INFORMATICS INSTITUTE OF TECHNOLOGY

Level 4 - Computer Science

Module: Software Development I

Module Leader: Mr. Guhanathan Poravi

Type Of Assignment: Individual Coursework

Student ID : M. H. D. De Costa

Student ID: 20231262

Table of Contents

1 Problem

Overview:

Building on your knowledge of Python, dictionaries, and file I/O, your next challenge is to enhance the Personal Finance Tracker by developing a graphical user interface (GUI) using Tkinter. This advanced version should not only display the information from a provided JSON file but also incorporate object-oriented programming (OOP) concepts for the GUI components. Additionally, your application will include a search function and a sorting feature, similar to a file explorer, to manage and analyze financial transactions more effectively.

Objectives:

1. Integrate a GUI using Tkinter and OOP concepts.
2. Load and display data from a JSON file upon GUI invocation.
3. Implement search and sorting functionalities within the GUI.
4. Ensure the application is user-friendly and robust.

2 Python Code

2.1 Screenshots of the GUI code

File Edit Format Run Options Window Help

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import json
import datetime

# Defining the FinanceTrackerGUI class
class FinanceTrackerGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Personal Finance Tracker")
        self.transactions = self.load_transactions("expenses.json")
        self.create_widgets()

    # load transactions from a JSON file
    def load_transactions(self, filename):
        try:
            with open(filename, "r") as file:
                data = json.load(file)
                transactions = {}
                for category, items in data.items():
                    for item in items:
                        if category.strip() not in transactions:
                            transactions[category.strip()] = []
                        transactions[category.strip()].append(item)
                return transactions
        except FileNotFoundError:
            return {}

    # Method to save transactions to a JSON file
    def save_transactions(self):
        filename = "transactions.json"
        with open(filename, "w") as file:
            json.dump(self.transactions, file, indent=4)
```

Figure 1 Screenshot 1

File Edit Format Run Options Window Help

```
    # Method to create GUI widgets
    def create_widgets(self):
        self.table_frame = tk.Frame(self.root)
        self.table_frame.pack(fill=tk.BOTH, expand=True)

        # Treeview widget to display transactions
        self.transaction_tree = ttk.Treeview(self.table_frame, columns=("transaction_number", "amount", "category", "date"), show="headings")
        self.transaction_tree.heading("transaction_number", text="Transaction Number")
        self.transaction_tree.heading("amount", text="Amount")
        self.transaction_tree.heading("category", text="Category")
        self.transaction_tree.heading("date", text="Date")
        self.transaction_tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Scrollbar for the transaction table
        self.scrollbar = ttk.Scrollbar(self.table_frame, orient=tk.VERTICAL, command=self.transaction_tree.yview)
        self.scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.transaction_tree.configure(yscrollcommand=self.scrollbar.set)

        # Search bar for filtering transactions
        self.search_bar = tk.Entry(self.root)
        self.search_bar.pack(pady=10)
        self.search_button = tk.Button(self.root, text="Search", command=self.search_transactions)
        self.search_button.pack(pady=10)

        # Sorting options dropdown menu
        sort_options = ["Date (Oldest to Newest)", "Category (A to Z)", "Amount (Smallest to Largest) \nCategory Wise", "Transaction Number (Ascending) \nCategory Wise"]
        self.sort_var = tk.StringVar(self.root)
        self.sort_var.set(sort_options[0])  # Default sorting option
        sort_dropdown = tk.OptionMenu(self.root, self.sort_var, *sort_options, command=self.sort_transactions)
        sort_dropdown.pack(pady=10)

        # Refresh button to refresh the transaction table
        self.refresh_button = tk.Button(self.root, text="Refresh", command=lambda: self.display_transactions(self.transactions))
        self.refresh_button.pack(pady=10)

    # Method to display transactions in the table
    def display_transactions(self, transactions):
        self.transaction_tree.delete(*self.transaction_tree.get_children())
        for category, data in transactions.items():
            for transaction in data:
                self.transaction_tree.insert("", tk.END, values=(
                    transaction["transaction_number"],
                    transaction["amount"],
                    category,
                    transaction["date"]
                ))
        self.save_transactions()
```

Figure 2 Screenshot 2

File Edit Format Run Options Window Help

