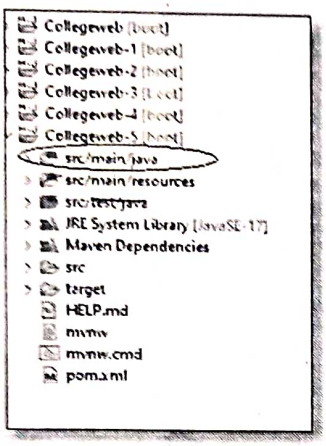
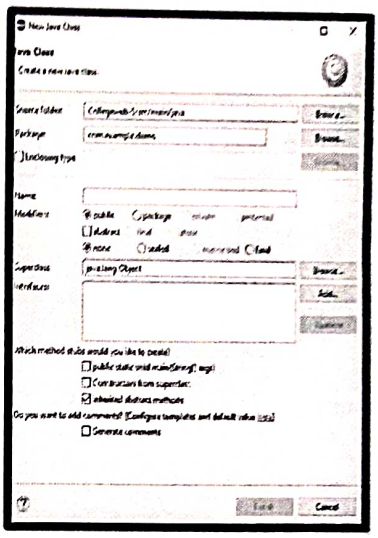
## Assi-1 Bootstrapping simple spring-boot Project

1. create a new Spring Boot project. Include the following dependencies:
   File---->new---------- □spring boot starter project
   Add dependencies as Spring Web




2) In project explorer select

Select **src/main/java** as shown in above figure Right click and select new ,
add class



**HelloController**

```
package com.example.simplespringboot;

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

@RestController
public class HelloController
{
   @RequestMapping("/getString")

   private String getString()
   {
     return "This is the first Spring Boot Practical";
   }
}
```

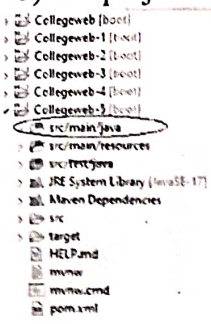
OUTPUT:-

This is the first Spring Boot Practical

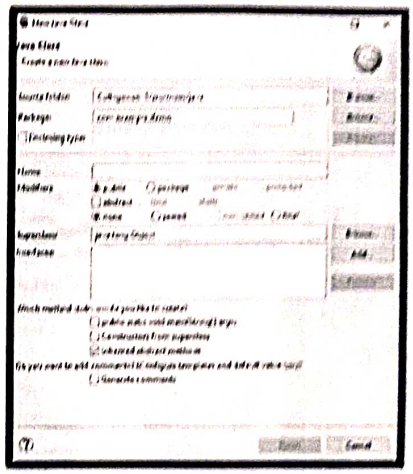
**Practical 2: Practical for Restful spring boot application student.**

1. create a new Spring Boot project. Include the following dependencies:
   File---->new ----------□spring boot starter project
   Add dependencies as Spring Web
   Spring Data JPA H2
   Database

3) In project explorer select



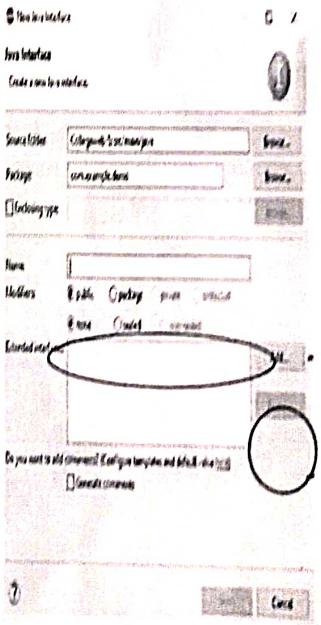

Select **src/main/java** as shown in above figure



Right click and select new , add class Enter name of class as **Student**

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;
    private String firstName; private String
    lastName; private int age;

    // Getters and setters
    //To string Method()
}
```

Now select source menu of eclipse and select generate getters and setters .....
Add getter and setters for all fields
Again select source menu of eclipse and select generate to string function .....



4) **create student repository interface Select** src/main/java
Right click and select new Select interface
Type Name as **StudentRepository.**
Click on add button and select interface as **JpaRepository**
Write following code in this java file

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<Student, Long> {
}
```

5) Create a Student Service:
Create a service class to handle business logic.
Add Class with name **StudentService** and add following code

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List; import
java.util.Optional;

