MSA UNIVERSITY
جامعة أكتوبر للعلوم الحديثة والآداب

Faculty of Engineering - CSE Department
Semester: Fall 2023 - Academic Year: 2023-2024
Module Title: Object Oriented Programming
Module Code: (CSE451/CSE462)

Instructor Name: Dr. Manal Mostafa
TA Name(s): Eng. Yomna Sameer

Report Title: Billing System

Prepared by:

Student Name: Islam Gomaa Student ID: 212495
Student Name: Mahmoud Bahaa Student ID: 210733

Table of Contents

Introduction

In the dynamic and competitive landscape of modern business operations, the precision and seamlessness of billing processes stand as pivotal components for sustained success. Billing systems serve as the financial bedrock of organizations, wielding immense influence not just in financial management but also in shaping client relationships. In this intricate interplay of commerce and client engagement, the integration of Python emerges as a transformative force, offering unparalleled adaptability and widespread acceptance in constructing billing systems that transcend conventional functionalities, simplifying procedures, and significantly minimizing the potential for errors.

The centrality of billing systems within businesses today extends far beyond the mere transactional processing of payments. They represent the nucleus of financial operations, influencing revenue streams, customer interactions, and overall organizational efficiency. Python, celebrated for its versatility and robust capabilities, stands out as an indispensable asset in the development of billing systems that not only streamline processes but also cater to the ever-evolving needs and complexities of modern enterprises.

Python's inherent flexibility and rich ecosystem of libraries and frameworks empower developers to craft bespoke billing systems tailored to specific organizational requirements. Its extensive support in data handling, manipulation, and computation ensures the accuracy and reliability necessary for error-free financial transactions. Moreover, Python's readability and collaborative community support provide businesses with the agility to swiftly adapt to changing market dynamics and incorporate innovative functionalities into their billing architectures.

Objectives

- **User-Friendly Interface:** Create an intuitive and user-friendly interface to ensure easy navigation and input for both staff and customers.

- **Readability:** Code should be well-organized that developers can understand it easily with clear comments and documentation.

- **Scalability:** The software should be designed to handle growth and increased usage. Scalable systems are easier to maintain as they adapt to changing needs.

- **Performance Optimization:** Ensure swift response times and efficient processing, even during high demand. Utilize caching, database optimization, and code efficiency enhancements for a responsive billing solution.

- **Modularity and Reusability:** Embrace OOP principles to design a modular and reusable codebase. Encapsulate functionalities into classes and objects, promoting code reusability across the billing system. This approach facilitates easier maintenance, updates, and expansion of the system while minimizing redundancy.

Main Code

```python
import datetime

class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

class Customer:
    def __init__(self, name='', address='', phone_number=''):
        self.name = name
        self.address = address
        self.phone_number = phone_number
        self.cart = []

    def add_to_cart(self, product, quantity):
        self.cart.append((product, quantity))

    def remove_from_cart(self, product_name):
        self.cart = [(product, quantity) for product, quantity in self.cart if product.name != product_name]

    def calculate_total(self):
        return sum(item.price * quantity for item, quantity in self.cart)

    def apply_discount(self, total, discount_rate):
        return total * (1 - discount_rate)

    def print_receipt(self, payment_type, cash_given=0, is_new=False):
        total = self.calculate_total()
        discount_rate = 0.05 if is_new else 0
        total_after_discount = self.apply_discount(total, discount_rate)
        return_amount = cash_given - total_after_discount if payment_type == 'cash' else 0
```

```python
        print("\n------ Receipt ------")
        print(f"Market Name: {market.name}")
        print(f"Date: {datetime.datetime.now().strftime('%Y-%m-%d')}")
        print(f"Time: {datetime.datetime.now().strftime('%H:%M:%S')}")
        print("\n{:<20} {:<10} {:<15}".format("Product", "Quantity", "Total Price"))

        for item, quantity in self.cart:
            total_price_for_item = item.price * quantity
            print(f"{item.name:<20} {quantity:<10} {total_price_for_item:<15.2f}")

        print(f"\nTotal: {total:<15.2f}")
        if is_new:
            print(f"Discounted Total: {total_after_discount:<10.2f}")
        print(f"Payment Type: {payment_type}")
        if payment_type == 'cash':
            print(f"Cash Given: {cash_given:<15.2f}")
            print(f"Return Amount: {return_amount:<15.2f}")
        print("--------------------\n")

class RegularCustomer(Customer):
    def __init__(self, name='', address='', phone_number=''):
        super().__init__(name, address, phone_number)

class NewCustomer(Customer):
    def __init__(self, name='', address='', phone_number=''):
        super().__init__(name, address, phone_number)
```

```python
class Market:
    def __init__(self):
        self.name = input("Please enter your market name: ")
        print(f"\nWelcome to {self.name} market!\n")
        self.products = []
        self.setup_products()

    def setup_products(self):
        num_of_products = int(input("How many products do you have in your market? "))
        for _ in range(num_of_products):
            name = input("Enter product name: ")
            price = float(input(f"Enter {name}'s price: "))
            self.add_product(Product(name, price))

    def add_product(self, product):
        self.products.append(product)

    def find_product(self, product_name):
        for product in self.products:
            if product.name == product_name:
                return product
        return None

    def display_menu(self):
        print("\nPlease choose an action:")
        print("  [add] - Add a product to the cart")
        print("  [remove] - Remove a product from the cart")
        print("  [total] - Display the current total")
        print("  [print] - Print the bill and proceed to payment")
        print("  [quit] - Exit without printing the bill")
```

```python
def start_billing_process(self):
    is_new = input("Are you a new customer? (yes or no): ").strip().lower() == 'yes'
    if is_new:
        customer = NewCustomer()
        customer.name = input("Please, enter your name: ")
        customer.address = input("Please, enter your address: ")
        customer.phone_number = input("Please, enter your phone number: ")
    else:
        customer = RegularCustomer()
```

```
        while True:
            self.display_menu()
            action = input("What would you like to do? ").strip().lower()
            if action == 'print':
                break
            elif action == 'add':
                product_name = input("Enter the name of product: ").strip()
                product = self.find_product(product_name)
                if product:
                    quantity = int(input("Enter quantity: "))
                    customer.add_to_cart(product, quantity)
                else:
                    print("Product not found.")
            elif action == 'remove':
                product_name = input("Enter the name of product to remove: ").strip()
                customer.remove_from_cart(product_name)
                print(f"{product_name} has been removed from the cart.")
            elif action == 'total':
                print(f"Current total: {customer.calculate_total():.2f}")
            elif action == 'quit':
                print("Exiting without printing the bill.")
                return
            else:
                print("Invalid action. Please try again.")

        payment_type = input("Enter payment type (cash/credit): ").strip().lower()
        cash_given = 0
        if payment_type == 'cash':
            cash_given = float(input("Enter the amount of cash given: "))
        customer.print_receipt(payment_type, cash_given, is_new=is_new)

market = Market()
market.start_billing_process()
```

