Q. Write a Java program to fetch data from web.xml to Servlet using ServletConfig.

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.ServletConfig;

@WebServlet("/configDemo")

public class ConfigDemoServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

private String appName;

@Override

public void init(ServletConfig config) throws ServletException {

super.init(config);

// Fetching data from web.xml using ServletConfig

appName = config.getInitParameter("appName");

}

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

// Displaying the fetched data

response.getWriter().println("<html><body>");

response.getWriter().println("<h2>Servlet Configuration Demo</h2>");

response.getWriter().println("<p>Application Name: " + appName + "</p>");

response.getWriter().println("</body></html>");

}

}

In Web.xml:

<servlet>

<servlet-name>ConfigDemoServlet</servlet-name>

<servlet-class>configuration.ConfigDemoServlet</servlet-class>

<init-param>

<param-name>appName</param-name>

<param-value>YourAppName</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>ConfigDemoServlet</servlet-name>

<url-pattern>/configDemo</url-pattern>

</servlet-mapping>

Q. Write a Java program to fetch data from web.xml to Servlet using ServletCotext.

import java.io.IOException;

import javax.servlet.ServletConfig;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/contextDemo")

public class ContextDemoServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

private String appName;

@Override

public void init(ServletConfig config) throws ServletException {

super.init(config);

// Fetching data from web.xml using ServletContext

ServletContext context = getServletContext();

appName = context.getInitParameter("appName");

}

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

// Displaying the fetched data

response.getWriter().println("<html><body>");

response.getWriter().println("<h2>Servlet Context Demo</h2>");

response.getWriter().println("<p>Application Name: " + appName + "</p>");

response.getWriter().println("</body></html>");

}

}

In Web.xml:

<context-param>

<param-name>appName</param-name>

<param-value>YourAppName</param-value>

</context-param>

<servlet>

<servlet-name>ContextDemoServlet</servlet-name>

<servlet-class>your.package.name.ContextDemoServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ContextDemoServlet</servlet-name>

<url-pattern>/contextDemo</url-pattern>

</servlet-mapping>

Q. Write a Java program to submit student information (fname, lname, email, mobile, gender, password) using jsp form to servlet. Fetch data at servlet and print all the data in console.

Form.jsp file:

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Student Information Form</title>

</head>

<body>

<h2>Student Information Form</h2>

<form action="StudentInfoServlet" method="post">

First Name: <input type="text" name="fname" required><br>

Last Name: <input type="text" name="lname" required><br>

Email: <input type="email" name="email" required><br>

Mobile: <input type="tel" name="mobile" required><br>

Gender:

<input type="radio" name="gender" value="Male" checked> Male

<input type="radio" name="gender" value="Female"> Female<br>

Password: <input type="password" name="password" required><br>

<input type="submit" value="Submit">

</form>

</body>

</html>

In StudentServlet.java file:

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/StudentInfoServlet")

public class StudentInfoServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

// Retrieve data from the form

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

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

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

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

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

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

// Print data to the console

System.out.println("First Name: " + fname);

System.out.println("Last Name: " + lname);

System.out.println("Email: " + email);

System.out.println("Mobile: " + mobile);

System.out.println("Gender: " + gender);

System.out.println("Password: " + password);

// Send a response to the browser

PrintWriter out = response.getWriter();

out.println("<html><body>");

out.println("<h2>Student Information</h2>");

out.println("<p>First Name: " + fname + "</p>");

out.println("<p>Last Name: " + lname + "</p>");

out.println("<p>Email: " + email + "</p>");

out.println("<p>Mobile: " + mobile + "</p>");

out.println("<p>Gender: " + gender + "</p>");

out.println("<p>Password: " + password + "</p>");

out.println("</body></html>");

}

}

Q. Write a Java program to fetch all the data from database table and print on jsp page using JSTL SQL tag library.

In DBConnection.java file:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnection {

private static final String JDBC\_URL = "jdbc:mysql://localhost:3306/mydb";

private static final String JDBC\_USER = "root";

private static final String JDBC\_PASSWORD = "123456";

public static Connection getConnection() throws SQLException {

return DriverManager.getConnection(JDBC\_URL, JDBC\_USER, JDBC\_PASSWORD);

