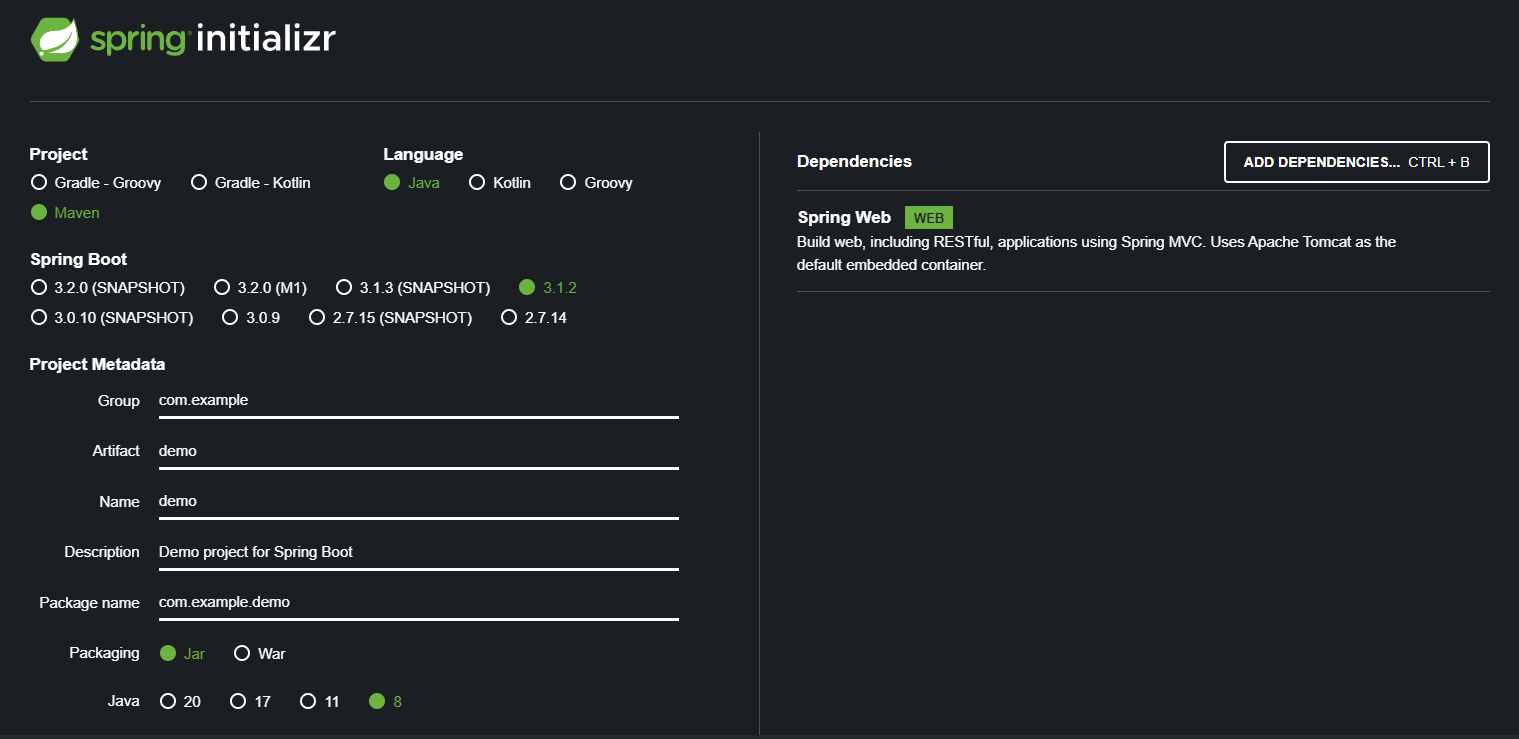
# Create Spring Boot Microservice With Spring Initializer

Web site: [**https://start.spring.io/**](https://start.spring.io/)

