**Spring Boot**

1. Spring Boot is a java-based framework.
2. Spring Boot is used for the faster Development and it also reduce the development efforts.
3. Using Spring Boot you can develop a product ready applications.
4. Spring Boot also provides the embedded tomcat server and the database such as H2 Database.
5. Spring Boot provides multiple predefine functionality using which with min configuration you can achieve the maximum functions. This are also customized as per requirement.
6. Spring Boot is based on Spring Framework.
7. Spring Boot is used for Creating Full-stack or Micro services application.
8. Spring boot is easy to learn and also Implement because it is distributed into a multiple modules, and to use spring boot you do not have to lean all the spring modules.
9. You can use the specific module as per your project requirement.
10. Some Examples of spring boot modules
    1. Spring IOC/Core
    2. Spring JDBC
    3. Spring ORM
    4. Spring REST
    5. Spring Batch
    6. Spring MVC etc..
11. Spring Boot is a wrapper of Spring Framework. Spring boot has provided an easy way to create a project, to add multiple modules dependency and configuration.
    1. Spring Boot Provided the starter projects using which you can add the dependencies easily and just by adding single dependency.
    2. Spring boot also provide the auto configuration functionality using with you will get the default configuration from the maven which is highly customizable.
    3. Spring Boot provides the Embedded tomcat server and H2 Database so that you can run the application with minimum setup.

**Create Spring Boot Project**

1. Spring CLI (Command Line Interface)
   1. In this approach you have to install CLI tool and using the command you can create spring boot project.
2. Spring STS (Spring Tool Suit)
   1. Spring suit is and IDE which is created as an extension on eclipse and IntelliJ IDE

<https://spring.io/tools>

1. Spring Initializer
   1. This is an web application provided by spring to create a spring boot project.
   2. Here, you can provide the details for creating spring boot application and you can download the application will all the features provided.

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

**Spring Common Properties**

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

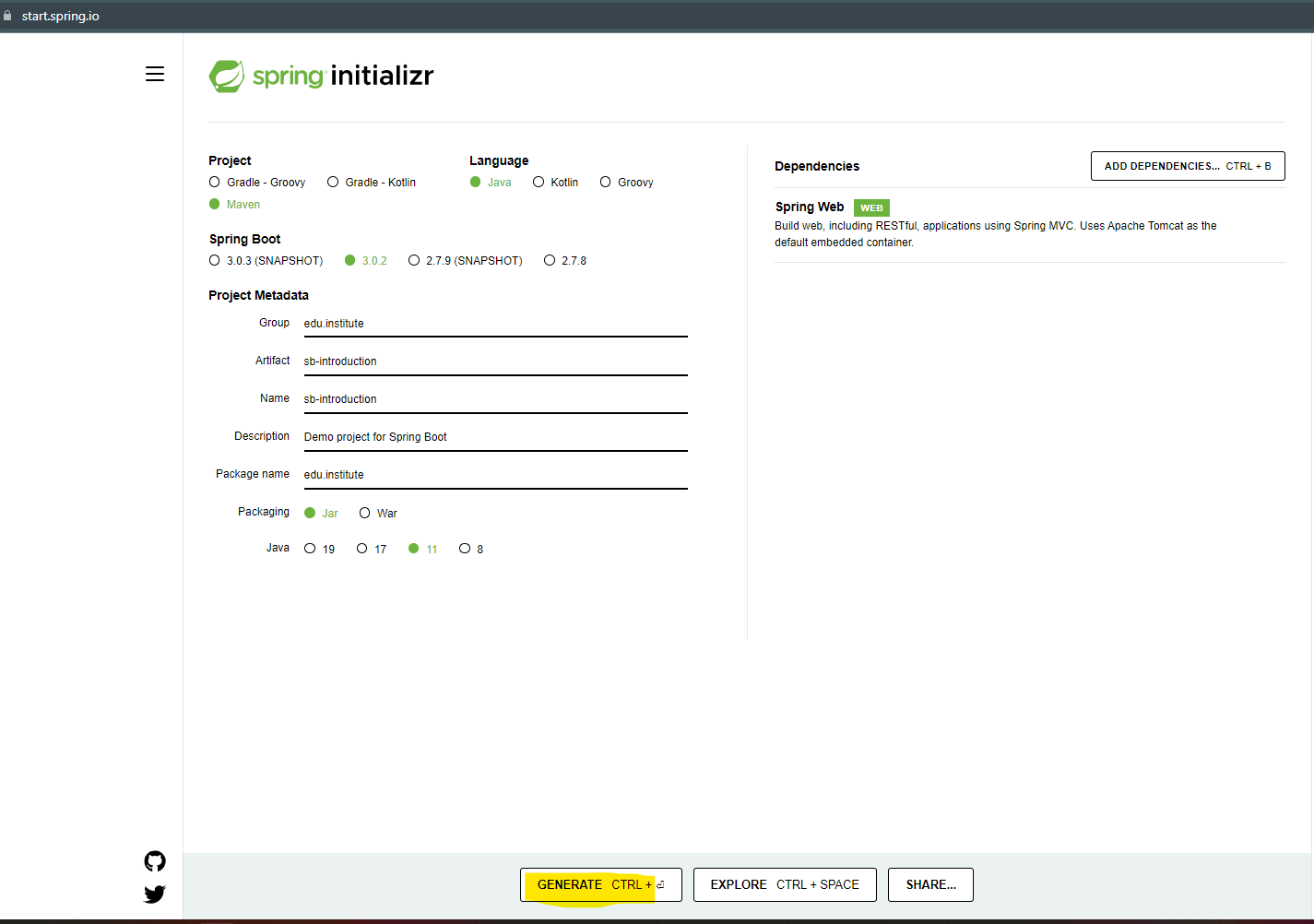
**Steps to create and import Spring boot Project in Eclipse**

