1. Create a GUI with a button. When the button is clicked, display "Button Clicked” in a JLabel.

This simple Swing application creates a GUI with a JButton and a JLabel. When the button is clicked, the label displays the message "Button Clicked!".

import javax.swing.\*;

import java.awt.event.ActionEvent;

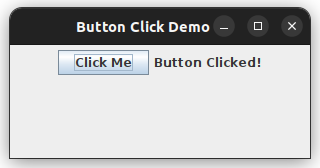
import java.awt.event.ActionListener;

import java.awt.FlowLayout;

public class ButtonClickDemo {

public static void main(String[] args) {  
 // Create the main JFrame container  
 JFrame frame = new JFrame("Button Click Demo");  
  
 // Create a JLabel to display the message  
 JLabel label = new JLabel("Click the button!");  
  
 // Create a JButton  
 JButton button = new JButton("Click Me");  
  
 // Add an ActionListener to the button  
 button.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 label.setText("Button Clicked!"); // Change the label text when button is clicked  
 }  
 });  
  
 // Set the layout and add components  
 frame.setLayout(new FlowLayout());  
 frame.add(button);  
 frame.add(label);  
  
 // Set frame properties  
 frame.setSize(300, 150); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}

}



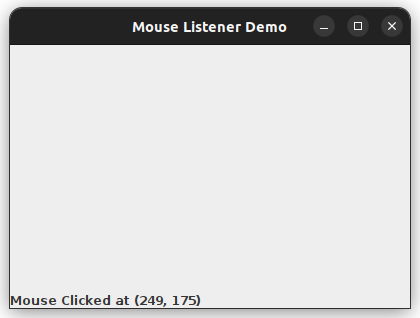
1. Implement a mouse listener using an adapter class to handle mouse events like mouseClicked() and mouseEntered().

This Swing application demonstrates the use of a MouseAdapter to handle mouse events such as mouseClicked() and mouseEntered(). It updates a JLabel to display messages when the mouse interacts with a JPanel.

import javax.swing.\*; import java.awt.event.MouseAdapter; import java.awt.BorderLayout; import java.awt.event.MouseEvent;

public class MouseListenerDemo {

public static void main(String[] args) {  
 // Create the main JFrame container  
 JFrame frame = new JFrame("Mouse Listener Demo");  
  
 // Create a JLabel to display messages  
 JLabel label = new JLabel("Interact with the panel!");  
  
 // Create a JPanel  
 JPanel panel = new JPanel();  
  
 // Add a MouseAdapter to handle mouse events  
 panel.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 label.setText("Mouse Clicked at (" + e.getX() + ", " + e.getY() + ")");  
 }  
  
 @Override  
 public void mouseEntered(MouseEvent e) {  
 label.setText("Mouse Entered the panel!");  
 }  
  
 @Override  
 public void mouseExited(MouseEvent e) {  
 label.setText("Mouse Exited the panel!");  
 }  
 });  
  
 // Set the layout and add components  
 frame.setLayout(new BorderLayout());  
 frame.add(panel, BorderLayout.CENTER);  
 frame.add(label, BorderLayout.SOUTH);  
  
 // Set frame properties  
 frame.setSize(400, 300); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
 }



1. Create a GUI with buttons labeled "Red," "Green," and "Blue." Change the background color of a panel based on the button clicked.

This Swing application creates a GUI with three buttons labeled "Red," "Green," and "Blue." When each button is clicked, the background color of a JPanel changes to the corresponding color.

import javax.swing.;

import java.awt.;

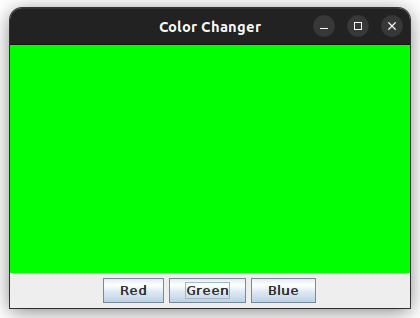
import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class ColorChanger {

public static void main(String[] args) {  
 // Create the main JFrame container  
 JFrame frame = new JFrame("Color Changer");  
  
 // Create a JPanel to change background color  
 JPanel panel = new JPanel();  
  
 // Create buttons for Red, Green, and Blue colors  
 JButton redButton = new JButton("Red");  
 JButton greenButton = new JButton("Green");  
 JButton blueButton = new JButton("Blue");  
  
 // Add ActionListeners to the buttons  
 redButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 panel.setBackground(Color.RED);  
 }  
 });  
  
 greenButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 panel.setBackground(Color.GREEN);  
 }  
 });  
  
 blueButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 panel.setBackground(Color.BLUE);  
 }  
 });  
  
 // Create a JPanel for buttons and add buttons to it  
 JPanel buttonPanel = new JPanel();  
 buttonPanel.add(redButton);  
 buttonPanel.add(greenButton);  
 buttonPanel.add(blueButton);  
  
 // Set layout for the main frame and add panels  
 frame.setLayout(new BorderLayout());  
 frame.add(panel, BorderLayout.CENTER);  
 frame.add(buttonPanel, BorderLayout.SOUTH);  
  
