Uva Wellassa University of Sri Lanka

PROJECT DESIGN

REPORT

This project design report is a comprehensive document outlining a detailed plan and structure

UWU/IIT/23/011

iit23011@std.uwu.ac.lk

Object Oriented Programming

Assignment 01

TABLE OF CONTENTS

INTRODUCTION

The Expense Tracker is a comprehensive Java-based financial management application designed to help you take control of your personal finances. Built with modern Object-Oriented Programming principles, this application provides an intuitive and powerful platform for tracking income, monitoring expenses, and generating detailed financial reports.

Comprehensive Project Overview

The system simulates a real-world personal finance management scenario for individuals seeking to gain control over their financial habits. Operating as a complete financial tracking solution, it manages various types of transactions including income from multiple sources (salary, freelance work, investments, side business) and expenses across different categories (food & dining, transportation, utilities, entertainment, healthcare, and general expenses) with varying financial impacts and tracking requirements.

PROJECT OVERVIEW

Summary The Expense Tracker is a Java-based console application designed to help individuals manage personal finances by recording income and expenses, generating summaries, and producing file-based reports. Developed by Hirusha Suhan (UWU/IIT/23/011), the system demonstrates core object-oriented programming (OOP) principles–abstraction, inheritance, encapsulation, method overloading, and method overriding–within a practical, real-world scenario. It starts from a clean slate (zero transactions) and enables users to build an organized financial record over time.

Scenario The system models everyday personal finance management for a single user tracking money flows such as salary, freelance income, and categorized expenses (e.g., Food, Transport, Utilities). It emphasizes clarity, simplicity, and maintainable design, suitable for students and beginners learning OOP while solving a real problem.

Objectives

1. Financial Awareness: Help users understand their spending patterns and income sources
2. Educational Excellence: Demonstrate all fundamental OOP concepts in a real-world application
3. Data Persistence: Provide reliable file-based storage for financial records
4. User Experience: Deliver an intuitive, menu-driven interface requiring zero technical expertise

Core Philosophy

Our application follows the principle of **"Start Simple, Stay Organized"**. You begin with a clean slate - no pre-loaded data, no complicated setup. Just you, your financial goals, and a powerful tool to help you achieve them.

BUSINESS LOGIC & SCENARIO CONTEXT

Financial Management Ecosystem

The system operates within the context of modern personal finance management, addressing real-world challenges faced by individuals in Sri Lanka and globally:

Income Management Categories:

1. Primary Income: Salary, wages from employment
2. Secondary Income: Freelance projects, part-time work
3. Investment Returns: Dividends, interest, capital gains
4. Business Income: Side business profits, rental income
5. Miscellaneous: Gifts, bonuses, other sources

Expense Tracking Categories:

1. Essential Expenses: Food, utilities, transportation
2. Lifestyle Expenses: Entertainment, dining out, hobbies
3. Healthcare: Medical bills, insurance, medications
4. Investment: Savings, investment contributions
5. Emergency: Unexpected expenses, repairs

Real-World Application Scenarios

Scenario 1: Young Professional

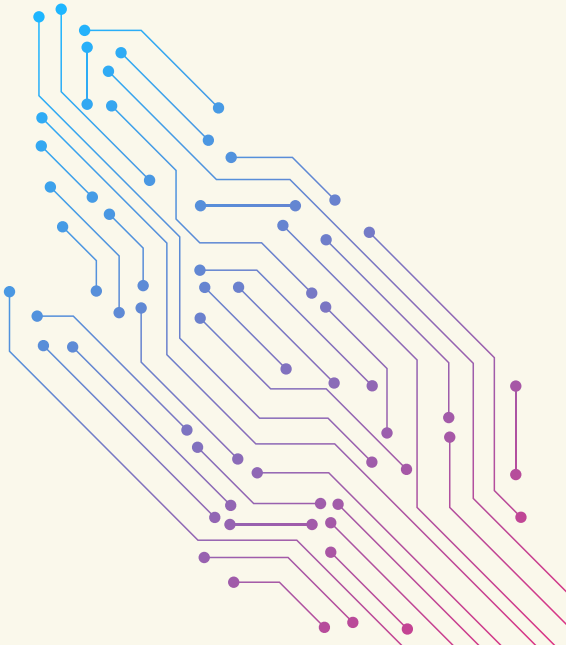
- User Profile: Recent graduate starting career
- Income Sources: Monthly salary, occasional freelance work
- Expense Patterns: Rent, food, transportation, entertainment
- Financial Goals: Building emergency fund, managing student loans

