JAVA ASSIGNMENT SOLUTIONS:

Q1. Please do all the crud operations using JDBC with Table EMPLOYEE

A.

package com.package.jdbc;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class ExampleCRUD\_JDBCOperations {

  public static void main(String[] args) throws SQLException {

    String url ="jdbc:mysql://localhost:3306/TestDatabase"; //update connection string

    String user = "user";//add your db user id here

    String password = "password";//add your db password here

    Connection conn = DriverManager.getConnection(url, user, password);

    System.out.println("Successfully connected");

    //insert employee record into database

    Statement stmt = conn.createStatement();

    int rows = stmt.executeUpdate("insert into employee(age,name) values(23,'James')");

    System.out.println("Rows inserted = "+ rows);

    //update employee record

    rows= stmt.executeUpdate("Update employee set age=31 where name='James'");

    System.out.println("Rows updated = "+ rows);

    //read employee records

    ResultSet rs = stmt.executeQuery("Select \* from employee");

    while(rs.next()){

      System.out.println("Emp Id : " + rs.getInt("id") + ", Name : " + rs.getString("name") + ", Age : " + rs.getInt("age"));

    }

    //delete employee record

    rows = stmt.executeUpdate("delete from employee where name = 'James'");

    System.out.println("Rows deleted = "+ rows);

  }

}

Output:

Successfully connected  
Rows inserted = 1  
Rows updated = 1  
Emp Id : 8, Name : John Doe, Age : 21  
Emp Id : 10, Name : James, Age : 31  
Rows deleted = 1

Q2. Develop a restful webservice 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

Q1. What is JDBC Driver? Please explain in detail

JDBC DRIVER

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

JDBC-ODBC bridge driver

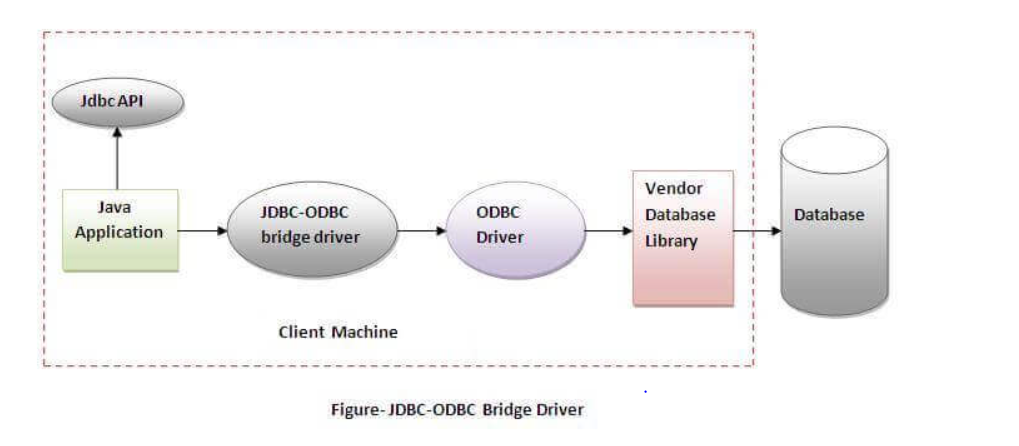
Native-API driver (partially java driver)

Network Protocol driver (fully java driver)

Thin driver (fully java driver)

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.

In Java 8, the JDBC-ODBC bridge has been removed.

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

easy to use.

can be easily connected to any database.

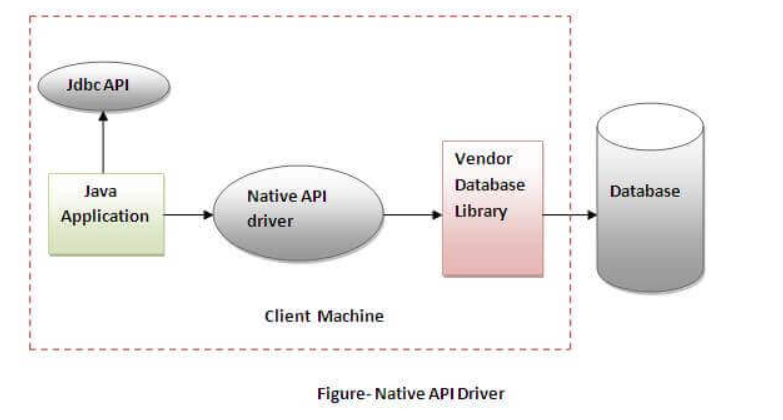
Disadvantages:

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

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

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.

