1. Analysis of Object-Oriented Programming (OOP) Features

As we know from lectures, Java is an **Object-Oriented Programming** (OOP) **Language,** which means that the program design is organised around data, or objects, rather than functions and logic and the complete software operates as a bunch of objects talking t o each other. The key of object-oriented programming object-oriented programming is ab out turning real-world problems into flexible, reusable code using classes and objects. In our budget management system, we made full use of key OOP features like encapsulatio n, inheritance, polymorphism, and the Single Responsibility Principle. The following sectio ns are detailed explanation with code examples:

1. Encapsulation: Protective Binding of Data and Operations

Encapsulation means binding data and methods into a single unit while hiding how they work inside, only exposing the needed parts. This idea is used in different parts of the system: **BudgetCategory Class:** Its important attributes—name (category name), monthlyLimit (monthly limit), and currentExpenditure (current expenditure)—are declared as private, which means other classes can't change them directly. The data can only be changed by using the public method like addExpense() (to add expenses and check limits) and resetExpenditure() (to reset expenditures). For example, when you add an expense, the addExpense() menthod will check whether currentExpenditure + amount exceeds monthlyLimit. If it does, a MonthlyLimitExceededException is thrown. External callers need only deal with the exception without understanding the validation logic.

Transaction Abstract Class: Its attributes—id (unique ID), date (date), and amount (amount)—are also private, which can only be viewed by using public methods like getId() and getDate(), which helps keep the transaction data safe and correct.

2. Inheritance: Code Reuse and Hierarchical Design

From lectures we know that when an object acquires all the attributes and behaviours of a parent object, it is known as inheritance. Inheritance creates subclasses which inherit properties from super classes. The Transaction abstract class defines common behaviors for all transactions, with its child clasees Income and Expense implementing differentiated logic:

Transaction Abstract Class: It includes an abstract method getEffectiveAmount() which is used to calculate the effective amount, making subclasses to implement specific logic. It also provides common methods like getId() and getDate(), so that the same code doesn’t have to be written again.

Income Subclass: Inherits Transaction’s attributes, adds a new field called source (where the income came from), and overrides getEffectiveAmount() which directly returns the original amount, as income has no fees. For example, the toString() method in Income uses parent methods like getDate() and getAmount() while adding source for customized output.

Expense Subclass: Also inherits Transaction, adds paymentMethod (payment type) and category (expense category) fields, and overrides getEffectiveAmount() (calculating fees based on payment methods, e.g., deducting 1% for credit cards). This design unifies common transaction properties (ID, date, amount) while allowing subclasses to independently implement differences (effective amount calculation, additional fields).

3. Polymorphism: Dynamic Binding of Behaviors

Polymorphism enables subclasses to exhibit different behaviors at runtime by overriding parent class methods. The implementation of getEffectiveAmount() in the system is a classic example:

Parent Declaration, Subclass Implementation: Transaction defines getEffectiveAmount() as an abstract method, which Income and Expense implement differently. For instance, Income returns getAmount() (effective amount equals the original amount), while Expense deducts fees based on payment methods (e.g., returning getAmount() * (1 - 0.01) for credit cards).

Unified Interface, Differentiated Invocation: When BudgetManager calculates total income/expenditure (via methods like calculateTotalIncome() and calculateTotalExpenditure()), it iterates over all Transaction objects using transactions.stream().filter(t -> t instanceof Income).mapToDouble(Transaction::getEffectiveAmount). It calls getEffectiveAmount() without needing to know if the object is Income or Expense, ensuring correct results. This polymorphic design eliminates the need for subclass-specific code in statistics, significantly enhancing extensibility (e.g., adding a Transfer type only requires inheriting Transaction and overriding getEffectiveAmount()).

4. Single Responsibility Principle: Functional Focus of Classes

The Single Responsibility Principle requires a class to handle only one function. The system achieves this through modular design:

BudgetManager Class: As the core management class, it solely handles transaction CRUD operations (addTransaction(), getAllTransactions()), budget category management (addCategory(), removeCategory()), and statistical queries (calculateTotalIncome()). It does not involve specific transaction logic or UI interaction.

BudgetCategory Class: Focuses on budget category limit control (addExpense() for over-limit validation) and expenditure tracking (currentExpenditure updates), avoiding transaction storage or user input handling.

BudgetManagerUI Class: Manages user interaction logic (menu display, input validation, result output) but not business rules (e.g., fee calculation, limit checks). For example, handleExpenseEntry() calls getPaymentMethod() to retrieve the payment type, selectExistingCategory() to choose a category, and finally BudgetManager’s addTransaction() to add the transaction. All business logic is delegated to other classes, with BudgetManagerUI only "relaying commands" and "displaying results."

2. Description of functional module tests

1. Transaction Management Module Testing

The transaction management module is the core of the system, directly affecting the accuracy and integrity of users' income and expenditure data. Testing should focus on typical scenarios in actual user operations, such as adding normal income or expenditure records, boundary inputs (e.g., amount = 0, invalid date formats), calculation of fees for payment methods (differences between credit cards or cash), and critical validation of budget limits (exactly equal to the limit, exceeding the limit). Below is a detailed test plan for this module:

Test Type	Test Case Description	Test Method	Expected Result
Normal Flow (Income)	Add a ¥1000 income ("Salary") with date 2023-10-01	Use JUnit to call BudgetManager.addTransaction(new Income(...)) and query getAllTransactions()	The transaction list includes this income, and getEffectiveAmount() returns 1000 (no fees for income)
Boundary Condition (Amount = 0)	Enter "0" as the income amount in the "Record Income" interface	Manual UI test (input 0 and confirm)	The system prompts "Please enter a number greater than 0.00" (triggered by input validation)
Exception Scenario (Invalid Date)	Enter date "2023/10/01" (non-yyyy-MM-dd format) in the income date field	Manual UI test (input invalid date and confirm)	The system prompts "Invalid date format. Please use yyyy-MM-dd." (triggered by date validation)
Normal Flow (Expense)	Add a ¥1000 expense with "Credit Card" payment (category "Food")	Use JUnit to verify Expense.getEffectiveAmount() calculation (1000 ×(1-1%))	Effective amount = ¥990, and BudgetCategory.currentExpenditure updates to 990 (within the ¥500 limit)
Boundary Condition (Expense = Limit)	A "Food" category with a ¥500 limit, current expenditure ¥499, add a ¥1 effective expense	Call BudgetCategory.addExpense(1) via JUnit	No exception thrown, currentExpenditure updates to 500 (exactly equal to the limit)
Exception Scenario (Expense > Limit)	A "Food" category with a ¥500 limit, current expenditure ¥500, add a ¥1 effective expense	Call BudgetCategory.addExpense(1) via JUnit	Throws MonthlyLimitExceededException with message "Expense exceeds monthly limit..." (intercepted by limit validation)

