Workshop

▼ **Index for the Workshop**

1. **What is Problems with the Spring for programmers ?**

   ▼ **Ans**

   1. **If we want to create or develop an application using Spring framework, configuration is the important thing.**
   2. **In every Spring project, We will write the configuration again (boiler plate configuration)**
   3. **Spring is huge, It contains lot of modules like Spring core, Spring ORM, Spring Dao, Spring AOP, Spring MVC, Spring context which will lead to confusion in programmers mind what to use how to get know needs a lot of configuration.**
   4. **Sometimes programmers will not get know from where they have to start the configuration as spring needs a lot of configuration.**

   Note :- We can overcome these problems by using Spring Boot.

2. **What is Spring Boot ?**

   ▼ **Ans**

   - **Spring Boot is an extension of Spring Framework using which we an develop REST API's, as well as Enterprise Applications.**
   - **Using Spring Boot, We can remove all the boiler plate configuration.**
   - **Spring Boot will have all the future of Spring Core.**

3. **What is API (Application Programming Interface) ?**

   ▼ **Ans**

   - **API is an interface which can be used for Inter Application Communication.**
   - **API will have the description about the feature which are required to connect two application.**
   - **Using API, We can transfer the data from one application to another securely.**
   - **Authentication → Verification → The process of validating the user.**
   - **Authorization →**

4. **What REST or SOAP (Representational State Transfer) ?**

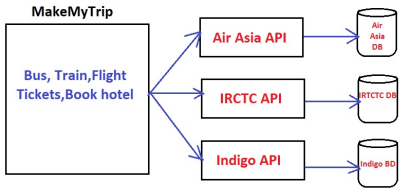
   ▼ **Ans**

   - **It is an Architecture which is used to the development of Web Service.**
   - **Using REST, We can transfer the data from one application to another using JSON, XML, etc.**

5. **What is REST API ?**

   ▼ **Ans**

   - **It is an API which is based on REST Architecture.**
   - **Using REST API, We can transfer the data from another application is JSON format or XML format.**
   - **REST API will have supports different HTTP request like POST, PUT, GET, DELETE to perform CRUD operation.**
   - **Using REST API, We can transfer the data from one application to another application safely with security.**
   - **Using Spring Boot, We can develop REST API as it contains embedded server and we can develop a web application.**



6. **What is JSON (JavaScript Object Notation) ?**

   ▼ **Ans**

   - **It is a data format which can be used to store the data or to transfer the data from one application to another application.**
   - **In JSON, the Object is represented by Curley Braces → { }**
   - **In JSON, the Array is represented by Array operator → [ ]**

7. **JSON Example-1 ?**

   ▼ **Ans**

   ▼ **Java 1**

```
public class Person
{
  private int id;
  private String name;
  private long phone;
  //Setters and Getters
    public static void main (String arg[])
    {
      Person p1 = new Person();
      p1.setInt(1);
      p1.setName("ABC");
      p1.setLong(888);
    }
}
```

   ▼ **JSON 1**

```
{
  "id" : 1,
  "name" : "ABC",
```

```
  "phone" : 888
}
```

8. **JSON Example-2 ?**

▼ **Ans**

▼ **Java 1**

```
public class Person
{
  private int id;
  private String name;
  private int age;
  private PanCard card;
  //Setters and Getters

}
```

▼ **Java 2**

```
public class PanCard
{
  private int id;
  private String number;
  private String state;
  private int PanCard;
  //Setters and Getters

}
```

▼ **JSON 1**

```
{
  "id" : 1,
  "name" : "ABC",
  "age" : 24,
  "card" : {
            "id" : 1,
            "num" : "KM12FGHG",
            "state" : "karnataka",
            "pincode" : 560004
            }
}
```

9. **JSON Example-3 ?**

▼ **Ans**

▼ **Java 1**

```
public class Hospital
{
  private int id;
  private String name;
  private int found;
  private List<Branch> branches;
  //Setters and Getters

}
```

▼ **Java 2**

```
public class Branch
{
  private int id;
```

```
  private String name;
  private long phone;
  //Setters and Getters

}
```

