## Universidad Autónoma de Ciudad Juárez

## División Multidisciplinaria Ciudad Universitaria

## 

## Cuadro comparativo de lenguajes de Programación Orientada a Objetos

## Programación II

## Docente: Alan Ponce

## 

## 

## Alumno Ricardo Jiménez Ivarra

## Licenciatura en Ingeniería en Sistemas Computacionales

## 08 de Marzo de 2019

# Introducción

El objetivo de este trabajo es presentar su trabajo por lo que se debe incluir:

1.-Definicion de lenguajes procedural:

Se trata de un estilo de programación basado en estructurar el

código de un programa en componentes, que reciben el nombre

de procedimientos, subrutinas o funciones.

**Beneficios de la programación procedural:**

Comprensibilidad:

La estructuración de un programa en procedimientos proporciona

una comprensión más clara del código al programador, lo cual facilita

las labores de diseño, depuración y mantenimiento.

Desarrollo modular:

Los procedimientos facilitan el desarrollo de un programa en módulos

independientes, permitiendo que personas o equipos diferentes

trabajen en cada módulo por separado.

Capacidad de reutilización:

El código de los procedimientos puede ser compilado y almacenado

en ficheros de librería. Éstas pueden ser reutilizadas en cualquier

programa mediante el enlazado.

**(Oviedo)**

**Indicar lenguajes de programación**

JAVA

C

PASCAL

BASIC

COBOL

FORTRAN

APL

RPG

CLIPPER

**(Morero, 1999)**

**ejemplos de código.**

introduzco los datos x y y

input x

input y

intercambio los valores x y y

auxiliar = x //guardo una copia del valor de x en auxiliar

x = y // guardo el valor de y en x, perdiendo su valor anterior, pero manteniendo una copia en auxiliar.

y = auxiliar //copio a y el valor de auxiliar, que es el valor inicial de x.

imprimo las variables (que ahora tienen los valores intercambiados)

print x

print y

**(EcuRed)**

A modo de ejemplo, si queremos mostrar el anterior, el posterior y un propio número de una lista (vector), un pseudocódigo por procedimientos o funciones (a alto nivel) sería:

funcion anterior\_posterior( numero ) {

out(numero-1);

out(numero);

out(numero+1);

}

for( i = 0; i < tamaño(lista); i++) {

anterior\_posterior( lista[i] );

}

2.- Definición de lenguaje orientada a objetos

Los lenguajes de programación orientados a objetos tratan a los programas como conjuntos de objetos que se ayudan entre ellos para realizar acciones. Entendiendo como objeto al entidades que contienen datos. Permitiendo que los programas sean más fáciles de escribir, mantener y reutilizar.

Los objetos tienen toda la información (atributos) que los diferencia de otros pertenecientes a otra clase. Por medio de unos métodos se comunican los objetos de una misma o diferente clase produciendo el cambio de estado de los objetos. Esto hace que a los objetos se les trate como unidades indivisibles en las que no se separan la información ni los métodos usados en su tratamiento.

**Indicar lenguajes de programación y una reseña de creación y evolución (C++, Java, Python, C#)**

JAVA

Java nace en 1991 con el nombre "OAK", posteriormente cambiado por Green por problemas legales, y finalmente con la denominación actual JAVA.

El objetivo de java era crear un lenguaje de programación parecido a C++ en estructura y sintaxis, fuertemente orientado a objetos, pero con una máquina virtual propia.

A día de hoy, podemos decir, que Java es uno de los lenguajes más importantes del mundo. Con una comunidad extendida en todos los componentes y más de 4 millones de desarrolladores, existen millones de dispositivos que lo usan. Ademas, tras el surgimiento de android, java se establecido como el lenguaje de programación para móviles más extendido del planeta.

C++

Su origen data del año 1979 y se le atribuye a Bjarne Stroustrup. El lenguaje que inspiró a Stroustrup fue el lenguaje Simula (lenguaje usado para simulaciones), que es considerado el primer lenguaje en permitir programación orientada a objetos.

En 1983 el nombre del lenguaje fue cambiado de “C with classes” a “C++”. Podemos entender con esto la imagen que tenía Stroustrup de su lenguaje como una mejora del lenguaje “C” (al ser ++ un incrementador de variable).

