Shop – Programming Paradigms Comparison

Assignment project for Multi-Paradigm Programming module at GMIT, 2020

Author: Slawomir Sowa

Email: G00376519@gmit.ie

Subject:

Add additional functionality to the shop program developed in the lecturer video series.

Functionality:

- The shop CSV should hold the initial cash value for the shop.
- Read in customer orders from a CSV file.

- That file should include all the products they wish to buy and in what quantity
- It should also include their name and their budget.

- The shop must be able to process the orders of the customer.
    - It is important that the state of the shop be consistent.
        - You should create customer test files (CSVs) which cannot be completed by the shop e.g. customer wants 400 loaves of bread but the shop only has 20, or the customer wants 2 cans of coke but can only afford 1.
        - If these files don't exist marks penalties will be applied.
    - Know whether or not the shop can fill an order
        - Thrown an appropriate error.
- Operate in a live mode, where the user can enter a product by name, specify a quantity, and pay for it. The user should be able to buy many products in this way.

1. Project Description

The programming paradigm defines the way of looking programmer on control flow and execution computer program. The task was to develop the Shop program using two different programming paradigms and two different languages so that the user interface and functionality of the Shop looked the same from the user level.

2. The Procedural Paradigm

As a paradigm of procedural programming, the Shop program was developed in C and Python.
The procedural programming paradigm is a programming paradigm that recommends dividing code into procedures i.e. fragments that perform strictly defined operations. Procedures should not use global variables (if possible), but retrieve and pass all data (or pointers to them) as call parameters.

Comparing both languages, I must admit that Python is much more easy to use. It has a simpler syntax and a large library of built-in functions. In C we have to create many of these functions ourselves.

The problem I encountered when creating a C program was that the return value has to be declared at the beginning of the function, and it is not possible to return two different data types.

```c
struct Product getProduct(struct Shop s, char* pname)
/*
    Function to check the shop for a product and return it
    Parameters:
        s : struct Shop : datatype struct contain shop stock and cash
        pname: char : searched item in shop stock

    Return:
        p : struct Product : datatype struct contain product name and price
*/
{
    struct Product p;
    // loop over a shop index and compare related to index product name to
searched phrase
    for (int i = 0; i < s.index; i++){
        // if strcmp returns 0 product was found and product p is returned
```

```
        if(strcmp(s.stock[i].product.name,pname)==0){
            p = s.stock[i].product;
        }
    }
    return p;
};
```

The above function, through for loop, compare the searched product with the products in the Stock Shop, and if a product is found it return the type struct Product. However, I was unable to find a solution in a situation where the product is not in stock.

So it was necessary to create a new function *getProductName()* that returns a value of type char.

```
char* getProductName(struct Shop* s, char *pname)
/*
    Function to check the shop for a product and return it
    Parameters:
        s : struct Shop : datatype struct contain shop stock and cash
        pname: char : searched item in shop stock

    Return:
        p : char : product name
*/
{
    for(int j = 0; j < s->index; j++){
        if(strcmp(s->stock[j].product.name,pname)==0){
            return s->stock[j].product.name;
         }
   }
    return "NULL";
}
```

Both functions work very similarly, but *getProductName()* returns "NULL" if the product is not found in the Shop Stock, this allowed me to return the information to the user that the product is out of stock.

For Python programming, this problem does not occur, Python can return different data types. The *getItem()* function compares the searched product name with the products in the Shop using a for loop. When the names match, the function returns the ProductStock type. Thanks to this, I have access to information on the price and number of available items of a given product. If the product is not found, the value *None* is returned. Which allows displaying the appropriate message to the user.

```
def getItem(shop, name):
    '''
        I introduced this method in order to be able to quickly find a prod
uct from the client's list in the store, searching by the product name.
        Thanks to this, I will have access to the price and number availabl
e in the store for the product that the customer is looking for.

        Parameters:
            shop : Shop : instance of Shop class
            name : str  : product name

        Returns:
            i :  ProductStock : if True instance of ProductStock class, Non
e otherwise

    '''

    # iterate over the shop.stock,
    for i in shop.stock:
        # for each ProductStock compare product.name with searched string
        if i.product.name == name:
            # if found return ProductStock
            return i
    # if not found return None
    return None
```

3. Object oriented programming Paradigm

For this project I chose Python as the Object Oriented Programming language.

Object Oriented Programming is a paradigm where entities are presented as objects that have both data, also known as attributes or properties and functionality knows as methods. A feature of objects is that an object's own procedures can access and often modify the data fields of itself. In OOP, computer programs are designed by making them out of objects that interact with one another. *[3]*

The most popular OOP languages are class-based, which means that objects are instances of classes. Objects may represent a shop, customer, product or any other data. *[5]*

Many concepts that are available for OOP are not available for procedural paradigm.

1. Abstraction - is the concept of object-oriented programming that "shows" only essential attributes and "hides" unnecessary information. The main purpose of abstraction is hiding the unnecessary details from the users. [4]

2. Encapsulation - we can restrict access to methods and variables. This prevents data from direct modification which is called encapsulation [6]
   Method *self.__getItem()* from Shop class is a private method, Is accessible only from the inside of the class.

3. Inheritance - It is the process to create new classes or subclasses from an existing class. The existing class is called the parent class, and the derived class is called sub-class or inherited class. The inherited class have the behaviour of parent class, and in addition, can have its own characteristics.[5]
   In my code, the *LiveCustomer()* class inherits from the *Customer()* class, thanks to which I was able to add additional functionality in the form of downloading customer data from the user's console, at the same time leaving the functionality from the *Customer()* class, i.e. calculating the purchase costs and a special method *__repr __()* to print customer.

