Universidad Autónoma de Ciudad Juárez

División Multidisciplinaria Ciudad Universitaria

Cuadro comparativo de lenguajes de Programación Orientada a Objetos

Programación II

Docente: Alan Ponce

Alumno : Roberto Mares Ceballos 159971

Licenciatura en Ingeniería de Software

08 de Marzo de 2019

Introducción

El propósito de este cuadro comparativo es poder concretar las ideas, bondades, y enfoques de los paradigmas de programación, en especial el orientado a objetos, puede parecer innecesaria la forma en la que se debe programar para crear sistemas utilizando este paradigma, más de estudiantes que vienen de un curso de programación estructurada, la complejidad de tal filosofía pudiera parecer superficial. Al examinar los diversos lenguajes de cada paradigma, sus aplicaciones, historia y bondades; el estudiante será capaz de obtener perspectiva de los beneficios y aplicaciones de cada uno.

Definición de lenguaje procedural.

El paradigma procedural, también conocido como imperativo, es un lenguaje orientado a las sentencias. Sentencias que trabajan como uno solo para poder obtener resultados mediante cálculos. Su concepción es una forma de liberar a los programadores de codificar en ensamblador.

Dentro de las bondades de este tipo de lenguajes se tienen variables, las cuales se almacenan en la memoria del computador, y cuyo valor dentro del sistema va a cambiar según se vaya realizando cálculos sobre ella.

Como mencionamos a las variables se les otorga un valor dentro del sistema, a esta acción o proceso se le conoce como ligadura, al espacio de memoria o la celda de la memoria en la que se tiene ubicada la variable se le otorga un valor utilizando la sentencia de asignación.

Repetición, debido a que la mayoría de los programas o sistemas creados en este tipo de lenguajes tienden a repetir varias veces las mismas instrucciones, se necesita esta característica para poder transformar la información en lo que necesitamos ejecutando cuantas veces sea necesaria para obtener la salida requerida. (Reyna, 2012)

Lenguajes de Programación procedural y código de ejemplo:

- FORTRAN.

  Lenguaje de programación utilizado para matemáticas, especialmente en aplicaciones de cálculo científico. (Asensio, 2008)

  Ejemplo de código:

```
PROGRAM FACTORIAL
IMPLICIT NONE
REAL::J,N
INTEGER::P
WRITE(*,*)"NUMERO AL CUAL DESEA CALCULAR SU FACTORIAL"
READ(*,*)N
P=1
DO J=0,N-1,1
P=P*(N-J)
END DO
WRITE(*,*)"EL FACTORIAL ES:",P
END PROGRA
```

- COBOL.

  Common Business Oriented Language, diseñado para el desarrollo de negocios, como archivos y aplicaciones; no para sistemas. (Gomez, 2016)

  Ejemplo de Código:

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.    HOLAMUNDO.
000300
000400*
000500 ENVIRONMENT DIVISION.
000600 CONFIGURATION SECTION.
000700 SOURCE-COMPUTER. RM-COBOL.
000800 OBJECT-COMPUTER. RM-COBOL.
000900
001000 DATA DIVISION.
001100 FILE SECTION.
001200
100000 PROCEDURE DIVISION.
100100
100200 MAIN-LOGIC SECTION.
100300 BEGIN.
100400     DISPLAY " " LINE 1 POSITION 1 ERASE EOS.
100500     DISPLAY "Hola mundo" LINE 15 POSITION 10.
100600     STOP RUN.
100700 MAIN-LOGIC-EXIT.
100800     EXIT.
```

- Pascal.

  Su objetivo es ser un lenguaje creado para facilitar el aprendizaje de la programación. Sin embargo, a pesar de su propósito, terminó siendo una herramienta para la creación de herramientas de todo tipo. (EcuRed, 2011)

  Ejemplo de código:

```
Program ejA;
uses crt;
Var
R,Num1,num2:integer;