 // Set frame properties  
 frame.setSize(400, 300); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}

}



1. Write a program to capture and display keystrokes in a JTextArea.

This Swing application captures and displays keystrokes in a JTextArea using a KeyListener. Each key pressed by the user is displayed in the text area.

import javax.swing.\*;

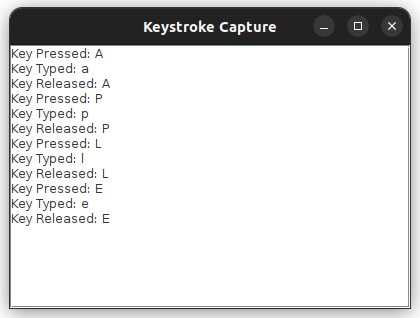
import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import java.awt.BorderLayout;

public class KeystrokeCapture {

public static void main(String[] args) {  
 JFrame frame = new JFrame("Keystroke Capture");  
  
 JTextArea textArea = new JTextArea(10, 30);  
 textArea.setEditable(false); // Make the text area non-editable by the user  
  
 frame.addKeyListener(new KeyListener() {  
 @Override  
 public void keyTyped(KeyEvent e) {  
 textArea.append("Key Typed: " + e.getKeyChar() + "\n");  
 }  
  
 @Override  
 public void keyPressed(KeyEvent e) {  
 textArea.append("Key Pressed: " + KeyEvent.getKeyText(e.getKeyCode()) + "\n");  
 }  
  
 @Override  
 public void keyReleased(KeyEvent e) {  
 textArea.append("Key Released: " + KeyEvent.getKeyText(e.getKeyCode()) + "\n");  
 }  
 });  
  
 JScrollPane scrollPane = new JScrollPane(textArea);  
 frame.setLayout(new BorderLayout());  
 frame.add(scrollPane, BorderLayout.CENTER);  
 frame.setSize(400, 300);   
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);   
 frame.setVisible(true); // Make the frame visible  
  
 // Set focus to the frame to capture keystrokes  
 frame.requestFocus();  
}  
}



1. Create an application that draws a circle at the position where the user clicks the mouse.

This Swing application allows the user to draw circles at the positions where they click the mouse on a JPanel. The circles are drawn using a MouseListener.

import javax.swing.;

import java.awt.;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

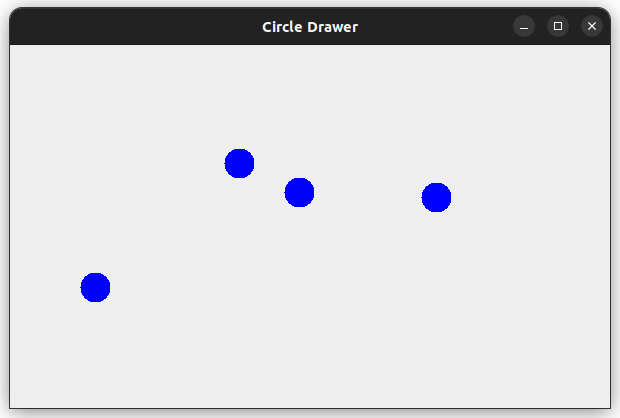
import java.util.ArrayList;

public class CircleDrawer {

public static void main(String[] args) {  
 JFrame frame = new JFrame("Circle Drawer");  
  
 // Create a JPanel to draw circles  
 DrawingPanel panel = new DrawingPanel();  
  
 // Add a MouseAdapter to capture mouse clicks and draw circles  
 panel.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mouseClicked(MouseEvent e) {  
 panel.addCircle(e.getX(), e.getY());  
 }  
 });  
  
 frame.add(panel);  
 frame.setSize(600, 400); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
}

class DrawingPanel extends JPanel { private final ArrayList circles = new ArrayList<>();

public void addCircle(int x, int y) {  
 circles.add(new Point(x, y));  
 repaint(); // Repaint the panel to update the drawing  
}  
@Override  
protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 g.setColor(Color.BLUE);  
 for (Point p : circles) {  
 g.fillOval(p.x - 15, p.y - 15, 30, 30); // Draw circle with center at mouse click  
 }  
}  
}



1. Write a program that shows a confirmation dialog when the user attempts to close the window.

This Swing application displays a confirmation dialog when the user attempts to close the window, asking them to confirm their action before the window closes.