2. Budget Category Management Module Testing

The budget category management module is responsible for creating, deleting, and querying budget categories, directly impacting the accuracy of budget control. Tests should cover scenarios such as uniqueness validation for new category names (avoid duplicates), existence checks for deletions (prevent accidental deletions), and handling of empty name inputs. Below is a detailed test plan for this module:

Test Type	Test Case Description	Test Method	Expected Result
Normal Flow (Add Category)	Add a new category "Travel" with a monthly limit of ¥800	Use JUnit to call BudgetManager.addCategory(new BudgetCategory(...)) and query getAllCategories()	The category list includes "Travel" with a limit of ¥800 (successfully added)
Boundary Condition (Duplicate Name)	Attempt to add an existing category "Food" (already exists)	Use JUnit to call BudgetManager.addCategory(new BudgetCategory("Food", 500))	Throws IllegalArgumentException with message "Category [Food] already exists" (triggered by uniqueness check)
Exception Scenario (Empty Name)	Enter an empty name in the "Add Category" interface	Manual UI test (input empty name and confirm)	The system prompts "Input cannot be empty. Please try again." (triggered by empty input validation)
Normal Flow (Delete Category)	Select to delete the existing "Entertainment" category in the "Manage Categories" interface	Manual UI test (select delete operation and confirm)	The category list removes "Entertainment" (successfully deleted)
Exception Scenario (Delete Non-Existent Category)	Attempt to delete a non-created category "Health"	Manual UI test (input "Health" and confirm deletion)	The system prompts "Error: Category [Health] not found" (triggered by existence check)

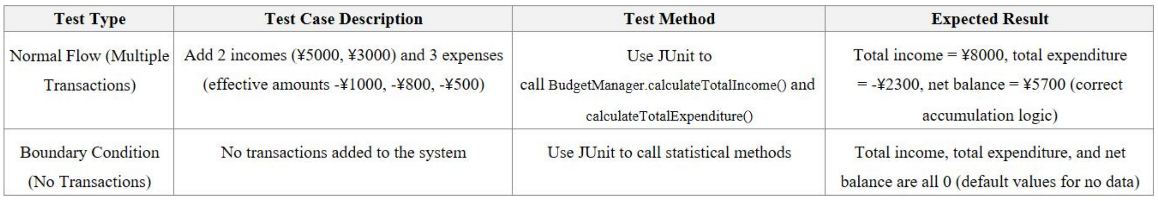
3. Complex Query Functions Testing

Complex query functions are key tools for users to analyze financial data, requiring accuracy and practicality in query results. Tests should cover common user needs such as filtering expenses by category (case-insensitive), filtering large expenses by amount threshold (boundary values), and querying by date range (including start or end dates). Below is a detailed test plan for this module:

Test Type	Test Case Description	Test Method	Expected Result
Category Query (Case-Insensitive)	Query for category "food" (actual category is "Food")	Use JUnit to call BudgetManager.getExpensesByCategory("food")	Returns all expenses in the "Food" category (matched via equalsIgnoreCase)
Threshold Query (Boundary Value)	Query expenses with effective amount > ¥500 (including ¥501 and ¥500)	Use JUnit to call BudgetManager.getExpensesAboveAmount(500)	Only returns the ¥501 expense (¥500 does not meet the > condition)
Date Range Query (Including Endpoints)	Query transactions between 2023-10-01 and 2023-10-31, including records on both dates	Use JUnit to call BudgetManager.getTransactionsBetweenDates("2023-10-01", "2023-10-31")	Returns records within the date range, sorted in ascending order (includes endpoints)

4. Statistical Functions Testing

Statistical functions help users quickly understand their financial situation, so it requires accuracy and reliability in results. Tests should include scenarios such as cumulative statistics for multiple transactions (total income or expenditure) and default statistics when no transactions exist (results = 0). Below is a detailed test plan for this module:



Test Type	Test Case Description	Test Method	Expected Result
Normal Flow (Multiple Transactions)	Add 2 incomes (¥5000, ¥3000) and 3 expenses (effective amounts -¥1000, -¥800, -¥500)	Use JUnit to call BudgetManager.calculateTotalIncome() and calculateTotalExpenditure()	Total income = ¥8000, total expenditure = -¥2300, net balance = ¥5700 (correct accumulation logic)
Boundary Condition (No Transactions)	No transactions added to the system	Use JUnit to call statistical methods	Total income, total expenditure, and net balance are all 0 (default values for no data)

3. AI-Assisted User Interface Design

This system employs a text-based menu-driven command-line interface (CLI), simulating user interaction through keyboard input. The UI design focuses on functional accessibility and input efficiency, with XipuAI assisting in optimizing interaction logic and error handling to ensure clear operational guidance within the constraints of text-based interaction.

1.Interface Functionality

The interface focuses on main steps of budget management. When the program starts, it shows a main menu and asks users to choose what they want to do by typing a number (1–6). For example, entering "1" starts income recording, "2" starts expense recording, and "3" opens budget category management. When recording income or expenses, the system guides users step-by-step through text simple prompts. Thay will be asked to enter amount (positive number), date (yyyy-MM-dd), source or payment method, and budget category (selected from a numbered list of existing categories). Inputs are validated in real-time (e.g., amount must be positive, date format must be valid). The category management function allows users to view existing categories (displaying limits and current expenditures), add new categories (enter name and limit), delete categories (with confirmation), or reset monthly expenditures (zero out current spending). The query function supports filter transactions by category, amount limit, or date range, with results showned in a text table (including transaction ID, date, amount, etc.).