▼ JSON 1

```
{
  "id" : 1,
  "name" : "Apollo",
  "founder" : "ABC",
  "branches" : [{
                 "id" : 1,
                 "name" : "Apollo-A",
                 "phone" : 888
                },
                {
                 "id" : 2,
                 "name" : "Apollo-B",
                 "phone" : 888
                }]
}
```

10. **Creation of Spring Boot Project using Spring initializer ?**

▼ Ans

1. **Open Browser search for “Spring initializr”.**
2. **Open Spring initializr and Select the project as maven.**
3. **Select Java and provide the Group ID as well as Artifact ID.**
4. **Add the required dependencies**
   a. **Spring Dev Tools**
   b. **Spring web**
5. **Generate the project and extract the project from zip file to specific folder.**
6. **Import the extracted project to your work space.**
7. **If it is working, Do not touch if not Install JDK 17**

11. **What is @RestContoller ?**

▼ Ans

- **It is class level annotation which is used to make the class as RestController.**
- **Using @RestController, We can create REST API’s.**
- **@RestController is the combination of @Controller and @Responesbody.**
- **In @Controller is a method give some Results by default that result considered as view. if We do not want, we have to @Responsebody. then result considered as response.**

12. **Code - 1**

▼ Ans

▼ Example

```
package org.jsp.SpringbootDemo1.controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HomeController {
  @RequestMapping("/home")
```

```
    public String home() {
      return "Hi from HomeController";
    }
}
```

- **We cannot send POST,DELETE, PUT from Chrome. So we use Postman tool. with Postman, no need to create form for sending the data**
- **A method is nothing but a API.**

13. **What is Postman ?**

    ▼ **Ans**

    - **Postman is an API testing tool which is used to test the REST API's.**
    - **Using Postman, We can send POST, PUT, DELETE, GET request for an API.**

14. **What is @PathVariable ?**

    ▼ **Ans**

    - **It is an annotation present in org.springframework.wen.bind.annotation.**
    - **It can be used for a method parameter which is annotated with @RequestParam.**
    - **This annotation is used to indicate that a method parameter should be bounded with an URI (Uniform Resource Information) template variable.**
    - **parameter are nothing but local variables and local variable doesn't have any default value.**

15. **What is Query String and Query Parameter ?**

    ▼ **Ans**

    - **The data which has been send on the server along with the URL in the form of Key and value pair.**

      Example: http://Localhost:8080/user/x=10&y=20 → these are Query String.
    - **Query Parameter is nothing but key and value pair and join by ampersand "&".**
    - **In REST API query parameter are used to control what data is returned to the client usually. The Query parameter is a simple storing or an Array.**

16. **What is difference the between @RequestParam and @PathVariable ?**

    ▼ **Ans**

@PathVariable	@ResquestParam
1. It is used to indicate that a method parameter should bound with a URL template variable.	**1. It is used to indicate that a method parameter should bound with a web request parameter.**

17. **What is @RequestParam ?**

    ▼ **Ans**

    - **It is an annotation present org.springframework.web.bind.annotation.**
    - **It can be used for the annotated handler methods. (handler method means the methods are annotated with) which are present inside Request Handling classes (controllers).**
    - **This annotation is used to indicate that the method parameter id bound to web request parameter (Query parameters)**
    - **It can be used for annotated methods resent in controller (Request Handling Class)**

18. **What is @RestController ?**
19. **What is REST ?**

20. **What is API ?**
21. **What is REST API ?**
22. **What is Spring Boot ?**
23. **What is Query String and Query Parameter?**
24. **@RequestBody ?**

    ▼ **Ans**

    - **This annotation can be used for parameter.**
    - **This annotation is used to indicate that a method parameter should be bound to the web request body.**
    - **If we use this annotation in a JSON body will directly converted into Java Object.**

