Universidad Autónoma de Ciudad Juárez

División Multidisciplinaria Ciudad Universitaria

Cuadro comprativo de lenguajes de Programacion Orientada a Objetos

Programacion II

Docente: Alan Ponce

Edgar Omar Reyes Olivas 169818

Licenciatura en Ingeniería de Software

08 de Marzo de 2019

Introducción

En este trabajo se definirán dos de los principales paradigmas de programación que existen y se brindarán ejemplos de lenguajes de cada uno de los paradigmas, también se realizará una comparación entre ambos paradigmas de programación.

Hay diversos lenguajes de programación que soportan el paradigma orientado a objetos por lo cual se hará una comparación entre cuatro de ellos y se mostrarán ejemplos de cómo diversas características del paradigma son implementadas en cada uno de los lenguajes.

1. Definición de lenguajes procedural

La programación procedural es un paradigma de programación que se basa en el concepto las llamadas a procedimientos remotos. Los procedimientos, también conocidos como rutinas, subrutinas o funciones, son los principales elementos de la programación procedural.

La programación procedural usa la estrategia de procesamiento de información *top-down*, el cual consiste en escribir una lista de instrucciones que la computadora deberá ejecutar paso por paso basándose en los elementos previamente mencionados. C, Pascal y FORTRAN son ejemplos de lenguajes de programación procedurales.

1.1 C

C es un lenguaje de programación desarrollado en los laboratorios Bell de AT&T en el año de 1972. Fue diseñado y escrito por Dennis Ritchie. A finales de los 70, C comenzó a reemplazar a lenguajes de programación más reconocidos en la época, como lo fueron PL/1 y ALGOL.

```
main( )
{
    int  i = 20 ;
    display ( i ) ;
}

display ( int  j )
{
    int  k = 35 ;
    printf ( "\n%d", j ) ;
    printf ( "\n%d", k ) ;
}
```

Figura 1. Ejemplo de Código en C.

1.2 Pascal

Pascal es un lenguaje de programación procedural, diseñado en 1968 y publicado en 1970 por Niklaus Wirth y nombrado en honor al matemático y filósofo francés Blaise Pascal.

Pascal es ejecutado en una variedad de plataforma, como Windows, Mac OS y varias versiones de UNIX/Linux.

```
(* function returning the max between two numbers *)
function max(num1, num2: integer): integer;

var
   (* local variable declaration *)
   result: integer;

begin
   if (num1 > num2) then
      result := num1

   else
      result := num2;
   max := result;
end;
```

Figura 2. Ejemplo de Código en Pascal.

1.3 FORTRAN

FORTRAN (mathematical FORmula TRANslation system) fue originalmente desarrollado en 1954 por IBM. Fortran fue uno de los primeros lenguajes en permitirle al programador el uso de sentencias de alto nivel antes que un language de ensamblador en particular. Esto resultó en que los programas fueran más fáciles de leer, entender, y debuggear.

```
PROGRAM circle_area
   IMPLICIT NONE
   !reads a value representing the radius of a circle,
   !then calculates and writes out the area of the circle.

   REAL :: radius, area
   REAL, PARAMETER :: pi=3.141592

   READ (5,*) radius
   area = pi*radius*radius      !calculate area
   WRITE (6,*) area

END PROGRAM circle_area
```

Figura 3. Ejemplo de Código en FORTRAN.

2. Definición de lenguaje orientada a objetos

Indicar lenguajes de programacion y una resena de creacion y evolucion (C++, Java, Python, C#)

La programación orientada a objetos es un paradigma de programación basado en el concepto de objetos, los cuales contienen datos en forma de campos, comunmente conocidos como atributos; y código, en forma de funciones, a menudos conocidos como métodos.

2.1 C++

C++ fue diseñado para proveer de las facilidades de Simula para la organización del programa así como la eficiencia y flexibilidad de C. Simula es la fuente inicial de los

mecanismos abstractos de C++. También el concepto de clases fue obtenido de ahí. Sin embargo, plantillas y excepciones llegaron a C++ desde diferentes fuentes de inspiración.

La evolución de C++ se dio siempre en el contexto de su uso. El creador pasó grandes cantidades de tiempo escuchando a los usuarios y buscando opiniones de programadores experimentados. En particular, los colegas del creador en los laboratorios Bell de AT&T fueron esenciales para el crecimiento de C++ durante su primera década.

En 1979, el trabajo en “C con clases” comenzó. Las principales características incluían clases y clases derivadas, control de acceso público/privado, constructores y destructores.