2.Navigation Logic

Navigation follows a layered menu + quick return structure: The main menu serves as the root, with sub-functions (e.g., category management) as secondary menus (each containing 4-5 sub-options). Users return to the previous menu by entering "5" or another set of number. For example, entering "5" in the expense recording interface abandons the current operation and returns to the main menu, avoid repeated or mistaken entries. The system follows an 'input-process-feedback' pattern—after each action, it instantly shows a result like 'Income added successfully, ID: xxx' or an error message, so users always know what happened."

3.Input Validation and Exception Handling

Input validation and error handling rely on text prompts + program checks. For amount input, users must enter positive numbers, non-numeric or negative inputs trigger a prompt: "Please enter a valid positive number (e.g., 100.00)". Date input uses LocalDate.parse for format validation—invalid formats like "2023/10/01" prompt: "Date format must be yyyy-MM-dd". When selecting budget categories, entering a non-existent index triggers: "Category not found, please reselect". During expense entry, the system also checks if the new spending goes over the limit. If it does, it throws a custom MonthlyLimitExceededException with a message: "Current expenditure + new expense exceeds the monthly limit—operation failed".

4.XipuAI Integration and Reflection

XipuAI mainly helped improve how the program works and how the code was written. It gave Java code examples for things like checking dates (using DateTimeFormatter), setting up menu navigation (using loops and switch-case), and handling errors, which sped up the development process. Some of the AI's suggestions had to be changed to fit the command-line style of the app—for instance, form validation code meant for front-end use was rewritten to work with console input. Still, the AI's general advice, like making input steps simpler and showing clear feedback messages, worked well and made the interface easier to use in a CLI setting. In conclusion, XIpuAI provided knowledge unknow for me and helped me with guidance and examples of codes even if its answers need to be critical reflected and evaluated for better fitting the project.

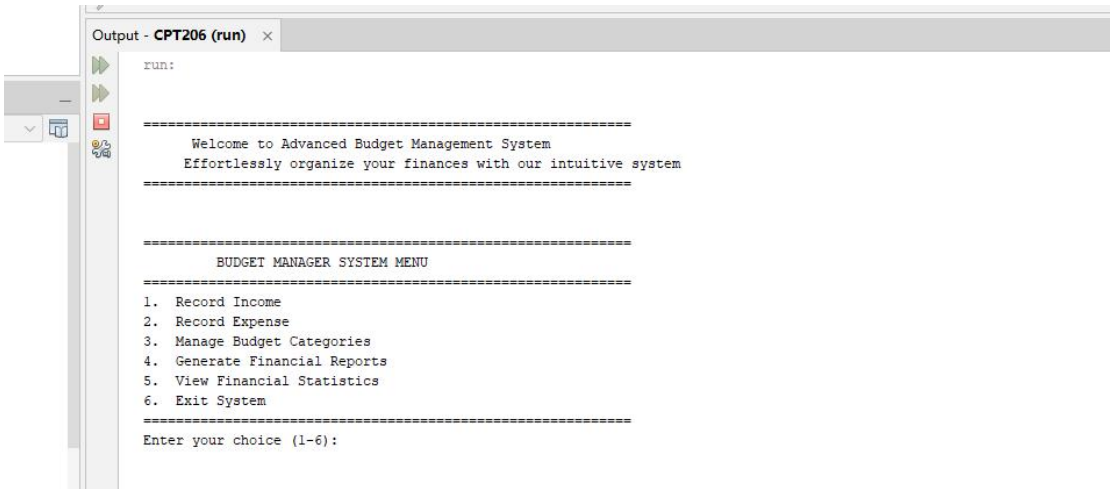
Appendix:

Full source code and course report for this project are available at:

https://github.com/zziyang2102-sketch/BudgetManager

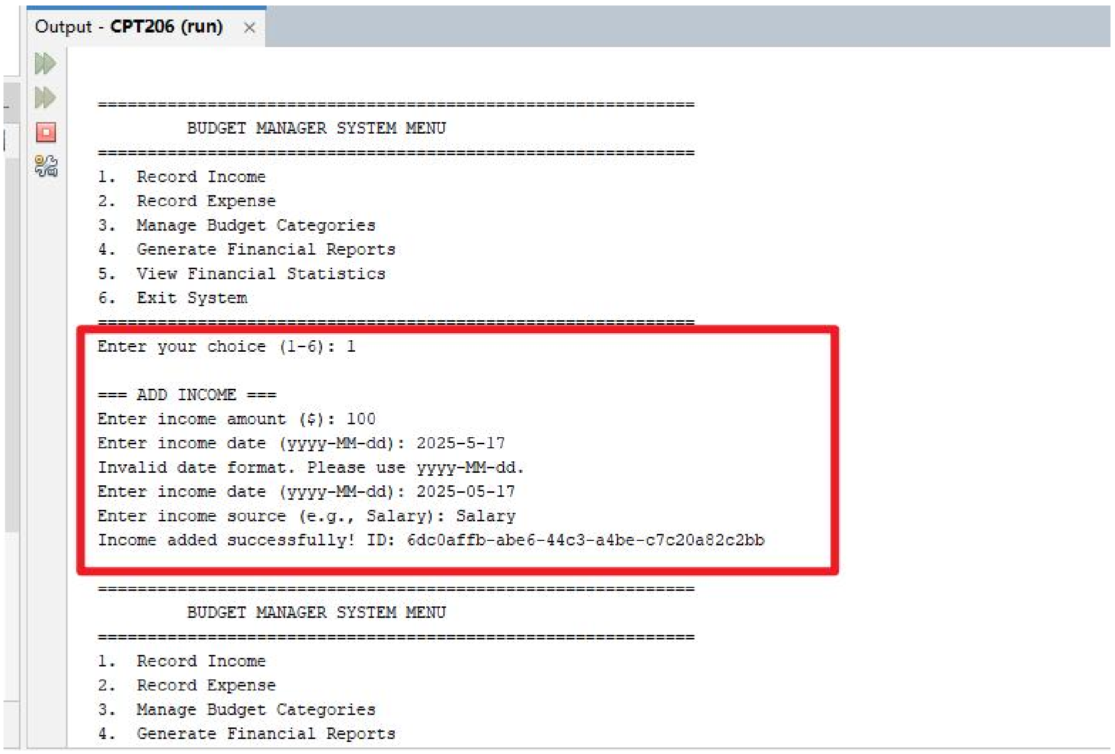
Demonstration of results

1.main menu



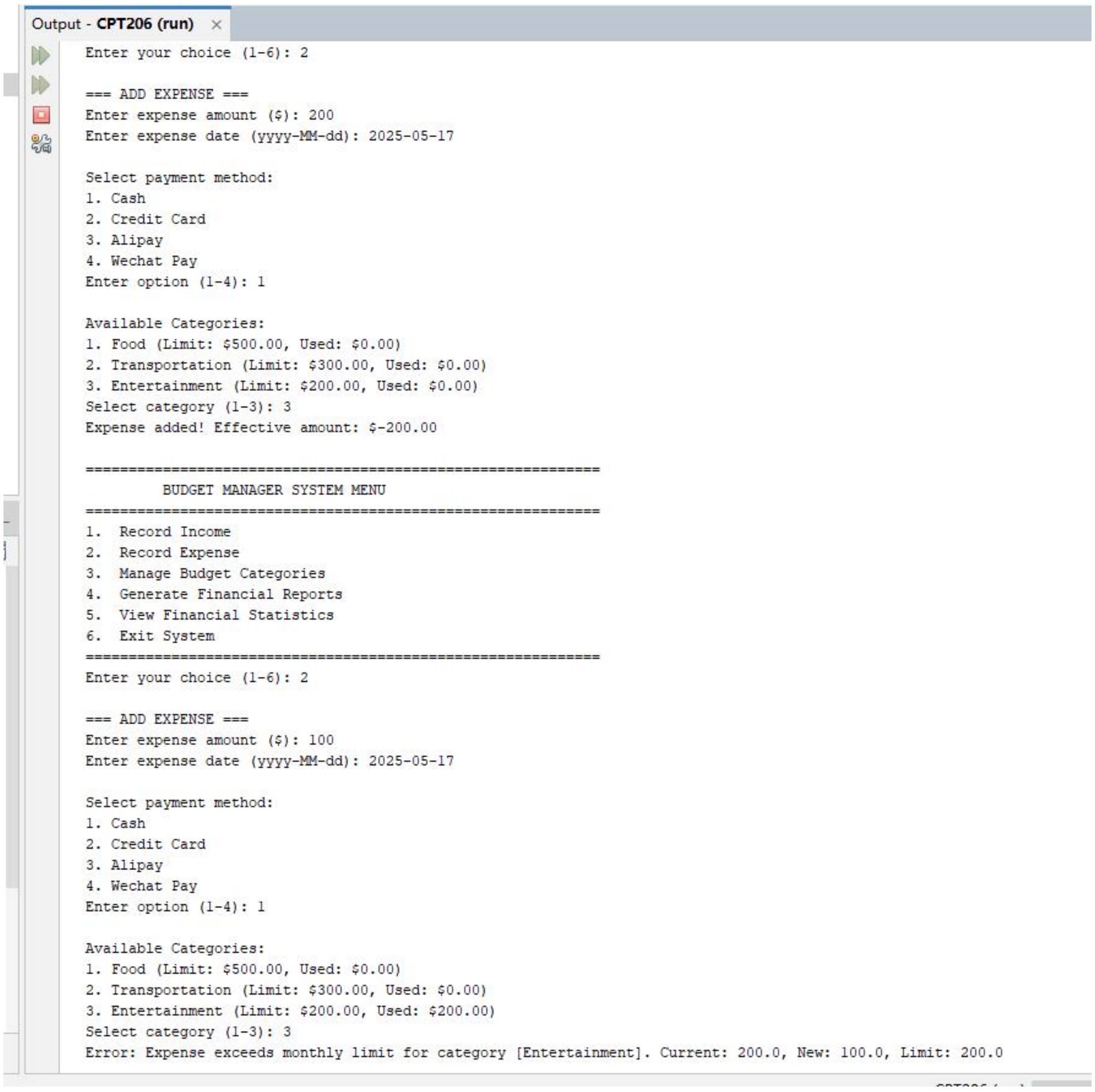
```
Output - CPT206 (run)

run:


============================================================
      Welcome to Advanced Budget Management System
     Effortlessly organize your finances with our intuitive system
============================================================


============================================================
         BUDGET MANAGER SYSTEM MENU
============================================================
1.  Record Income
2.  Record Expense
3.  Manage Budget Categories
4.  Generate Financial Reports
5.  View Financial Statistics
6.  Exit System
============================================================
Enter your choice (1-6):
```