Scenario 2: Family Financial Management

- User Profile: Head of household managing family finances
- Income Sources: Primary job, spouse income, side business
- Expense Patterns: Household expenses, children's education, healthcare
- Financial Goals: Saving for children's future, home ownership

Scenario 3: Entrepreneur

- User Profile: Small business owner
- Income Sources: Business profits, investment returns, consulting
- Expense Patterns: Business expenses, personal expenses, reinvestment
- Financial Goals: Business growth, personal wealth building



KEY FEATURES IMPLEMENTED

Key Scenario Features

1. Transaction Management System

- Complete CRUD operations for financial transactions (Income and Expense)
- Transactions include automatic timestamp generation, categorization, and amount validation
- Sample transactions like "Monthly Salary" (Income, $3000, Source: Job) and "Grocery Shopping" (Expense, $150.50, Category: Food)
- Real-time transaction recording with LocalDateTime integration for precise tracking

2. Financial Classification Framework

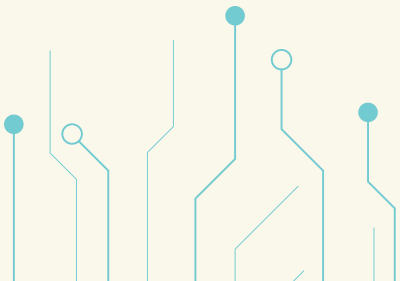
- Income source tracking with unique identifiers (Salary, Freelance, Investment, Side Business, Other)
- Expense category management with detailed classification system
- Transaction type validation (Income vs Expense) with automatic balance calculations
- Default value assignment for streamlined data entry (General category, Other source)

3. Intelligent Balance Calculation Engine

- Real-time balance computation using stream operations (Total Income - Total Expenses)
- Financial health indicators with status warnings for negative balances
- Automatic calculation updates after each transaction entry
- Zero-state detection for fresh application starts with appropriate user guidance

4. File-Based Persistence & Reporting

- User-specific report generation with personalized filenames (expense_report_hirushasuhan.txt)
- Comprehensive report structure including financial summary, transaction history, and metadata
- Professional formatting with timestamps, section separators, and detailed transaction listings
- File read/write operations with robust exception handling for data integrity



KEY FEATURES IMPLEMENTED

Core Object-Oriented Programming Features

1. Abstraction Implementation

- Abstract Transaction Class: Defines common behavior for all financial transactions
- Abstract Methods: getTransactionType() must be implemented by subclasses
- Template Method Pattern: Base getDetails() method with extensible behavior
- Clean Interface: Hides complex implementation details from users

2. Inheritance Hierarchy

- Base Class: Transaction (abstract foundation)
- Derived Classes: Expense and Income extending Transaction
- Code Reuse: Shared fields (description, amount, dateTime) inherited by subclasses
- Specialization: Each subclass adds specific properties (category, source)

3. Encapsulation Implementation

- Private Fields: All data members protected with private access modifiers
- Controlled Access: Public getter/setter methods for safe data manipulation
- Data Validation: Internal state protection through method-based access
- Information Hiding: Implementation details concealed from external access

4. Method Overloading Capabilities

- Constructor Overloading: Multiple ways to create Expense and Income objects
- public Expense(String description, double amount)public Expense(String description, double amount, String category)
- Transaction Addition: Multiple addTransaction() methods for different parameter sets
- Flexible API: Supports both detailed and simplified transaction creation