Begin
clrscr;
Writeln('Ingrese Un numero');
Readln(Num1);
Writeln('Ingrese Un numero');
Readln(Num2);
R:=Num1+Num2;
Writeln('El resultado es: ',R);
readln;
ENd.
```

- Ada.

  Lenguaje complejo que puede tener la simpleza de Pascal hasta la flexibilidad de C++. Diseñado con la seguridad en mente, y con la finalidad de tratar de reducir errores comunes y difíciles de descubrir. (EcuRed, Lenguaje de Programacion Ada, 2011)

  Ejemplo de código:

```
Éste es el típico "Hola Mundo" escrito en Ada:
with Ada.Text_IO;

procedure Hola_Mundo is
begin
   Ada.Text_IO.Put("¡Hola, Mundo!");
end Hola_Mundo;
```

Definición de lenguaje orientado a objetos.

Los lenguajes orientados a objetos están definidos por los siguientes conceptos: encapsulación, herencia, y polimorfismo. También por el uso de objetos y sus interacciones, para la creación de aplicaciones y sistemas informáticos.

Para iniciar, hay que definir lo que es un objeto y una clase, se tiene como concepto que el objeto es la instanciación de una clase, cada objeto tiene un estado el cual está conformado por atributos o datos miembro, comportamiento o métodos lo cuales es capaz de realizar o ejecutar. También se cuentan con clases, las cuales se pueden considerar como el template de los objetos, la base de cómo el objeto será construido. Algo como el dilema del huevo y la gallina, para poder existir un objeto se necesita la clase y viceversa.

Dentro de las bondades de los lenguajes orientados a objetos y específicamente de las clases, podemos heredar de otras clases para crear una nueva que contenga algunos o todos los métodos de la clase padre, con esto nace en concepto de Superclases y Subclases, las subclases contienen todos los atributos y comportamientos que son comunes en las clases que heredarán de él, las subclases es la extensión de la superclase, es decir una especialización de la clase padre. Con este concepto también nace la abstracción, la cual puede ayudarnos a crear clases abstractas las cuales posteriormente serán especializadas por medio de la herencia.

En el caso de los métodos y los objetos, un objeto puede actuar de diferentes maneras dependiendo de la situación en la que este se desenvuelve, cuando un mensaje o solicitud es enviada a un objeto este tiene que responder a tal llamado de una forma específica.

La composición es una de las partes más complejas de la programación orientada a objetos, ya que un objeto puede ser capaz de contener objetos, y crear clases compuestas, creando algo como una extensión de un objeto.

En cuanto a los objetos, como ya se mencionó, todos y cada uno de los objetos que hayan sido creados, tienen atributos que están definidos por los datos que almacenará el objeto y guiará su comportamiento. Comportamiento, son las funciones o métodos que realizaran operaciones con los atributos enviados por medios de mensajes o llamadas.

Uno de los beneficios más importantes de los objetos es la encapsulación, que tiene como filosofía principal dejar el control del acceso al objeto, sus atributos y comportamiento solamente al creador, esto de logra por medio de modificadores de acceso, los cuales pueden ser públicos o privados, y aplicados a los datos o métodos. La interfaz es lo que el creador decide que será lo que el cliente pueda utilizar sin dañar el funcionamiento del objeto o la clase. La implementación es la parte interna del objeto, la cual dicta su comportamiento y es de vital importancia que no sea modificada. (Weisfield, 2013)

Ejemplos de lenguajes de Programación Orientada a Objetos:

- C++. Derivado de C, es el resultado de la adición de cualidades y características que carecía. En 1983 se creó el comité ANSI C que tenía como objetivo crear un lenguaje uniforme a partir de C, este fue desarrollado por medio de Kernighan y Ritchie en 1972, en la ATT. Pero no fue hasta 1980 que se empezó a desarrollar C++. Nombrado así por el operador de incremento. Debido a su gran difusión y éxito entre los programadores, se estandarizó en 1987, posteriormente en 1989 un comité ANSI los estandarizó para lanzarlo a nivel global.
  A lo largo del tiempo ha evolucionado, es cierto sigue manteniendo la riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia de C, eliminar las dificultades y limitaciones.Cambio de paradigma a OO, siendo uno de los lenguajes más complejos y ricos con los que se pueda trabajar en este filosofia. Hoy en día ocupa el primer lugar en los lenguajes de programación para la creación de aplicaciones y sistemas. (EcuRed, C++, 2011)

- Java. Creado como un proyecto de investigación dentro de Sun Microsystems, creando un lenguaje basado en C++, en 1991. Este proyecto se vio en peligro debido a la falta de mercado de dispositivos inteligentes, sin embargo gracias al crecimiento de Internet, se utilizó para crear contenido dinámico, interactividad, animaciones a las páginas web detonando su popularidad.
  Hoy en día Java es utilizado para desarrollar aplicaciones empresariales, servidores Web, aplicaciones para distintos dispositivos, y más ya que cuenta con una API muy rica. Fue comprado por Oracle en 2010. Como se mencionó la sintaxis de Java trata de ser la versión simplificada de C++. Es un lenguaje orientado a objetos, compilado virtualmente por medio de una máquina virtual, lo que permite que se ejecute en cualquier entorno, dígase UNIX, Mac, Windows.
