**Week-2 Assignment**

**PRACTICAL :**

**Q1)** **Please do all the CRUD operations using JDBC with Table Employee.**

**ANS: Select.java**

package com.crud\_operations;

import java.sql.Connection;

import java.sql.DatabaseMetaData;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class Select {

public static void main(String[] args) {

// TODO Auto-generated method stub

try {

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/employeetable", "root", "root");

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

ResultSet resultset = stmt.executeQuery("select \* from employee;");

while (resultset.next()) {

System.out.println(resultset.getString("EmpName"));

}

System.out.println("Query has been executes Successfully");

stmt.close();

con.close();

} catch (Exception e) { // Handling Exception in case of Exception

System.out.println("Something went wrong " + e);

}

}

}

**CreateTable.java**

public class CreateTable {

public static void main(String[] args) {

// TODO Auto-generated method stub

try {

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/employeetable", "root", "root");

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

// SQL statement execution

stmt.execute("create table employee(EmpId int, EmpName varchar(20), phone text);");

System.out.println("Query has been executes Successfully");

stmt.close();

con.close();

} catch (Exception e) { // Handling Exception in case of Exception

System.out.println("Something went wrong " + e);

}

}

}

**InsertData.java**

package com.crud\_operations;

import java.sql.Connection;

import java.sql.DatabaseMetaData;

import java.sql.DriverManager;

import java.sql.Statement;

public class InsertData {

public static void main(String[] args) {

// TODO Auto-generated method stub

try {

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/employeetable", "root", "root");

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

// SQL statement execution

stmt.execute("insert into employee values(1,'Rajkumar', '802043');");

stmt.execute("insert into employee values(2,'Mr. ABC', '12345');");

System.out.println("Query has been executes Successfully");

stmt.close();

con.close();

} catch (Exception e) { // Handling Exception in case of Exception

System.out.println("Something went wrong " + e);

}

}

}

**UpdateData.java**

package com.crud\_operations;

import java.sql.Connection;

import java.sql.DatabaseMetaData;

import java.sql.DriverManager;

import java.sql.Statement;

public class UpdateData {

public static void main(String[] args) {

// TODO Auto-generated method stub

try {

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/employeetable", "root", "root");

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

stmt.executeUpdate("update employee set EmpName='Rajkumar V' where EmpId =1;");

System.out.println("Query has been executes Successfully");

stmt.close();

con.close();

} catch (Exception e) { // Handling Exception in case of Exception

System.out.println("Something went wrong " + e);

}

}

}

**DeleteData.java**

package com.crud\_operations;

import java.sql.Connection;

import java.sql.DatabaseMetaData;

import java.sql.DriverManager;

import java.sql.Statement;

public class DeleteData {

public static void main(String[] args) {

// TODO Auto-generated method stub

try {

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/employeetable", "root", "root");

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

// SQL statement execution

stmt.execute("delete from employee where EmpId=2;");

System.out.println("Query has been executes Successfully");

stmt.close();

con.close();

} catch (Exception e) { // Handling Exception in case of Exception

System.out.println("Something went wrong " + e);

}

}

}

**CreateDatabase.java**

package com.crud\_operations;

import java.sql.Connection;

import java.sql.DatabaseMetaData;

import java.sql.DriverManager;

import java.sql.Statement;

public class CreateDataBase {

public static void main(String[] args) {

// TODO Auto-generated method stub

try {

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306", "root", "root");

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

// SQL statement execution

stmt.execute("create database employeetable;");

System.out.println("Query has been executes Successfully");

stmt.close();

con.close();

} catch (Exception e) { // Handling Exception in case of Exception

System.out.println("Something went wrong " + e);

}

}

}

**Q2)** **Develop a restful web service to perform CRUD operations. Entities should have Student , Courses and Teachers.**

**ANS:Student.java**

package com.restful\_crud\_operations;

import java.io.Serializable;

import java.util.List;

import javax.persistence.CascadeType;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

import javax.persistence.OneToOne;

import javax.persistence.Table;

@Entity

@Table(name = "Student")

public class Student implements Serializable{

/\*\*

\*

\*/

private static final long serialVersionUID = -5326101768774864410L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

@Column(name = "StudentID") // naming column as specified name

private int studentID;

@Column(name = "StudentName")

private String studentName;

@Column(name = "StudentAddress")

private String studentAddress;

@Column(name = "StudentContact")

private long studentContact;

@OneToOne(cascade = CascadeType.ALL) // perform Operation on the child class

private Teachers teachers;

@OneToMany(cascade = CascadeType.ALL)

private List<Courses> fees;

public int getStudentID() {

return studentID;

}

public void setStudentID(int studentID) {

this.studentID = studentID;

}

public String getStudentName() {

return studentName;

}

public void setStudentName(String studentName) {

this.studentName = studentName;

}

public String getStudentAddress() {

return studentAddress;

}

public void setStudentAddress(String studentAddress) {

this.studentAddress = studentAddress;

