ระบบรายรับรายจ่าย

(Income & Expense System)

นางสาวกชกร กิติมา 672110132

960242 Digital Industry Solution 3

อาจารย์กฤตย์ จินกลับ

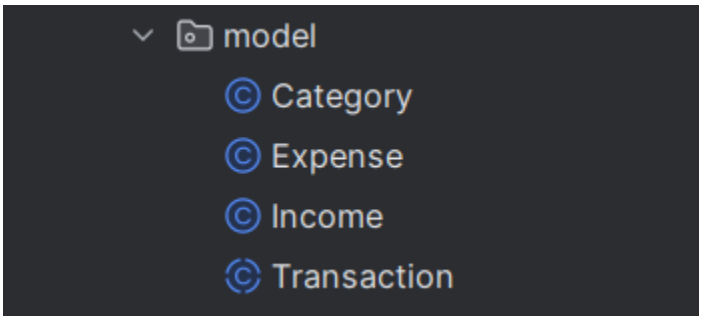
โปรเจกต์นี้พัฒนาขึ้นโดยใช้ภาษา Java และยึดตามแนวคิดของ **Object-Oriented Programming (OOP)** เพื่อให้โค้ดมีโครงสร้างที่ชัดเจนและยืดหยุ่นต่อการปรับปรุงในอนาคต นอกจากนี้ยังมีการนำ **Design Patterns** มาใช้ในกระบวนการออกแบบ เพื่อให้ระบบสามารถแยกความรับผิดชอบของแต่ละส่วนได้อย่างเป็นระบบ บันทึกข้อมูลถาวรแบบไฟล์ CSV และมีหน้าจอ GUI ที่ใช้งานง่าย ผู้ใช้สามารถเพิ่ม แก้ไข ลบ และค้นหารายการธุรกรรม พร้อมแสดงยอดเงินคงเหลือและสรุปรายรับรายจ่ายในแต่ละช่วงเวลาได้

วัตถุประสงค์

1. เพื่อสร้างระบบสำหรับบันทึกรายรับรายจ่ายของผู้ใช้งาน
2. เพื่อศึกษาและประยุกต์ใช้หลัก OOP
3. เพื่อฝึกออกแบบ GUI ด้วย Java Swing
4. เพื่อฝึกใช้ Design Pattern เพื่อเพิ่มความยืดหยุ่นของโค้ด

โครงสร้างระบบ และการแบ่ง Package

1. **model – โมเดลข้อมูลของระบบ**



หน้าที่

เก็บคลาสที่เป็นข้อมูลหลักของระบบ (Domain Model) เช่น ธุรกรรมแต่ละรายการ

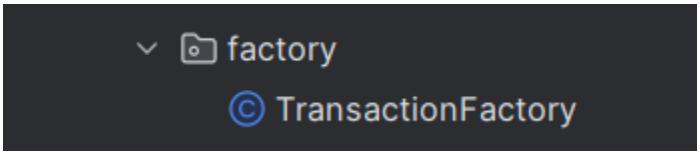
ประกอบด้วยคลาส

- **Transaction** (abstract class) โครงสร้างของธุรกรรม (รายรับหรือรายจ่าย)
- **Income** คลาสสำหรับธุรกรรมประเภท "รายรับ"
- **Expense** คลาสสำหรับธุรกรรมประเภท "รายจ่าย"
- **Category** เก็บหมวดหมู่ของธุรกรรมในรูปแบบคงที่ (static final array)

ลักษณะ

- ใช้หลัก Inheritance (Income และ Expense สืบทอดจาก Transaction)
- ห่อหุ้มข้อมูล (Encapsulation) ผ่าน protected field และ getter methods

2. **factory – สร้างอ็อบเจกต์ธุรกรรม**



หน้าที่

สร้างอ็อบเจกต์ Income หรือ Expense โดยไม่ให้โค้ดฝั่ง UI ต้องรู้รายละเอียดภายใน

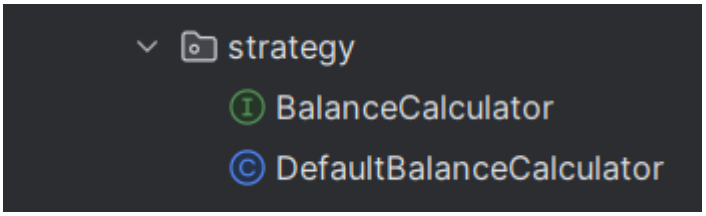
ประกอบด้วยคลาส

- **TransactionFactory** มีเมธอด static สำหรับสร้าง Transaction ตามประเภทที่กำหนด

