Universidad Autónoma de Ciudad Juárez

División Multidisciplinaria Ciudad Universitaria

Cuadro comprativo de lenguajes de Programacion Orientada a Objetos

Programacion II

Docente: Alan Ponce

Alumno Jorge Lozoya Acosta

Licenciatura en Ingeniería de Software

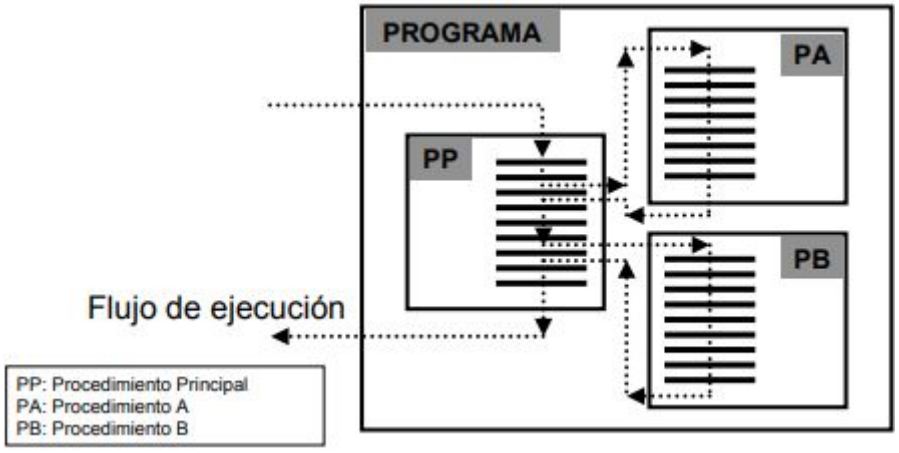
08 de Marzo de 2019

Introduccion

1.-Lenguajes procedurales o estructurados

Son aquellos cuyo código está formado por un único bloque contiguo de instrucciones. Es decir, se codifica de manera lineal, con un flujo que se sigue de principio a fin. Generalmente entre más grande y largo sea un código procedural, más difícil es de comprender y depurar.

Su estilo consiste de estructurar el código de un programa en funciones, en donde el flujo de ejecución puede ser auxiliado por diversos módulos para dividir el problema.



(Universidad de Oviedo, s.f.)

Lenguajes de programación estructurada y ejemplos de código

Uno de los lenguajes más conocidos es “C”, que a diferencia de Pascal (otro lenguaje de programación estructurada) no es posible declarar subrutinas dentro de otras subrutinas, o sea, no es estructurado por bloques.

Ejemplo de códigos en C:

```c
#include <stdio.h>
int main(void)
{
    // printf() muestra la cadena dentro de las comillas
    printf("Hello, World!");
    return 0;
}
```

```c
#include <stdio.h> /* Necesario para la función printf() */

int main(void) /* Función principal del programa */
{
    char resultado; /* Variable de tipo carácter donde se almacenará el resultado de las operaciones. */

    resultado=5+2; /*Realizamos una suma.*/
    printf("Resultado de la suma: %i\n",resultado);
    resultado=5-2; /*Realizamos una resta.*/
    printf("Resultado de la resta:%i\n",resultado);
    resultado=5*2; /*Realizamos una multiplicación.*/
    printf("Resultado de la multiplicacion: %i\n",resultado);
    resultado=5/2; /*Realizamos una división entera.*/
    printf("Resultado de la division:%i\n",resultado);

    return(0); /* Salimos del programa con el código 0 porque no ha habido errores. */

}
```

Ejemplo de “Hello World” en Pascal:

```pascal
program Hello;
begin
  writeln ('Hello, world.');
end.
```

Código de parte de la wiki de pascal, "Hello, World" (2018).

2.- Definición de lenguaje orientada a objetos

“Es el paradigma que define objetos y clases como la base para la programación. Cada objeto está definido por sus atributos y su comportamiento está definido por las operaciones que dichos objetos pueden hacer. La programación orientada a objetos expresa un programa como un conjunto de estos objetos,que colaboran entre ellos para realizar tareas ” (Universidad Nacional Autónoma de México, s.f.). Es decir que se puede lograr programas y módulos con una facilidad mayor a comparación con otros paradigmas.