  Otro de los puntos fuertes de Java es la inmensa cantidad de bibliotecas a disposición del programador, haciéndole capaz de crear cualquier tipo de aplicación, desde aplicaciones de red como sockets y hacer conexiones con servidores o clientes remotos, haciendo posible la creación de aplicaciones distribuidas. (Holzner, 2012)

- Python. Creado por Guido Van Rossum alrededor de finales de los 80 y principios de los 90. Inicialmente fue concebido para manejar excepciones e interfaces con Amoeba (sistema operativo en el cual se encontraba trabajando). En Octubre del 2000 se lanza Python 2.0, con nuevas características como recolección de basura y soporte a Unicode, todo esto gracias al trabajo de una comunidad de programadores bajo la dirección de Guido. El 3 de diciembre de 2008 llegó la versión 3.0 la cual es considerada

como el cambio más radical, ya que en esta version esta mas involucrada mas personas y hay mas trabajo invertido se diferencia mucho de la versión 2.0

Los principios de Python definidos por su comunidad son:

- Hermoso es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Plano es mejor que anidado.

  entre otras.

La sintaxis de Python es muy sencilla, tanto que puede llegar a parecer pseudocódigo. Otra de las fortalezas de Python es la librería estándar con la que cuenta, decenas de módulos que pueden cubrir cualquier necesidad de un programador, desde cadenas, estructuras de datos, funciones numéricas y matemáticas, compresión de datos, formatos de archivos, criptografía, servicios de sistemas operativos, comunicación entre procesos, manejo de datos de Internet, servicios multimedia, manejo excepciones. El ser un lenguaje interpretado no baja su rendimiento, ya que la librería estándar está en C hace a la funciones primitivas eficientes, además que se puede compilar a bytecodes. (Challenger-Pérez, Yanet , & Roberto Antonio, 2014)

- C#. Lenguaje con seguridad de tipos, orientado a objetos que permite a los programadores una gran variedad de aplicaciones seguras y sólida ejecutadas en .NET Framework .NET.

  En enero de 1999, se formó un equipo liderado por Anders Hejlsberg, que tenía como misión desarrollar un nuevo lenguaje de programación, C#, el cual utiliza la biblioteca de clases de la plataforma .NET. En 2001 se normalizo por ECMA. En 2005 salió la versión 2.0, la cual incluía tipos genéricos, métodos anónimos, iteradores, tipos parciales y anulables. En 2007, la versión 3.0 hizo una mejor en los tipos implícitos, tipos anónimos, y LINQ. (EcuRed, Lenguaje de Programación C Sharp, 2011)

  Dentro de las bondades de C# sobre C++, es la aceptación de valores NULL, enumeraciones, delegados, expresiones lambda y acceso directo a memoria, cosa que no se puede hacer en Java. Admite todos los conceptos de la programación orientada a objetos, todas las variables y métodos incluido el método main, el punto de entrada de la aplicación se encapsulan dentro de las definiciones de la clase. La clase puede implementar cualquier número de interfaces, los métodos que invalidan a los métodos virtuales en una clase primaria requieren la palabra override. (Microsoft.inc, Unspecified)

Cuadro comparativo.

	Paradigma Procedural	Paradigma Orientada a Objetos
Descripción	Son más simples, lo cual puede hacerlo más fáciles de leer, ya que son secuenciales. La secuencia de las instrucciones debe ser lógica, cada cálculo o función es más fácil de comprenderse. Los errores hasta cierto punto son más fáciles de detectar debido a la lógica con la que están estructurados, generando así como mayor producción por parte del programador y tester.	Encapsula los atributos y métodos de los objetos. Reusabilidad. La clases que están bien diseñadas pueden ser mejoradas y reutilizadas en la posterioridad. Mantenibilidad. Debido a la abstracción de los sistemas, tienden a ser más intuitivos. Modificabilidad. Es posible añadir, suprimir, o modificar las clases. Modularidad. Se pueden dividir los problemas en pequeñas partes para generar un approach específico y más fácil de realizar. Trata de modelar de forma más cercana a algo tangible la programación.
Diferencias	Series de instrucciones las cuales se van ejecutando paso a paso, por lo tanto se debe tener una secuencia lógica para que el resultado de la computación sea la correcta y eficiente. La programación procedural trata de darle solución a un problema de principio a fin con un código Se describe un algoritmo el cual se detallan los pasos secuenciales necesarios	La forma del planteamiento de los problemas es más compleja, las variables elegantes, u objetos son más complejos que muchos sistemas procedurales. La programación orientada a objetos trata de solucionar problemas más apegados a la realidad, por lo tanto el sistema es más complejo y dinámico. Al contrario de la programación imperativa, se

	para la resolución del problema.	genera una estructura abstracta, por medio de interfaces, abstracción de datos, encapsulación, eventos, modularidad, herencia y polimorfismo.

Lenguajes de Programación Orientada a Objetos

En esta sección se muestran las diferencias entre los diversos lenguajes de programación que soportan el paradigma de la programación orientada a objetos.

C++

Una clase:

```
#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    // Access specifier
    public:

    // Data Members
    string geekname;

    // Member Functions()
    void printname()
    {
       cout << "Geekname is: " << geekname;
    }
};
```

Usando la palabra reservada “class” junto al nombre de la clase, dentro de las llaves se declara los métodos y datos miembros dentro del accesador que querramos.

Herencia:

```
#include <bits/stdc++.h>
using namespace std;

//Base class
class Parent
{
    public:
      int id_p;
};

// Sub class inheriting from Base Class(Parent)
class Child : public Parent
{
    public:
      int id_c;
};
```

Una vez y se haya creado la superclase, se puede crear una subclase, por medio de dos puntos se instancia que está heredando la clase Parent.

Poliformismo:

```
using namespace std;
class Geeks
{
    public:

    // function with 1 int parameter
    void func(int x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name but 1 double parameter
    void func(double x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name and 2 int parameters
    void func(int x, int y)
    {
        cout << "value of x and y is " << x << ", " << y << endl;
    }
};
```

Con el método func, dependiendo del mensaje que se le envía al método, será la forma en que este actúe, si se le instancia con un entero, con un double, o con dos enteros, como lo muestra el ejemplo.

Encapsulamiento:

```
#include<iostream>
using namespace std;

class Encapsulation
{
    private:
        // data hidden from outside world
        int x;

    public:
        // function to set value of
        // variable x
        void set(int a)
        {
            x =a;
        }

        // function to return value of
        // variable x
        int get()
        {
            return x;
        }
};
```

El dato miembro privado x, jamás es directamente modificado por parte del usuario, sino que se utiliza un método Setter para poder asignarle un valor.

Java

Una clase:

```
public class Dog {
    String breed;
    int age;
    String color;

    void barking() {
    }

    void hungry() {
    }

    void sleeping() {
    }
}
```

Se especifica el asesor de la clase, y dentro de las llaves el tipo de accesor retorno del método, nombre del método y su definición.

Herencia:

```
//Java program to illustrate the
// concept of single inheritance
import java.util.*;
import java.lang.*;
import java.io.*;

class one
{
    public void print_geek()
    {
        System.out.println("Geeks");
    }
}

class two extends one
{
    public void print_for()
    {
        System.out.println("for");
    }
}
```

Para heredar de una clase, basta con poner “extends” después de nombrar la subclase, y se podrá acceder a los métodos de la superclase.

Poliformismo:

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

En este ejemplo se utiliza tanto herencia como polimorfismo, dependiendo de la clase de la que se instancie el objeto, éste realizará de diferente forma el método “animalSound()”.

Encapsulamiento:

```
// Java program to demonstrate encapsulation
public class Encapsulate
{
    // private variables declared
    // these can only be accessed by
    // public methods of class
    private String geekName;
    private int geekRoll;
    private int geekAge;

    // get method for age to access
    // private variable geekAge
    public int getAge()
    {
      return geekAge;
    }

    // get method for name to access
    // private variable geekName
    public String getName()
    {
      return geekName;
    }
```

Solamente los métodos públicos son los que podrán acceder a los atributos privados, por medio de setters y getters, como “getAge” o “getName”.

Python

Una clase:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Con la palabra reservada “class” seguido del nombre de la clase se crea la clase. Def_init es como una especie de constructor que permite asignar o inicializar los datos miembro.

Herencia:

```
class Polygon:
     def __init__(self, no_of_sides):
          self.n = no_of_sides
          self.sides = [0 for i in range(no_of_sides)]

     def inputSides(self):
          self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in range(s

     def dispSides(self):
          for i in range(self.n):
               print("Side",i+1,"is",self.sides[i])
```

```
class Triangle(Polygon):
     def __init__(self):
          Polygon.__init__(self,3)

     def findArea(self):
          a, b, c = self.sides
          # calculate the semi-perimeter
          s = (a + b + c) / 2
          area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
          print('The area of the triangle is %0.2f' %area)
```

Dado que un triángulo es un polígono, esta clase puede heredar los métodos de Polygon, solo que como es una clase especializada, el método área,es diferente y solo para un triángulo, que tiene 3 lados.

Poliformismo:

```
class Bird:
  def intro(self):
    print("There are many types of birds.")

  def flight(self):
    print("Most of the birds can fly but some cannot.")

class sparrow(Bird):
  def flight(self):
    print("Sparrows can fly.")

class ostrich(Bird):
  def flight(self):
    print("Ostriches cannot fly.")
```

Al igual que en Java, mostraremos el polimorfismo por medio de la herencia, la clase Bird, es heredada a sparrow y ostrich, las cuales redefinen el método flight.

Encapsulamiento:

```
#!/usr/bin/env python

class Car:

def __init__(self):
self.__updateSoftware()

def drive(self):
print 'driving'

def __updateSoftware(self):
print 'updating software'

redcar = Car()
redcar.drive()
#redcar.__updateSoftware()  not accesible from object.
```

C#

Una clase:

```
// C# program to illustrate the
// Initialization of an object
using System;

// Class Declaration
public class Dog {

    // Instance Variables
    String name;
    String breed;
    int age;
    String color;

    // Constructor Declaration of Class
    public Dog(String name, String breed,
                int age, String color)
    {
        this.name = name;
        this.breed = breed;
        this.age = age;
        this.color = color;
    }

    // method 1
    public String getName()
    {
        return name;
    }
```

Herencia:

```
// C# program to illustrate the
// concept of inheritance
using System;
namespace ConsoleApplication1 {

// Base class
class GFG {

   // data members
    public string name;
    public string subject;

    // public method of base class
    public void readers(string name, string subject)
    {
        this.name = name;
        this.subject = subject;
        Console.WriteLine("Myself: " + name);
        Console.WriteLine("My Favorite Subject is: " + subject);
    }
}

// inheriting the GFG class using :
class GeeksforGeeks : GFG {

    // constructor of derived class
    public GeeksforGeeks()
    {
        Console.WriteLine("GeeksforGeeks");
    }
}
```

Poliformismo:

```
// C# program to demonstrate the method overriding
// without using 'virtual' and 'override' modifiers
using System;