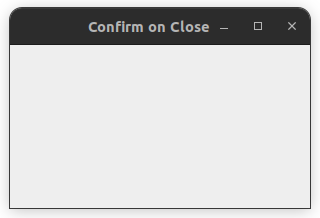
import javax.swing.\*;

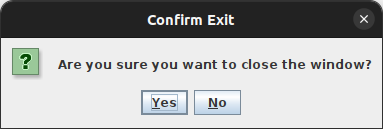
import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

public class ConfirmOnClose {

public static void main(String[] args) {  
 // Create the main JFrame container  
 JFrame frame = new JFrame("Confirm on Close");  
  
 // Add a WindowListener to show a confirmation dialog on close  
 frame.addWindowListener(new WindowAdapter() {  
 @Override  
 public void windowClosing(WindowEvent e) {  
 int response = JOptionPane.showConfirmDialog(  
 frame,  
 "Are you sure you want to close the window?",  
 "Confirm Exit",  
 JOptionPane.YES\_NO\_OPTION,  
 JOptionPane.QUESTION\_MESSAGE  
 );  
  
 if (response == JOptionPane.YES\_OPTION) {  
 frame.dispose(); // Close the frame  
 } // If NO\_OPTION, do nothing, and the window remains open  
 }  
 });  
  
 // Set frame properties  
 frame.setSize(300, 200); // Set window size  
 frame.setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE); // Prevent default close operation  
 frame.setVisible(true); // Make the frame visible  
}  
}





1. Create a GUI with checkboxes for hobbies. Display the selected hobbies in real time.

This Swing application creates a GUI with checkboxes for selecting hobbies. It displays the selected hobbies in real time using a JLabel.

import javax.swing.;

import java.awt.;

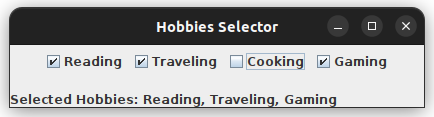
import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import java.util.ArrayList;

public class HobbiesSelector {

public static void main(String[] args) {  
 // Create the main JFrame container  
 JFrame frame = new JFrame("Hobbies Selector");  
  
 // Create a JLabel to display selected hobbies  
 JLabel selectedHobbiesLabel = new JLabel("Selected Hobbies: None");  
  
 // Create checkboxes for different hobbies  
 JCheckBox readingCheckbox = new JCheckBox("Reading");  
 JCheckBox travelingCheckbox = new JCheckBox("Traveling");  
 JCheckBox cookingCheckbox = new JCheckBox("Cooking");  
 JCheckBox gamingCheckbox = new JCheckBox("Gaming");  
  
 // List to hold selected hobbies  
 ArrayList<String> selectedHobbies = new ArrayList<>();  
  
 // Create an ItemListener to update selected hobbies  
 ItemListener itemListener = new ItemListener() {  
 @Override  
 public void itemStateChanged(ItemEvent e) {  
 JCheckBox source = (JCheckBox) e.getSource();  
 if (e.getStateChange() == ItemEvent.SELECTED) {  
 selectedHobbies.add(source.getText());  
 } else {  
 selectedHobbies.remove(source.getText());  
 }  
 updateLabel();  
 }  
  
 // Method to update the JLabel with selected hobbies  
 private void updateLabel() {  
 if (selectedHobbies.isEmpty()) {  
 selectedHobbiesLabel.setText("Selected Hobbies: None");  
 } else {  
 selectedHobbiesLabel.setText("Selected Hobbies: " + String.join(", ", selectedHobbies));  
 }  
 }  
 };  
  
 // Add the ItemListener to the checkboxes  
 readingCheckbox.addItemListener(itemListener);  
 travelingCheckbox.addItemListener(itemListener);  
 cookingCheckbox.addItemListener(itemListener);  
 gamingCheckbox.addItemListener(itemListener);  
  
 // Create a JPanel and add checkboxes to it  
 JPanel checkBoxPanel = new JPanel();  
 checkBoxPanel.add(readingCheckbox);  
 checkBoxPanel.add(travelingCheckbox);  
 checkBoxPanel.add(cookingCheckbox);  
 checkBoxPanel.add(gamingCheckbox);  
  
 // Set layout for the main frame and add components  
 frame.setLayout(new BorderLayout());  
 frame.add(checkBoxPanel, BorderLayout.CENTER);  
 frame.add(selectedHobbiesLabel, BorderLayout.SOUTH);  
  
 // Set frame properties  
 frame.setSize(400, 300); // Set window size  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Close on exit  
 frame.setVisible(true); // Make the frame visible  
}  
}



1. Write a program to connect to a MySQL/PostgreSQL database and display the database metadata.

This Java program connects to a PostgreSQL database and displays the database metadata, such as the database product name, version, and driver information.

import java.sql.Connection;

import java.sql.DatabaseMetaData;

import java.sql.DriverManager;

import java.sql.SQLException;

public class PostgreSQLMetadata {

public static void main(String[] args) {

// JDBC URL, username, and password of PostgreSQL server

String url = "jdbc:postgresql://localhost:5432/";

String user = "postgres";

String password = "password";

// Establish the connection

try (Connection connection = DriverManager.getConnection(url, user, password)) {

// Get the metadata

DatabaseMetaData metadata = connection.getMetaData();

// Display the metadata

System.out.println("Database Product Name: " + metadata.getDatabaseProductName());

System.out.println("Database Product Version: " + metadata.getDatabaseProductVersion());

System.out.println("Database Driver Name: " + metadata.getDriverName());

System.out.println("Database Driver Version: " + metadata.getDriverVersion());