ลักษณะ

- ใช้ **Factory Pattern** เพื่อแยก logic การสร้างอ็อบเจกต์ออกจากผู้เรียกใช้งาน

3. **strategy – คำนวณยอดเงินคงเหลือ**



หน้าที่

แยก logic การคำนวณยอดเงิน (Balance) ออกจากคลาสหลัก

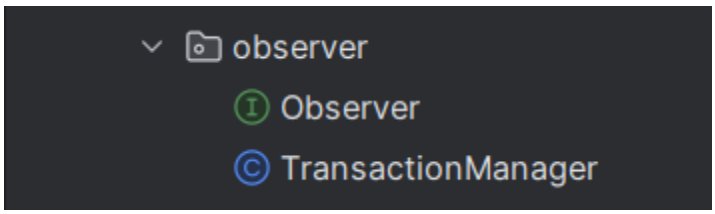
ประกอบด้วยคลาส

- **BalanceCalculator** มี Interface ที่กำหนดสัญญาการคำนวณยอดเงิน
- **DefaultBalanceCalculator** คำนวณยอดโดยใช้รายรับ - รายจ่าย

ลักษณะ

- ใช้ Strategy Pattern เพื่อให้เปลี่ยนวิธีคำนวณได้โดยไม่กระทบคลาสอื่น

4. **observer — ระบบแจ้งเตือน (Observer Pattern)**



หน้าที่

แจ้งให้ GUI ทราบเมื่อข้อมูลธุรกรรมมีการเปลี่ยนแปลง เช่น เพิ่ม ลบ แก้ไข

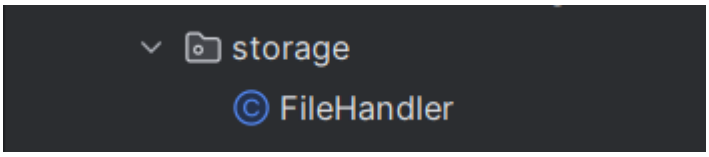
ประกอบด้วยคลาส

- **Observer** มี Interface สำหรับคลาสที่ต้องการติดตามการเปลี่ยนแปลง
- **TransactionManager** เป็น Subject ที่จัดการรายการธุรกรรมทั้งหมด และแจ้ง Observer

ลักษณะ

- ใช้ **Observer Pattern** ให้ GUI อัปเดตข้อมูลโดยอัตโนมัติ

5. **storage – การจัดเก็บและโหลดข้อมูลจากไฟล์**



หน้าที่

จัดการการบันทึกและอ่านข้อมูลธุรกรรมจากไฟล์ .csv

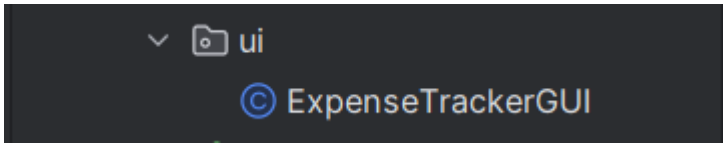
ประกอบด้วยคลาส

- **FileHandler** มีเมธอด static สำหรับ saveTransactions() และ loadTransactions()

ลักษณะ

- ใช้ไฟล์ transactions.csv สำหรับเก็บข้อมูลอย่างถาวร
- แยกออกจาก logic ของ UI และ TransactionManager เพื่อให้โค้ดสะอาดและจัดการง่าย

6. **Ui - ส่วนของกราฟิกผู้ใช้ (Graphical User Interface)**



หน้าที่

ดูแลการแสดงผลหน้าจอ, รับข้อมูลจากผู้ใช้ และเชื่อมต่อกับระบบภายใน

ประกอบด้วยคลาส

- **ExpenseTrackerGUI** คลาสหลักของ GUI ที่สื่อสารกับผู้ใช้
- รับข้อมูลจากผู้ใช้ และส่งไปยัง TransactionManager เพื่อจัดการ
- แสดงผลธุรกรรม, ค้นหา, สรุปรายวัน/เดือน/ปี และแสดงยอดเงินคงเหลือ

