Table of Contents

Step 1: Spring Boot Project creation steps

1. BitBucket url for spring boot : https://bitbucket.org/shalini-ws/springbootdemorepo/src/main/

2. Open below url on the browser:
   https://start.spring.io/
3. Enter details as shown below:
   There is a mp4 video as well for the steps as well.
   Click on generate and the zip will be downloaded. Extract and open the project on intellij.

4. Go to Intellij ->

Step 2: Understand Spring Boot as opinionated

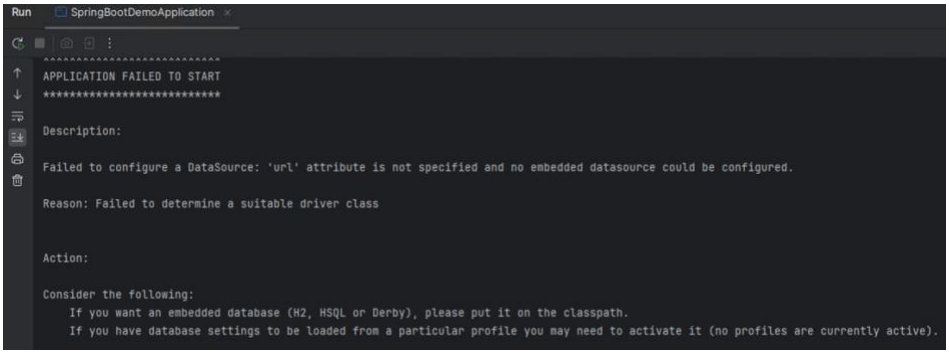
1. View the pom.xml file that has spring boot starter parent.

   It is a special starter project that provides default configurations for our application and a complete dependency tree to quickly build our Spring Boot project.
   It also provides default configurations for Maven plugins, such as maven-failsafe-plugin, maven-jar-plugin, maven-surefire-plugin, and maven-war-plugin.
   Beyond that, it also inherits dependency management from spring-boot-dependencies, which is the parent to the spring-boot-starter-parent.

2. @SpringBootApplication on the class with the main method:

   This annotation is used to enable three features, that is:
   a. @EnableAutoConfiguration: enable Spring Boot's auto-configuration mechanism
   b. @ComponentScan: enable @Component scan on the package where the application is located.
   c. @Configuration: allow to register extra beans in the context or import additional configuration classes

3. Run the application, you should see the error with respect to database configuration.
   This is the default behaviour of spring boot. It sees mysql dependency but did not find database connection parameter information.



4. Open database and create a database/schema named “neueda”

5. Open application.properties file within the resources folder and add the database related connection parameters:

   spring.datasource.url=jdbc:mysql://localhost:3306/neueda
   spring.datasource.username=root
   spring.datasource.password=c0nygre
   spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver

```
# below property if used will perform DDL operations to create table
# create will always drop and recreate
# update will create only if none exists
spring.jpa.hibernate.ddl-auto=create

# below property if used will show the sql queries generated by hibernate
spring.jpa.show-sql=true
```

6. Run the main method again and you should see the error is GONE!!!

7. Following the package structure is very important with spring boot for it to follow the default configurations.

Step 3: Understand Object creation in spring boot

1. Create class as follows:

```
package com.training.SpringBootDemo.component;

import java.util.Random;

public class TokenGenerator{

    public TokenGenerator (){
        System.out.println("Token default constructor");
    }
    public int generateToken(){
        return new Random().nextInt(1,10);
    }
}
```

2. Normally we create object and then call the respective method of the class.
`TokenGenerator` ob = new `TokenGenerator` ();
ob.generateToken();

3. Object creation part is taken care by spring. Just add the annotation @Component on the class Token as follows:
The classes whose objects are created by spring are called as **" SPRING MANAGED BEAN "**

```
import org.springframework.stereotype.Component;
@Component
public class TokenGenerator {
  // other methods
}
```

4. Just run the main method and you should see the output from the constructor.

```
2024-07-29T16:25:32.870+05:30  INFO 9678 --- [SpringBootDemo] [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start compl
2024-07-29T16:25:32.911+05:30  WARN 9678 --- [SpringBootDemo] [           main] org.hibernate.dialect.Dialect            : HHH000511: The 5.7.34 vers
2024-07-29T16:25:33.096+05:30  INFO 9678 --- [SpringBootDemo] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform
2024-07-29T16:25:33.098+05:30  INFO 9678 --- [SpringBootDemo] [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityMana
Token default constructor
2024-07-29T16:25:33.193+05:30  INFO 9678 --- [SpringBootDemo] [           main] c.t.S.SpringBootDemoApplication          : Started SpringBootDemoAppl
2024-07-29T16:25:33.196+05:30  INFO 9678 --- [SpringBootDemo] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerF
2024-07-29T16:25:33.198+05:30  INFO 9678 --- [SpringBootDemo] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown in
2024-07-29T16:25:33.223+05:30  INFO 9678 --- [SpringBootDemo] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown co
```