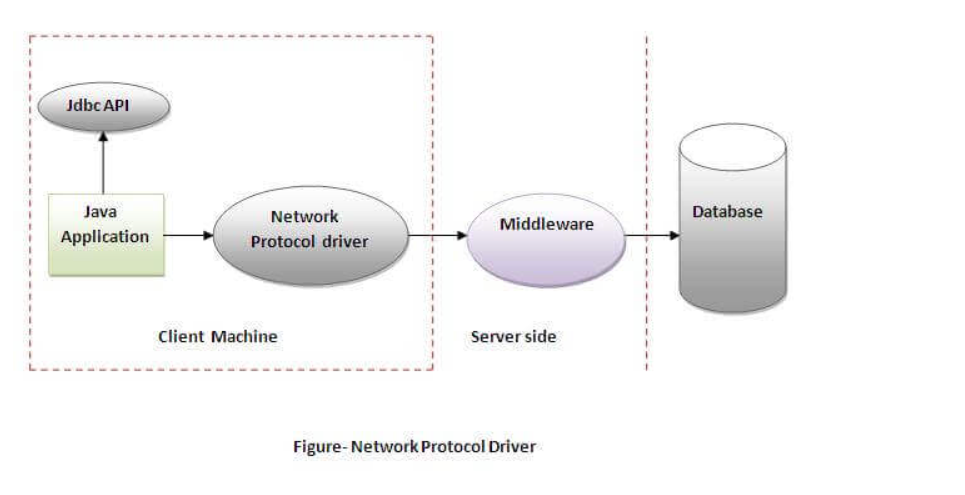
Disadvantage:

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

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

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:

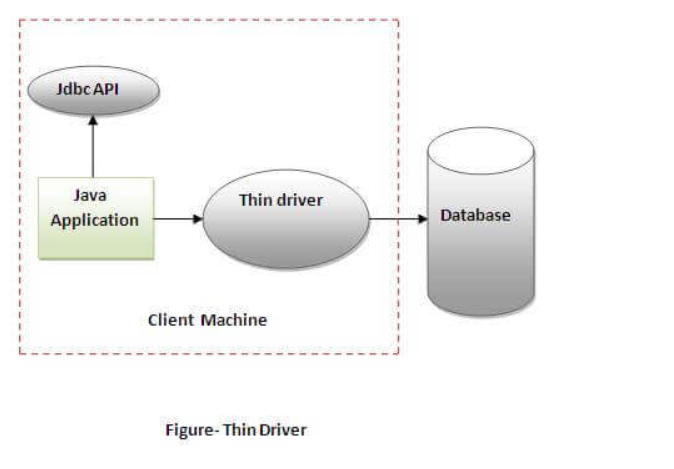
Network support is required on client machine.

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

Maintenance of network protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

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.

Q2. WHAT ARE THE JDBC API COMPONENTS?

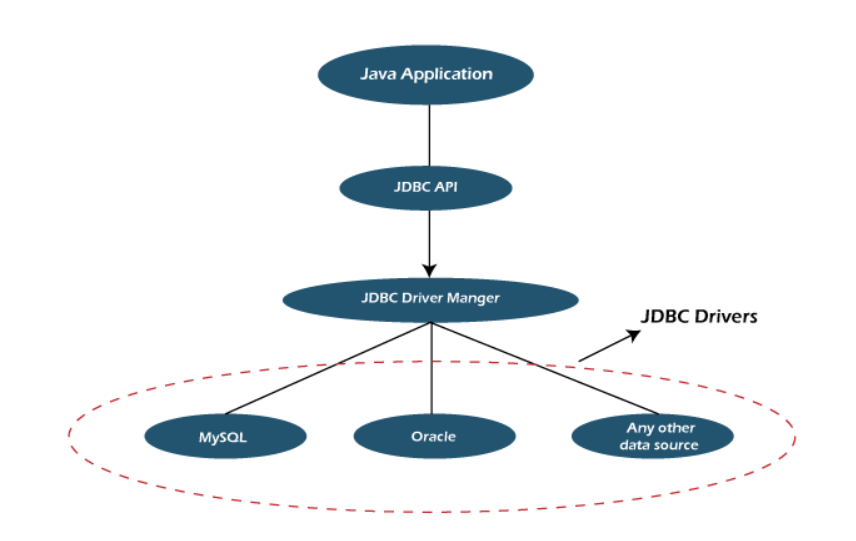
DESIGN OF JDBC:

Java Database Connectivity(JDBC) is an Application Programming Interface(API), from Sun microsystem that is used by the Java application to communicate with the relational databases from different vendors.

JDBC and database drivers work in tandem to access spreadsheets and databases.

