1. **Write a JDBC connection program with following details**

   (a) Drive: com.mysql.jdbc.Driver

   (b) Host: localhost:

   (c) Database: students

   (d) User: root

   (e) Password: secret

   (f) Post: 3306

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBCExample {
  public static void main(String[] args) {
    // JDBC URL for MySQL database
    String url = "jdbc:mysql://localhost:3306/students";

    // Database credentials
    String user = "root";
    String password = "secret";

    // Connection object
    Connection conn = null;

    try {
      // Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      // Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(url, user, password);
```

```
            // Display connection successful message
            System.out.println("Connected to the database.");

            // Perform database operations here

        } catch (SQLException e) {
            // Handle errors for JDBC
            e.printStackTrace();
        } catch (Exception e) {
            // Handle errors for Class.forName
            e.printStackTrace();
        } finally {
            // Finally block to close resources
            try {
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

2. **Write an applet program to calculate x^y where both x and y are integers**

//Applet Program

```
import java.applet.Applet;
import java.awt.Graphics;

public class PowerCalculator extends Applet {
    int base = 2; // Default base value
    int exponent = 3; // Default exponent value
    long result;

    public void init() {
        // You can set the values of base and exponent here if needed
        // base = 2;
        // exponent = 3;
        result = power(base, exponent);
    }

    public void paint(Graphics g) {
        String baseStr = String.valueOf(base);
        String exponentStr = String.valueOf(exponent);
        String resultStr = String.valueOf(result);

        g.drawString(baseStr + " ^ " + exponentStr + " = " + resultStr, 20, 20);
    }

    private long power(int base, int exponent) {
        long result = 1;
        for (int i = 0; i < Math.abs(exponent); i++) {
            result *= base;
        }
        return exponent < 0 ? 1 / result : result;
    }
}
//html code
<!DOCTYPE html>
<html>
```

```
<head>
   <title>Power Calculator Applet</title>
</head>
<body>
   <applet code="PowerCalculator.class" width="300" height="100">
      <!-- This message is displayed if the applet cannot be loaded -->
      Your browser does not support Java applets.
   </applet>
</body>
</html>
```

3. **What is JDBC? Explain different types of JDBC drivers. Write a JDBC Program to display student details from Student table.**

Java Database Connectivity (JDBC) is an API (Application Programming Interface) that enables Java programs to interact with databases. It provides a standard interface for accessing relational databases from Java applications. JDBC allows developers to execute SQL statements, retrieve results, and handle database transactions programmatically.

There are four types of JDBC drivers:

JDBC-ODBC Bridge Driver: This driver uses the ODBC (Open Database Connectivity) API provided by the operating system to connect to the database. It acts as a bridge between JDBC and ODBC. Java applications communicate with the ODBC driver, which in turn communicates with the database. This type of driver is platform-dependent and requires the ODBC driver to be installed on the client machine.

Native-API Driver: This driver converts JDBC calls into calls specific to the database API. It interacts directly with the database through a native library provided by the database vendor. This type of driver is partially platform-dependent, as it relies on native code.

Network Protocol Driver (Middleware Driver): This driver uses a middle-tier server to communicate with the database. The client-side JDBC driver communicates with a middleware server using a protocol like TCP/IP, which in turn communicates with the database using a database-specific API. This type of driver is platform-independent and requires the middleware server to be installed.

Thin Driver (Direct-to-Database Pure Java Driver): This driver communicates directly with the database using a protocol specific to the database. It does not require any additional software to be installed on the client or server side, as it is implemented entirely in Java. This type of driver is platform-independent and provides better performance compared to other types.

```
import java.sql.*;


public class StudentDetails {
   // JDBC URL for MySQL database
   static final String JDBC_URL = "jdbc:mysql://localhost:3306/students";
   static final String USER = "root";
   static final String PASSWORD = "secret";


   public static void main(String[] args) {
      Connection conn = null;
      Statement stmt = null;
      ResultSet rs = null;


      try {
         // Register JDBC driver
         Class.forName("com.mysql.jdbc.Driver");


         // Open a connection
         conn = DriverManager.getConnection(JDBC_URL, USER, PASSWORD);


         // Execute a query
         stmt = conn.createStatement();
         String sql = "SELECT * FROM Student";
         rs = stmt.executeQuery(sql);


         // Display student details
         while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                String address = rs.getString("address");
                int age = rs.getInt("age");


                System.out.println("ID: " + id + ", Name: " + name + ", Address: " + address + ",
Age: " + age);
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            // Close resources
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

4. **What is JDBC and ODBC? Write a java Program using JDBC to extract name of those students who live in Morang district, assuming that students table has four attributes(ID, name, district, and age).**

JDBC (Java Database Connectivity) and ODBC (Open Database Connectivity) are both APIs (Application Programming Interfaces) that provide a standard way for applications to interact with databases.

JDBC (Java Database Connectivity): JDBC is a Java API that enables Java programs to interact with databases. It provides classes and interfaces to connect to a database, send SQL queries, retrieve results, and manage database transactions. JDBC drivers are used to connect Java applications to different types of databases.

ODBC (Open Database Connectivity): ODBC is a C-based API that provides a standard interface for accessing databases. It allows applications to access databases using SQL queries regardless of the database management system (DBMS) being used. ODBC drivers act as intermediaries between applications and databases, translating ODBC calls into native calls understood by the database.