Step 4: Get bean context

1. To get access to the bean [class] created by spring, get it from spring context. Update main method as follows:

```
public static void main(String[] args) {

    ApplicationContext context =
    SpringApplication.run(SpringBootDemoApplication.class, args);

    TokenGenerator token = context.getBean(TokenGenerator.class);
    System.out.println(token.generateToken());

}
```

Step 5: @Value Annotation

1. Instead of manually setting the bounds for generatetoken() function, add origin and bound as instance variables.
2. Add parameterized constructor, getters and setters for the same as follows:
3. DO UPDATE THE generateToken() for the parameters.

```
@Component
public class TokenGenerator {

    private int origin;
    private int bound;
    public TokenGenerator (){
        System.out.println("Token default constructor");
    }

    public int getOrigin() {
        return origin;
    }

    public void setOrigin(int origin) {
        this.origin = origin;
    }

    public int getBound() {
        return bound;
    }

    public void setBound(int bound) {
        this.bound = bound;
    }

    public int generateToken(){
        return new Random().nextInt(origin,bound);
    }
}
```

4. To set values for origin and bound, spring provides 3 ways to inject values
    a. Field
    b. Constructor
    c. Setter

5. To do field injection update Token class as follows :

```
@Component
public class TokenGenerator {

    @Value("10")
    int origin;
    @Value("20")
    int bound;
     // other methods
}
```

6. Now run the main method and you should get the range of values between 10 and 20.
7. For setter injection, update Token class as follows **[CAN CHOOSE TO COMMENT FIELD INJECTION]**

```
@Value("20")
public void setOrigin(int origin) {
    this.origin = origin;
}

@Value("30")
public void setBound(int bound) {
    this.bound = bound;
}
```

Lab 1:

1. Update the EmailNotification to be a spring managed bean.
   [HINT: Add @Component on the class]
2. Run the main method and you should get the default constructor output for this service.
3. Get access to EmailNotification class from spring context and call the sendMessage() to see the output.
4. Use @Value annotation to inject value for message property and then uncomment the print method in sendMessage() method to display the value of message.

Step 6: @Autowired and @Service Annotation

1. Consider below class BankService that has a dependency on TokenGenerator class.

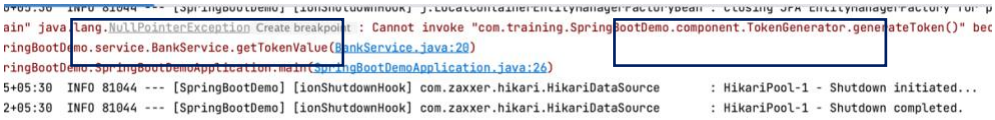
```
public class BankService {

    private TokenGenerator tokenGenerator;

    public BankService() {
            System.out.println("Bank Service default constructor");
```

```
    }

    public BankService(TokenGenerator tokenGenerator) {
        this.tokenGenerator = tokenGenerator;
    }

    public TokenGenerator getTokenGenerator() {
            return tokenGenerator;
    }

    public void setTokenGenerator(TokenGenerator tokenGenerator) {
            this.tokenGenerator = tokenGenerator;
    }
    public void getTokenValue(){
        System.out.println(tokenGenerator.generateToken());
    }
}
```

2. Can use @Component or @Service annotation on this class for spring to create object of BankService class.
   Update class as follows:
```
@Service
public class BankService {
}
```
3. Run the main method and will see the output from default constructor of BankService class.
4. Now get access of BankService class in the main method as follows and call the getTokenValue() method:

```
BankService bankService = context.getBean(BankService.class);
bankService.getTokenValue();
```

5. It will throw NullPointerException for TokenGenerator class.



6. Even though TokenGenerator class object was created, spring needs to know it is required by BankService class. Normally we would do as follows:
```
TokenGenerator tokenGenerator = new TokenGenerator();
BankService bankService = new BankService(tokenGenerator);
//OR
bankService.setTokenGenerator(tokenGenerator);
```

