**Spring Framework**

1. Framework is predefined functionalities/code provided by any third party which can be use to reduce development efforts, time and minimize the errors.
2. Spring Framework is also known as framework of framework.
3. Spring distributed into a multiple module which can be use either separately or can be use in the combination of multiple modules.
4. Some of the important modules are listed below

**Spring IOC/Core**

Spring MVC

**Spring JDBC**

**Spring ORM (JPA/Hibernate)**

**Spring REST**

Spring Security

Spring JMS

Spring Batch

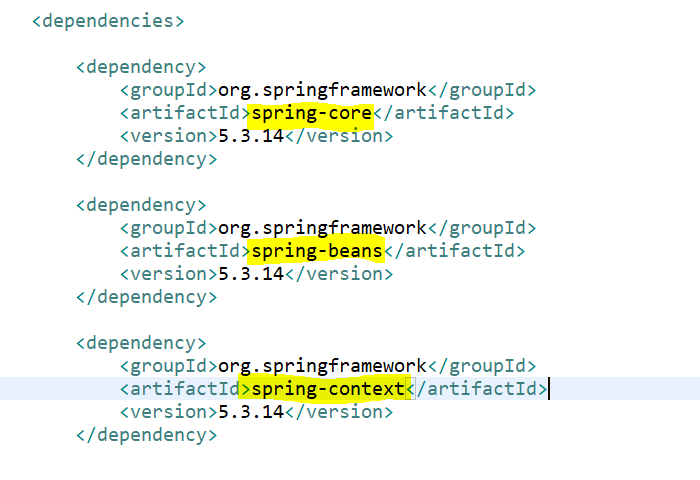
Spring Cloud etc…

1. To Implement any of the spring module you have to follow 3 steps at higher level.
   1. Add a jar files (Dependencies)
   2. Add Configuration of the module (by Spring suggested way)
   3. Customize the spring functionalities (Optional)
   4. Use/Implement the functionalities provided by Spring

Spring Docs : <https://docs.spring.io/spring-framework/docs/current/reference/html/>

**Spring IOC (Inverse Of Control)**

1. Spring IOC is also called as spring core.
2. This is the base module of the spring.
3. It can be implemented for core java application, Web App, Restful API Application etc…
4. In this module you will majorly implement following functionalities
   1. Spring Configuration.
   2. Spring Bean Class.
   3. Spring Container.
   4. Create/Get the spring bean class object using Spring Container.
   5. Spring Bean scope
   6. \*\*\*Spring CI(Constructor Injection), SI(Setter Injection), DI(Dependency Injection)
   7. \*\*\*Spring Autowire
5. Dependencies (jar) for Spring IOC
   1. Spring-core
   2. Spring bean
   3. Spring context



**Spring Configuration**

1. Spring configuration can be done using 2 ways
   1. **Using XML file and XML Tags**
      1. Create resources folder
      2. Create xml file
      3. Add following Xsd inside xml file

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beanshttp://www.springframework.org/schema/beans/spring-beans.xsd"*>

<!-- bean definitions here -->

</beans>

* 1. **Using Configuration class and Annotations**.
     1. Configuration Class is a java class with **@Configuration** annotation
     2. Use **@ComponentScan** annotation to provide the location of you component classes, so that spring will search all the component/bean classes in the given package.

**Spring Bean class**

1. These are the java classes whose object creation and object maintained by Spring container.
2. To convert any java class into spring bean class you have to configure it inside XML file or you can use Annotations provided by spring.
3. XML tags to set Java class as spring bean class.

<bean id=*"ObjName"* class=*"package.ClassName"*>

</bean>

1. Define Class as a spring bean class using annotation

**@Component annotation** or its sub annotation like **(@Controller, @Service, @Repository, @RestController** etc.)

**Spring Container**

1. Spring containers are the classes which create, holds and manage the spring bean classes.
2. There are 2 types of spring container available
   1. ApplicationContext
      1. Application context container is the advance container and support more functionalities than BeanFactory.
      2. This can be use for large applications like Web Application
      3. To get an application Context container there are multiple options provided by spring.
         1. Using **ClassPathXmlApplicationContext** when configuration is using XML file.
         2. Using **AnnotationConfigApplicationContext** when configuration is using Annotation.
   2. BeanFactory
      1. This container support less functionalities than the Application Context.
      2. This container can be use for small applications like Core Java application.
3. The objects created by spring can be get using the method provided by container.

container.getBean("ID", ClassName.**class**);

**Spring Bean Scope**

1. There are total 5 spring bean scopes
   1. **Singleton**: Is the default scope if you not explicitly provided. In this scope only a single object of the class will be created for the given id inside a spring container.
   2. **Prototype**: In this scope multiple objects will be created for a single id whenever use request to a container.
   3. **Request**: This scope is applicable for web application, in this scope new object will be created for every request.
   4. **Session**: This scope is applicable for web application, In this scope only one object gets created for a session.
   5. **GlobleSession**: This scope is applicable for web application, in this scope only one object present in a application which is share by all users.

**Setter Injection (SI)**

1. This is the process where you can pass the values for the variables using a setter method.
2. This can be achieved by using following tag in the XML file

<property name=*"variableName"* value=*"value"*></property>

**Constructor Injection (CI)**

1. This is the process where you can pass the values for the variables using a constructor.
2. This can be achieved by using following tag in the XML file

<constructor-arg value=*"value"*></constructor-arg>

**Dependency Injection (DI)**

1. This is a process where object of one class injected inside a Object of another class (HAS-A relation in core java).
2. DI can be achieved using SI or CI.
3. DI using SI

<property name=*"addr"* **ref=*"ad"***></property> <!-- DI by SI -->

1. DI using CI

<constructor-arg **ref=*"ad"***></constructor-arg> <!-- DI by CI -->

**Autowire**

1. Autowire is a process where you can automate the dependency injection(DI) process.
2. There are different type of autowire
   1. ByName: is use for a DI using SI. DI happens based on the name of your variable.
   2. ByType: is use for a DI using SI. DI happens based on the type of your variable.
   3. Constructor: is use for a DI using CI
   4. No: no DI injection happened automatically in this option.