} catch (SQLException e) {

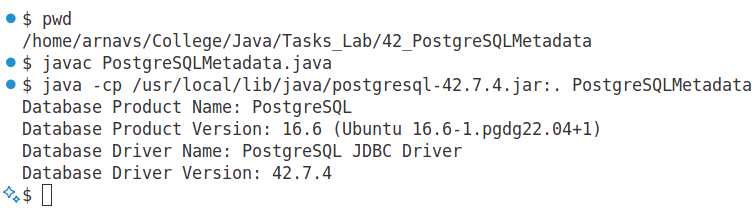
// Handle SQL exception

e.printStackTrace();

}

}

}



1. Create a table Students with columns id, name, and grade using a Java program.

This Java program connects to a PostgreSQL database and creates a table named Students with columns id, name, and grade.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

import java.sql.SQLException;

public class CreateStudentsTable {

public static void main(String[] args) {

// JDBC URL, username, and password of PostgreSQL server

String url = "jdbc:postgresql://localhost:5432/java\_lab";

String user = "postgres";

String password = "password";

// SQL query to create the Students table

String createTableSQL = "CREATE TABLE IF NOT EXISTS Students ("

+ "id SERIAL PRIMARY KEY, "

+ "name VARCHAR(100), "

+ "grade CHAR(2)"

+ ");";

// Establish the connection and execute the query

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

Statement statement = connection.createStatement()) {

statement.executeUpdate(createTableSQL);

System.out.println("Students table created successfully.");

} catch (SQLException e) {

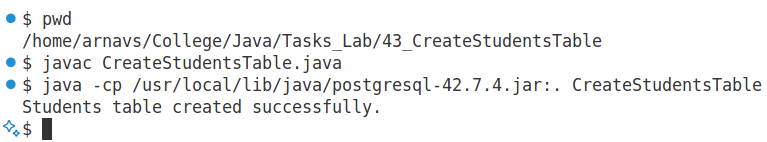
// Handle SQL exceptions

e.printStackTrace();

}

}

}



1. Write a program to insert, update, and delete records from the Students table.

This Java program demonstrates how to insert, update, and delete records from the Students table in PostgreSQL using JDBC. This Java program demonstrates how to insert, update, and delete records from the Students table in PostgreSQL using JDBC. The program updates the grade of the student John Doe to B. The program deletes the record of the student John Doe.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class StudentCRUDOperations {

public static void main(String[] args) {

String url = "jdbc:postgresql://localhost:5432/java\_lab";

String user = "postgres";

String password = "password";

String insertSQL = "INSERT INTO Students (name, grade) VALUES (?, ?)";

String updateSQL = "UPDATE Students SET grade = ? WHERE name = ?";

String deleteSQL = "DELETE FROM Students WHERE name = ?";

try (Connection connection = DriverManager.getConnection(url, user, password)) {

// Insert a new student

try (PreparedStatement insertStatement = connection.prepareStatement(insertSQL)) {

insertStatement.setString(1, "John Doe");

insertStatement.setString(2, "A");

int rowsInserted = insertStatement.executeUpdate();

System.out.println(rowsInserted + " record(s) inserted.");

}

// Update an existing student

try (PreparedStatement updateStatement = connection.prepareStatement(updateSQL)) {

updateStatement.setString(1, "B");

updateStatement.setString(2, "John Doe");

int rowsUpdated = updateStatement.executeUpdate();

System.out.println(rowsUpdated + " record(s) updated.");

}

// Delete a student record

try (PreparedStatement deleteStatement = connection.prepareStatement(deleteSQL)) {

deleteStatement.setString(1, "John Doe");

int rowsDeleted = deleteStatement.executeUpdate();

System.out.println(rowsDeleted + " record(s) deleted.");

}

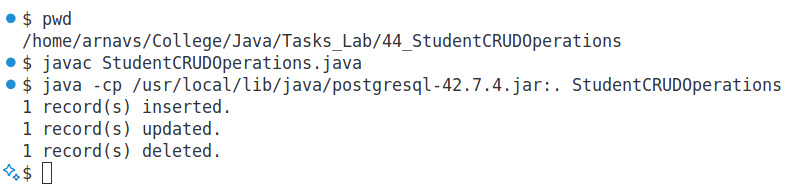
} catch (SQLException e) {

e.printStackTrace();

}

}

}



1. Use a PreparedStatement to query student details based on their grade.

This Java program uses a PreparedStatement to query student details based on their grade. Before running the program, you will need to insert sample data into the Students table.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class QueryStudentsByGrade {

public static void main(String[] args) {

// JDBC URL, username, and password of PostgreSQL server

String url = "jdbc:postgresql://localhost:5432/java\_lab";

String user = "postgres";

String password = "password";

// SQL query to select students based on grade

String querySQL = "SELECT id, name, grade FROM Students WHERE grade = ?";

// Grade to search for

String gradeToSearch = "A"; // Change this value to search for other grades

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

PreparedStatement preparedStatement = connection.prepareStatement(querySQL)) {

// Set the grade parameter in the query

preparedStatement.setString(1, gradeToSearch);

// Execute the query and get the result set

ResultSet resultSet = preparedStatement.executeQuery();

