

Worksheet No.: 4

Student Name: Harsh

Branch: MCA (General)

Semester: 2nd

Subject Name: Advanced internet programming
lab

UID: 24MCA20045

Section/Group: 1-A

Date of Performance :
03/03/2025

Subject Code: 24CAP-652

1. Aim of the practical:

This project is a **Java-based Employee Registration Web Application** using **Jakarta EE, Hibernate, and MySQL**. The application allows users to submit employee details, including their name, email, phone number, and address, through an **HTML form**. The data is then processed and stored in a **MySQL database** using **Hibernate ORM**.

Key Components:

1. **Frontend (index.html):** A responsive **Employee Form** for collecting user input.
2. **Backend Servlet (EmployeeServlet.java):** Handles form submissions, validates input, and persists employee data.
3. **Data Access Layer (EmployeeDAO.java):** Manages Hibernate sessions and saves the employee to the database.
4. **Model Classes (Employee.java, Address.java):** Define entity relationships with **JPA annotations**.
5. **Hibernate Configuration (hibernate.cfg.xml):** Configures database connection settings.
6. **Maven (pom.xml):** Manages dependencies like **Hibernate, MySQL, and Jakarta EE APIs**.

Working Process:

1. The user fills out the **employee registration form** and submits it.
2. The **servlet receives the request**, validates input using **Hibernate Validator**, and maps data to **Employee & Address entities**.
3. The **Hibernate ORM** saves the employee and address details in the **MySQL database**.
4. A success or error message is displayed based on the validation and database operation.

This project demonstrates **CRUD operations, database interaction, form validation, and MVC architecture** in a Java web application.

Code for experiment/practical:

- **Index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Employee Form</title>
  <style>
body {
  font-family:
Arial, sans-serif;
background-color: #f4f4f4;
align-items: center;
height: 100vh;
}
.container {
background: #fff;
padding: 20px;
border-radius: 10px;
box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);
width: 90%;
max-width: 400px;
}
h2 {
text-align: center;
color: #333;
}
label {
display: block;
margin-top: 10px;
}
input {
width: 100%; padding: 8px; margin-top: 5px; border:
1px solid #ccc;
border-radius: 5px;
```

```
@media (max-width: 500px) {  
  .container {  
    width: 100%;  
    padding: 15px;  
  }  
}  
</style>  
</head>  
<body>  
  <div class="container">  
    <h2>Employee Form</h2>  
    <form action="employee" method="POST">  
      <label for="name">Name:</label>  
      <input type="text" id="name" name="name" required>  
      <label for="email">Email:</label>  
      <input type="email" id="email" name="email" required>  
      <label for="phone">Phone:</label>  
      <input type="text" id="phone" name="phone" required>  
      <strong>Address</strong>  
      <label for="street">Street:</label>  
      <input type="text" id="street" name="street" required>  
      <label for="city">City:</label>  
      <input type="text" id="city" name="city" required>  
      <label for="state">State:</label>  
      <input type="text" id="state" name="state" required>  
      <label for="zipCode">Zip Code:</label>  
      <input type="text" id="zipCode" name="zipCode" required>  
      <button type="submit" class="submit-btn">Submit</button>  
    </form>  
  </div>  
</body>  
</html>
```

EmployeeDAO.java: -

```
package com.employee.dao;  
import  
org.hibernate.Session;  
import  
org.hibernate.SessionFacto  
ry; import
```

```
org.hibernate.cfg.Configur
ation; import
com.employee.model.Employe
e; import
com.employee.model.Address
; public class EmployeeDao
{
    private static final
SessionFactory factory;
static {
    try {
        factory = new Configuration().configure("hibernate.cfg.xml")
            .addAnnotatedClass(Employee.class)
            .addAnnotatedClass(Address.class)
            .buildSessionFactory();
    } catch (Throwable ex) {
        throw new ExceptionInInitializerError("SessionFactory creation failed: " + ex);
    }
}

    public static void saveEmployee(Employee employee) {
        Session session =
factory.openSession();        try {
            session.beginTransaction();
session.save(employee);
            session.getTransaction().commit();
        } catch (Exception e) {
            if (session.getTransaction() != null) {
session.getTransaction().rollback();
            }

e.printStackTrace();
        } finally {
            session.close();
        }
    }

    public static void
closeFactory() {                if
(factory != null) {
        factory.close();
    }
}
}
```

EmployeeServlet.java: -