2.Record Income



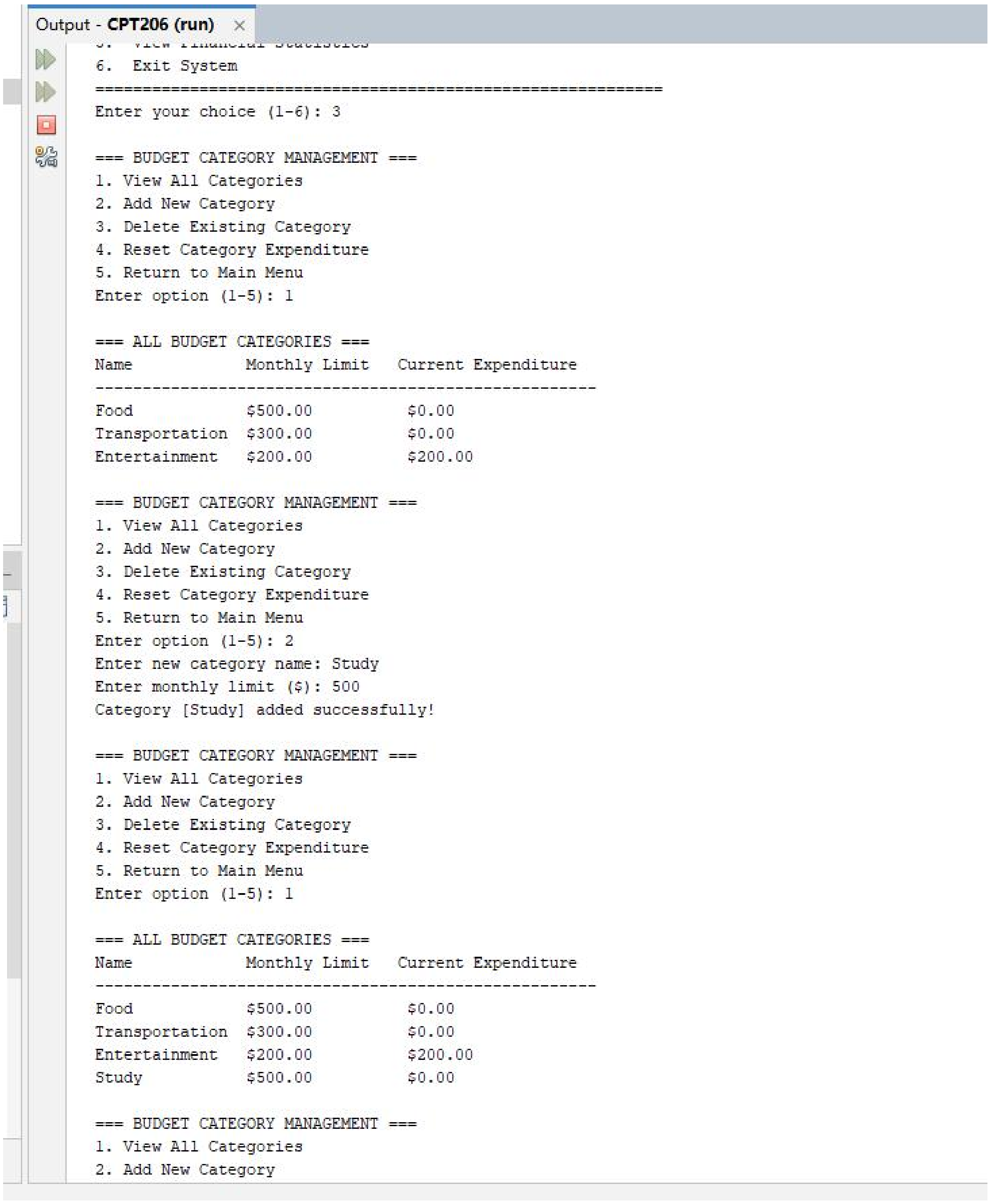
```
Output - CPT206 (run)

============================================================
         BUDGET MANAGER SYSTEM MENU
============================================================
1.  Record Income
2.  Record Expense
3.  Manage Budget Categories
4.  Generate Financial Reports
5.  View Financial Statistics
6.  Exit System
============================================================
Enter your choice (1-6): 1

=== ADD INCOME ===
Enter income amount ($): 100
Enter income date (yyyy-MM-dd): 2025-5-17
Invalid date format. Please use yyyy-MM-dd.
Enter income date (yyyy-MM-dd): 2025-05-17
Enter income source (e.g., Salary): Salary
Income added successfully! ID: 6dc0affb-abe6-44c3-a4be-c7c20a82c2bb
============================================================
         BUDGET MANAGER SYSTEM MENU
============================================================
1.  Record Income
2.  Record Expense
3.  Manage Budget Categories
4.  Generate Financial Reports
```

3.Record Expense


Output - CPT206 (run)

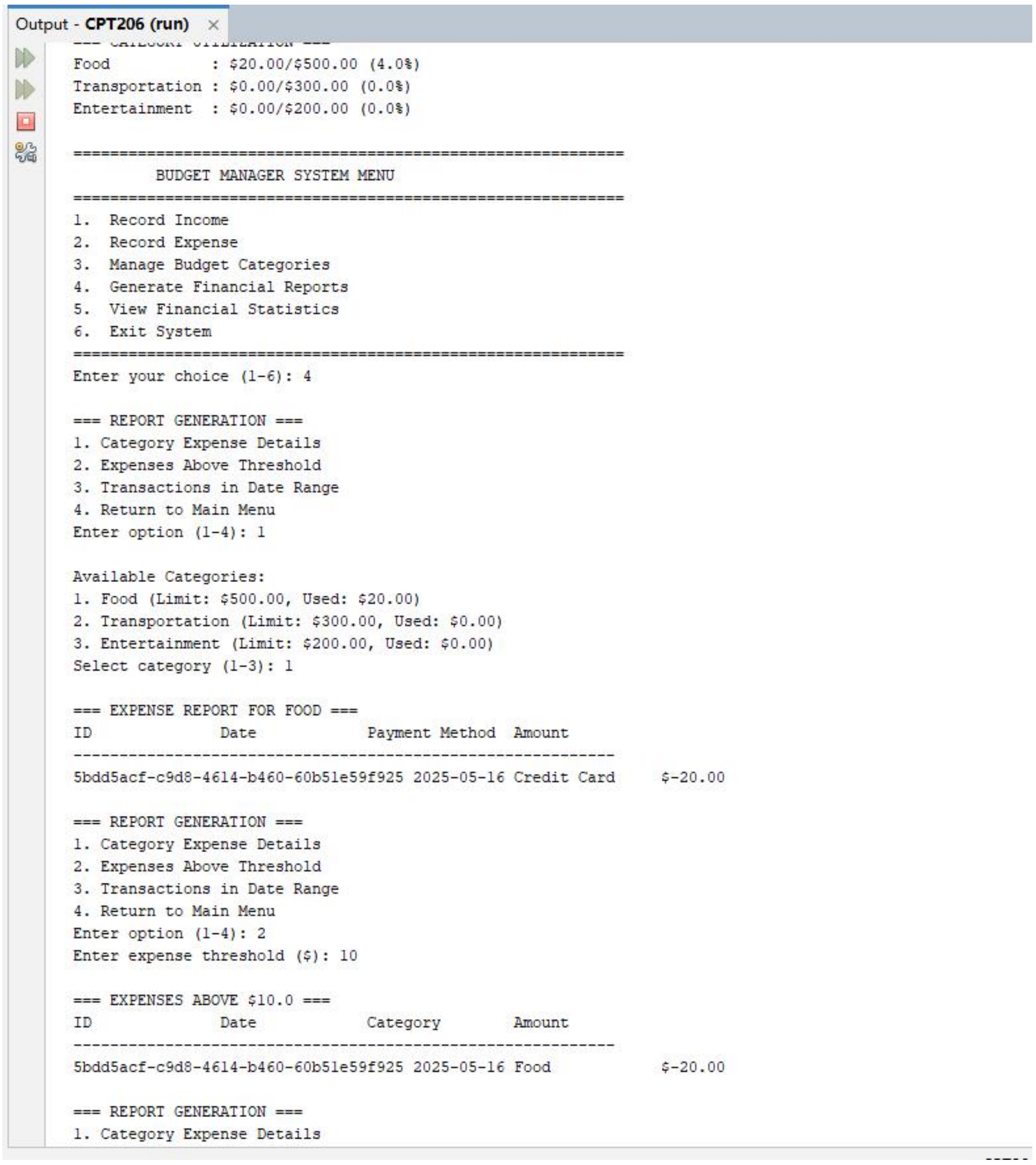
```
Enter your choice (1-6): 2

=== ADD EXPENSE ===
Enter expense amount ($): 200
Enter expense date (yyyy-MM-dd): 2025-05-17

Select payment method:
1. Cash
2. Credit Card
3. Alipay
4. Wechat Pay
Enter option (1-4): 1

Available Categories:
1. Food (Limit: $500.00, Used: $0.00)
2. Transportation (Limit: $300.00, Used: $0.00)
3. Entertainment (Limit: $200.00, Used: $0.00)
Select category (1-3): 3
Expense added! Effective amount: $-200.00

============================================================
         BUDGET MANAGER SYSTEM MENU
============================================================
1.  Record Income
2.  Record Expense
3.  Manage Budget Categories
4.  Generate Financial Reports
5.  View Financial Statistics
6.  Exit System
============================================================
Enter your choice (1-6): 2

=== ADD EXPENSE ===
Enter expense amount ($): 100
Enter expense date (yyyy-MM-dd): 2025-05-17

Select payment method:
1. Cash
2. Credit Card
3. Alipay
4. Wechat Pay
Enter option (1-4): 1

Available Categories:
1. Food (Limit: $500.00, Used: $0.00)
2. Transportation (Limit: $300.00, Used: $0.00)
3. Entertainment (Limit: $200.00, Used: $200.00)
Select category (1-3): 3
Error: Expense exceeds monthly limit for category [Entertainment]. Current: 200.0, New: 100.0, Limit: 200.0
```

