Programming Assignment 3:

“Explain the characteristics of procedural, object-orientated and event-driven programming.”

**Introduction:**

A paradigm can be defined as a set of principles, blueprint or a method to solve a problem or to do a task. A paradigm in programming can therefore be understood to be a model for programming so we can refer to it as an approach to solve problems using a particular programming language.

It includes defined guidelines on how programs & programming languages should be written, what patterns they should observe, and how these languages work. Each paradigm will consist of certain structures and features on how to tackle problem and different paradigms will be better suited for certain problems.

Programming languages will be based on a defined paradigm, and the user will follow a set of rules whilst utilising a particular language. In basic terms, a programming paradigm is a specific style of programming such as procedural, object-orientated or event driven.

**Procedural:**

Procedural programming derives from imperative programming, with the addition of functions (procedures). This type of language (as the name suggests) follows a procedure; meaning a set of commands that must be followed, working on a step-by-step basis, to ensure a smooth execution of the program.

Fundamentally, the procedural language requires direct instructs on how to complete a task in logical steps i.e., detailing a specific list of instructions informing the computer what to do and how to do it. This means that procedural programming follows a “top-to-bottom” approach as the code is written first and then executed with conditions. This programming style executes in a linear fashion with control flowing from one statement to the next and the sequence determining the order in which tasks are performed.

It focuses on separating code into reusable blocks of code called procedures (functions, methods etc).

Generally, a procedure will contain certain parameters that will allow the developer to pass data into the functions when they are called. The function can then use those to modify data and return or output the result.

There are formal parameters, which are the variables that are defined by the function and arguments which are the values or variables passed to the function when it is called.

In my Gas Account system parameters are used in constructors and methods to pass the necessary data for creating and managing the different accounts. The image below shows evidence of this with “pNewUnitCost” as the parameter in this case.

A black background with white text

Description automatically generated

When working with procedural programming developers will make use of variables and loops to help build a program to be able to carry out specific tasks involving calculations and displaying a desired output.

Variables are used to store data that can be manipulated and accessed throughout the program. Local variables are declared and only accessible inside a function and will be limited to the execution of the function. In contrast, global variables are the opposite, as they are declared outside of all the functions and are accessible from any function and doesn’t have a limited lifespan.

An example of this in my Gas Account system could be the static field “private static double dblUnitCost = 0.02” as this variable is shared by all instances in the “GasAccount” class.

A screen shot of a computer program

Description automatically generated

Return values can be defined as the output of a function that is passed back to the part of the program that called the function. They are an important characteristic of procedural programming, which allows functions to process data and provide results that can be used elsewhere in the program.

Functions can return data types such as “int” “double” “char”, Boolean values or more complex structures such as objects or arrays.

This is beneficial as it improves the reusability as functions with return values can be reused in other parts of your program, readability as it shows the flow of data and results, and testability of the code as the output can be verified against expected results.

Bubble sort:

A further example of procedural programming implemented in my GasAccount system is evident in the sorting algorithm “Bubble sort”. As we have discussed, procedural programming focuses on a linear approach to accomplish a task, in a step-by-step approach and the bubble sort algorithm works using clear steps of conditional statements, array manipulation and multiple loops.

The bubble sort algorithm iterates through the list multiple times, comparing adjacent elements and swapping them if they are in the wrong order until the entire list is sorted.

The image below shows evidence of this algorithm that will be used to sort Gas Accounts “By Balance”.

A computer screen shot of a code

Description automatically generated

Object-orientated:

Object-oriented programming or OOP concentrates on organising software design around data structures or objects as opposed to functions and logic. (Tech Target, 2024)

The structure of object-oriented programming includes classes, objects, methods and attributes. Classes are templates for creating objects, attribute and methods. Objects are instances of classes and are the building blocks of OOP and are made up of attributes and methods.

A method is a block of code containing a series of statements. A program will execute these statements by calling the method and passing any necessary arguments. In C#, every instruction executed is done so within the context of a method (Microsoft, 2024)

Attributes also known as properties or fields, represent the state of the object and will define its characteristics, this could be things like name, address etc and can be public, private, protected or internal which means controlling access from outside the class. Public attributes can be accessed from any part of the program and private can only be accessed from within the class. Protected are accessible within the class and its subclasses and internal means accessible within the same assembly.

According to GeeksforGeeks (2020) the central goal of OOP is “to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.”

This process is called “encapsulation” and is a major principle in object-oriented programming and means that attributes and methods are held together in a class. This process will restrict access to some of the components of the object and will permit access to only specific methods. As a result, it ensures that the data can only be accessed in a controlled manner, and this is evident in the Gas Account system I created:

As the image below shows I have several private fields that can’t be accessed directly from outside of the “GasAccount” class.

A screen shot of a computer program

Description automatically generated

I created methods to initialise these private fields to create “GasAccount” objects. The image below shows evidence of this.

