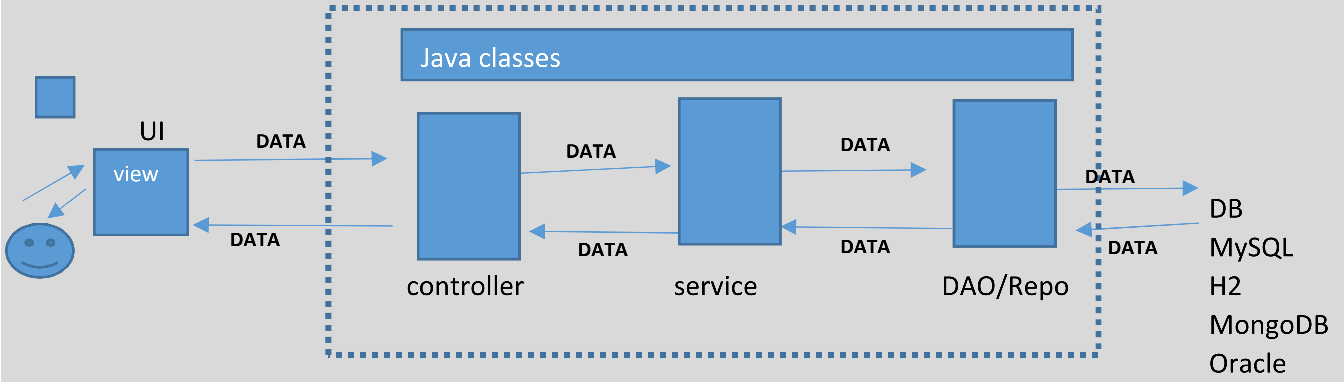
spring

spring webmvc

mvc

M	Model	Represents data which flows between components
V	View	UI which end user can view in browser
C	Controller	Who receives request from end user, gives response to end user



UI

view

DATA

Java classes

controller

service

DAO/Repo

DB

MySQL

H2

MongoDB

Oracle

Controller

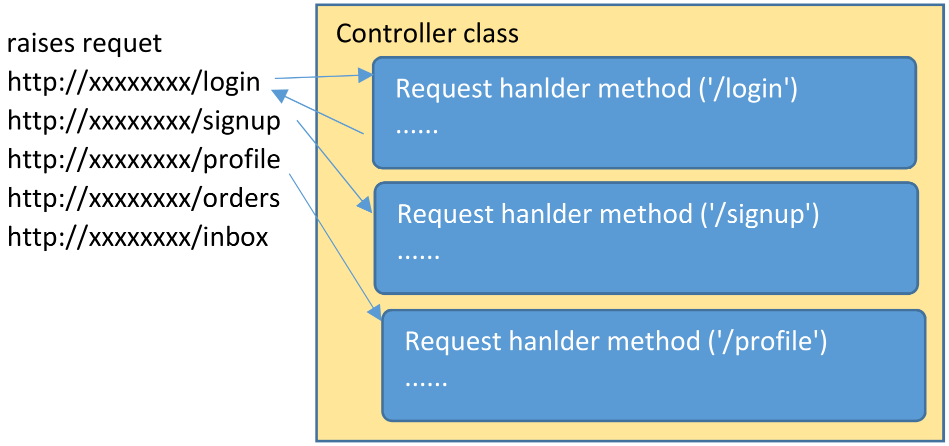
Java class

Holds many request handling methods

These methods are called as 'Request handler methods'

Request hanlder methods take request and give response

if end user raises requet



http://xxxxxxxx/login

http://xxxxxxxx/signup

http://xxxxxxxx/profile

http://xxxxxxxx/orders

http://xxxxxxxx/inbox

Controller class

Request hanlder method ('/login')
......

Request hanlder method ('/signup')
......

Request hanlder method ('/profile')
......