**Annotations**

1. @Configuration
   1. This annotation is use to define any class as a configuration for the Spring.
2. @ComponentScan
   1. Using this annotation you can instruct spring to scan the given package for the bean class.
   2. Here you have to provide the package name which needs to scan for the bean classes.
3. @Component
   1. Using this annotation, you can define any java class as a spring bean class.
   2. The object of these java classes will create by spring after scanning the package
   3. There are child annotations which can be used for the same purpose,

@Controller, @RestController, @Service, @Repository, @ControllerAdivce etc.

**Spring Boot**

1. Spring boot is a wrapper of spring framework.
2. Spring boot is use for faster development and for minimum configurations.
3. Spring Boot also provides an easy way to manage the dependencies.
4. Spring boot provided an auto configuration feature to configure you application automatically.
5. These configurations can be easily customize using the properties file present inside springs boot.
6. Spring boot has an embedded tomcat service by default.
7. You can create spring boot application by following options.
   1. STS (Spring Tool Suite) IDE

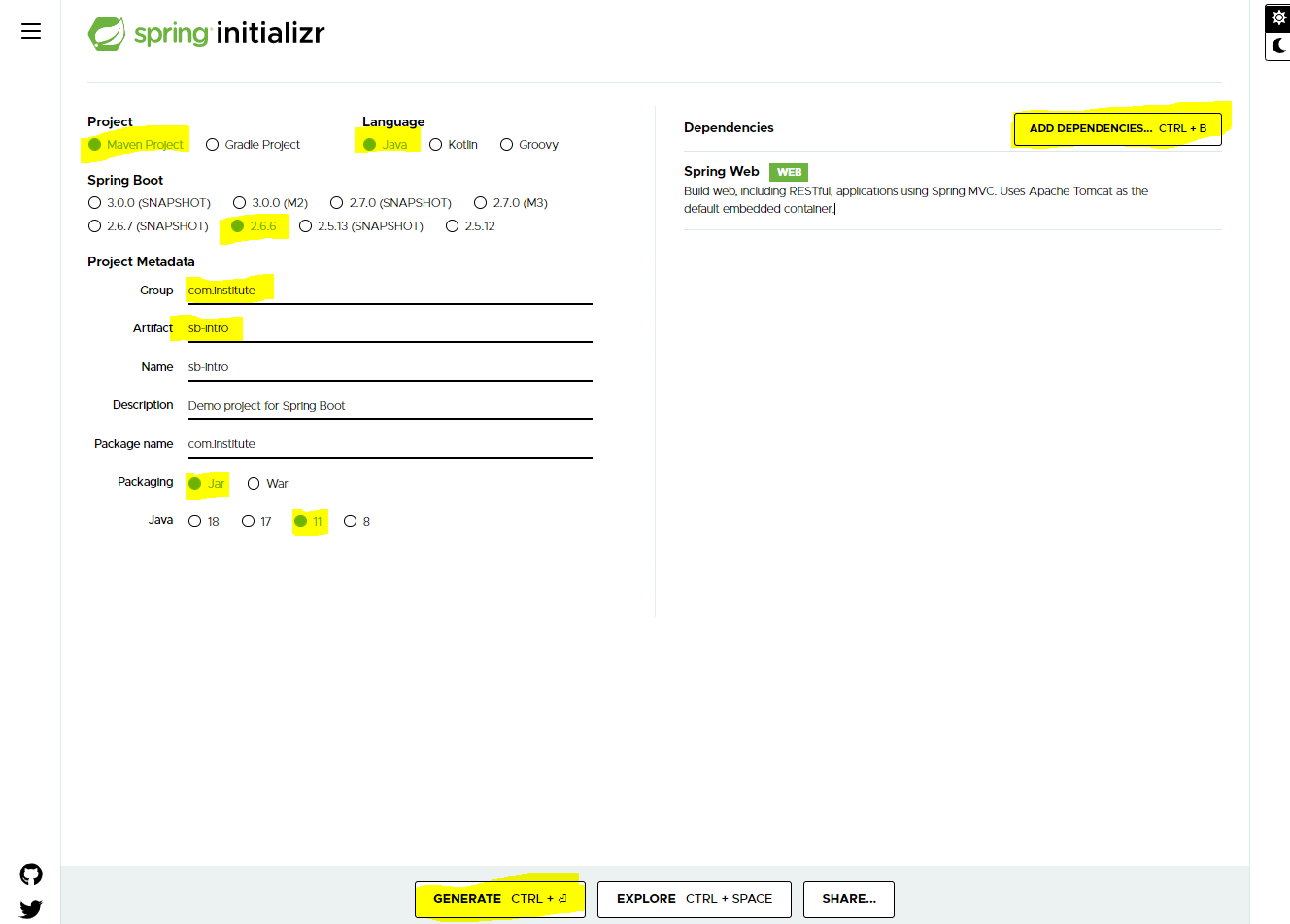
<https://spring.io/tools>

* 1. Using CLI (Command line interface)

<https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html#getting-started.installing.cli>