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

C++: A partir de la creación de C a los inicios de los 70, Bjarnes Stroutstrup en el mismo laboratorio donde se desarrolló C (AT&T Bell) se diseño y desarrollo C++ buscando un lenguaje con las opciones de la programación orientada a objetos. Como curiosidad el nombre original era “C with classes”, sin embargo fue cambiado a como lo conocemos ahora C++. La evolución de C++ fue gracias a los lenguajes C y Simula 67 quienes agregaron las primeras características de “C with classes”, tiempo después se implementaron cualidades de Algol 68, generando así C++. (Universidad Virtual Biomédica, 2012).

Java: “Desde 1995, Java ha cambiado nuestro mundo … nuestras expectativas” (Oracle, s.f.). En los inicios de los 90, James Gosling y su equipo desarrolló el lenguaje orientado a objetos llamado Java. A diferencia de los lenguajes convencionales que son generalmente diseñados para ser compilados dentro de su propia máquina, Java corre a través de la máquina virtual de Java. ("History of Java programming language", 2006).

Este lenguaje toma mucha sintaxis de C y C++ pero contiene un modelado de objetos más simple. Java evoluciona siempre a favor de la compatibilidad de la tecnología y ser utilizado en todos los sistemas que existan.

Python: 2008, Paque y Abolafia describen la historia de Python, a finales de los años 80, pero se hizo público a finales de año 2000, su antecesor fue “ABC” y actualmente la última versión es Python 3.0, publicada en diciembre 2008. La característica de Python es que puede ser descompuesto en módulos, sentencias, expresiones y objetos. A evolucionado con una filosofía de ser multi paradigma, en donde la orientación a objetos y la estructurada está totalmente soportada, y rechaza las sintaxis exuberantes.

El “Zen” de Python se basa en ser bonito, simple y plano, a parte de tener una gran legibilidad.

C#: Es un lenguaje de programación orientado a objetos creado por Microsoft que nació como la sombra o la copia de Java en el 2000. Creció a través de diversas versiones y ahora permite crear aplicaciones de escritorio, aplicaciones Web y aplicaciones móviles. Contiene muchas características que hacen que el desarrollo de aplicaciones sea realmente rápido. Y Microsoft trata de dirigirse hacia cualquier sistema operativo y tiene la vista en la tecnología en la nube y el poder de la portabilidad.(CampusMVP, 2017).

3.- Cuadro comparativo de estos dos paradigmas

Paradigma Procedural	Paradigma Orientada a Objetos
Una orientación estructurada	Una orientación a objetos
Principal enfoque en conseguir el resultado	Principal enfoque es en la seguridad de los datos
Está dividido en unidades llamadas funciones	Está dividido en objetos
Sin modificadores de acceso	Con public, private y protect como modificadores de acceso
No cuenta con nada parecido a herencia	Se logra herencia con los modos de acceso
Los datos son compartidos a través de funciones	Los datos son compartidos en objetos a través de funciones miembros
Sin concepto de clase virtual	El concepto de funciones virtuales aparecen durante la herencia
Ejemplos: C, FORTRAN, Pascal	Ejemplos: C++, Java, C#, Python

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:

```
#include <iostream>  //librería estándar para entradas y salidas
using namespace std; //contiene palabras clave estandar
// Se crea una clase rectángulo con la siguiente sintaxis
// palabra reservada class y el nombre de la clase
class Rectangle {
    int width, height; //Se inicializan los valores de altura y ancho
  public:
    void set_values (int,int); //Funcion para obtener valores enteros
    int area() {return width*height;} //Funcion que devuelve el entero de la operacion
};
//Función para obtener valores
// De la clase rectangulo
void Rectangle::set_values (int x, int y) {
  width = x; // se igualan los valores de la clase con los valores que se obtendrán al llamar la función
  height = y;
}
// Inicio del main
int main () {
  Rectangle rect; //se crea un objeto de la clase rectángulo
  rect.set_values (3,4); // Se llama a la función con los valores dados
  cout << "area: " << rect.area(); //se llama a la función area, que devuelve la multiplicación
  return 0;
}
```