}

public long getStudentContact() {

return studentContact;

}

public void setStudentContact(long studentContact) {

this.studentContact = studentContact;

}

public Teachers getTeachers() {

return teachers;

}

public void setTeachers(Teachers teachers) {

this.teachers = teachers;

}

public List<Courses> getTelephones() {

return fees;

}

public void setTelephones(List<Courses> telephones) {

this.fees = telephones;

}

}

**Teacher.java:**

package com.restful\_crud\_operations;

import java.io.Serializable;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Teachers implements Serializable{

/\*\*

\*

\*/

private static final long serialVersionUID = 8200960945324991376L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int teacherId;

private String subject;

public int getTeacherId() {

return teacherId;

}

public void setTeacherId(int teacherId) {

this.teacherId = teacherId;

}

public String getSubject() {

return subject;

}

public void setSubject(String subject) {

this.subject = subject;

}

}

**Courses.java**

package com.restful\_crud\_operations;

import java.io.Serializable;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Courses implements Serializable{

/\*\*

\*

\*/

private static final long serialVersionUID = 8804092356325376228L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private int courseId, fees;

private String courseName;

public int getCourseId() {

return courseId;

}

public void setCourseId(int courseId) {

this.courseId = courseId;

}

public String getCourseName() {

return courseName;

}

public void setCourseName(String courseName) {

this.courseName = courseName;

}

public long getFees() {

return fees;

}

public void setFees(int fees) {

this.fees = fees;

}

}

**OneToManyRelationship.java**

package com.restful\_crud\_operations;

import java.util.ArrayList;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class OneToManyRelationShip {

public static void main(String[] args) {

// TODO Auto-generated method stub

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

Session session = factory.openSession();

Transaction tx= session.beginTransaction();

// Process to execute the Query

Courses c1=new Courses();

c1.setCourseName("ECE");

c1.setFees(80000);

Courses c2=new Courses();

c2.setCourseName("CSE");

c2.setFees(120000);

List<Courses> fees = new ArrayList<Courses>();

fees.add(c2);

fees.add(c1);

Student student=new Student();

student.setStudentName("Raj");

student.setStudentContact(12345678);

student.setStudentAddress("Plvd");

session.save(c1);

session.save(c2);

session.save(student);

tx.commit();

factory.close();

}

}

**Q3)** **Create your own github account and repository and push, pull and clone a file from command prompt?**

**ANS:**

Firstly get into GITHUB.COM and then sign in into your account or we could sign up by clicking on the signup button and then give the required details to create an account. Then in order to do the next steps to perform the actions.

1. You need to create a new repository and click on the plus sign.

Fill up all the required details, i.e., repository name, description and also make the repository public this time as it is free.

2. Open your Git Bash.

Git Bash can be downloaded in here, and it is a shell used to interface with the operating system which follows the UNIX command.

3. Create your local project in your desktop directed towards a current working directory.

pwd stands for 'print working directory', which is used to print the current directory.

4. Initialize the git repository

Use git init to initialize the repository. It is used to create a new empty repository or directory consisting of files' with the hidden directory. '.git' is created at the top level of your project, which places all of the revision information in one place.

5. Add the file to the new local repository.

Use git add . in your bash to add all the files to the given folder.

Use git status in your bash to view all the files which are going to be staged to the first commit.

6. Commit the files staged in your local repository by writing a commit message.

You can create a commit message by git commit -m 'your message', which adds the change to the local repository.

git commit uses '-m' as a flag for a message to set the commits with the content where the full description is included, and a message is written in an imperative sentence up to 50 characters long and defining "what was changed", and "why was the change made".

7. Copy your remote repository's URL from GitHub.

The HTTPS or URL is copied from the given GitHub account, which is the place of the remote repository.

8. Add the URL copied, which is your remote repository to where your local content from your repository is pushed.

git remote add origin 'your\_url\_name'

9. Push the code in your local repository to GitHub

git push -u origin master is used for pushing local content to GitHub.

In the code, the origin is your default remote repository name and '-u' flag is upstream, which is equivalent to '-set-upstream.' and the master is the branch, name.upstream is the repository that we have cloned the project.

Fill in your GitHub username and password.

10. View your files in your repository hosted on GitHub.

You can finally see the file hosted on GitHub.

Similarly, we can use:

git clone which means you are making a copy of the repository in your system.

git fork which means you are copying the repository to your Github account.

git pull which means you are fetching the last modified repository.

git push which means you are returning the repository after modifying it

**THEORETICAL Q/A:**

**Q1) What is JDBC Driver? Please explain in detail.**

**ANS:** JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. **JDBC-ODBC bridge driver--** The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.
2. **Native-API driver (partially java driver)--** The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.
3. **Network Protocol driver (fully java driver)--** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.
4. **Thin driver (fully java driver)--** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

**Q2)** **What are the JDBC API components?**

**ANS:** The JDBC API provides the following interfaces and classes −

* **DriverManager** − This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
* **Driver** − This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
* **Connection** − This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
* **Statement** − You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
* **ResultSet** − These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
* **SQLException** − This class handles any errors that occur in a database application.