Development Steps



1. Group ID : that identifies the uniqueness of your project among all the projects. Ex: com.example.basic

2. Artifact : artifact is nothing but an application name. Ex: microservice-basic

3. Name: microservice-basic

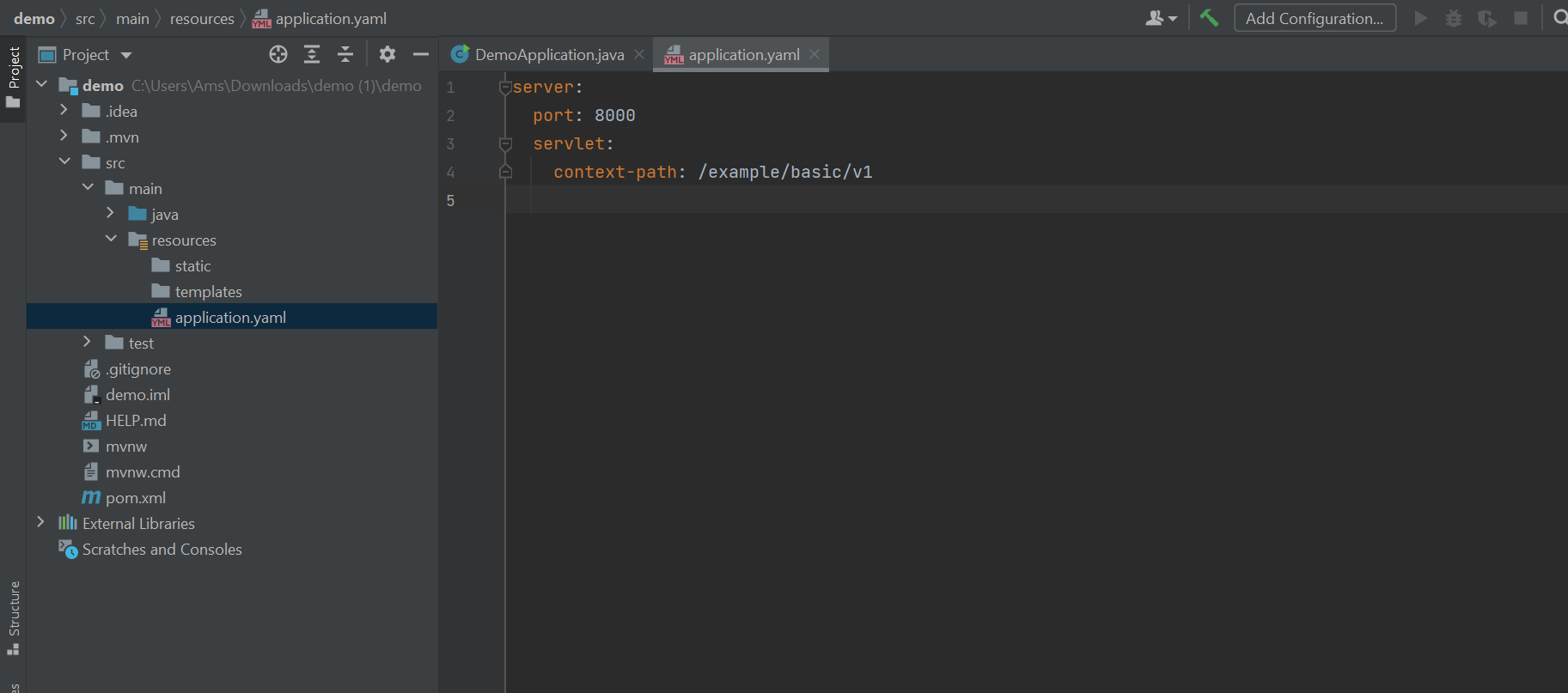
4. Description: Microservice basic example