Design of JDBC defines the components of JDBC, which is used for connecting to the database.

COMPONENTS OF JDBC:



JDBC has four major components that are used for the interaction with the database.

JDBC API

JDBC Test Suite

JDBC Driver Manager

JDBC ODBC Bridge Driver

JDBC API:

JDBC API provides various interfaces and methods to establish easy connection with different databases.

Javax.sql.\*;

Java.sql.\*;

JDBC Test Suite:

JDBC Test Suite facilitates the programmer various operations such as deletion, updation, insertion that are being executed by the JDBC Drivers.

JDBC Driver Manager:

JDBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database. The JDBC Driver manager is also used to make the database-specific call to the database in order to do the processing of a user request.

JDBC-ODBC Bridge Drivers:

JDBC-ODBC Bridge Drivers are used to connect the database drivers to the database. The bridge does the translation of the JDBC method calls into the ODBC method call. It makes the usage of the sun.jdbc.odbc package that encompasses the native library in order to access the ODBC (Open Database Connectivity) characteristics.

ARCHITECTURE OF JDBC:

Application: It is the Java [servlet](https://www.javatpoint.com/servlet-tutorial) or an applet that communicates with the data source.

The JDBC API:

It allows the java programs to perform the execution of the SQL statements and then get the results.

A few of the crucial interfaces and classes defined in the JDBC API are the following:

Drivers

DriverManager

Statement

Connection

CallableStatement

PreparedStatement

ResultSet

SQL data

Driver Manager:

DriverManager plays a crucial role in the architecture of JDBC.

It Uses database specific drivers to connect the enterprise applications to various databases.

JDBC DRIVERS:

To interact with a data source with the help of the JDBC, one needs a JDBC driver which conveniently interacts with the respective data source.

DIFFERENT TYPES OF ARCHITECTURE OF JDBC:

The architecture of the JDBC consists of two and three tiers model in order to access the given database.

TWO-TIER MODEL:

In this model, the application interacts directly with the source of data. The JDBC driver establishes the interaction between the data source and the application. When a query is sent by the user to the data source, the reply of those sent queries is sent directly to the user.

The source of data can be located on a different machine, and that machine is connected to the user machine following a client-server paradigm, where the machine which is sending the query is the client machine, and the machine that is sending the result of those queries is acting as the server.

**Three-tier model:** In this model, the queries of the user are being sent to the middle-tier services, from where the commands are sent again to the source of data. The answers to those queries are reverted to the middle tier, and from there, it is again sent to the user.

JDBC WORKING:

Any Java application that needs to interact with a database needs to be programmed using the JDBC API. The JDBC driver that supports the data sources like Oracle or MySql needs to be added; then, only the interaction happens with the data source.

**FileName:** JDBCExample.java

// required import statements

import java.sql.\*;

public class JDBCExample

{

// URL for establishing the connection to the database

// TCP port number is 3306

// Name of the database is mydb

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

// Mysql driver class

final static String DB\_DRIVER = "com.mysql.jdbc.Driver";

// Password and User name for using the database

static String uName = "root";

static String psd = "root";

// main method

public static void main(String argvs[])

{

Connection conn = null; // for establishing connection

String query = null; // for storing the queries

Statement sttment = null; // for executing the query

ResultSet resultSet = null; // for storing the response of query

try

{

// Registering the database driver

Class.forName(DB\_DRIVER);

System.out.println("Database connection established");

// Creating a connection to the database

conn = DriverManager.getConnection(DB\_URL, uName, psd);

// the query be executed

// EmployeeId, EmployeeName, Department are the fields or column names

// mytable is the table name available in the mydb database

query = "select EmployeeId, EmployeeName, Department from mytable";

// for query execution

sttment = conn.createStatement();

// the query is executed and the result is stored

resultSet = sttment.executeQuery(query);

while(resultSet.next() )

{

// receiving the results using the table column name

int eId = resultSet.getInt("EmployeeId");

String eName = resultSet.getString("EmployeeName");

String department = resultSet.getString("Department");

// printing the values

System.out.print("Employee ID: " + eId);

System.out.print(", Employee Name: " + eName);

System.out.println(", Department: " + department);

}

}

catch(SQLException sqlExp)

{

// For handling the exception raised from JDBC

sqlExp.printStackTrace();

}

catch(Exception e)

{

// For handing the issues raised from Class.forName

e.printStackTrace();

}

finally

{

try

{

// performing the clean-up work

// terminating the connection

resultSet.close();

sttment.close();

conn.close();

System.out.println("The Connection is closed.");

}

catch(SQLException sqlExp)

{

sqlExp.printStackTrace();

}

}

}

}

OUTPUT:

Database connection established

Employee ID: 100, Employee Name: Nitesh Singh, Department: Project Management

Employee ID: 104, Employee Name: Amit Kumar, Department: Game Development

Employee ID: 105, Employee Name: Amrit Kumar, Department: Database Management

Employee ID: 109, Employee Name: Rohit Kumar, Department: Software Testing

Employee ID: 120, Employee Name: Ajeet Chouhan, Department: Software Design

Employee ID: 155, Employee Name: Aman Jatt, Department: Art Integration

The Connection is closed.