ลักษณะ

- ใช้ Java Swing ในการสร้าง GUI
- ทำงานร่วมกับ Observer เพื่อให้ GUI อัปเดตแบบ Real-time
- รองรับการกรอก/แก้ไข/ค้นหา/สรุป/ลบข้อมูลผ่านปุ่มในหน้าจอเดียว

หลักการ OOP ที่นำมาใช้

1. Encapsulation (การห่อหุ้มข้อมูล)

แนวคิด

ซ่อนรายละเอียดของข้อมูลและการทำงานภายในไว้ และเปิดเผยเฉพาะสิ่งที่จำเป็นผ่านเมธอด

การใช้งานในโปรเจกต์

คลาส Transaction (abstract class) กำหนด description, amount, date, category ไว้เป็น protected เพื่อไม่ให้ถูกเข้าถึงตรงๆ จากภายนอกและเปิดให้เข้าถึงผ่าน getter methods เช่น getDescription(), getAmount() เป็นต้น

```java
public abstract class Transaction {  26 usages  2 inheritors
    protected String description;  2 usages
    protected double amount;  2 usages
    protected Date date;  2 usages
    protected String category;  2 usages

    public Transaction(String description, double amount, Date date, String category) {  2 usages
        this.description = description;
        this.amount = amount;
        this.date = date;
        this.category = category;
    }

    public abstract String getType();  4 usages  2 implementations

    public String getDescription() {  4 usages
        return description;
    }
    public double getAmount() {  8 usages
        return amount;
    }
    public String getCategory() {  2 usages
        return category;
    }
    public Date getDate() {  5 usages
        return date;
    }
```

TransactionManager ทำหน้าที่เป็นตัวกลางในการจัดการข้อมูล เช่น เพิ่ม ลบ แก้ไขธุรกรรม โดยไม่ให้ GUI เข้าถึง transactions โดยตรง

ข้อดีที่ได้จากการใช้ Encapsulation

- เพิ่มความปลอดภัยให้กับข้อมูล
- ลดโอกาสที่ข้อมูลจะถูกเปลี่ยนแปลงโดยไม่ตั้งใจ
- ควบคุมการเข้าถึงและตรวจสอบข้อมูลได้ง่ายขึ้น

2. **Inheritance (การสืบทอดคลาส)**

แนวคิด

คลาสลูกสามารถสืบทอดพฤติกรรมและข้อมูลจากคลาสแม่

การใช้งานในโปรเจกต์

คลาส Income และ Expense สืบทอดจาก Transaction ทำให้สามารถใช้โครงสร้างข้อมูลร่วมกันได้ เช่น description, amount, date, category และOverride เมธอด getType() เพื่อระบุประเภทธุรกรรม

```
public class Income extends Transaction {  8 usages
    public Income(String description, double amount, Date date, String category) {  4 usages
        super(description, amount, date, category);
    }

    @Override  4 usages
    public String getType() {
        return "Income";
    }
}
```

```
public class Expense extends Transaction {  7 usages
    public Expense(String description, double amount, Date date, String category) {  5 usages
        super(description, amount, date, category);
    }

    @Override  4 usages
    public String getType() {
        return "Expense";
    }
}
```

ข้อดีที่ได้จากการใช้ Inheritance

- ลดการเขียนโค้ดซ้ำ
- ทำให้สามารถจัดการข้อมูลธุรกรรมทั้งรายรับและรายจ่ายด้วยรูปแบบเดียวกัน
- สามารถ override เมธอด getType() เพื่อแสดงประเภทที่แตกต่างกันได้

3. Polymorphism (พอลิมอร์ฟิซึม)

แนวคิด

อ็อบเจ็กต์หลายชนิดสามารถเรียกใช้เมธอดเดียวกันได้ และมีพฤติกรรมต่างกันตามชนิดของอ็อบเจ็กต์

การใช้งานในโปรเจกต์

- ใช้ Transaction เป็นตัวแทนของทั้ง Income และ Expense ในลิสต์ List<Transaction>
- ใช้ TransactionFactory สร้างอ็อบเจ็กต์ Transaction โดยไม่ต้องรู้ว่าจะเป็นคลาสใด (ใช้ polymorphism)
- ใช้ BalanceCalculator เป็น Interface แล้วให้คลาส DefaultBalanceCalculator เป็นคลาสที่ทำงานจริง