5. Packaging: I choose a jar because we are creating a simple standalone spring project with the embedded tomcat server

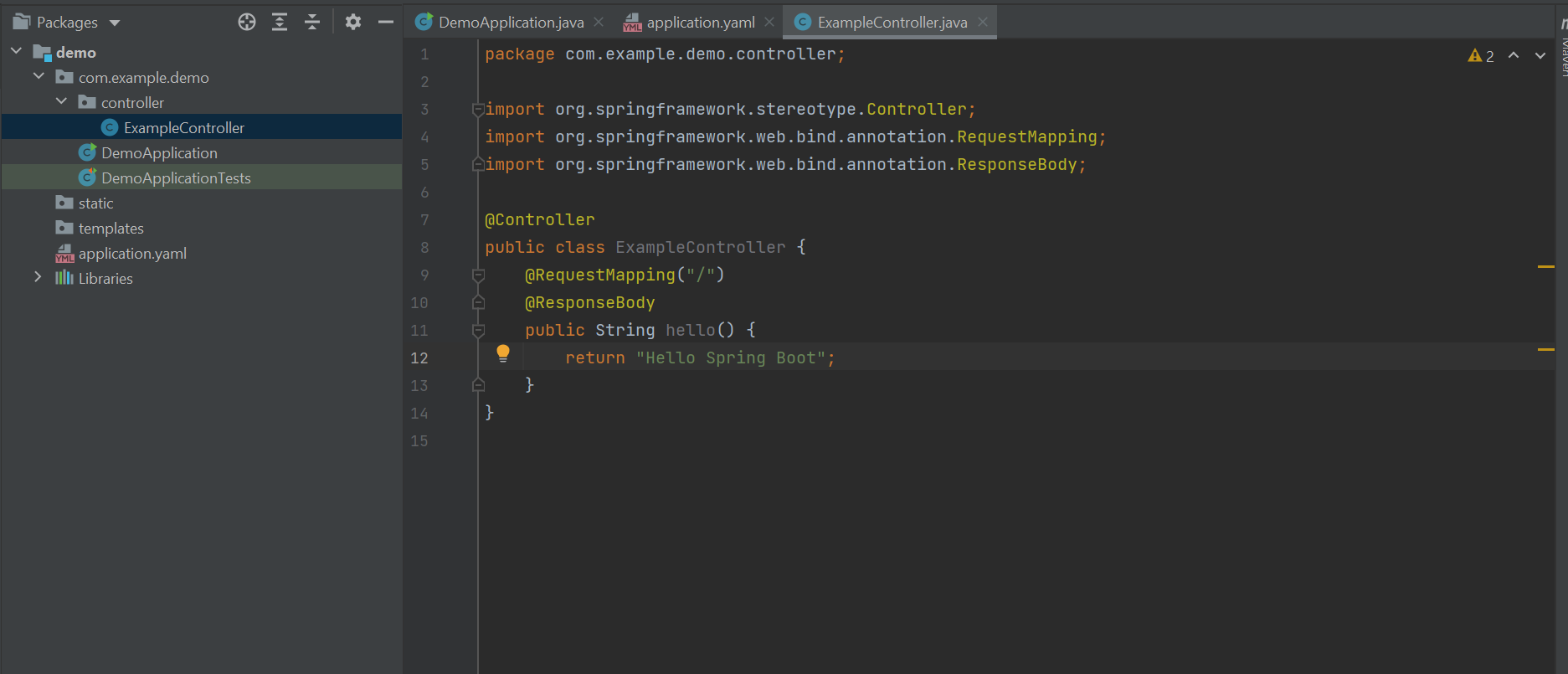
6. Java Version : 8 (you can choose any)