```
    # Method to filter transactions based on search criteria
    def search_transactions(self):
        search_term = self.search_bar.get().lower()
        filtered_transactions = {}
        for category, data in self.transactions.items():
            filtered_data = []
            for transaction in data:
                # Check if the search term matches any value in the transaction or the category name
                if any(term.lower() in str(value).lower() for term in search_term.split() for value in transaction.values()) or search_term.lower() in category.lower():
                    filtered_data.append(transaction)
            if filtered_data:
                filtered_transactions[category] = filtered_data

        # If no search results found, display a message box
        if not filtered_transactions:
            messagebox.showinfo("Search Results", "No search found.")

        self.display_transactions(filtered_transactions)

    # Method to sort transactions based on selected option
    def sort_transactions(self, selected_option):
        if selected_option == "Date (Oldest to Newest)":
            # Create a dictionary of transaction numbers and their corresponding dates
            transactions_with_dates = {}
            for category, data in self.transactions.items():
                for transaction in data:
                    transactions_with_dates[transaction["transaction_number"]] = transaction["date"]

            # Sort transactions by date
            sorted_transactions = sorted(transactions_with_dates.items(), key=lambda x: datetime.datetime.strptime(x[1], "%Y-%m-%d"))

            # Clear existing entries in the transaction tree
            self.transaction_tree.delete(*self.transaction_tree.get_children())

            # Insert sorted transactions into the transaction tree
            for transaction_number, date in sorted_transactions:
                category = next(category for category, data in self.transactions.items() if any(tx["transaction_number"] == transaction_number for tx in data))
                transaction = next(tx for tx in self.transactions[category] if tx["transaction_number"] == transaction_number)
                self.transaction_tree.insert("", tk.END, values=(
                    transaction_number,
                    transaction["amount"],
                    category,
                    date
                ))
```

Figure 3 Screenshot 3

```
        else:
            if selected_option == "Category (A to Z)":
                sorted_transactions = sorted(self.transactions.items(), key=lambda x: x[0])
            elif selected_option == "Amount (Smallest to Largest) \nCategory Wise":
                sorted_transactions = [(category, sorted(data, key=lambda x: float(x["amount"]))) for category, data in self.transactions.items()]
            elif selected_option == "Transaction Number (Ascending) \nCategory Wise":
                sorted_transactions = [(category, sorted(data, key=lambda x: int(x["transaction_number"]))) for category, data in self.transactions.items()]

            # Clear existing entries in the transaction tree
            self.transaction_tree.delete(*self.transaction_tree.get_children())

            # Insert sorted transactions into the transaction tree
            for category, data in sorted_transactions:
                for transaction in data:
                    self.transaction_tree.insert("", tk.END, values=(
                        transaction["transaction_number"],
                        transaction["amount"],
                        category,
                        transaction["date"]
                    ))

def main():
    root = tk.Tk()
    app = FinanceTrackerGUI(root)
    app.display_transactions(app.transactions)
    root.mainloop()

if __name__ == "__main__":
    main()
```

Figure 4 Screenshot 4

2.2 Changes that made in CW2 Code

File Edit Format Run Options Window Help

```
#SD1_COURSEORK_3
#ID - 20231262

import json
from datetime import datetime
import tkinter as tk
from tkinter import messagebox
from CW3_GUI import FinanceTrackerGUI

# Global dictionary to store expenses by category
expenses = {}
transaction_counter = 1  # Initialize transaction counter

# File handling functions
# Loads transactions from a JSON file.
def load_transactions():
    global expenses
    try:
        with open("expenses.json", "r") as file:
            expenses = json.load(file)
    except FileNotFoundError:
        expenses = {}  # Initialize as empty if file doesn't exist
    except json.decoder.JSONDecodeError:
        expenses = {}  # Handle invalid JSON format

# Saves transactions to the JSON file.
def save_transactions():
    with open('expenses.json', 'w+') as file:
        json.dump(expenses, file)

# Get a valid date
def valid_date(prompt):
    while True:
        date_str = input(prompt)
        try:
            date_object = datetime.strptime(date_str, "%Y-%m-%d")
            return date_object.strftime("%Y-%m-%d")  # Return formatted date string
        except ValueError:
            print("Invalid date format. Please use YYYY-MM-DD.")