En 1998 el “Comité de estándares de C++” publicó su primera estandarización internacional ISO/IEC 14882:1998 (conocida también como C++98) la cual, al tener varios problemas, fue actualizada en 2003 (C++03). En 2011 se terminó y publicó la nueva estandarización del lenguaje (C++11).

PYTHON

Guido van Rossum ideó el lenguaje Python a finales de los 80 y comenzó a implementarlo en diciembre de 1989. En febrero de 1991 publicó la primera versión pública, la versión 0.9.0. La versión 1.0 se publicó en enero de 1994, la versión 2.0 se publicó en octubre de 2000 y la versión 3.0 se publicó en diciembre de 2008. hasta 2018, el desarrollo de Python lo lleva a cabo un colectivo de programadores dirigido por Guido van Rossum y bajo el paraguas de la fundación Python Software Foundation. En julio de 2018 Guido van Rossum anunció que dejaría de dirigir el desarrollo de Python y a principios de 2019 se pondrá en marcha un consejo director de cinco miembros elegidos entre los desarrolladores de Python.

C#

La última variante que ha surgido de C es el moderno C#. En el año 2000, Microsoft presenta su plataforma .NET junto con un nuevo lenguaje, C# (diseñado por Anders Hejlsberg), que servirá de lenguaje principal de la plataforma. C# es un híbrido de C++ y Java que fusiona, principalmente, la capacidad de combinar operadores propia del primero (no incorpora la herencia múltiple) con la plena orientación a objetos del segundo. La orientación a objetos es tal que el propio programa está encapsulado en una clase.Actualmente C# se encuentra entre los 10 lenguajes más utilizados. A pesar de su corta historia, ha recibido la aprobación del estándar de dos organizaciones: en el 2001 se aprueba el ECMA y en el 2003 el ISO.

3.- Cuadro comparativo de estos dos paradigmas

|  |  |
| --- | --- |
| Paradigma Procedural | Paradigma Orientada a Objetos |
| Tipo de programación imperativa, donde las declaraciones se ponen en procedimientos, que se pueden volver a llamar cuando sea necesario. C usa programación procedimental. | Estilo que trata los datos como objetos con atributos y métodos que pueden aplicarse a estos objetos y también ser heredados por otros objetos. |
| La programación procedimental no necesita objetos. Como su nombre lo indica, tiene procedimientos que podrían ser estructuras de datos, rutinas y subrutinas. | La POO se centra en clases y objetos. Al representar variables como objetos, se le puede pasar una función (método). Un objeto que pertenece a una clase en particular se puede tratar de forma independiente. |
| Un programa escrito de acuerdo a los principios de programación estructurada no solamente tendrá una mejor estructura sino también una excelente presentación. | La POO puede estar basada en clases, que, en este caso, los objetos se basan en clases predefinidas. |
| Los programas son más fáciles de entender, pueden ser leídos de forma secuencial. | La POO basada en prototipos también existe, por lo que no hay necesidad de clases y solo se utilizan objetos. |
| La programación procedimental no admite la herencia. La herencia solo se puede aplicar a los objetos. Debido a que la programación procedimental carece de objetos, carece de esta característica, por lo que la distingue de la POO. | La característica más distinguible del paradigma de POO es la herencia. La herencia da un impulso a POO, ya que permite una facilidad general a través de la cual el código se puede reutilizar y extender sin cambiar el código existente. |
| La programación procedimental no tiene esta habilidad. Como tal, subtipos y supertipos no pueden ser declarados. Tampoco se puede lograr la subtipificación del comportamiento. | En POO, se puede lograr un polimorfismo de subtipo, mediante el cual una función escrita para los elementos de un tipo de datos (supertipo) se puede hacer funcionar en los elementos de otro tipo de datos relacionado (subtipo). |

# 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 código de cómo implementar:

**Una clase**

class Empleado {

char\* m\_nombre;

char\* m\_departamento;

char\* m\_posicion;

long m\_salario;

void Imprimir( Empleado infoEmpleado);

}

**Herencia**

class Deposito {

private:

Persona\* titular;

double capital;

int plazoDias;

double tipoInteres;

public:

Deposito(…);

double liquidar();

double getIntereses();

double getCapital();

int getPlazoDias();

double getTipoInteres();

Persona\* getTitular();

};

**Polimorfismo**

#include <iostream>

#include <string>

using namespace std;

class Mascota

{

public:

Mascota(string nombre, int patas);

virtual string hablar() = 0;

private:

string nombre;

int patas;