En 1984, “C con clases” fue renombrado a C++. Para ese entonces, C++ había adquirido funciones virtuales, sobrecarga de operadores y funciones, referencias, y las librerías de entrada y salida, y de números complejos.

En 1985 fue el primer lanzamiento comercial de C++.

En 1991 se lanzó la segunda edición de C++, la cual presentaba programación genérica con plantillas y el manejo de errores basado en excepciones.

En 1997 la tercera edición de C++ introdujo ISO C++, incluyendo espacios de nombres y conversión dinámica de tipos.

En 2011, el estándar ISO C++ 11 fue formalmente aprobado.

En 2013 la cuarta edición del lenguaje fue publicada.

```
class Distance                    //English Distance class
   {
   private:
      int feet;
      float inches;
   public:
                                  //constructor (no args)
      Distance() : feet(0), inches(0.0)
         {  }
      //Note: no one-arg constructor
                                  //constructor (two args)
      Distance(int ft, float in)  : feet(ft), inches(in)
         {  }

      void getdist()              //get length from user
         {
         cout << "\nEnter feet: ";  cin >> feet;
         cout << "Enter inches: ";  cin >> inches;
         }
      void showdist()             //display distance
         { cout << feet << "\'-" << inches << '\"'; }
   };
//////////////////////////////////////////////////////////////
int main()
   {
   Distance dist1(11, 6.25);      //two-arg constructor
   Distance dist2(dist1);         //one-arg constructor
   Distance dist3 = dist1;        //also one-arg constructor

                                  //display all lengths
   cout << "\ndist1 = ";  dist1.showdist();
   cout << "\ndist2 = ";  dist2.showdist();
   cout << "\ndist3 = ";  dist3.showdist();
   cout << endl;
   return 0;
   }
```

Figura 4. Ejemplo de Código en C++.

2.2 Java

Java fue creado por James Gosling, Patrick Naughton, Chris Warth, Ed Frank, y Mike Sheridan en Sun Microsystems, Inc. en 1991. Tomó 18 meses en desarrollar la primer versión funcional. Este lenguaje fue llamado inicialmente “Oak”, pero fue renombrado a “Java” en 1995.

La motivación principal para el desarrollo de java fue la necesidad de un lenguaje de programación que fuera independiente de la plataforma, que pudiera ser usado para crear software que estuviera embebido en distintos aparatos electrónicos como hornos de microondas y controles remotos. Con el surgimiento de la *World Wide Web*, java fue propulsado al primer plano de los lenguajes de programación.

El lanzamiento inicial de java fue revolucionario pero eso no marcó el fin de la era de innovación rápida para java. Después del lanzamiento de java 1.0, los diseñadores ya habían creado java 1.1. La versión 1.1 añadió nuevas librerías y redefinió la manera en la que se manejaban los eventos. Java 2 mejoró la máquina virtual de java así como agregó

nuevas características como *Swing*. La siguiente gran versión de java fue la J2SE 5, fundamentalmente expandió el alcance, poder y rango del lenguaje.

El siguiente lanzamiento de java fue la versión llamada Java SE 6 la cual mejoró las librerías de la API. Java SE 7 fue el siguiente lanzamiento en donde se siguió mejorando el API.La versión más reciente de Java es la de Java SE 8 la cual incluyó las expresiones lambda.

```
class DrumKit {

  boolean topHat = true;
  boolean snare = true;

  void playTopHat() {
    System.out.println("ding ding da-ding");
  }

  void playSnare() {
    System.out.println("bang bang ba-bang");
  }
}

class DrumKitTestDrive {
  public static void main(String [] args) {

    DrumKit d = new DrumKit();
    d.playSnare();
    d.snare = false;
    d.playTopHat();

    if (d.snare == true) {
      d.playSnare();
    }
  }
}
```

Figura 5. Ejemplo de Código en Java.

2.3 Python

Python fue creado por Guido van Rossum a finales de los años 80's, fue diseñado como una respuesta al lenguaje de programación ABC que era el lenguaje de primer plano en los Países Bajos. Entre las características principales de Python tenía manejo de excepciones y estaba dirigido al sistema operativo Amoeba.

Python 0.9.0 fue publicado en 1991, la versión incluía lambda, mapas, entre otras características. En el año 200 la versión 2.0 fue lanzada. Esta versión era más de código abierto, entre las características añadidas se encuentra el recolector de basura. Python 3.0 fue la siguiente versión y fue publicada en diciembre del 2008, entre las características añadidas está la de la sentencia print que fue reemplazada por la funcion print().

```
class PartyAnimal:
   x = 0

   def party(self) :
     self.x = self.x + 1
     print("So far",self.x)

an = PartyAnimal()
an.party()
an.party()
an.party()
PartyAnimal.party(an)
```

Figura 6. Ejemplo de Código de Python.