1. Create Spring Boot Project

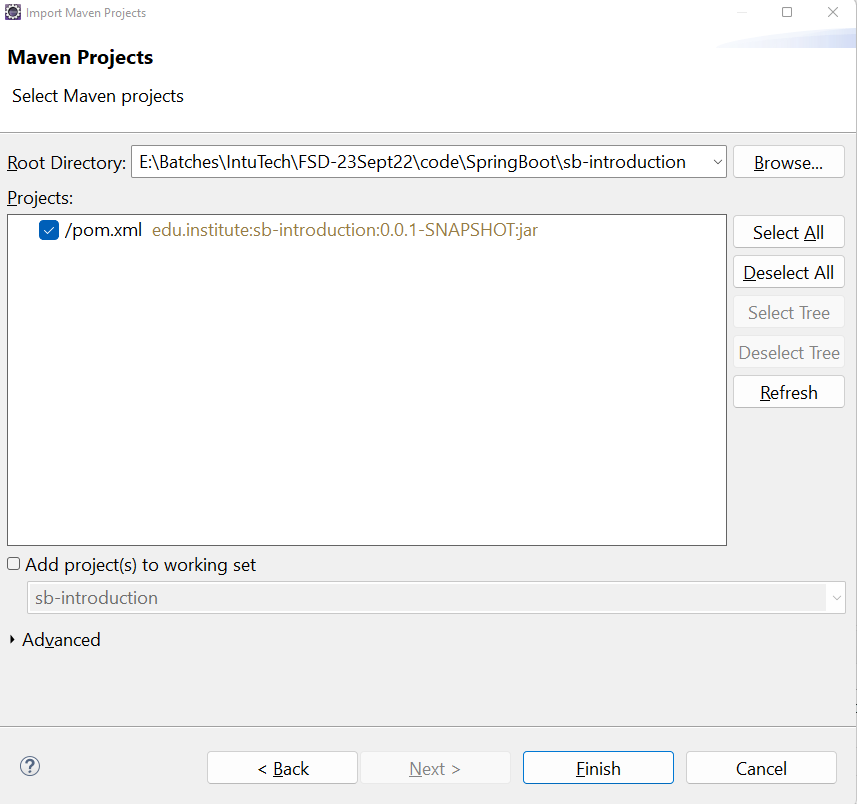
Go to <https://start.spring.io/>

Fill up the project details into form

Click on Generate button. It will download the ZIP file.



1. Import the project into Eclipse
   1. Copy and paste the project ZIP file into eclipse workspace
   2. Extract the ZIP file.
   3. Open an Eclipse workspace
   4. “File” menu -> click “Import” Option
   5. Search for “Maven” Option in the Wizard -> Select the “Existing Maven Projects” option and click on “Next”
   6. Browse for an extracted project location and select the parent folder of the pom.xml file.
   7. Click on Finish



**Spring Boot Application Main Class**

1. This class is present in all the spring boot project
2. This class is use to start/run the spring boot application
3. In this class there are 2 statements which help to start with spring boot project
   1. **@SpringBootApplication** Annotation
      1. This annotation denote the application is a spring boot application.
      2. This annotation is a combination of 3 annotations
      3. **@Configuration** annotation is use to load the configurations. All the configurations are loaded inside spring application at the start of the project.
      4. **@EnableAutoConfiguration** annotation is use to enable the spring boot default configuration. The configuration such as Tomcat Server configuration, DB configurations are added inside application
      5. **@ComponentScan** annotation is use to load all the spring bean class object inside spring container.
4. **Execution of the Run methods**
   1. This method is use to start the spring boot application with the startup arguments.
   2. This method return the Object of Spring Container which is ApplicationConext.

**Spring IOC/Core**

* + - 1. Spring IOC is an invers of Control
      2. In this module you can learn how to create spring bean classes.
      3. Spring container which is use to create and maintain the object of spring bean classes
      4. In this module you can perform the Dependency Injection (DI) process which is equivalent to HAS-A relation in java.
      5. The Dependency Injection process can be automated using the Autowire process

**Spring Bean Class**

1. It is a java class (build-in or custom class) whose object will be created and maintain by the spring.
2. To mark your custom class as a spring bean class you can use following annotation
   1. @Component
   2. @Controller
   3. @RestController
   4. @Service
   5. @Repository
   6. @ControllerAdvice

**Spring Container**

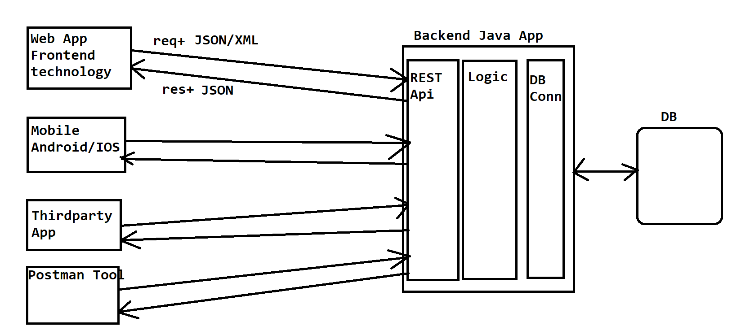
1. Spring Container is a predefine component of the spring
2. Spring container is responsible to scan all the bean classes and create and manage object of Spring Bean class.
3. You can get the Object of the class using spring container
4. ApplicationContext is a container which can be use to get a spring bean objects.