7. Hence use @Autowired annotation to tell spring to inject the TokenGenerator object created by spring.
   It can be done **ANY** of the following 3 ways.
   Use anyone, rerun the program and it should work now.

   a. Field injection: Add @Autowired on the filed as follows:
```
@Service
public class BankService {
    @Autowired
    private TokenGenerator tokenGenerator;
    // …
}
```
   b. Constructor injection:
```
@Autowired
public BankService(TokenGenerator tokenGenerator) {
    System.out.println("Bank Service parameterized consturctor");
    this.tokenGenerator = tokenGenerator;
}
```
   c. Setter injection:
```
@Autowired
public void setTokenGenerator(TokenGenerator tokenGenerator) {
    System.out.println("Set token generator");
    this.tokenGenerator = tokenGenerator;
}
```

Lab 2:

1. Update the BillingService to be a spring managed bean.
   [HINT: Add @Service on the class]
2. Add @Autowired to inject EmailNotification object as a dependency. Try all the 3 ways of dependency injection.
3. Get access to BillingService class from spring context and call the sendMessage() to see the output.

IF TIME PERMITS:

Step 7: @Bean Annotation

1. CREATE a simple maven java project "PaymentProject" with quickstart artifact and create a class as follows:

Below code is within PaymentProject which is a Java Maven Project.
[PLEASE NOTE : IT IS NOT SPRING PROJECT]

It has only one class as follows

```
package com.payment;

public class PaymentService {

   public double makePayment(double amount, double discount){
      amount = amount - amount * discount/100;
      double total = amount + 0.18;
      return  total;
   }
}
```

VVIMP:
Run maven install to create a jar file of PaymentProject and install within local repository.

2. Add PaymentProject as maven dependency in pom.xml of **SpringBootDemo project**

```
<dependency>
        <groupId>com.payment</groupId>
        <artifactId>PaymentProject</artifactId>
        <version>1.0-SNAPSHOT</version>
</dependency>
```

3. Add PaymentService as a dependency in BillingService of **SpringBootDemo** project as follows

```
@Service
public class BillingService {

   /**
    * payment service is not a part of this project and has been added as a dependency.
    * IN this case we cannot add @Component on the PaymentService class.
    */

   @Autowired
   private PaymentService paymentService;


   public void calculateCustomerPayment ()
   {
      System.out.println(paymentService.makePayment(12000,10));
   }
}
```

4. To inject PaymentService as a dependency, use @Bean annotation as follows within the AppConfig class :

```
package com.training.SpringBootDemo.config;

@Configuration
public class AppConfig
{
    @Bean
    public PaymentService service(){
        return  new PaymentService();
    }
}
```

5. Update the App class main method as follows:

```
BillingService bservice = context.getBean(BillingService.class);
bservice. calculateCustomerPayment ();
```

Advance Lab:

Exercise 1 :

Create a quiz application for a batch of learners. Using Spring boot as the technology, annotate classes with respective annotations [Use @Component and @Service] and develop application as follows:

1. IQuizMaster – Interface
   public String popQuestion();

2. SpringQuizMaster – Class that implements IQuizMaster interface
   and returns a question related to spring

3. JavaQuizMaster – Class that implements IQuizMaster interface
   and returns a question **that it reads from properties file**

4. QuizMasterService – Class that has IQuizMaster as a dependency with following methods
   a. public void setQuizMaster(IQuizMaster quizMaster) : sets the reference for quizmaster
   b. public void askQuestion() : invokes popQuestion of respective implementation of IQuizMaster
   NOTE : Explore @Qualifier.

5. main method: Get the QuizMasterService bean and call the askQuestion() method

Exercise 2:

Create an application to store the transactions made by customers. Application should display the list of all transactions made by a particular customer. Using Spring boot as the technology, annotate classes with respective annotations [Use respective annotations] and develop application as follows:

1. entity.Transaction class :
   private int txid;
   private String customerid;
   private String type; [It can be withdraw / deposit]
   [If possible use type as enum]
   private LocalDate date;

2. service.TransactionService : this class stores List of transactions and has following methods:
   class TransactionService{
   List<Transaction> tx ;
   }
   a. getTransactions(String customer id) that return list of all the transactions that match the customerid
   b. getTransactionsByType(String customerid, String type) that return list of all the transactions that match the customerid and the type
   c. Use @Bean annotated method to populate list of transaction with few dummy transaction objects.

3. main method: Get the TransactionService bean and test respective methods.