}

}

In FetchDataServlet.java file:

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@WebServlet("/FetchDataServlet")

public class FetchDataServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

try {

Connection connection = DBConnection.getConnection();

String query = "SELECT \* FROM your\_table\_name";

PreparedStatement preparedStatement = connection.prepareStatement(query);

ResultSet resultSet = preparedStatement.executeQuery();

HttpSession session = request.getSession();

session.setAttribute("resultSet", resultSet);

response.sendRedirect("displayData.jsp");

} catch (SQLException e) {

e.printStackTrace();

response.getWriter().println("Error fetching data from the database.");

}

}

}

In DisplayData.jsp file:

<%@ page import="java.sql.ResultSet" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Display Data</title>

</head>

<body>

<h2>Data from Database</h2>

<sql:setDataSource var="dataSource" driver="com.mysql.jdbc.Driver"

url="jdbc:mysql://localhost:3306/mydb" user="root"

password="123456" />

<sql:query var="result" dataSource="${dataSource}">

SELECT \* FROM your\_table\_name;

</sql:query>

<table border="1">

<tr>

<th>ID</th>

<th>Name</th>

</tr>

<c:forEach var="row" items="${result.rows}">

<tr>

<td><c:out value="${row.id}" /></td>

<td><c:out value="${row.name}" /></td>

</tr>

</c:forEach>

</table>

</body>

</html>

Q. W.A.J.P to insert below data from jsp to MySQL database using “. cfg.cml” and “.hbm.xml” file. Variable id must be primary key and auto increment Int id, String first name, last name, email, mobile, password, gender

Hibernate.cfg.xml file:

<!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.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

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

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

<property name="hibernate.connection.password">123456</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>

<property name="hibernate.c3p0.max\_statements">50</property>

<property name="hibernate.c3p0.idle\_test\_period">3000</property>

<!-- Specify dialect -->

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

<mapping resource="Student.hbm.xml"/>

</session-factory>

</hibernate-configuration>

Student.hbm.xml file:

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="com.example.Student" table="student">

<id name="id" type="int">

<column name="id" />

<generator class="identity" />

</id>

<property name="firstName" type="string">

<column name="first\_name" />

</property>

<property name="lastName" type="string">

<column name="last\_name" />

</property>

<property name="email" type="string">

<column name="email" />

</property>

<property name="mobile" type="string">

<column name="mobile" />

</property>

<property name="password" type="string">

<column name="password" />

</property>

<property name="gender" type="string">

<column name="gender" />

</property>

</class>

</hibernate-mapping>

Student.java file:

package com.example;

public class Student {

private int id;

private String firstName;

private String lastName;

private String email;

private String mobile;

private String password;

private String gender;

// Getters and setters

// Constructors

}

InsertDataServlet.java file:

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/InsertDataServlet")

public class InsertDataServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) {

try {

Configuration cfg = new Configuration().configure("hibernate.cfg.xml");

SessionFactory factory = cfg.buildSessionFactory();

Session session = factory.openSession();

Transaction t = session.beginTransaction();

// Get data from JSP

String firstName = request.getParameter("fname");

String lastName = request.getParameter("lname");

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

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

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

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

// Create Student object and set values

Student student = new Student();

student.setFirstName(firstName);

student.setLastName(lastName);

student.setEmail(email);

student.setMobile(mobile);

student.setPassword(password);

student.setGender(gender);

// Save the student object to the database

session.save(student);

t.commit();

session.close();

response.getWriter().println("Data inserted successfully!");

} catch (Exception e) {

e.printStackTrace();

response.getWriter().println("Error inserting data into the database.");

}

}

}

Q. Write a program to establish One to One Relationship between given 2 classes and perform a CRUD operation.

Student:

Int studentId; String studentName; Address; Address: Int addressId;

String street, city, state, zip code;

In Student.java file:

import javax.persistence.\*;

@Entity

@Table(name = "students")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "student\_id")

private int studentId;

@Column(name = "student\_name")

private String studentName;

@OneToOne(cascade = CascadeType.ALL)

@JoinColumn(name = "address\_id")

private Address address;

// getters and setters

}

In Address.java file:

import javax.persistence.\*;