A computer screen shot of a code

Description automatically generated

Getters:

Are important components of OOP that provide controlled access to the private fields. These methods allow other parts of the program to access the data without modifying it directly.

Within the GasAccount program I have created methods like “getAccRefNo”, “getName”, “getAddress”, “getBalance”, and “getUnits” to return the values of the object without exposing them to the external code. The image below shows this:

A screen shot of a computer code

Description automatically generated

Additionally, “updateUnitCost(double pNewUnitCost)” allows controlled updates to the static “dblUnitCost” field.

“getUnitCost()” provides read-only access to the dblUnitCost field.

A screen shot of a computer program

Description automatically generated

There are various benefits of the encapsulation process starting with the fact that it restricts access to some of the components of the object and will provide controlled access to specific public methods making the class easier to maintain. As a result, this means that the internal state can only be changed via those specific methods which helps to protect data integrity. It acts as security protecting the data from outside interference or bugs and it can be used in different parts of the program as it remains consistent.

Setters are also important in OOP and use an implicit parameter called “value”, whose type is the type of the property and when a value is assigned to the property, the set accessor is invoked by using an argument that provides the new value.

In object-orientated programming, parameters and return types play an important role in defining methods. A parameter is a variable including include strings, integers etc representing data that the method expects to receive when it’s called. A parameter will enables the passing of information into functions and methods, enabling them to perform specific tasks.

In OOP “return types” refer to the data type that a method is expected to return (output data) after it has been executed. Every method in C# must declare a return type, even if it is void, demonstrating that the method does not return any value.

For example, in my GasAccount system, the “recordUnits” method accepts a parameter “pUnitsUsed”, representing the gas units read from a meter reading. This parameter is used to calculate the current cost and update the gas account's balance.

A screenshot of a computer program

Description automatically generated

**Event-driven:**

Event-driven programming is a programming paradigm where the flow of the program is determined by external events that occur asynchronously during the program's execution.

“Asynchronously” refers to a mode of operation where events are executed independently of the main program flow or timeline. (Tech Target, 2024)

This means that the program does not wait for a task to complete before moving on to the next task. Alternatively, it will continue to execute other tasks while the asynchronous task continues in the background. This is an important characteristic of event driven programming. Advantages of this programming style include the fact that it permits elements to respond quickly to events resulting in a more responsive system.

These events can be triggered by user actions, such as mouse clicks or keyboard inputs and in this programming style, the program will usually wait, (and listen) for events and reacts to them as they occur by executing predefined event handlers.

Further important characteristics of event-driven programming include events, event listeners and event handlers. An event could be someone making an online payment (GAS ACCOUNT) the event listener will listen for events and specify which function to call as a result and the event handler refers to the callback function that gets executed in response to the event. By registering these callbacks with event listeners, the program will respond dynamically to the actions of the user.

Event loops are important in this programming style which are continuously listening and will transmit events to the associated event handlers ensuring that events are processed in the order they occur resulting in a responsive program.

The fact that they can handle multiple events at once can help with a more efficient use of resources.

Event driven programming also offers good scalability as elements operate independently and can therefore be added or removed without having an adverse effect on everything else in the system.

Disadvantages include the fact that it can be quite challenging to implement particularly in large, complex systems. It has difficulties with error handling as each component must be able to do this and be able to report that data back into the system.

In my Gas Account system there is evidence of event driven programming, for example, an event would be clicking on the button to add a payment. “btnPayment\_Click” is a method that handles the click event of a button “btnPayment”. This method is automatically executed when the button is clicked by the user.

The image below shows this:

A computer screen shot of a program

Description automatically generated

To help me build this Gas Account program I used Visual Studio 2022 as my Integrated Development Environment (IDE).

IDEs are pieces of software that encompass a text editor and additional tools such as code libraries which are a collection of prewritten code that a user can use to enhance their tasks, and compliers which will turn source code into a readable language that can be executed.

IDEs also include debuggers meaning that the user can inspect and amend the program in real time and test platforms. Testing is a critical component as it helps to make sure the calibre of the final product meets the set requirements. An IDE will bring all these features together as one single application making it easier for a developer to complete a project.

I used Visual Studio as it numerous useful features such as its feature called “Intellicode” which is an A.I powered code completion tool. The benefits of this are that it allows the user to complete code based on the context, for example, variable names, functions etc. This feature will complete the whole line for the user. A simple click of the tab button will carry this out. This means a reduction in user keystrokes thus saving time and can assist with consistency.

Other features include underlining your source code with a red squiggly line and a lightbulb to the left of your code (on the relevant line) which will prompt you to double check your work and offer potential solutions. Examples of syntax errors are things like you may have forgotten a “;” or have too many (or not enough) “{}” and if you try and run the code you will have a window at the bottom showing your error list which is quite helpful.

The evidence shows that my Gas Account system was created with elements of all three programming paradigms.

