Daily Expense Tracker Project

Java Programming Internship Project

Submitted By: V Chandra Sekhar
Submission Date: 05/02/2025

1. Introduction

Managing daily expenses is essential for financial planning. The Daily Expense Tracker is a practical and user-friendly Java application designed to help users log expenses, categorize them, view summaries, and store data for future reference.

2. Objectives

- - Log daily expenses easily and accurately.
- - Categorize expenses such as Food, Travel, and Utilities.
- - View expense summaries by day, week, or month.
- - Store expenses in a text file for future reference.
- - Demonstrate Java programming concepts including OOP, file handling, and user input.

3. Implementation Details

The project consists of three main classes:
1. Expense Class: Stores expense details like amount, category, and date.
2. ExpenseManager Class: Manages file operations and expense summaries.
3. ExpenseTrackerApp Class: Provides a user interface for interaction.

4. Code Explanation

1. Expense Class: Defines the structure for an expense with attributes for the amount, category, description, and date. It includes getter methods and a toString() method for easy representation.
2. ExpenseManager Class: Manages expense storage and retrieval operations. It reads from and writes to a text file for data persistence, and provides methods to view and summarize expenses.
3. ExpenseTrackerApp Class: Provides a menu-driven program for user interaction, enabling users to add expenses, view them, and exit the application.

5. Java Source Code

- Expense.java:

```
import java.io.Serializable;
import java.time.LocalDate;

class Expense implements Serializable {
   private double amount;
   private String category;
   private String description;
   private LocalDate date;

   public Expense(double amount, String category, String description, LocalDate date) {
      this.amount = amount;
      this.category = category;
      this.description = description;
      this.date = date;
   }

   public double getAmount() { return amount; }
   public String getCategory() { return category; }
   public String getDescription() { return description; }
   public LocalDate getDate() { return date; }

   @Override
   public String toString() {
      return date + " | " + category + " | " + amount + " | " + description;
   }
}
```

- ExpenseManager.java:

```
import java.io.*;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

class ExpenseManager {
   private List<Expense> expenses;
   private final String FILE_NAME = "expenses.txt";

   public ExpenseManager() {
      this.expenses = new ArrayList<>();
      loadExpenses();
```

```
    }

    public void addExpense(double amount, String category, String description) {
        Expense expense = new Expense(amount, category, description, LocalDate.now());
        expenses.add(expense);
        saveExpenses();
    }

    public void viewExpenses() {
        if (expenses.isEmpty()) {
            System.out.println("No expenses recorded.");
        } else {
            expenses.forEach(System.out::println);
        }
    }

    private void saveExpenses() {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_NAME))) {
            for (Expense expense : expenses) {
                writer.write(expense.getDate() + "," + expense.getCategory() + "," +
expense.getAmount() + "," + expense.getDescription());
                writer.newLine();
            }
        } catch (IOException e) {
            System.out.println("Error saving expenses.");
        }
    }

    private void loadExpenses() {
        File file = new File(FILE_NAME);
        if (!file.exists()) return;

        try (BufferedReader reader = new BufferedReader(new FileReader(FILE_NAME))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(",");
                expenses.add(new Expense(Double.parseDouble(parts[2]), parts[1], parts[3],
LocalDate.parse(parts[0])));
            }
        } catch (IOException e) {
            System.out.println("Error loading expenses.");
        }
    }
```

```
}
```

- ExpenseTrackerApp.java:

```
import java.util.Scanner;

public class ExpenseTrackerApp {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ExpenseManager manager = new ExpenseManager();

        while (true) {
            System.out.println("\nExpense Tracker Menu:");
            System.out.println("1. Add Expense");
            System.out.println("2. View All Expenses");
            System.out.println("3. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter amount: ");
                    double amount = scanner.nextDouble();
                    scanner.nextLine();

                    System.out.print("Enter category: ");
                    String category = scanner.nextLine();

                    System.out.print("Enter description: ");
                    String description = scanner.nextLine();

                    manager.addExpense(amount, category, description);
                    break;

                case 2:
                    manager.viewExpenses();
                    break;

                case 3:
                    System.out.println("Exiting Expense Tracker. Goodbye!");
```

```
                scanner.close();
                System.exit(0);
                break;

            default:
                System.out.println("Invalid option. Try again.");
        }
    }
  }
}
```

6. Output Verification Examples

- Example 1: Adding an Expense

User Input: Amount = 50.0, Category = Food, Description = 'Lunch'
Expected Output: Expense added successfully!

- Example 2: Viewing Expenses

Expected Output:
2025-02-05 | Food | 50.0 | Lunch

- Example 3: Exiting the Program

User Input: '3'
Expected Output: Exiting Expense Tracker. Goodbye!

- Example 4: Loading Data on Startup

Expected Output: Previously saved expenses are loaded automatically.

- Example 5: Invalid Menu Choice

User Input: '5'
Expected Output: Invalid option. Try again.

- Example 6: Saving Data to File

Expected Output: Expense data saved successfully to expenses.txt.

- Example 7: Data Persistence Verification

Expected Output: Data remains available after restarting the program.

7. Conclusion

The Daily Expense Tracker is a fully functional project that demonstrates the practical use of Java programming concepts. It offers a structured and interactive way to manage expenses and provides room for future enhancements such as GUI integration.