Week 3 : Task 4

**Building a Basic Expense Tracker application.**

**Overview**

The Expense Tracker is a Java-based application designed to help users manage their expenses. It provides basic functionalities such as adding expenses, viewing expenses, generating expense summaries, and saving data to a file for persistence.

**Features involves in the expense tracker**

**Add Expenses:**

Users can add expenses by providing the expense category, amount, and date.

Input validation ensures that the user provides valid information.

**View Expenses:**

Users can view a detailed list of all expenses, including the category, amount, and total spending.

The application provides a user-friendly interface for better interaction.

**Expense Summaries:**

Users can view summaries of expenses, broken down by category.

The summary includes the total spending for each category and an overall total.

**User-Friendly Interface:**

The application features a text-based user interface, making it easy for users to interact with the program.

Clear prompts and messages guide users through the different functionalities.

**Data Persistence:**

The application supports data persistence by saving expenses to a file.

Expenses are loaded from the file when the application starts, ensuring data continuity.

**Input Validation:**

Input validation is implemented to ensure that users provide correct and valid information.

The application provides informative messages for incorrect inputs, guiding users to correct them.

Usage

**Running the Application**

Compile the Java source file: javac ExpenseTrackerWithPersistence.java

Run the compiled class: java ExpenseTrackerWithPersistence

**Menu Options**

Add Expense (Option 1):

Choose option 1 to add a new expense.

Enter the expense category, amount, and date as prompted.

View Expenses (Option 2):

Choose option 2 to view detailed expenses.

The application displays a list of expenses with categories, amounts, and the total spending.

Expense Summaries (Option 3):

Choose option 3 to view summaries of expenses by category.

The summary includes the total spending for each category and an overall total.

Save Expenses to File (Option 4):

Choose option 4 to save the current expenses to a file named "expenses.txt".

This ensures that expenses are persisted between application sessions.

Exit (Option 5):

Choose option 5 to exit the application.

**Data Persistence**

The application uses a simple text file, "expenses.txt," to store expense data.

Expenses are loaded from this file when the application starts.

Saving expenses to the file ensures that the data is preserved between application sessions.

Future Enhancements

**Graphical User Interface (GUI):**

Consider transitioning to a graphical user interface for improved user experience.

**Budgeting:**

Implement budgeting features, allowing users to set budgets for each category.

**Data Analysis:**

Extend the application to provide more in-depth data analysis, such as spending trends and averages.

**Authentication:**

Add user authentication features for multi-user scenarios.

**Conclusion:**

The Expense Tracker application provides a straightforward solution for users to manage their expenses. It can be extended and customized based on specific needs and requirements.

