## Universidad Autónoma de Ciudad Juárez

## División Multidisciplinaria Ciudad Universitaria

## 

## Cuadro comparativo de lenguajes de Programación Orientada a Objetos

## Programación II

## Docente: Alan Ponce

## 

## 

## George Iván Rodríguez Gómes 16000

## Licenciatura en Ingeniería de Software

## 08 de Marzo de 2019

# Introduccion

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

**1.-Definicion de lenguajes procedural**

Programación imperativa o procedural, tiene como característica principal en sus programas poseer secuencias de instrucciones que deben llevarse a cabo como una receta o guión para resolver un problema determinado(Galindo y Pastó, 2015). Es importante aclarar que estos pueden trabajar con procedimientos y funciones, pero no debe confundirse con el paradigma funcional, en el cual todo se describe con una función(Juganaru Mathieu, 2014).

Algunos ejemplos de lenguajes de programación procedurales o imperativos son: lenguaje máquina, lenguaje ensamblador, C, Fortran, Cobol, Pascal, Ada, C++, C#, Java. Fortran, Cobol, Pascal, Ada, C++, C#, Java. A excepción del lenguaje máquina y el lenguaje ensamblador, los demás son lenguajes estructurados. Un lenguaje de programación puede verificar uno o más paradigmas. Por ejemplo, el lenguaje Java comprueba el paradigma orientado a objetos y el código que compone la parte de métodos de los objetos verifica el paradigma estructurado(Juganaru Mathieu, 2014).

Ejemplo de un paradigma funcional y orientado a objetos en java(Hoyos y Puertas, 2017):

El método mostrado en el código a) calcula la solución para una solución en particular, así el método debe ser aplicado en forma recursiva a cada una de las entradas, mediante la llamada al método invoke que recibe como parámetro el objeto task, representativo de una tarea recursiva, que se traduce en un conjunto de tareas individuales generadas para cada uno de las entradas contenidas en el objeto juegos (arreglo de String), tal como se muestra en el código b) . En este caso, es el método compute de la clase Task, el encargado de crear las tareas individuales para cada una de las entradas, tal como se muestra en el código c).

a)

public Grid solve(String cadena){

Grid grid = parseGrid(cadena);

return search(grid);

}

b)

Task task = new Task(juegos);

pool.invoke(task);

c)

protected void compute() {

Grid all = (new Sudoku()).solve(juego);

if(enlistados<juegos.length)

{

Task task2 = new Task(juegos);

invokeAll(task2);

}

}

**2.- Definición de lenguaje orientada a objetos**

Los objetos son todo, son una copia simplificada que definen su estado y su comportamiento. Existen tres principios fundamentales que gobiernan este tipo de programación(Juganaru Mathieu, 2014):

Encapsulación: en este principio se encapsulan datos, estados, operaciones y, en ocasiones, también eventos, en objetos. El código sería ejecutado, entonces, según la ocurrencia de eventos o de creación/destrucción de instancia de objetos. Prototipos, clases y herencias: El prototipo y la clase son las abstracciones del objeto; otros prototipos se definen de acuerdo con un prototipo existente. Tipificación y polimorfismo: constituyen la comprobación del tipo con respecto a la jerarquía de las clases.

**C++**

El lenguaje fue originalmente desarrollado alrededor de los años 80’s por Bjarne Stroustrup. Se diseño C++ para que fuera un mejor C, por lo que la mayoría de los programas en C también lo son de C++ (Savitch, 2007).Después de su lanzamiento y su estandarización C++ se ha desarrollado lentamente, en 2011 la organización internacional de estándares publicó el nuevo estándar, le dio un impulso para adaptarse a las necesidades del siglo 21(Horton y Weert, 2018).