# Function to open the GUI
def open_gui():
    root = tk.Tk()
    app = FinanceTrackerGUI(root)
    app.display_transactions(app.transactions)
    root.mainloop()
```

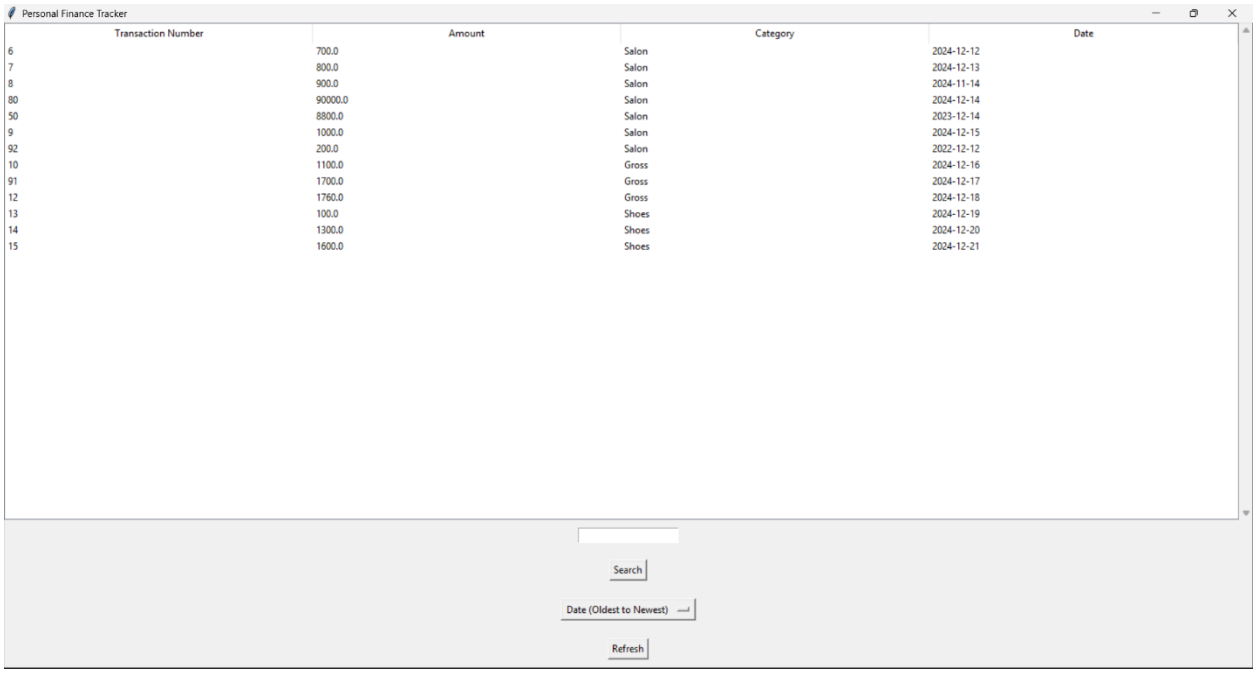
```
# main menu at the start
def main_menu():
    load_transactions()
    global transaction_counter
    # Update transaction counter based on existing transactions
    if expenses:
        transaction_counter = max(int(t["transaction_number"]) for t_list in expenses.values() for t in t_list) + 1