* 1. Using Spring Inititalizer web application

<https://start.spring.io/>



**Create Spring Boot Project using Spring Initializer**

1. Go to Spring Inititalizer portal

<https://start.spring.io/>

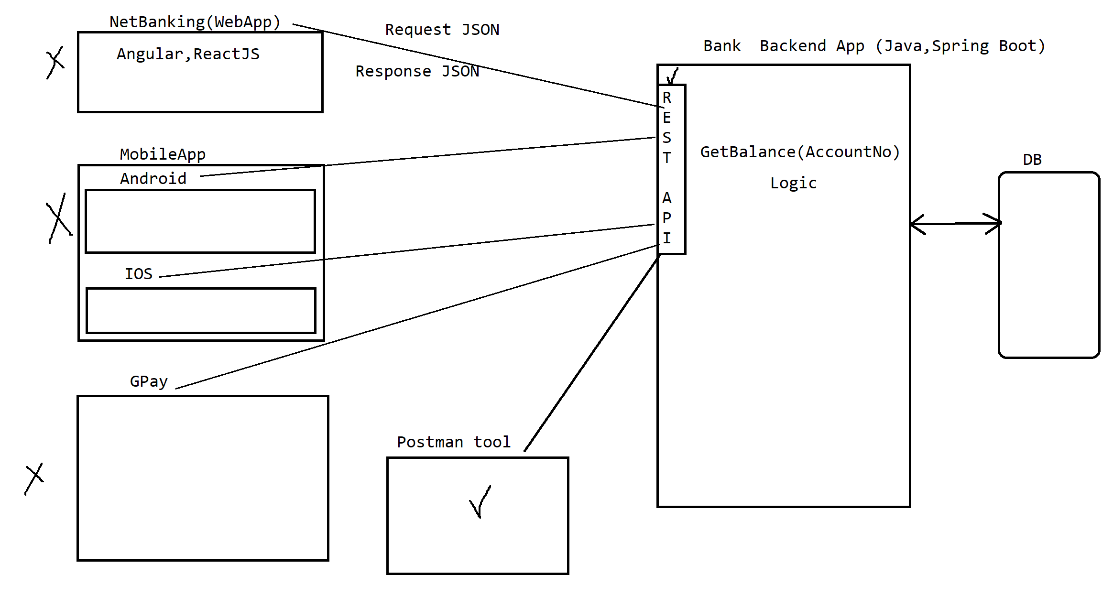
1. Fill up the details on the portal and click on Generate Button.
2. Get the .zip file which is downloaded.
3. Extract the zip file into your eclipse workspace
4. Follow the steps to open project into Eclipse
   1. Got To “File” Menu -> “Import” -> Search for “Maven” -> select “Existing Maven Projects”
   2. Click On “Next” -> Set the path of the project where you extracted the zip file.
   3. Make sure that the project pom.xml must be listed into a Projects section.
   4. Click On Finish and wait for the Project setup to complete.

@SpringBootApplication

1. This annotation is use on the Main class of the spring Boot application.
2. This annotation is a combination of 3 annotations
   1. @Configuration
   2. @ComponentScan
   3. @EnableAutoConfiguration

REST API

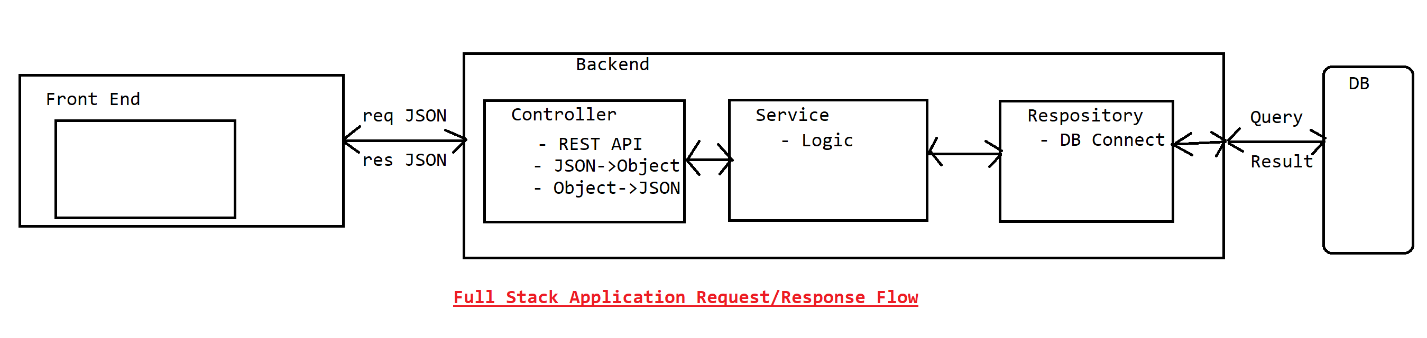
1. REST API stands for **RE**presentation **S**tate **T**ransfer **A**pplication **P**rogramming **I**nterface.
2. Rest APIs can be consumed by any application irrespective of their language. Can access it cross platform.
3. Rest API use HTTP protocol.
4. The communication with the REST API can be done using XML or JSON language.



Download and Install Postman to Test Rest API

<https://www.postman.com/downloads/>

Rest API Application Flow In Full stack development



@RestController

1. This annotation is use to make you java class ready to develop REST API.
2. This annotation is a child annotation of the @Controller (Used for Non-Rest API app/MVC app)
3. This annotation will help us to expose the rest API to the external application and also It will help us to convert JSON into Java Object and Java Object into JSON.

@RequestBody

1. This annotation helps to set the JSON value into a Java Object.

**JSON**:

1. **J**ava **S**cript **O**bject **N**otation
2. JOSN is a way to exchange a data between 2 application which can be develop in different languages and running on different platform.
3. This is a common way of data representation which can be used by all programming languages.
4. In the JSON data will be store in the form key and value like Map in java.
5. JSON structure can be represented by JSON Object or JSON array.
6. In Java every JSON structure can be convert it Java class/Object and wise versa.
7. There are multiple pre define libraries are use to do these conversion(JSON->Java Object and Java Object to JSON) like gson, Jaskson(default for Spring Boot).
8. Example:

**JSON Object**

{

"id":12, -----> JSON Attribute/Element

"Name":"Abc",

"contact":"76876987665",

"email":"abc@gmail.com",

"salary": 456456.45

}

**JSON Array (Contains the value)**

[34,56,98]

**JSON Array (Contains the JSON Object)**

[

{

"id":12,

"Name":"Abc",

"contact":"76876987665",

"email":"abc@gmail.com",

"salary": 456456.45

},

{

"id":13,

"Name":"Xyz",

"contact":"99876987665",

"email":"xyz@gmail.com",

"salary": 256456.45

},

{

"id":14,

"Name":"Pqr",

"contact":"99876987665",

"email":"pqr@gmail.com",

"salary": 87876.45

}

]

**Nested JSON Object**

{

"id":121,

"Name":"Abcd",

"address": {

"city":"Pune",

"state":"MH"