4.Manage Budget Categories


Output - **CPT206 (run)**

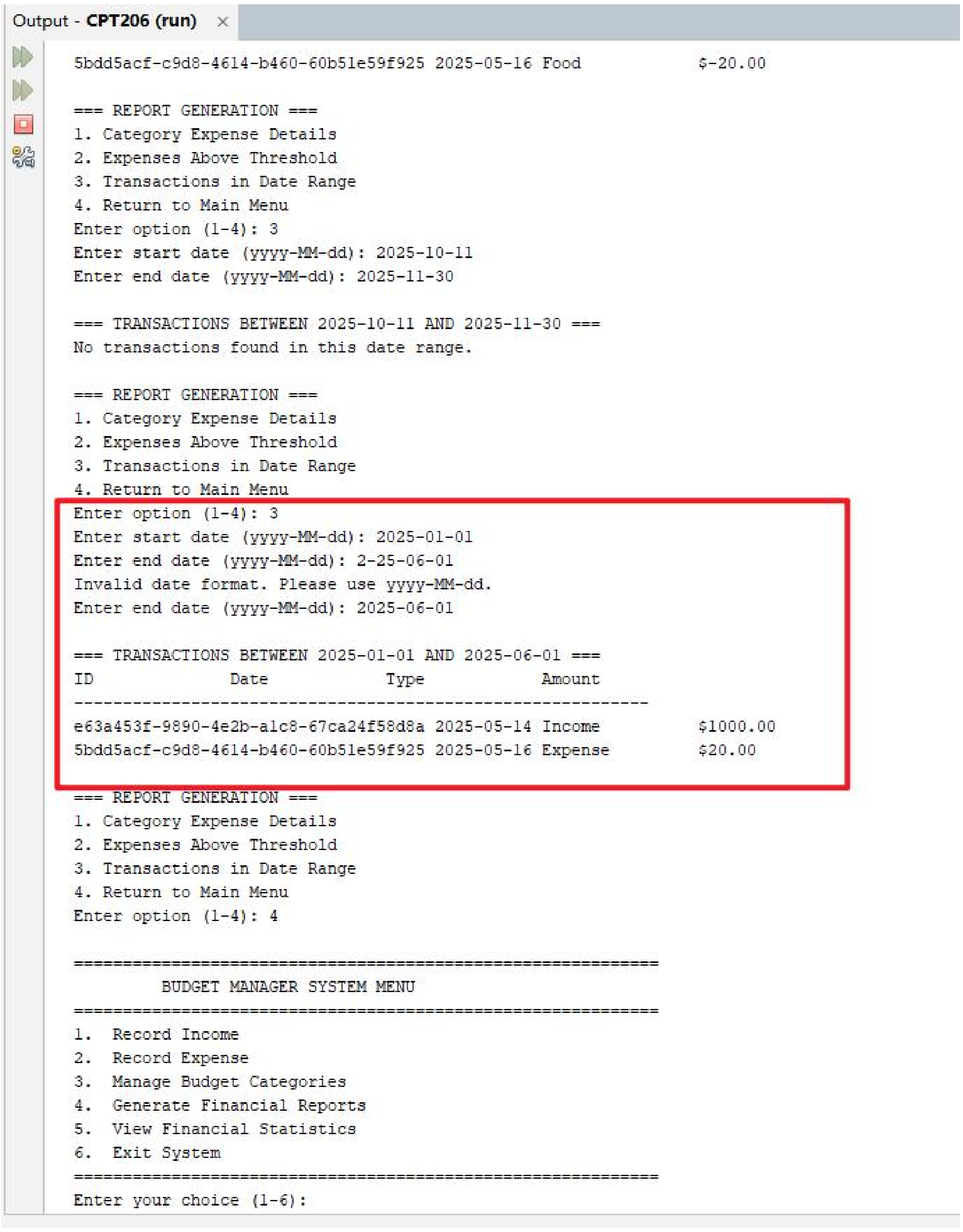
```
5.  View Financial Statistics
6.  Exit System
============================================================
Enter your choice (1-6): 3

=== BUDGET CATEGORY MANAGEMENT ===
1. View All Categories
2. Add New Category
3. Delete Existing Category
4. Reset Category Expenditure
5. Return to Main Menu
Enter option (1-5): 1

=== ALL BUDGET CATEGORIES ===
Name            Monthly Limit   Current Expenditure
-----------------------------------------------------
Food            $500.00          $0.00
Transportation  $300.00          $0.00
Entertainment   $200.00          $200.00

=== BUDGET CATEGORY MANAGEMENT ===
1. View All Categories
2. Add New Category
3. Delete Existing Category
4. Reset Category Expenditure
5. Return to Main Menu
Enter option (1-5): 2
Enter new category name: Study
Enter monthly limit ($): 500
Category [Study] added successfully!

=== BUDGET CATEGORY MANAGEMENT ===
1. View All Categories
2. Add New Category
3. Delete Existing Category
4. Reset Category Expenditure
5. Return to Main Menu
Enter option (1-5): 1

=== ALL BUDGET CATEGORIES ===
Name            Monthly Limit   Current Expenditure
-----------------------------------------------------
Food            $500.00          $0.00
Transportation  $300.00          $0.00
Entertainment   $200.00          $200.00
Study           $500.00          $0.00

=== BUDGET CATEGORY MANAGEMENT ===
1. View All Categories
2. Add New Category
```