EXPLANATION:

The above Java application connects to the MYSQL Database System. Therefore, we need the driver for MySQL in order to access the data. That driver (com.mysql.jdbc.Driver) is provided in the  mysqlconnector.jar file, which must be included in the classpath when the above program is executed. Similarly, if instead of MySQL had we used Oracle, then the drivers corresponding to Oracle must be used.

Q3. WHAT ARE THE DIFFERENCES BETWEEN STATEMENT AND PREPAREDSTATEMENT INTERFACE?

A.

The Java Database Connectivity (JDBC) API is essentially used to connect Java applications with databases. It could be any relational or OLAP database.

The JDBC API offers different interfaces to connect to particular databases and execute numerous types of SQL queries such as follows:

Statement

PreparedStatement

These JDBC interfaces offers different functionalities, properties and methods which enables us to connect to databases and execute SQL or PL/SQL commands.

Difference between Statement and PreparedStatement in JAVA:

|  |  |  |
| --- | --- | --- |
| SNo. | Statement | PreparedStatement |
| 1. | This JDBC API interface is used for static SQL statements at run time. | The PreparedStatement interface is used for dynamic SQL statements at run time. |
| 2. | There is no specific protocol in Statement interface. | The PreparedStatement used the non sql binary protocol |
| 3. | The Statement interface does not allow accepting parameters at runtime. | The PreparedStatement interface allow accepting parameters at runtime. |
| 4. | This interface is used when the SQL query is required to be executed only once. | This interface is used when the SQL query is required to be executed multiple times. |
| 5. | The performance of the Statement interface is comparatively not upto the mark. | The performance of the PreparedStatement interface is better than Statement. |
| 6. | The Statement interface enforces SQL injection. | The PreparedStatement does not enforce SQL injection. |
| 7. | The Statement interface does not extends the PreparedStatement interface. | The PreparedStatement interface extends the Statement interface. |
| 8. | There could be a possibility of writing concatenated SQL statements while using the Statement interface. | There’s no need of writing concatenated SQL statements when using the PreparedStatement interface. |
| 9. | The SQL queries that are executed using Statement interface are executed at runtime, and therefore, it is a little slower performance wise. | The SQL queries executed using PreparedStatement are pre compiled and therefore, it offers a better performance. |
| 10. | The statement interface cannot be used for retrieving or storing image and files in the databases. | The PreparedStatement interface can be used for retrieving or storing image and files in the databases. |
| 11. | This interface does not offer using setArray method in Java. | This interface offers using setArray method in Java. |
| 12. | The statement interface is beneficial when using Data Definition Language (DDL) commands. | The PreparedStatement interface is beneficial when using Data Manipulation Language (DML) commands. |
| 13. | The commands that are mostly used in this interface are create, drop, truncate and alter. | The commands that are mostly used in this interface are select, delete, update and insert. |

Q4. WHAT IS THE MAJOR DIFFERENCE BETWEEN JAVA.UTIL.DATE AND JAVA.SQL.DATE DATA TYPE?

Java.util.Date: It just represents both date and time information. This is the major difference.

Java.sql.Date: Its just represents Date without time information.

Q5. WHAT DO YOU UNDERSTAND BY REFLECTION IN JAVA PROGRAMMING LANGUAGE?

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Reflection in java is an API which is used to examine or modify the behaviour of methods, classes, interfaces at run time, 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.

Q6.WHAT IS GANG OF FOUR(GOF)?

The authors Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides are often referred to as the GoF, or **Gang** of **Four**. They are the authors of the book Design Patterns: Elements of Reusable Object-Oriented Software

Q7. WHAT IS FACTORY PATTERN AND ABSTRACT FACTORY PATTERN?