// Display the results

System.out.println("Students with grade " + gradeToSearch + ":");

while (resultSet.next()) {

int id = resultSet.getInt("id");

String name = resultSet.getString("name");

String grade = resultSet.getString("grade");

System.out.println("ID: " + id + ", Name: " + name + ", Grade: " + grade);

}

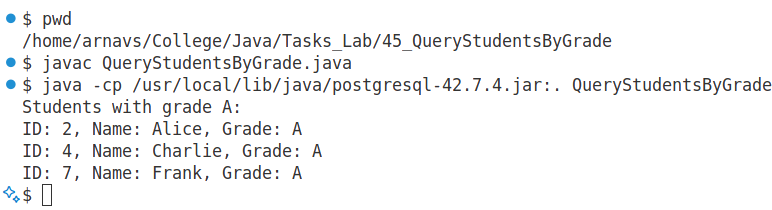
} catch (SQLException e) {

e.printStackTrace();

}

}

}



1. Create a program that fetches student records in both forward and reverse directions using scrollable result sets.

This Java program demonstrates how to fetch student records in both forward and reverse directions using a scrollable result set in JDBC. A scrollable result set allows you to move through the data in both directions and even jump to a specific row.

import java.sql.Connection;

import java.sql.DriverManager;

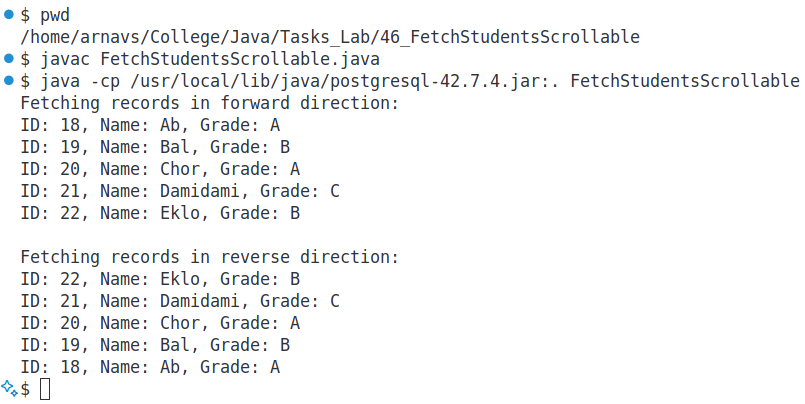
import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class FetchStudentsScrollable {

public static void main(String[] args) {  
 // JDBC URL, username, and password of PostgreSQL server  
 String url = "jdbc:postgresql://localhost:5432/java\_lab";  
 String user = "postgres";  
 String password = "password";  
  
 // SQL query to fetch all student records  
 String querySQL = "SELECT id, name, grade FROM Students";  
  
 try (Connection connection = DriverManager.getConnection(url, user, password);  
 Statement statement = connection.createStatement(ResultSet.TYPE\_SCROLL\_INSENSITIVE, ResultSet.CONCUR\_READ\_ONLY)) {  
  
 // Execute the query and get the scrollable result set  
 ResultSet resultSet = statement.executeQuery(querySQL);  
  
 // Fetch records in forward direction  
 System.out.println("Fetching records in forward direction:");  
 while (resultSet.next()) {  
 int id = resultSet.getInt("id");  
 String name = resultSet.getString("name");  
 String grade = resultSet.getString("grade");  
 System.out.println("ID: " + id + ", Name: " + name + ", Grade: " + grade);  
 }  
  
 // Move the cursor to the last row to enable reverse fetching  
 if (resultSet.last()) {  
 System.out.println("\nFetching records in reverse direction:");  
 do {  
 int id = resultSet.getInt("id");  
 String name = resultSet.getString("name");  
 String grade = resultSet.getString("grade");  
 System.out.println("ID: " + id + ", Name: " + name + ", Grade: " + grade);  
 } while (resultSet.previous()); // Move to the previous record  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
}  
 }



1. Write a program to perform a transaction with multiple SQL statements. Rollback the transaction in case of an error.

This Java program demonstrates how to perform a transaction with multiple SQL statements. If any error occurs during the execution of the statements, the program will rollback the entire transaction to maintain data integrity.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class TransactionExample {

public static void main(String[] args) {

// JDBC URL, username, and password of PostgreSQL server

String url = "jdbc:postgresql://localhost:5432/";

String user = "postgres";

String password = "password";

// SQL queries for inserting records into the Students table

String insertStudent1 = "INSERT INTO Students (name, grade) VALUES (?, ?)";

String insertStudent2 = "INSERT INTO Students (name, grade) VALUES (?, ?)";

// SQL queries for update and delete (optional for demonstration)

// String updateStudent = "UPDATE Students SET grade = ? WHERE name = ?";

// String deleteStudent = "DELETE FROM Students WHERE name = ?";

Connection connection = null;

try {

connection = DriverManager.getConnection(url, user, password);

// Disable auto-commit to start a transaction

connection.setAutoCommit(false);

// Insert first student

try (PreparedStatement pstmt1 = connection.prepareStatement(insertStudent1)) {

pstmt1.setString(1, "John Doe");

pstmt1.setString(2, "A");

pstmt1.executeUpdate();