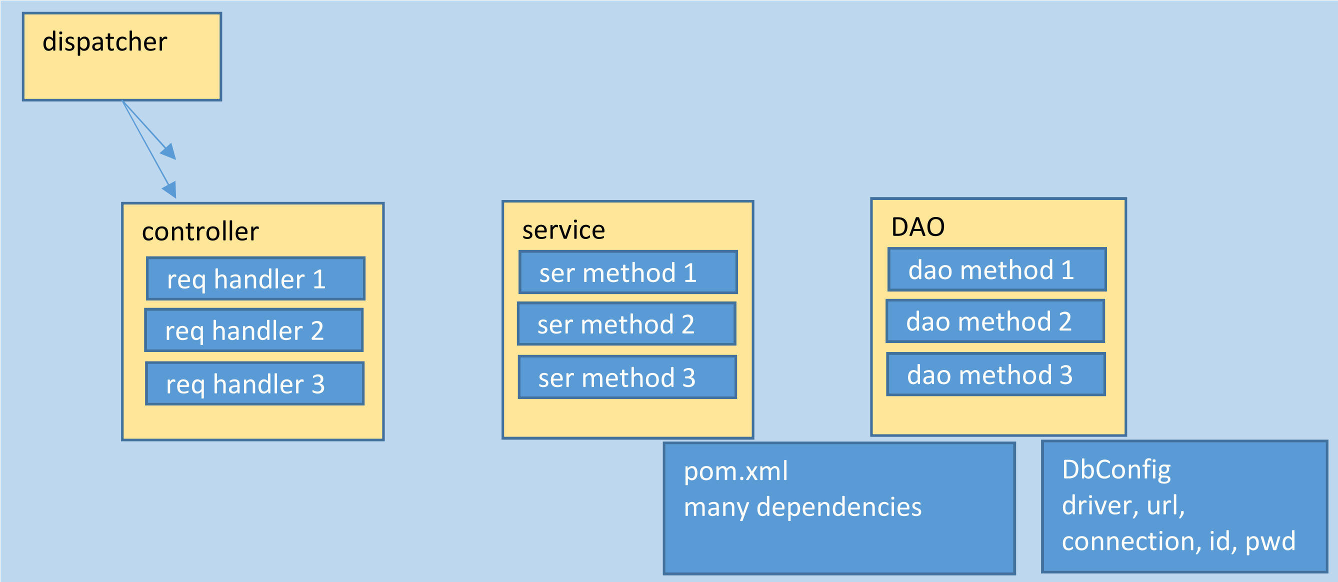
Service

Java class
Middle layer within application
Connects Controller layer to Data layer

DAO

Data Access Object
Java class
Can be called as Repository
Performs data operations on data source
Data source can be

- database
- file
- collection



dispatcher

controller
req handler 1
req handler 2
req handler 3

service
ser method 1
ser method 2
ser method 3

DAO
dao method 1
dao method 2
dao method 3

pom.xml
many dependencies

DbConfig
driver, url,
connection, id, pwd

Dependencies
spring web
spring webmvc
taglibs
jstls

Configurations
dispatcher
controler

Webserver
to be integrated
configured

in sprngmvc application development

Programmer needs to add many inidividual dependencies in pom.xml
Dispatcher needs to be configured manually
Webserver needs to be attached manully to spring application
All required beans need to be loaded to container manually
Database related configurations need to be loaded manually