```
import java.sql.*;


public class StudentExtractor {
    // JDBC URL for MySQL database
    static final String JDBC_URL = "jdbc:mysql://localhost:3306/students";
    static final String USER = "root";
    static final String PASSWORD = "secret";


    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;


        try {
            // Register JDBC driver
            Class.forName("com.mysql.jdbc.Driver");


            // Open a connection
            conn = DriverManager.getConnection(JDBC_URL, USER, PASSWORD);


            // Execute a query
            stmt = conn.createStatement();
            String sql = "SELECT name FROM students WHERE district='Morang'";
            rs = stmt.executeQuery(sql);


            // Display names of students in Morang district
            System.out.println("Names of students living in Morang district:");
            while (rs.next()) {
                String name = rs.getString("name");
                System.out.println(name);
            }
        } catch (SQLException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            // Close resources
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

5. **Describe the function of File class? Create a DataInputStream for a file name "purbanchal.dat" and store "I am student of BIT VI semester " in that file.**

The File class in Java provides functionalities for working with files and directories. It represents the properties and behaviors of files and directories on the system. Some of the functions of the File class include:

Creating, Deleting, and Renaming Files and Directories: The File class allows you to create new files and directories, delete existing files and directories, and rename files and directories.

Checking File and Directory Existence: You can use the exists() method to check if a file or directory exists.

Getting File and Directory Information: You can retrieve information such as file or directory name, absolute path, parent directory, file size, last modified timestamp, etc., using various methods provided by the File class.

Listing Files and Directories: You can list the contents of a directory using the list() or listFiles() methods.

File System Navigation: The File class allows you to navigate through the file system by representing files and directories as objects and manipulating them accordingly.

```
import java.io.*;

public class DataInputExample {
    public static void main(String[] args) {
        // Define the file name
        String fileName = "purbanchal.dat";

        try {
            // Create a FileOutputStream to write data to the file
            FileOutputStream fos = new FileOutputStream(fileName);
            // Create a DataOutputStream to write primitive data types to the file
            DataOutputStream dos = new DataOutputStream(fos);

            // Write the string to the file
            dos.writeUTF("I am a student of BIT VI semester");

            // Close the DataOutputStream and FileOutputStream
            dos.close();
            fos.close();

            System.out.println("Data has been written to " + fileName);
        } catch (IOException e) {
            e.printStackTrace();
            }
        }
    }
```

6. **Write a program in java that reads line of text from keyboard and write to file. Also read the content of the same file and display on monitor.**

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;

public class FileReadWriteExample {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        // File name
        String fileName = "textfile.txt";

        // Write to file
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName))) {
            System.out.println("Enter a line of text (Press Enter to finish):");
            String line;
            while (!(line = reader.readLine()).isEmpty()) {
                writer.write(line);
                writer.newLine();
            }
            System.out.println("Text has been written to " + fileName);
        } catch (IOException e) {
            e.printStackTrace();
        }
        // Read from file and display on monitor
        try (BufferedReader fileReader = new BufferedReader(new FileReader(fileName))) {
            System.out.println("\nContent of " + fileName + ":");
            String line;
            while ((line = fileReader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

7. Which classes are used to read and write data to file? Write a simple code to write a word to file.

FileInputStream and FileOutputStream: These classes are used for reading and writing raw bytes from and to files, respectively. They are typically used for binary file I/O operations.

FileReader and FileWriter: These classes are used for reading and writing text data from and to files, respectively. They handle characters and are typically used for text file I/O operations.

BufferedReader and BufferedWriter: These classes are used for efficient reading and writing of text data by buffering the input and output streams, respectively. They provide better performance compared to reading and writing data directly from/to files.

```
import java.io.FileWriter;
import java.io.IOException;

public class WriteToFileExample {
    public static void main(String[] args) {
        String fileName = "output.txt"; // File name
        try (FileWriter writer = new FileWriter(fileName)) {
            String word = "Hello"; // Word to write to the file
            writer.write(word);
            System.out.println("Word \"" + word + "\" has been written to " + fileName);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

8. Chat Application in java

```
// Server Program
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) {
        try {
            //creating server socket
            ServerSocket serverSocket = new ServerSocket(9999);

            //wait for client Connection
            System.out.println("Waiting for Client Connection........");

            Socket clientSocket = serverSocket.accept();
            System.out.println("Client Connected...........");

            //create input and output stream
            BufferedReader inFromClient = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter outToClient = new PrintWriter(clientSocket.getOutputStream(),true);
}

            //Communication loop
            String clientMessage;
            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

            while (true) {
                //Read message from client
                clientMessage = inFromClient.readLine();
                if (clientMessage.equals("exit")) {
                    System.out.println("Client Disconnected.................");
                    break;
                }
                System.out.println("Client: " + clientMessage);

                //send response to the client
                System.out.println("Server: ");
                String response = reader.readLine();
                outToClient.println(response);
            }

            inFromClient.close();
            outToClient.close();
            serverSocket.close();
            clientSocket.close();
        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }
    }
}
```

```
//Client Program

import java.io.*;
import java.net.*;
public class Client {
   public static void main(String[] args) {
      try {
         //creating a client socket
         Socket clientSocket = new Socket("localhost", 9999);

         //create a input output Stream
         BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

         PrintWriter outToServer = new PrintWriter(clientSocket.getOutputStream(), true);

         String serverMessage;
         BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

         while (true) {
            //sende message to server
            System.out.println("Client: ");
            String message = reader.readLine();
            outToServer.println(message);

            if (message.equals("exit")) {
               System.out.println("Disconnected from server.................");
               break;
            }
            //receiving response
            serverMessage = inFromServer.readLine();
```

```
                System.out.println("Server: " + serverMessage);
            }

            inFromServer.close();
            outToServer.close();
            clientSocket.close();


        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }
    }
}
```