**Q3)** **What are the differences between Statement and PreparedStatement interface?**

**ANS: Difference between Statement and PreparedStatement :** 

|  |  |
| --- | --- |
| **Statement** | **PreparedStatement** |
| It is used when SQL query is to be executed only once. | It is used when SQL query is to be executed multiple times. |
| You cannot pass parameters at runtime. | You can pass parameters at runtime. |
| Used for CREATE, ALTER, DROP statements. | Used for the queries which are to be executed multiple times. |
| Performance is very low. | Performance is better than Statement. |
| It is base interface. | It extends statement interface. |
| Used to execute normal SQL queries. | Used to execute dynamic SQL queries. |
| We cannot use statement for reading binary data. | We can use Preparedstatement for reading binary data. |
| It is used for DDL statements. | It is used for any SQL Query. |
| We cannot use statement for writing binary data. | We can use Preparedstatement for writing binary data. |
| No binary protocol is used for communication. | Binary protocol is used for communication. |

**Q4)** **What is the major difference between java.util.Date and java.sql.Date data type?**

**ANS:** java.sql.Date just represent DATE without time information while java.util.Date represents both Date and Timeinformation. This is the major difference why java.util.Date cannot directly map to java.sql.Date.  
  
Q5) **What do you understand by Reflection in Java programming language?**

**ANS:** Reflection is a feature in the Java programming language. It allows an executing Java program to examine or "introspect" upon itself, and manipulate internal properties of theprogram. For example, it's possible for a Java class to obtain the names of all its members and display them.

Q6) **What is Gang of Four (GOF)?**

**ANS:** In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published a book titled **Design Patterns - Elements of Reusable Object-Oriented Software** which initiated the concept of Design Pattern in Software development.

These authors are collectively known as **Gang of Four (GOF)**. According to these authors design patterns are primarily based on the following principles of object orientated design.

* Program to an interface not an implementation
* Favor object composition over inheritance

**Q7)** **What is Factory pattern and Abstract factory pattern?**

# ANS:The Abstract Factory Pattern

* Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
* The Abstract Factory pattern is very similar to the Factory Method pattern. One difference between the two is that with the Abstract Factory pattern, a class delegates the responsibility of object instantiation to another object via composition whereas the Factory Method pattern uses inheritance and relies on a subclass to handle the desired object instantiation.
* Actually, the delegated object frequently uses factory methods to perform the instantiation!
* **Factory pattern**
* Factory patterns are examples of creational patterns
* Creational patterns abstract the object instantiation process. They hide how objects are created and help make the overall system independent of how its objects are created and composed.
* Class creational patterns focus on the use of inheritance to decide the object to be instantiated Factory Method

Object creational patterns focus on the delegation of the instantiation to another object Abstract Factory

Q8) **What is Singleton pattern? How can you create Singleton class in java?**

**ANS:** In Java, **Singleton class** is a class that controls the object creation. It means the **singleton class** allows us to create a single object of the class, at a time. It is usually used to control access to resources, such as database connections or sockets. It ensures that only one connection is made and a thread can access the connection at a time.

To create a singleton class, we must follow the steps, given below:

1. Ensure that only one instance of the class exists.
2. Provide global access to that instance by:
   1. Declaring all constructors of the class to be private.
   2. Providing a static method that returns a reference to the instance. The lazy initialization concept is used to write the static methods.
   3. The instance is stored as a private static variable.

Q9) **What is a version control system (VCS) and what is git repository?**

ANS**:** **Repositories in GIT** contain a collection of files of various different versions of a Project. These files are imported from the repository into the local server of the user for further updations and modifications in the content of the file. A VCS is used to create these versions and store them in a specific place termed as a repository. The process of copying the content from an existing Git Repository with the help of various Git Tools is termed as **cloning**. Once the cloning process is done, the user gets the complete repository on his local machine. Git by default assumes the work to be done on the repository is as a user, once the cloning is done. **Version control systems** are a category of software tools that helps in recording changes made to files by keeping a track of modifications done to the code.

Q10) **Can you explain head in terms of git and also tell the number of heads that can be present in a repository?**

ANS: The HEAD is the commit or branch you are presently viewing. Notice we have used all capital letters to denote this status.You may see “head” written in lowercase. When “head” is written in lowercase, it can refer to any one of the “heads” in a repository. For instance, “master” is a “head” because it is a reference to a branch.If we are viewing the master branch, then “master” is also our HEAD. If we are not viewing the master branch, then whatever branch or commit we are viewing is our HEAD.A repository can contain a number of heads but only one HEAD.This may sound confusing. Let’s summarize HEAD vs. head in a sentence: A HEAD in all caps is a reference or commit in your repository that you are viewing, whereas a “head” with no caps is a head that you are not viewing.