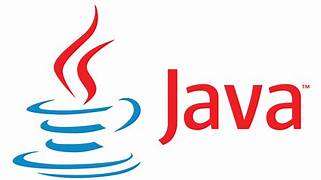
JAVA PROJECT 

PROJECT NAME: -

THE EXPENSE TRACKER

DESCRIPTION

The source code provided outlines a basic expense tracker application developed using Java Swing for the graphical user interface (GUI) and MySQL database for data storage. Here’s a detailed description of the key components.

SOURCE CODE

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.sql.\*;

import java.util.Vector;

import javax.swing.table.DefaultTableModel;

import java.sql.DriverManager;

import java.sql.Connection;

import java.sql.SQLException;

public class ExpenseTrackerApp extends JFrame implements ActionListener {

private JLabel lblDate, lblDescription, lblCategory, lblAmount;

private JTextField txtDate, txtDescription, txtCategory, txtAmount;

private JButton btnSave, btnLoad;

private JTable expenseTable;

private JScrollPane scrollPane;

public DefaultTableModel tableModel;

private Connection conn;

private PreparedStatement pstmt;

private ResultSet rs;

public ExpenseTrackerApp() {

setTitle("Expense Tracker");

setSize(800, 400);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

// Initialize components

lblDate = new JLabel("Date:");

lblDescription = new JLabel("Description:");

lblCategory = new JLabel("Category:");

lblAmount = new JLabel("Amount:");

txtDate = new JTextField(10);

txtDescription = new JTextField(20);

txtCategory = new JTextField(10);

txtAmount = new JTextField(10);

btnSave = new JButton("Save");

btnLoad = new JButton("Load Expenses");

btnSave.addActionListener(this);

btnLoad.addActionListener(this);

// Table setup

tableModel = new DefaultTableModel();

expenseTable = new JTable(tableModel);

scrollPane = new JScrollPane(expenseTable);

scrollPane.setPreferredSize(new Dimension(700, 200));

JPanel inputPanel = new JPanel(new GridLayout(2, 4));

inputPanel.add(lblDate);

inputPanel.add(txtDate);

inputPanel.add(lblDescription);

inputPanel.add(txtDescription);

inputPanel.add(lblCategory);

inputPanel.add(txtCategory);

inputPanel.add(lblAmount);

inputPanel.add(txtAmount);

JPanel buttonPanel = new JPanel();

buttonPanel.add(btnSave);

buttonPanel.add(btnLoad);

JPanel mainPanel = new JPanel(new BorderLayout());

mainPanel.add(inputPanel, BorderLayout.NORTH);

mainPanel.add(scrollPane, BorderLayout.CENTER);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

add(mainPanel);

setVisible(true);

// Connect to database

try {

Class.forName("com.mysql.cj.jdbc.Driver");

conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/expense\_tracker", "root", "Root");

} catch (ClassNotFoundException | SQLException e) {

e.printStackTrace();

}

}

public void actionPerformed(ActionEvent e) {

if (e.getSource() == btnSave) {

saveExpense();

} else if (e.getSource() == btnLoad) {

loadExpenses();

}

}

private void saveExpense() {

String date = txtDate.getText();

String description = txtDescription.getText();

String category = txtCategory.getText();

double amount = Double.parseDouble(txtAmount.getText());

try {

String sql = "INSERT INTO expenses (date, description, category, amount) VALUES (?, ?, ?, ?)";

pstmt = conn.prepareStatement(sql);

pstmt.setString(1, date);

pstmt.setString(2, description);

pstmt.setString(3, category);

pstmt.setDouble(4, amount);

pstmt.executeUpdate();

JOptionPane.showMessageDialog(this, "Expense saved successfully!");

clearFields();

} catch (SQLException ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(this, "Error saving expense.");