}

}

Nested JSON Object and JSON Array

{

"id":121,

"Name":"Abcd",

"contact":["678678676","986786787"] ------> JSON Array of values

"eduDetails": [{ -------> JSON Array of JSON Object

"title":"10th",

"percent":89.12

},

{

"title":"12th",

"percent":79.12

},

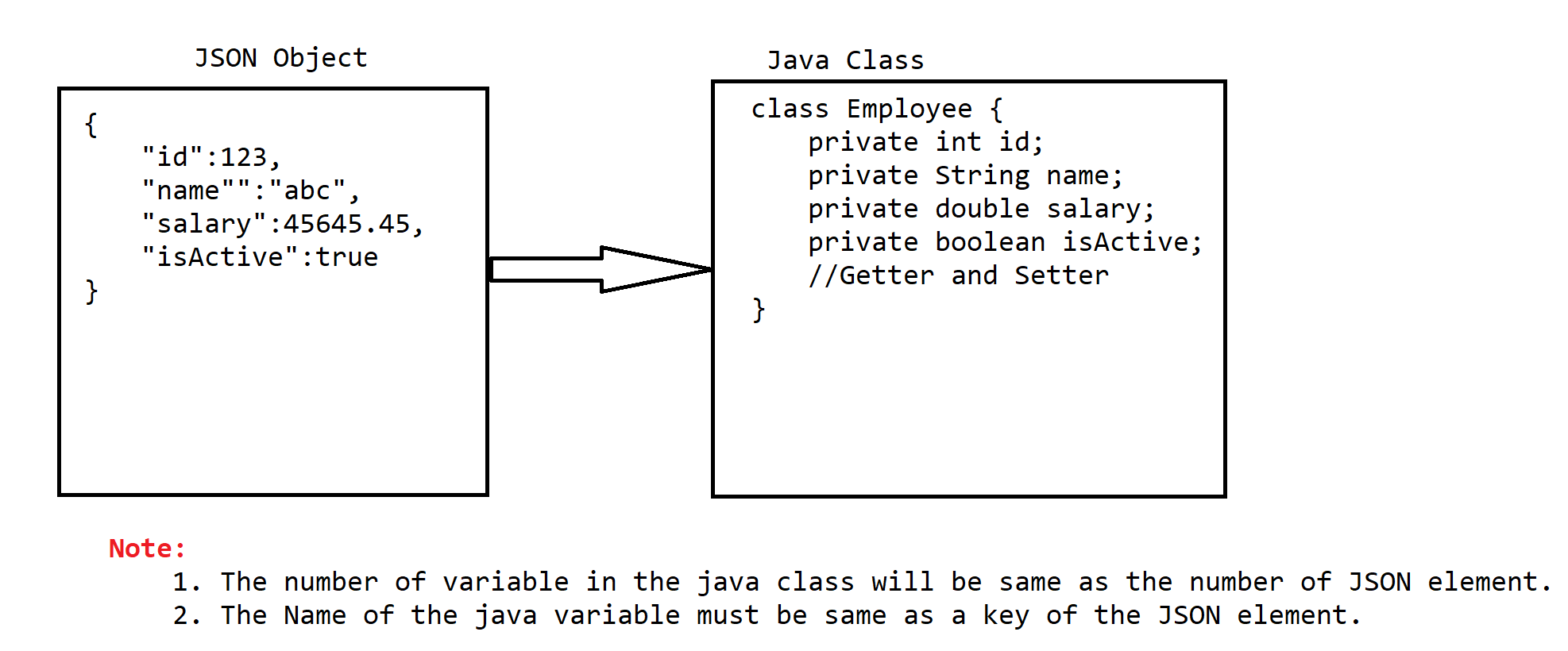
{

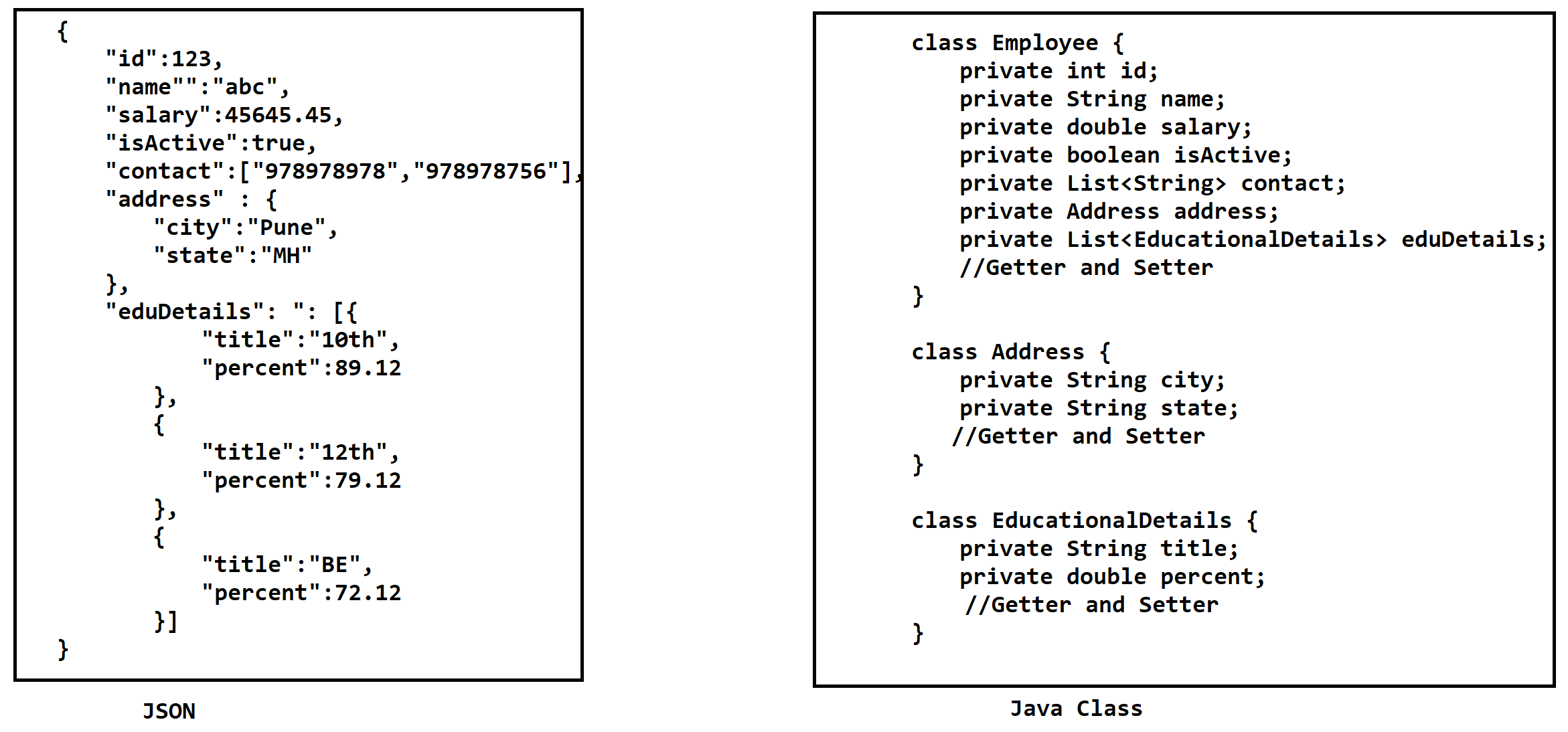
"title":"BE",

"percent":72.12

}]

}





**HTTP Methods**

1. There are HTTP methods which are recommended to use for a different type operations.
2. GET:
   1. It is recommended to use **GET** method to return a response for a REST API to a user.
   2. In Spring it can be achieve by using **@GetMapping** annotation
3. POST:
   1. Is use to create a new entry in the backend.
   2. Mostly it is use to insert a new record into a backend.
   3. It can be created using **@PostMapping** annotation in Spring
4. PUT:
   1. Use this method to update the existing entry in the backend.