ExpenseTrackerGUI.java

```
manager.addTransaction(TransactionFactory.createTransaction(type, description, amt, new Date(), category));
```

ใช้ Factory เพื่อสร้าง Transaction แล้วส่งไปยัง TransactionManager
ซึ่งใช้ polymorphism แบบเต็ม ๆ: สร้าง Income หรือ Expense แบบ dynamic

TransactionManager.java

```
private List<Transaction> transactions;  12 usages
```

เก็บ Income และ Expense ไว้ใน List<Transaction> ตัวเดียว ซึ่งใช้ polymorphism

ข้อดีที่ได้จากการใช้ Polymorphism

- ทำให้โค้ดมีความยืดหยุ่น (Flexible)
- เพิ่มความสามารถในการรองรับการเปลี่ยนแปลงในอนาคต เช่น เพิ่มประเภทธุรกรรมใหม่โดยไม่กระทบโค้ดเดิม

4. Abstraction

แนวคิด

การซ่อนรายละเอียดการทำงานภายใน และเปิดเผยเฉพาะสิ่งที่จำเป็นต่อผู้ใช้งานผ่าน Interface หรือ Abstract class

การใช้งานในโปรเจกต์

- คลาส Transaction เป็น abstract class ที่บังคับให้คลาสลูกต้องกำหนด getType() เอง
- Interface BalanceCalculator ถูกใช้เพื่อแยก logic การคำนวณยอดเงินออกจากคลาสที่จัดการธุรกรรม

```
public abstract String getType();   4 usages  2 implementations
```

```
public interface BalanceCalculator {  4 usages  1 implementation
    double calculateBalance(List<Transaction> transactions);  1 usage  1 implementation
}
```

ข้อดีที่ได้จากการใช้ Abstraction

- ทำให้ระบบมีโครงสร้างที่ชัดเจน
- สามารถเปลี่ยนวิธีคำนวณยอดเงินได้ง่าย (เช่น เพิ่ม Strategy ใหม่)
- ลดการผูกติดกันของคลาส (Low Coupling)

การใช้ Design Pattern

1. Factory Pattern

แนวคิด

แยกกระบวนการสร้างอ็อบเจกต์ออกจากการใช้งานจริง โดยให้คลาส Factory รับผิดชอบในการสร้างอ็อบเจกต์แทน เพื่อให้โค้ดยืดหยุ่นและง่ายต่อการขยาย

การประยุกต์ใช้ในโปรเจกต์

ในระบบนี้คลาส TransactionFactory ทำหน้าที่เป็นผู้สร้างอ็อบเจกต์ Income หรือ Expense ตามประเภทที่ผู้ใช้เลือก โดยไม่ให้ฝั่ง GUI ต้องรู้ว่ากำลังสร้างอ็อบเจกต์จากคลาสไหน

TransactionFactory.java

```
public class TransactionFactory {  3 usages
    public static Transaction createTransaction(String type, String description, double amount, Date date, String category) {  2 usages
        if (type.equalsIgnoreCase( anotherString: "Income")) {
            return new Income(description, amount, date, category);
        } else if (type.equalsIgnoreCase( anotherString: "Expense")) {
            return new Expense(description, amount, date, category);
        }
        return null;
    }
}
```

ประกาศในคลาส ExpenseTrackerGUI

```
manager.addTransaction(TransactionFactory.createTransaction(type, description, amt, new Date(), category));
```

และถูกเรียกใช้จากปุ่มใน GUI

```
addIncome.addActionListener(e -> addTransaction( type: "Income", descField.getText(), amountField.getText(), categoryBox.getSelectedItem().toString()));
addExpense.addActionListener(e -> addTransaction( type: "Expense", descField.getText(), amountField.getText(), categoryBox.getSelectedItem().toString()));
```

2. Strategy Pattern

แนวคิด

แยก logic การทำงานที่สามารถเปลี่ยนแปลงได้ออกจากคลาสหลัก โดยให้เป็น Interface และเปลี่ยนได้ตามการทำงานที่ต้องการใช้งาน

การประยุกต์ใช้ในโปรเจกต์

ระบบใช้ BalanceCalculator เป็นอินเทอร์เฟซในการคำนวณยอดเงินคงเหลือ ซึ่งสามารถเปลี่ยนวิธีคำนวณได้โดยไม่ต้องแก้โค้ดใน TransactionManager เลย