**Spring Dependency Injection (DI) and Autowire**

1. Dependency Inject (DI) is a process in which you can create object of one class into another class. In Java it is also known as HAS-A relation.
2. To automate the DI process you can use autowire concept.
3. TO perform the auto wiring you have to use @Autowire annotation.
4. You can use the autowire annotation only for the spring bean class.

**REST API**

1. REST API is also known as Restful APIs
2. Using REST API you can create a web services which can be consume by frontend application or any other application or any third party.
3. REST stands for **RE**presentational **S**tate **T**ransport
4. This use to manage the communication over a network.



**JSON**

1. JSON is a JavaScript Object Notation
2. JSON is use to exchange the data between two application which is developed in different technologies and languages.
3. JSON is used in the form of Key and Value pair which is also as element.
4. Where Key is in String format and values can be string, numeric, Boolean, JSON object or JSON array.
5. JSON can be use in two forms
   1. JSON Object
   2. JSON Array
6. Example of JSON

{

"name":"John",

"age":30,

"car":null,

"flag":false

}

**JSON Object**

1. JSON Object can be represented as curly brackets

Symbol:

{

JSON Elements

}

**JSON Array**

1. JSON Array represented as a square bracket

Symbol:

[

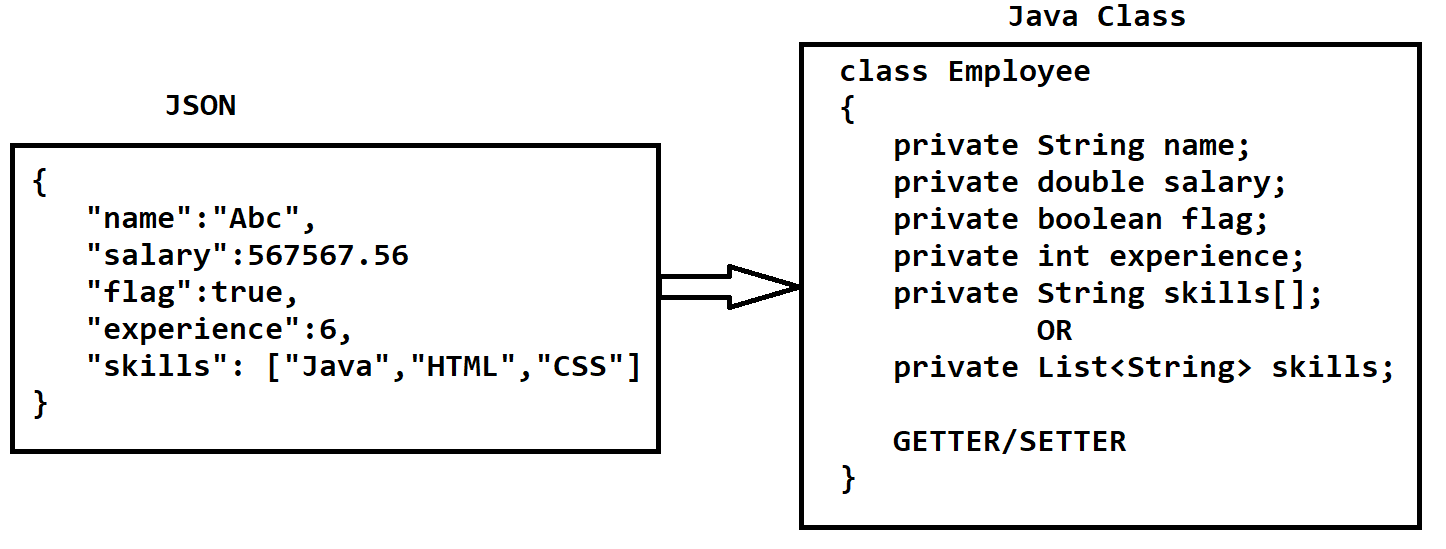
Value/JSON Object

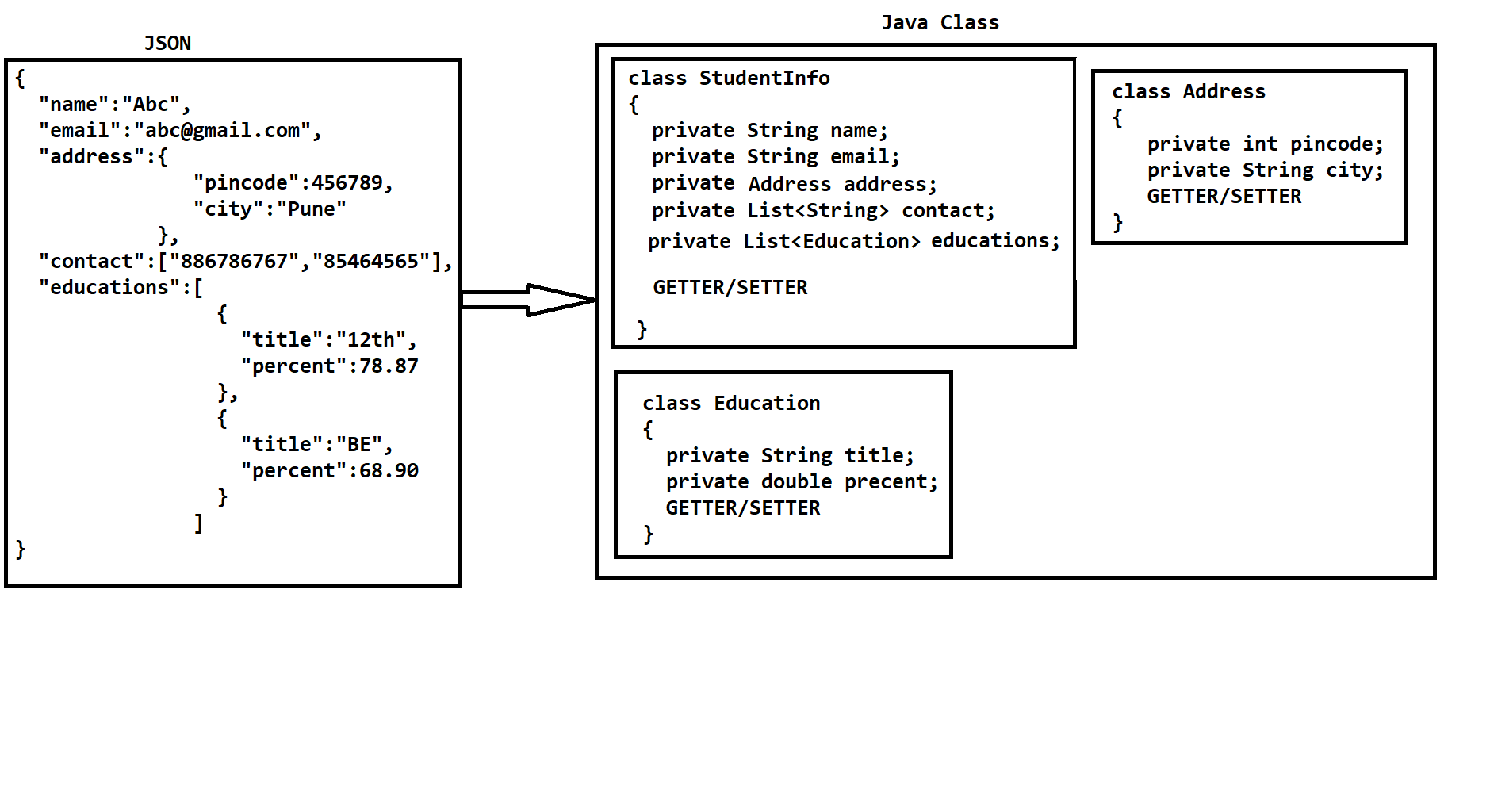
]