Result and Test Cases

Test Case 1: Customer Registration and Handling for New Customers

```
Please enter your market name: msa

Welcome to msa  market!

How many products do you have in your market? 3
Enter product name: pizza
Enter pizza's price: 150
Enter product name: water
Enter water's price: 7
Enter product name: coffee
Enter coffee's price: 25
Are you a new customer? (yes or no): yes
Please, enter your name: islam
Please, enter your address: 20 ahmed mohamed street
Please, enter your phone number: 01024402461
```

Test Case 2: Customer Registration and Handling for regular Customers

```
Please enter your market name: msa

Welcome to msa market!

How many products do you have in your market? 3
Enter product name: pizza
Enter pizza's price: 150
Enter product name: coffee
Enter coffee's price: 25
Enter product name: water
Enter water's price: 7
Are you a new customer? (yes or no): no

Please choose an action:
  [add] - Add a product to the cart
  [remove] - Remove a product from the cart
  [total] - Display the current total
  [print] - Print the bill and proceed to payment
  [quit] - Exit without printing the bill
What would you like to do?
```

Test Case 3: Billing Menu functionality validation

```
What would you like to do? add
Enter the name of product: coffee
Enter quantity: 1

Please choose an action:
  [add] - Add a product to the cart
  [remove] - Remove a product from the cart
  [total] - Display the current total
  [print] - Print the bill and proceed to payment
  [quit] - Exit without printing the bill
What would you like to do? total
Current total: 25.00
```

```
Please choose an action:
  [add] - Add a product to the cart
  [remove] - Remove a product from the cart
  [total] - Display the current total
  [print] - Print the bill and proceed to payment
  [quit] - Exit without printing the bill
What would you like to do? remove
Enter the name of product to remove: coffee
coffee has been removed from the cart.

Please choose an action:
  [add] - Add a product to the cart
  [remove] - Remove a product from the cart
  [total] - Display the current total
  [print] - Print the bill and proceed to payment
  [quit] - Exit without printing the bill
What would you like to do? total
Current total: 0.00
```

Test Case 4: Print Bill for Each type of customers

```
------ Receipt ------
Market Name: msa
Date: 2023-12-23
Time: 07:42:29

Product              Quantity   Total Price
pizza                2          300.00
water                2          14.00

Total: 314.00
Discounted Total: 298.30
Payment Type: cash
Cash Given: 350.00
Return Amount: 51.70
--------------------
```

```
------ Receipt ------
Market Name: msa
Date: 2023-12-23
Time: 07:44:43

Product              Quantity   Total Price
pizza                2          300.00
water                5          35.00

Total: 335.00
Payment Type: credit
--------------------
```

Conclusion

The successful realization of this billing system stands as a testament to the seamless fusion of Object-Oriented Programming (OOP) principles with the versatile capabilities of Python. Python's innate support for OOP has been instrumental in shaping the architecture of this system, facilitating the cultivation of crucial attributes like modularity, reusability, and scalability, which form the bedrock of an advanced billing solution.

The coherent integration of Python's OOP functionalities has bestowed the billing system with a structured, sustainable, and adaptable framework. This integration expedited the development lifecycle, enhanced code readability, and consistently adhered to industry best practices. As a result, it culminated in the creation of a sophisticated billing solution adept at not only fulfilling the organization's dynamic requirements but also adapting seamlessly to evolving business landscapes.

References

- Real Python (2023) *Object-oriented programming (OOP) in python 3*, *Real Python*. Available at: https://realpython.com/python3-object-oriented-programming/
- Vanderheyden, T. (2022) *Object-oriented programming in Python (oop): Tutorial*, *DataCamp*. Available at: https://www.datacamp.com/tutorial/python-oop-tutorial/
- Krishna, A. (2022) *Object-oriented programming in python*, *freeCodeCamp.org*. Available at: https://www.freecodecamp.org/news/object-oriented-programming-in-python/
- *Python - object oriented* (no date) *Online Tutorials, Courses, and eBooks Library*. Available at: https://www.tutorialspoint.com/python/python_classes_objects.htm/