5.Generate Financial Reports



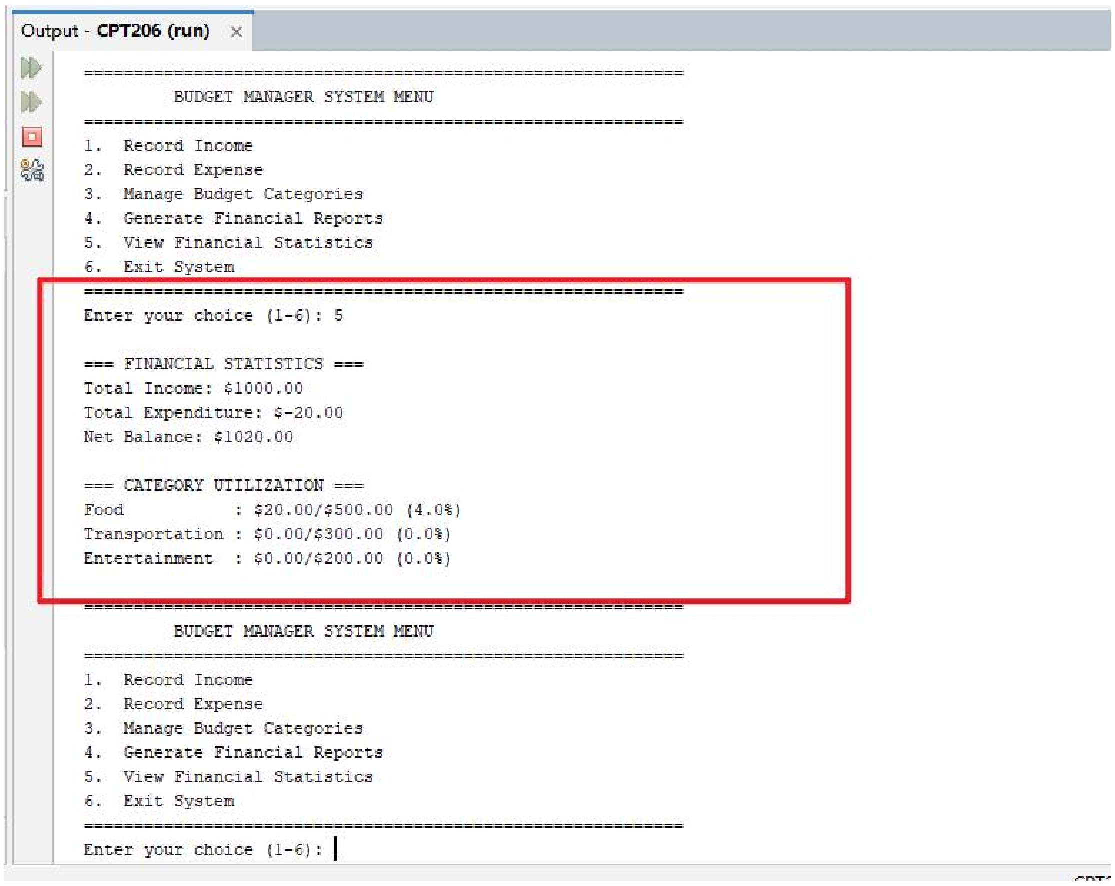
Output - CPT206 (run)

```
Food           : $20.00/$500.00 (4.0%)
Transportation : $0.00/$300.00 (0.0%)
Entertainment  : $0.00/$200.00 (0.0%)

============================================================
         BUDGET MANAGER SYSTEM MENU
============================================================
1.  Record Income
2.  Record Expense
3.  Manage Budget Categories
4.  Generate Financial Reports
5.  View Financial Statistics
6.  Exit System
============================================================
Enter your choice (1-6): 4

=== REPORT GENERATION ===
1. Category Expense Details
2. Expenses Above Threshold
3. Transactions in Date Range
4. Return to Main Menu
Enter option (1-4): 1

Available Categories:
1. Food (Limit: $500.00, Used: $20.00)
2. Transportation (Limit: $300.00, Used: $0.00)
3. Entertainment (Limit: $200.00, Used: $0.00)
Select category (1-3): 1

=== EXPENSE REPORT FOR FOOD ===
ID              Date            Payment Method  Amount
-----------------------------------------------------------
5bdd5acf-c9d8-4614-b460-60b51e59f925 2025-05-16 Credit Card     $-20.00

=== REPORT GENERATION ===
1. Category Expense Details
2. Expenses Above Threshold
3. Transactions in Date Range
4. Return to Main Menu
Enter option (1-4): 2
Enter expense threshold ($): 10

=== EXPENSES ABOVE $10.0 ===
ID              Date            Category        Amount
-----------------------------------------------------------
5bdd5acf-c9d8-4614-b460-60b51e59f925 2025-05-16 Food            $-20.00

=== REPORT GENERATION ===
1. Category Expense Details
```