@Entity

@Table(name = "addresses")

public class Address {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "address\_id")

private int addressId;

private String street;

private String city;

private String state;

@Column(name = "zip\_code")

private String zipCode;

// getters and setters

}

In CRUDOperations.java file:

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class CRUDOperations {

private static SessionFactory factory;

static {

factory = new Configuration().configure("hibernate.cfg.xml")

.addAnnotatedClass(Student.class)

.addAnnotatedClass(Address.class)

.buildSessionFactory();

}

public static void main(String[] args) {

CRUDOperations crud = new CRUDOperations();

// Create operation

int studentId = crud.createStudent("John Doe", "123 Main St", "City", "State", "12345");

// Read operation

Student student = crud.getStudent(studentId);

System.out.println("Read operation: " + student);

// Update operation

crud.updateStudent(studentId, "John Updated");

// Read operation after update

student = crud.getStudent(studentId);

System.out.println("Read operation after update: " + student);

// Delete operation

crud.deleteStudent(studentId);

// Read operation after delete

student = crud.getStudent(studentId);

System.out.println("Read operation after delete: " + student);

}

private int createStudent(String studentName, String street, String city, String state, String zipCode) {

Session session = factory.openSession();

Transaction tx = null;

int studentId = 0;

try {

tx = session.beginTransaction();

Address address = new Address();

address.setStreet(street);

address.setCity(city);

address.setState(state);

address.setZipCode(zipCode);

Student student = new Student();

student.setStudentName(studentName);

student.setAddress(address);

studentId = (int) session.save(student);

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return studentId;

}

private Student getStudent(int studentId) {

Session session = factory.openSession();

Student student = null;

try {

student = session.get(Student.class, studentId);

} catch (Exception e) {

e.printStackTrace();

} finally {

session.close();

}

return student;

}

private void updateStudent(int studentId, String updatedName) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Student student = session.get(Student.class, studentId);

student.setStudentName(updatedName);

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

private void deleteStudent(int studentId) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Student student = session.get(Student.class, studentId);

if (student != null) {

session.delete(student);

}

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

}

Q. Write a program to establish one to many and Many to One relationship between given 2 classes and perform a CRUD operation.

Cart:

int cartId; double total;

String name; Set<Item> items;

Items:

Int id;

String itemId;

Double itemTotal;

Int quantity;

Cart cart;

In Cart.java file:

import javax.persistence.\*;

import java.util.HashSet;

import java.util.Set;

@Entity

@Table(name = "carts")

public class Cart {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "cart\_id")

private int cartId;

private double total;

private String name;

@OneToMany(mappedBy = "cart", cascade = CascadeType.ALL, fetch = FetchType.EAGER)

private Set<Item> items = new HashSet<>();

// getters and setters

}

In Item.java file:

import javax.persistence.\*;

@Entity

@Table(name = "items")

public class Item {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "item\_id")

private int id;

private String itemId;

private double itemTotal;

private int quantity;

@ManyToOne

@JoinColumn(name = "cart\_id")

private Cart cart;

// getters and setters

}

In CRUDOperation.java file:

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class CRUDOperations {

private static SessionFactory factory;

static {

factory = new Configuration().configure("hibernate.cfg.xml")

.addAnnotatedClass(Cart.class)

.addAnnotatedClass(Item.class)

.buildSessionFactory();

}

public static void main(String[] args) {

CRUDOperations crud = new CRUDOperations();

// Create operation

int cartId = crud.createCart("Shopping Cart");

int itemId1 = crud.createItem("item1", 10.0, 2, cartId);

int itemId2 = crud.createItem("item2", 15.0, 1, cartId);

// Read operation

crud.readCart(cartId);

// Update operation

crud.updateItem(itemId1, 3);

// Read operation after update

crud.readCart(cartId);

// Delete operation

crud.deleteItem(itemId2);

// Read operation after delete

crud.readCart(cartId);

}

private int createCart(String name) {

Session session = factory.openSession();

Transaction tx = null;

int cartId = 0;

try {

tx = session.beginTransaction();

Cart cart = new Cart();

cart.setName(name);

cartId = (int) session.save(cart);

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return cartId;

}

private int createItem(String itemId, double itemTotal, int quantity, int cartId) {

