Name:Sowmya Vundavalli

Theoretical Questions:

1) What is JDBC Driver? Please explain in brief?

1) JDBC-ODBC bridge driver:

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-discouraged because of thin driver.

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now ers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

easy to use.

can be easily connected to any database.

Disadvantages:

Performance degraded because JDBC method call is converted into the ODBC function calls.

The ODBC driver needs to be installed on the client machine.

2) Native-API 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.

Advantage:

performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

The Native driver needs to be installed on the each client machine.

The Vendor client library needs to be installed on client machine.

3) Network Protocol 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.

Advantage:

No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

Network support is required on client machine.

Requires database-specific coding to be done in the middle tier.

4) Thin 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.

Advantage:

Better performance than all other drivers.

No software is required at client side or server side.

Disadvantage:

Drivers depend on the Database.

2)What are the JDBC APIcomponents?

1. The JDBC API : The JDBC API gives access of programming data from the Java. To use this, applications can execute SQL statements and retrieve results and updation to the database. The JDBC API is part of the Java platform, it includes the Java Standard Edition.

2. JDBC Driver Manager : The JDBC DriverManager is the class in JDBC API. The objects of this class can connect Java applications to a JDBC driver. DriverManager is the very important part of the JDBC architecture.

3. JDBC Test Suite : The JDBC driver test suite helps JDBC drivers to run your program. They are not exhaustive,they do exercise with important features in the JDBC API.

4. JDBC-ODBC Bridge : The Java Software bridge provides JDBC access via ODBC drivers.You have to load ODBC binary code for client machines for using this driver. This driver is very important for application server code has to be in Java in a three-tier architecture.

3)What are the differences between Statement and PreparedStatement interface?

1)Statement

It is used when SQL query is to be executed only once.

You can not pass parameters at runtime

Used for CREATE, ALTER, DROP statements.

Performance is very low

It is base interface

Used to execute normal SQL queries.

We can not used statement for reading binary data.

It is used for DDL statements.

We can not used statement for writing binary data.

No binary protocol is used for communication.

2)PreparedStatement

It is used when SQL query is to be executed multiple times.

You can pass parameters at runtime.

Used for the queries which are to be executed multiple times

It extends statement interface

Used to execute dynamic SQL queries

We can used Preparedstatement for reading binary data.

It is used for any SQL Query

We can used Preparedstatement for writing binary data.

Binary protocol is used for communication.

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

The java.util.date class represents a particular moment in time, with millisecond precision since the 1st of January 1970 00:00:00 GMT (the epoch time). The class is used to keep coordinated universal time (UTC).

The java.sql.Date extends java.util.Date class.

Its main purpose is to represent SQL DATE, which keeps years, months and days. No time data is kept.

In fact, the date is stored as milliseconds since the 1st of January 1970 00:00:00 GMT and the time part is normalized, i.e. set to zero.

Basically, it’s a wrapper around java.util.Date that handles SQL specific requirements. java.sql.Date should be used only when dealing with databases.

5)What do you understand by Reflection in java programming language?

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 the program. For example, it's possible for a Java class to obtain the names of all its members and display them.

The ability to examine and manipulate a Java class from within itself may not sound like very much, but in other programming languages this feature simply doesn't exist. For example, there is no way in a Pascal, C, or C++ program to obtain information about the functions defined within that program.

One tangible use of reflection is in JavaBeans, where software components can be manipulated visually via a builder tool. The tool uses reflection to obtain the properties of Java components (classes) as they are dynamically loaded.

6)What is Gang of Four(GOF)?

1)Creational Design Patterns

Abstract Factory. Allows the creation of objects without specifying their concrete type.

Builder. Uses to create complex objects.

Factory Method. Creates objects without specifying the exact class to create.

Prototype. Creates a new object from an existing object.

Singleton. Ensures only one instance of an object is created.

2)Structural Design Patterns

Adapter. Allows for two incompatible classes to work together by wrapping an interface around one of the existing classes.