In Spring/Java application JSON can be convert into Java format

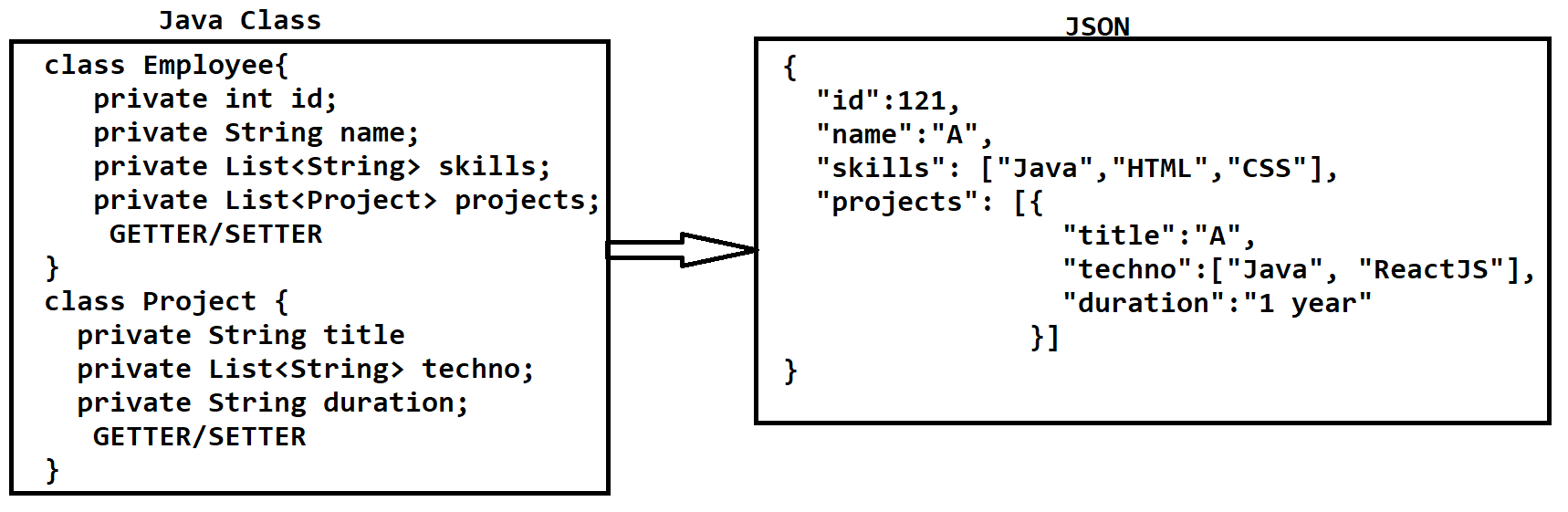
1. Request JSON has to convert into java Object. (JSON 🡪 Java Object)
2. To return the response Java Object has to convert into json. (Java Object 🡪 JSON)
3. To do this conversion you can use GSON or Jackson libraries.
4. Spring boot internally use Jackson library for this conversion.

**JSON To Java Object**





Java Class to JSON



**Steps to create REST API in Spring Boot**

1. Create a java class
2. Annotated the java class with @RestController annotation.
3. Create a methods with input parameter and return type as per you REST API requirement.
4. Annotate the method using @GetMapping, @PostMapping, @PutMapping or @DeleteMapping for GET, POST, PUT or DELETE method of HTTP respectively.
5. Provide the URL for the REST API.