25. **Example for @RequestBody ?**

    ▼ **Ans**

```
package org.jsp.SpringbootDemo1.controller;

import org.jsp.SpringbootDemo1.dto.User;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HomeController {
  @PostMapping("/home")
  public String home() {
    return "Hi from HomeController";
  }

  @PutMapping("/sum")
  public String sum(@RequestParam int a, int b) {
    return "Sum is " + (a + b);
  }

  @PutMapping("/diff")
  public String diff(@RequestParam int a, int b) {
    return "Sum is " + (a - b);
  }

  @PutMapping("/pro")
  public String pro(@RequestParam int a, int b) {
    return "Sum is " + (a * b);
  }

  @PutMapping("/div")
  public String div(@RequestParam int a, int b) {
    return "Sum is " + (a / b);
  }

  @GetMapping("/largest/{a}/{b}")
  public String largest(@PathVariable(name = "a") int n1, @PathVariable(name = "b") int n2) {
    return (n1 > n2 ? n1 : n2) + " is the largest number";
  }

  @PostMapping("/print")
  public String printDetails(@RequestBody User user) {
    return user.toString();
  }

  @GetMapping("/get")
  public User getUser() {
    return new User(1, "ABC", 999, "a123");
  }
}
```

26. **What is End-Points ?**

    ▼ **Ans**

    - **End-points is noting but URL.**

27. **Designing of End-Points ?**

    ▼ **Ans**

    - **Let us consider, We have an Entity User**
    - **For this Entity, We should have methods to save, update, delete and fetch. and these method should bound with web requests, to bound these with requests, We need to End-Points(URL)**
    - **Following are the End-Points to perform CRUD operations on User Entity.**

      ▼ **User.java**

```
@Entity
public class User
{
  @Id
  @GeneratedValue(stratgy=GeneratedId.IDENTIY)
  private int id;
  private String name;
  private long phone;
  //getter's and setter's
}
```

Task	Request Mapping	User / End - Points
save	POST	/user
update	PUT	/user
delete	DELETE	/user/{id}
fetch	GET	/user/{id}
fetch all	GET	/user/

28. **What is Spring data JPA ?**

    ▼ **Ans**

    - **It is integration of Spring with JPA.**
    - **JPARepository → Interface present in Spring data JPA. (We have to create our Repository)**
    - **interface UserRepository extends JpaResponse <Entity , Primary Key>**

29. **JPA Repository<T , ID> ?**

    ▼ **Ans**

    - **It is an Interface, By extending this Interface, We can perform CRUD Operations on an Entity.**
    - **To save and update, We need to use save(T entity) method and return type of this is same as the type of Entity.**
    - **We can fetch the record by using findById(Object , PK) method the return type of this method is java.util.Optional<T>**
    - **We can delete the record by using either delete(Object entity) or deleteById(object , PK) method.**
    - **We can fetch all the records by using findAll() method the return type of this method is List<T>.**
    - **get() → Present in Optional class and it throws an exception called NoSuchElementEXception**

30. **What is the difference @Controller and @RestController ?**

    ▼ **Ans**

- **@Controller map the model object to view or template and make it human readable but @RestController simply returns the object and object data is directly written in HTTP response as JSON or XML.**

31. **What is the draw back of Response Structure ?**

    ▼ **Ans**

    - **The draw back is Postman Status and ResponseBody Status will be different. For this Tester gets confused.**
    - **To overcome we will use ResponseEntity. ResonseEntity has constructor and it accepts 2 parameters Body T, HttpStatus.**

32. **Exception Handling in Spring Boot Project ?**

    ▼ **Ans**

    - **@ControllerAdvice or @RestControllerAdvice ResponseEntityExceptionHandler. [REEH]**
    - **It is used to handle the exception in service layer by avoiding the transfer to the controller layer to handle.**
    - **It is not recommended to handle the exception in controller.**
    - **It is used to deviate the exception from transfer into the controller.**
    - **ExceptionHandler class extends REEH and this class annotated with @ControllerAdvice. and We have to create methods and this method should be annotated with @ExceptionHandling (IdnotFoundException.java).**
    - **Response return @ControllerAdvice**
        - **@EnableJpaRepository → We can use this when your Repository**
        - **@EntityScan → To scan your Entity**

---

28. **Steps to perform CRUD Operation using Spring Boot with the help of Spring-data JPA ?**