2.4 C#

Mientras que Java se dirigía exitosamente a las cuestiones que rodeaban a la portabilidad en el internet, aún le hacían falta algunas funciones importantes. Una de ellas es la interoperabilidad entre lenguajes de programación. Esta es la habilidad de que el código producido en un lenguaje pueda trabajar fácilmente con el código producido por otro lenguaje. La interoperabilidad de lenguajes es necesaria para la creación de grandes sistemas. Otra de característica que faltaba en Java era la completa integración en un ambiente de Windows.

La respuesta a estas y otras necesidades fue la creación del C#. C# fue creado en Microsoft a finales de los años 90. Su primer lanzamiento fue a mediados de los años 2000. C# está directamente relacionado con C, C++ y Java. Al haber construido con unos buenos cimientos, C# ofrecía una migración fácil desde lenguajes como C, C++ o Java.

Desde su primer lanzamiento, C# ha ido evolucionando a un paso muy rápido. Con el lanzamiento de la versión 2.0 se añadieron varias características fundamentales tales como métodos anónimos, tipos genéricos, entre otros. La siguiente gran publicación fue la versión 3.0 donde la principal adición fue la de las funciones lambda. La versión 4.0 introdujo características como los enlaces dinámicos.

```
using System;

namespace BoxApplication {
   class Box {
      public double length;   // Length of a box
      public double breadth;  // Breadth of a box
      public double height;   // Height of a box
   }
   class Boxtester {
      static void Main(string[] args) {
         Box Box1 = new Box();   // Declare Box1 of type Box
         Box Box2 = new Box();   // Declare Box2 of type Box
         double volume = 0.0;    // Store the volume of a box here

         // box 1 specification
         Box1.height = 5.0;
         Box1.length = 6.0;
         Box1.breadth = 7.0;

         // box 2 specification
         Box2.height = 10.0;
         Box2.length = 12.0;
         Box2.breadth = 13.0;

         // volume of box 1
         volume = Box1.height * Box1.length * Box1.breadth;
         Console.WriteLine("Volume of Box1 : {0}",  volume);

         // volume of box 2
         volume = Box2.height * Box2.length * Box2.breadth;
         Console.WriteLine("Volume of Box2 : {0}", volume);
         Console.ReadKey();
      }
   }
}
```

Figura 7. Ejemplo de Código en C#.

3. Cuadro comparativo de Programación Procedural y POO.

Paradigma Procedural	Paradigma Orientada a Objetos
1. Está basado en procedimientos. 2. Sigue un alcance de arriba abajo. 3. No tiene especificadores de acceso. 4. Los datos se pueden mover libremente de función en función 5. La mayoría de las funciones utilizan datos globales para que puedan ser accedidos por las funciones en el sistema. 6. No contiene ninguna forma adecuada de ocultamiento de datos. 7. La sobrecarga no es posible.	1. Está basado en clases y objetos. 2. Sigue un alcance de abajo arriba. 3. Contiene especificadores de acceso como público, privado, protegido. 4. Los objetos pueden moverse y comunicarse entre ellos mediante funciones miembro o métodos. 5. Los datos no se pueden mover tan fácilmente de función en función, se tiene un control de los datos mediante los especificadores de acceso. 6. Provee del ocultamiento de datos lo cual brinda más seguridad. 7. Brinda sobrecarga de funciones y de operadores.

Lenguajes de Programación Orientada a Objetos

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

C++

Incluir codigo de como implementar:

- Una clase
- Herencia
- Encapsulamiento

```
// counten.cpp
// inheritance with Counter class
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
class Counter                           //base class
   {
   protected:                           //NOTE: not private
      unsigned int count;               //count
   public:
      Counter() : count(0)              //no-arg constructor
         {  }
      Counter(int c) : count(c)         //1-arg constructor
         {  }
      unsigned int get_count() const    //return count
         { return count; }
      Counter operator ++ ()            //incr count (prefix)
         { return Counter(++count); }
   };
////////////////////////////////////////////////////////////////
class CountDn : public Counter          //derived class
   {
   public:
      Counter operator -- ()            //decr count (prefix)
         { return Counter(--count); }
   };
////////////////////////////////////////////////////////////////
int main()
   {
   CountDn c1;                          //c1 of class CountDn

   cout << "\nc1=" << c1.get_count();    //display c1

   ++c1; ++c1; ++c1;                    //increment c1, 3 times
   cout << "\nc1=" << c1.get_count();    //display it

   --c1; --c1;                          //decrement c1, twice
   cout << "\nc1=" << c1.get_count();    //display it
   cout << endl;
   return 0;
   }
```

Figura 8. Ejemplo de Código en C++ Implementando Una Clase, Herencia y Encapsulamiento.