springboot

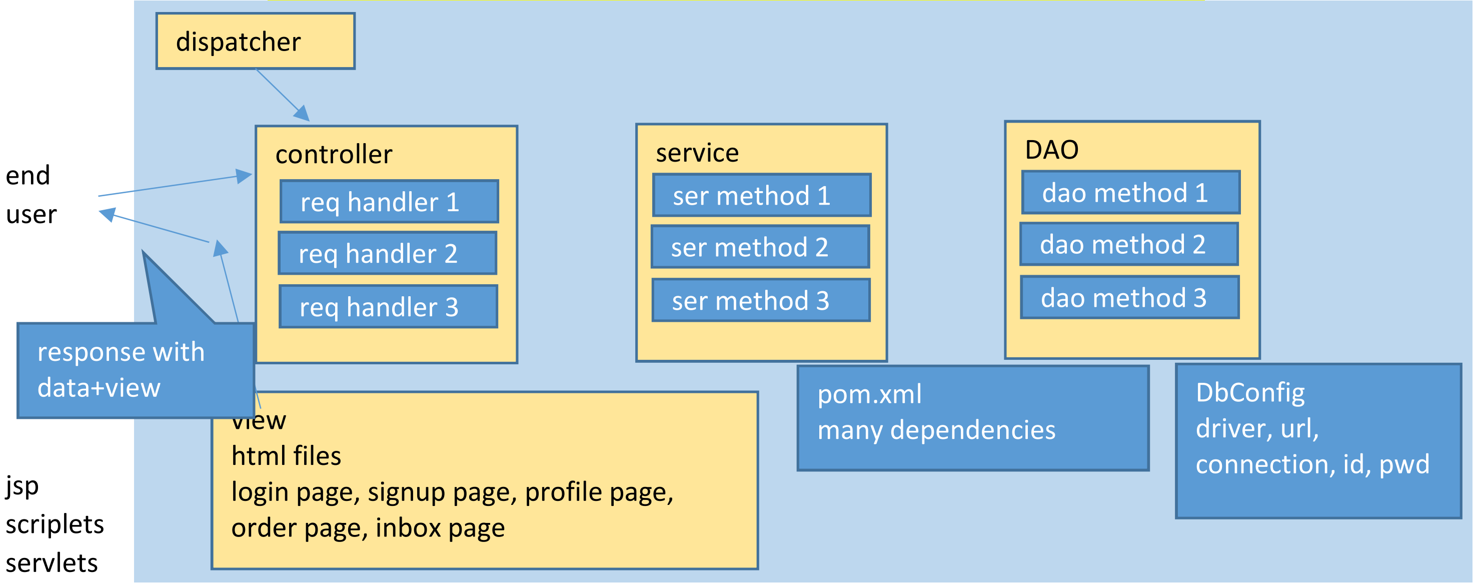
above all points can be automated

Web application types

1 Web application with UI (with view)

UI <-> Req Controller <-> Service <-> DAO <-> DB

UI	Req Controller	Service	DAO	DB
html	Java code	Java code	Java code	MySql
			to interact with DB	



end user

dispatcher

controller
req handler 1
req handler 2
req handler 3

service
ser method 1
ser method 2
ser method 3

DAO
dao method 1
dao method 2
dao method 3

response with data+view

view
html files
login page, signup page, profile page, order page, inbox page

pom.xml
many dependencies

DbConfig
driver, url, connection, id, pwd

jsp
scriplets
servlets
spring controller

views can go along with data

if end user send request for login
then controller picks loginpage.html and gives loginpage view as response to end user