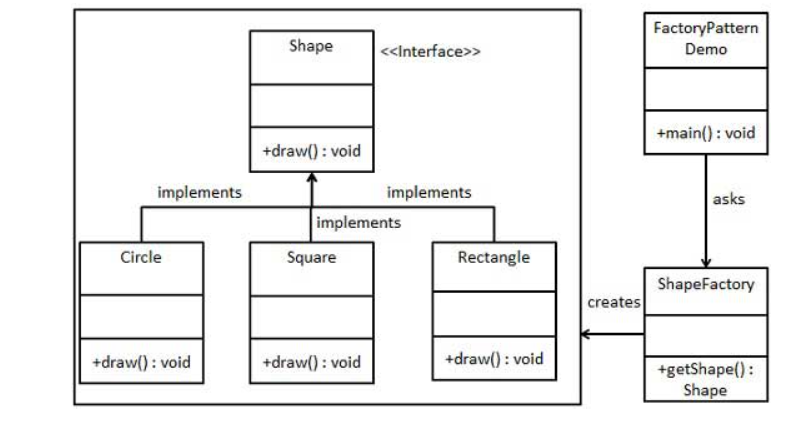
A. Factory pattern is one of the most used design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Implementation

We're going to create a *Shape* interface and concrete classes implementing the *Shape* interface. A factory class *ShapeFactory* is defined as a next step.

*FactoryPatternDemo*, our demo class will use *ShapeFactory* to get a *Shape* object. It will pass information (*CIRCLE / RECTANGLE / SQUARE*) to *ShapeFactory* to get the type of object it needs



STEP 1:

Create an interface.

*Shape.java*

public interface Shape {

void draw();

}

STEP 2:

Create concrete classes implementing the same interface.

Rectangle.java

public class Rectangle implements Shape {

@Override

public void draw() {

System.out.println("Inside Rectangle::draw() method.");

}

}

Square.java

public class Square implements Shape {

@Override

public void draw() {

System.out.println("Inside Square::draw() method.");

}

}

Circle.java

public class Circle implements Shape {

@Override

public void draw() {

System.out.println("Inside Circle::draw() method.");

}

}

STEP 3:

Create a Factory to generate object of concrete class based on given information.

ShapeFactory.java

public class ShapeFactory {

//use getShape method to get object of type shape

public Shape getShape(String shapeType){

if(shapeType == null){

return null;

}

if(shapeType.equalsIgnoreCase("CIRCLE")){

return new Circle();

} else if(shapeType.equalsIgnoreCase("RECTANGLE")){

return new Rectangle();

} else if(shapeType.equalsIgnoreCase("SQUARE")){

return new Square();

}

return null;

}

}

STEP 4:

Use the Factory to get object of concrete class by passing an information such as type.

FactoryPatternDemo.java

public class FactoryPatternDemo {

public static void main(String[] args) {

ShapeFactory shapeFactory = new ShapeFactory();

//get an object of Circle and call its draw method.

Shape shape1 = shapeFactory.getShape("CIRCLE");

//call draw method of Circle

shape1.draw();

//get an object of Rectangle and call its draw method.

Shape shape2 = shapeFactory.getShape("RECTANGLE");

//call draw method of Rectangle

shape2.draw();

//get an object of Square and call its draw method.

Shape shape3 = shapeFactory.getShape("SQUARE");

//call draw method of square

shape3.draw();

}

}

STEP 5:

Verify the output.

Inside Circle::draw() method.

Inside Rectangle::draw() method.

Inside Square::draw() method.

ABSTRACT FACTORY METHOD:

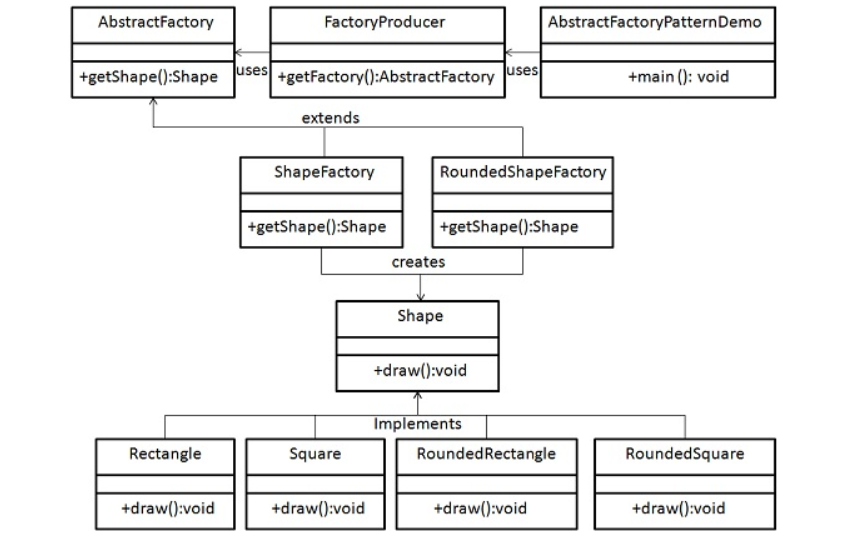
Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.

