**Java Practical Assessment- 02**

**Theoretical Ques&Ans:**

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

**Ans: JDBC Drivers:**

**Java Database Connectivity (JDBC)** is an application programming interface (API) for the programming language Java, which defines how a client may access any kind of tabular data, especially relational database. It is part of Java Standard Edition platform, from Oracle Corporation. It acts as a middle layer interface between java applications and database, the JDBC classes are contained in the Java Package **java.sql** and **javax.sql**. JDBC helps you to write Java applications that manage these three programming activities:

1. Connect to a data source, like a database.
2. Send queries and update statements to the database
3. Retrieve and process the results received from the database in answer to your query.

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand.

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

**Ans: JDBC includes four components:**

* The JDBC API. The JDBC API gives access of programming data from the Java.
* JDBC **Driver** Manager. The JDBC DriverManager is the **class** in JDBC API.
* JDBC Test Suite.
* JDBC-ODBC Bridge.
* Connection.
* **Class**.forName(String **driver**)
* DriverManager.
* getConnection(String url, String userName, String password).

**Q3. What are the differences between Statement and Prepared Statement interface?**

**Ans:**

**Statement:**

* It is used when SQL query is to be executed only once.
* Used for CREATE, ALTER, DROP statements.
* It is base interface.
* We cannot pass parameters at runtime.
* No binary protocol is used for communication.
* It is used for DDL statements.

**Prepared statement:**

* It extends statement interface.
* Binary protocol is used for communication.
* It is used for any SQL Query.
* It is used when SQL query is to be executed multiple times.
* Used for the queries which are to be executed multiple ties.
* We can used prepared statement for reading binary data.

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

**Ans:**

* As per Javadoc java.sql.Date is a thin wrapper around millisecond value which is used by JDBC to identify an SQL DATE type.
* java.sql.Date just represent DATE without time information while java.util.Date represent both Date and Time information. This is the major differences why java.util.Date cannot directly map to java.sql.Date.
* In order to suppress time information and to confirm with definition of ANSI SQL DATE type, the millisecond values used in java.sql.Date instance must be "normalized by setting the hours, minutes, seconds and milliseconds to zero in the timezone with with DATE instance is associated. In other words all time related information is removed from java.sql.Date class.

**Q5. What do you understand by Reflection in Java programming language?**

**Ans:**

* Reflection is an API which is used to examine or modify the behavior of methods, classes, interfaces at runtime.
* The required classes for reflection are provided under java.lang.reflect package.
* Reflection gives us information about the class to which an object belongs and also the methods of that class which can be executed by using the object.
* Through reflection we can invoke methods at runtime irrespective of the access specifier used with them.

**Reflection can be used to get information about:**

* **Class** The getClass() method is used to get the name of the class to which an object belongs.
* **Constructors** The getConstructors() method is used to get the public constructors of the class to which an object belongs.
* **Methods** The getMethods() method is used to get the public methods of the class to which an objects belongs.

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

**Ans:** 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.

**GOF** Design Patterns are divided into three categories:

1. **Creational**: The design patterns that deal with the creation of an object.
2. **Structural**: The design patterns in this category deals with the class structure such as Inheritance and Composition.
3. **Behavioral**: This type of design patterns provide solution for the better interaction between objects, how to provide lose coupling, and flexibility to extend easily in future.

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

**Ans: Abstract Design Pattern:**

Abstract Factory design pattern is one of the Creational Pattern. Abstract Factory pattern is almost similar to [**Factory Pattern**](https://www.geeksforgeeks.org/design-patterns-set-2-factory-method/) is considered as another layer of abstraction over factory pattern. Abstract Factory patterns work around a super-factory which creates other factories.  
Abstract factory pattern implementation provides us with a framework that allows us to create objects that follow a general pattern. So, at runtime the abstract factory is coupled with any desired concrete factory which can create objects of the desired type.

* **AbstractFactory**: Declares an interface for operations that create abstract product objects.
* **ConcreteFactory**: Implements the operations declared in the AbstractFactory to create concrete product objects.
* **Product**: Defines a product object to be created by the corresponding concrete factory and implements the AbstractProduct interface.
* **Client**: Uses only interfaces declared by AbstractFactory and AbstractProduct classes.

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

**Ans:** In object-oriented programming, a singleton class is a class that can have only one object (an instance of the class) at a time. After first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created.

So, the modifications we do to any variable inside the class through any instance, it affects the variable of the single instance created and is visible if we access that variable through any variable of that class type defined.

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 updating 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.

**Practical Ques & Ans:**

**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", "Mysql#803454");

// Here employeetable is database name, root is the username and Mysql#803454 is the password

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

// SQL statement execution

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", " Mysql#803454");

// Here employeetable is database name, root is the username and Mysql#803454is the password

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", " Mysql#803454");

// Here employeetable is database name, root is the username and Mysql#803454 is the password

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

// SQL statement execution

stmt.execute("insert into employee values(1,'pranay', '8343');");

stmt.execute("insert into employee values(2,'Mr. ram', '8584');");

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", " Mysql#803454");

// Here employeetable is database name, root is the username and Mysql#803454 is the password

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

// SQL statement execution

stmt.executeUpdate("update employee set EmpName='Mr. rajkumar' 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", " Mysql#803454");

// Here employeetable is database name, root is the username and Mysql#803454 is the password

Statement stmt = con.createStatement();

DatabaseMetaData dm = con.getMetaData();

System.out.println(dm);

// SQL statement execution

stmt.execute("delete from employee 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);

}

}

}

**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", " Mysql#803454");

// Here sampledb is database name, root is the username and Mysql#803454 is the password

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(90000);

Courses c2=new Courses();

c2.setCourseName("CSE");

c2.setFees(150000);

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

fees.add(c2);

fees.add(c1);

Student student=new Student();

student.setStudentName("pranay");

student.setStudentContact(837478);

student.setStudentAddress("kadapa");

session.save(c1);

session.save(c2);

session.save(student);

tx.commit(); // Saving Object Permanently ans closing session

factory.close(); // closing very expensive connection

}

}

**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.