As previously discussed, within procedural programming the focus is on functions and promotes the breaking down of problems into smaller procedures. This programming style executes the code in a linear fashion with clearly defined sequential steps. Furthermore, this programming paradigm relies on global variables or other data that can be accessed and modified by any function within the program.

Within my Gas Account program, the procedural elements can be identified in the methods like “Record Units”, “Deposit” etc as they perform specific operations and is further reflected in the use of Global Variables like “dblUnitCost” and the static method of “UpdateUnitCost”. The use of the bubble sort algorithm is a further example of procedural programming as it also follows a linear, step-by-step approach to accomplish a task.

As we can see from the evidence, the procedural paradigm can be beneficial, particularly in how simple and efficient it can be. However, it also has its limitations, and this will be evident as a program expands becoming more complex and facing more challenges. This programming style has limited reusability in comparison to other styles such as OOP and its lack of encapsulation means that changes in one part of your program will affect other aspects. This style offers less security to your data meaning the integrity of your data could be affected by user interference (even accidently) and bugs.

In object-orientated programming, we know that the attention is on objects and encapsulation. In my Gas Account system OOP is reflected in the “Gas Account” class which encapsulates the data from the accounts. I had to create methods that will allow access to these controlled fields, examples of these “Getters” include “getAccRefNo” and “getName” and additionally, “updateUnitCost(double pNewUnitCost)” allows controlled updates to the static “dblUnitCost” field.

Clearly encapsulation and code modularity are advantages you get when using this programming style but there are also limitations such as the increase in size and complexity that this style brings making it more challenging to read and consuming more memory and CPU resources. Furthermore, it is limited in debugging in comparison with procedural programming. Trying to learn the key concepts of OOPS can be challenging, especially for a beginner so I would stay it has a steep learning curve.

As we have discussed, event driven paradigms focus on events which can be triggered by user actions, such as mouse clicks and in this programming style, the program will usually wait, (and listen) and react to them as they occur by executing predefined event handlers.

As my Gas Account system utilises a Graphical User Interface (GUI), event driven programming will be implemented here to handle user interactions. This is evident when a user clicks on a button to make a payment for example which would trigger an event handler method to execute the payment operation on the selected Gas Account. “btnPayment\_Click” is a method that handles the click event of a button “btnPayment”. This method is automatically executed when the button is clicked by the user.

Event-driven programming is very useful for building responsive user interfaces like my Gas Account program, it also has its limitations such as handling asynchronous events which can be challenging for less experienced developers which can also lead to difficulties in error handling in comparison to other programming styles. It is also limited in the fact that event duplication can occur in this style of programming which can lead to inconsistencies and other challenges and additionally, by continuously listening for events can have adverse effects. This process will mean consuming system resources which could impact the overall system performance.

Whilst each style has their own strengths and weaknesses, I found that by combining all three programming paradigms, it helped me successfully build the Gas Account program that met the user requirements and worked to a good standard.

**References:**

Chhitarka, D. (2022, March 27). Introduction to Programming - What is Programming Paradigm? DEV Community. Available at: <https://dev.to/dchhitarka/introduction-to-programming-what-is-programming-paradigm-3le1> (Accessed May 22,2024).

CodeLikeAGirl. (2024, February 15). Exploring Procedural Programming: A fundamental paradigm. Code with C. Available at: <https://www.codewithc.com/exploring-procedural-programming-a-fundamental-paradigm/> (Accessed May 22, 2024).

Bhatia, S. *Procedural Programming [definition]*. Hackr.Io. Available from: <https://hackr.io/blog/procedural-programming>. (Accessed May 22, 2024).

Gillis, A. S. and Lewis, S. (2021) What is Object-Oriented Programming (OOP)?, App Architecture. TechTarget. Available at: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP (Accessed: 28 May 2024).

Introduction of object-oriented programming, GeeksforGeeks. (2020). Available at: https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/ (Accessed: 28 May 2024).

Tutorchase.com. Available at: https://www.tutorchase.com/notes/ib/computer-science/d-2-3-challenges-and-limitations-of-object-oriented-programming-oop (Accessed: 28 May 2024).

Bill, Wagner. Using properties - C#, Microsoft.com. Available at: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties (Accessed: 29 May 2024).

Praveen, R. G. (2023) Parameters in C# - Praveen Rao G, Medium. Available at: https://medium.com/@praveen.rao.g.1990/parameters-in-c-16d80b61e81a (Accessed: 29 May 2024).

Geeksforgeeks.org. Available at: https://www.geeksforgeeks.org/what-is-the-event-driven-programming-paradigm/ (Accessed: 30 May 2024).

Techtarget.com. Available at: https://www.techtarget.com/searchnetworking/definition/asynchronous (Accessed: 30 May 2024).

Medium.com. Available at: https://itcrats.medium.com/what-is-an-event-driven-architecture-benefits-and-drawbacks-53145d377505 (Accessed: 30 May 2024).

Quix.io. Available at: https://quix.io/blog/what-why-how-of-event-driven-programming (Accessed: 30 May 2024).