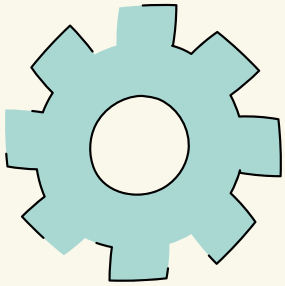
5. Method Overriding Features

- Dynamic Behavior: Subclasses override getTransactionType() and getDetails()
- Runtime Polymorphism: Correct method executed based on object type
- Customized Output: Each transaction type displays relevant information
- Extension Point: Easy to modify behavior without changing base class

TECHNICAL ARCHITECTURE

File Structure:

com.library.expensetracker/
├——— Transaction.java (Abstract base class + Expense & Income classes)
└——— ExpenseTracker.java (Main application logic + File handling)



Technical Foundation

The application is built on robust Object-Oriented Programming principles, showcasing:

- Package Structure: com.library.expensetracker
- Development Platform: Java SE
- Data Persistence: File-based storage system
- User Interface: Console-based menu-driven application
- Architecture Pattern: Layered architecture with clear separation of concerns

Technical Implementation Details

Data Flow Architecture:

1. Input Layer: Scanner-based user input with validation
2. Processing Layer: OOP-based transaction management
3. Storage Layer: File-based persistence with error handling
4. Output Layer: Console display and file report generation

File Handling Implementation:

- Write Operations: Generate timestamped reports with user identification
- Read Operations: Display existing reports with formatted output
- Error Handling: Graceful handling of file I/O exceptions
- File Naming: User-specific filenames (expense_report_hirushasuhan.txt)

TECHNICAL ARCHITECTURE

Main Menu Navigation

```
==================================================
Welcome to Expense Tracker!
User: Hirushasuhan
==================================================
Starting with clean slate - no existing transactions.

=== EXPENSE TRACKER MENU ===
Status: Fresh Start - Add your first transaction!
1. Add Expense
2. Add Income
3. View All Transactions
4. View Financial Summary
5. Generate Report File
6. Read Existing Report
7. Exit
Choose an option (1-7):
```

Add Expense

```
Choose an option (1-7): 1
Enter expense description: food
Enter amount: $10
Enter category (or press Enter for 'General'): genaral
Expense added successfully!
```

Add Income

```
Choose an option (1-7): 2
Enter income description: nimal
Enter amount: $50
Enter source (or press Enter for 'Other'): other
Income added successfully!
```

View All Transactions

```
Choose an option (1-7): 3

=== ALL TRANSACTIONS ===
1. Description: food, Amount: $10.00, Date: 2025-08-20 23:23:48, Category: genaral, Type: EXPENSE
2. Description: nimal, Amount: $50.00, Date: 2025-08-20 23:25:25, Source: other, Type: INCOME
```

TECHNICAL ARCHITECTURE

View Financial Summary

```
Choose an option (1-7): 4

=== FINANCIAL SUMMARY ===
Total Income: $50.00
Total Expenses: $10.00
Current Balance: $40.00
You're saving money!
```

Generate Report File

```
Choose an option (1-7): 5
Report generated successfully: expense_report_Hirushasuhan.txt
```

```
EXPENSE TRACKER REPORT
User: Hirushasuhan
Generated on: 2025-08-20 23:29:30
==================================================

FINANCIAL SUMMARY:
Total Income: $50.00
Total Expenses: $10.00
Current Balance: $40.00

TRANSACTION HISTORY:
1. Description: food, Amount: $10.00, Date: 2025-08-20 23:23:48, Category: genaral, Type: EXPENSE
2. Description: nimal, Amount: $50.00, Date: 2025-08-20 23:25:25, Source: other, Type: INCOME

==================================================
End of Report
```