}

try (PreparedStatement pstmt2 = connection.prepareStatement(insertStudent2)) {

pstmt2.setString(1, "Jane Doe");

pstmt2.setString(2, "B");

pstmt2.executeUpdate();

}

// Commit the transaction if no error occurs

connection.commit();

System.out.println("Transaction completed successfully.");

} catch (SQLException e) {

// If an error occurs, rollback the transaction to ensure data consistency

if (connection != null) {

try {

System.out.println("Transaction failed. Rolling back...");

connection.rollback();

} catch (SQLException rollbackEx) {

rollbackEx.printStackTrace();

}

}

e.printStackTrace();

} finally {

try {

if (connection != null) {

connection.setAutoCommit(true);

connection.close();

}

} catch (SQLException e) {

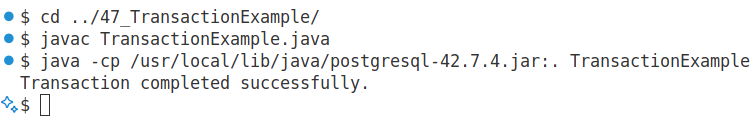
e.printStackTrace();

}

}

}

}



1. Demonstrate the use of CachedRowSet to fetch data from a database and update it offline.

This Java program demonstrates how to use the CachedRowSet to fetch data from a database, manipulate it offline (without holding an active connection to the database), and update the database once the changes are ready. CachedRowSet is a disconnected version of ResultSet, which allows you to fetch, update, and modify data offline, and then apply changes to the database when needed.

import javax.sql.rowset.CachedRowSet;

import javax.sql.rowset.RowSetProvider;

import java.sql.\*;

public class CachedRowSetExample {

public static void main(String[] args) {

String url = "jdbc:postgresql://localhost:5432/java\_lab";

String user = "postgres";

String password = "password";

String querySQL = "SELECT id, name, grade FROM Students";

try (Connection connection = DriverManager.getConnection(url, user, password)) {

// Disable auto-commit for manual transaction control

connection.setAutoCommit(false);

// Create a CachedRowSet object

CachedRowSet cachedRowSet = RowSetProvider.newFactory().createCachedRowSet();

// Set the connection information for the CachedRowSet

cachedRowSet.setUrl(url);

cachedRowSet.setUsername(user);

cachedRowSet.setPassword(password);

cachedRowSet.setCommand(querySQL);

// Populate the CachedRowSet with data from the database

cachedRowSet.execute();

// Display original data

System.out.println("Original Data:");

while (cachedRowSet.next()) {

int id = cachedRowSet.getInt("id");

String name = cachedRowSet.getString("name");

String grade = cachedRowSet.getString("grade");

System.out.println("ID: " + id + ", Name: " + name + ", Grade: " + grade);

}

// Modify the data offline (e.g., update grades)

cachedRowSet.beforeFirst(); // Move cursor to the first row

while (cachedRowSet.next()) {

String name = cachedRowSet.getString("name");

if (("Bal".equals(name)) || ("Eklo".equals(name))) {

cachedRowSet.updateString("grade", "C"); // Update grade for John Doe

cachedRowSet.updateRow(); // Commit the change to the row

}

}

// Display modified data offline

System.out.println("\nModified Data (Offline Changes):");

cachedRowSet.beforeFirst();

while (cachedRowSet.next()) {

int id = cachedRowSet.getInt("id");

String name = cachedRowSet.getString("name");

String grade = cachedRowSet.getString("grade");

System.out.println("ID: " + id + ", Name: " + name + ", Grade: " + grade);

}

// Apply changes back to the database

cachedRowSet.acceptChanges(connection); // This writes the changes back to the database

connection.commit();

System.out.println("\nChanges have been committed to the database.");

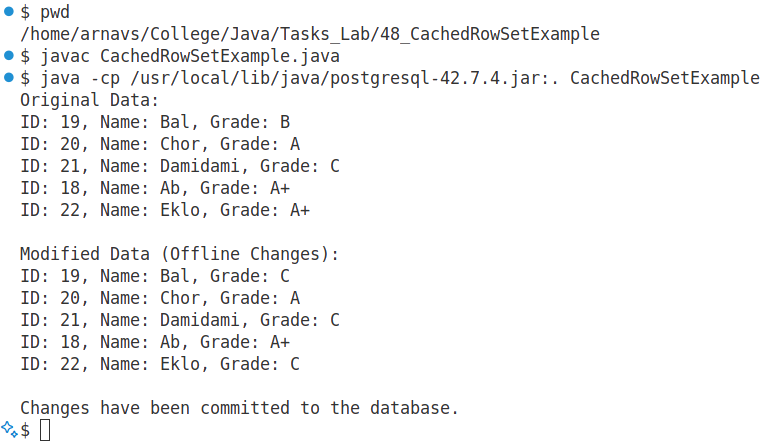
} catch (SQLException e) {

e.printStackTrace();

}

}

}



1. Write a program to display the local IP address and port number of the host machine.

This program will fetch and display the local IP address and the port number of the host machine using Java's networking capabilities.