    while True:
        print()
        print(" ****Personal Finance Tracker**** ")
        print()
        print("1. Add Transaction")
        print("2. View Transactions")
        print("3. Update Transaction")
        print("4. Delete Transaction")
        print("5. Display Summary")
        print("6. Bulk Import Transactions from File")
        print("7. Open GUI")
        print("8. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            add_transaction()
        elif choice == '2':
            view_transactions()
        elif choice == '3':
            update_transaction()
        elif choice == '4':
            delete_transaction()
        elif choice == '5':
            display_summary()
        elif choice == '6':
            bulk_import_transactions()
        elif choice == '7':
            open_gui()
        elif choice == '8':
            print("Exiting program.")
            save_transactions()
            break
        else:
            print("Invalid choice. Please try again.")
```

3 Test Cases

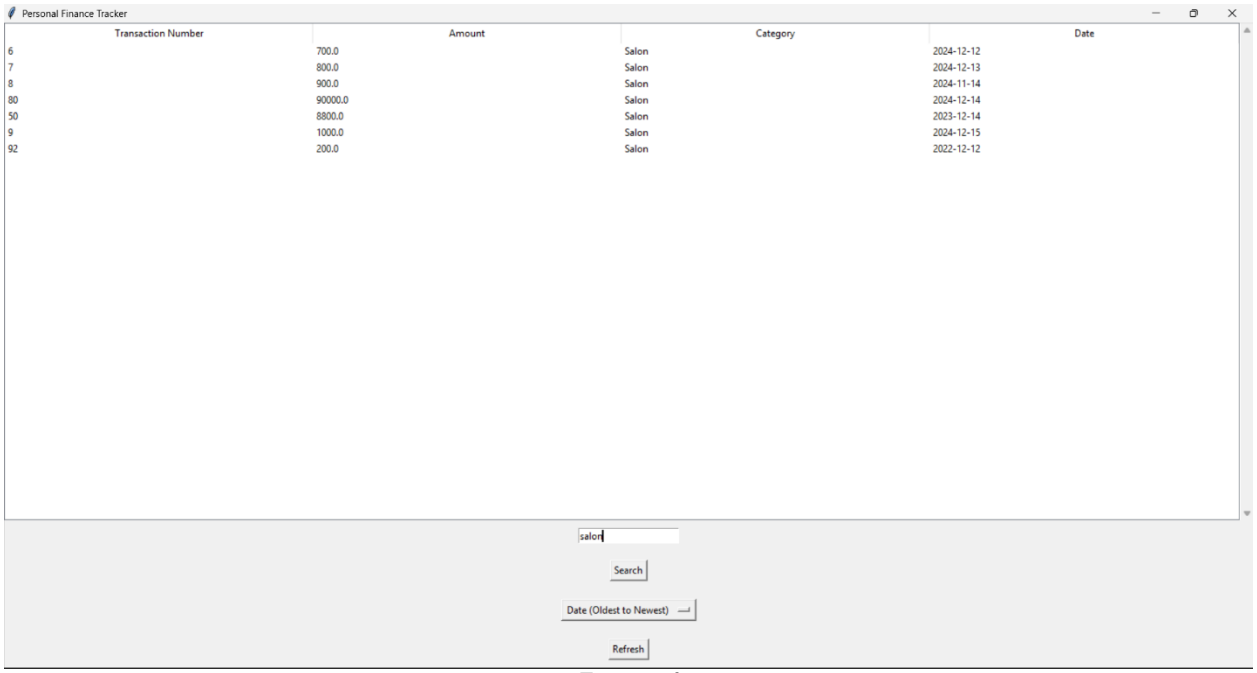
Test Case ID	Description	Input	Expected Output	Actual Output	Remark
1	Display the transactions in GUI	Input “7” in Main menu	Display transactions	Transactions displayed	Pass
2	Search for a transaction with valid input word	salon	Display the available transaction	Displayed the available transaction	Pass
3	Search for a transaction with valid input numbers	200	Display the available transaction	Displayed the available transaction	Pass
4	Search for a transaction with invalid input	food	Display "No search found." message	Display "No search found." message	Pass
5	Sort transactions by date (Oldest to Newest)	-	Transactions should be sorted by amount in ascending order	Transactions sorted by amount in ascending order	Pass
6	Sort transactions by category (A to Z)	-	Transactions should be sorted by category in ascending order	Transactions sorted by category in ascending order	Pass