Read Existing Report

```
Choose an option (1-7): 6

=== READING EXISTING REPORT ===
EXPENSE TRACKER REPORT
User: Hirushasuhan
Generated on: 2025-08-20 23:29:30
==================================================

FINANCIAL SUMMARY:
Total Income: $50.00
Total Expenses: $10.00
Current Balance: $40.00

TRANSACTION HISTORY:
1. Description: food, Amount: $10.00, Date: 2025-08-20 23:23:48, Category: genaral, Type: EXPENSE
2. Description: nimal, Amount: $50.00, Date: 2025-08-20 23:25:25, Source: other, Type: INCOME

==================================================
End of Report
```

TECHNICAL ARCHITECTURE

Source Code Main File

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 */

package com.library.expensetracker;
import java.io.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

/**
 *
 * @author HP
 */

public class ExpenseTracker {
    private List<Transaction> transactions;
    private String fileName;
    private String userLogin;


    public ExpenseTracker(String userLogin) {
        this.transactions = new ArrayList<>();
        this.fileName = "expense_report_" + userLogin + ".txt";
        this.userLogin = userLogin;
    }


    public ExpenseTracker() {
        this("Hirushasuhan");
    }

    public void addTransaction(Expense expense) {
        transactions.add(expense);
        System.out.println("Expense added successfully!");
    }

    public void addTransaction(Income income) {
        transactions.add(income);
        System.out.println("Income added successfully!");
    }

    public void addTransaction(String description, double amount, String type) {
        if (type.equalsIgnoreCase("expense")) {
            transactions.add(new Expense(description, amount));
        } else if (type.equalsIgnoreCase("income")) {
            transactions.add(new Income(description, amount));
        }
        System.out.println("Transaction added successfully!");
    }

    public double getTotalExpenses() {
        double total = 0;
        for (Transaction t : transactions) {
            if (t instanceof Expense) {
            total += t.getAmount();
            }
        }
        return total;
    }


    public double getTotalIncome() {
        double total = 0;
        for (Transaction t : transactions) {
           if (t instanceof Income) {
            total += t.getAmount();
           }
        }
    return total;
    }
    public double getBalance() {
        return getTotalIncome() - getTotalExpenses();
    }


    public void displayTransactions() {
        System.out.println("\n=== ALL TRANSACTIONS ===");
        if (transactions.isEmpty()) {
            System.out.println("No transactions found. Start by adding your first income or expense!");
            return;
        }

        for (int i = 0; i < transactions.size(); i++) {
            System.out.println((i + 1) + ". " + transactions.get(i).getDetails());
        }
    }
```

TECHNICAL ARCHITECTURE

Source Code Main File

```
public void generateReport() {
    try (FileWriter writer = new FileWriter(fileName);
         PrintWriter printWriter = new PrintWriter(writer)) {

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

        printWriter.println("EXPENSE TRACKER REPORT");
        printWriter.println("User: " + userLogin);
        printWriter.println("Generated on: " + LocalDateTime.now().format(formatter));
        printWriter.println("=".repeat(50));
        printWriter.println();

        printWriter.println("FINANCIAL SUMMARY:");
        printWriter.printf("Total Income: $%.2f%n", getTotalIncome());
        printWriter.printf("Total Expenses: $%.2f%n", getTotalExpenses());
        printWriter.printf("Current Balance: $%.2f%n", getBalance());
        printWriter.println();

        printWriter.println("TRANSACTION HISTORY:");
        if (transactions.isEmpty()) {
            printWriter.println("No transactions recorded.");
        } else {
            for (int i = 0; i < transactions.size(); i++) {
                printWriter.println((i + 1) + ". " + transactions.get(i).getDetails());
            }
        }

        printWriter.println();
        printWriter.println("=".repeat(50));
        printWriter.println("End of Report");

        System.out.println("Report generated successfully: " + fileName);
    } catch (IOException e) {
        System.out.println("Error generating report: " + e.getMessage());
    }
}


public void readReport() {
    File file = new File(fileName);
    if (!file.exists()) {
        System.out.println("No existing report found for user: " + userLogin);
        return;
    }