7. Dependencies: Spring Web, which is because we are creating a simple web application or REST API application

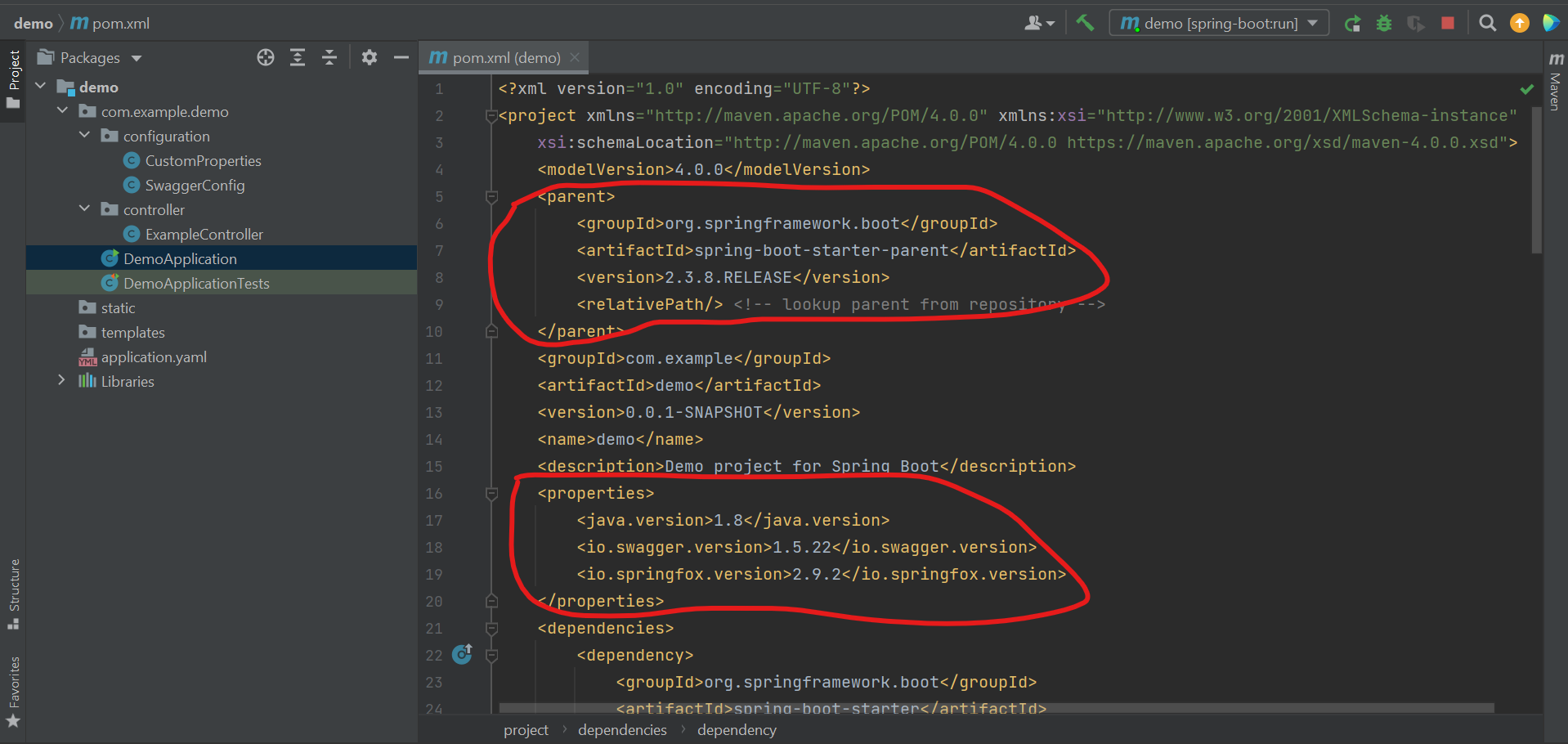
8. Change file name application.properties to application.yaml and add below properties