   2. To update the record in the DB/backend
   3. It can be created using **@PutMapping** annotation in Spring.
5. DELETE:
   1. Use to remove any entry from the backend.
   2. It is use to delete the record from the Backend/DB
   3. It can be created using **@DeleteMapping** annotation in Spring

**Swagger**

1. Swagger is a tool to generate the REST API documentation.
2. Along with Documentation you can also test you REST API just like a postman.
3. Swagger is also known as OPEN API in the latest version.
4. To generate a Swagger document of your REST API you have to follow the steps
   1. Add Swagger Dependency.

<dependency>

<groupId>org.springdoc</groupId>

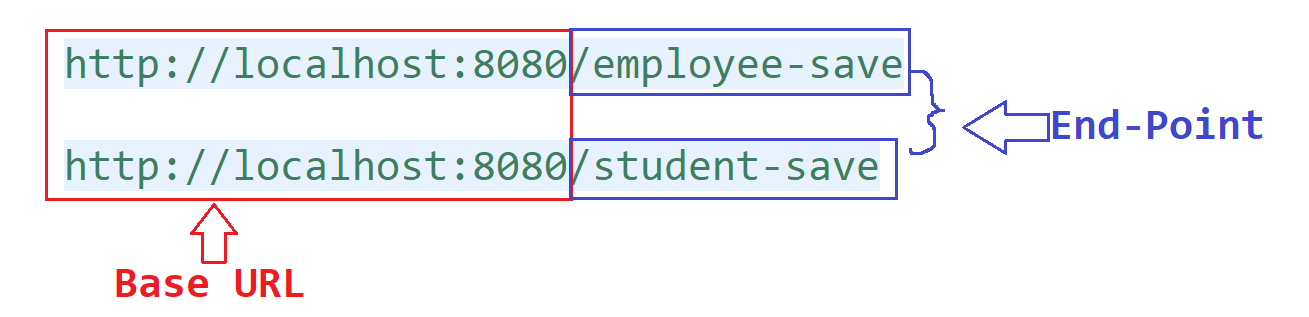
<artifactId>springdoc-openapi-ui</artifactId>

<version>1.6.7</version>

</dependency>

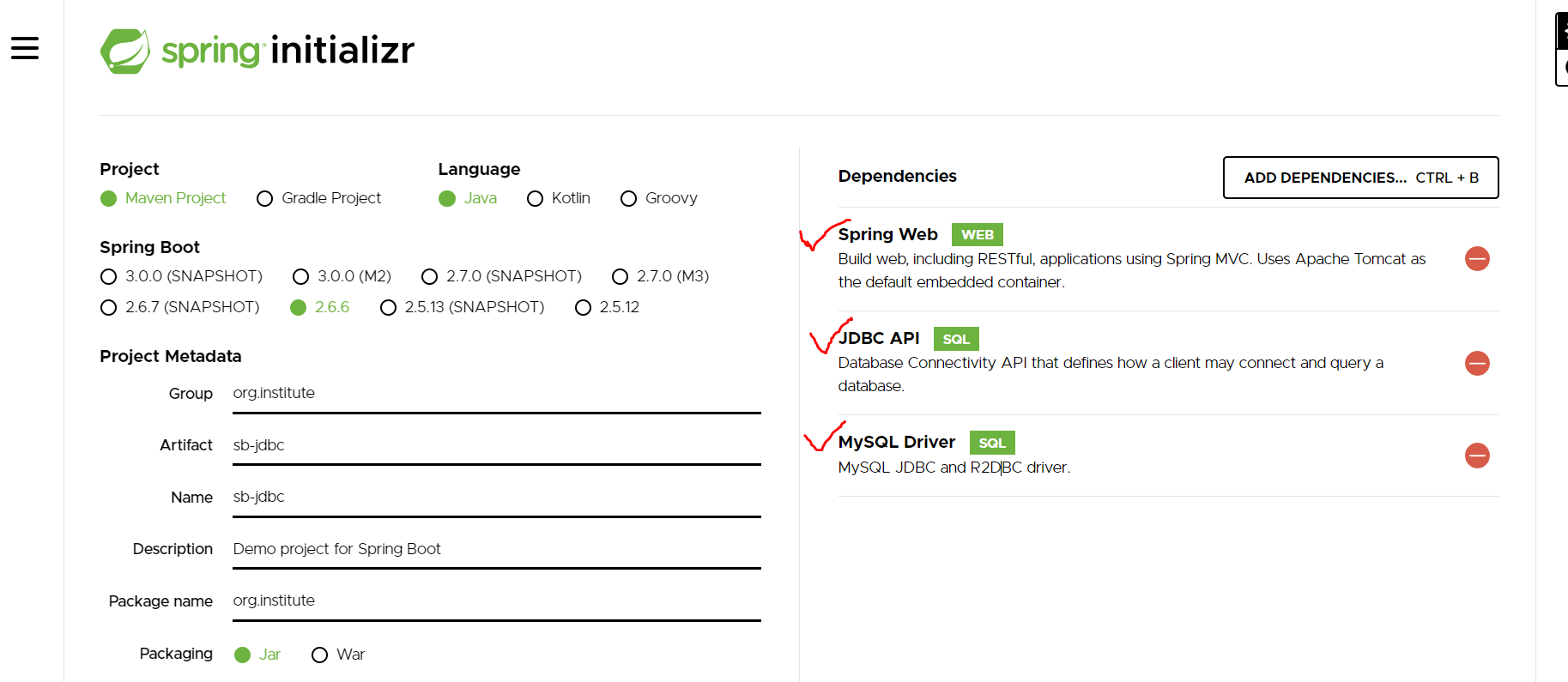
* 1. To Access Swagger UI use following URL