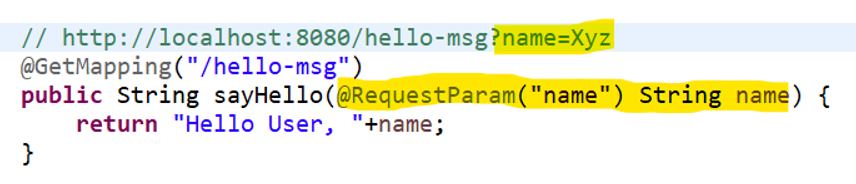
**Postman**

1. Postman is a tool using which you can test the REST APIs.
2. You have to install the Postman tool on your local

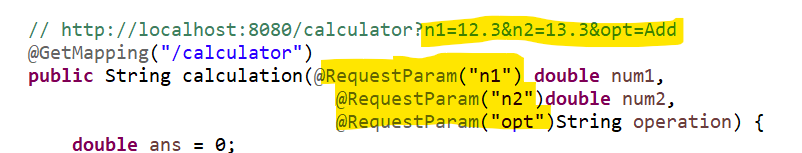
<https://www.postman.com/downloads/>

**Accept the Data from the request using REST API**

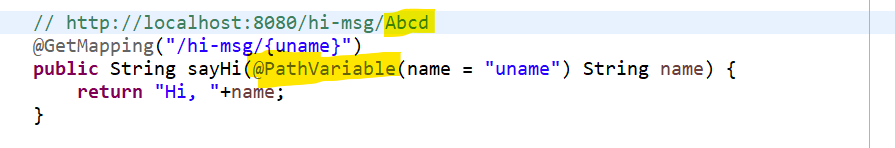
1. You can accept the use data in REST API using 3 approaches.
2. **Using Request Parameter**
   1. In the request parameter you can pass the data in the form of parameters in the URL.
   2. In Spring You can receive the request parameter values as a input parameter of the method
   3. The Input parameter has annotated by **@ReuqestParam** annotation.
   4. Example1:

****

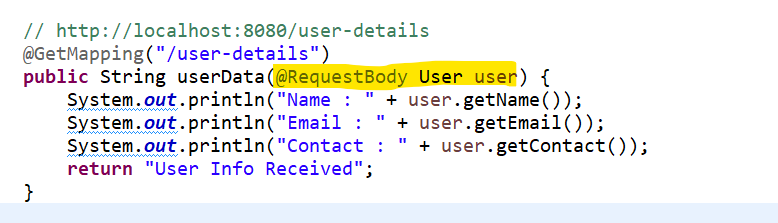
* 1. Example2:

****

1. **Using Request Path Variable**
   1. You can accept the user info as a part of url itself instead of creating parameter.
   2. This will be a dynamic part of the URL.
   3. To receive such type of values in the application you have to use **@PathVariable** annotation for the input parameter.
   4. There can be a multiple path variable in a URL.
   5. Example:

****

1. **Using Request Body in from of JSON**
   1. The data can be accept as a JSON from the REST API.
   2. This JSON data gets converted into Java Object automatically by Spring.
   3. To Set the data into Java Object you have to make sure that the Key of the JSON element has to match with the variable of java class.
   4. To receive the JSON data in form of Java Object you have to create a reference of java class as a input parameter and use **@RequestBody** annotation for the input parameter.
   5. Example:



**Http Methods for REST API**

1. GET
   1. GET method is recommended to retrieve the records from the backend application.
   2. To create HTTP GET method you can use a **@GetMapping** annotation.
   3. GET is an **idempotent** method. Which means if you use same request multiple times then it will return the **same result**.
2. POST
   1. POST method is use to create/insert the new records in the backend application.
   2. To create HTTP POST method you can use **@PostMapping** annotation.
   3. This metho is **non-idempotent**. If you execute the same request multiple time then it will create new entries every time.
3. PUT
   1. PUT method is use to update the existing record from the backend.
   2. To create HTTP PUT method you can use **@PutMaapping** annotation.
   3. This method is an **Idempotent**. if you use same request multiple times then it will return the **same result**.
4. DELETE
   1. DELETE method is use to delete/remove the existing record.
   2. You can use **@DeleteMapping** annotation to create HTTP POST method.
   3. This method is an **Idempotent**. if you use same request multiple times then it will return the **same result**.

**REST API Documentation**

1. API documentation will contain details about the rest API. Such as URL, Request, Response.
2. This documentation can be generated automatically without any manual effort using Swagger Tool.
3. Swagger is also known as Open API in the its latest version.
4. Swagger will provide the UI on which user can refer the API documentation.
5. Along with API documentation you can also execute the API using Swagger.
6. To More details you can refer this link <https://springdoc.org/>
7. Steps to add swagger tool in spring boot application
   1. Add swagger/openapi Dependency

<dependency>

<groupId>org.springdoc</groupId>

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

<version>1.6.14</version>

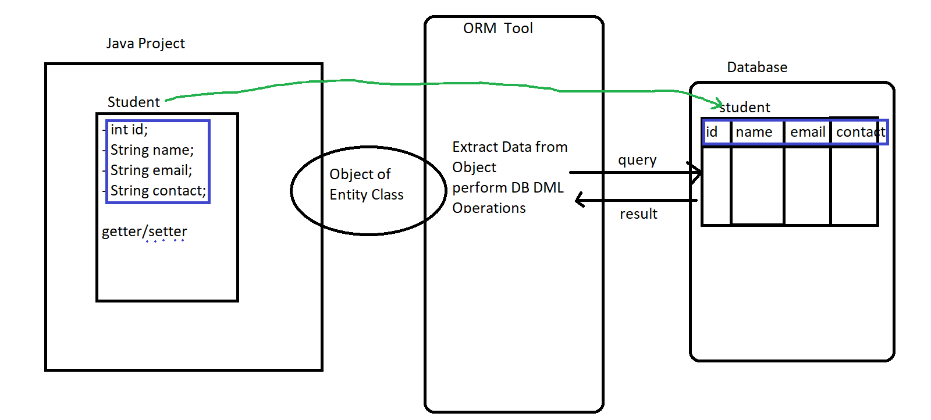
</dependency>

* 1. Access the URL to get API documentation

[**http://localhost:8080/swagger-ui.html**](http://localhost:8080/swagger-ui.html)