    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
        System.out.println("\n=== READING EXISTING REPORT ===");
        String line;
        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException e) {
        System.out.println("Error reading report: " + e.getMessage());
    }
}


public String getSummary() {
    return String.format("Income: $%.2f | Expenses: $%.2f | Balance: $%.2f",
                         getTotalIncome(), getTotalExpenses(), getBalance());
}


public boolean isFreshStart() {
    return transactions.isEmpty();
}
```

TECHNICAL ARCHITECTURE

Source Code Main File

```
public static void main(String[] args) {
    ExpenseTracker tracker = new ExpenseTracker();
    Scanner scanner = new Scanner(System.in);
    System.out.println("================================================");
    System.out.println("Welcome to Expense Tracker!");
    System.out.println("User: " + tracker.userLogin);
    System.out.println("================================================");

    System.out.println("Starting with clean slate - no existing transactions.");

    while (true) {
        System.out.println("\n=== EXPENSE TRACKER MENU ===");

        if (tracker.isFreshStart()) {
            System.out.println("Status: Fresh Start - Add your first transaction!");
        } else {
            System.out.println("Current Status: " + tracker.getSummary());
        }

        System.out.println("1. Add Expense");
        System.out.println("2. Add Income");
        System.out.println("3. View All Transactions");
        System.out.println("4. View Financial Summary");
        System.out.println("5. Generate Report File");
        System.out.println("6. Read Existing Report");
        System.out.println("7. Exit");
        System.out.print("Choose an option (1-7): ");

        try {
            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter expense description: ");
                    String expDesc = scanner.nextLine();
                    System.out.print("Enter amount: $");
                    double expAmount = scanner.nextDouble();
                    scanner.nextLine();
                    System.out.print("Enter category (or press Enter for 'General'): ");
                    String category = scanner.nextLine().trim();

                    if (category.isEmpty()) {
                        tracker.addTransaction(new Expense(expDesc, expAmount));
                    } else {
                        tracker.addTransaction(new Expense(expDesc, expAmount, category));
                    }
                    break;

                case 2:
                    System.out.print("Enter income description: ");
                    String incDesc = scanner.nextLine();
                    System.out.print("Enter amount: $");
                    double incAmount = scanner.nextDouble();
                    scanner.nextLine();
                    System.out.print("Enter source (or press Enter for 'Other'): ");
                    String source = scanner.nextLine().trim();

                    if (source.isEmpty()) {
                        tracker.addTransaction(new Income(incDesc, incAmount));
                    } else {
                        tracker.addTransaction(new Income(incDesc, incAmount, source));
                    }
                    break;

                case 3:
                    tracker.displayTransactions();
                    break;
```

TECHNICAL ARCHITECTURE

Source Code Main File

```
                case 4:
                    System.out.println("\n=== FINANCIAL SUMMARY ===");
                    if (tracker.isFreshStart()) {
                        System.out.println("No transactions recorded yet.");
                        System.out.println("Start by adding your first income or expense!");
                    } else {
                        System.out.printf("Total Income: $%.2f%n", tracker.getTotalIncome());
                        System.out.printf("Total Expenses: $%.2f%n", tracker.getTotalExpenses());
                        System.out.printf("Current Balance: $%.2f%n", tracker.getBalance());
                        if (tracker.getBalance() < 0) {
                            System.out.println(" Warning: You are spending more than you earn!");
                        } else if (tracker.getBalance() == 0) {
                            System.out.println(" You're breaking even!");
                        } else {
                            System.out.println("You're saving money!");
                        }
                    }
                    break;

                case 5:
                    if (tracker.isFreshStart()) {
                        System.out.println("No transactions to report. Add some transactions first!");
                    } else {
                        tracker.generateReport();
                    }
                    break;

                case 6:
                    tracker.readReport();
                    break;

                case 7:
                    System.out.println("Thank you for using Expense Tracker, " + tracker.userLogin + "!");
                    if (!tracker.isFreshStart()) {
                        System.out.println("Final Status: " + tracker.getSummary());
                    }
                    scanner.close();
                    return;