<http://localhost:8080/swagger-ui.html>

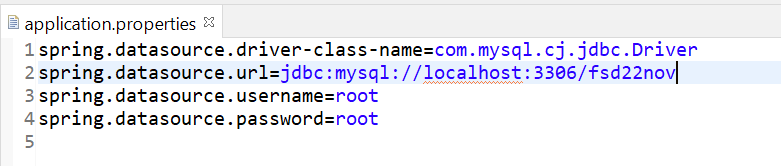


**Spring JDBC**

1. This spring module help you to connect with the database in more efficient way.
2. In the JDBC You have to use Spring predefine class named as JdbcTemplate and DataSource
   1. **DataSource** Class is use to connect with the Database and provide the data base connection to execute the SQL statements
   2. **JdbcTemplate** Class is use to execute the queries and get result back into a java application.
3. To Use JDBC ins spring you have to configure the URL, UserName, password and Driver in the application.properties file.
4. Step to Use Spring JDBC in the Spring Boot Application
   1. Create Spring Boot Application and import into Eclipse.



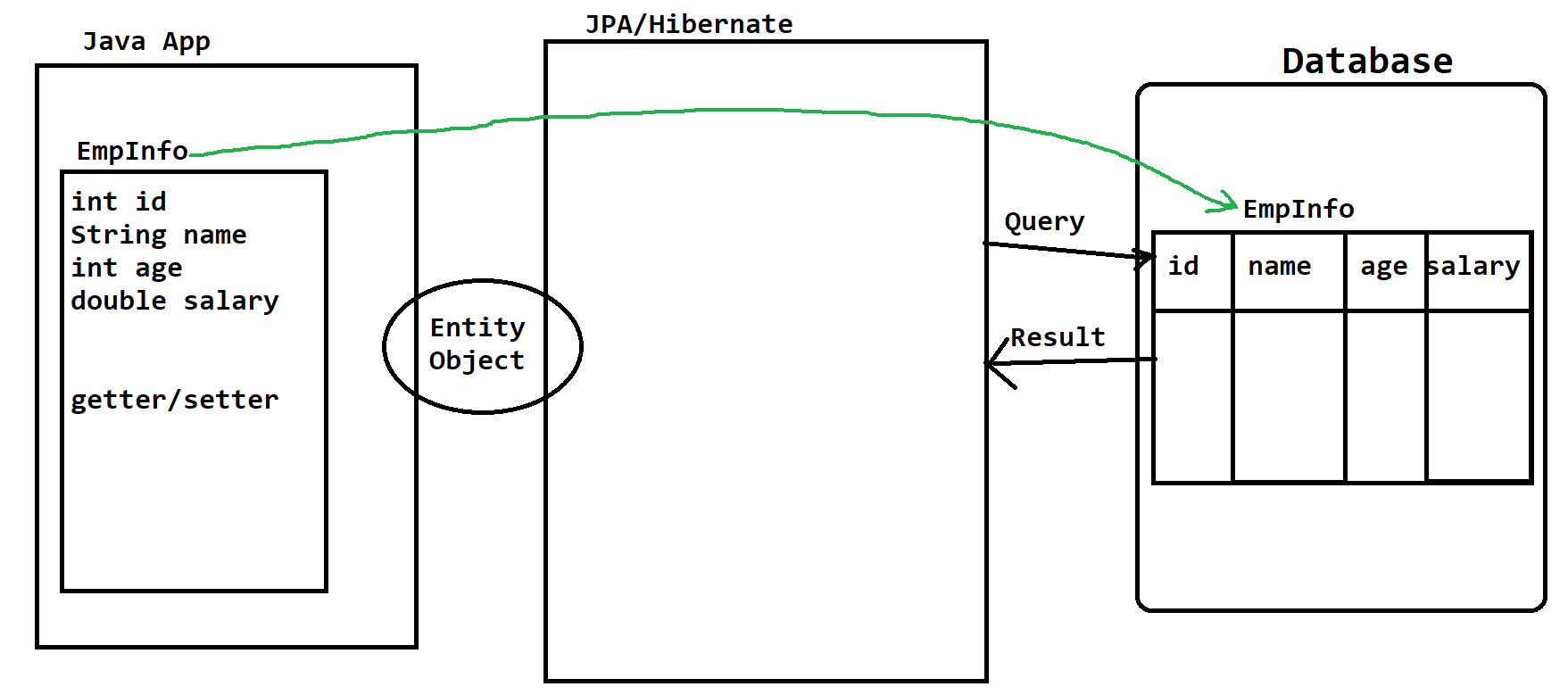
* 1. Provide DB Configuration(URL, username, password, driver) into application.properties file



* 1. Use JdbcTemplate Object into a spring application to execute your queries and get the result back. To use JdbcTemplate Object you need to autowire this object.

