Task 1: Create setters and getters for properties, and create the bean definition file

Step 1: Create a Java Web Application using Eclipse IDE

1. Open Eclipse IDE.
2. Create a new dynamic web project:
    - `File` > `New` > `Dynamic Web Project`
    - Name the project `SpringApp`.

Step 2: Create Necessary Packages and Configure Build Path

1. Create packages for your classes, for example:
    - `com.example.model`
    - `com.example.controller`
    - `com.example.service`
2. Add the Spring library to your project:
    - Right-click on the project > `Build Path` > `Configure Build Path`.
    - Add Spring JARs or use Maven/Gradle to manage dependencies.

Step 3: Compose Code for Creating Classes and Interfaces

```
// Employee.java
package com.example.model;

public class Employee {
    private String name;
    private int id;
    private double salary;

    // Setters and Getters
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
```

```
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

Step 4: Deploy the Project in Server

1. Right-click on the project > `Run As` > `Run on Server`.
2. Select a server (e.g., Tomcat) to deploy the project.

Step 5: Display the Results

- You can create a `controller` class to display results using Spring's MVC or REST APIs.

```
// EmployeeController.java
package com.example.controller;

import com.example.model.Employee;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeController {

    @GetMapping("/employee")
    public Employee getEmployee() {
        Employee employee = new Employee();
        employee.setName("John Doe");
        employee.setId(1);
```

```
        employee.setSalary(50000);
        return employee;
    }
}


<!-- beans.xml (Spring Configuration) -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/sp

    <!-- Define Bean for Employee -->
    <bean id="employee" class="com.example.model.Employee">
        <property name="name" value="John Doe"/>
        <property name="id" value="1"/>
        <property name="salary" value="50000"/>
    </bean>
</beans>
```

Task 2: Write a Spring Application Program to Print Employee Details with All of His/Her Addresses Using Collection

```
// Address.java
package com.example.model;

public class Address {
    private String street;
    private String city;
    private String state;
    private String zipCode;

    // Setters and Getters
    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
```

```
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getZipCode() {
        return zipCode;
    }

    public void setZipCode(String zipCode) {
        this.zipCode = zipCode;
    }
}

// Employee.java (updated)
package com.example.model;

import java.util.List;

public class Employee {
    private String name;
    private int id;
    private double salary;
    private List<Address> addresses;

    // Setters and Getters
    public List<Address> getAddresses() {
        return addresses;
    }

    public void setAddresses(List<Address> addresses) {
        this.addresses = addresses;
    }
```

```
    // Other getters and setters as before
}

// EmployeeController.java
package com.example.controller;

import com.example.model.Employee;
import com.example.model.Address;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.Arrays;

@RestController
public class EmployeeController {

    @GetMapping("/employee")
    public Employee getEmployee() {
        Employee employee = new Employee();
        employee.setName("John Doe");
        employee.setId(1);
        employee.setSalary(50000);

        Address address1 = new Address();
        address1.setStreet("123 Main St");
        address1.setCity("Cityville");
        address1.setState("State");
        address1.setZipCode("12345");

        Address address2 = new Address();
        address2.setStreet("456 Oak St");
        address2.setCity("Townsville");
        address2.setState("State");
        address2.setZipCode("67890");

        employee.setAddresses(Arrays.asList(address1, address2));

        return employee;
    }
}
```

Task 3: Write a Spring Application Program to Display the Simple Interest

```
// SimpleInterestService.java
package com.example.service;

public class SimpleInterestService {
    public double calculateSimpleInterest(double principal, double rate,
        return (principal * rate * time) / 100;
    }
}

// SimpleInterestController.java
package com.example.controller;

import com.example.service.SimpleInterestService;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class SimpleInterestController {

    @GetMapping("/simple-interest")
    public double getSimpleInterest() {
        SimpleInterestService service = new SimpleInterestService();
        double principal = 1000;
        double rate = 5;
        double time = 2;
        return service.calculateSimpleInterest(principal, rate, time);
    }
}
```

Task 4: Implement Code for Testing the Application Using REST API and Generate the Test Report

You can test your Spring REST API using tools like Postman or JUnit testing in Java.

For JUnit testing:

```
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
```

```
public class EmployeeControllerTest {

    @Test
    public void testGetEmployee() {
        EmployeeController controller = new EmployeeController();
        Employee employee = controller.getEmployee();
        assertNotNull(employee);
        assertEquals("John Doe", employee.getName());
    }
}
```

Task 5: Create a Project for Implementing the REST API and Display the Outputs

The REST API is already covered in the previous steps using `@RestController` and `@GetMapping`. You can run this project in a Spring Boot application and test it using Postman or directly in your browser.

Task 6: Develop a Project Using Spring Framework and Integrate the JSON Format Response

Spring automatically converts Java objects to JSON format using Jackson (included by default in Spring Boot). Ensure your Spring Boot project has the `spring-boot-starter-web` dependency.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Your controller's response will be automatically serialized into JSON:

```
@RestController
public class EmployeeController {

    @GetMapping("/employee")
    public Employee getEmployee() {
        Employee employee = new Employee();
        employee.setName("John Doe");
        employee.setId(1);
```

```
        employee.setSalary(50000);
        return employee;  // This will be returned as a JSON response
    }
}
```

Task 7: Create a Project with the Integration of HTTP CRUD Methods and Display the Result

Using `@RestController`, you can implement CRUD operations like this:

```
@RestController
@RequestMapping("/employee")
public class EmployeeController {

    private Map<Integer, Employee> employees = new HashMap<>();

    @PostMapping
    public void createEmployee(@RequestBody Employee employee) {
        employees.put(employee.getId(), employee);
    }

    @GetMapping("/{id}")
    public Employee getEmployee(@PathVariable int id) {
        return employees.get(id);
    }

    @PutMapping("/{id}")
    public void updateEmployee(@PathVariable int id, @RequestBody Employe
        employees.put(id, employee);
    }

    @DeleteMapping("/{id}")
    public void deleteEmployee(@PathVariable int id) {
        employees.remove(id);
    }
}
```