```
package
com.employee.servlet;
import
com.employee.dao.Empl
oyeeDao; import
java.io.IOException;

import jakarta.servlet.ServletException;

import
jakarta.servlet.annotation.WebS
ervlet; import
jakarta.servlet.http.HttpServlet;
import
jakarta.servlet.http.HttpServlet
Request; import
jakarta.servlet.http.HttpServlet
Response; import
jakarta.validation.Validation;
import
jakarta.validation.Validator;
import
jakarta.validation.ValidatorFacto
ry; import java.util.Set; import
jakarta.validation.ConstraintViol
ation; import
com.employee.model.Address;
import
com.employee.model.Employee
; @WebServlet("/employee")

public class EmployeeServlet extends HttpServlet {

    @Override

    protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {        String name = request.getParameter("name");

        String email = request.getParameter("email");

        String phone = request.getParameter("phone");
```

```
String street = request.getParameter("street");

String city = request.getParameter("city");

String state = request.getParameter("state");

String zipCode = request.getParameter("zipCode");

Address address = new
Address();
address.setStreet(street);
address.setCity(city);
address.setState(state);
address.setZipCode(zipCod
e);    Employee employee
= new Employee();
employee.setName(name);
employee.setEmail(email);
employee.setPhone(phone)
;
employee.setAddress(addre
ss);

ValidatorFactory factory = Validation.buildDefaultValidatorFactory();

Validator validator = factory.getValidator();

Set<ConstraintViolation<Employee>> violations =
validator.validate(employee);    if (violations.isEmpty()) {

    EmployeeDao.saveEmployee(employee);

    response.getWriter().println("Employee saved successfully!");

} else {

    for (ConstraintViolation<Employee> violation : violations) {
response.getWriter().println(violation.getMessage());

    }

}

}

}
```

Employee.java :-

```
package com.employee.model;
import jakarta.persistence.*;
import
jakarta.validation.constraints.E
mail; import
jakarta.validation.constraints.N
otEmpty; import
jakarta.validation.constraints.P
attern;

@Entity

public class Employee {

    @Id

    @GeneratedValue(strategy =
GenerationType.IDENTITY) private int id;

    @NotEmpty(message = "Name cannot
be empty") private String name;

    @Email(message = "Email should be
valid") @NotEmpty(message = "Email
cannot be empty") private String email;

    @Pattern(regexp = "[0-9]{10}$", message = "Phone should be 10 digits")

    @NotEmpty(message = "Phone cannot
be empty") private String phone;

    @OneToOne(cascade = CascadeType.ALL)

    @JoinColumn(name = "address_id", referencedColumnName = "id") // Explicit reference to the
Address ID private Address address;

    // Default constructor
    (required by JPA) public
Employee() {}

    // Parameterized constructor (optional)
```

```
public Employee(String name, String email, String phone,
Address address) {    this.name = name; this.email =
email;    this.phone = phone;

    this.address = address;
}

// Getters
and Setters
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 String
getEmail() {
return email;
}

public void setEmail(String email) {

    this.email = email;
}

public String getPhone() {

    return phone;
```



```
}  
  
public void setPhone(String phone) {  
    this.phone = phone;  
}  
  
public Address getAddress() {  
    return address;  
}  
  
public void setAddress(Address address) {  
    this.address = address;  
}  
  
@Override  
de  
public  
String  
toString()  
{  
    return  
"Employee  
{" +  
        "id=" + id +  
        ", name=" + name + "\" +  
        ", email=" + email + "\" +  
        ", phone=" + phone + "\" +  
        ", address=" + address +  
        "};  
}  
}
```

Address.java :-

```
package com.employee.model;
import jakarta.persistence.*;
import
jakarta.validation.constraints.N
otEmpty; import
jakarta.validation.constraints.P
attern;

@Entity

public class Address {

    @Id

    @GeneratedValue(strategy =
GenerationType.IDENTITY) private int id;

    @NotEmpty(message = "Street cannot
be empty") private String street;

    @NotEmpty(message = "City cannot be empty")

    private String city;

    @NotEmpty(message = "State cannot
be empty") private String state;

    @Pattern(regexp = "[0-9]{5,6}$", message = "Zip Code should be 5 or 6 digits") private String
zipCode;

    // Default constructor (required by JPA)

    public
Address() {}
    //
    Parameterized
    constructor

    public Address(String street, String city, String state,
String zipCode) { this.street = street; this.city
= city; this.state = state;
```

```
this.zipCode = zipCode;

}

// Getters
and Setters
public int
getId() {

    return id;

}

public void setId(int id) {

    this.id = id;