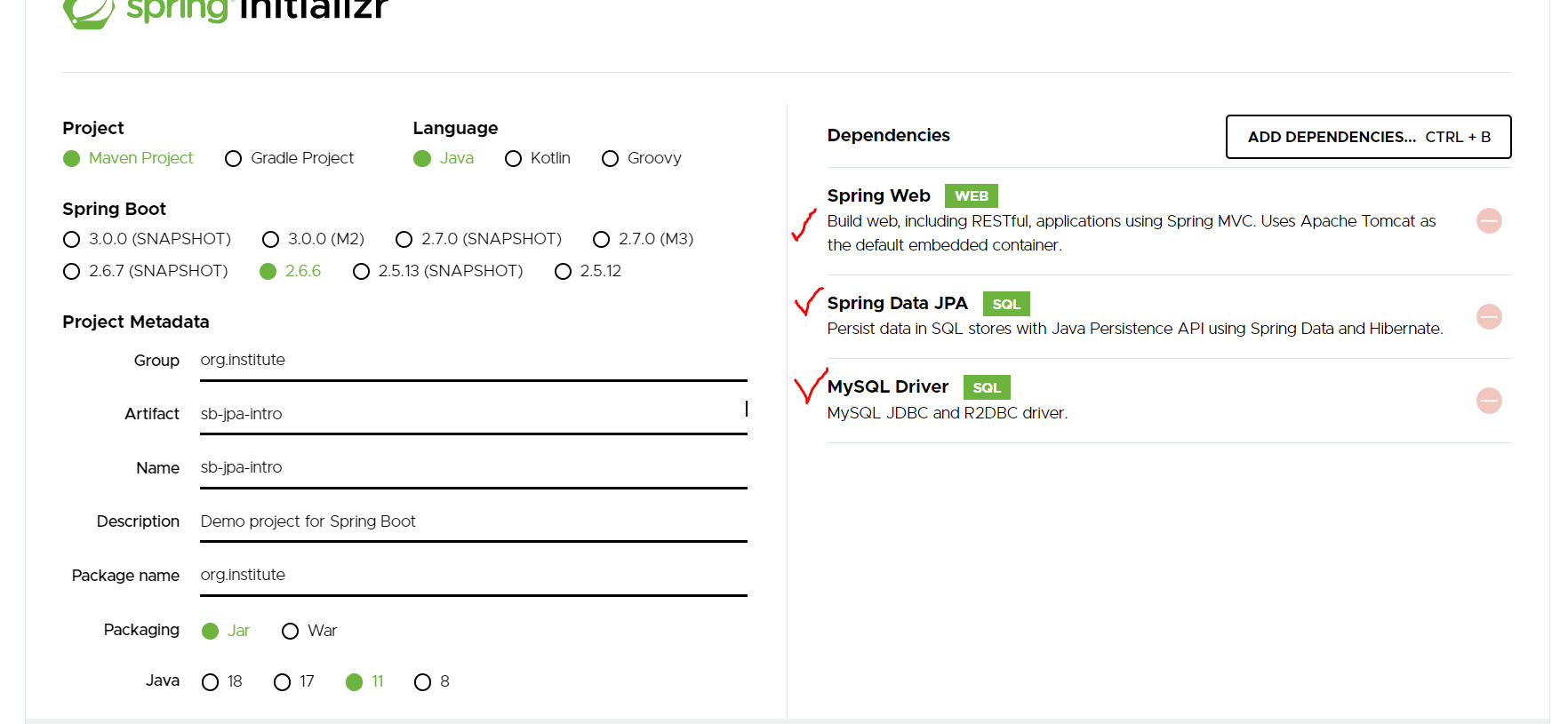
**Spring Boot ORM (JPA/Hibernate)**

1. ORM is the Object Relational Mapping.
2. The Java Objects will be mapped with the table in the Database.
3. There are multiple ORM tools/framework available which can be use independently like JPA, Hibernate, EJB.
4. In the ORM you will get the following feature
   1. These tools can create/generate and Execute DDL and DML queried internally.
   2. ORM tools handles the Database checked exception internally. And provides us as an Unchecked exception.
   3. ORM tools also support the caching techniques. To improve the performance of the application.
   4. ORM tool like Hibernate implemented primary cache internally and also you can implement third party cache explicitly as a secondary cache in hibernate.
   5. ORM tools support the collection mapping.
5. In the ORM tool you will required an **Entity classes** which will Mapped with the table.
6. These entity classes are useful to interact with the hibernate and we will always communicate with the ORM toll using Entity class Object.
7. Every Entity class must have a primary key.
8. To Convert java class into Entity class use **@Entity** annotation on the class declaration and to make any variable as primary key use **@Id** annotation on the variable declaration.