9. Create ExampleController as below

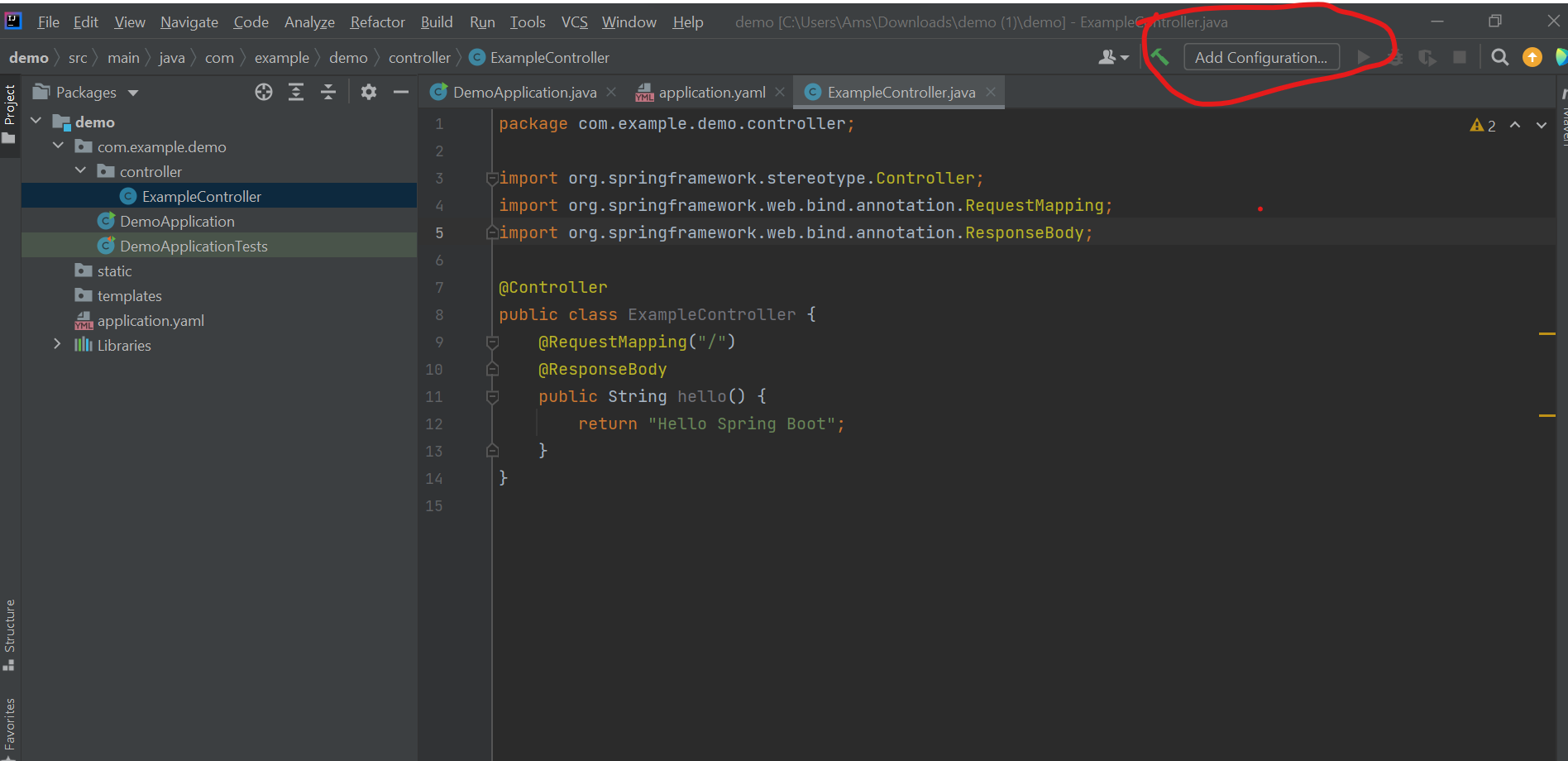


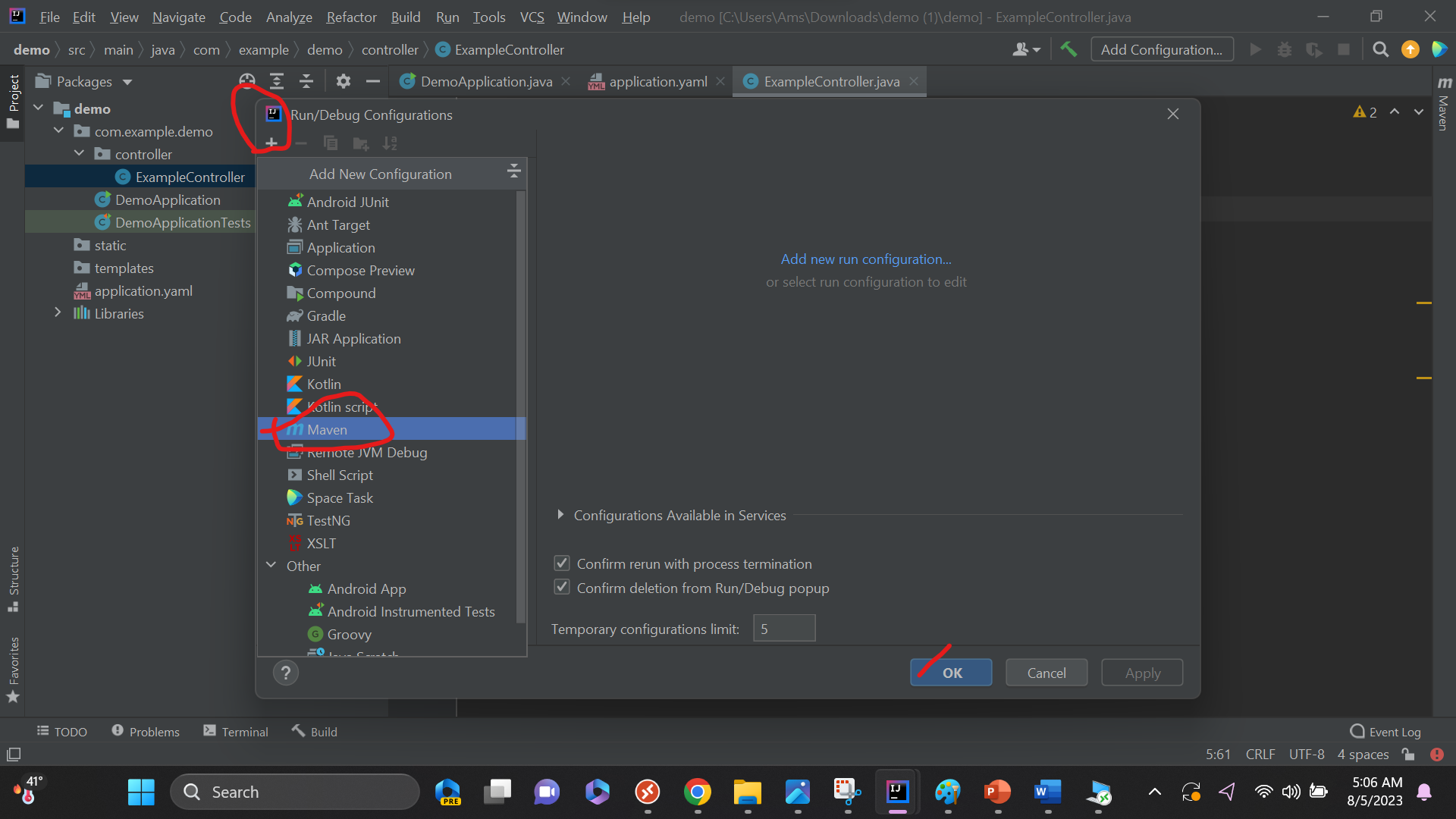
10. Change pom.xml file as below: spring-boot-starter-parent version to 2.3.8.RELEASE and java version