4. Polymorphism – is the ability to leverage the same interface for different underlying forms such as data types or classes. This permits functions to use entities of different types at different times.[7]

For Customer class I redesigned *__init__()* method:

```
class Customer:
    '''
        Class to create and store information about customer
    '''
    def __init__(self, path = None):
        '''
            Parameters:
                path : str : Path to the CSV file, Default = None
        '''
        # shopping list variable
        self.shopping_list = []

        # if path to the CSV file is provided by the function caller
        # then customer is created from that CSV
        # When path is not provided live customer can be created
        if path:
            self.read_csv(path)
```

By assigning the path variable to None as default value, I will be able to inherit functionality from Customer Class to a new Class *LiveCustomer(Customer).*

Customer class is a parent class and a LiveCustomer() is a child class that inherits from Customer class. The *create_live_customer()* method from the *LiveCustomer()* class is used to create a live client, data are retrieved from the user and setting instance variables with name and budget.

```
def create_live_customer(self):
        '''
            Creates a client using user input from the console, the user is pr
ompted for name and budget
        '''
        # gets from user
        self.name = input("\nPLEASE ENTER YOUR NAME: ")
        self.budget = float(input("\nPLEASE ENTER YOUR BUDGET: "))
```

The shopping list is then created using the *create_shopping_list()* method. Using the for loop, the user enters the name of the product and its quantity, the action is repeated until the user presses the "n" key

```
def create_shopping_list(self):
        '''
            Creates a shopping list for live mode. The user is asked to enter
the name of the product
            and the quantity he wants to buy. Thanks to the use of the loop, i
t is possible to add many products.
            The loop ends when the question "WOULD YOU LIKE TO BUY ANOTHER PRO
DUCT?" the user presses the "n" key
        '''
        choice = -1
        # While loop breaks when choice == "n"
        while choice != "n":
            customer_item = input("ENTER PRODUCT NAME: ")
            customer_quantity = int(input("\nENTER PRODUCT QUANTITY: "))
            # create instance of a Product class
            p = Product(customer_item)
            # create instance of a ProductStock class
            ps = ProductStock(p,customer_quantity)
            self.shopping_list.append(ps)
            choice = input("\nWOULD YOU LIKE TO BUY ANOTHER PRODUCT? y/n: ")
```

The code below shows a situation in which we have access to methods from parent class through inheritance.

live_customer instance is created from LiveCustomer() class, and create_live_costomer() method is called to set the customer name and customer budget. create_sopping_list() is called to get information for user about the products and quantity.

Last line of the code is a calculate_costs() method called, this method is from the Customer Class, not from the *LiveCustomer()* class, but thanks to inheritance we have access to all methods from the parent class. Inheritance is a big advantage of the OOP paradigm

```
live_customer = LiveCustomer()

# call create_live_customer method where name and budget are entered
live_customer.create_live_customer()

print("\nWhat Would you likte to buy?\n")

# call create_shopping_list method where shopping list is created
live_customer.create_shopping_list()

# Method calculate_costs is inherited from Customer Class
live_customer.calculate_costs(shop.stock)
```

4. Comparison of Procedural and OOP paradigm

Object-oriented Programming is a programming language that uses classes and objects to create models based on the real world environment. In OOPs it makes it easy to maintain and modify existing code as new objects are created inheriting characteristics from existing ones. On other hand Procedural Oriented Programming is a programming language that follows a step-by-step approach to break down a task into a collection of variables and routines (or subroutines) through a sequence of instructions. Each step is carried out in order in a systematic manner so that a computer can understand what to do. *[1]*

- Procedural programming has Local variables, sequence, selection, iteration, and modularisation. Object-oriented programming has Objects, methods, message passing, information hiding, data abstraction, encapsulation, polymorphism, inheritance.
- In Object-oriented programming, there are three accessing modes – Public, Private, and Protected. There are no such access modes in Procedural programming.
- In Object-oriented programming, various functions can execute simultaneously. In procedural programming, there is a systematic approach in which functions get executed step-by-step.
- In Object-oriented programming, data and functions are accessible within the same class while in procedural programming, data can move freely.
- Object-oriented programming is more secure than procedural programming, because of the level of abstraction or we can say data hiding property. It limits the access of data to the member functions of the same class. While there is no such data hiding in the procedural programming paradigm.

- Object-Oriented programming follows the **bottom-up approach** while Procedural programming follows the **top-down approach** while designing a program.
- In Object-oriented programming, the program is divided into small entities called objects whereas in Procedural programming the program is divided into sub-procedures.

References

1. Difference between C and Python; https://www.tutorialspoint.com/difference-between-c-and-python
2. C Memory Management; https://stackoverflow.com/questions/24891/c-memory-management
3. Object-Oriented Programming; https://en.wikipedia.org/wiki/Object-oriented_programming
4. What is abstraction in OOP?; https://www.guru99.com/java-data-abstraction.html
5. Difference: Procedural and OOP Programming; December 18, 2020; https://www.codingninjas.com/blog/2020/12/18/difference-procedural-object-oriented-programming/
6. Python Object Oriented Programming; https://www.programiz.com/python-programming/object-oriented-programming
7. How To Apply Polymorphism to Classes in Python 3; April 13, 2017; https://www.digitalocean.com/community/tutorials/how-to-apply-polymorphism-to-classes-in-python-3