Codigo por la página oficial CPlusPlus.com

Herencia:

```
#include <iostream>  //librería estándar para entradas y salidas
using namespace std; //contiene palabras clave estandar
//Clase base de polígono
class Polygon {
//Valores protegidos para que solo clases derivadas puedan acceder a ellos
  protected:
    int width, height;
  public:
//Función para obtener valores
    void set_values (int a, int b)
```

```
    { width=a; height=b;}
 };

// Clase derivada rectángulo
class Rectangle: public Polygon {
  public:
    int area ()
      { return width * height; }
 };
// Clase derivada Triángulo
class Triangle: public Polygon {
  public:
    int area () // Función propia del área del triángulo
      { return width * height / 2; }
  };

int main () {
  Rectangle rect; //se crea un objeto de la clase Rectangulo
  Triangle trgl;
  rect.set_values (4,5); // Función heredada de polígonos
  trgl.set_values (4,5); //Función heredada de polígonos
  cout << rect.area() << '\n'; //Se imprime el resultado de la función área.
  cout << trgl.area() << '\n';
  return 0;
}
```

Codigo por la página oficial CPlusPlus.com

Poliformismo:

```
#include <iostream>  //librería estándar para entradas y salidas
using namespace std; //contiene palabras clave estandar
//Clase Base Poligono
class Polygon {
//Valores protegidos para que solo clases derivadas puedan acceder a ellos
  protected:
    int width, height;
  public:
//Función para obtener valores
    void set_values (int a, int b)
      { width=a; height=b; }
};
// Clase derivada rectángulo
class Rectangle: public Polygon {
  public:
    int area ()
      { return width * height; }
 };
// Clase derivada Triángulo
class Triangle: public Polygon {
  public:
    int area () // Función propia del área del triángulo
      { return width * height / 2; }
  };
int main () {
  Rectangle rect; //se crea un objeto de la clase Rectangulo
  Triangle trgl;
```

```
// Aquí es donde se aplica el polimorfismo
// Se crean dos apuntadores de la misma clase (ppoly) donde apuntan a distintos objetos
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
// los apuntadores de la clase polígono reciben valores con la función set_values
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << '\n';
  cout << trgl.area() << '\n';
  return 0;
}
```

Codigo por la página oficial CPlusPlus.com

Encapsulamiento:

```
class TObjGraf {

public: // Aquí estará el estado de la clase
  int      X;       // Propiedades
  int      Y;

  void Mostrar (void); // Comportamiento de la clase, métodos
};
```

/* Se “encapsulan” o guardan estos datos con el propósito de implementarlo mas adelante de el código* /

Código por Cortijo (s.f.).

Java

Incluir codigo de como implementar:

Una clase

```
//Declaración de una clase
//Palabra reservada “class” + el nombre de la clase
class Cuenta {
//Atributos de la clase
        String titular ;
        double saldo ;
//Métodos de la clase
         void ingreso (double cantidad){
                saldo = saldo + cantidad;
        }
        void reintegro (double cantidad){
                if (cantidad <= saldo)
                        saldo = saldo – cantidad;
        }
}
```

Código realizado por Moros (2009).

