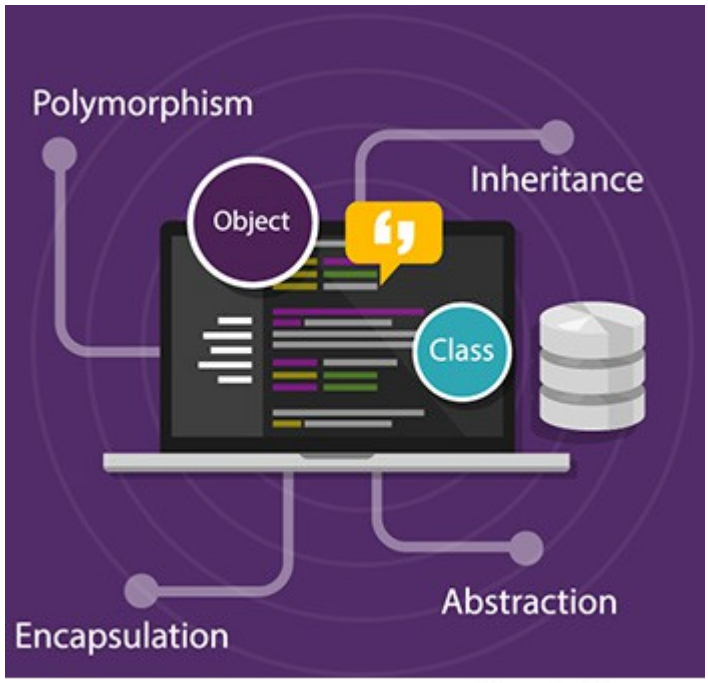
Shop – programming paradigms comparison

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

Lecturer: dr Dominic Carr

Author: **Andrzej Kocielski**
Email: G00376291@gmit.ie
Submission: December 2020



Introduction

This report forms the descriptive part of my assignment project to Multi-Paradigm Programming module, Galway-Mayo Institute of Technology, 2020.

Assignment objectives

The project concerns the creation of a simulation of a shop in three different approaches:

- C language – procedural programming,
- Python language – procedural programming,
- Python language – object-oriented programming.

The objective of the assignment was to become aware of and more conversant with different paradigms of programming through required research and the practical project.

Functionality

The program simulates a shop and processes customers' orders.

The shop is characterised by the stock of items for sale and current cash. The purchase can be realised two folds: by reading customer data from a file or in live (interactive) mode, where the orders are typed in by the user.

The program must ensure required checks for successful purchase, like sufficient stock or sufficient customer's cash. Each successful purchase should update the two parameters. The shop parameters should remain consistent as long as the program runs.

The program's functionality (and UX) should be the same for each implementation.

Assumptions (selected)

- Each time a customer data is read from a file, his/her budget and shopping list restores (no memory of previous shopping).
- Shopping lists are treated as an entities. This is manifested when the customer has the insufficient budget for all the desired items and quantities. In such a case the entire shopping is aborted (with an adequate message).
- User inputs are not checked against data type and case types. For instance, product names must be typed precisely to be recognised.

Project delivery

The project is delivered as a package of relevant files via submission link. The package consists of source code for each of the programming languages/paradigms, data files (csv) and this report.

Used paradigms

This report compares procedural programming (PP) and object-oriented programming (OOP) paradigms, without consideration to programming languages.

Procedural

The procedural paradigm origins from imperative programming. Compared to the imperative, PP paradigm gives extra enhancement that allows for function calls.

Functions (known also as procedures) can be perceived as collections of instructions. The idea and a good practice behind functions are that they specialise in a specific task and do not do anything more.

The communication between the function is done via calls. The functions can be called as needed during the program execution. An example of a function call in C :

```
total_cost = print_customers_details(&customer_C, &sh);
```

This functionality opens the possibility to reuse code blocks and to control the program flow in a non-linear way, rather than execution of commands in a predefined sequence.

Functions can accept parameters (variables) passed on when the function is called, and transfer the data from one function to another. This results in code reusability and reduction of error likelihood.

PP paradigm is based on data structures. In contrast to OOP, the datatypes in the procedural paradigm do not contain any methods. Functions are independent of the data structures.

Object-oriented

Object-oriented programming is the paradigm, where programming entities are represented as objects. In comparison to procedural programming, which focuses on logic and functions (how the program does), OOP revolves around the objects and answers the question of what the program is to do.

The value of the OOP approach is particularly visible when the complexity of the program grows. It offers many advantages, such as objects promote a higher quality code (reduce the likelihood of an error), higher reusability, streamline cooperation between developers or a better code maintainability.