    ▼ **Ans**

    1. **Open the Browser and search for Spring initializer.**
    2. **Click on Spring Initializer select project as Maven and language as Java.**
    3. **Add the following dependencies.**
        a. **Spring Boot Developer Tools**
        b. **Spring Web**
        c. **Spring Data JPA**
        d. **MySQL Driver**
    4. **Provide the group ID as well as Artifact ID and click on generate.**
    5. **Extract the project into a specific folder and import that into work space.**
    6. **Open application.properties configure User-Name, Password, URL, Dialect etc. (for code we will get it in Satish Sir Git-Hub. file name(application.properties) or link→ "**https://github.com/sathishnyadav/supporting-files/blob/master/application.properties**")**
    7. **We have to create Repository for User Entity class. how to create By extending JpaRepository (User. integer)**
    8. **Interface User extends Jparepository<User,Integer>. That interface User should be Annotated with Autowired.**

    ▼ **src/main/java**

    ▼ **org.jsp.User**

    ▼ **UserApplication.java**

```
package org.jsp.User;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class UserApplication {

  public static void main(String[] args) {
    SpringApplication.run(UserApplication.class, args);
  }

}
```

▼ **org.jsp.User.controller**

▼ **UserController.java**

```
package org.jsp.User.controller;

import java.util.List;

import org.jsp.User.dto.ResponseStructure;
import org.jsp.User.dto.User;
import org.jsp.User.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {
  @Autowired
  private UserService service;

  @PostMapping("/user")
  public ResponseStructure<User> saveUser(@RequestBody User user) {
    return service.saveUser(user);
  }

  @PutMapping("/user")
  public ResponseStructure<User> updateUser(@RequestBody User user) {
    return service.updateUser(user);
  }

  @GetMapping("/user/{id}")
  public ResponseStructure<User> findUserById(@PathVariable int id) {
    return service.findUserById(id);
  }

  @GetMapping("/user")
  public ResponseStructure<List<User>> findAll() {
    return service.findAll();

  }

  @DeleteMapping("/user/{id}")
  public ResponseStructure<String> deleteById(@PathVariable int id) {
    // Optional<User> u=service.findById(id);
    // if(u.isPresent())
    // {
    // service.delete(u.get());
    // return "user deleted";
    // }
    // else
    // {
    // return "invalid id";
    // }
    return service.delete(id);
  }

}
```

▼ **org.jsp.User.dao**

▼ **UserDao.java**

```
package org.jsp.User.dao;

import java.util.List;
import java.util.Optional;

import org.jsp.User.dto.User;
import org.jsp.User.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class UserDao
{
  @Autowired
  private UserRepository repository;
  public User saveUser(User user)
  {
    return repository.save(user);
  }
  public User updateUser(User user)
  {
    return repository.save(user);
  }
  public Optional<User> findUserById(int id)
  {
    return repository.findById(id);
  }

  public void delete(User user)
  {
    repository.delete(user);
  }
  public List<User> findAll()
  {
    return repository.findAll();
  }
}
```

▼ **org.jsp.User.dto**

▼ **User.java**

```
package org.jsp.User.dto;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class User
{
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;
  private String name;
  private long phno;
  private String password;
  public String getPassword() {
    return password;
  }
  public void setPassword(String password) {
    this.password = password;
  }
  public int getId() {
    return id;
  }
  public void setId(int id) {
    this.id = id;
  }
  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }
```

```
  public long getPhno() {
    return phno;
  }
  public void setPhno(long phno) {
    this.phno = phno;
  }
}
```

▼ **ResponseStructure.java**

```
package org.jsp.User.dto;

public class ResponseStructure<T> {
  private String message;
  private T body;
  private int code;

  public String getMessage() {
    return message;
  }

  public void setMessage(String message) {
    this.message = message;
  }

  public T getBody() {
    return body;
  }

  public void setBody(T body) {
    this.body = body;
  }

  public int getCode() {
    return code;
  }

  public void setCode(int code) {
    this.code = code;
  }
}
```

▼ **org.jsp.User.repository**

▼ **UserRepository.java**

```
package org.jsp.User.repository;

import org.jsp.User.dto.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository  extends JpaRepository<User,Integer>{

}
```