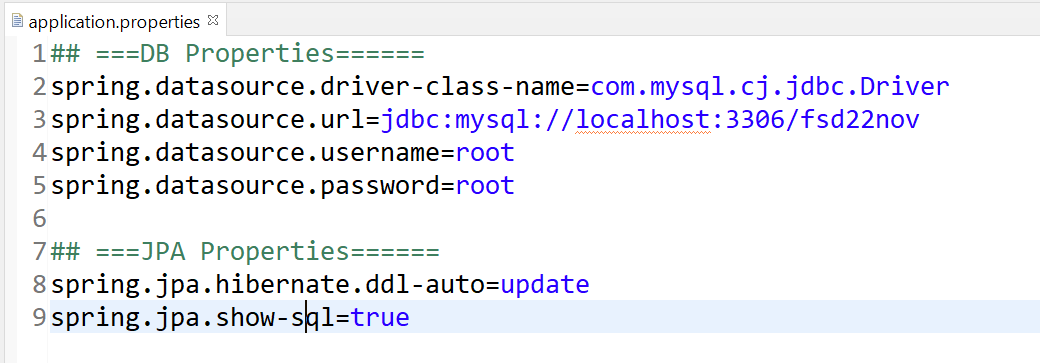


Steps To Create Spring JPA Project

1. Create Project, Add Dependencies and Import into Eclipse



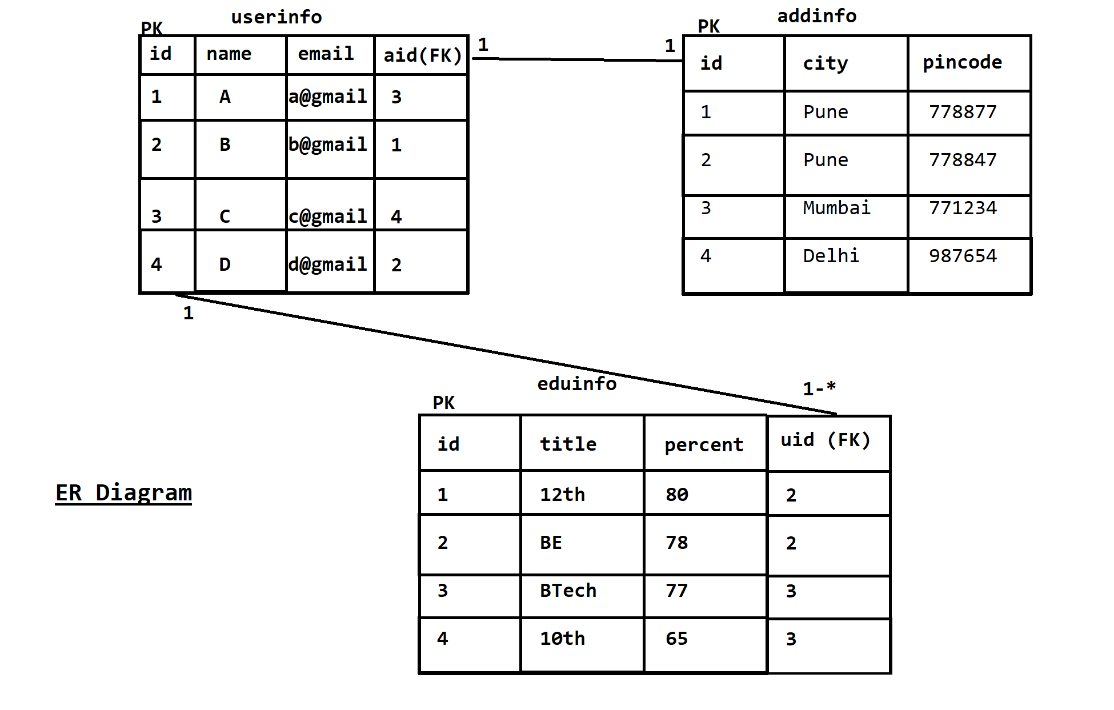
1. Add DB and JPA configuration

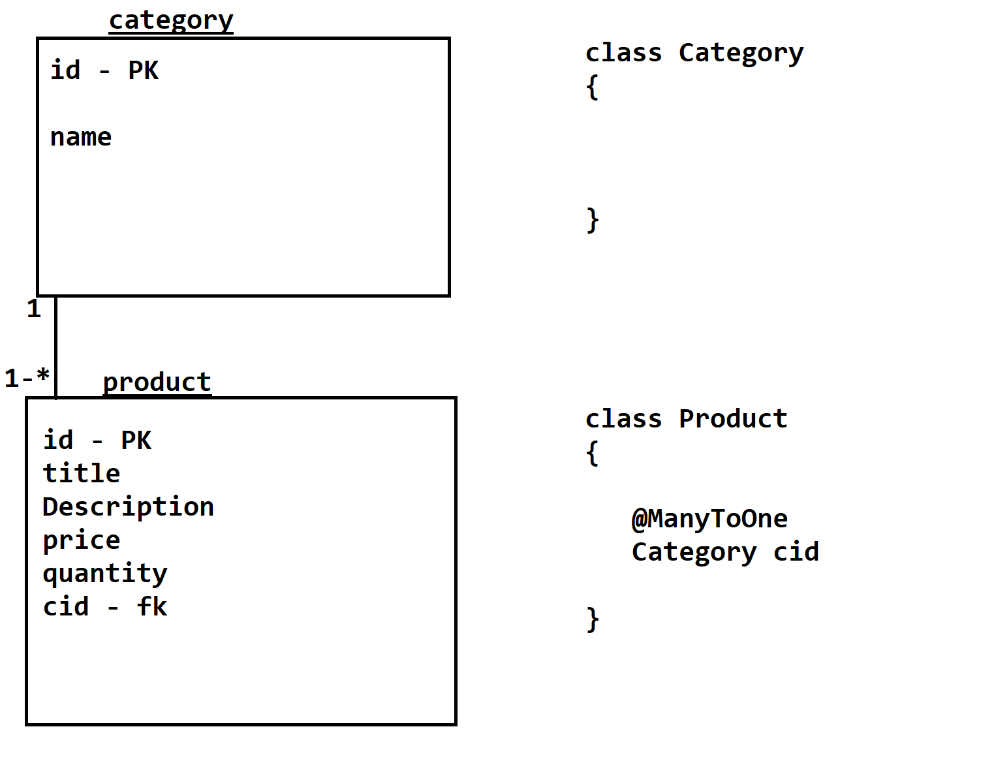


1. Create Entity Classes
2. Create Repository interface.
   1. Repositors are the interfaces which extend **JpaRepository Interface** provided by Spring.
   2. JpaRepository interface has all the basic operation pre define so that you need not have to write query for basic and common operations.
   3. Basic Operation as follows
      1. save(Object) : You can insert or update the record.
      2. findAll() : you can select all the records from the Table.
      3. deleteById(ID) : to delete a record by ID.

**Relational Mapping in the JPA:**

1. In this you can work with the multiple table which is having relation between them.
2. There are 4 type of relational mapping in JPA
   1. @OneToOne
   2. @OneToMany
   3. @ManyToOne
   4. @ManyToMany
3. To create a foreign key, you have to create a reference of the one class into another class.
4. The reference of the original class will be created into the class where you wanted to create a foreign key.
5. Annotate this reference with any one of the annotation mention above.
6. Fetch type for OneToOne and ManyToOne is by default EAGER and for OneToMany and ManytoMany id LAZY
7. Fetch type:
   1. EAGER: It will select the records from the original table along with all the related tables by applying joins or sub query
   2. LAZY: It will only select the records form the original table and not from the related table.





**Important Links:**

Spring Properties : <https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

Spring Boot Application Creation: <https://start.spring.io/>

Spring Boot Documentation: <https://docs.spring.io/spring-boot/docs/current/reference/html/>

Swagger Documentation URL : <http://localhost:8080/swagger-ui.html>