http://xxxxxx/loginpage

in above type of application

response used to be view for every request made by enduser

Note: view is in the form of webpage

here response holds data+ view

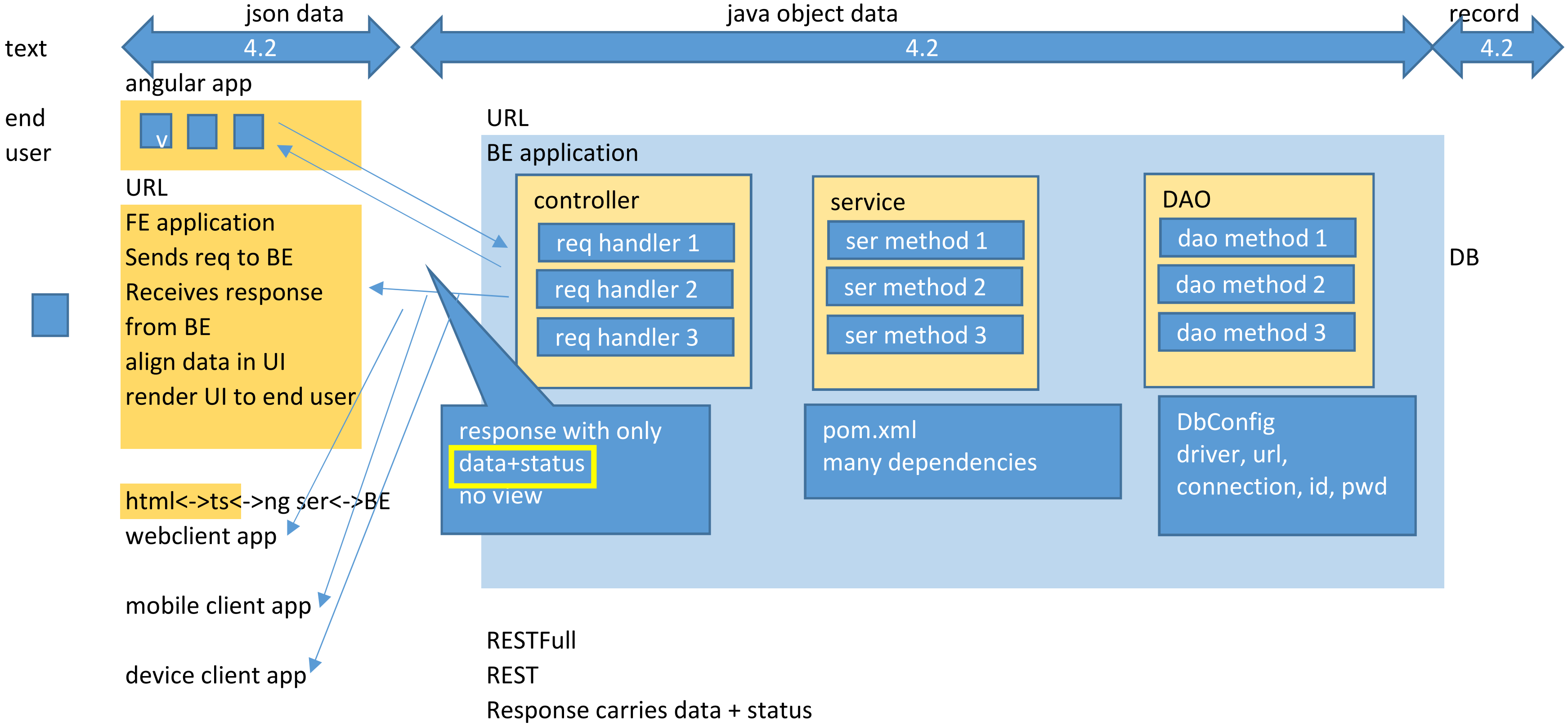
end user

Computer
browser

Android user
app

IOS user
app

type 2 REST application (no view)



text

end user

json data 4.2

java object data 4.2

record 4.2

angular app

v

URL

FE application
Sends req to BE
Receives response from BE
align data in UI
render UI to end user

html<->ts<->ng ser<->BE

webclient app

mobile client app

device client app

URL

BE application

controller
- req handler 1
- req handler 2
- req handler 3

service
- ser method 1
- ser method 2
- ser method 3

DAO
- dao method 1
- dao method 2
- dao method 3

DB

response with only
data+status
no view

pom.xml
many dependencies

DbConfig
driver, url,
connection, id, pwd

RESTFull
REST
Response carries data + status

Conclusion

Traditional spring application needs
- manual configuration for Dispatcher
- many dependencies to be managed in pom
- DB configuration are manual
- Beans to be loaded manually
- additional configuration required to work with tomcat

Springboot application

auto configurations
beans load automatic
no need of adding many dependencies in pom
embedded tomcat