}

}

private void loadExpenses() {

try {

tableModel.setColumnCount(0);

tableModel.setRowCount(0);

pstmt = conn.prepareStatement("SELECT \* FROM expenses");

rs = pstmt.executeQuery();

ResultSetMetaData rsmd = rs.getMetaData();

int columnCount = rsmd.getColumnCount();

// Add columns to table model

for (int i = 1; i <= columnCount; i++) {

tableModel.addColumn(rsmd.getColumnName(i));

}

// Add rows to table model

while (rs.next()) {

Vector<String> row = new Vector<>();

for (int i = 1; i <= columnCount; i++) {

row.add(rs.getString(i));

}

tableModel.addRow(row);

}

} catch (SQLException ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(this, "Error loading expenses.");

}

}

private void clearFields() {

txtDate.setText("");

txtDescription.setText("");

txtCategory.setText("");

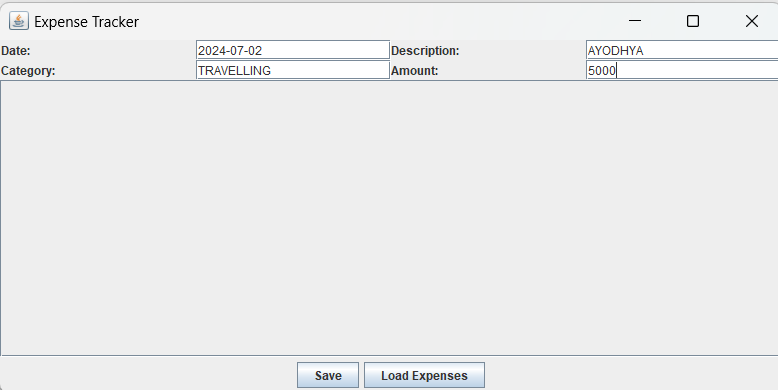
txtAmount.setText("");

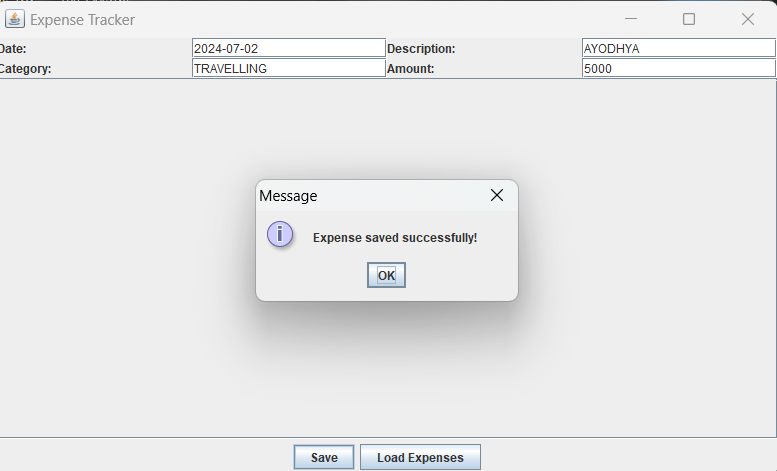
}

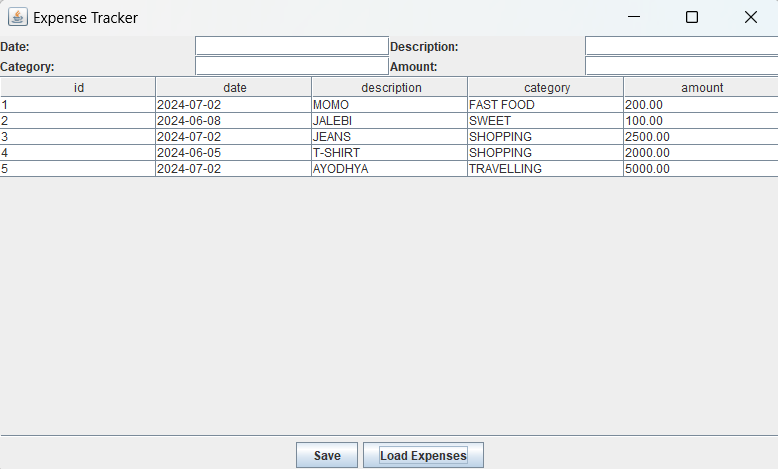
public static void main(String[] args) {

new ExpenseTrackerApp();