7	Sort transactions by amount (Smallest to Largest) - Category wise	-	Transactions should be sorted by amount in ascending order (Category wise)	Transactions sorted by amount in ascending order (Category wise)	Pass

8	Sort transactions by transaction number (Ascending) - Category wise	-	Transactions should be sorted by transaction number in ascending order (Category wise)	Transactions sorted by transaction number in ascending order (Category wise)	Pass
9	GUI responsiveness on different screen sizes		Elements adjust properly	GUI elements adjusted	Pass
10	Return to main menu	-	When user close the GUI main menu should display	When user close the GUI main menu displayed	Pass
11	Refresh	-	When user click the refresh button table has to be refresh with current transactions	When user click the refresh button table refreshed with current transactions	Pass

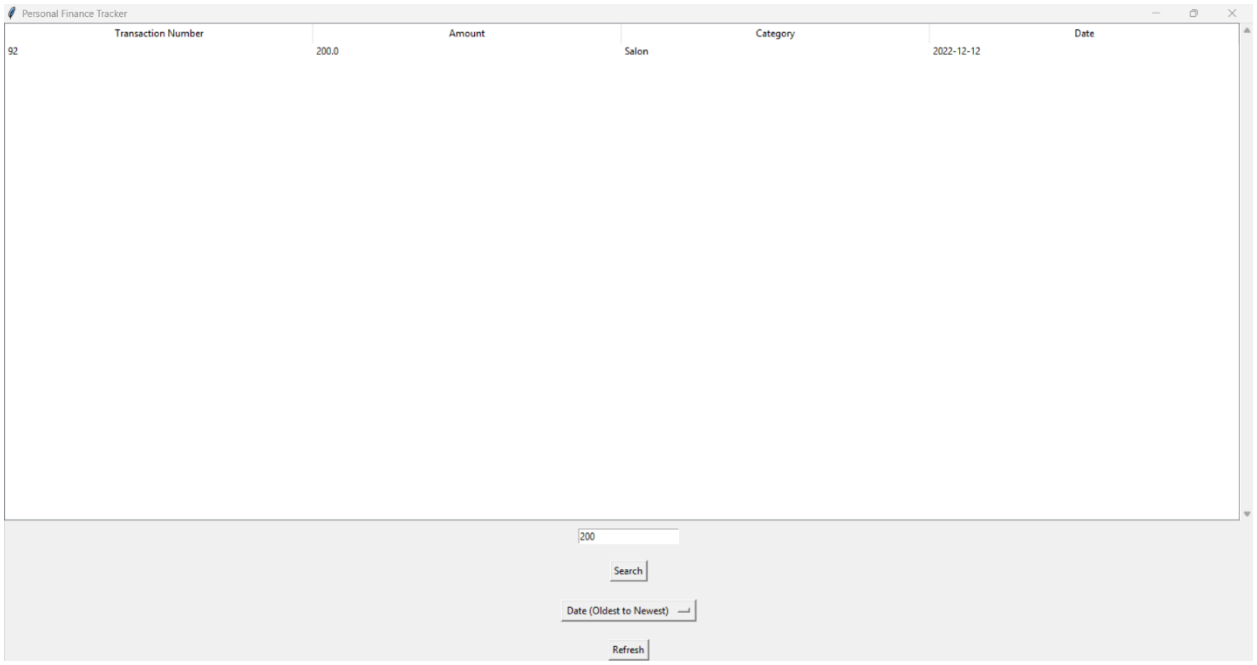
4 Screenshots for the Test Cases



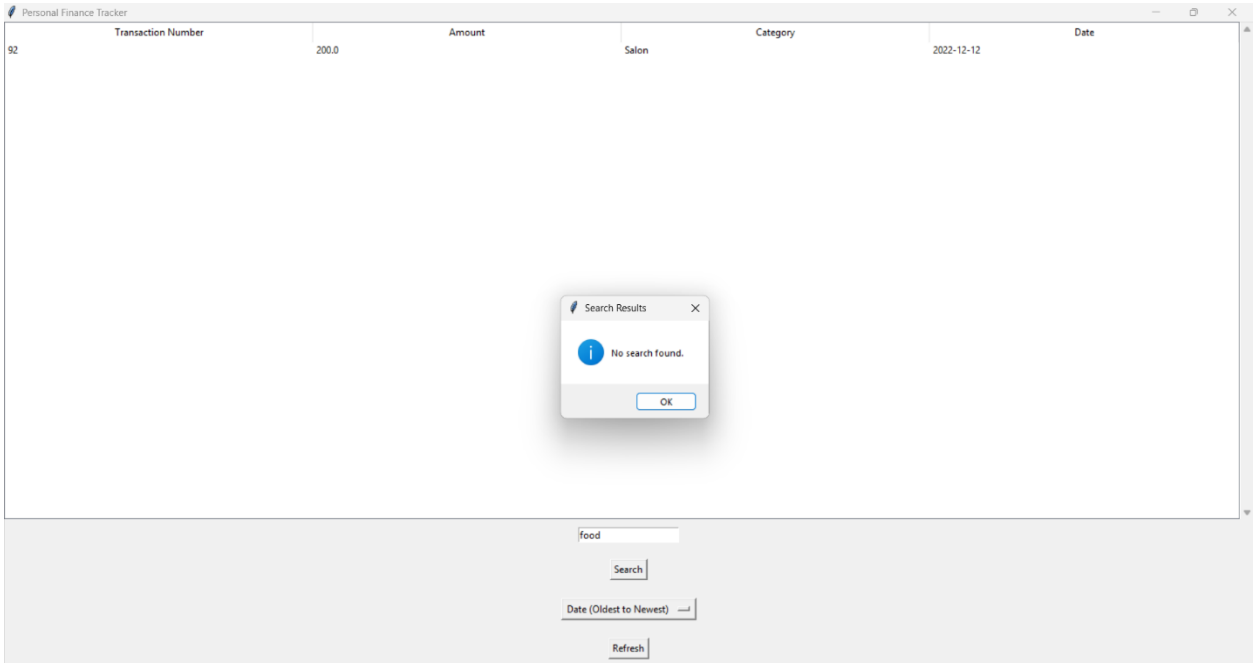
Test case 1