// base class name 'baseClass'
class baseClass

{
    public void show()
    {
        Console.WriteLine("Base class");
    }
}

// derived class name 'derived'
// 'baseClass' inherit here
class derived : baseClass
{

    // overriding
    new public void show()
    {
        Console.WriteLine("Derived class");
    }
}
```

Encapsulamiento:

```
// C# program to illustrate encapsulation
using System;

public class DemoEncap {

    // private variables declared
    // these can only be accessed by
    // public methods of class
    private String studentName;
    private int studentAge;

    // using accessors to get and
    // set the value of studentName
    public String Name
    {

        get
        {
            return studentName;
        }

        set
        {
            studentName = value;
        }

    }
```

Conclusiones

Después de analizar desde varias fuentes lo que era el paradigma procedural y compararlo con el orientado a objetos he llegado a dos conclusiones. La primera es que no es que un paradigma sea inferior o superior a otro, sino que dependiendo del problema que se intente resolver, este debería ser el paradigma que deberíamos utilizar. En contexto, si queremos desarrollar un sistema para poder almacenar y procesar datos en una calculadora, o un dispositivos con capacidades de procesamiento y almacenamiento reducidas, no podemos generar un sistema orientado a objetos que termine por pesar más la documentación, que lo que tenemos a nuestro alcance. De igual manera sería imposible llegar a crear aplicaciones empresariales, generar conexión a bases de datos y crear una interfaz gráfica a base instrucciones secuenciales.

Y mi segunda conclusión sería que después de poco más de un mes del curso de Programación II, encuentro cada vez más ventajas de la programación orientada a objetos, y como ésta en realidad es toda una forma de ver la programación, una filosofía que en realidad trata de modelar la vida real, y lo compleja que esta puede llegar a ser.

No queda más que seguir indagando y capacitando para obtener las competencias necesarias para ser un buen programador en cualquier sea el enfoque del sistema a desarrollar.

Bibliografía

Asensio, R. M.-B. (2008, Febrero 07). *Informática, lenguajes de programación*. Retrieved from Universidad de Murcia: https://www.um.es/docencia/barzana/DIVULGACION/INFORMATICA/Lenguaje-FORTRAN.html

Challenger-Pérez, I., Yanet , D. R., & Roberto Antonio, B. G. (2014, Junio). *El lenguaje de programación Python/The programming language Python.* Retrieved from Redalyc.org: https://www.redalyc.org/html/1815/181531232001/

EcuRed. (2011). *C++*. Retrieved from EcuRed.cu: https://www.ecured.cu/C%2B%2B

EcuRed. (2011). *Lenguaje de Programación Ada*. Retrieved from EcuRed.cu: https://www.ecured.cu/Lenguaje_de_programación_Ada

EcuRed. (2011). *Lenguaje de Programación C Sharp.* Retrieved from EcuRed.cu: https://www.ecured.cu/Lenguaje_de_Programación_C_Sharp#Historia

EcuRed. (2011). *Pascal*. Retrieved from Ecured.cu: https://www.ecured.cu/Pascal

Gómez, I. R. (2016, Mayo 10). *Cobol ¿Que es Cobol?* Retrieved from Medium.com: https://medium.com/enredando-con-programacion/cobol-que-es-cobol-3f86fa3a4394

Holzner, S. (2012). *La biblia de Java 2.* editorial Anaya.

Microsoft.inc. (Unspecified). *Introducción al lenguaje C# y .NET Framework.* Retrieved from Microsoft.com: https://docs.microsoft.com/es-es/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

Reyna, D. (2012, Mayo). *Lenguajes Procedurales y No Procedurales.* Retrieved from DuocUC: https://www.duocuc.com/doc/98355274/Procedural-vs-No-Porcedural

Unknown. (2009). *Paradigmas de Programación.* Retrieved from UTN: http://www.frlp.utn.edu.ar/materias/sintaxis/ApunteLengImperativos.pdf

Weisfield, M. (March de 2013). The Object-Oriented Thought Process. NewYork, New York, United States.