Output - **CPT206 (run)**

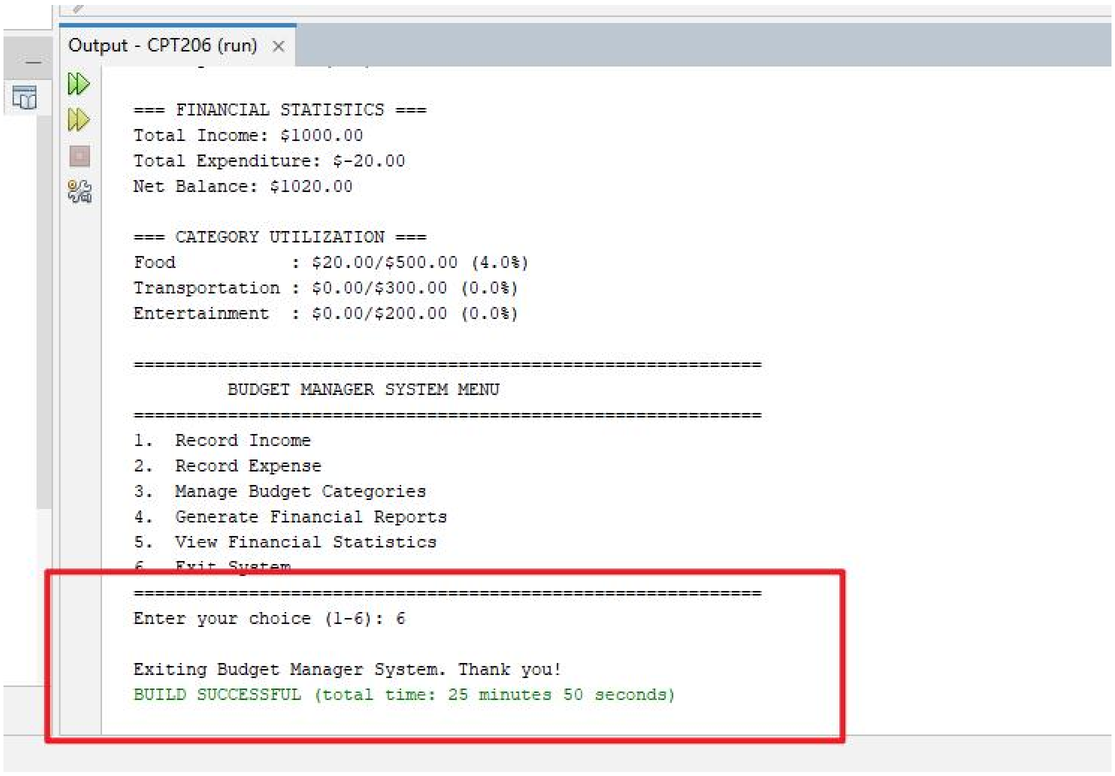
```
5bdd5acf-c9d8-4614-b460-60b51e59f925 2025-05-16 Food            $-20.00

=== REPORT GENERATION ===
1. Category Expense Details
2. Expenses Above Threshold
3. Transactions in Date Range
4. Return to Main Menu
Enter option (1-4): 3
Enter start date (yyyy-MM-dd): 2025-10-11
Enter end date (yyyy-MM-dd): 2025-11-30

=== TRANSACTIONS BETWEEN 2025-10-11 AND 2025-11-30 ===
No transactions found in this date range.

=== REPORT GENERATION ===
1. Category Expense Details
2. Expenses Above Threshold
3. Transactions in Date Range
4. Return to Main Menu
Enter option (1-4): 3
Enter start date (yyyy-MM-dd): 2025-01-01
Enter end date (yyyy-MM-dd): 2-25-06-01
Invalid date format. Please use yyyy-MM-dd.
Enter end date (yyyy-MM-dd): 2025-06-01

=== TRANSACTIONS BETWEEN 2025-01-01 AND 2025-06-01 ===
ID              Date            Type            Amount
------------------------------------------------------------
e63a453f-9890-4e2b-a1c8-67ca24f58d8a 2025-05-14 Income          $1000.00
5bdd5acf-c9d8-4614-b460-60b51e59f925 2025-05-16 Expense         $20.00

=== REPORT GENERATION ===
1. Category Expense Details
2. Expenses Above Threshold
3. Transactions in Date Range
4. Return to Main Menu
Enter option (1-4): 4

============================================================
         BUDGET MANAGER SYSTEM MENU
============================================================
1.  Record Income
2.  Record Expense
3.  Manage Budget Categories
4.  Generate Financial Reports
5.  View Financial Statistics
6.  Exit System
============================================================
Enter your choice (1-6):
```

6.View Financial Statistics



```
============================================================
         BUDGET MANAGER SYSTEM MENU
============================================================
1.  Record Income
2.  Record Expense
3.  Manage Budget Categories
4.  Generate Financial Reports
5.  View Financial Statistics
6.  Exit System
============================================================
Enter your choice (1-6): 5

=== FINANCIAL STATISTICS ===
Total Income: $1000.00
Total Expenditure: $-20.00
Net Balance: $1020.00

=== CATEGORY UTILIZATION ===
Food          : $20.00/$500.00 (4.0%)
Transportation : $0.00/$300.00 (0.0%)
Entertainment  : $0.00/$200.00 (0.0%)

============================================================
         BUDGET MANAGER SYSTEM MENU
============================================================
1.  Record Income
2.  Record Expense
3.  Manage Budget Categories
4.  Generate Financial Reports
5.  View Financial Statistics
6.  Exit System
============================================================
Enter your choice (1-6):
```

7.Exit System



```
=== FINANCIAL STATISTICS ===
Total Income: $1000.00
Total Expenditure: $-20.00
Net Balance: $1020.00

=== CATEGORY UTILIZATION ===
Food          : $20.00/$500.00 (4.0%)
Transportation : $0.00/$300.00 (0.0%)
Entertainment  : $0.00/$200.00 (0.0%)

============================================================
         BUDGET MANAGER SYSTEM MENU
============================================================
1.  Record Income
2.  Record Expense
3.  Manage Budget Categories
4.  Generate Financial Reports
5.  View Financial Statistics
6.  Exit System
============================================================
Enter your choice (1-6): 6

Exiting Budget Manager System. Thank you!
BUILD SUCCESSFUL (total time: 25 minutes 50 seconds)
```