NOVEMBER 12, 2019 / #FUNCTIONAL PROGRAMMING

What exactly is a programming paradigm?

By Thanoshan MV

> *Any fool can write code that a computer can understand. Good programmers write code that humans can understand.*
> *— Martin Fowler*

When programming, complexity is always the enemy. Programs with great complexity, with many moving parts and interdependent components, seem initially impressive. However, the ability to translate a real-world problem into a simple or elegant solution requires a deeper understanding.

While developing an application or solving a simple problem, we often say “If I had more time, I would have written a simpler program”. The reason is, we made a program with greater complexity. The less complexity we have, the easier it is to debug and understand. The more complex a program becomes, the harder it is to work on it.

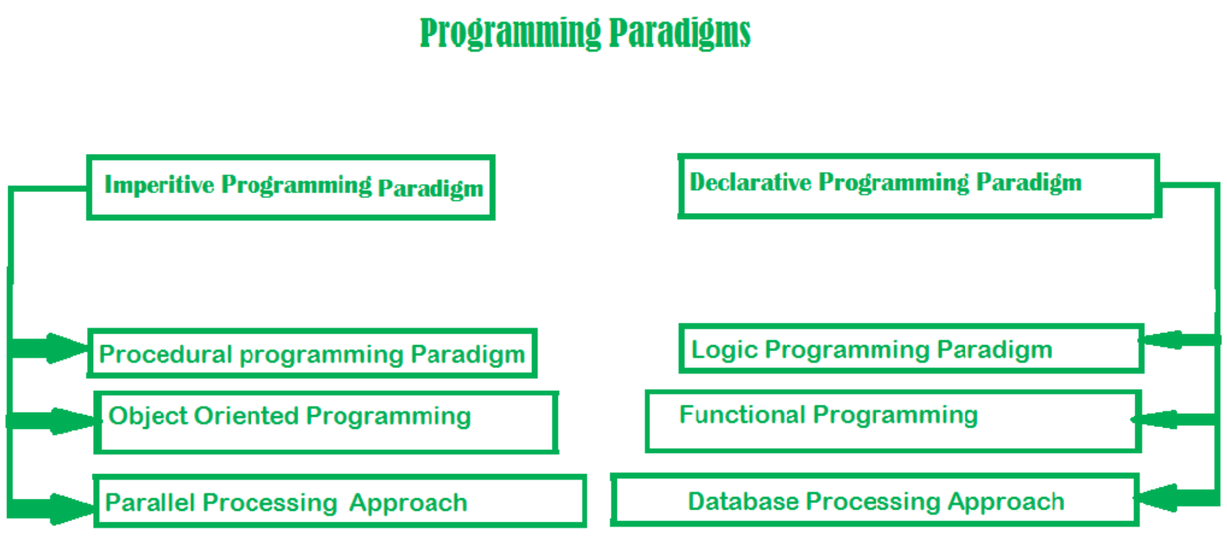
Managing complexity is a programmer’s main concern. So how do programmers deal with complexity? There are many general approaches that reduce complexity in a program or make it more manageable. One of

Introduction to programming paradigms

The term programming paradigm refers to **a style of programming.** It does not refer to a specific language, but rather it refers to the way you program.

There are lots of programming languages that are well-known but all of them need to follow some strategy when they are implemented. And that strategy is a paradigm.

The types of programming paradigms



Describes the different styles of programming (Source: geeksforgeeks.org)

Imperative programming paradigm

The word “imperative” comes from the Latin “impero” meaning “I command”.

It’s the same word we get “emperor” from, and that’s quite apt. You’re the emperor. You give the computer little orders to do and it does them one at a time and reports back.

The paradigm consists of several statements, and after the execution of all of them, the result is stored. It’s about writing a list of instructions to tell the computer what to do step by step.

In an imperative programming paradigm, the order of the steps is crucial, because a given step will have different consequences depending on the current values of variables when the step is executed.

To illustrate, let's find the sum of first ten natural numbers in the imperative paradigm approach.

Example in C:

```
int main()
{
    int sum = 0;
    sum += 1;
    sum += 2;
    sum += 3;
    sum += 4;
    sum += 5;
    sum += 6;
    sum += 7;
    sum += 8;
    sum += 9;
    sum += 10;

    printf("The sum is: %d\n", sum); //prints-> The sum is 55

    return 0;
}
```

In the above example, we are commanding the computer what to do line by line. Finally, we are storing the value and printing it.

1.1 Procedural programming paradigm

Procedural programming (which is also imperative) **allows splitting those instructions into procedures**.