Objects communicate with one another by the means of message passing, whereby a method (with passed parameters as needed) acts upon the receiver object:

`receiver_object.method_name(argument1, argument2, ...)`.

OOP takes advantage of several concepts, not available for procedural paradigm, namely inheritance, polymorphism, encapsulation and abstraction.

Inheritance defines the relationship between certain objects in hierarchial, family-tree-like structure. Children's classes take all properties and all functionality after the parent class. Example of inheritance in the Shop program was shop-object -> product_stock-object -> product-object.

Polymorphism gives the chance to minimise the code by creating different forms of an object, used depending on the needs. It allows for children to deviate from the inherited properties or behaviour. In other words, the children objects take after the parent object but can have individual properties or functionality if required.

Encapsulation is a way of protection (access control) from changing the values of the variable from outside of the class (certain properties are private to the class). The idea of encapsulation is to improve the quality of the code by preventing from intentional or unintentional changing the state of program variables.

Abstraction in terms of OOP simplifies what is inherited from the parent class in such a way it takes after only data and processes that are relevant to this particular class.

Main similarities

Both paradigms encourage to split the code into smaller blocks (functions in procedural and objects in OOP), that carry out a specific task.

The code blocks can be evoked as needed in order to control the flow of program.

Cross-reference is possible between functions and between objects. In both paradigms the references is performed through so-called chain access, for example:

- in Python PP: `sub_total_full = cust_item.quantity * sh_item.product.price`
- in Python OOP: `cust_item.product.price = shop_item.unit_price()`

It is a good practice to keep either the functions or methods performing a specific task only.

Both PP and OPP allow for modular programming (built up from code blocks)

Main differences

In procedural paradigms data structures and functions are separate entities, whereas in OOP – data structures and methods are combined into a single entity – object.

OOP offers extra functionality via concepts of inheritance, encapsulation, polymorphism and abstraction. No counterpart exists in the procedural paradigm. There are consequences of these concepts (for the likes of OOP there is better scalability, maintainability, cooperativity, security but also the complexity of the code).

Procedural paradigm may be more suitable for simpler and smaller projects.

In OOP the relationship between data (variables) can be inscribed inside the object and form a logical linkage, whereas in PP the relationship is achieved through arguments passed on calls. That means in PP it is up to the programmer to ensure error-free communications.

Summary

There is a variety of programming paradigms in existence. Under this project, three different approaches were tested (C procedural, Python procedural, Python OOP). After the completion of the project, I can draw the following conclusions.

- Each paradigm has pros and cons, with a preferable area of application. For a given programming problem, the most appropriate paradigm should be selected, based on the requirements, level of complexity, available resources, etc.
- Both C and Python languages support the procedural paradigm. However, even under the same paradigm, the implementation of the solution may vary significantly, on account of the language specifics (Python is more modern and higher level language than C).
- Procedural paradigm appears to be significantly easier to follow compared to OOP for relatively simple programming problems. However, the implementation of a solution increases as the complexity of the problem increases.
- OOP may be more difficult to comprehend conceptually and, even more, to implement in practical terms. It is particularly advantageous and useful for complex programming problems.
- OOP is more scalable and adaptive as the problem complexity increases. This is achieved through the introduction of objects (classes and instances) as well as the OOP concepts. Once a class is defined (data and methods template), it can be reused in a controlled manner elsewhere.
- OPP is less error prone, due to built-in mechanisms (the OPP concepts mentioned above).
- The paradigms chosen for a given programming problem will affect the way of thinking of the problem, how the problem is addressed and how the program is structured.

References

- Dominic Carr – Project brief; online at: https://learnonline.gmit.ie/pluginfile.php/200466/mod_resource/content/0/Assignment%201%20-%20Shop.pdf

- Dominic Carr – Lecture materials; online at: https://learnonline.gmit.ie/course/view.php?id=1902
- Nathan Emberton - Computer Hope – Dictionary; online at: https://www.computerhope.com/jargon.htm
- Wikipedia contributors – Procedural programming; online at: https://en.wikipedia.org/wiki/Procedural_programming
- Wikipedia contributors – Object-oriented programming; online at: https://en.wikipedia.org/wiki/Object-oriented_programming
- Wikipedia contributors – Comparison of programming paradigms; online at: https://en.wikipedia.org/wiki/Comparison_of_programming_paradigms
- Lili Ouaknin Felsen – Functional vs Object-Oriented vs Procedural Programming; online at: https://medium.com/@LiliOuakninFelsen/functional-vs-object-oriented-vs-procedural-programming-a3d4585557f3