Presented By

SHAHIN SULAIMAN.M

10/18/2024

SUPER MARKET SALES DATA MANAGEMENT SYSTEM

SUPERMARKET SALES DATA MANAGEMENT SYST

NE)



**Aim of the Project:**

The primary aim of the **Supermarket Sales Data Management System** is to streamline the process of managing and recording sales transactions within a supermarket environment. This project seeks to implement an efficient and user-friendly solution for inputting, storing, and retrieving sales data using Object-Oriented Programming (OOP) principles in Python. By offering features like handling discounts, calculating total sales amounts, and saving records to files, the project aims to enhance the productivity and accuracy of sales management. Ultimately, the system simplifies the daily operations of supermarket sales management, making data handling more structured and reliable.

**Problem Statement:**

In a typical supermarket, managing daily sales data can be a challenging task. Businesses often struggle with accurately recording transaction details, applying discounts, and keeping a record of sales data for future analysis. This can lead to errors in billing, loss of critical data, and inefficiencies in understanding sales trends. Without a structured approach, it becomes difficult to monitor sales performance, apply discounts consistently, and ensure that the sales data is stored securely for future reference.

The **Supermarket Sales Data Management System** addresses these challenges by providing a digital solution that allows the staff to input sales data, automatically apply discounts, calculate totals, and save the information to a file. It reduces the chances of manual errors, ensures consistency in applying discounts, and helps maintain a comprehensive record of transactions. This enables supermarket managers to access sales data easily, analyze trends, and make informed business decisions. Additionally, by using Python's exception handling, the system manages errors gracefully, preventing crashes during data entry or file operations.

**Project Description:**

The **Supermarket Sales Data Management System** is designed to automate the process of recording and managing sales transactions in supermarkets using Python. The system uses **Object-Oriented Programming (OOP)** to represent sales data through classes, such as Sale and DiscountedSale. The Sale class handles standard transactions, while the DiscountedSale class extends it to incorporate discount functionality, demonstrating inheritance and method overriding in OOP.

The project's scope includes:

* **Input Collection**: Accepting user input for product name, quantity, price per unit, and optional discounts.
* **Calculation of Totals**: Automatically computing the total sale amount based on the quantity and unit price, and applying discounts if applicable.
* **Data Storage**: Storing each transaction record in a text file to maintain a persistent record of sales.
* **Error Management**: Implementing exception handling for invalid inputs or file writing issues, ensuring a smooth user experience.
* **User Interaction**: A simple, text-based interface that guides users through the data entry process.

**Functionalities:**

1. **Input Collection**:
   * **Description**: The system prompts users to input details like product name, quantity, and price per unit. If a discount applies, users can also enter the discount percentage. The input is validated to ensure that only correct data types are accepted (e.g., integers for quantity).
2. **Calculation of Total Sales Amount**:
   * **Description**: This functionality computes the total amount for each sale transaction. If no discount is applied, it multiplies the quantity by the price per unit. For discounted sales, it calculates the discount and subtracts it from the total amount, ensuring accurate billing.
3. **Discount Application**:
   * **Description**: By using a child class (DiscountedSale), the system can handle transactions where discounts are applied. The discount percentage is taken into account when calculating the final total, showcasing the system's flexibility in handling different types of sales.

**4.Data Storage in Files**:

* + **Description**: The system saves each sale transaction's details into a text file. This allows for long-term storage and future reference of sales records. The file-saving mechanism ensures that data is appended without overwriting previous records.

1. **Error and Exception Handling**:
   * **Description**: The system manages common errors like invalid inputs (e.g., entering text instead of numbers for quantity). It also catches file-related errors, such as issues in opening or writing to the file, and provides appropriate feedback to users.

**5.Input Versatility with Error Handling and Exception Handling:**

The system is designed to handle a variety of user inputs efficiently. For instance, it checks if the input for quantity is an integer and if the price per unit is a float, ensuring that calculations can proceed without errors. Additionally, when the user enters a discount percentage, it ensures that the value is within a reasonable range (0-100).

To manage errors, the project implements **exception handling** using try-except blocks. For example, if a user mistakenly inputs a non-numeric value when asked for the quantity, the system catches this error and prompts the user to re-enter a valid number. Similarly, if there are issues with writing data to a file (e.g., file permission issues), the system captures the error and informs the user without crashing.

**6. Code Implementation:**

The project is implemented using Python, primarily utilizing **classes and inheritance** for structured data management. The Sale class includes methods to calculate the total sale amount and display sale details. The DiscountedSale class inherits from Sale and overrides the calculate\_total method to incorporate discount logic. This use of inheritance demonstrates OOP principles like **polymorphism** and **method overriding**.