BalanceCalculator.java (Interface)

```
public interface BalanceCalculator {  4 usages  1 implementation
    double calculateBalance(List<Transaction> transactions);  1 usage  1 implementation
}
```

DefaultBalanceCalculator.java (การทำงานเริ่มต้น)

```
public class DefaultBalanceCalculator implements BalanceCalculator {  2 usages
    @Override  1 usage
    public double calculateBalance(List<Transaction> transactions) {
        double balance = 0;
        for (Transaction t : transactions) {
            if (t instanceof Income) {
                balance += t.getAmount();
            } else if (t instanceof Expense) {
                balance -= t.getAmount();
            }
        }
        return balance;
    }
}
```

TransactionManager.java

```
    private BalanceCalculator balanceCalculator;  2 usages
public TransactionManager(BalanceCalculator balanceCalculator) {  1 usage
    this.balanceCalculator = balanceCalculator;
```

3. Observer Pattern

แนวคิด

ทำให้คลาสหนึ่ง (Subject) แจ้งเตือนไปยังคลาสอื่น ๆ (Observer) เมื่อมีการเปลี่ยนแปลงข้อมูล โดยไม่ต้องรู้ว่า Observer มีอยู่กี่ตัว

การประยุกต์ใช้ในโปรเจกต์

ใช้ให้ GUI (ExpenseTrackerGUI) อัปเดตอัตโนมัติเมื่อมีการเพิ่ม แก้ไข หรือลบธุรกรรมใน TransactionManager

Observer.java

```
public interface Observer {  5 usages  1 implementation
    void update();  5 usages  1 implementation
}
```

TransactionManager.java

```
private List<Observer> observers = new ArrayList<>();  2 usages
```

```
public void addTransaction(Transaction transaction) {  1 usage
    transactions.add(transaction);
    saveAndUpdate();
}
```

```
private void notifyObservers() {  1 usage
    for (Observer observer : observers) {
        observer.update();
    }
}
```

ในคลาส GUI (ExpenseTrackerGUI.java)

```
public class ExpenseTrackerGUI extends JFrame implements Observer {  1 usage
```

```
    @Override  5 usages
    public void update() {
        if (listModel == null || balanceLabel == null) return;

        balanceLabel.setText("Balance: " + manager.getBalance() + " Baht");

        listModel.clear();
        for (Transaction t : manager.getTransactions()) {
            listModel.addElement(dateFormat.format(t.getDate()) + " | " + t.getType() + ": " + t.getDescription() + " (" + t.getAmount() + " Baht)");
        }
    }
}
```

ขั้นตอนการทำงานของระบบ (Flow Diagram)

1. ผู้ใช้กรอกข้อมูลใน GUI

- กรอกรายละเอียดธุรกรรม: คำอธิบาย (Description), จำนวนเงิน (Amount), ประเภท (รายรับ/รายจ่าย), หมวดหมู่ (Category)
- คลิกปุ่ม “Income” หรือ “Expense” เพื่อเพิ่มรายการ

2. GUI เรียก TransactionFactory เพื่อสร้างอ็อบเจกต์

- GUI ส่งข้อมูลไปให้ TransactionFactory.createTransaction(...)
- Factory ตรวจสอบประเภท (Income/Expense) และสร้างอ็อบเจกต์ Income หรือ Expense ตามที่เลือก
- Factory ส่งคืนอ็อบเจกต์ที่สร้างแล้วกลับไปให้ GUI

3. GUI ส่งอ็อบเจกต์ไปให้ TransactionManager

- GUI เรียก manager.addTransaction(transaction)
- Manager จะเพิ่มธุรกรรมลงใน List ที่เก็บข้อมูลธุรกรรมทั้งหมด

4. TransactionManager บันทึกข้อมูลลงไฟล์

- Manager เรียก FileHandler.saveTransactions(transactions) เพื่อเขียนรายการทั้งหมดลงไฟล์ transactions.csv ทันที
- การบันทึกจะทำทุกครั้งที่มีการเพิ่ม ลบ หรือแก้ไขข้อมูล

5. TransactionManager แจ้ง Observer ให้อัปเดต

- Manager เรียก notifyObservers() เพื่อแจ้ง GUI ว่าข้อมูลเปลี่ยนแล้ว
- GUI ที่เป็น Observer ได้ลงทะเบียนไว้ก่อนหน้า จะถูกเรียกเมธอด update()
- GUI จะรีเฟรชหน้าจอใหม่: แสดงรายการธุรกรรมล่าสุด + ยอดเงินคงเหลือใหม่