**Code :**

**import java.io.\*;**

**import java.text.ParseException;**

**import java.text.SimpleDateFormat;**

**import java.util.\*;**

**class Expense {**

**String category;**

**double amount;**

**Date date;**

**Expense(String category, double amount, Date date) {**

**this.category = category;**

**this.amount = amount;**

**this.date = date;**

**} }**

**public class ExpenseTrackerWithPersistence {**

**private static final String DATA\_FILE = "expenses.txt";**

**private static final SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");**

**public static void main(String[] args) {**

**Map<String, Double> expensesByCategory = loadExpensesFromFile();**

**Scanner scanner = new Scanner(System.in);**

**while (true) {**

**displayMenu();**

**System.out.print("Enter your choice: ");**

**int choice;**

**try {**

**choice = scanner.nextInt();**

**scanner.nextLine(); // Consume the newline character**

**} catch (InputMismatchException e) {**

**System.out.println("Invalid input. Please enter a number.");**

**scanner.nextLine(); // Consume the invalid input**

**continue;**

**}**

**switch (choice) {**

**case 1:**

**addExpense(scanner, expensesByCategory);**

**break;**

**case 2:**

**viewExpenses(expensesByCategory);**

**break;**

**case 3:**

**viewExpenseSummaries(expensesByCategory);**

**break;**

**case 4:**

**saveExpensesToFile(expensesByCategory);**

**System.out.println("Expenses saved to file.");**

**break;**

**case 5:**

**System.out.println("Exiting Expense Tracker. Goodbye!");**

**System.exit(0);**

**default:**

**System.out.println("Invalid choice. Please enter a valid option.");**

**} } }**

**private static void displayMenu() {**

**System.out.println("Expense Tracker Menu:");**

**System.out.println("1. Add Expense");**

**System.out.println("2. View Expenses");**

**System.out.println("3. Expense Summaries");**

**System.out.println("4. Save Expenses to File");**

**System.out.println("5. Exit");**

**}**

**private static void addExpense(Scanner scanner, Map<String, Double> expensesByCategory) {**

**System.out.print("Enter expense category: ");**

**String category = scanner.nextLine();**

**double amount;**

**while (true) {**

**try {**

**System.out.print("Enter expense amount: ");**

**amount = scanner.nextDouble();**

**scanner.nextLine(); // Consume the newline character**

**break;**

**} catch (InputMismatchException e) {**

**System.out.println("Invalid input. Please enter a valid number.");**

**scanner.nextLine(); // Consume the invalid input**

**} }**

**Date date;**

**while (true) {**

**try {**

**System.out.print("Enter expense date (yyyy-MM-dd): ");**

**String dateStr = scanner.nextLine();**

**date = dateFormat.parse(dateStr);**

**break;**

**} catch (ParseException e) {**

**System.out.println("Invalid date format. Please use the format yyyy-MM-dd.");**

**} }**

**expensesByCategory.put(category, expensesByCategory.getOrDefault(category, 0.0) + amount);**

**System.out.println("Expense added successfully!");**

**}**

**private static void viewExpenses(Map<String, Double> expensesByCategory) {**

**if (expensesByCategory.isEmpty()) {**

**System.out.println("No expenses recorded yet.");**

**return;**

**}**

**System.out.println("Expense Details:");**

**for (Map.Entry<String, Double> entry : expensesByCategory.entrySet()) {**

**System.out.println(entry.getKey() + ": $" + entry.getValue());**

**}**

**System.out.println("Total: $" + getTotalExpenses(expensesByCategory));**

**}**

**private static void viewExpenseSummaries(Map<String, Double> expensesByCategory) {**

**if (expensesByCategory.isEmpty()) {**

**System.out.println("No expenses recorded yet.");**

**return;**

**}**

**System.out.println("Expense Summaries:");**

**for (Map.Entry<String, Double> entry : expensesByCategory.entrySet()) {**

**System.out.println(entry.getKey() + ": $" + entry.getValue());**

**}**

**System.out.println("Total: $" + getTotalExpenses(expensesByCategory));**

**}**

**private static double getTotalExpenses(Map<String, Double> expensesByCategory) {**

**return expensesByCategory.values().stream().mapToDouble(Double::doubleValue).sum();**

**}**

**private static Map<String, Double> loadExpensesFromFile() {**

**Map<String, Double> expensesByCategory = new HashMap<>();**

**try (Scanner fileScanner = new Scanner(new File(DATA\_FILE))) {**

**while (fileScanner.hasNextLine()) {**

**String line = fileScanner.nextLine();**

**String[] parts = line.split(",");**

**if (parts.length == 2) {**

**String category = parts[0];**

**double amount = Double.parseDouble(parts[1]);**

**expensesByCategory.put(category, amount);**

**} } }**

**catch (FileNotFoundException e) {**

**System.out.println("No previous data found. Starting with an empty expense list.");**

**} catch (Exception e) {**

**System.out.println("Error loading expenses from file: " + e.getMessage());**

**}**

**return expensesByCategory;**

**}**

**private static void saveExpensesToFile(Map<String, Double> expensesByCategory) {**

**try (PrintWriter writer = new PrintWriter(DATA\_FILE)) {**

**for (Map.Entry<String, Double> entry : expensesByCategory.entrySet()) {**

**writer.println(entry.getKey() + "," + entry.getValue());**

**}**

**} catch (FileNotFoundException e) {**

**System.out.println("Error saving expenses to file: " + e.getMessage());**

**} } }**