Polimorfismo

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

class Shape {
   protected:
      int width, height;
   public:
      Shape( int a = 0, int b = 0){
         width = a;
         height = b;
      }
      int area() {
         cout << "Parent class area :" <<endl;
         return 0;
      }
};
class Rectangle: public Shape {
   public:
      Rectangle( int a = 0, int b = 0):Shape(a, b) { }
      int area () {
         cout << "Rectangle class area :" <<endl;
         return (width * height);
      }
};

class Triangle: public Shape {
   public:
      Triangle( int a = 0, int b = 0):Shape(a, b) { }
      int area () {
         cout << "Triangle class area :" <<endl;
         return (width * height / 2);
      }
};

// Main function for the program
int main() {
   Shape *shape;
   Rectangle rec(10,7);
   Triangle  tri(10,5);
   // store the address of Rectangle
   shape = &rec;
   // call rectangle area.
   shape->area();
   // store the address of Triangle
   shape = &tri;
   // call triangle area.
   shape->area();
   return 0;
}
```

Figura 9. Ejemplo de Código en C++ Implementando el Polimorfismo.

Java

Incluir codigo de como implementar:

- Una clase
- Herencia
- Polifirmismo
- Encapsulamiento

```
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");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object

    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

Figura 10. Implementación de una clase, herencia, polimorfismo y encapsulamiento en Java.

Python

Incluir codigo de como implementar:

- Una clase
- Herencia
- Polifirmismo
- Encapsulamiento

```python
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.")

obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()

obj_bird.intro()
obj_bird.flight()

obj_spr.intro()
obj_spr.flight()

obj_ost.intro()
obj_ost.flight()
```

*Figura 11.*Implementación de una clase, herencia, polimorfismo y encapsulamiento en Python.

C#

Incluir codigo de como implementar:

- Una clase
- Herencia
- Polimorfismo

```
using System;

namespace PolymorphismApplication {
   class Printdata {
      void print(int i) {
         Console.WriteLine("Printing int: {0}", i );
      }
      void print(double f) {
         Console.WriteLine("Printing float: {0}" , f);
      }
      void print(string s) {
         Console.WriteLine("Printing string: {0}", s);
      }
      static void Main(string[] args) {
         Printdata p = new Printdata();

         // Call print to print integer
         p.print(5);

         // Call print to print float
         p.print(500.263);

         // Call print to print string
         p.print("Hello C++");
         Console.ReadKey();
      }
   }
}
```

Figura 12. Implementación de una clase, encapsulamiento y polimorfismo en C#.

Encapsulamiento

```
using System;

namespace InheritanceApplication {
   class Shape {
      public void setWidth(int w) {
         width = w;
      }
      public void setHeight(int h) {
         height = h;
      }
      protected int width;
      protected int height;
   }

   // Derived class
   class Rectangle: Shape {
      public int getArea() {
         return (width * height);
      }
   }
   class RectangleTester {
      static void Main(string[] args) {
         Rectangle Rect = new Rectangle();

         Rect.setWidth(5);
         Rect.setHeight(7);

         // Print the area of the object.
         Console.WriteLine("Total area: {0}",  Rect.getArea());
         Console.ReadKey();
      }
   }
}
```

Figura 13. Implementación de Encapsulamiento en C#.

Conclusiones

El propósito de este trabajo fue comparar dos de los principales paradigmas de programación que existen, el paradigma procedural y el paradigma orientado a objetos.

Primero, se definió el paradigma procedural y después se dio a conocer un poco de historia y ejemplos de código de tres lenguajes de programación pertenecientes a este paradigma, los cuales fueron C, Pascal y FORTRAN.

Después, se brindó una definición del paradigma orientado a objetos así como una breve explicación de la historia y evolución de cuatro lenguajes de programación que soportan este paradigma, los lenguajes incluidos fueron C++, Java, Python y C#.

Luego de haber dejado claro y de haber definido ambos paradigmas se realizó una tabla comparativa que muestra diferencias entre los paradigmas presentados, tomando en cuenta aspectos como la seguridad, la estructura y las bases de los paradigmas.

Finalmente se añadieron códigos en los cuatro lenguajes de programación previamente mencionados que mostrarán aspectos principales del paradigma orientado a objetos para mostrar así que hay diferentes formas de implementar los conceptos de dicho paradigma.