}

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;  
}  
  
@Override  
public  
String  
toString()  
{  
    return "Address{" +  
        "id=" + id +  
        ", street=" + street + "\" +  
        ", city=" + city + "\" +  
        ", state=" + state + "\" +  
        ", zipCode=" + zipCode + "\" +  
        '};  
}  
}
```

Hibernate.cfg.xml:-

```
<!DOCTYPE hibernate-configuration PUBLIC  
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">  
<hibernate-configuration>  
    <session-factory>
```

```
<!-- JDBC Database connection settings -->

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>

<property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/employeedb</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">Nandan@123</property>

<!-- JDBC connection pool settings -->

<property name="hibernate.c3p0.min_size">5</property>

<property name="hibernate.c3p0.max_size">20</property>

<property name="hibernate.c3p0.timeout">300</property>

<!-- Enable Hibernate's automatic session context management -->

<property name="hibernate.current_session_context_class">thread</property>

<!-- Echo all executed SQL to stdout -->

<property name="hibernate.show_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Mention annotated classes -->

<mapping class="com.employee.model.Employee"/>

<mapping class="com.employee.model.Address"/>

</session-factory>

</hibernate-configuration>
```

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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>

<groupId>com.employee</groupId>
<artifactId>EmployeeApp</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>
<name>EmployeeApp-1.0-SNAPSHOT</name>
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
<dependencies>
  <!-- Hibernate Core -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>6.3.1.Final</version>
  </dependency>
  <!-- MySQL Driver -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.33</version>
  </dependency>
  <!-- Jakarta Persistence API -->
  <dependency>
    <groupId>jakarta.persistence</groupId>
    <artifactId>jakarta.persistence-api</artifactId>
    <version>3.1.0</version>
  </dependency>
  <!-- Hibernate Validator (Bean Validation Implementation) -->
  <dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>8.0.1.Final</version>
  </dependency>
  <!-- Jakarta Expression Language (Required for Hibernate Validator) -->
  <dependency>
    <groupId>org.glassfish.expressly</groupId>
    <artifactId>expressly</artifactId>
    <version>5.0.0</version>
```

```
</dependency>
<!-- Jakarta EE API -->
<dependency>
  <groupId>jakarta.platform</groupId>
  <artifactId>jakarta.jakartaee-api</artifactId>
  <version>${jakartaee}</version>
  <scope>provided</scope>
</dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
      <version>3.13.0</version>
      <configuration>
        <release>11</release>
      </configuration>
    </plugin>

    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>3.3.2</version>
    </plugin>
  </plugins>
</build>
</project>
```

2. Result/Output/Writing Summary:

Employee Form

Name:

Email:

Phone:

Address

Street:

City:

State:

Zip Code:

[Submit](#)

Database Output:

<input type="checkbox"/>	Edit	Copy	Delete	1	Ananyasharma7654321@gmail.com	Ananya Sharma	8219260016	1
<input type="checkbox"/>	Edit	Copy	Delete	2	Ananyasharma721@gmail.com	hfhfhf	8786876621	2
<input type="checkbox"/>	Edit	Copy	Delete	3	neetsharma7654321@gmail.com	neeta	8219260016	3
<div> ← → </div> <div> <div>id</div> <div>city</div> <div>state</div> <div>street</div> <div>zipCode</div> </div>								
<input type="checkbox"/>	Edit	Copy	Delete	1	chandigarh	punjab	mohali	172839
<input type="checkbox"/>	Edit	Copy	Delete	2	chandigarh	punjab	mohali	172666
<input type="checkbox"/>	Edit	Copy	Delete	3	chandigarh	punjab	mohali	172666

3. Learning outcomes (What I have learnt):

1. Understanding Full-Stack Web Development
 - Learn how to integrate frontend (HTML, CSS, forms) with backend (Java, Servlets, Hibernate, MySQL) in a web application.
2. Working with Hibernate & JPA for Database Management
 - Gain hands-on experience in mapping Java objects to database tables, using Hibernate ORM for efficient data persistence.
3. Servlets & HTTP Request Handling
 - Learn how to handle form submissions, process HTTP POST requests, and manage server-side validation using Jakarta EE Servlets.
4. Data Validation using Hibernate Validator
 - Understand how to enforce data integrity through annotations like @NotEmpty, @Email, and @Pattern for input validation.
5. Maven Dependency Management & Configuration
 - Learn how to configure a Maven-based project, manage dependencies (pom.xml), and use Hibernate, MySQL Connector, and Jakarta EE APIs efficiently.

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Worksheet		8 Marks
2.	Viva		10 Marks
3.	Simulation		12 Marks
	Total		30 Marks

Teacher Signature|