jdk 11, open jdk

ide -- intellij community edition

mysql

h2

postman, SOAPUI, curl

1.Spring boot basics

2.Spring boot jdbc template

3.Spring boot with spring data jpa

4.Spring boot SOAP/Rest Application

--- Exception Handling

--- Spring Boot Security (jwt,oauth)

5.Spring Boot MicroService

6.Spring Boot Cloud(Security)

========================================================

build tool

maven -- pom.xml

gradle build file

spring boot application

-->cli tool

--> spring initliser -- browser,

having embedded tomcat server, optioanl jetty web server

embedded data base h2

ide

command mode(terminal)

Spring

@Configuration

@Bean

@Configuration

class Employee{

@Bean

public Employee getEmployee(){

}

}

Indicates a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning.

This is a convenience annotation that is equivalent to declaring

@SpringBootConfiguration, @EnableAutoConfiguration and @ComponentScan.

@SpringBootApplication= @SpringBootConfiguration+

@EnableAutoConfiguration+

@ComponentScan

@ComponentScan -- scan the component having annotation @Component

lombok tool

--- getter and setter

--- toString, default and overloaded constructor

@Getter

@Setter

@Data -- @Getter @Setter @RequiredArgsConstructor @ToString @EqualsAndHashCode.

from terminal

we can run in two ways

1st way - by creating jar file then run the jar file

mvn clean compile package

-- it will delete directory target then download required dependency

then compile all .java files then package command will package it in .jar file inside target folder

C:\Users\shankar\Desktop\springBootApp1>mvn clean compile package

running

C:\Users\shankar\Desktop\springBootApp1>java -jar target\springBootApp1-0.0.1-SNAPSHOT.jar

2nd ways

mvn spring-boot:run

-- it will do all steps and run the application without creating jar file

spring

spring boot

extended/advanced form of spring

--> auto-configuration

--> auto scan

--> easier

--> bootstrap tool -- spring boot initialiser(io.spring, plugins (intellij, eclipse, sts) )

--> embedded server(tomcat)

while run -- main method --whole application

build tool for spring boot project

a) maven

b) gradle

more than build tool it is project management tool

both tool gives you same project structure

src folder

two common folder

1. main --- java --- create java files

2. test --- test cases (junit, mockito)

maven -- pom.xml -- java build tool

gradle -- build.gradle

gradle build tool dsl(domain specific language )

--> using tasks it work

used for c,c++, python, java, android studio

maven and gradle can run application from terminal/command model also

maven in terminal mode

1st way

mvn spring-boot:run

2nd way

mvn clean compile package

--it will generate jar file inside target folder

-- run jar by java -jar app.jar

gradle in terminal

1st way

gradle bootSpring

2nd way

gradle build

it will give the jar file and then run the jar file as above

class having main method use annotation

@SpringBootApplication

-- its a spring boot application

-- its equivalent to three annotation

@ComponentScan

@SpringBootConfiguration

@EnableAutoConfiguration

SpringApplication.run(File.class, args);

model/bean file

simple java file (POJO)

POJO -- plain old java object

-> it have private attributes and

-> getter setter for each attribute

Spring Boot is manged by its Spring Container

-- life cycle

1. init

2. service-- @PostConstuct

3. destroy -- @PreDestroy

scope ---

bean is singleton in nature

singleton-- single static bean object is created and shared everywhwere

lombok plugin

-- using annotation it generates

getter, setter, toString, constructor

@Getter

@Setter

@toString

-- all above 3 are combined in @Data

@Autowire

--autmatically wire the given bean with the other bean

Employee Address

id

name

salary

@autowire

Address address

@autowire

Car car

Has a relationship

Employee has address and car

**annotations**

→ @Autowire

→ @component --- make class as the component, it makes instance of the given class

→ @repository --- it declares the class as dao/repository class, it makes instance of the given

class

→ @service -- it declares the class as service class. it makes instance of the given

class

@component, @repository, @service – all have similar work

DAO --

=> data access object

=> it makes a layer where will do all data/database works

**dao layer application**

model --> Customer.java

|

CustomerDao.java

|

Client

**dao,service layer application**

|

CustomerDao.java

| model --> Customer.java

CustomerSevice

| Exception

Client

**Service --**

→ Client will deal directly with service layer

→ Client should not deal directly with dao layer

→ service layer will connect with dao layer

→ service layer may deal with exception

→ all values client

→ take values from .properties file

→ create .properties file in resources folder

@Value – it will assign default value to the field

**@PropertySource**

-- it us used to read the properties file

**-- after reading render the property using @Value**

@PropertySource("bus.properties")public class Bus {  
  
 [//@Value](mailto://@Value)("67676776")// default value  
 @Value("${bus.id}") // it will read bus.id property from .properties file and set the value to the given field

private int busId;  
 @Value("${bus.name}")  
 private String busName;  
 @Value("${bus.fare}")  
 private float busfare;  
}

**jdbc --**

**JdbcTemplate**

* using JdbcTemplate connect to database
* better than simple jdbc
* datasource – instead of creating connection database makes the pool of connection  
   on requirement it gets the connection from the pool

SpringBoot

jdbctemplate can be used in 2 ways

**1.**using @Bean get connection

**2.**add all database configuration values to the application.properties

using application.properties file it makes object of jdbctemplate, datasource