▼ **org.jsp.User.service**

▼ **UserService.java**

```
package org.jsp.User.service;

import java.util.List;
import java.util.Optional;
import org.jsp.User.dao.UserDao;
import org.jsp.User.dto.ResponseStructure;
import org.jsp.User.dto.User;
import org.jsp.User.exception.IdNotFoundException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Service;
import org.springframework.http.ResponseEntity;
```

```
@Service
public class UserService {
  @Autowired
  private UserDao dao;

//  public User saveUser(User user)
//  {
//    return dao.saveUser(user);
//  }
  public ResponseEntity<ResponseStructure<User>> saveUser(User user) {
    ResponseStructure<User> structure = new ResponseStructure<>(); // Response structure is used for telling the
                                    // message
    structure.setBody(dao.saveUser(user));
    structure.setMessage("user registered with ID:" + user.getId());
    structure.setCode(HttpStatus.ACCEPTED.value());
    return new ResponseEntity<ResponseStructure<User>>(structure, HttpStatus.ACCEPTED);

  }

//  public User updateUser(User user)
//  {
//    return dao.updateUser(user);
//  }
  public ResponseEntity<ResponseStructure<User>> updateUser(User user) {
    ResponseStructure<User> structure = new ResponseStructure<>(); // Response structure is used for telling the
                                    // message
    structure.setBody(dao.updateUser(user));
    structure.setMessage("user updated succesfully");
    structure.setCode(HttpStatus.ACCEPTED.value());
    return new ResponseEntity<ResponseStructure<User>>(structure, HttpStatus.ACCEPTED);
  }

  public ResponseEntity<ResponseStructure<User>> findUserById(int id) {
    ResponseStructure<User> structure = new ResponseStructure<>();
    Optional<User> u = dao.findUserById(id);
    if (u.isPresent()) {
      structure.setBody(u.get());
      structure.setMessage("user Found");
      structure.setCode(HttpStatus.FOUND.value());
      return new ResponseEntity<ResponseStructure<User>>(structure, HttpStatus.FOUND);

    } else {
      /*
       * structure.setBody(u.get()); structure.setMessage("user NOT Found");
       * structure.setCode(HttpStatus.NOT_FOUND.value()); return new
       * ResponseEntity<ResponseStructure<User>>(structure, HttpStatus.NOT_FOUND);
       */
      throw new IdNotFoundException();
    }
  }

  public ResponseEntity<ResponseStructure<List<User>>> findAll() {
    ResponseStructure<List<User>> structure = new ResponseStructure<List<User>>();
    structure.setBody(dao.findAll());
    structure.setMessage("List Of All User");
    structure.setCode(HttpStatus.OK.value());
    return new ResponseEntity<ResponseStructure<List<User>>>(structure, HttpStatus.OK);
  }
//  public String delete( int id)
//  {
//    Optional<User> u=dao.findUserById(id);
//    if(u.isPresent())
//    {
//      dao.delete(u.get());
//      return "user deleted";
//    }
//    else {
//      return "invalid data";
//    }
//  }
//  public List<User> findAll()
//  {
//    return dao.findAll();
//  }

  public ResponseEntity<ResponseStructure<String>> delete(int id) {
    Optional<User> u = dao.findUserById(id);
    ResponseStructure<String> structure = new ResponseStructure<>(); // Response structure is used for telling th
                                    // message
    if (u.isPresent()) {
```

```
      structure.setBody("User Deleted");
      structure.setMessage("User found");
      structure.setCode(HttpStatus.OK.value());
      dao.delete(u.get());
      return new ResponseEntity<ResponseStructure<String>>(structure, HttpStatus.OK);
    } else {
//      structure.setBody("User Not found");
//      structure.setMessage("INvaild IDS");
//      structure.setCode(HttpStatus.NOT_FOUND.value());
//      return new ResponseEntity<ResponseStructure<String>>(structure,HttpStatus.NOT_FOUND);
      throw new IdNotFoundException();
    }
  }
}
```