Java

Java es un descendiente de C++ y C, hay características heredadas de estos lenguajes, por lo que la sintaxis y las características fundamentales de la programación orientada a objetos. El diseño original fue concebido por James Gosling, Patrick Naughton, Chris Warth, Ed Frank y Mike Sheridan en Sun Microsystem en 1991. El lenguaje se llamó originalmente Oak, para 1995 se cambió el nombre por java. Se buscaba un lenguaje independiente de la plataforma que se pudiera utilizar para crear software que se incrustará en dispositivos electrónicos. Las primeras aplicaciones fueron diseñadas para internet, su objetivo era ejecutar aplicaciones residentes en Web en la maquina local. La primera versión estable se comercializó rápidamente, a pesar de que las versiones apenas se mueven, los cambios suelen ser profundos. Los colaboradores involucrados en Java son numerosos, sin embargo, se ejerce cierto control sobre las versiones para garantizar la independencia de la plataforma(Azuela y Joyanes Aguilar, 2001) .

**Python**

Python fue concebido por Guido van Rossum, un programador holandés a finales de los 80 cuando se encontraba trabajando en el sistema operativo Amoeba. Se concibió para manejar excepciones y tener interfaces con Amoeba como sucesor del lenguaje ABC.

Para octubre del 2000 se lanza Python 2.0 que contenía nuevas características como completa recolección de basura y completo soporte a Unicode. Pero el mayor avance lo constituye que este comenzó a ser verdaderamente desarrollado por la comunidad, bajo la dirección de Guido. El Python 3.0 es incompatible con las anteriores. Muchas de las características introducidas en la versión 3 han sido compatibilizadas en la versión 2.6 para hacer de forma más sencilla la transición entre estas(Ivet, Diaz, y Becerra García, 2014).

**C#**

Microsoft anuncia en el año 2000 el lenguaje de programación C#, como respuesta a las nuevas necesidades de dispositivos móviles. Fue desarrollado por un equipo de programadores liderados por Anders Helsjberg y Scott Wiltamuth, basándose en la plataforma .NET como un lenguaje que permitieran a los programadores migrar a dicha plataforma. Basados en C, C++ y java. Las mejoras se basan en una poderosa librerías y componentes preconstruidos. La plataforma .Net es la infraestructura sobre la cual están basadas en web y de una gran cantidad de dispositivos(Deitel y Deitel, 2010).

3.- Cuadro comparativo de estos dos paradigmas(Joyanes Aguilar y Sánchez García, 2006)

|  |  |
| --- | --- |
| Paradigma Procedural | Paradigma Orientada a Objetos |
| • El programa completo tiene un diseño modular. | Abstracción (tipos abstractos de datos y clases), es la propiedad de los objetos que consiste en tener en cuenta sólo los aspectos más importantes desde un punto de vista determinado y no tener en cuenta los restantes aspectos. |
| • Los módulos se diseñan con metodología descendente (puede hacerse también ascendente). | • Encapsulado de datos y ocultación de datos, es el proceso de agrupar datos y operaciones relacionadas bajo la misma unidad de programación. |
| • Cada módulo se codifica utilizando las tres estructuras de control básicas: secuenciales, selectivas y repetitivas. | • Herencia, el mecanismo que implementa la propiedad de generalización. |
| • Estructuración y modularidad son conceptos complementarios (se solapan). | • Polimorfismo, o es aquella en que una operación tiene el mismo nombre en diferentes clases, pero se ejecuta de diferentes formas en cada clase. |

Lenguajes de Programación Orientada a Objetos

C++

Una clase

class CRender {

char buffer[256];

public:

void m\_Renderizar();

};

Herencia

class <clase\_derivada> :

[public|private] <base1> [,[public|private] <base2>] {};

Polimorfismo

|  |
| --- |
| virtual float earningsO const = 0; |

Encapsulamiento

class Fecha

{

public:

int anho; // El anho con cuatro cifras, ej. 2004

int mes; // El mes, de 1 a 12

int dia; // El dia, de 1 a 31

void metodoMaravilloso1();

void metodoMaravilloso2();

};

### Java

Una clase

[modificadores] class IdentificadorClase {

// Declaraciones de atributos y metodos

...

