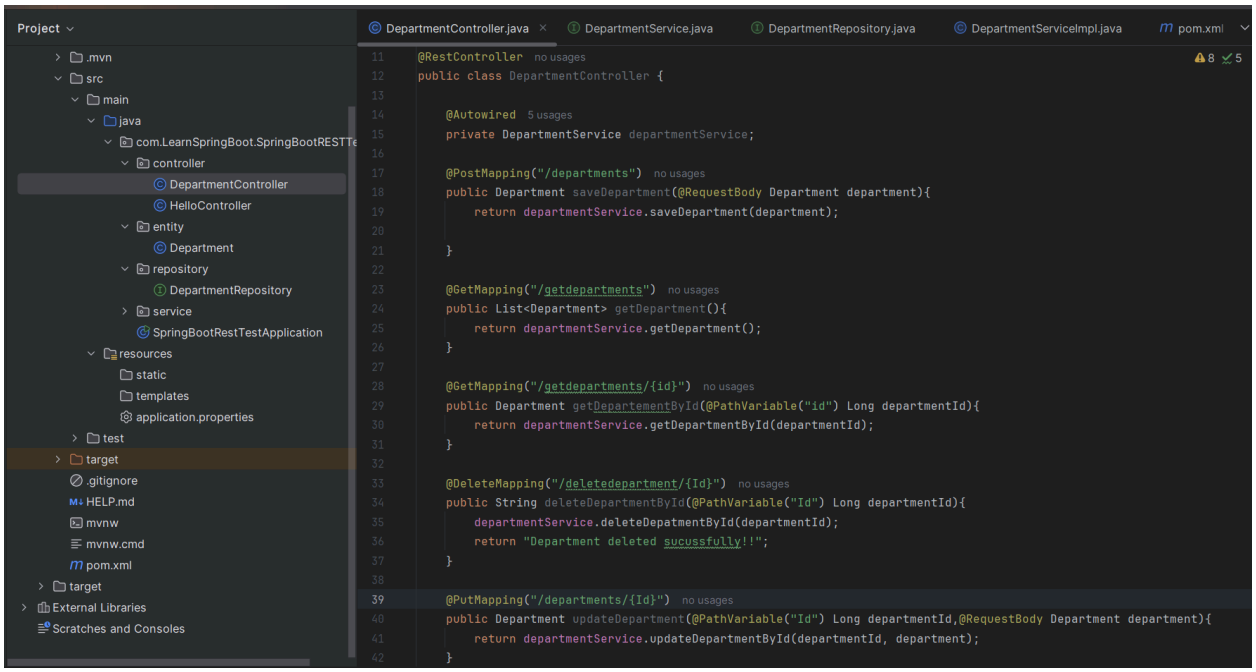
1 entity model with getters and setters



2 Repository Layer(Interface) (JPA module extension)

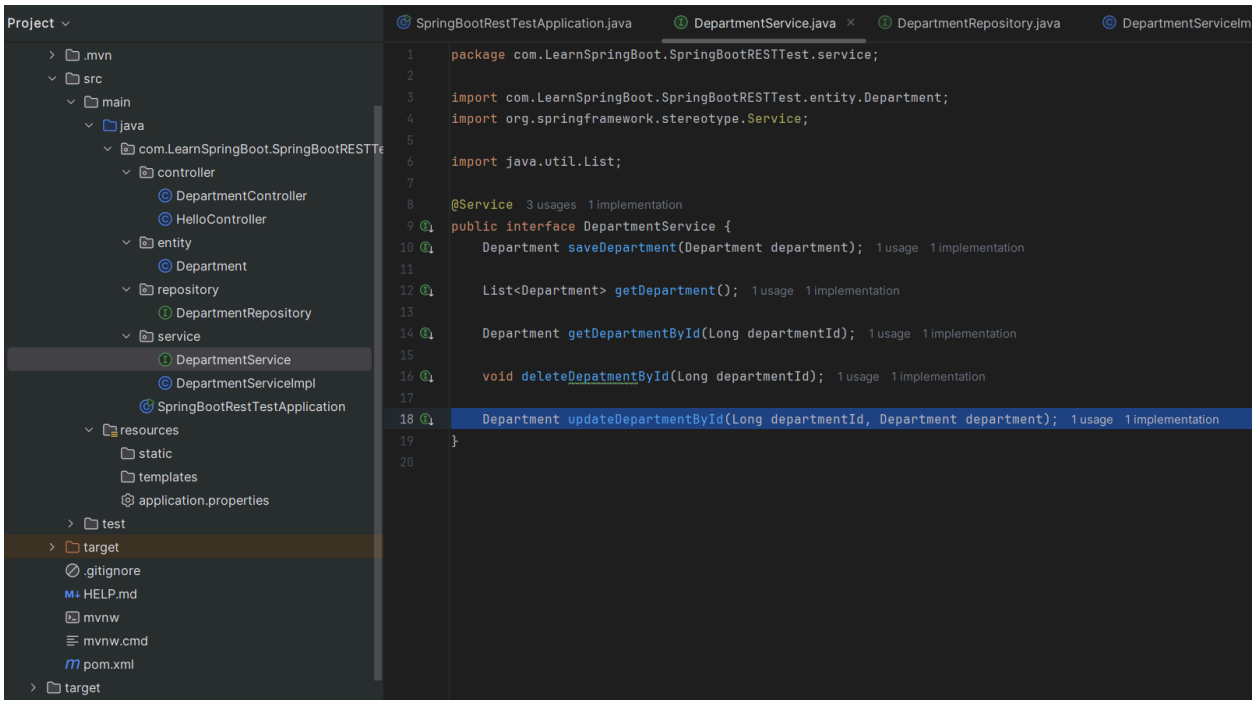


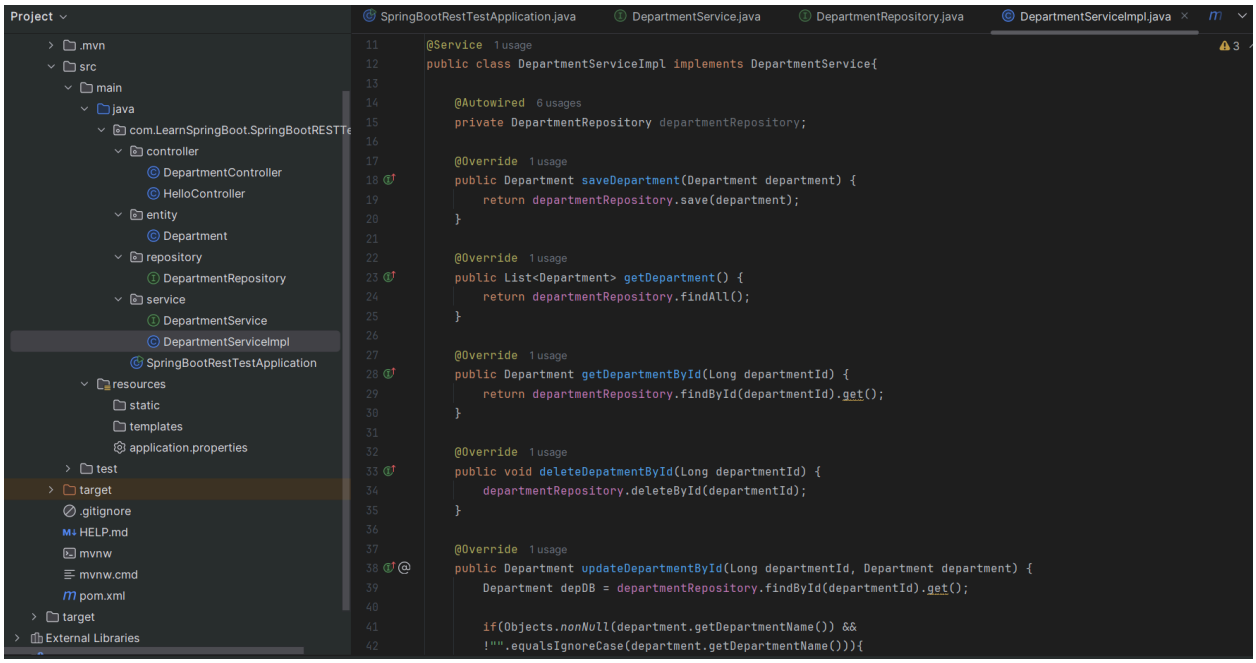
3 Controller Layer (Define End points and service layers function calls)



```java
    @Override  1 usage
    public Department updateDepartmentById(Long departmentId, Department department) {
        Department depDB = departmentRepository.findById(departmentId).get();

        if(Objects.nonNull(department.getDepartmentName()) &&
        !"".equalsIgnoreCase(department.getDepartmentName())){
            depDB.setDepartmentName(department.getDepartmentName());
        }

        if(Objects.nonNull(department.getDepartmentAddress()) &&
                !"".equalsIgnoreCase(department.getDepartmentAddress())){
            depDB.setDepartmentAddress(department.getDepartmentAddress());
        }

        if(Objects.nonNull(department.getDepartmentCode()) &&
                !"".equalsIgnoreCase(department.getDepartmentCode())){
            depDB.setDepartmentCode(department.getDepartmentCode());
        }
        return departmentRepository.save(depDB);
    }
}
```

4 Service Layer all the logic and implementation




Project
.mvn
src
main
java
com.LearnSpringBoot.SpringBootRESTTe
controller
DepartmentController
HelloController
entity
Department
repository
DepartmentRepository
service
DepartmentService
DepartmentServiceImpl
SpringBootRestTestApplication
resources
static
templates
application.properties
test
target
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
target
External Libraries
SpringBootRestTestApplication.java
DepartmentService.java
DepartmentRepository.java
DepartmentServiceImpl.java
@Service 1 usage
public class DepartmentServiceImpl implements DepartmentService{
@Autowired 6 usages
private DepartmentRepository departmentRepository;
@Override 1 usage
public Department saveDepartment(Department department) {
return departmentRepository.save(department);
}
@Override 1 usage
public List<Department> getDepartment() {
return departmentRepository.findAll();
}
@Override 1 usage
public Department getDepartmentById(Long departmentId) {
return departmentRepository.findById(departmentId).get();
}
@Override 1 usage
public void deleteDepatmentById(Long departmentId) {
departmentRepository.deleteById(departmentId);
}
@Override 1 usage
public Department updateDepartmentById(Long departmentId, Department department) {
Department depDB = departmentRepository.findById(departmentId).get();
if(Objects.nonNull(department.getDepartmentName()) &&
!"".equalsIgnoreCase(department.getDepartmentName())){