Herencia:

```
// Clase base
class Bicycle
{
    // La clase base contendrá dos atributos
    public int gear;
    public int speed;

    // Este es el constructor de la Bicicleta
    public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }

 // Tres métodos de la bicicleta
    public void applyBrake(int decrement) // Enfrenar
    {
        speed -= decrement;
    }

    public void speedUp(int increment) //Acelerar
    {
        speed += increment;
    }

    // toString() método para imprimir la informacion de nuestra clase
    public String toString()
    {
        return("No of gears are "+gear
                +"\n"
                + "speed of bicycle is "+speed);
    }
}

// Clase derivada
// Palabra clave extends, quiere decir que recibe los atributos y el comportamiento de
//Bicycle
class MountainBike extends Bicycle
{
    // Se añade un elemento de la clase derivada MountainBike
    public int seatHeight;

    // MountainBike tiene su propio constructor
    public MountainBike(int gear,int speed,
                        int startHeight)
```

```
    {
        // Invocando al constructor de la clase base
            //super es la palabra clave para activar a un constructor de una clase superior
        super(gear, speed);
        seatHeight = startHeight;
    }


    //MountainBike añade su propia función
    public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }

    // Sobrecarga del método toString()
    // de Bicicleta para imprimir más información
    @Override // Sobrecarga
    public String toString()
    {
        return (super.toString()+
                "\nseat height is "+seatHeight);
    }

}

// driver, testeo de la clase
public class Test
{
    public static void main(String args[])
    {
         // Se crea un objeto de la clase MountainBike y se le dan ciertos valores
        MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println(mb.toString());

    }
}
```

Código por Miglani (2017) en Geeks for Geeks.

Poliformismo

```
public class Animal{
   ...
   public void sound(){ // Comportamiento de la clase
      System.out.println("Animal is making a sound");
   }
}
public class Horse extends Animal{
...
    @Override
    public void sound(){ // Una clase derivada utilizando la función sound
        System.out.println("Neigh");
    }
}
public class Cat extends Animal{
...
    @Override
 public void sound(){
 //Otra clase derivada utilizando la función sound de distinta manera
// A eso se le conoce como polimorfismo
        System.out.println("Meow");
    }
}
```

Código por Singh (2014) en BeginnersBook

Encapsulamiento:

```
public class EncapTest {
// Variables privadas
// Sólo se pueden acceder a ellas por métodos públicos de la clase
   private String name;
   private String idNum;
   private int age;

//método para acceder a la edad
   public int getAge() {
      return age;
   }

// Método para acceder al nombre
   public String getName() {
      return name;
   }
// Método para obtener un Id y acceder al la variable idNum
   public String getIdNum() {
      return idNum;
   }
```

Código obtenido de Tutorials Point (s.f.).

Python

Incluir codigo de como implementar:

Una clase

```
class Person: //Definición de la clase
  def __init__(self, name, age): //se define una función con la palabra reservada def
          self.name = name //se asignan valores para nombre y edad
         self.age = age
// se crea un objeto de tipo persona
p1 = Person("John", 36)
//se imprimen los valores dados
print(p1.name)
print(p1.age)
```

Código obtenido de W3School.

Herencia:

```
# se crea una clase persona
class Person:
# se declara una función donde estarán los datos
    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last
# Función para devolver el nombre
    def Name(self):
        return self.firstname + " " + self.lastname

class Employee(Person): # Aquí es donde se ve la herencia
# Hereda los datos de __init__ y agrega el número de Staff
    def __init__(self, first, last, staffnum):
        Person.__init__(self,first, last)
        self.staffnumber = staffnum
# Función donde obtiene el empleado con su nombre y ID
    def GetEmployee(self):
        return self.Name() + ", " +  self.staffnumber

x = Person("Marge", "Simpson") # Aquí se crea a una Persona
y = Employee("Homer", "Simpson", "1007") #Aquí se crea a un Empleado

print(x.Name()) # Se imprime el nombre del objeto x
print(y.GetEmployee()) # se imprime los datos del empleado del objeto “y”
```

Código Python Tutorials (Klein, 2011).

Poliformismo:

```
# se crea una clase India
class India():
# se crean tres funciones donde cada una imprime una característica del país
    def capital(self):
        print("New Delhi is the capital of India.")

    def language(self):
        print("Hindi the primary language of India.")

    def type(self):
        print("India is a developing country.")

class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")

    def language(self):
        print("English is the primary language of USA.")

    def type(self):
        print("USA is a developed country.")

  # Se crean dos objetos, uno de cada clase
obj_ind = India()
obj_usa = USA()
// Se mandan a llamar cada función de cada clase
for country in (obj_ind, obj_usa):
# El polimorfismo se nota al momento en que las dos clases usan“country”
# y se comportan de diferente manera
    country.capital()
    country.language()
    country.type()
```