▼ **org.jsp.User.exception**

▼ **IdNotFoundException.java**

```
package org.jsp.User.exception;

public class IdNotFoundException extends RuntimeException {
  @Override
  public String getMessage() {
    return "User Id is not Found";
  }
}
```

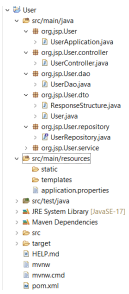
▼ **UserAppExceptionHandler.java**

```
package org.jsp.User.exception;

import org.jsp.User.dto.ResponseStructure;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@RestControllerAdvice
public class UserAppExceptionHandler extends ResponseEntityExceptionHandler {
  @ExceptionHandler(IdNotFoundException.class)
  public ResponseEntity<ResponseStructure<String>> handleIdNotFoundhandle(IdNotFoundException e) {
    ResponseStructure<String> structure = new ResponseStructure<>();
    structure.setCode(HttpStatus.NOT_FOUND.value());
    structure.setBody("User not found");
    structure.setMessage(e.getMessage());
    return new ResponseEntity<ResponseStructure<String>>(structure, HttpStatus.NOT_FOUND);
  }
}
```

▼ **src/test/java**

▼ **application.properties**

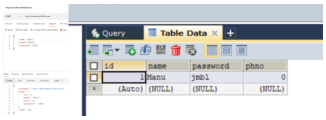
```
spring.datasource.url= jdbc:mysql://localhost:3306/User-SpringBoot?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=admin

spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect

#spring.mvc.pathmatch.matching-strategy=ant-path-matcher
```



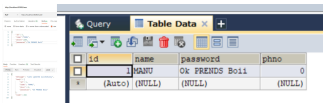
29. **Testing above CRUD Operation Project Using Postman ?**

▼ **Ans**

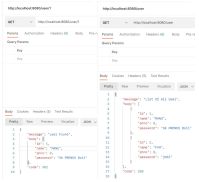
1. **POST**



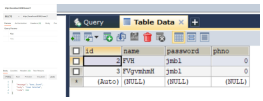
2. **PUT**



3. **GET**



4. **DELETE**



30. **Assignment Create UserProduct Project ?**

▼ **Ans**

---

31. **What is Lombok ?**

▼ **Ans**

- **Lombok is a java library using which we can reduce the boiler plate code which exist in java application.**
- **Generally, We have to create getter's , setter's , toString(), equals(), hashcode(), and NoArguments as well as parameterized constructor , noarguments just by using simple annotation. Lombok improves the readability of source code (maintaining source code is easy)**

Following are the important Annotation present in Lombok

1. **@Setter → It is used to generate the setter for the fields.**
2. **@Getter → It is used to generate the getter for the fields.**
3. **@toString → It is used to Override the toString method in the annotated class. We can also exchange the fields by using exclude attribute.**
4. **@equalsAndHashcode → It is used to override equals and hashCode method in the annotated class.**
5. **@NoArgsConstructor → It is used to create NoArgsContsructor or the annotated.**
6. **@AllargConstructor → It is used to create by using all the fields.**
7. **@Builder → It can be used for Type, method, constructor. this annotation will static method called builder() for the annotation class using which we can initialize in single line (method chaining).**
8. **@Data → It is a combination of the following annotation.**
    a. **@Getter**
    b. **@Setter**
    c. **@toString**
    d. **@EqualsAndHashcode**

Note :- Pre-Requisites to use Lombok

1. **IDE should have project Lombok 1.18.26 version**
2. **The project should have lombok jar. We can get that from Maven Repository.**

▼ **Example**

▼ **pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.jsp</groupId>
  <artifactId>lombok_Demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
      <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.26</version>
        <scope>provided</scope>
      </dependency>
  </dependencies>
</project>
```

▼ **Student.java**

```
package org.jsp;

import lombok.Data;

@Data
public class Student {
  private int id;
  private String name;
  private long ph;
  private double perc;
}
```

▼ **Test.java**