NOTE: Procedures aren't functions. The difference between them is that functions return a value, and procedures do not. More specifically, functions are designed to have minimal side effects, and always produce the same output when given the same input. Procedures, on the other hand, do not have any return value. Their primary purpose is to accomplish a given task and cause a desired side effect.

A great example of procedures would be the well known for loop. The for loop's main purpose is to cause side effects and it does not return a value.

To illustrate, let's find the sum of first ten natural numbers in the procedural paradigm approach.

Example in C:

```
#include <stdio.h>

int main()
{
    int sum = 0;
    int i =0;
    for(i=1;i<11;i++){
        sum += i;
    }

    printf("The sum is: %d\n", sum); //prints-> The sum is 55
```

In the example above, we've used a simple for loop to find the summation of the first ten natural numbers.

Languages that support the procedural programming paradigm are:

- C
- C++
- Java
- ColdFusion
- Pascal

Procedural programming is often the best choice when:

- There is a complex operation which includes dependencies between operations, and when there is a need for clear visibility of the different application states ('SQL loading', 'SQL loaded', 'Network online', 'No audio hardware', etc). This is usually appropriate for application startup and shutdown (Holligan, 2016).
- The program is very unique and few elements were shared (Holligan, 2016).
- The program is static and not expected to change much over time (Holligan, 2016).
- None or only a few features are expected to be added to the project over time (Holligan, 2016).

Why should you consider learning the procedural programming paradigm?

- It's simple.
- An easier way to keep track of program flow.
- It has the ability to be strongly modular or structured.
- Needs less memory: It's efficient and effective.

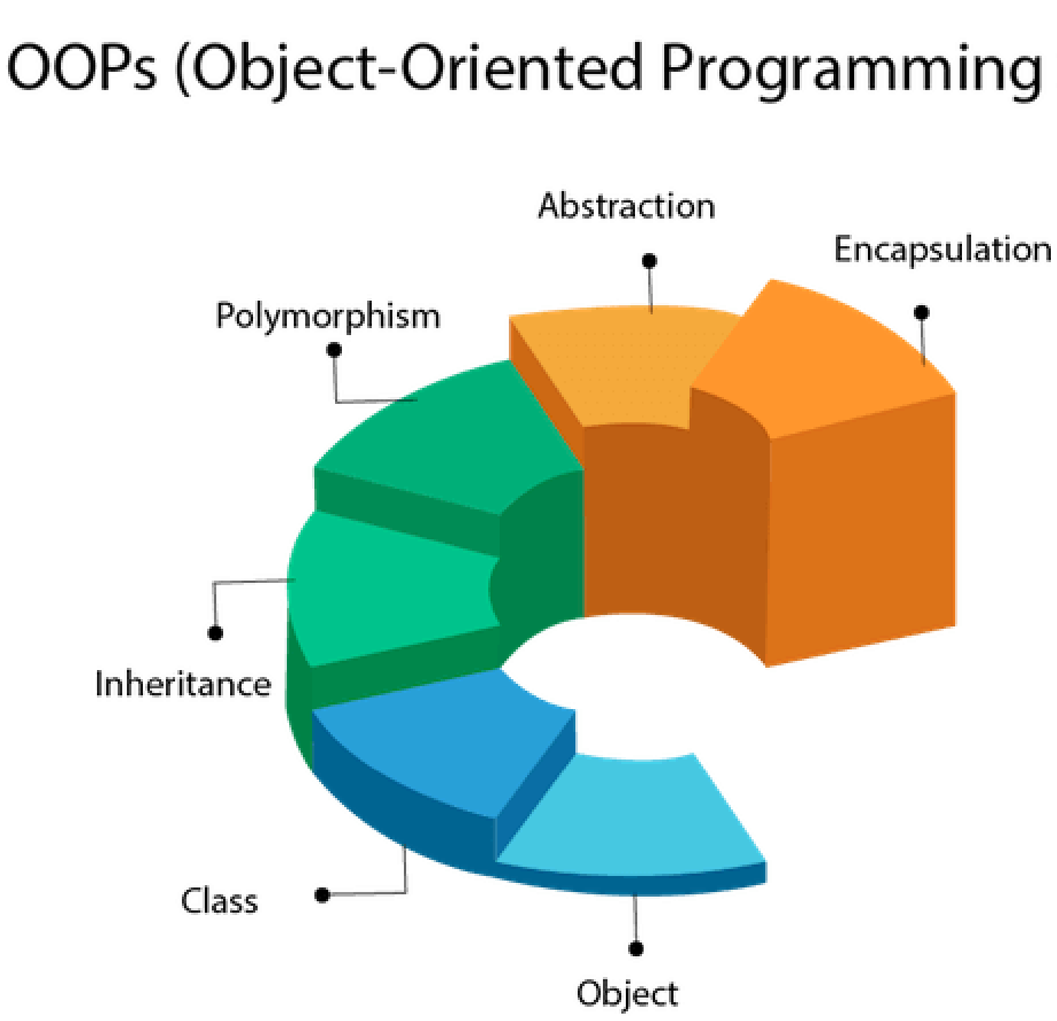
1.2 Object-oriented programming paradigm

OOP is the most popular programming paradigm because of its unique advantages like the modularity of the code and the ability to directly associate real-world business problems in terms of code.