import java.net.\*;

public class LocalIPAddress {

public static void main(String[] args) {

try {

// Get the local host IP address

InetAddress localHost = InetAddress.getLocalHost();

String ipAddress = localHost.getHostAddress();

int port = 8080;

// Display the IP address and port number

System.out.println("Local IP Address: " + ipAddress);

System.out.println("Port Number: " + port);

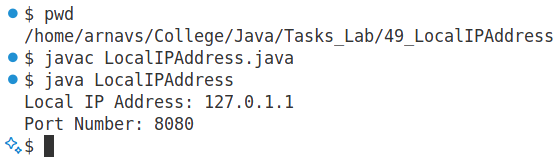
} catch (UnknownHostException e) {

e.printStackTrace();

}

}

}



1. Use the InetAddress class to find the hostname and IP address of a given domain name.

This program uses the InetAddress class in Java to retrieve the hostname and IP address of a given domain name. The InetAddress class allows you to resolve a domain name to an IP address.

import java.net.\*;

public class DomainInfo {

public static void main(String[] args) {

// The domain name for which we need to find the hostname and IP address

String domainName = "www.example.com"; // Replace with any domain name

try {

// Get the InetAddress for the given domain

InetAddress inetAddress = InetAddress.getByName(domainName);

// Get the hostname and IP address

String hostname = inetAddress.getHostName();

String ipAddress = inetAddress.getHostAddress();

// Display the results

System.out.println("Domain Name: " + domainName);

System.out.println("Hostname: " + hostname);

System.out.println("IP Address: " + ipAddress);

} catch (UnknownHostException e) {

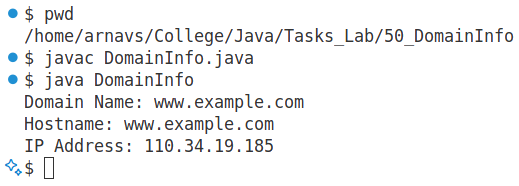
System.out.println("Unable to resolve domain name: " + domainName);

e.printStackTrace();

}

}

}



1. Create a client-server application where the client sends a message, and the server responds with a reversed version of the message.

This program demonstrates a simple client-server application using sockets in Java. The client sends a message to the server, and the server responds with the reversed version of the message. The server listens for incoming client connections, reads the message from the client, reverses it, and sends it back. The client connects to the server, sends a message, and prints the reversed message received from the server.

Server Code:

import java.net.\*;

import java.io.\*;

public class ReversalServer {

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(12345)) {

System.out.println("Server is listening on port 12345");

// Accept client connection

try (Socket socket = serverSocket.accept()) {

System.out.println("Client connected: " + socket.getInetAddress());

// Create input and output streams

BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

// Read the message from the client

String message = input.readLine();

System.out.println("Received from client: " + message);

// Reverse the message

String reversedMessage = new StringBuilder(message).reverse().toString();

// Send the reversed message back to the client

output.println(reversedMessage);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

Client Code:

import java.net.\*;

import java.io.\*;

public class ReversalClient {

public static void main(String[] args) {

String serverAddress = "localhost"; // Server address

int port = 12345; // Port number

try (Socket socket = new Socket(serverAddress, port)) {

// Create input and output streams

BufferedReader consoleInput = new BufferedReader(new InputStreamReader(System.in));

BufferedReader serverInput = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter serverOutput = new PrintWriter(socket.getOutputStream(), true);

// Read a message from the console

System.out.print("Enter a message: ");

String message = consoleInput.readLine();

// Send the message to the server

serverOutput.println(message);

// Receive the reversed message from the server

String reversedMessage = serverInput.readLine();

System.out.println("Reversed message from server: " + reversedMessage);

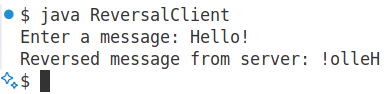
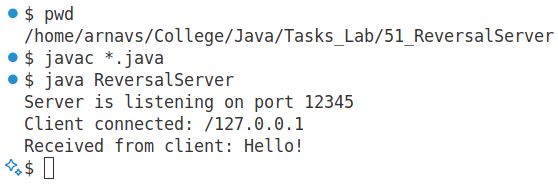
} catch (IOException e) {

e.printStackTrace();

}

}

}



1. Implement a UDP-based program where the client sends a number, and the server responds with its square.

This program demonstrates a simple client-server communication using the UDP protocol. The client sends a number to the server, and the server responds with the square of that number. The server waits for a UDP packet from the client, extracts the number, computes its square, and sends the result back. The client sends a number to the server and waits for the squared result.