};

Mascota::Mascota(string nombre, int patas):nombre(nombre),patas(patas)

{

cout << "Acaba de nacer tu mascota "<<nombre<<" con "<<patas<<"patas"<<endl;

}

class Perro : public Mascota

{

public:

Perro(string nombre);

string hablar();

};

Perro::Perro(string nombre):Mascota(nombre, 4)

{

}

string Perro::hablar()

{

return "GUAU";

}

class Gato : public Mascota

{

public:

Gato(string nombre);

string hablar();

};

Gato::Gato(string nombre):Mascota(nombre, 4)

{

}

string Gato::hablar()

{

return "MIAU";

}

**Encapsulamiento**

class TObjGraf {

public:

int X; // Propiedades

int Y;

TColor Color;

TPaintBox \*PaintBox;

void Mostrar (void); // Métodos

};

### Java

Incluir código de cómo implementar:

**Una clase**

public byte[] invoke(Map String,Object metaData, MboRemote sourceMbo, MboRemote targetMbo, String endPointName) throws MXException, RemoteException;

**Herencia**

Construyamos la clase Taxista.java con el siguiente código:

public class Taxista extends Persona {

private int nLicencia;

public void setNLicencia(int num)

{

nLicencia = num;

}

public int getLicencia()

{

return nLicencia;

}

}

Y construyamos ArranqueTaxista.java:

public class ArranqueTaxista {

public static void main (String arg[]){

Taxista tax1 = new Taxista();

tax1.setNombre("Luis");

tax1.setEdad(50);

System.out.println( tax1.getNombre());

System.out.println(tax1.getEdad());

}

}

Ahora intentemos usar el constructor que existía en la clase Persona que recibia el nombre de la persona y vamos a usarlo para la clase Taxista. Para ello construyamos la clase ArranqueTaxista2.java:

public class ArranqueTaxista2 {

public static void main (String arg[]){

Taxista tax1 = new Taxista("Jose");

tax1.setEdad(50);

System.out.println( tax1.getNombre());

System.out.println(tax1.getEdad());

System.out.println(tax1.getNLicencia());

}

}

**Polimorfismo**

public abstract class SeleccionFutbol {

protected int id;

protected String nombre;

protected String apellidos;

protected int edad;

// constructores, getter y setter

public void viajar() {

System.out.println("Viajar (Clase Padre)");

}

public void concentrarse() {

System.out.println("Concentrarse (Clase Padre)");

}

// IMPORTANTE -> METODO ABSTRACTO => no se implementa en la clase abstracta pero si en la clases hijas

public abstract void entrenamiento();

public void partidoFutbol() {

System.out.println("Asiste al Partido de Fútbol (Clase Padre)");

}

}

**Encapsulamiento**

public class Persona{

//Atributos

prívate int altura;

//Métodos

public int getAltura(){

return this.Altura;

}

public void setAltura(int unaAltura){

this.altura = unaAltura;

}

}

### Python

Incluir código de cómo implementar:

**Una clase**

class Fraccion:

def \_\_init\_\_(self,arriba,abajo):

self.num = arriba

self.den = abajo

**Herencia**

class Persona(object):

"Clase que representa una persona."

def \_\_init\_\_(self, identificacion, nombre, apellido):

"Constructor de Persona"

self.identificacion = identificacion

self.nombre = nombre

self.apellido = apellido

def \_\_str\_\_(self):

return " %s: %s, %s" %

(str(self.identificacion), self.apellido, self.nombre)

**Polifirmismo**

def frecuencias(secuencia):

""" Calcula las frecuencias de aparición de los elementos de

la secuencia recibida.

Devuelve un diccionario con elementos: {valor: frecuencia}

"""

# crea un diccionario vacío

frec = dict()

# recorre la secuencia

for elemento in secuencia:

frec[elemento] = frec.get(elemento, 0) + 1

return frec

**Encapsulamiento**

#!/usr/bin/env python

# Nombre de Fichero : encapsulacion.py

class Persona(object) :