> *spaghetti code. It lets you accrete programs as a series of patches.*
>
> *— Paul Graham*

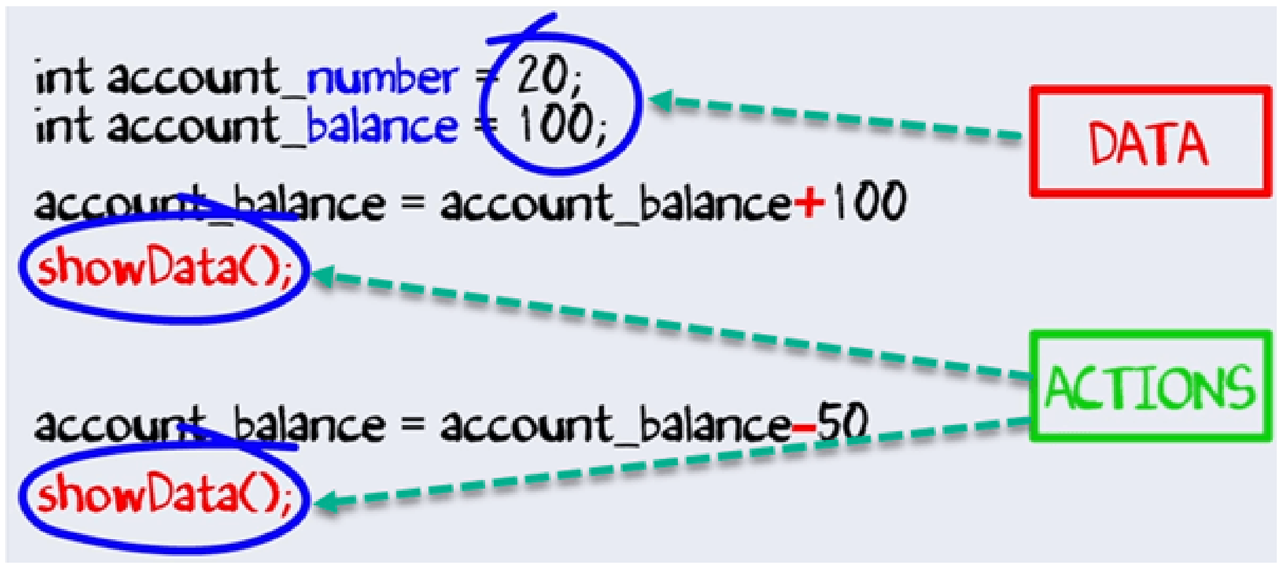
The key characteristics of object-oriented programming include Class, Abstraction, Encapsulation, Inheritance and Polymorphism.

A **class** is a template or blueprint from which objects are created.



Describes the key concepts of object-oriented programming (Source: javatpoint.com)

Objects are instances of classes. Objects have attributes/states and methods/behaviors. Attributes are data associated with the object while methods are actions/functions that the object can perform.



Explains the states and behaviors of an object (Source: guru99.es)

Abstraction separates the interface from implementation. Encapsulation is the process of hiding the internal implementation of an object.

same message and respond in different ways.

To illustrate, let's find the sum of first ten natural numbers in the object-oriented paradigm approach.

Example in Java:

```
public class Main
{
    public static void main(String[] args) {
        Addition obj = new Addition();
        obj.num = 10;
        int answer = obj.addValues();
        System.out.println("The sum is = "+answer); //prints-> The sum is 55
    }
}

class Addition {
    int sum =0;
    int num =0;
    int addValues(){
        for(int i=1; i<=num;i++){
            sum += i;
        }
        return sum;
    }
}
```

We have a class `Addition` that has two states, `sum` and `num` which are initialized to zero. We also have a method `addValues()` which returns the sum of `num` numbers.

In the `Main` class, we've created an object, `obj` of Addition class. Then, we've initialized the `num` to 10 and we've called `addValues()` method to get the sum.

