System Requirements Specification Index

For

Personal Finance Management

Version 1.0

IIHT Pvt. Ltd.
fullstack@iiht.com

Table of Contents

Personal Finance Management

System Requirements Specification

1 Project Abstract

Personal Finance Management Console is a Java-based console application designed to help users efficiently manage their personal finances. The system allows users to track expenses, categorize transactions, and maintain a record of financial activities. The application enables users to perform Create, Read, Update operations on expense records, view transactions by category. Users can update and analyse their spending habits, ensuring effective financial management. The system provides an interactive menu-driven interface, making it simple and user-friendly for managing personal expenses and budgeting effectively.

2 Business Requirements:

Screen Name	Console input screen
Problem Statement	1. User needs to enter into the application. 2. The user should be able to do the particular operations 3. The console should display the menu 1) Add Expense 2) View All Expenses 3) View Expenses by Category 4) Update Expense 5) View Balance 6) Exit

2.1 PersonalFinanceApp Class - Method Descriptions

Method	Task	Implementation Details	Return Value
`main(String[] args)`	Entry point for the application	- Initialize `Scanner` and `TransactionManager`. - Set **default monthly income** to `10,000`. - Display a **menu-driven console** for expense management. - Process user input and call respective methods based on menu selection.	`void` (Runs continuously until user selects exit)

`addExpense(TransactionManager transactionManager, Scanner scanner)`	Add a new expense	- Prompt the user for `amount`, `description`, and `category`.
		- Call `transactionManager.addExpense()`.
		- Handle `InvalidAmountException` if thrown.
			`void` (Prints success or error message).
			Exception: `InvalidAmountException` exception should be caught and it should print a message as: "Error: " + e.getMessage().
			Example Output:
			`"Expense added successfully."`
			or
			`"Error: Expense amount must be positive."`

`viewAllTransactions(TransactionManager transactionManager)`	Display all expenses	- Fetch the list of transactions using `transactionManager.getAllTransactions()`. - If the list is **empty**, print `"No transactions found."`. - Otherwise, print all transactions.	`void` (Prints list of transactions or `"No transactions found."` message) Print the list of all transactions by iterating over each transaction and print using

			System.out.println(transaction);
viewTransactionsByCategory(TransactionManager transactionManager, Scanner scanner)	Display expenses for a specific category	- Prompt the user to enter a `category`. - Fetch transactions by category using `transactionManager.getTransactionsByCategory(category)`. - If no transactions exist for the category, print `"No transactions found for category: [category]"`.	`void` (Prints transactions for the category or a message for not found as `“No transactions found for category:”` + category). Print the list of all transactions by iterating over each transaction and print using System.out.println(transaction);

updateTransaction(TransactionManager transactionManager, Scanner scanner)	Modify an existing expense	- Prompt the user for `index`, `new amount`, `new description`, and `new category`.
		- Call `transactionManager.updateTransaction()`.
		- Handle `InvalidAmountException` or `IndexOutOfBoundsException` if thrown.
			`void` (Print "Transaction updated successfully." if successfully updated or print error message if there is an exception.
			Exception: `InvalidAmountException` or `IndexOutOfBounds Exception` exception should be caught and it should print a message as: "Error: " + e.getMessage().
			Example Output:
			`"Transaction updated successfully."`
			or
			`"Error: Transaction not`

			`found at index: "+ index.`
`viewBalance(TransactionManager transactionManager, double monthlyIncome)`	Calculate and display remaining balance	- Fetch all transactions and sum their amounts. - Subtract total expenses from `monthlyIncome`. - Print `"Current Balance: [balance]"`.	`void` (Prints remaining balance) **Example Output:** `"Current Balance: [balance]"`