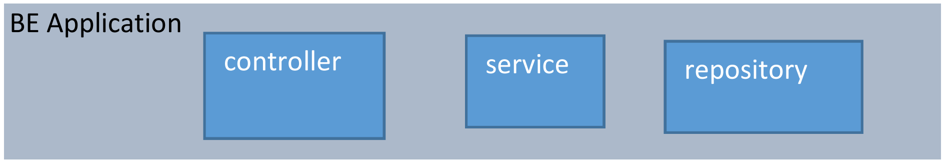
Traditional spring webmvc application

Gives response as View+Data (view and data are tightly coupled)

RESTfull api

Gives response as Data+Status code (view is ditached)
Note: View logic to be implemented in FE application

@Bean	Makes an object to load as bean into container
@Controller	Makes java class as spring controller, loads this class object as bean into container
@Service	Makes class as spring service, loads this class object as bean into container
@Repository	Makes class as DAO, loads this class object as bean into container
@Configuration	
@Component	
@Entity	
@Id	



BE Application

controller

service

repository

Demo 1

Simple springboot application (not a webapp)

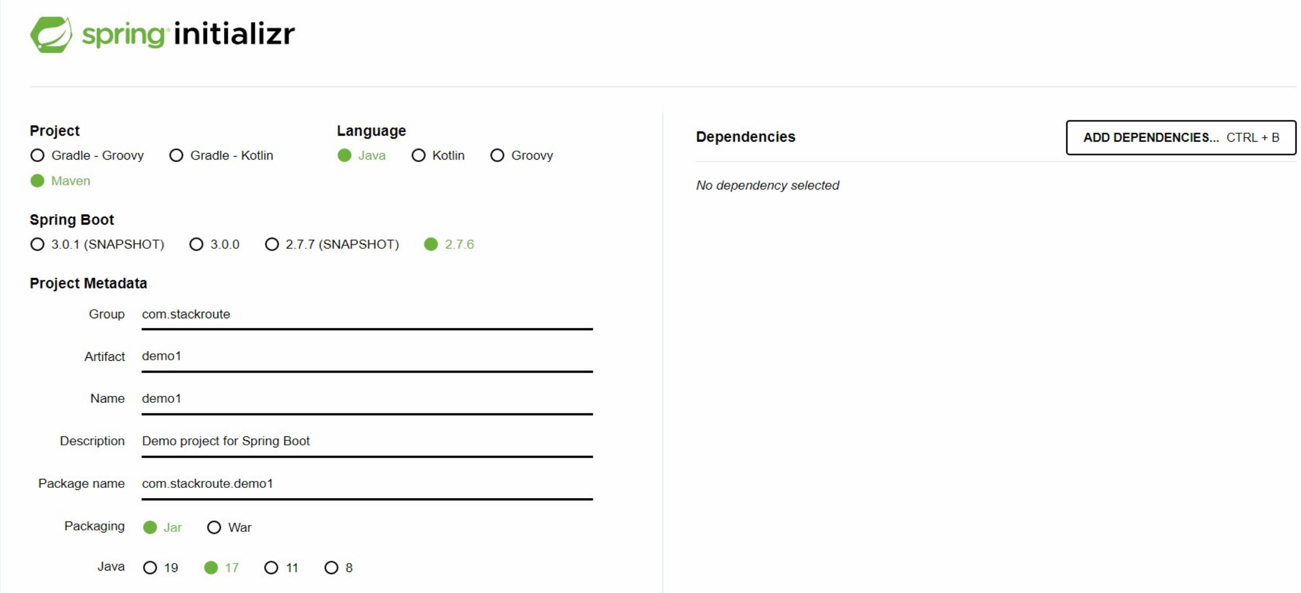
BE Application

main class

Steps to create simple springboot application

Step 1 Create new springboot application

https://start.spring.io/


spring initializr

Project
○ Gradle - Groovy ○ Gradle - Kotlin
● Maven

Language
● Java ○ Kotlin ○ Groovy

Spring Boot
○ 3.0.1 (SNAPSHOT) ○ 3.0.0 ○ 2.7.7 (SNAPSHOT) ● 2.7.6

Project Metadata

Group	com.stackroute
Artifact	demo1
Name	demo1
Description	Demo project for Spring Boot
Package name	com.stackroute.demo1
Packaging	● Jar ○ War
Java	○ 19 ● 17 ○ 11 ○ 8