Implementation

We are going to create a Shape interface and a concrete class implementing it. We create an abstract factory class AbstractFactory as next step. Factory class ShapeFactory is defined, which extends AbstractFactory. A factory creator/generator class FactoryProducer is created.

AbstractFactoryPatternDemo, our demo class uses FactoryProducer to get a AbstractFactory object. It will pass information (CIRCLE / RECTANGLE / SQUARE for Shape) to AbstractFactory to get the type of object it needs.



Step 1

Create an interface for Shapes.

Shape.java

public interface Shape {

void draw();

}

Step 2

Create concrete classes implementing the same interface.

RoundedRectangle.java

public class RoundedRectangle implements Shape {

@Override

public void draw() {

System.out.println("Inside RoundedRectangle::draw() method.");

}

}

RoundedSquare.java

public class RoundedSquare implements Shape {

@Override

public void draw() {

System.out.println("Inside RoundedSquare::draw() method.");

}

}

Rectangle.java

public class Rectangle implements Shape {

@Override

public void draw() {

System.out.println("Inside Rectangle::draw() method.");

}

}

Step 3

Create an Abstract class to get factories for Normal and Rounded Shape Objects.

AbstractFactory.java

public abstract class AbstractFactory {

abstract Shape getShape(String shapeType) ;

}

Step 4

Create Factory classes extending AbstractFactory to generate object of concrete class based on given information.

ShapeFactory.java

public class ShapeFactory extends AbstractFactory {

@Override

public Shape getShape(String shapeType){

if(shapeType.equalsIgnoreCase("RECTANGLE")){

return new Rectangle();

}else if(shapeType.equalsIgnoreCase("SQUARE")){

return new Square();

}

return null;

}

}

RoundedShapeFactory.java

public class RoundedShapeFactory extends AbstractFactory {

@Override

public Shape getShape(String shapeType){

if(shapeType.equalsIgnoreCase("RECTANGLE")){

return new RoundedRectangle();

}else if(shapeType.equalsIgnoreCase("SQUARE")){

return new RoundedSquare();

}

return null;

}

}

Step 5

Create a Factory generator/producer class to get factories by passing an information such as Shape

FactoryProducer.java

public class FactoryProducer {

public static AbstractFactory getFactory(boolean rounded){

if(rounded){

return new RoundedShapeFactory();

}else{

return new ShapeFactory();

}

}

}

Step 6

Use the FactoryProducer to get AbstractFactory in order to get factories of concrete classes by passing an information such as type.

AbstractFactoryPatternDemo.java

public class AbstractFactoryPatternDemo {

public static void main(String[] args) {

//get shape factory

AbstractFactory shapeFactory = FactoryProducer.getFactory(false);

//get an object of Shape Rectangle

Shape shape1 = shapeFactory.getShape("RECTANGLE");

//call draw method of Shape Rectangle

shape1.draw();

//get an object of Shape Square

Shape shape2 = shapeFactory.getShape("SQUARE");

//call draw method of Shape Square

shape2.draw();

//get shape factory

AbstractFactory shapeFactory1 = FactoryProducer.getFactory(true);

//get an object of Shape Rectangle

Shape shape3 = shapeFactory1.getShape("RECTANGLE");

//call draw method of Shape Rectangle

shape3.draw();

//get an object of Shape Square

Shape shape4 = shapeFactory1.getShape("SQUARE");

//call draw method of Shape Square

shape4.draw();

}

}

Step 7

Verify the output.

Inside Rectangle::draw() method.

Inside Square::draw() method.

Inside RoundedRectangle::draw() method.

Inside RoundedSquare::draw() method.

Q8. WHAT IS SINGLETON PATTERN? HOW CAN YOU CREATE SINGLETON CLASS IN JAVA?

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.

Nevertheless, if we try to create the second object of the singleton class, it points to the first instance of the singleton class.

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:

i.Declaring all constructors of the class to be private.

ii.Providing a static method that returns a reference to the instance. The lazy initialization concept is used to write the static methods.

iii.The instance is stored as a private static variable.

The example of singleton classes is Runtime class, Action Servlet, Service Locator. Private constructors and factory methods are also an example of the singleton class.

Factory method

By using the class name if we are calling a method and that method returns the same class object such type of method is called a factory method.