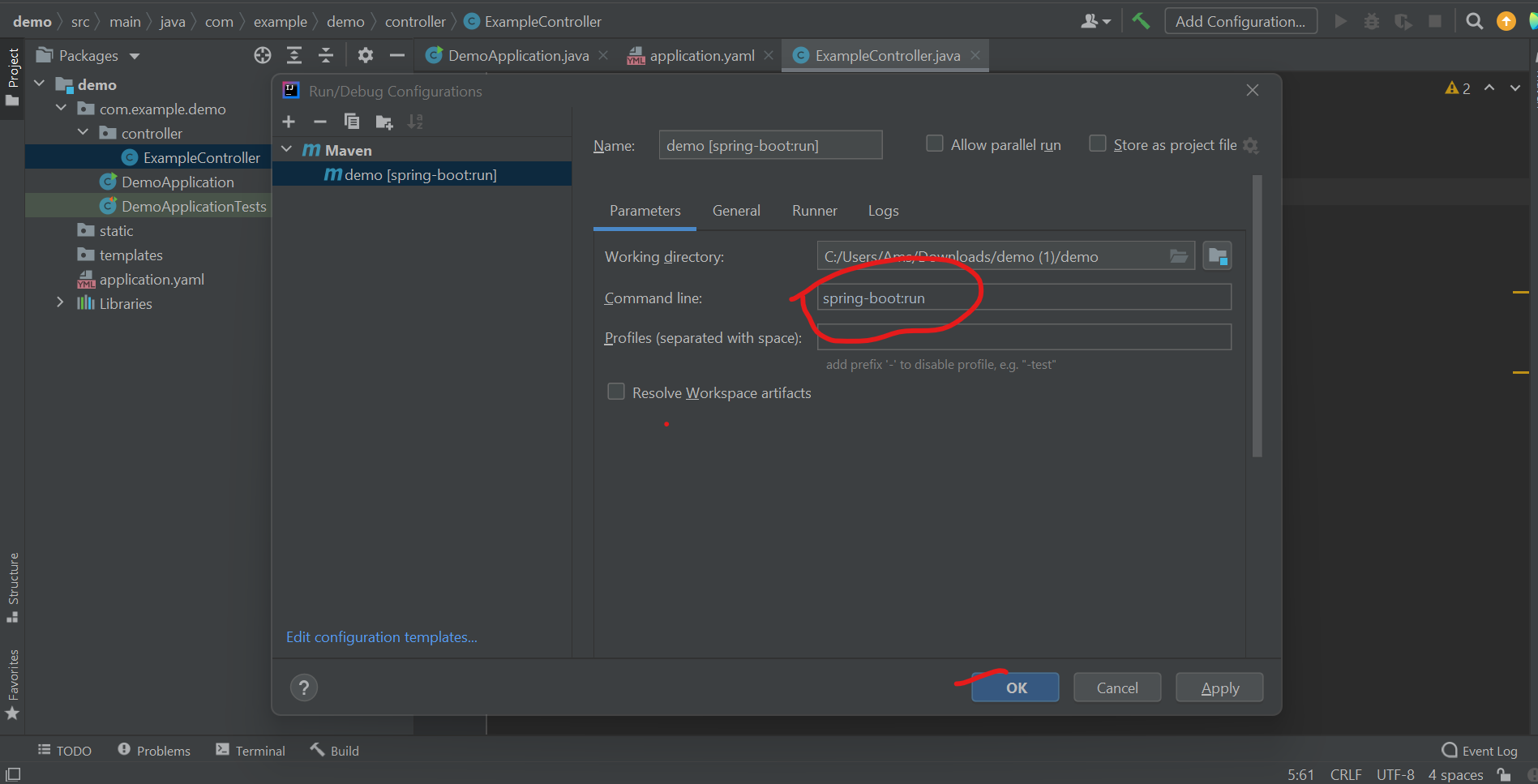
11. Create application run profile :