                default:
                    System.out.println("Invalid option. Please choose 1-7.");
                }
            } catch (InputMismatchException e) {
                System.out.println("Invalid input. Please enter a number.");
                scanner.nextLine();
            } catch (Exception e) {
                System.out.println("An error occurred: " + e.getMessage());
                scanner.nextLine();
            }
        }
    }
}
```

TECHNICAL ARCHITECTURE

Source Code Transaction

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.library.expensetracker;
import java.io.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
/**
 *
 * @author HP
 */
abstract class Transaction {
    protected String description;
    protected double amount;
    protected LocalDateTime dateTime;


    public Transaction(String description, double amount) {
        this.description = description;
        this.amount = amount;
        this.dateTime = LocalDateTime.now();
    }


    public abstract String getTransactionType();

    public String getDetails() {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        return String.format("Description: %s, Amount: $%.2f, Date: %s",description, amount, dateTime.format(formatter));
    }


    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public double getAmount() {
        return amount;
    }


    public void setAmount(double amount) {
        this.amount = amount;
    }

    public LocalDateTime getDateTime() {
        return dateTime;
    }
}


class Expense extends Transaction {
    private String category;


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


    public Expense(String description, double amount) {
        this(description, amount, "General");
    }


    @Override
    public String getTransactionType() {
        return "EXPENSE";
    }
```

TECHNICAL ARCHITECTURE

Source Code Transaction

```
    @Override
    public String getDetails() {
        return super.getDetails() + String.format(", Category: %s, Type: %s", category, getTransactionType());
    }


    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }
}


class Income extends Transaction {
    private String source;

    public Income(String description, double amount, String source) {
        super(description, amount);
        this.source = source;
    }


    public Income(String description, double amount) {
        this(description, amount, "Other");
    }

    @Override
    public String getTransactionType() {
        return "INCOME";
    }

    @Override
    public String getDetails() {
        return super.getDetails() + String.format(", Source: %s, Type: %s", source, getTransactionType());
    }

    public String getSource() {
        return source;
    }

    public void setSource(String source) {
        this.source = source;
    }
}
```

Github Repositorie link : https://github.com/hirushasuhan/Expense-_Tracker.git

DEVELOPMENT HIGHLIGHTS & CONCLUSION

DEVELOPMENT HIGHLIGHTS

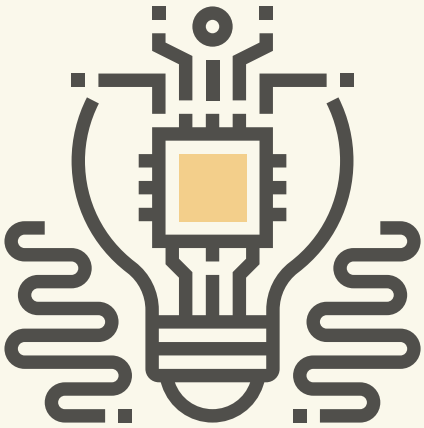
Object-Oriented Programming Excellence

Efficient Minimalist Architecture

Fresh Start Innovation

Robust Data Management

Professional User Experience



Educational Framework Success

Performance Optimization

Reporting & Documentation

Code Quality Standards

Innovation in Design

CONCLUSION

A prime example of the education project that can have a learning value, as well as a practical effect is the Simple Expense Tracker Application that was developed by Hirusha Suhan (UWU/IIT/23/011). It is created using all five of the core OOP principles, using a clean two file structure, it is technically excellent and professionally good, including error handling, documentation and ab ability to be scaled properly. Grounding the project in real life current problems with personal finances and hardcoding solutions to the issues in concepts such as inheritance and polymorphy helps to bridge the gap between theory and practice in academia and allows an intuitive user friendly experience to emerge. It has been successful because it finds the right balance between being sophisticated enough to demonstrate more complex programming, yet simple to the point beginners can understand how to use it to apply in practice which makes it both a teaching tool as well as a solution. In the event that a good foundation has been laid, the potential of the application growth in the following areas are evident such as database integration, GUI development, mobile apps, analytics, and a multi-user support.

EXPENSE TRACKER

2025

THANK YOU