For example:

Runtime rt=Runtime.getRuntime();

DateFormat df=DateFormat.getInstance();

The above method returns an object of that class's type.

Difference Between Singleton Class and Normal Class

The main difference between these two classes is an instantiation. To create an instance of a normal class, we use a constructor. On the other hand, to create an instance of a singleton class, we use getInstance() method.

Generally, we use the class name as the method name. It avoids confusion.

Advantages

Singleton controls concurrent access to the resource.

It ensures there is only one object available across the application in a controlled state.

Example of Singleton Class

Let's create a singleton class.

SingletonClass

Example.java

class SingletonClassExample

{

// static variable s of type Singleton

private static Singleton s = null;

// variable of type String

public String str;

//private constructor of the Singleton class that restricted to this class itself

private Singleton()

{

str = "it is an example of singleton class.";

}

//static method to create an instance of the Singleton class

// we can also create a method with the same name as the class name

public static Singleton getInstance()

{

//lazy initialization

if (s== null)

s = new SingletonClassExample();

return s;

}

}

Let's create the main class with the name Test.

Test.java

class Test

{

public static void main(String args[])

{

// instantiating Singleton class with variable a

SingletonClassExample a = SingletonClassExample.getInstance();

// instantiating Singleton class with variable b

SingletonClassExample b = SingletonClassExample.getInstance();

// instantiating Singleton class with variable c

SingletonClassExample c = SingletonClassExample.getInstance();

// changing variable of instance a

a.str = (a.str).toUpperCase();

System.out.println("String from a is " + a.str);

System.out.println("String from b is " + b.str);

System.out.println("String from c is " + c.str);

}

}

Output:

IT IS AN EXAMPLE OF SINGLETON CLASS.

IT IS AN EXAMPLE OF SINGLETON CLASS.

IT IS AN EXAMPLE OF SINGLETON CLASS.

Explanation:

Have you noticed the concept of instantiation in the SingletonClassExample class? No, let's understand the working. Whenever we call getInstance() method, the first time, it creates an object of the class with the name s and returns it the variable. The variable s changed from null to some object because it is a static variable. The second time when we call the getInstance() method it returns the same variable s, instead of creating the new one object.

In the main class Test, we have created three instances (a, b, and c) of the Singleton class by invoking the getInstance() method. Here, a point should be noted that objects b and c are not formed in real, instead, both objects point to the previously created object i.e. a.

Another point to be noted that if we perform any modification to the object (such as variables) a, it also reflects the objects b and c. On the other hand, if we change the variable of object c, it also reflects changes in other objects.

Q9. WHAT IS A VERSION CONTROL SYSTEM(VCS) AND WHAT IS GIT REPOSITORY?

Git Version Control System

A version control system is a software that tracks changes to a file or set of files over time so that you can recall specific versions later. It also allows you to work together with other programmers.

The version control system is a collection of software tools that help a team to manage changes in a source code. It uses a special kind of database to keep track of every modification to the code.

Developers can compare earlier versions of the code with an older version to fix the mistakes.

Benefits of the Version Control System

The Version Control System is very helpful and beneficial in software development; developing software without using version control is unsafe. It provides backups for uncertainty. Version control systems offer a speedy interface to developers. It also allows software teams to preserve efficiency and agility according to the team scales to include more developers.

Some key benefits of having a version control system are as follows.

Complete change history of the file

Simultaneously working

Branching and merging

Traceability

Git Repository

In Git, the repository is like a data structure used by VCS to store metadata for a set of files and directories. It contains the collection of the files as well as the history of changes made to those files. Repository in Git is considered as your project folder. A repository has all the project-related data. Distinct projects have distinct repositories.

Getting a Git Repository

There are two ways to obtain a repository. They are as follows:

Create a local repository and make it as Git repository.

Clone a remote repository (already exists on a server).

In either case, you can start working on a Git repository.

Initializing a Repository

If you want to share your project on a version control system and control it with Git. Then, browse your project's directory and start the git command line (Git Bash for Windows) here. To initialize a new repository, run the below command:

23M

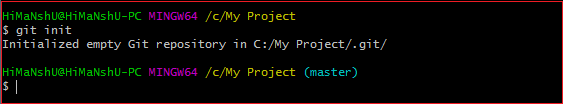
547

OOPs Concepts in Java

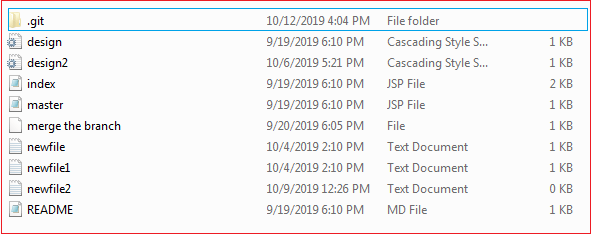
**Syntax:**

$ git init

**Output:**



The above command will create a new subdirectory named .git that holds all necessary repository files. The **.git** subdirectory can be understood as a Git repository skeleton. Consider the below image:

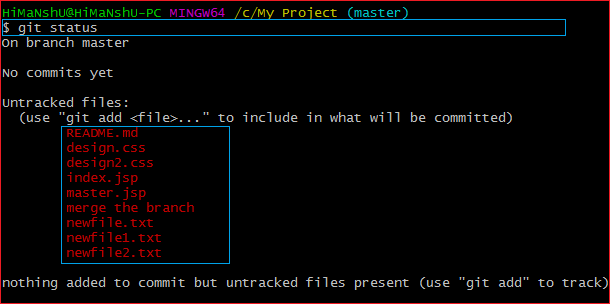


An empty repository .git is added to my existing project. If we want to start version-controlling for existing files, we should track these files with git add command, followed by a commit.

We can list all the untracked files by git status command.

$ git status

Consider the below output:



In the above output, the list of all untracked files is displayed by the git status command. To share these files on the version control system, we have to track it with git add command followed by a commit. To track the files, operate git add command as follows:

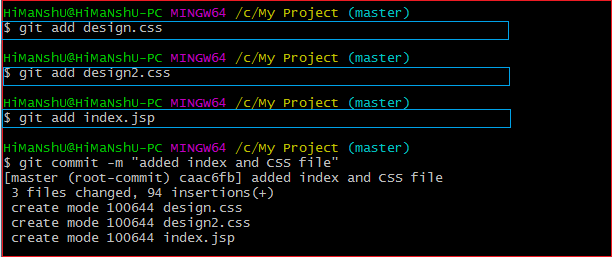
**Syntax:**

$ git add **<filename>**

To commit a file, perform the git commit command as follows:

$ git commit -m "Commit message."

**Output:**



In the above output, I have added three of my existing files by git add command and commit it for sharing.

We can also create new files. To share the new file, follow the same procedure as described above; add and commit it for sharing. Now, you have a repository to share.

Cloning an Existing Repository

We can clone an existing repository. Suppose we have a repository on a version control system like subversion, GitHub, or any other remote server, and we want to share it with someone to contribute. The git clone command will make a copy for any user to contribute.

We can get nearly all data from server with git clone command. It can be done as:

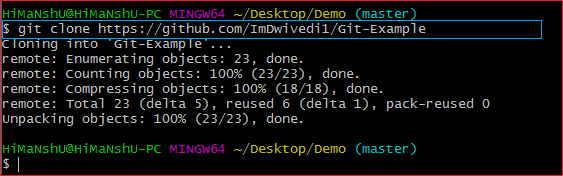
**Syntax:**

$ git clone **<Repository** URL**>**

Suppose one of my friends has a repository on my GitHub account, and I want to contribute to it. So the first thing I will do, make a copy of this project to my local system for a better work interface. The essential element needed for cloning the repository URL. I have a repository URL "<https://github.com/ImDwivedi1/Git-Example>". To clone this repository, operate the clone command as:

$ git clone https://github.com/ImDwivedi1/Git-Example

Consider the below output:



In the above output, the repository Git-Example has been cloned. Now this repository is available on your local storage. You can commit it and contribute to the project by pushing it on a remote server.

A single repository can be cloned any number of times. So we can clone a repository on various locations and various systems.

Q10. CAN YOU EXPLAIN HEAD IN TERMS OF GIT AND ALSO TELL THE NUMBER OF HEADS THAT CAN BE PRESENT IN A REPOSITORY?

A head is nothing but a reference to the last commit object of a branch.

For every repository, there will always be a default head referred to as “master” or now “main” (as per GitHub) but there is no restriction to the count of heads available. In other words, it can have any number of heads.

**Usages:**  
  
- To go or checkout to 1 commit before the latest commit, we use git checkout HEAD~1  
  
- To uncommit the last 3 commits without losing the changes, we first run git reset HEAD~3. Then we can see the changes made in the last 3 commits and then update it manually and commit it finally.  
  
- In order to uncommit the last 3 commits and also remove the changes, we can run the command: git reset --hard HEAD~3. This command will completely remove all the changes.  
  
- To look into the changes made in the last 3 commits, we can run git diff HEAD~3  
  
- To make a new commit by reverting the last 3 commits, we can run the command: git revert --no-commit HEAD~3...HEAD