Languages that support the object-oriented paradigm:

- Python
- Ruby
- Java
- C++
- Smalltalk

Object-oriented programming is best used when:

- You have multiple programmers who don’t need to understand each component (Holligan, 2016).

- The project is anticipated to change often and be added to over time (Holligan, 2016).

Why should you consider learning the object-oriented programming paradigm?

- Reuse of code through Inheritance.
- Flexibility through Polymorphism.
- High security with the use of data hiding (Encapsulation) and Abstraction mechanisms.
- Improved software development productivity: An object-oriented programmer can stitch new software objects to make completely new programs (The Saylor Foundation, n.d.).
- Faster development: Reuse enables faster development (The Saylor Foundation, n.d.).
- Lower cost of development: The reuse of software also lowers the cost of development. Typically, more effort is put into the object-oriented analysis and design (OOAD), which lowers the overall cost of development (The Saylor Foundation, n.d.).
- Higher-quality software: Faster development of software and lower cost of development allows more time and resources to be used in the verification of the software. Object-oriented programming tends to result in higher-quality software (The Saylor Foundation, n.d.).

1.3 Parallel processing approach

Parallel processing is the processing of program instructions by dividing them among multiple processors.

A parallel processing system allows many processors to run a program in less time by dividing them up.

Languages that support the Parallel processing approach:

- NESL (one of the oldest ones)
- C
- C++

Parallel processing approach is often the best use when:

- You need to solve some computational problems that take hours/days to solve even with the benefit of a more powerful microprocessor.
- You work with real-world data that needs more dynamic simulation and modeling.

Why should you consider learning the parallel processing approach?

- Speeds up performance.
- Often used in Artificial Intelligence. Learn more here: Artificial Intelligence and Parallel Processing by Seyed H. Roosta.
- It makes it easy to solve problems since this approach seems to be like a divide and conquer method.

Here are some useful resources to learn more about parallel processing:

1. Parallel Programming in C by Paul Gribble
2. Introduction to Parallel Programming with MPI and OpenMP by Charles Augustine
3. INTRODUCTION TO PARALLEL PROGRAMMING WITH MPI AND OPENMP by Benedikt Steinbusch

2. Declarative programming paradigm

Declarative programming is a style of building programs that expresses the logic of a computation without talking about its control flow.

Declarative programming is a programming paradigm in which the programmer defines what needs to be accomplished by the program without defining how it needs to be implemented. In other words, the approach focuses on what needs to be achieved instead of instructing how to achieve it.

Imagine the president during the state of the union declaring their intentions for what they want to happen. On the other hand, imperative programming would be like a manager of a McDonald's franchise. They are very imperative and as a result, this makes everything important. They, therefore, tell everyone how to do everything down to the simplest of actions.

2.1 Logic programming paradigm

The logic programming paradigm takes a declarative approach to problem-solving. It's based on formal logic.

The logic programming paradigm isn't made up of instructions - rather it's made up of facts and clauses. It uses everything it knows and tries to come up with the world where all of those facts and clauses are true.

For instance, Socrates is a man, all men are mortal, and therefore Socrates is mortal.

The following is a simple Prolog program which explains the above instance:

```
man(Socrates).
mortal(X) :- man(X).
```

The first line can be read, "Socrates is a man.'' It is a base clause, which represents a simple fact.

The second line can be read, "X is mortal if X is a man;'' in other words, "All men are mortal.'' This is a clause, or rule, for determining when its input X is "mortal.'' (The symbol ":-'', sometimes called a turnstile, is pronounced "if''.) We can test the program by asking the question:

```
?- mortal(Socrates).
```

that is, "Is Socrates mortal?'' (The " `?-` '' is the computer's prompt for a question). Prolog will respond " `yes` ''. Another question we may ask is:

```
?- mortal(X).
```

That is, "Who (X) is mortal?'' Prolog will respond " `X = Socrates` ''.

To give you an idea, John is Bill's and Lisa's father. Mary is Bill's and Lisa's mother. Now, if someone asks a question like "who is the father of Bill and Lisa?" or "who is the mother of Bill and Lisa?" we can teach the computer to answer these questions using logic programming.

```
/*We're defining family tree facts*/
father(John, Bill).
father(John, Lisa).
mother(Mary, Bill).
mother(Mary, Lisa).

/*We'll ask questions to Prolog*/
?- mother(X, Bill).
X = Mary
```

Example explained:

```
father(John, Bill).
```

The above code defines that John is Bill's father.