a. click on Add configuration

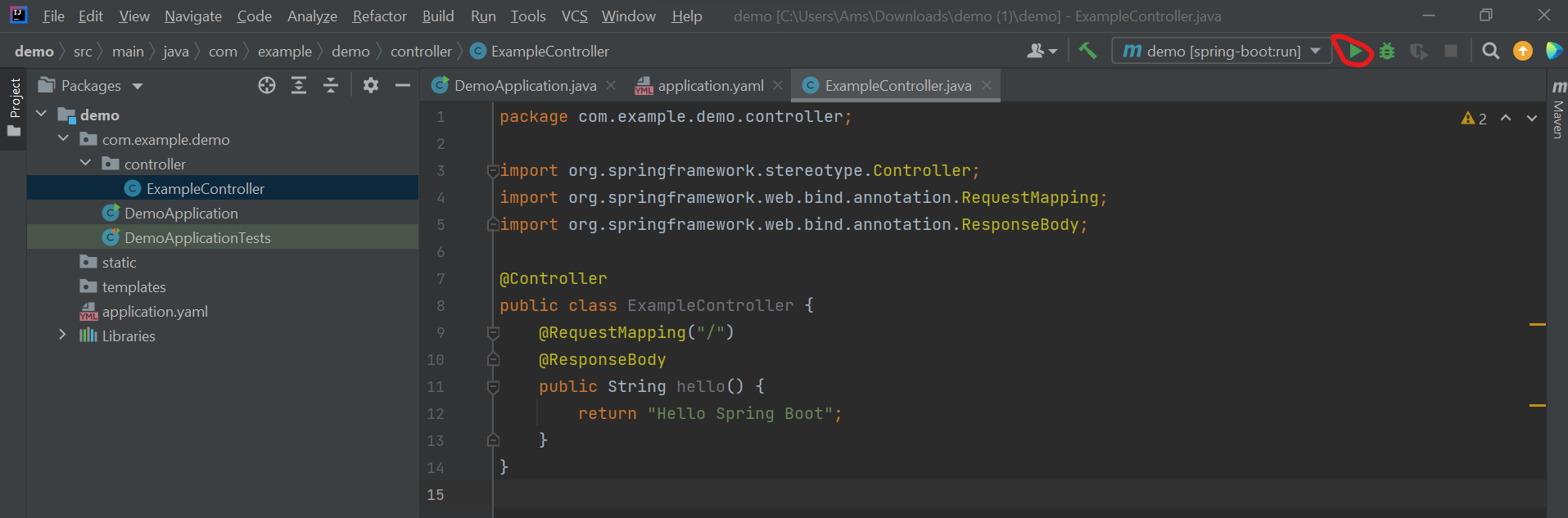
  
b. Select maven as below



c. Enter “spring-boot:run” to command line input, click OK



d. Run the application



Create Sample REST API

Follow the link : <https://spring.io/guides/gs/rest-service/>

## Core java design pattern

* **Singleton**

// Java program implementing Singleton class

// with using getInstance() method

// Class 1

// Helper class

class Singleton {

// Static variable reference of single\_instance

// of type Singleton

private static Singleton single\_instance = null;

// Declaring a variable of type String

public String s;

// Constructor

// Here we will be creating private constructor

// restricted to this class itself

private Singleton()

{

s = "Hello I am a string part of Singleton class";

}

// Static method

// Static method to create instance of Singleton class

public static synchronized Singleton getInstance()

{

if (single\_instance == null)

single\_instance = new Singleton();

return single\_instance;

}

}

// Class 2

// Main class

class SingletonMain {

// Main driver method

public static void main(String args[])

{

// Instantiating Singleton class with variable x

Singleton x = Singleton.getInstance();