**Spring Boot ORM/JPA**

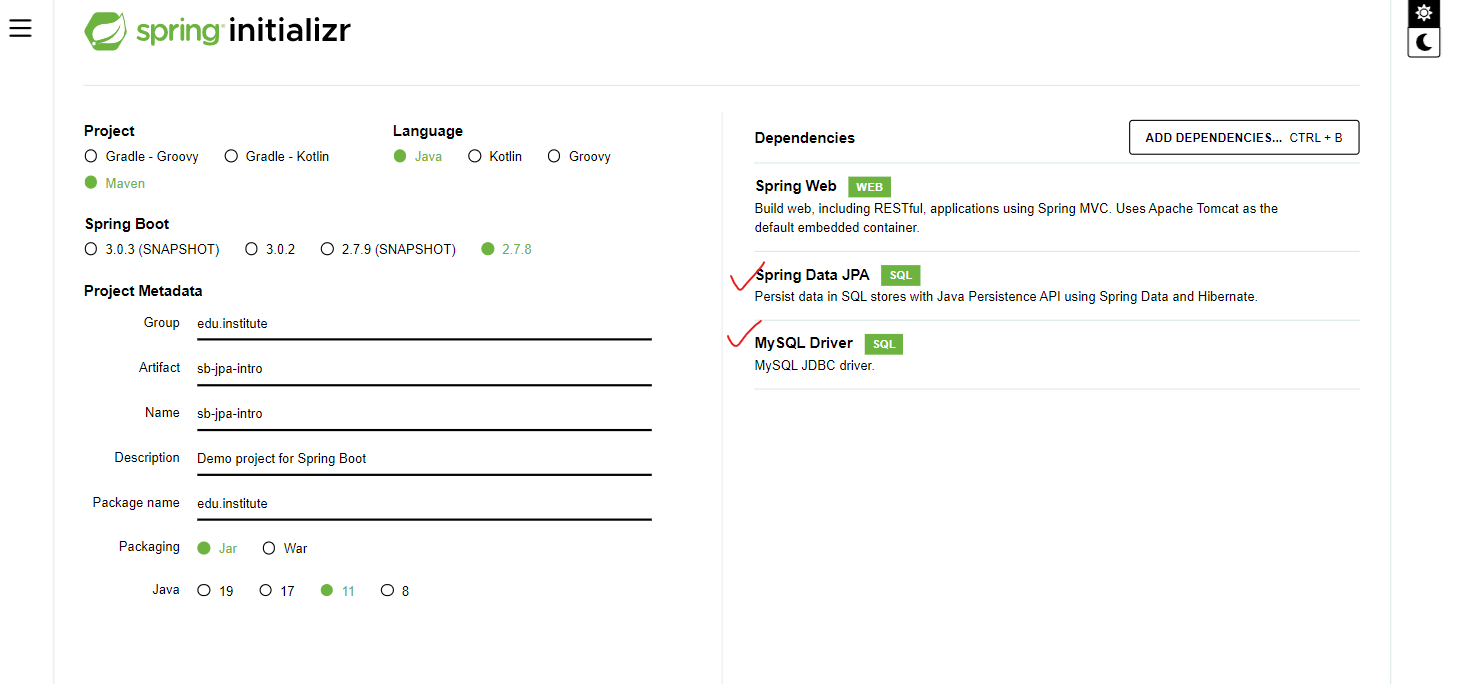
1. This module is use to connect the spring boot application with the database using a ORM tools.
2. ORM stands for Object Relational Mapping..
3. What is ORM
   1. In the ORM tool the java objects will be mapped with the table in the Database.
   2. It will create a table for every entity class in the project.
   3. It will also map the java class with the database table and the variables with the table column



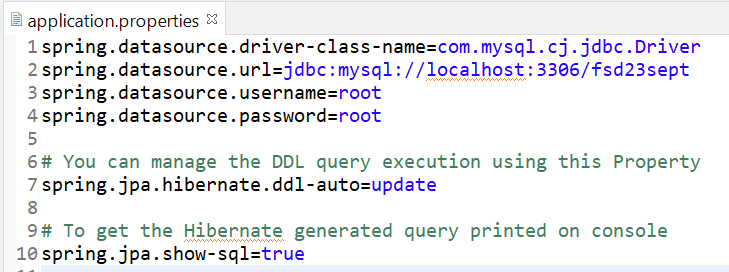
1. JPA is one of the ORM tool which is mostly used in the java framework.
2. JPA is a set of standards which is followed by all other ORM tool.
3. In Spring Boot Hibernate is use internally as a JPA implementation.
4. Advantages
   1. ORM Tool will take care about all the Database operations.
   2. It automate you database operations and steps internally.
   3. The Connection opening and closing will be manage by ORM tool.
   4. The database operation such as DDL and DML will be perform by ORM tool.
   5. All the checked SQL exception handle by ORM tool and converted into an unchecked exception.
   6. Using this you can perform Db operation with minimum coding efforts and less defect.