}

Herencia

public class Futbolista extends SeleccionFutbol

{

private int dorsal;

private String demarcacion;

public Futbolista() {

super();

}

Polimorfismo

Coche miCoche = new Coche("Ford Focus 2.0");

//la variable miCoche apunta a un objeto de la clase coche

//si lo deseo, mañana podrá apuntar a otro objeto diferente, pero siempre tendrá que ser de la clase Coche

miCoche = new Coche("Renault Megane 1.6");

Encapsulamiento

class Persona {

private String nombre;// <-- ya nadie lo puede ver

private int age;// <-- ya nadie lo puede ver

public int age() {

return this.age;

}

}

### Python

Una clase

class Clase:

<declaración-1>

.

<declaración-N>

Herencia

class AlumnoFIUBA(Persona):

"Clase que representa a un alumno de FIUBA."

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

"Constructor de AlumnoFIUBA"

# llamamos al constructor de Persona

Persona.\_\_init\_\_(self, identificacion, nombre, apellido)

# agregamos el nuevo atributo

self.padron = padron

Polimorfismo

def obtener\_versor(punto):

norma = punto.norma()

return punto \* (1.0 / norma)

Encapsulamiento

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

### C#

Una clase

public class Point

{

public int x, y;

public Point(int x, int y)

{

this.x = x;

this.y = y;

}

}

Herencia

using System;

public class A

{

private int value = 10;

public class B : A

{

public int GetValue()

{

return this.value;

}

}

}

Polimorfismo

using System;

using System.Collections.Generic;

public class Shape

{

// A few example members

public int X { get; private set; }

public int Y { get; private set; }

public int Height { get; set; }

public int Width { get; set; }

// Virtual method

public virtual void Draw()

{

Console.WriteLine("Performing base class drawing tasks");

}

}

class Circle : Shape

{

public override void Draw()

{

// Code to draw a circle...

Console.WriteLine("Drawing a circle");

base.Draw();

}

}

class Rectangle : Shape

{

public override void Draw()

{

// Code to draw a rectangle...

Console.WriteLine("Drawing a rectangle");

base.Draw();

}

}

Encapsulamiento

public string Nombre

{

//GET es el métodos que utilizaras para entregar tu nombre al desconocido

get { return nombre; }

//SET es el método que se utiliza para asignar un valor al atributo

//cuando naciste tu mamá y tu papá utilizarón este bloque para bautizarte

set { nombre = value; }

}

public string Apellidos

{

get { return apellidos; }

set { apellidos = value; }

}

public int Clave

{

get { return clave; }

set { clave = value; }

}

}

}

# Conclusiones

La forma en que se aborda cada paradigma a través de su respectivo compilador, este responde a necesidades específicas, por lo que cada paradigma ofrece una solución adecuada al problema. El paradigma procedural se centra en la reutilización de código a partir de módulos basados en algoritmos iterativos, mientras que con la programación orientada a objetos se centra en problemas complejos, con sus características pretenden mejorar el proceso de programación.

# Bibliografía

Azuela, M. F., y Joyanes Aguilar, L. (2001). *Java 2: manual de programación*. Mexico, D.F.: McGraw-Hill.

Deitel, H., y Deitel, P. (2010). *Visual C# 2010 How to Program*. Prentice Hall Press.

Galindo, J. M., y Pastó, V. J. (2015). *Introducción a la programación*. Cataluña.

Horton, I., y Weert, P. Van. (2018). *Beginning C ++ 17*. New York: Apress Media.

Hoyos, J. G., y Puertas, A. (2017). Desempeño de los paradigmas funcional y orientado a objetos en arquitecturas multicore: Caso de estudio haskell-Java. *Informacion Tecnologica*, *28*(5), 65-74. http://doi.org/10.4067/S0718-07642017000500009

Ivet, C.-P., Diaz, R., y Becerra García, R. (2014). El lenguaje de programación Python. *Ciencias holguín*, *XX*(2), 1-13.

Joyanes Aguilar, L., y Sánchez García, L. (2006). *Programación en C++: un enfoque práctico*. Mexico, D.F.: McGraw-Hill.

Juganaru Mathieu, M. (2014). *Introducción a la programación*. *Universitat Jaume I*. Mexico, D.F.: Grupo Editorial Patria.

Savitch, W. (2007). *Resolución de problemas con C++* (5ta ed.). Mexico, D.F.: Pearson educación.