```
package org.jsp;
```

```
public class Test {
  public static void main(String[] args) {
    Student s1 = new Student();
    s1.setId(1);
    s1.setName("name");
    s1.setPh(133546l);
    s1.setPerc(12.12);
    System.out.println(s1);
  }
}
```

32. **What is GIT ?**

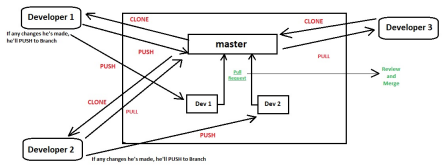
▼ **Ans**

- **GIT stands for Global Information Tracker.**
- **GIT is a open source distributed version control System software used for Source Code Management. The main advantage is to manage our source code.**
- **Multiple people can contribute for one application with the help of GIT.**

33. **What is GIT-HUB ?**

▼ **Ans**

- **It is a service which is used to host GIT Repositories.**
- **Local Repository → which is particular to one system.**
- **Remote Repository → which can accessed by anywhere.**
- **Local Repository —> Remote Repository —> accessed by anywhere.**
- **Repository means It is place where we have source code.**



34. **GIT Commands ?**

▼ **Ans**

1. **git config - -global user.name" " → It is used to configure the User Name.**
2. **git config - -global user..email" " → It is used to configure the Email.**
3. **git init → It is used to initialize an empty GIT Repository.**
4. **git status → It is used to check the status of untracked files.**
5. **git add . → It is used to staged all the files for the Local Repository.**
6. **git add FileName → It is used to add or staged a specific files to the Local Repository.**
7. **git remote add origin "URL" → It is used to add a remote origin.**
8. **git remote -v → It is used to check whether the Remote Repository present or not, and It is used to list all the remote origins.**
9. **git commit -m "message" → It is used to commit the changes when we staged (add).**
10. **git push origin branchname → It is used to push the changes from Local Repository to Remote Repository.**
11. **git clone "URL" → It is used to clone a Remote Repository.**
12. **git branch branchName → It is used to create a new branch.**

13. **git checkout branchName → It is used to switch the header or controller to specified branch.**

35. **Playing PUSH, PULL or CLONE through GIT and GIT-HUB with 2 Developers ? (Practice)**

▼ **Ans**

- **Create 2 Folder, Dev1 and Dev2 in Desktop Window.**
- **Write any Program in Dev1 Folder.**
- **Create a new Repository in GIT-HUB.**

1. **Pushing Dev1 code to GIT-HUB.**
   a. **Go to Dev1 Folder and Open git-bush there**
   b. **Pass command as git init**
   c. **Pass command as git status**
   d. **Pass command as git add Test.java**
   e. **Go to GIT-HUB and Copy Repository' URL**
   f. **Pass command as git remote add origin "URL"**
   g. **Pass command as git remote -v**
   h. **Pass command as git commit -v**
   i. **Pass command as git push origin master**
2. **Pulling or cloning the code from GIT-HUB to Dev2 Folder**
   a. **Go to Dev2 Folder and Open git-bush there**
   b. **Pass command as git init**
   c. **Pass command as git clone "Repository URL" (Now Dev1 code will come to Dev2 Folder, So he can modify)**
3. **Dev2 modified Pulled Code and then he's Pushing into branch**
   a. **Go to Dev2 Folder and Open git-bush there**
   b. **Pass command as cd GIT-HUB RepositoryName**
   c. **Pass command as git branch Dev2**
   d. **Pass command as git checkout Dev2**
   e. **Pass command as git commit -m "message"**
   f. **Pass command as git add Test.java**
   g. **Pass command as git commit -m "message"**
   h. **Pass command as git push origin Dev2 (In GIT-HUB → master → We will get Dev2)**
4. **Pushing Dev2 branch to master In GIT-HUB**
   a. **In Git-Hub, Go to Respective Repository and click Pull Request → new pull request**
   b. **Change the compare : Dev2**
   c. **Click create pull request type some message and click create pull request**
   d. **Click merge pull request → Confirm merge**
5. **How to Collaborate my Team-Mates in GIT-HUB**
   a. **In GIT-HUB, Open your respective Repository and Go to Settings**
   b. **In Setting, We will have option called Collaborators, click and add your Team-mates Email ID.**