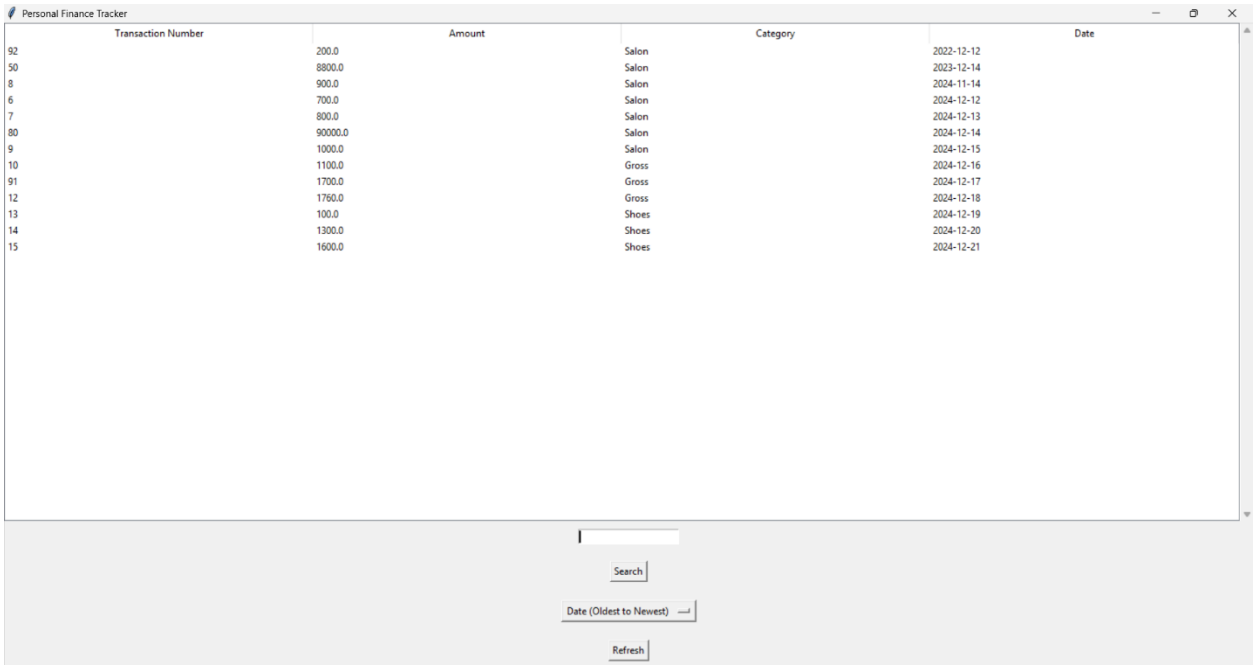
Test case 2



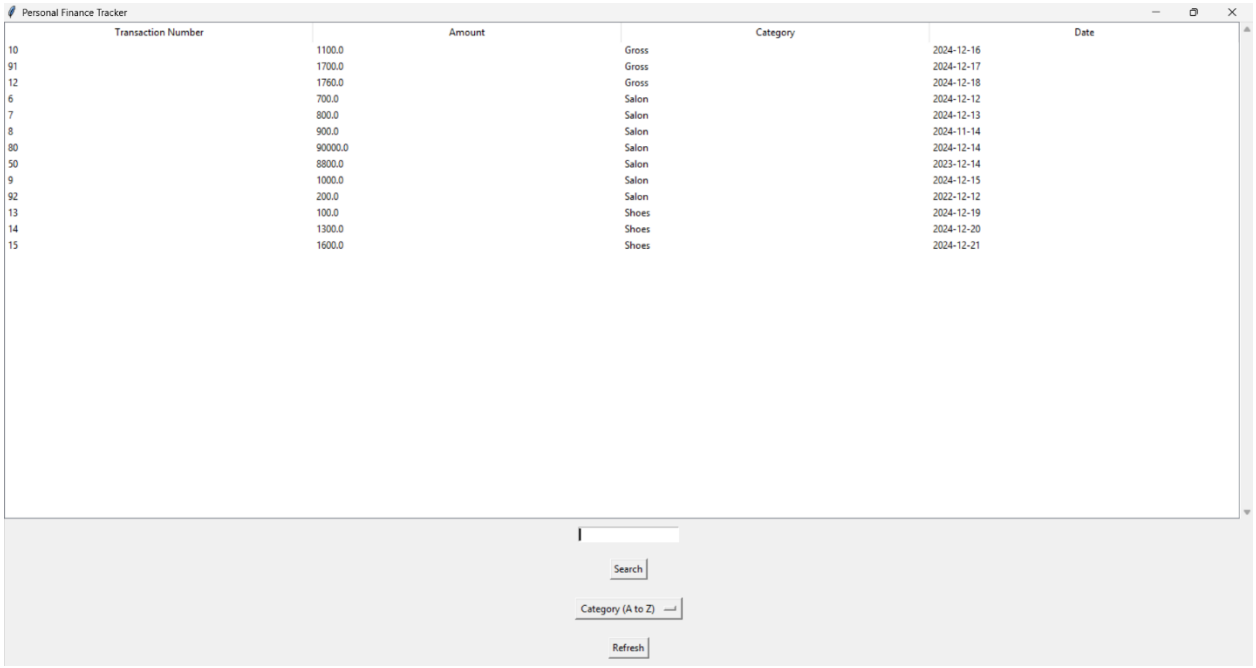
Test case 3



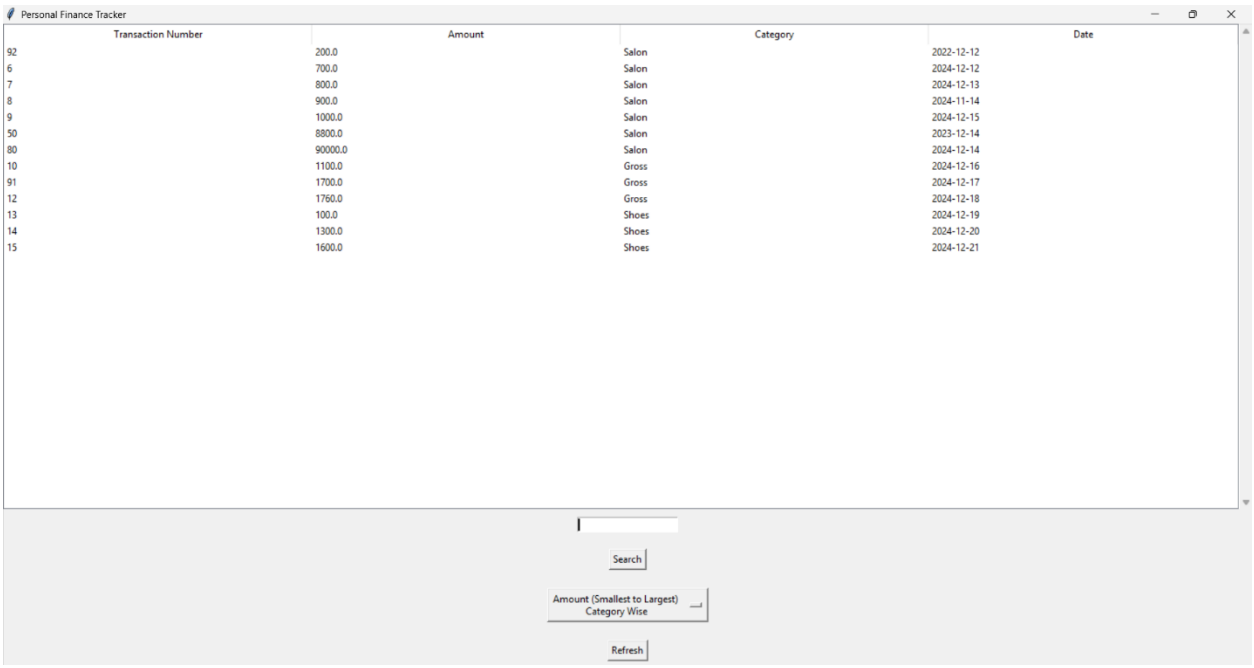
Test case 4



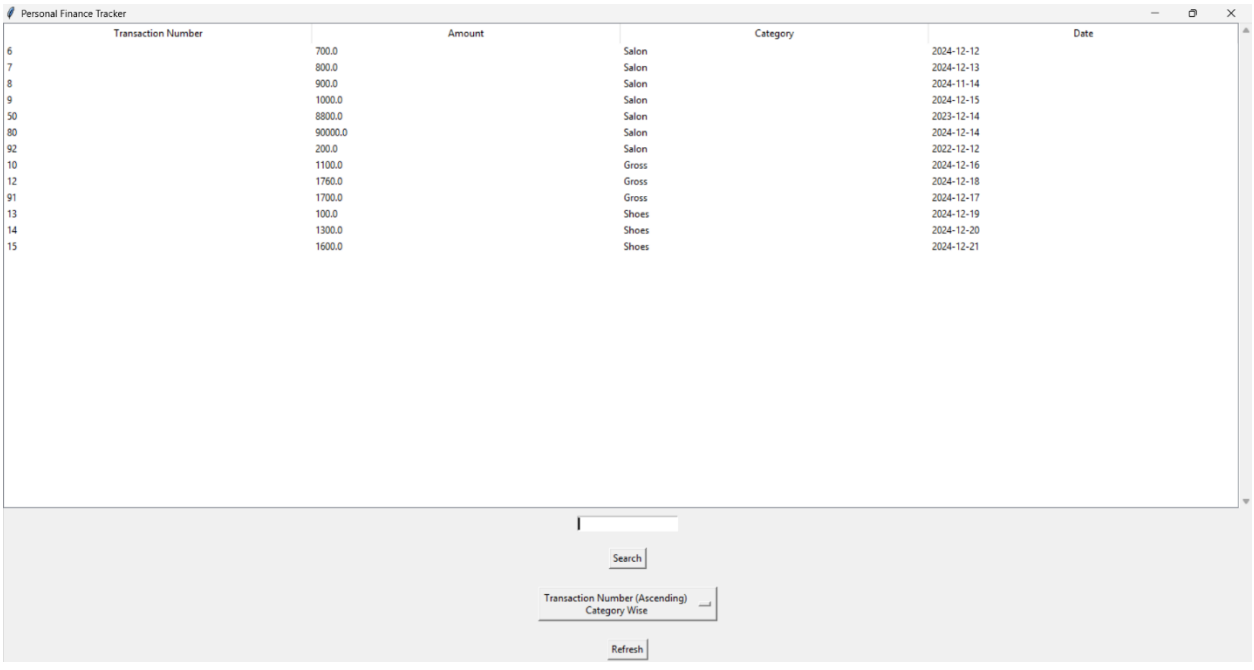
Test case 5



Test case 6



Test case 7



Test case 8

Design Decisions:

Overall Structure:

The GUI application is designed using the Tkinter library in Python to create a simple Personal Finance Tracker. It consists of a main window with widgets such as Treeview to display transactions, Entry for searching transactions, Button for search, Option menu for sorting options, and Button for refreshing the transaction table.

Class Structure:

The main class `FinanceTrackerGUI` is responsible for managing the GUI components and handling user interactions. It has functions for loading transactions from a JSON file, saving transactions, creating GUI widgets, displaying transactions in the table, filtering transactions based on search term, and sorting transactions based on selected options.

Functions:

Loading Transactions:

- The application loads transactions from a JSON file (`expenses.json`) when initialized.

Displaying Transactions:

- Transactions are displayed in a Treeview widget with columns for transaction number, amount, category, and date.

Searching Transactions:

- Users can search for transactions by entering keywords in the search bar.
- If matching transactions are not founded, it displays a message "No search found." with the message box

Sorting Transactions:

- Users can sort transactions by date (oldest to newest), category (A to Z), amount (smallest to largest), or transaction number (ascending). (Amount & Transaction number is sorting as category wise)

Running the Application:

To run the application, run the provided code in a Python environment with Tkinter installed.

Summary:

The Finance Tracker GUI provides a user-friendly interface for managing personal finances, including display, searching, and sorting transactions.