Session session = factory.openSession();

Transaction tx = null;

int itemIdGenerated = 0;

try {

tx = session.beginTransaction();

Cart cart = session.get(Cart.class, cartId);

if (cart != null) {

Item item = new Item();

item.setItemId(itemId);

item.setItemTotal(itemTotal);

item.setQuantity(quantity);

item.setCart(cart);

itemIdGenerated = (int) session.save(item);

}

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return itemIdGenerated;

}

private void readCart(int cartId) {

Session session = factory.openSession();

try {

Cart cart = session.get(Cart.class, cartId);

if (cart != null) {

System.out.println("Cart Details: " + cart.getName());

System.out.println("Items in Cart:");

for (Item item : cart.getItems()) {

System.out.println("Item ID: " + item.getItemId() + ", Quantity: " + item.getQuantity());

}

} else {

System.out.println("Cart not found.");

}

} catch (Exception e) {

e.printStackTrace();

} finally {

session.close();

}

}

private void updateItem(int itemId, int newQuantity) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Item item = session.get(Item.class, itemId);

if (item != null) {

item.setQuantity(newQuantity);

}

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

private void deleteItem(int itemId) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Item item = session.get(Item.class, itemId);

if (item != null) {

session.delete(item);

}

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

}

Q. Write a program to establish many to many relationships between given 2 classes and perform a CRUD operation.

Reader:

Int readerId;

String email, firstName, lastName;

Set<Subscription> subscriptions; Subscription: Int subscriptionId;

String subscriptionName; Set<Reader> readers;

In Reader.java file:

import javax.persistence.\*;

import java.util.HashSet;

import java.util.Set;

@Entity

@Table(name = "readers")

public class Reader {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "reader\_id")

private int readerId;

private String email;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}, fetch = FetchType.EAGER)

@JoinTable(

name = "reader\_subscription",

joinColumns = @JoinColumn(name = "reader\_id"),

inverseJoinColumns = @JoinColumn(name = "subscription\_id")

)

private Set<Subscription> subscriptions = new HashSet<>();

// getters and setters

}

In Subscription.java file:

import javax.persistence.\*;

import java.util.HashSet;

import java.util.Set;

@Entity

@Table(name = "subscriptions")

public class Subscription {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "subscription\_id")

private int subscriptionId;

@Column(name = "subscription\_name")

private String subscriptionName;

@ManyToMany(mappedBy = "subscriptions", cascade = {CascadeType.PERSIST, CascadeType.MERGE}, fetch = FetchType.EAGER)

private Set<Reader> readers = new HashSet<>();

// getters and setters

}

In CRUDOperation.java file:

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class CRUDOperations {

private static SessionFactory factory;

static {

factory = new Configuration().configure("hibernate.cfg.xml")

.addAnnotatedClass(Reader.class)

.addAnnotatedClass(Subscription.class)

.buildSessionFactory();

}

public static void main(String[] args) {

CRUDOperations crud = new CRUDOperations();

// Create operation

int readerId1 = crud.createReader("reader1@example.com", "John", "Doe");

int readerId2 = crud.createReader("reader2@example.com", "Jane", "Smith");

int subscriptionId1 = crud.createSubscription("Subscription1");

int subscriptionId2 = crud.createSubscription("Subscription2");

crud.addSubscriptionToReader(readerId1, subscriptionId1);

crud.addSubscriptionToReader(readerId1, subscriptionId2);

crud.addSubscriptionToReader(readerId2, subscriptionId1);

// Read operation

crud.readReader(readerId1);

// Update operation

crud.updateReaderLastName(readerId1, "UpdatedLastName");

// Read operation after update

crud.readReader(readerId1);

// Delete operation

crud.removeSubscriptionFromReader(readerId1, subscriptionId1);

// Read operation after delete

crud.readReader(readerId1);

}

private int createReader(String email, String firstName, String lastName) {

Session session = factory.openSession();

Transaction tx = null;

int readerId = 0;

try {

tx = session.beginTransaction();

Reader reader = new Reader();

reader.setEmail(email);

reader.setFirstName(firstName);

reader.setLastName(lastName);

readerId = (int) session.save(reader);

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return readerId;