We're asking Prolog what value of X makes this statement true? X should be Mary to make the statement true. It'll respond `X = Mary`

```
?- mother(X, Bill).
X = Mary
```

Languages that support the logic programming paradigm:

- Prolog
- Absys
- ALF (algebraic logic functional programming language)
- Alice
- Ciao

Logic programming paradigm is often the best use when:

- If you're planning to work on projects like theorem proving, expert systems, term rewriting, type systems and automated planning.

Why should you consider learning the logic programming paradigm?

- Easy to implement the code.
- Debugging is easy.

- As it's based on thinking, expression and implementation, it can be applied in non-computational programs too.
- It supports special forms of knowledge such as meta-level or higher-order knowledge as it can be altered.

2.2 Functional programming paradigm

The functional programming paradigm has been in the limelight for a while now because of JavaScript, a functional programming language that has gained more popularity recently.

The functional programming paradigm has its roots in mathematics and it is language independent. The key principle of this paradigm is the execution of a series of mathematical functions.

You compose your program of short functions. All code is within a function. All variables are scoped to the function.

In the functional programming paradigm, the functions do not modify any values outside the scope of that function and the functions themselves are not affected by any values outside their scope.

To illustrate, let's identify whether the given number is prime or not in the functional programming paradigm.

Example in JavaScript:

```
function isPrime(number){
 for(let i=2; i<=Math.floor(Math.sqrt(number)); i++){
  if(number % i == 0 ){
   return false;
  }
 }
 return true;
}
isPrime(15); //returns false
```

In the above example, we've used `Math.floor()` and `Math.sqrt()` mathematical functions to solve our problem efficiently. We can solve this problem without using built-in JavaScript mathematical functions, but to run the code efficiently it is recommended to use built-in JS functions.

`number` is scoped to the function `isPrime()` and it will not be affected by any values outside its scope. `isPrime()` function always produces the same output when given the same input.

iteration (Bhadwal, 2019).

Languages that support functional programming paradigm:

- Haskell
- OCaml
- Scala
- Clojure
- Racket
- JavaScript

Functional programming paradigm is often best used when:

- Working with mathematical computations.
- Working with applications aimed at concurrency or parallelism.

Why should you consider learning the functional programming paradigm?

- Functions can be coded quickly and easily.
- General-purpose functions can be reusable which leads to rapid software development.
- Unit testing is easier.
- Debugging is easier.
- Overall application is less complex since functions are pretty straightforward.

2.3 Database processing approach

This programming methodology is based on data and its movement. Program statements are defined by data rather than hard-coding a series of steps.

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS) ("What is a Database", Oracle, 2019).

To process the data and querying them, databases use **tables**. Data can then be easily accessed, managed, modified, updated, controlled and organized.

organization. This is because the database stores all the pertinent details about the company such as employee records, transaction records and salary details.

Most databases use Structured Query Language (SQL) for writing and querying data.

Here's an example in database processing approach (SQL):

```
CREATE DATABASE personalDetails;
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

The `PersonID` column is of type int and will hold an integer. The `LastName`, `FirstName`, `Address`, and `City` columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

The empty `Persons` table will now look like this:

PersonID	LastName	FirstName	Address	City

Describes how the Persons table will look after the execution

Database processing approach is often best used when:

- Working with databases to structure them.
- Accessing, modifying, updating data on the database.
- Communicating with servers.

Why are databases important and why should you consider learning database processing approach?

- Massive amount of data is handled by the database: Unlike spreadsheet or other tools, databases are used to store large amount of data daily.
- Accurate: With the help of built-in functionalities in a database, we can easily validate.

- Data integrity: With the help of built-in validity checks, we can ensure the consistency of data.

Conclusion

Programming paradigms reduce the complexity of programs. Every programmer must follow a paradigm approach when implementing their code. Each one has its advantages and disadvantages.

If you're a beginner, I would like to suggest learning object-oriented programming and functional programming first. Understand their concepts and try to apply them in your projects.

For example, if you're learning object-oriented programming, the pillars of object-oriented programming are Encapsulation, Abstraction, Inheritance and Polymorphism. Learn them by doing it. It will help you to understand their concepts on a deeper level, and your code will be less complex and more efficient and effective.

I strongly encourage you to read more related articles on programming paradigms. I hope this article helped you.

Please feel free to let me know if you have any questions.

You can contact and connect with me on Twitter @ThanoshanMV.

Thank you for reading.

Happy Coding!

References

- Akhil Bhadwal. (2019). Functional Programming: Concepts, Advantages, Disadvantages, and Applications
- Alena Holligan. (2016). When to use OOP over procedural coding
- The Saylor Foundation. (n.d.). Advantages and Disadvantages of Object-Oriented Programming (OOP)
- What is a Database | Oracle. (2019).