2.2 TransactionManager Class - Method Descriptions

Method	Task	Implementation Details	Return Value
`addExpense(double amount, String description, String category)`	Add a new expense to the list	- **Validate `amount`:** Check if `amount > 0`, else throw `InvalidAmountException with message "Expense amount must be positive."`. - **Create Transaction:** Instantiate a `Transaction` object with the provided `amount`, `description`, and `category`. - **Add to List:** Add the newly created `Transaction` object to the `transactions` list.	**Returns**: `void` (Adds transaction to list or throws exception). **Throw Exception**: `InvalidAmountException` if `amount <= 0` with message `"Expense amount must be positive."`.
`updateTransaction(int index, double amount, String description, String category)`	Modify an existing expense	- **Validate Index:** Check if `index` is within the range of the `transactions` list else throw `IndexOutOfBoundsException` with message: "Transaction not found at index: " + index.	**Returns:** `void` (Updates the transaction or throws exception). **Throw Exception:** `IndexOutOfBoundsException` if `index` is invalid with message: "Transaction not found at index: " + index.

		- **Retrieve Transaction:** Get the `Transaction` at the specified `index`. - **Validate Amount:** Check if `amount > 0`, else throw `InvalidAmountException` with message: "Amount must be positive." - **Update Transaction:** Update the `amount`, `description`, and `category` of the retrieved `Transaction`.	**Throw Exception:** `InvalidAmountException` if `amount <= 0` with message: "Amount must be positive."
`getAllTransactions()`	Retrieve all recorded expenses	- **Return List:** Return the `transactions` list containing all recorded expenses.	**Returns:** `List<Transaction>` (All transactions).
`getTransactionsByCategory(String category)`	Fetch all expenses for a specific category	- **Filter Transactions:** Loop through each `Transaction` in `transactions`. - **Category Match:** If `transaction.getCategory().equals(category)`, add it to a new `filteredTransactions` list. - **Return Filtered List:** Return the `filteredTransactions` list.	**Returns:** `List<Transaction>` (Filtered transactions).
`getBalance(double monthlyIncome)`	Calculate remaining balance	- **Sum Expenses:** Loop through each `Transaction` in `transactions` and return the balance amount. - **Return Balance:** Return the calculated balance.	**Returns:** `double` (Remaining balance).

3 Constraints

3.1 Transaction Constraints

- When adding an expense with a negative or zero amount, the method should throw an **InvalidAmountException** with the message:

  "Expense amount must be positive."

- When trying to update a transaction:

  ➔ If the **index is less than 0** or **greater than or equal to the size of the transactions list**, then the method should throw an **IndexOutOfBoundsException** with the message:

  "Transaction not found at index: [index]."

  ➔ If the **amount is negative or zero**, the method should throw an **InvalidAmountException** with the message:

  "Amount must be positive."

3.2 Common Constraints

- The system should allow users to input and store details of multiple transactions.
- Users should be able to categorize each transaction during input.
- Users should be able to retrieve and filter expenses based on categories.
- Users should be able to update a transaction by specifying its index (and index value must start with 0).
- The system should calculate and display the remaining balance after expenses have been recorded.

4 Template Code Structure

4.1 Package: com.finance

Resources

Class/Interface	Description	Status
PersonalFinanceApp.java(class)	This represents bootstrap class i.e class with Main method, that shall contain all console interaction with the user.	**Partially implemented**

4.2 Package: com.finance.model

Resources

Class/Interface	Description	Status
Transaction (class)	● This class contains all the properties of the Transaction class.	Already implemented.

4.3 Package: com.finance.inventory

Resources

Class/Interface	Description	Status
TransactionManager (class)	● This class contains all the methods which are used to write the business logic for the application ● You can create any number of private methods in the class	Partially implemented.

4.4 Package: com.finance.exception

Resources

Class/Interface	Description	Status
InvalidAmountException (Class)	● Custom Exception to be thrown when an invalid amount (negative or zero) is entered for a transaction.	Already created.

5 Execution Steps to Follow

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers need to go to the Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.**
3. **This editor Auto Saves the code.**

4. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**
5. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
6. **To run your project use command:**
   mvn clean install exec:java -Dexec.mainClass="com.finance.PersonalFinanceApp"
7. **To test your project, use the command**
   mvn test
8. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**