Bridge. Decouples an abstraction so two classes can vary independently.

Composite. Takes a group of objects into a single object.

Decorator. Allows for an object’s behavior to be extended dynamically at run time.

Facade. Provides a simple interface to a more complex underlying object.

Flyweight. Reduces the cost of complex object models.

Proxy. Provides a placeholder interface to an underlying object to control access, reduce cost, or reduce complexity.

3)Behavior Design Patterns

Chain of Responsibility. Delegates commands to a chain of processing objects.

Command. Creates objects which encapsulate actions and parameters.

Interpreter. Implements a specialized language.

Iterator. Accesses the elements of an object sequentially without exposing its underlying representation.

Mediator. Allows loose coupling between classes by being the only class that has detailed knowledge of their methods.

7)What is Factory pattern and Abstract factory pattern?

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

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

Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.

The singleton class must provide a global access point to get the instance of the class.

Singleton pattern is used for logging, drivers objects, caching and thread pool.

Singleton design pattern is also used in other design patterns like Abstract Factory, Builder, Prototype, Facade etc.

Singleton design pattern is used in core java classes also, for example java.lang.Runtime, java.awt.Desktop.

Singleton is a creational design pattern, which ensures that only one object of its kind exists and provides a single point of access to it for any other code. You can't just use a class that depends on Singleton in some other context.

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.
4. Eager initialization
5. Static block initialization
6. Lazy Initialization
7. Thread Safe Singleton
8. Bill Pugh Singleton Implementation
9. Using Reflection to destroy Singleton Pattern
10. Enum Singleton
11. Serialization and Singleton

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

A Version Control System (VCS) refers to the method used to save a file's versions for future reference.

Intuitively, many people already version control their projects by renaming different versions of the same file in various ways like blogScript.js, blogScript\_v2.js, blogScript\_v3.js, blogScript\_final.js, blogScript\_definite\_final.js, and so on. But this approach is error-prone and ineffective for team projects.

Also, tracking what changed, who changed it, and why it was changed is a tedious endeavor with this traditional approach. This illuminates the importance of a reliable and collaborative version control system like Git.

However, to get the best of Git, it is essential to understand how Git handles your files.

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 or the Version Control System is used to create these versions and store them in a specific place termed as a repository

Files states in Git

In Git, there are three primary states (conditions) in which a file can be: modified state, staged state, or committed state.

1)Modified state

A file in the modified state is a revised — but uncommitted (unrecorded) — file.

In other words, files in the modified state are files you have modified but have not explicitly instructed Git to monitor.

2)Staged state

Files in the staged state are modified files that have been selected — in their current state (version) — and are being prepared to be saved (committed) into the .git repository during the next commit snapshot.

Once a file gets staged, it implies that you have explicitly authorized Git to monitor that file’s version.

3)Committed state

Files in the committed state are files successfully stored into the .git repository.Thus, a committed file is a file in which you have recorded its staged version into the Git directory (folder).

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

The HEAD is the commit or branch you are presently viewing. ... 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.

**PRACTICAL Q/A:**

**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", "SudebLaltu6@");

// Here employeetable is database name, root is the username and SudebLaltu6@ 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", "SudebLaltu6@");

// Here employeetable is database name, root is the username and SudebLaltu6@ is 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", "SudebLaltu6@");

// Here employeetable is database name, root is the username and SudebLaltu6@ 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,'Sudeb Dolui', '8343');");

stmt.execute("insert into employee values(2,'Mr. Venu Pupalla', '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", "SudebLaltu6@");

// Here employeetable is database name, root is the username and SudebLaltu6@ 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. Sudeb Dolui' 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", "SudebLaltu6@");

// Here employeetable is database name, root is the username and SudebLaltu6@ 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", "SudebLaltu6@");

// Here sampledb is database name, root is the username and SudebLaltu6@ 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(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("Sudeb DOlui");

student.setStudentContact(89194);

student.setStudentAddress("Vizag");

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

.