

## Table of Contents

|  |          |
|--|----------|
| <b>Step 1: Spring Boot Project creation steps.....</b>         | <b>1</b> |
| <b>Step 2: Understand Spring Boot as opinionated.....</b>      | <b>2</b> |
| <b>Step 3: Understand Object creation in spring boot .....</b> | <b>3</b> |
| <b>Step 4: Get bean context .....</b>                          | <b>3</b> |
| <b>Step 5: @Value Annotation.....</b>                          | <b>3</b> |
| <b>Step 6: @Autowired and @Service Annotation .....</b>        | <b>4</b> |
| <b>Step 7: @Bean Annotation.....</b>                           | <b>5</b> |

## 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/>

- 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.

The screenshot shows the Spring Initializr web form. The following fields are highlighted with red boxes:

- Project:** Maven
- Language:** Java
- Spring Boot:** 3.3.2
- Project Metadata:**
  - Group: com.training
  - Artifact: SpringBootDemo
  - Name: SpringBootDemo
  - Description: Demo project for Spring Boot
  - Package name: com.training.SpringBootDemo
  - Packaging: Jar
  - Java: 17
- Dependencies:**
  - Spring Data JPA
  - MySQL Driver

4.

## Step 2: Understand Spring Boot as opinionated

- 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.

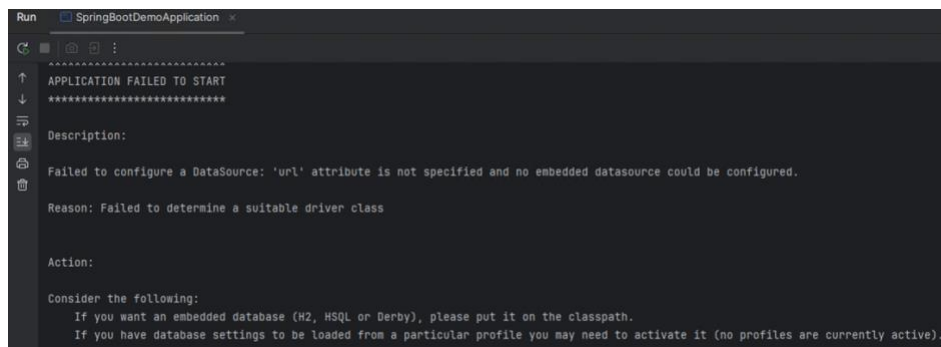
- @SpringBootApplication on the class with the main method:

This annotation is used to enable three features, that is:

- @EnableAutoConfiguration: enable Spring Boot's auto-configuration mechanism
- @ComponentScan: enable @Component scan on the package where the application is located.
- @Configuration: allow to register extra beans in the context or import additional configuration classes

- 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.



- Open database and create a database/schema named "neueda"
- Open application.properties file within the resources folder and add the database related connection parameters:

```

spring.datasource.url=jdbc:mysql://localhost:3306/java
spring.datasource.username=root
spring.datasource.password=root1234
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 7070 --- [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 : HHN000511: 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 : HHN000489: 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;
}

```

#### 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.

```
07:05:30 INFO 81044 --- [SpringBootDemo] [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory and p
ain" java.lang.NullPointerException: Cannot invoke "com.training.SpringBootDemo.component.TokenGenerator.generateToken()" bec
ringBootDemo.service.BankService.getTokenValue(BankService.java:20)
ringBootDemo.SpringBootDemoApplication.main(SpringBootDemoApplication.java:26)
5+05:30 INFO 81044 --- [SpringBootDemo] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2+05:30 INFO 81044 --- [SpringBootDemo] [ionShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

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 field as follows:

```
@Service
public class BankService {
    @Autowired
    private TokenGenerator tokenGenerator;
    // ...
}
```

- b. Constructor injection:

```
@Autowired
public BankService(TokenGenerator tokenGenerator) {
    System.out.println("Bank Service parameterized constructor");
    this.tokenGenerator = tokenGenerator;
}
```

- c. Setter injection:

```
@Autowired
public void setTokenGenerator(TokenGenerator tokenGenerator) {
    System.out.println("Set token generator");
    this.tokenGenerator = tokenGenerator;
}
```

## 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();
```