"Calse Persona"

def \_\_init\_\_(self, pNombre,pEdad,pSueldo) :

self.setNombre(pNombre)

self.setEdad(pEdad)

self.\_\_setSueldo(pSueldo);

def setEdad(self, pEdad) :

self.\_\_edad = pEdad

def getEdad(self) :

return self.\_\_edad

def setNombre(self, pNombre) :

self.\_\_nombre = pNombre

def getNombre(self) :

return self.\_\_nombre

def \_\_setSueldo(self,pSueldo):

self.\_\_sueldo = pSueldo

def getSueldo(self):

return self.\_\_sueldo

nombre = property(getNombre, setNombre)

edad = property(getEdad, setEdad)

class Gerente(Persona) :

def \_\_init\_\_(self, pNombre,pEdad):

Persona.\_\_init\_\_(self, pNombre,pEdad,5000)

class Secretaria(Persona) :

def \_\_init\_\_(self, pNombre,pEdad) :

Persona.\_\_init\_\_(self, pNombre,pEdad,500)

if \_\_name\_\_ == '\_\_main\_\_':

g = Gerente("Mariano", 56)

s = Secretaria("Rocio", 33)

print "El Gerente es", g.nombre , " gana ", g.getSueldo()

print "La Secretaria es ", s.getNombre(), " gana ",s.getSueldo

### C#

Incluir código de cómo implementar:

**Una clase**

//Declaring an object of type MyClass.

MyClass mc = new MyClass();

//Declaring another object of the same type, assigning it the value of the first object.

MyClass mc2 = mc;

**Herencia**

using System;

public class A

{

private int value = 10;

public class B : A

{

public int GetValue()

{

return this.value;

}

}

}

public class C : A

{

// public int GetValue()

// {

// return this.value;

// }

}

public class Example

{

public static void Main(string[] args)

{

var b = new A.B();

Console.WriteLine(b.GetValue());

}

}

**Polifirmismo**

Persona[] personas = new Persona[2];

personas[0] = new Alumno("Jaime Correa");

personas[1] = new Empleado("Juan Perez");

for (int i = 0; i < personas.Length; i++)

{

Console.WriteLine(personas[i].Saludar());

}

**Encapsulamiento**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FrancysSytem\_Encapsulamiento

{

class Program

{

class Salon

{

private string nombre = "Omar";

public string Nombre

{

get { return nombre; }

set { nombre = value; }

}

public string nombres = "Carlos";

}

class MainClass

{

public static void Main()

{

Salon nuevo\_salon = new Salon();

Console.WriteLine("Encapsulación en c#");

Console.WriteLine("Nombre uno : {0}", nuevo\_salon.nombres); ///Variable Publica

Console.WriteLine("Nombre dos : {0}", nuevo\_salon.Nombre); ///Encapsulación

Console.WriteLine("\n\nfrancysystem.blogspot.com");

Console.ReadLine();

}

}

}

}

Conclusiones

En esta sección deberá plantear sus conclusiones de acuerdo a su análisis comparativo

No todos los lenguajes de programación se utilizan para los mismo, hay que saber que tipo de lenguaje de programación es el adecuado para un proyecto a realizar ya que algunos lenguajes son mas eficientes en tareas especificas que otras, lo que importa en los proyectos o aplicaciones a realizar es que tenga las menos líneas de código ya que así tendrá un mejor funcionamiento y será mas rápido a la hora de ejecutarlo.

Las principales diferencias entre las programación procedural y la orientada a objetos son que en la procedural los programas son mas legibles a la hora de entenderlos y la orientada a objetos nos acerca un poco mas a la experiencia de la vida real y por esta razón son 2 cosas muy diferentes.

Así que cuando se vaya a hacer un proyecto lo primero es averiguar cual es el lenguaje mas conveniente para realizar dicho proyecto cual de estos lenguajes será mas eficaz con los recursos a utilizar y lo que hará dicho programa, si el programa con un lenguaje usa 100 líneas de código que otro lenguaje que solamente se necesitan 50 líneas de código y el funcionamiento es el mismo pues por obvias razones tomaríamos el lenguaje que menos líneas de código se necesite.

# Referencias

EcuRed. (s.f.). *Ecured.* Obtenido de Programacion estructurada: https://www.ecured.cu/Programacion\_estructurada

Morero, F. (1999). *Introducción a la oop.* Obtenido de Desarrollo de software: https://kataix.umag.cl/~ruribe/Utilidades/Introduccion%20a%20la%20Programacion%20Orientada%20a%20Objetos.pdf

Oviedo, U. d. (s.f.). *uniovi.* Obtenido de Arquitectura de Computadores: http://www.atc.uniovi.es/telematica/2ac/Transparencias/T02-Programacion-Procedural.pdf