@Service
public class StudentService {
    @Autowired
    private StudentRepository studentRepository;

    public List<Student> getAllStudents() { return
        studentRepository.findAll();
    }
    public Optional<Student> getStudentById(Long id) { return
        studentRepository.findById(id);
    }
    public Student saveStudent(Student student) { return
        studentRepository.save(student);
    }
    public void deleteStudent(Long id) {
        studentRepository.deleteById(id);
    }

}
```

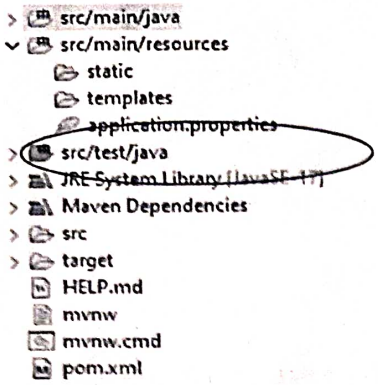
6) Create a **Student Controller**:

Create a REST controller to handle HTTP requests.

Add class with name **StudentController**

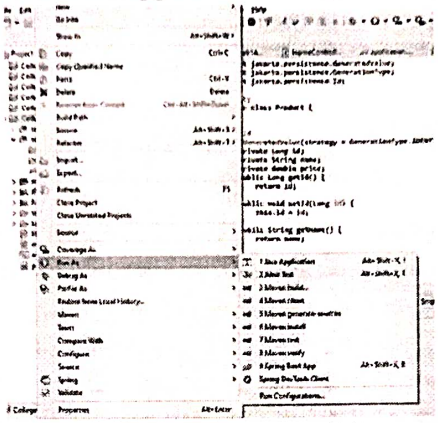
```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List; import
java.util.Optional;

@RestController @RequestMapping("/api/students") public class
StudentController {
    @Autowired
    private StudentService studentService;
    @GetMapping
    public List<Student> getAllStudents() {
    return studentService.getAllStudents();
    }
    @GetMapping("/{id}")
    public Optional<Student> getStudentById(@PathVariable Long id) {
        return studentService.getStudentById(id);
    }
    @PostMapping
    public Student saveStudent(@RequestBody Student student) { return
        studentService.saveStudent(student);
    }
    @DeleteMapping("/{id}")
    public void deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id);
    }
}
```

7) In Application.properties write the code

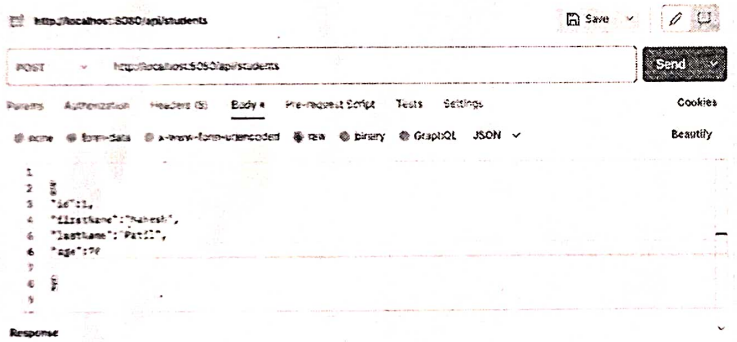



```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
```

8) Run the application using run as spring boot application



9) Now run postman application and set properties according to following figure




Select http type as "post"
Enter url as localhost:8080/api/students
Select body and enter information as
{ "id":1,
"firstName":"RC", "lastName":"Patel",
"age":70
}
Press send button
Create similar records and send
For display use http type as :- get
Enter url as localhost:8080/api/students You will get all the records in postman
You can also type the same url on browser you will get all the records
Similarly you can delete record of student by using Type as delete
And url as localhost:8080/api/students/1

Assi-3 **Implement a spring boot project to create student entity using CRUD Operation with exception handling.**

**1. Create Student Entity:**

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class Student {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY) private
   Long id;
   private   String   firstName;
   private   String   lastName;
   private int age;
   // Getters and setters
   // Add to string Method
}
```