// Instantiating Singleton class with variable y

Singleton y = Singleton.getInstance();

// Instantiating Singleton class with variable z

Singleton z = Singleton.getInstance();

// Printing the hash code for above variable as

// declared

System.out.println("Hashcode of x is "

+ x.hashCode());

System.out.println("Hashcode of y is "

+ y.hashCode());

System.out.println("Hashcode of z is "

+ z.hashCode());

// Condition check

if (x == y && y == z) {

// Print statement

System.out.println(

"Three objects point to the same memory location on the heap i.e, to the same object");

}

else {

// Print statement

System.out.println(

"Three objects DO NOT point to the same memory location on the heap");

}

}

}

* **Factory design pattern**

1. Interface

public interface Notification {

void notifyUser();

}

1. Implementation classes
   1. To send SMS

public class SMSNotification implements Notification {

@Override

public void notifyUser()

{

// TODO Auto-generated method stub

System.out.println("Sending an SMS notification");

}

}

* 1. To send email

public class EmailNotification implements Notification {

@Override

public void notifyUser()

{

// TODO Auto-generated method stub

System.out.println("Sending an e-mail notification");

}

}

* 1. Send push notification

public class PushNotification implements Notification {

@Override

public void notifyUser()

{

// TODO Auto-generated method stub

System.out.println("Sending a push notification");

}

}

1. Factory class

public class NotificationFactory {

public Notification createNotification(String channel)

{

if (channel == null || channel.isEmpty())

return null;

switch (channel) {

case "SMS":

return new SMSNotification();

case "EMAIL":

return new EmailNotification();

case "PUSH":

return new PushNotification();

default:

throw new IllegalArgumentException("Unknown channel "+channel);

}

}

}

1. Notification service class

public class NotificationService {

public static void main(String[] args)

{

NotificationFactory notificationFactory = new NotificationFactory();

Notification notification = notificationFactory.createNotification("SMS");

notification.notifyUser();

}

}

## Core java logical program

1. Find the neighbors.

Enter number to find its neighbors from a matrix of 3X3, 4X4, 5X5

Eg. 1 2 3

4 5 6

7 8 9

**If user enter : 5, expected outcome of neighbors should be**

🡪top : 2, bottom : 8, right : 6, left : 4

**If user enter : 2, expected outcome of neighbors should be**

🡪no top then last item of column: 8, bottom : 5, right : 3, left : 1

**If user enter : 8, expected outcome of neighbors should be**

🡪top: 5, no bottom then first item of column: 2, right : 9, left : 7

**If user enter : 6, expected outcome of neighbors should be**

🡪top: 3, bottom: 9, no right then first item of row : 4, left : 5

**If user enter : 4, expected outcome of neighbors should be**

🡪top: 1, bottom: 7, right: 5, no left then last item of row : 6, right : 5

## Core java logical program 2

Identity : Class structure, Data types, Prepare data of list

Problem : While input is userId of any 🡪 56124598, 56137598, 78126709, 78145709

We need to return all items which are GENERIC and enable true and userId specific one.

Remove duplicate id in case it consist.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | name | type | userId | enable |
| SN1001 | View Account Balance | DASHBOARD | GENERIC | True |
| SN1002 | Convert to BNPL | DASHBOARD | GENERIC | True |
| SN1003 | Pay Bills | DASHBOARD | GENERIC | False |
| SN1004 | View Profile | SIDE MENU | GENERIC | True |
| SN1005 | View Card Information | SIDE MENU | GENERIC | True |
| SN1005 | Help Center | SIDE MENU | GENERIC | False |
| SN1003 | Pay Bills | DASHBOARD | 56124598 | True |
| SN1003 | Pay Bills | DASHBOARD | 56137598 | True |
| SN1005 | Help Center | SIDE MENU | 78126709 | False |
| SN1005 | Help Center | SIDE MENU | 78145709 | False |

## Rest API program

1. Create another controller from demo app called “Accounts”
2. Create ArrayList of below POJO class

public class Account{

private Name holderName;

private String number; // eg. 1001342565

private BigDecimal balance;

private AccountType type;

}

public class Name{

private String firstName;

private String middleName;

private String lastName;

}

public enum AccountType{

CURRENT,

SAVING,

LOAN

}

1. Create GET /accounts endpoint: Get all accounts
2. Create POST /accounts endpoint: Create account but “number” field should be unique within array list
3. Create DELETE /accounts endpoint : Header parameter using number remove item from array list