Dependencies

No dependency selected

generate

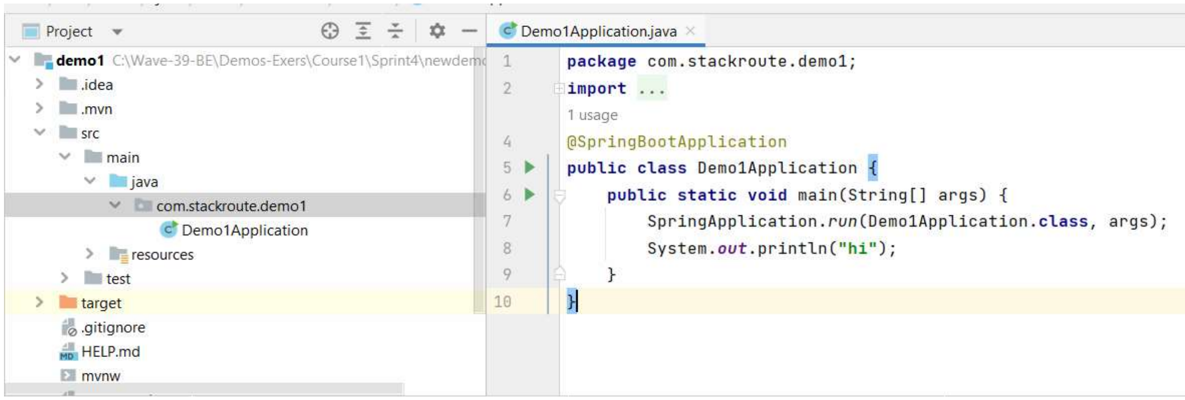
extract generated project

copy to workspace

open project in intellij

check jdk, maven properties

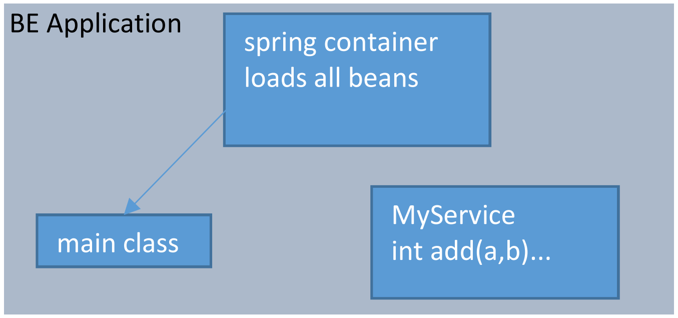
Step 2 run main()



```java
package com.stackroute.demo1;
import ...
1 usage
@SpringBootApplication
public class Demo1Application {
    public static void main(String[] args) {
        SpringApplication.run(Demo1Application.class, args);
        System.out.println("hi");
    }
}
```

Demo 2

Springboot application with service



controller
repo
conf
comp

```
main()
MyService m = new MyService();
sout(m.add(3,4));
```

```
@Service
class MyService  {
                public int add(int a, int b) { }
}
```

In springboot application
service class also gets loaded as bean into spring container automatically
So, in main() we can use 'myService' bean

containerobj.getBean("myService", MyService.class);