Código por Roy (2018) desde GeeksforGeeks

Encapsulamiento:

```
class Robot(object):
  def __init__(self):
     self.a = 123
     self._b = 123
 # Las variables de la clase que no deban ser directamente
#accesibles deberían de estar con un guión bajo de prefijo
     self.__c = 123

obj = Robot()
print(obj.a)
print(obj._b)
print(obj.__c)
```

// Al correr el programa, marcaría un error en obj.__c
Código obtenido de “Encapsulation”(2018)

C#

Incluir codigo de como implementar:

Una clase:

```
// Declaración de una clase pública
public class Hello
{

    // Variables
    public int a, b;

      // Funciones miembro
      public void display()
      {
          Console.WriteLine(“Class in C#”);
      }
}
```

Herencia

```
using System;
namespace ConsoleApplication1 {

// Clase Base
class GFG {

   // Datos
    public string name;
    public string subject;
```

```
    // Método público de la clase base
    public void readers(string name, string subject)
    {
        this.name = name;
        this.subject = subject;
        Console.WriteLine("Myself: " + name);
        Console.WriteLine("My Favorite Subject is: " + subject);
    }
}

// Herencia de la clase derivada GeeksforGeeks hacia la clase madre GFG:
class GeeksforGeeks : GFG {

    // constructor de la clase derivada
    public GeeksforGeeks()
    {
        Console.WriteLine("GeeksforGeeks");
    }
}

// Probador de la clase
class Sudo {

    // Método main
    static void Main(string[] args)
    {

        // Creando un objeto de la clase derivada
        GeeksforGeeks g = new GeeksforGeeks();

        //Llamando a una función de la clase base
        // Usando el objeto de la clase derivada
        g.readers("Kirti", "C#");
    }
}
}
```

Código por Mangal (2018) en GeeksforGeeks.

Poliformismo:

```
using System;

namespace PolymorphismApplication {
  class Printdata {
// Se declaran funciones que impriman diferentes tipos de datos
    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(); //Se crea solo un objeto de la clase Printdata

      // Se llama a print, y se activa la función int
      p.print(5);

      //Se llama a print, y se activa la función float
      p.print(500.263);

      // Se llama a print, y se activa la función string
      p.print("Hello C++");
      Console.ReadKey();
    }
  }
}
```

Código recuperado de Tutorials Point.

Encapsulamiento:

```
using System;

namespace RectangleApplication {
  class Rectangle {
    //member variables
    internal double length; //internal se puede reemplazar por public, privade o
protect
    internal double width;

    double GetArea() {
      return length * width;
    }
    public void Display() {
```

```
      Console.WriteLine("Length: {0}", length);
      Console.WriteLine("Width: {0}", width);
      Console.WriteLine("Area: {0}", GetArea());
    }
  }//end class Rectangle

  class ExecuteRectangle {
    static void Main(string[] args) {
      Rectangle r = new Rectangle();
      r.length = 4.5;
      r.width = 3.5;
      r.Display();
      Console.ReadLine();
    }
  }
}
```

Código recuperado de Tutorials Point.

Conclusiones

En esta seccion debera plantear sus conclusiones de acuerdo a su análisis comparativo.

El futuro de la programación por el momento está en la programación orientada a objetos, ya que los problemas actuales requieren de una gran cantidad de líneas de códigos, y la mejor manera de solucionar los problemas es dividiendo, cosa que la modularidad que OOP permite a los desarrolladores ayuda en gran escala. En contraste, la programación estructurada es buena para soluciones rápidas y pequeñas, algo que se está quedando obsoletas a las necesidades actuales.

Teniendo también la OOP una gran presencia en el campo laboral, es una necesidad para los aspirantes entender tanto la importancia como la implementación de este paradigma en en problemas reales.