}

private int createSubscription(String subscriptionName) {

Session session = factory.openSession();

Transaction tx = null;

int subscriptionId = 0;

try {

tx = session.beginTransaction();

Subscription subscription = new Subscription();

subscription.setSubscriptionName(subscriptionName);

subscriptionId = (int) session.save(subscription);

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

return subscriptionId;

}

private void addSubscriptionToReader(int readerId, int subscriptionId) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Reader reader = session.get(Reader.class, readerId);

Subscription subscription = session.get(Subscription.class, subscriptionId);

if (reader != null && subscription != null) {

reader.getSubscriptions().add(subscription);

subscription.getReaders().add(reader);

}

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

private void readReader(int readerId) {

Session session = factory.openSession();

try {

Reader reader = session.get(Reader.class, readerId);

if (reader != null) {

System.out.println("Reader Details: " + reader.getEmail() + ", " + reader.getFirstName() + ", " + reader.getLastName());

System.out.println("Subscriptions:");

for (Subscription subscription : reader.getSubscriptions()) {

System.out.println("Subscription ID: " + subscription.getSubscriptionId() + ", Name: " + subscription.getSubscriptionName());

}

} else {

System.out.println("Reader not found.");

}

} catch (Exception e) {

e.printStackTrace();

} finally {

session.close();

}

}

private void updateReaderLastName(int readerId, String updatedLastName) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Reader reader = session.get(Reader.class, readerId);

if (reader != null) {

reader.setLastName(updatedLastName);

}

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

private void removeSubscriptionFromReader(int readerId, int subscriptionId) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Reader reader = session.get(Reader.class, readerId);

Subscription subscription = session.get(Subscription.class, subscriptionId);

if (reader != null && subscription != null) {

reader.getSubscriptions().remove(subscription);

subscription.getReaders().remove(reader);

}

tx.commit();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

} finally {

session.close();

}

}

}

=======================================================

Q. Write a program to demonstrate the setter based dependency injection.

In Employee.java file:

public class Employee {

private String name;

private int employeeId;

// Getter and Setter methods

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getEmployeeId() {

return employeeId;

}

public void setEmployeeId(int employeeId) {

this.employeeId = employeeId;

}

@Override

public String toString() {

return "Employee [name=" + name + ", employeeId=" + employeeId + "]";

}

}

In EmployeeService.java file:

public class EmployeeService {

private Employee employee;

// Setter method for dependency injection

public void setEmployee(Employee employee) {

this.employee = employee;

}

public void displayEmployeeDetails() {

if (employee != null) {

System.out.println("Employee Details: " + employee);

} else {

System.out.println("Employee details not available.");

}

}

}

In applicationContext.xml file:

<?xml version="1.0" encoding="UTF-8"?>

<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/spring-beans.xsd">

<!-- Define the Employee bean -->

<bean id="employee" class="com.example.Employee">

<property name="name" value="John Doe"/>

<property name="employeeId" value="123"/>

</bean>

<!-- Define the EmployeeService bean with setter injection -->

<bean id="employeeService" class="com.example.EmployeeService">

<property name="employee" ref="employee"/>

</bean>

</beans>

In MainApp.java file:

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load the Spring context from the configuration file

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

// Retrieve the EmployeeService bean from the context

EmployeeService employeeService = context.getBean("employeeService", EmployeeService.class);

// Display employee details using the EmployeeService

employeeService.displayEmployeeDetails();

}

}

Q. Write a program to demonstrate the constructor based dependency injection.

In Customer.java file:

public class Customer {

private String name;

private int customerId;

// Constructor

public Customer(String name, int customerId) {

this.name = name;

this.customerId = customerId;

}

// Getter methods

public String getName() {

return name;

}

public int getCustomerId() {

return customerId;

}

@Override

public String toString() {

return "Customer [name=" + name + ", customerId=" + customerId + "]";

}

}

In CustomerService.java file:

public class CustomerService {

private Customer customer;

// Constructor-based dependency injection

public CustomerService(Customer customer) {

this.customer = customer;

}

public void displayCustomerDetails() {

if (customer != null) {

System.out.println("Customer Details: " + customer);

} else {

System.out.println("Customer details not available.");

}

}

}

In applicationContext.xml file:

<?xml version="1.0" encoding="UTF-8"?>

<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/spring-beans.xsd">

<!-- Define the Customer bean -->

<bean id="customer" class="com.example.Customer">

<constructor-arg name="name" value="Alice"/>

<constructor-arg name="customerId" value="456"/>

</bean>

<!-- Define the CustomerService bean with constructor injection -->

<bean id="customerService" class="com.example.CustomerService">

<constructor-arg ref="customer"/>

</bean>

</beans>

In MainApp.java file:

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load the Spring context from the configuration file

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

// Retrieve the CustomerService bean from the context

CustomerService customerService = context.getBean("customerService", CustomerService.class);

// Display customer details using the CustomerService

customerService.displayCustomerDetails();

}

}

Q. Write a program to demonstrate the object based dependency injection and also implement inner bean concept in your spring beans configuration file.

In Book.java file:

public class Book {

private String title;

private String author;

// Constructor

public Book(String title, String author) {

this.title = title;

this.author = author;

}

// Getter methods

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

@Override

public String toString() {

return "Book [title=" + title + ", author=" + author + "]";

}

}

In Library.java file:

public class Library {

private Book book;

// Setter method for object-based dependency injection

public void setBook(Book book) {

this.book = book;

}

public void displayLibraryDetails() {

if (book != null) {

System.out.println("Library Details: " + book);

} else {

System.out.println("Library details not available.");

}

}

}

In applicationContext.xml file:

<?xml version="1.0" encoding="UTF-8"?>

<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/spring-beans.xsd">

<!-- Define the Library bean with object-based dependency injection using inner bean -->

<bean id="library" class="com.example.Library">

<property name="book">

<bean class="com.example.Book">

<constructor-arg name="title" value="Spring in Action"/>

<constructor-arg name="author" value="Craig Walls"/>

</bean>

</property>

</bean>

</beans>

In MainApp.java file:

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load the Spring context from the configuration file

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

// Retrieve the Library bean from the context

Library library = context.getBean("library", Library.class);

// Display library details using the Library bean

library.displayLibraryDetails();

}

}

Q. Write a program to collect 5 student information using spring collection(List) in spring bean configuration file.

Student:

Int id;

String fname, lname, email, mobile.

In Student.java file:

public class Student {

private int id;

private String fname;

private String lname;

private String email;

private String mobile;

}

In applicationContext.xml file:

<?xml version="1.0" encoding="UTF-8"?>

<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/spring-beans.xsd">

<!-- Define the Student bean -->

<bean id="student1" class="com.example.Student">

<property name="id" value="1"/>

<property name="fname" value="John"/>

<property name="lname" value="Doe"/>

<property name="email" value="john.doe@example.com"/>

<property name="mobile" value="123-456-7890"/>

</bean>

<bean id="student2" class="com.example.Student">

<property name="id" value="2"/>

<property name="fname" value="Jane"/>

<property name="lname" value="Smith"/>

<property name="email" value="jane.smith@example.com"/>

<property name="mobile" value="987-654-3210"/>

</bean>

<bean id="student3" class="com.example.Student">

<!-- Properties for student3 -->

<!-- ... -->

</bean>

<bean id="student4" class="com.example.Student">

<!-- Properties for student4 -->

<!-- ... -->

</bean>

<bean id="student5" class="com.example.Student">

<!-- Properties for student5 -->

<!-- ... -->

</bean>

<!-- Define a List to hold the students -->

<bean id="studentList" class="java.util.ArrayList">

<constructor-arg>

<list>

<ref bean="student1"/>

<ref bean="student2"/>

<ref bean="student3"/>

<ref bean="student4"/>

<ref bean="student5"/>

</list>

</constructor-arg>

</bean>

</beans>

In MainApp.java file:

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.util.List;

public class MainApp {

public static void main(String[] args) {

// Load the Spring context from the configuration file

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

// Retrieve the List of students from the context

List<Student> studentList = context.getBean("studentList", List.class);

// Display student information

for (Student student : studentList) {

System.out.println("Student: " + student.getId() + ", " + student.getFname() + " " + student.getLname()

+ ", Email: " + student.getEmail() + ", Mobile: " + student.getMobile());

}

}

}