**2. Create Student Repository:**

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<Student, Long> {
}
```

**3. Create Student Service with CRUD Operations and Exception Handling:**

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
@Service
public class StudentService {
   @Autowired
   private  StudentRepository  studentRepository;
   public List<Student> getAllStudents() { return
      studentRepository.findAll();
   }
   public Optional<Student> getStudentById(Long id) { return
      studentRepository.findById(id);
   }
   public Student saveStudent(Student student) { return
      studentRepository.save(student);
   }


   public Student updateStudent(Long id, Student updatedStudent) {
   if (studentRepository.existsById(id)) {
         updatedStudent.setId(id);
         return studentRepository.save(updatedStudent);

      }
      else {
         throw new StudentNotFoundException("Student not found with id: " + id);
      }
```

```
    }

    public void deleteStudent(Long id) {
        if (studentRepository.existsById(id)) { studentRepository.deleteById(id);
        }
        else {
            throw new StudentNotFoundException("Student not found with id: " + id);
        }
    }
}
```

**4. Create Custom Exception Class:**

```
public class StudentNotFoundException extends RuntimeException {

    public StudentNotFoundException(String message) { super(message);
    }
}
```

**5. Create Student Controller:**

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List; import
java.util.Optional;
@RestController
@RequestMapping("/api/students")
public class StudentController {
    @Autowired
    private StudentService studentService;
    @GetMapping
    public List<Student> getAllStudents() { return
        studentService.getAllStudents();
    }
    @GetMapping("/{id}")
    public Student getStudentById(@PathVariable Long id) {
    return studentService.getStudentById(id)
.orElseThrow(() -> new StudentNotFoundException("Student not found with id: " + id));
    }
    @PostMapping
    public Student saveStudent(@RequestBody Student student) { return
        studentService.saveStudent(student);
    }
    @PutMapping("/{id}")
    public Student updateStudent(@PathVariable Long id, @RequestBody Student updatedStudent)
{
        return studentService.updateStudent(id, updatedStudent);
    }
    @DeleteMapping("/{id}")
    public void deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id);
    }
}
```

**6. Run the Application:**

Run the Spring Boot application, and it should start a server on **http://localhost:8080**by default.

**Assi-4 Develop the micro service for employee management implements endpoint for retrieve employee details. Utilize spring boot and spring JPA to store and retrieve employee data.**

**1. Create a new Spring Boot Project:** Use Spring Initializer to create a new project with the required dependencies.

**Dependencies:**

- Spring Web
- Spring Data JPA
- H2 Database (or any other database of your choice)

2. **Define the Employee Entity:** Create an Employee class to represent the data model.

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
@Entity
public class Employee {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
private String firstName;
private String lastName;
private String position;
// Getters and setters
//Generate To String Method
}
```

3. **Create an Employee Repository:** Create a repository interface which Extends JPARepository that for basic CRUD operations.

```
import org.springframework.data.jpa.repository.JpaRepository;
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

4. **Create an Employee Service:** Create a service class to handle business logic.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List; import java.util.Optional;
@Service
public class EmployeeService {
@Autowired
private EmployeeRepository employeeRepository;

public List<Employee> getAllEmployees() {
 return employeeRepository.findAll();
}
public Optional<Employee> getEmployeeById(Long id) {
return employeeRepository.findById(id);
}
public Employee saveEmployee(Employee employee) {
 return employeeRepository.save(employee);
}
public void deleteEmployee(Long id) {
employeeRepository.deleteById(id);
}
}
```

5. **Create an Employee Controller:** Create a REST controller to handle HTTP requests.

```
import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.web.bind.annotation.*;
import java.util.List; import
java.util.Optional;
@RestController
@RequestMapping("/api/employees")
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;
    @GetMapping
    public List<Employee> getAllEmployees() {
    return employeeService.getAllEmployees();
    }
    @GetMapping("/{id}")
    public Optional<Employee> getEmployeeById(@PathVariable Long id) {
        return employeeService.getEmployeeById(id);
    }
    @PostMapping
    public Employee saveEmployee(@RequestBody Employee employee)
{
        return employeeService.saveEmployee(employee);
    }
    @DeleteMapping("/{id}")
    public void deleteEmployee(@PathVariable Long id) {
    employeeService.deleteEmployee(id);
    }

}
```

**6. Configure Application Properties:** Ensure your **application.properties** or **application.yml** contains the necessary configuration for the H2 database and other settings.

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver spring.datasource.username=sa
spring.datasource.password=password
```

**7. Run the Application:** Run your Spring Boot application, and it should start a server on http://localhost:8080 by default.

Now you can use tools like Postman or curl to test your RESTful API for employee management. The endpoints include:

- GET /api/employees: Get all employees
- GET /api/employees/{id}: Get a specific employee by ID
- POST /api/employees: Create a new employee
- DELETE /api/employees/{id}: Delete an employee by ID