**Description**:

class Sale:

def \_\_init\_\_(self, product\_name, quantity, price\_per\_unit):

self.product\_name = product\_name

self.quantity = quantity

self.price\_per\_unit = price\_per\_unit

# Method to calculate total sale amount for the product

def calculate\_total(self):

return self.quantity \* self.price\_per\_unit

# Method to display sale details

def display\_sale(self):

return f"Product: {self.product\_name}, Quantity: {self.quantity}, Price per Unit: Rs {self.price\_per\_unit}, Total: Rs {self.calculate\_total()}"

# Child Class: DiscountedSale (inherits from Sale)

class DiscountedSale(Sale):

def \_\_init\_\_(self, product\_name, quantity, price\_per\_unit, discount\_percentage):

# Using super() to inherit from Sale class

super().\_\_init\_\_(product\_name, quantity, price\_per\_unit)

self.discount\_percentage = discount\_percentage

# Override the calculate\_total method to apply discount

def calculate\_total(self):

total = super().calculate\_total() # Get the total from the parent class method

discount = total \* (self.discount\_percentage / 100)

return total - discount

# Display sale details including discount

def display\_sale(self):

base\_sale = super().display\_sale()

return f"{base\_sale}, Discount: {self.discount\_percentage}%, Final Total: Rs {self.calculate\_total()}"

# File handling: Save sale data to a file

def save\_sale\_to\_file(sale\_info, filename="sales\_data.txt"):

try:

with open(filename, 'a') as file:

file.write(sale\_info + '\n')

except Exception as e:

print(f"An error occurred while writing to the file: {e}")

# Main function to simulate sales data input and file handling

def main():

print("Supermarket Sales Data Collection")

# Taking user input

try:

product\_name = input("Enter product name: ")

quantity = int(input("Enter quantity: "))

if quantity <= 0:

raise ValueError("Quantity must be a positive number.")

price\_per\_unit = float(input("Enter price per unit (Rs): "))

if price\_per\_unit <= 0:

raise ValueError("Price per unit must be a positive number.")

has\_discount = input("Is there a discount? (yes/no): ").strip().lower()

if has\_discount == "yes":

discount\_percentage = float(input("Enter discount percentage: "))

if discount\_percentage < 0:

raise ValueError("Discount percentage must be zero or a positive number.")

# Creating an instance of DiscountedSale class

sale = DiscountedSale(product\_name, quantity, price\_per\_unit, discount\_percentage)

else:

# Creating an instance of Sale class

sale = Sale(product\_name, quantity, price\_per\_unit)

except ValueError as e:

print(f"Invalid input: {e}")

return

# Display the sale data

sale\_info = sale.display\_sale()

print(sale\_info)

# Save sale data to a file

save\_sale\_to\_file(sale\_info)

print("Sales data saved to file.")

# Running the main function

if \_\_name\_\_ == "\_\_main\_\_":

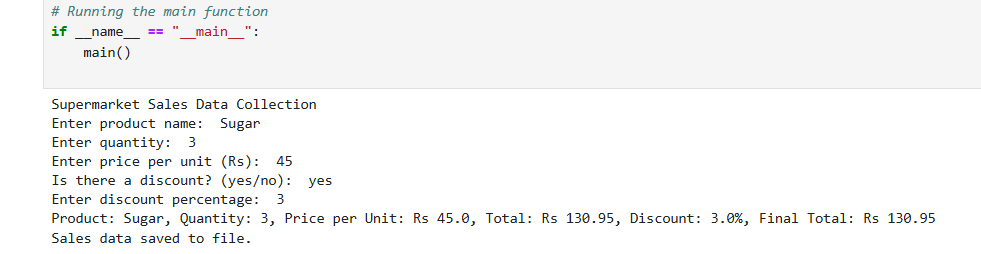
main()

The **file handling** component is implemented using the open() function with 'a' mode to append data, ensuring that previous records are not lost. Additionally, the **try-except blocks** handle potential errors during data entry and file operations, making the application robust. The code is organized in a way that keeps data processing, input handling, and output generation clearly separated, enhancing readability and maintainability.

**7. Results and Outcomes:**

**The Supermarket Sales Data Management System successfully automates the process of recording sales data, applying discounts, and storing records. The use of OOP concepts has led to a modular codebase, making the system easy to extend for future needs. Users are able to interact with the system smoothly, and the application provides feedback for errors and successful operations. The result is a more organized and efficient way to handle supermarket sales, minimizing manual errors and enhancing record-keeping.**

**OUTPUT:**



**8. Conclusion:**

The project effectively demonstrates the application of **OOP principles**, such as **inheritance** and **method overriding**, in creating a practical solution for supermarket sales management. The inclusion of **exception handling** and **file handling** ensures a robust and user-friendly experience. This project can be further enhanced by adding features like graphical user interfaces (GUIs) or integrating with databases for better data management. Ultimately, the system serves as a reliable foundation for managing sales in a supermarket environment, contributing to increased efficiency and data accuracy.