**Spring Boot JPA Application setup**

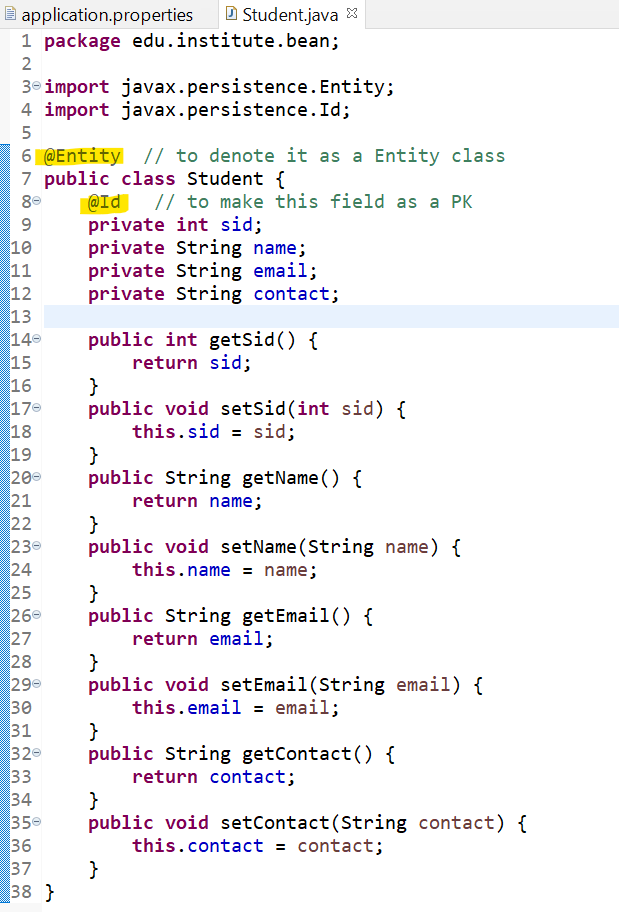
1. Create a Spring Boot application.
   1. Add Spring JPA dependency and Driver jar.



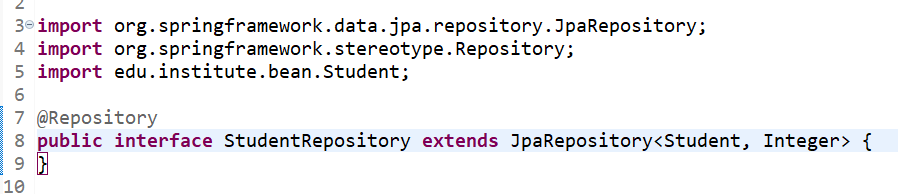
1. Import project into Eclipse
2. Adding Data Configurations
   1. Add Database configuration into application.properties file.
   2. This file is present inside src/main/resources folder.



1. Create an Entity class
   1. Entity class are the java encapsulated classes. Which has private variable and getter setter.
   2. You can convert the java class into Entity class using one annotation which is **@Entity** which has to be use on class.
   3. As per ORM rule entity class must have a primary key which is created using **@Id** annotation. This has to use on the variable which needs to mark as primary key.

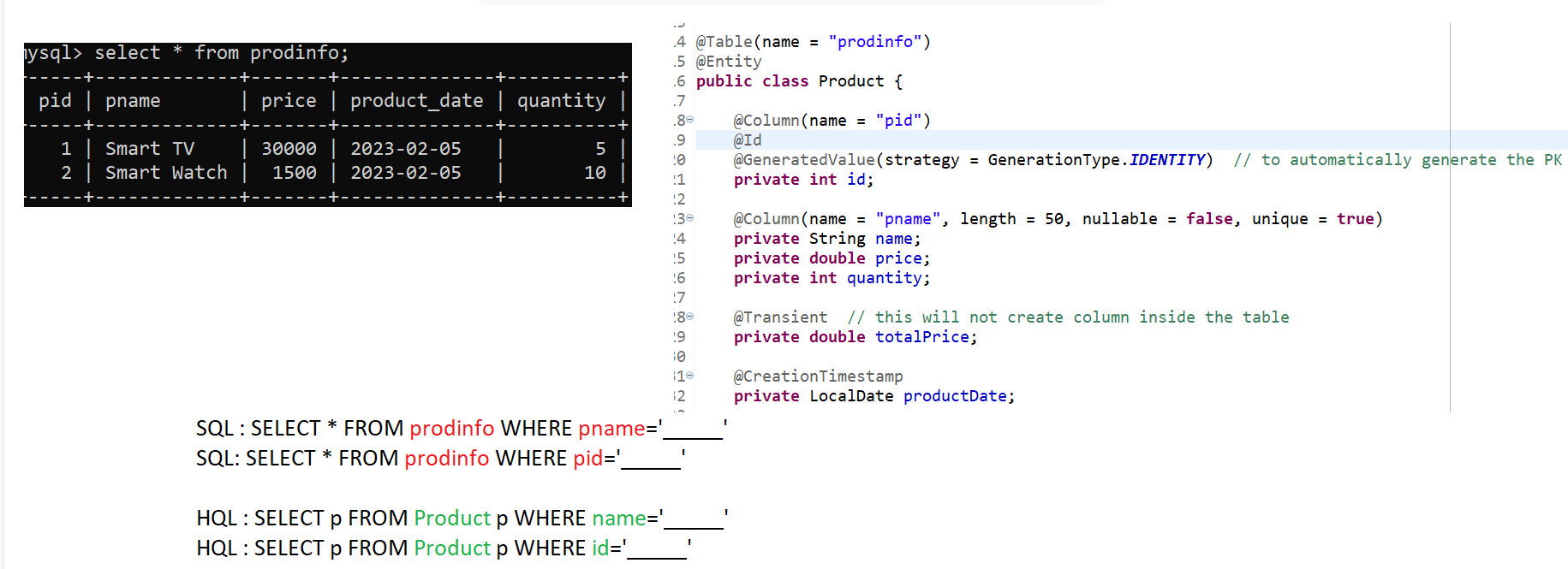


1. Use Spring Boot JPA apis to perform DB Operation. Use a JpaRepository interface
   1. JPA repository is predefine Spring boot JPA api
   2. Using this you can perform the DML and DQL operation.
   3. JPA repository provides the function which works on the Primary Key.
   4. It is an Interface which has to extends on custom interface.