Server Code:

import java.net.\*;

public class UDPSquareServer {

public static void main(String[] args) {

DatagramSocket socket = null;

try {

// Create a DatagramSocket to receive and send UDP packets

socket = new DatagramSocket(9876);

System.out.println("Server is listening on port 9876...");

while (true) {

byte[] receiveData = new byte[1024];

// Receive data from the client

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

socket.receive(receivePacket);

// Convert received data to string and extract the number

String receivedMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());

int number = Integer.parseInt(receivedMessage.trim());

System.out.println("Received number: " + number);

// Calculate the square of the number

int square = number \* number;

// Send the result back to the client

String response = String.valueOf(square);

byte[] sendData = response.getBytes();

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

// Send the squared value as a response

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, clientAddress, clientPort);

socket.send(sendPacket);

}

} catch (Exception e) {

e.printStackTrace();

} finally {

if (socket != null && !socket.isClosed()) {

socket.close();

}

}

}

}

Client Code:

import java.net.\*;

import java.util.Scanner;

public class UDPSquareClient {

public static void main(String[] args) {

DatagramSocket socket = null;

Scanner scanner = new Scanner(System.in);

try {

// Create a DatagramSocket to send and receive UDP packets

socket = new DatagramSocket();

InetAddress serverAddress = InetAddress.getByName("localhost");

int serverPort = 9876;

// Read the number from the user

System.out.print("Enter a number: ");

int number = scanner.nextInt();

String message = String.valueOf(number);

// Send the number to the server

byte[] sendData = message.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, serverAddress, serverPort);

socket.send(sendPacket);

// Receive the response from the server (squared number)

byte[] receiveData = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

socket.receive(receivePacket);

// Convert received data to string and print the squared number

String squaredNumber = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("Squared number from server: " + squaredNumber);

} catch (Exception e) {

e.printStackTrace();

} finally {

if (socket != null && !socket.isClosed()) {

socket.close();

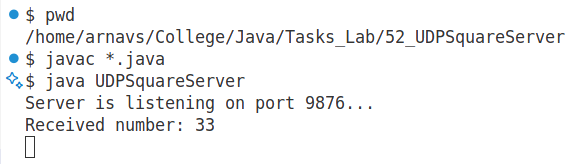
}

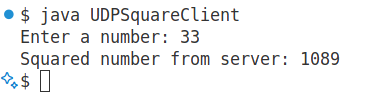
scanner.close();

}

}

}





1. Write a program to fetch and display the content of a given URL.

This program demonstrates how to fetch and display the content of a given URL using Java. The program uses the URL and URLConnection classes to connect to the URL and retrieve the content, which is then displayed in the console.

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.net.URL;

import java.net.HttpURLConnection;

public class FetchURLContent {

public static void main(String[] args) {

try {

// Define the URL to fetch

String urlString = "https://www.example.com";

URL url = new URL(urlString);

// Open a connection to the URL

HttpURLConnection connection = (HttpURLConnection) url.openConnection();

connection.setRequestMethod("GET"); // Set the request method to GET

// Get the input stream to read the content

BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));

String inputLine;

StringBuilder content = new StringBuilder();

// Read the content line by line

while ((inputLine = in.readLine()) != null) {

content.append(inputLine).append("\n");

}

in.close();

// Print the content of the URL

System.out.println(content.toString());

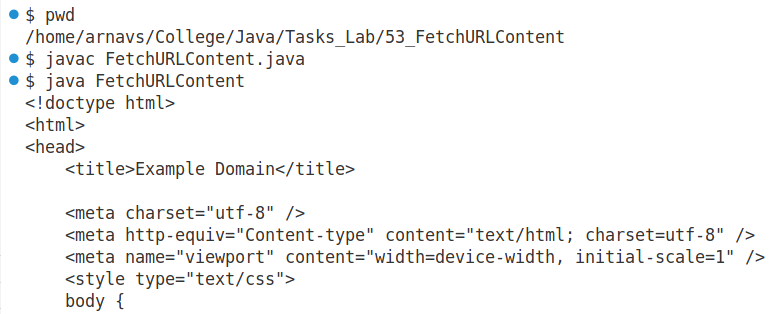
} catch (Exception e) {

e.printStackTrace();

}

}

}



1. Create a program to download and save an image file from a given URL.

This program demonstrates how to download an image from a given URL and save it to a local file. The program uses Java's URL and URLConnection classes to connect to the URL and retrieve the image, then it saves the content to a file using InputStream and FileOutputStream.

import java.io.InputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.net.URL;

import java.net.HttpURLConnection;

public class DownloadImage {

public static void main(String[] args) {

imageUrl = "https://www.google.com/images/branding/googlelogo/1x/googlelogo\_light\_color\_272x92dp.png";

String savePath = "downloaded\_image.png";

try {  
 URL url = new URL(imageUrl);  
   
 // Open a connection to the URL  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
   
 // Open an input stream to read the image data  
 InputStream inputStream = connection.getInputStream();  
 // Create an output stream to save the image to a file  
 FileOutputStream outputStream = new FileOutputStream(savePath);  
   
 // Read and write the image data in chunks  
 byte[] buffer = new byte[4096]; // 4KB buffer  
 int bytesRead;  
 while ((bytesRead = inputStream.read(buffer)) != -1) {  
 outputStream.write(buffer, 0, bytesRead);  
 }  
   
 // Close the input and output streams  
 inputStream.close();  
 outputStream.close();  
   
 System.out.println("Image downloaded successfully and saved as " + savePath);  
   
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}  
}