Step 1 Create new springboot application using spring initializer

download, extract, copy to workspace, open in intellij

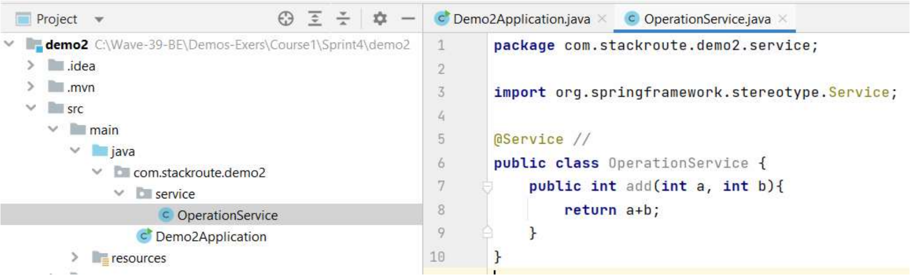
check settings if required

Step 2 check main() once by executing

make sure application running smooth

Step 3 Create one behaviour class with some methods

make this class as '@Service

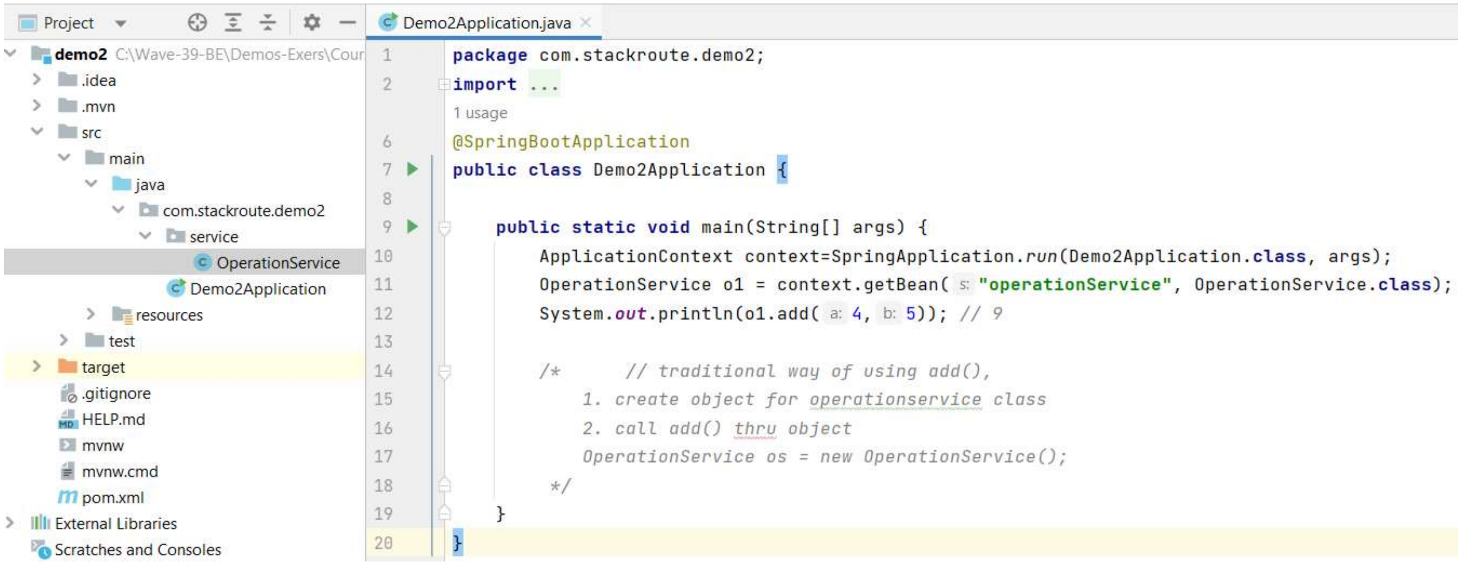


```java
package com.stackroute.demo2.service;

import org.springframework.stereotype.Service;

@Service //
public class OperationService {
    public int add(int a, int b){
        return a+b;
    }
}
```

Step 4 in main()

create ref of container

thru ref, get bean and use bean



```java
package com.stackroute.demo2;
import ...
1 usage
@SpringBootApplication
public class Demo2Application {

    public static void main(String[] args) {
        ApplicationContext context=SpringApplication.run(Demo2Application.class, args);
        OperationService o1 = context.getBean( s: "operationService", OperationService.class);
        System.out.println(o1.add( a: 4, b: 5)); // 9

        /*      // traditional way of using add(),
            1. create object for operationservice class
            2. call add() thru object
            OperationService os = new OperationService();
          */
    }
}
```