หมายเหตุเพิ่มเติม

- หากเป็นการ แก้ไข (Edit) หรือ ลบ (Delete) ธุรกรรม ระบบจะเข้าสู่ขั้นตอนเดียวกันในข้อ 4–5: บันทึกไฟล์ใหม่ และอัปเดต GUI
- ขั้นตอนการ ค้นหา หรือ สรุปรายเดือน/ปี จะไม่เขียนไฟล์ แต่จะดึงข้อมูลจาก List ที่โหลดมาจากไฟล์แล้วมาคำนวณและแสดงผลทันที

ฟีเจอร์ของโปรแกรม

1. เพิ่มธุรกรรม (Add Transaction)

คำอธิบาย

ผู้ใช้สามารถกรอกรายละเอียดของธุรกรรม เช่น รายการ (description), จำนวนเงิน, หมวดหมู่ และเลือกประเภทว่าเป็นรายรับ (Income) หรือรายจ่าย (Expense) จากนั้นกดปุ่มเพิ่มรายการ

วิธีการทำงานของระบบ

- ระบบใช้ TransactionFactory เพื่อสร้างอ็อบเจกต์ Income หรือ Expense
- ส่งไปที่ TransactionManager เพื่อเก็บไว้ใน List และบันทึกลงไฟล์ CSV
- แจ้ง GUI ให้อัปเดตหน้าจอทันทีผ่าน Observer Pattern

2. ลบธุรกรรม (Delete Transaction)

คำอธิบาย

ผู้ใช้สามารถคลิกรายการที่ต้องการลบจากรายการธุรกรรมทั้งหมด แล้วกดปุ่ม “Delete” เพื่อทำการลบ

วิธีการทำงานของระบบ

- เรียก manager.removeTransaction(index)

- บันทึกไฟล์ใหม่ และแจ้ง Observer ให้อัปเดต GUI

3. แก้ไขธุรกรรม (Edit Transaction)

คำอธิบาย

เมื่อผู้ใช้เลือกธุรกรรมรายการใดรายการหนึ่งจากหน้าจอ และเปลี่ยนแปลงข้อมูลในฟอร์ม จากนั้นกดปุ่ม “Edit” ระบบจะอัปเดตรายการนั้น

วิธีการทำงานของระบบ

- เรียก manager.editTransaction(index, updatedTransaction)
- บันทึกข้อมูลใหม่ลงไฟล์ CSV
- GUI อัปเดตทันทีผ่าน Observer

4. ค้นหาธุรกรรม (Search Transaction)

คำอธิบาย

ผู้ใช้สามารถพิมพ์คำค้น เช่น คำในรายการ หรือจำนวนเงิน เพื่อค้นหาธุรกรรมที่เกี่ยวข้องทั้งหมด

วิธีการทำงานของระบบ

- ระบบใช้เมธอด manager.searchTransactions(keyword) เพื่อกรองจาก List
- เปรียบเทียบกับ description, category, และ amount
- แสดงผลรายการที่ตรงกับคำค้นใน JList

5. แสดงยอดเงินคงเหลือ (Current Balance)

คำอธิบาย

ระบบจะแสดงยอดเงินคงเหลือที่คำนวณจาก “รายรับ - รายจ่าย” ทั้งหมดที่บันทึกไว้ในระบบแบบเรียลไทม์

วิธีการทำงานระบบ

- ใช้ BalanceCalculator (Strategy Pattern) เป็นตัวคำนวณ
- ยอดเงินคำนวณใหม่ทุกครั้งเมื่อเพิ่ม/ลบ/แก้ไขธุรกรรม
- GUI แสดงใน balanceLabel

6. สรุปรายรับรายจ่ายรายวัน / รายเดือน / รายปี

คำอธิบาย

ผู้ใช้สามารถดูสรุปยอดเงินรายวัน รายเดือน หรือรายปี โดยระบบจะคำนวณผลรวมรายรับลบรายจ่ายในช่วงเวลานั้น ๆ และแสดงผลในตาราง

วิธีการทำงานของระบบ

- TransactionManager มีเมธอด getDailySummary(), getMonthlySummary(), getYearlySummary()
- ใช้การจัดกลุ่มข้อมูลด้วย Stream + Collectors.groupingBy(...)
- GUI แสดงผลใน JTable ผ่าน Popup Dialog