Bibliografía

- CampusMVP. (2017, 18 octubre). Historia del lenguaje C#: pasado, presente y evolución. Recuperado 8 marzo, 2019, de https://www.campusmvp.es/recursos/post/historia-del-lenguaje-c-sharp-pasado-presente-y-evolucion.aspx
- Cortijo, F. (s.f.). Programación Orientada a Objetos en C++. Recuperado 7 marzo, 2019, de https://elvex.ugr.es/decsai/builder/intro/5.html
- CPlusPlus. (s.f.-a). Classes (I) - C++ Tutorials. Recuperado 7 marzo, 2019, de http://www.cplusplus.com/doc/tutorial/classes/
- CPlusPlus. (s.f.-b). Friendship and inheritance - C++ Tutorials. Recuperado 8 marzo, 2019, de http://www.cplusplus.com/doc/tutorial/inheritance/
- CPlusPlus. (s.f.-c). Friendship and inheritance - C++ Tutorials. Recuperado 7 marzo, 2019, de http://www.cplusplus.com/doc/tutorial/polymorphism/
- Encapsulation. (2018). Recuperado 7 marzo, 2019, de https://pythonprogramminglanguage.com/encapsulation/
- Evolution of C++. (s.f.). Recuperado 7 marzo, 2019, de https://gradestack.com/Programming-in-C-/Introduction-to-C-/Evolution-of-C-/21192-4329-47838-study-wtw
- Hello, World. (2018, 2 mayo). Recuperado 7 marzo, 2019, de http://wiki.freepascal.org/Hello,_World
- History of Java programming language. (2006). Recuperado 7 marzo, 2019, de http://www.freejavaguide.com/history.html
- Klein, B. (2011). Inheritance. Recuperado 8 marzo, 2019, de https://www.python-course.eu/python3_inheritance.php
- Mangal, K. (2018). C# | Class and Object. Recuperado 7 marzo, 2019, de https://www.geeksforgeeks.org/c-sharp-class-and-object/
- Miglani, G. (2017, enero). Inheritance in Java. Recuperado 7 marzo, 2019, de https://www.geeksforgeeks.org/inheritance-in-java/
- Moros, B. (2009). Clase y objetos en Java. Recuperado 7 marzo, 2019, de http://dis.um.es/~bmoros/privado/apuntes/Curso09-10/POO2-Java-0910.pdf
- Oracle. (s.f.). History of Java Technology. Recuperado 8 marzo, 2019, de https://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html
- Parewa Labs. (s.f.). C "Hello, World!" Program. Recuperado 7 marzo, 2019, de https://www.programiz.com/c-programming/examples/print-sentence
- Roy, R. (2018, 4 diciembre). Polymorphism in Python. Recuperado 7 marzo, 2019, de https://www.geeksforgeeks.org/polymorphism-in-python/

- Singh, C. (2014, octubre). Polymorphism in Java with example. Recuperado 7 marzo, 2019, de https://beginnersbook.com/2013/03/polymorphism-in-java/
- Tutorials Point. (s.f.-a). Java - Encapsulation. Recuperado 7 marzo, 2019, de https://www.tutorialspoint.com/java/java_encapsulation.htm
- Tutorials Point. (s.f.-b). C# Polymorphism. Recuperado 7 marzo, 2019, de https://www.tutorialspoint.com/csharp/csharp_polymorphism.htm
- Universidad de Oviedo. (s.f.). Arquitectura de Computadores. Recuperado 7 marzo, 2019, de http://www.atc.uniovi.es/telematica/2ac/Transparencias/T02-Programacion-Procedural.pdf
- Universidad Nacional Autónoma de México. (s.f.). Lenguaje de programación C#. Recuperado 8 marzo, 2019, de http://cursosenlinea.tic.unam.mx/cursos/Lenguaje_de_programacion_C_.htm
- Universidad Virtual Biomédica. (2012, 28 diciembre). Historia C++. Recuperado 7 marzo, 2019, de https://sites.google.com/site/universidadvirtualbiomedica/unidad-i-introduccion-a-el-lenguaje-c/historia-c
- W3School. (s.f.). Python Classes and Objects. Recuperado 8 marzo, 2019, de https://www.w3schools.com/python/python_classes.asp