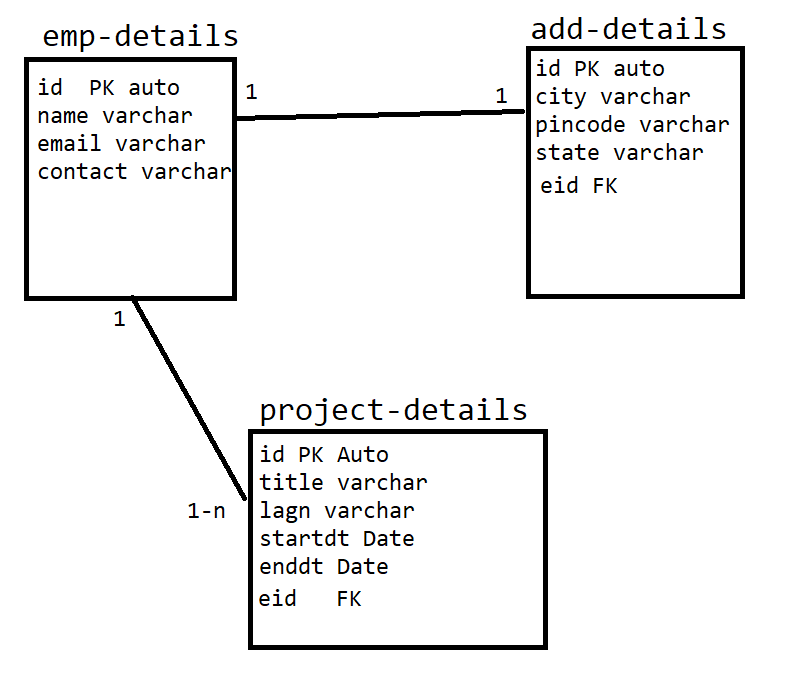
**Query Execution in Spring Boot JPA**

1. Using this you can execute the queried by your own.
2. The query execution can be achieved using **@Query annotation**.
3. You can also execute the DML type query for that you have to use **@Modifying and @Transactional annotation** also along with @Query
4. The queries has to write inside the Repository.
5. In this you can write a query by 2 ways
   1. SQL
      1. SQL is a structure query language, which is used to execute the query in the traditional way.
      2. In SQL you use a table name and column name.
      3. These queries are database dependent. If any changes happens inside table or column then it will affect the queries directly.
      4. SQL is also known as native query.
   2. HQL
      1. HQL is a Hibernate Query Language.
      2. It is exactly same as SQL the only different is instead of using Table and Column name you can use Class name and variable.
      3. These queries are database independent. The database changes will not affect on the query.



**Relational Mapping**

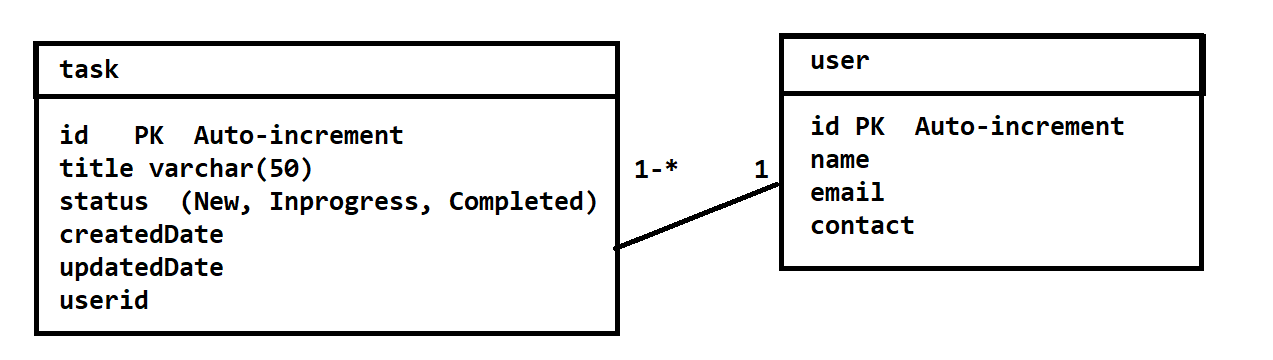
1. It Is use to perform the operation on Multiple table.
2. In the multiple tables there will be a primary and foreign key.
3. For this you needs to create a reference of one class into another class that is use a HAS-A relation in java classes.
4. This references are annotated by the following annotation
   1. @OneToOne
   2. @OneToMany
   3. @ManyToOne
   4. @ManyToMany



1. Create a foreign key, you have to create the reference of the class for which you wants to create a foreign key, inside the class where you wanted create a foreign key.

Task –

ToDoApplication



**Create Following REST APIs/Endpoints and Swagger REST API Document:**

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **Endpoint** | **Description** |
| **User Endpoint** | | |
| POST | /user/save | Save The user details |
| PUT | /user/update | Update user details |
| GET | /user/get-all | Get List of all available users |
| **Task Endpoint** | | |
| POST | /task/save | Add User Task |
| PUT | /task/update | Update Task Details |
| PUT | /task/update/{id}/{status} | Update Task status by ID Use HQL |
| GET | /task/task-by-id?id=<val> | Get Task by task ID |
| GET | /task/task-by-user?id=<val> | Get List of tasks by User Id USE HQL |
| GET | /task/task-by-status?id=<val>&status=<val> | Get list of tasks by status and user Id (All task for user by status) |
| DELETE | /task/delete-by-id/{id} | Delete Task by ID |