7. บันทึกข้อมูลถาวรลงไฟล์ CSV อัตโนมัติ

คำอธิบาย

ทุกธุรกรรมจะถูกบันทึกลงในไฟล์ transactions.csv โดยอัตโนมัติทุกครั้งที่มีการเปลี่ยนแปลงข้อมูล

วิธีการทำงานของระบบ

- คลาส FileHandler มีเมธอด saveTransactions() และ loadTransactions()

- ใช้ BufferedWriter และ BufferedReader สำหรับเขียน/อ่านไฟล์
- ฟอร์แมตไฟล์เป็นแบบ CSV (Comma-Separated Values)

8. โหลดข้อมูลจากไฟล์อัตโนมัติเมื่อเปิดโปรแกรม

คำอธิบาย

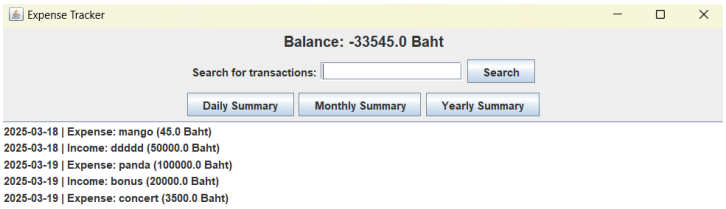
เมื่อเปิดโปรแกรม ระบบจะโหลดข้อมูลธุรกรรมที่เคยบันทึกไว้ทั้งหมดขึ้นมาแสดงในหน้าจอทันที

วิธีการทำงานของระบบ

- TransactionManager เรียก FileHandler.loadTransactions() ใน constructor
- ข้อมูลในไฟล์จะถูกแปลงเป็นอ็อบเจกต์ Income / Expense และเก็บไว้ใน List

ตัวอย่างหน้าจอ (Screenshot)

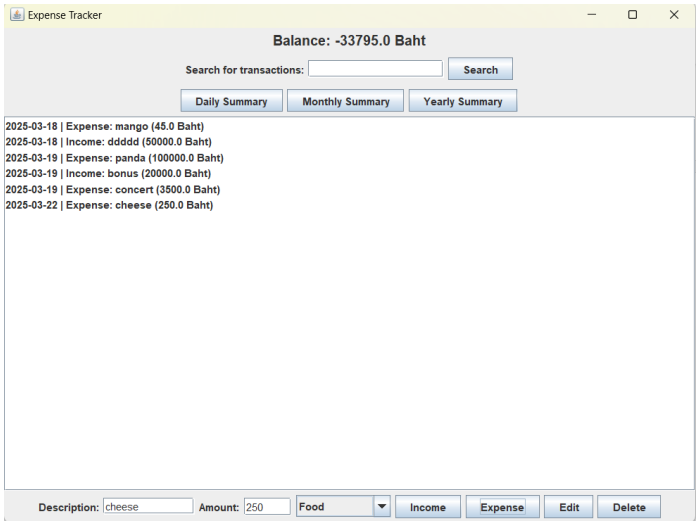
1. หน้าหลักของโปรแกรม



คำอธิบาย

หน้าหลักของโปรแกรมจะแสดงยอดเงินคงเหลือ (Balance) ไว้ด้านบนของหน้าจอ ตรงกลางคือรายการธุรกรรมทั้งหมดที่เคยบันทึกไว้ ด้านล่างคือฟอร์มสำหรับเพิ่มรายการใหม่ โดยสามารถเลือกประเภท (รายรับ/รายจ่าย), กรอกรายละเอียด และกดปุ่มเพื่อเพิ่ม ด้านบนยังมีช่องค้นหาและปุ่มสำหรับสรุปยอด

