

DCU School of Computing

Assignment Submission

Student Name(s): Khrissian Joi Neuda
Student Number(s): 17339711
Programme: BSc in Computer Applications
Project Title: Comparative Programming Languages - Assignment 1
Module code: CA341
Lecturer: David Sinclair
Project Due Date: Nov 25 2019

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at <http://www.library.dcu.ie/citing&refguide08.pdf> and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

I/me/my incorporates we/us/our in the case of group work, which is signed by all of us.

Signed: Khrissian Joi Neuda





Comparing Imperative and Object-Oriented Programming

The two languages I have chosen for this assignment are C# and Go. I created an object-oriented implementation of a to-do list program with the use of C#, as well as an imperative implementation using Go. C# is an object-oriented language while Go is constructed with imperative principles. I wanted to use languages that I had no experience with to avoid bias in my analytical comparison of the two implementations.

Object Oriented Programming is a programming paradigm based on the concepts of “objects”. It revolves around the 4 principles, encapsulation, abstraction, inheritance, and polymorphism. Imperative Programming is also another programming paradigm, which tells the computer how to “accomplish” a program with the use of statements. Object-Oriented Programming support objects, while Imperative does not. From the definitions of these programming paradigm, it is clear that there are already differences between the two.

Both the imperative and object-oriented to-do list programs were created to act the same way towards the user. When the programs are executed, an explanation for the user with commands on how to navigate through the to-do list is displayed. The user chooses the type of to-do they want to submit by either entering ‘event’ or ‘task’. The program then provides more prompts to the user, suited to the to-do they submitted. These inputs are required to fulfil the submission of a to-do list into the program. After the user has successfully managed to enter a to-do to the program’s list, they are able to add more to-do events and tasks. By typing in “view” the user will be able to see the contents of the to-do list (the tasks and events they have entered). The command “next” displays the oldest (first) to-do item the user has submitted and immediately removes it from the list. This program can be exit with the command “quit”.

The first part of both programs is that it first deals with user prompts. In the object-oriented implementation it encounters the main class which initialises a ToDoManager object, which then calls the run method of the ToDoManager object. While the imperative implementation encounters the main function. Both implementations display the commands the user will need to navigate through the program.

```
CA341-Assignment1/Imperative on  master  via  v1.10.4
→ ./Imperative
Hi there!
Welcome to the To-Do-List manager.
To exit enter 'quit'.
If you would like to view and remove the first item in your To-Do list, enter 'next'
If you would like to view all your items in your To-Do list, enter 'view'
To add an Event, enter 'event'
To add an Task, enter 'task'

```

As the ToDoManager object is initialised in the OO implementation, an attribute toDoList of type Queue is created. This object toDoList in turn initialises a List (a type of collection in C#) called myList. However, the slice that acts like a Queue in the imperative version called myList (same name as OO), is created once the program is run. Both myList in OO and imperative require pre-defined types, which means that it does not take elements other than the elements with the types it was assigned to. The OO myList was made to contain only *objects* of type Todo, whereas the Imperative myQueue was made to contain only *structs* of type Todo.

There is a similarity with types in this segment of the programs. Both the OO and the imperative require pre-defined types for storing data in myList. This was handled easily in the imperative implementation as I did not require to create a constructor to initialise the myList slice unlike the object-oriented implementation.

```
type Todo struct {  
    task *Task  
    event *Event  
}  
  
var myList []Todo
```

```
public class Queue  
{  
    public List<Todo> myList;  
    public Queue()  
    {  
        this.myList = new List<Todo>();  
    }  
}
```

Once the user has entered the type of to-do item they want to enter, the ToDoManager run method in OO and the main function in imperative handles which method and functions the program should execute to gain information from the user about the to-do item they want to include into myList. In OO, the List collection, myList takes in only the type Todo, which takes in attributes “date” and “startTime”, which are data required in both the event to-do item, and the task to-do item. However since myList can only take Todo types, but Todo types can’t take enough information for a valid task or event to-do items, inheritance was required. Subclasses namely Event and Task were created which inherited from the base class, Todo are able to store other information required for each to-do item, with the use of their constructors. This inheritance allows the program to create Event and Task objects and add them to the Queue object toDoList despite only accepting Todo objects.

```
public class Todo  
{  
    public Tuple<int, int, int> date = new Tuple<int, int, int>(0,0,0);  
    public Tuple<int, int> startTime = new Tuple<int, int>(0,0);  
  
    public Todo(Tuple<int, int, int> date, Tuple<int, int> startTime) {  
        this.date = date;  
        this.startTime = startTime;  
    }  
}
```

In comparison to the imperative version, myList takes only a ToDo type struct which contain pointers of created types Event and Task (both). Depending on the type of to-do item the user wants to “enqueue” or add, once the information has been gathered, the program creates a struct of the to-do item, and then creates a ToDo struct of containing a pointer of the struct of the to-do item and nil for the latter. For example, if the user chose to add an Event, an event struct will be created with the user’s inputs. This event struct will then be added to a new ToDo struct, where the event field is filled with a pointer of the event struct, while the task field is filled with a nil pointer. A pointer had to be used to allow for a nil value.

There is a stark difference between the implementation with this segment of the code. In my opinion the imperative implementation despite the use of pointers, is more understandable compared to the OO version. This is due to the access permissions, encapsulation in the object oriented style. Though I had some experience with this, it was still slightly frustrating and tedious. It was very calming to know that I didn’t require them when implementing the imperative program.

C#’s List collection data structure allows the user to be able to ‘dequeue’ by returning the item at index 0 and keeping the rest of the list except the item returned with the use of ‘RemoveAt’. Go’s slice data type allows a “slice” operator. This made the ‘dequeue’ version of the imperative implementation easy to create, as I managed to get the first to-do item the user had added with the syntax myList[0], while also being able to reassign myList without the first to-do item with the syntax myList[1:]. The way in which I could identify the first to-do item in both OO and Imperative were similar.

In my opinion I think that Object Oriented Programming looks neater, while Imperative Programming can be quicker, shorter and less tedious to write. I found myself not being able to work efficiently with having all the classes in one file, and therefore had to separate them. I personally enjoyed implementing the imperative program as I felt it was simpler to create. Though I do see the benefits of both Imperative and Object-Oriented. Imperative would be beneficial to programmers who are creating smaller programs, whereas for larger programs, there is great benefit to the Object-Oriented approach. The encapsulation is extremely beneficial for larger programs as it can maintain and protect the data from interferences.

To conclude, Object-Oriented Programming aims to be a beginner programming paradigm, like for example, Java which is an Object-Oriented language similar to C#, was designed to be *“relatively beginner-friendly in that it assumes the programmer is not that smart or careful, so programmers will be less likely to shoot themselves in the foot when coding Java apps.”* However after this analytical comparison it definitely is not in contrast to Imperative.

References:

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/base>
<http://www.bestprogramminglanguagefor.me/why-learn-java>
<https://www.computerhope.com/jargon/i/imp-programming.htm>