}}

OUTPUT





Components of the Expense Tracker:

**User Interface (UI) using Java Swing:**

* **Main Window:** This serves as the primary interface where users can view their expenses, add new expenses, and manage existing ones.
* **Forms:** Dialogs or panels for adding new expenses, editing existing ones, and viewing detailed expense information.
* **Menu and Toolbar:** Options to navigate between different functionalities like adding expenses, viewing reports, or managing categories.
* **Table or List Display:** Shows a list of expenses with columns for date, description, amount, and category. This can be implemented using JTable or JList components.

**Functionality:**

* **Adding Expenses:** Users can input details such as date, description, amount, and category for each expense.
* **Editing and Deleting Expenses:** Ability to modify or remove existing expenses from the database.
* **Categories:** Users can categorize expenses (e.g., food, transportation, bills) to better track spending habits.
* **Reports:** Generate summaries and reports based on expenses over specific periods, categories, or custom queries.
* **Data Validation:** Ensure that inputs are validated (e.g., amount should be numeric, date should be in a valid format) before saving to the database.

**Database Design (MySQL):**

* + **Tables:** Create tables in MySQL to store expense data. Example tables could include:
    - expenses: Fields might include expense\_id, date, description, amount, category\_id, etc.
    - categories: Categories for expenses (category\_id, category\_name).
  + **SQL Queries:** Use SQL queries to insert, update, delete, and retrieve expense data from the database.
  + **Connection Handling:** Use JDBC (Java Database Connectivity) to connect Java application (Swing UI) with MySQL database.

**Integration:**

* + **JDBC Connection:** Establish connection to MySQL database using JDBC drivers (mysql-connector-java) in your Java project.
  + **Database Operations:** Implement methods/functions in Java to execute SQL queries for CRUD (Create, Read, Update, Delete) operations on expense data.
  + **Error Handling:** Handle exceptions that may arise during database interactions (e.g., connection errors, SQL syntax errors).

**Implementation Steps:**

**+Set Up Development Environment:**

* + Install Java Development Kit (JDK) and MySQL server.
  + Configure your Java IDE (like IntelliJ IDEA, Eclipse) for Swing development and JDBC connectivity.

1. **Design UI:**
   * Create Swing components (frames, dialogs, panels) using layout managers (e.g., BorderLayout, FlowLayout, GridBagLayout) for arranging UI elements.
   * Implement event handling for buttons, menu items, etc., to respond to user actions.
2. **Database Configuration:**
   * Set up MySQL database with necessary tables (expenses, categories) and insert initial data if required.
   * Ensure JDBC driver (mysql-connector-java) is added to your Java project's build path.
3. **Connect UI with Database:**
   * Write Java code to establish JDBC connection to MySQL database.
   * Implement methods to perform CRUD operations using SQL queries (insert, update, delete, select).
4. **Testing and Refinement:**
   * Test the application thoroughly to ensure all features work as expected.
   * Refine UI layout and functionality based on user feedback and testing results.
5. **Deployment:**
   * Package your Java application into executable JAR file for distribution.
   * Ensure users have MySQL server installed and correctly configured to use the expense tracker.

**Benefits of Using Java Swing and MySQL:**

* **Cross-Platform:** Java applications using Swing can run on any platform where Java is supported.
* **Scalability:** MySQL is scalable, allowing the expense tracker to handle large volumes of data over time.
* **Community Support:** Both Java Swing and MySQL have extensive community support and documentation, making it easier to troubleshoot issues and find resources.