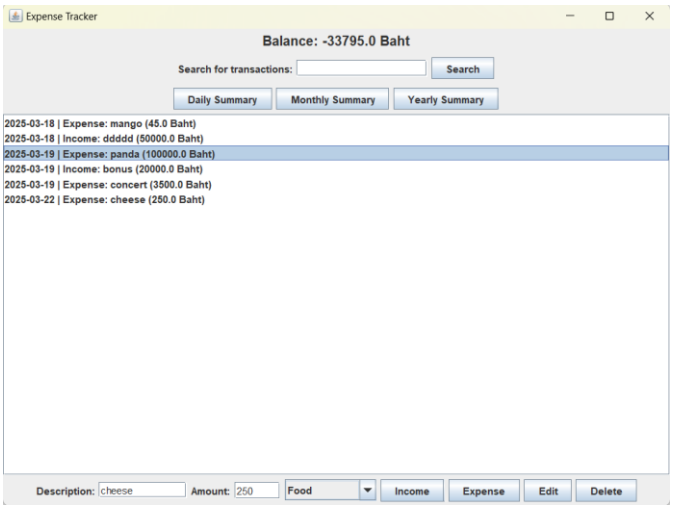
2. การเพิ่มธุรกรรม

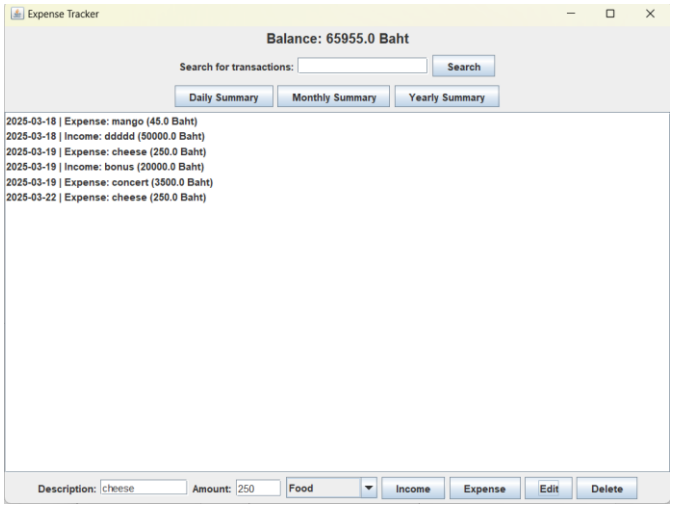


คำอธิบาย

เมื่อผู้ใช้กรอกรายละเอียด เช่น "ชีส", จำนวนเงิน 250 บาท, เลือกหมวดหมู่ "Food" และกดปุ่ม "Expense" ระบบจะเพิ่มรายการ พร้อมอัปเดตยอดเงินคงเหลือทันที

3. การลบธุรกรรม

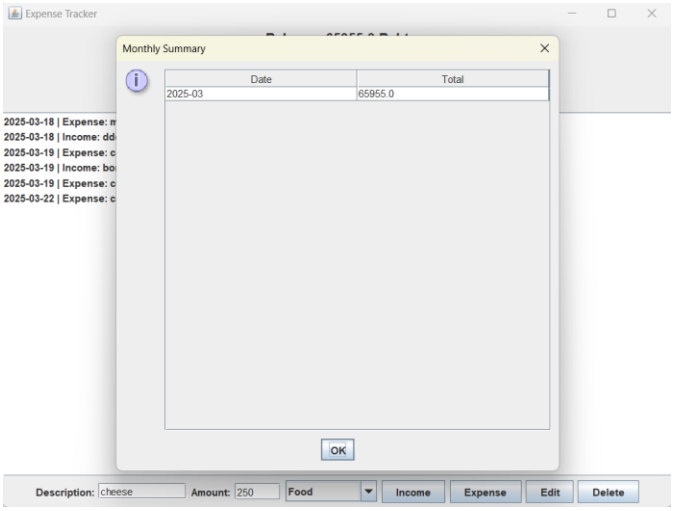




คำอธิบาย

ผู้ใช้สามารถเลือกธุรกรรมจากรายการ แล้วคลิกปุ่ม “Delete” ระบบจะลบรายการนั้นออกจากข้อมูลทั้งหมด บันทึกลงไฟล์ใหม่ และอัปเดตยอดเงินโดยอัตโนมัติ

4. การสรุปรายเดือน



คำอธิบาย

เมื่อผู้ใช้คลิกปุ่ม “Monthly Summary” ระบบจะแสดงตารางสรุปยอดเงินของแต่ละเดือน ตารางนี้แสดงวันที่ของ

แต่ละเดือน และยอดเงินสุทธิ (รายรับลบรายจ่าย) ข้อมูลนี้ดึงมาจากไฟล์ธุรกรรมโดยตรง และใช้การคำนวณผ่าน TransactionManager