

Universidad Autónoma de Ciudad Juárez

División Multidisciplinaria Ciudad Universitaria



Cuadro comparativo de lenguajes de Programación Orientada a Objetos

Programación II

Docente: Alan Ponce

Alumno: Villanueva Aguilar Carlos Raymundo. 169881

Licenciatura en Ingeniería de Software

08 de Marzo de 2019

Introducción

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

1.-Definición de lenguajes procedural

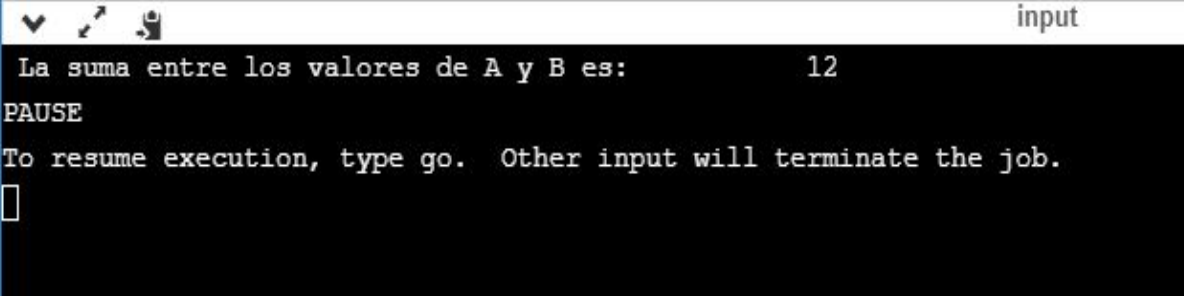
“Un programa en lenguaje procedimental es un conjunto de instrucciones o sentencias. En el caso de pequeños programas, estos principios de organización (denominados paradigma) se demuestran eficientes.” (Joyanes, 2007, p.29).

FORTRAN:

Creado en 1954, este lenguaje de programación, es el primer lenguaje de programación comercial de alto nivel. Joyanes (2008) afirma que fue concebido para resolver problemas científicos y de ingeniería. (p.40)

Ejemplo de Código:

```
Program EjemploFort    !Se inicia el programa con el nombre de EjemploFort
implicit none          !Se le indica al programa que las variables insertadas no son implícitas.
integer A,B,C           !Se declaran las variables con la palabra reservada integer.
A=5                     !Se le asigna un valor de 5 a A.
B=7                     !Se le asigna un valor de 5 a B.
C=A+B                   !Se le asigna a C el valor del resultado de la suma entre A y B.
Print *, "La suma entre los valores de A y B es: ", C           !La palabra reservada print imprime el
mensaje escrito en pantalla
pause                  !Se pone una pausa para que el programa se detenga antes de llegar al final,
donde se cierra automáticamente.
End Program EjemploFort !Se cierra el programa.
```

A screenshot of a terminal window titled 'input'. The window has a dark background with light-colored text. The text displayed is the output of the Fortran program: 'La suma entre los valores de A y B es: 12', followed by 'PAUSE' and a message 'To resume execution, type go. Other input will terminate the job.' with a cursor on the next line.

```
input
La suma entre los valores de A y B es: 12
PAUSE
To resume execution, type go. Other input will terminate the job.
█
```

ALGOL:

Este lenguaje de programación fue creado en la década de los sesentas, y según Balderas (2001), nunca llegó a ser ampliamente utilizado, sin embargo, es considerado como el antecesor de la mayoría de los lenguajes de programación que se emplean hoy en día. (p.4).

Ejemplo de Código:

```
BEGIN
print("Este es un ejemplo de programa en ALGOL.")
END
```

En este ejemplo se usan las palabras reservadas BEGIN y END para indicar dónde inicia y termina el programa, además de que para imprimir en pantalla aquí se usa la palabra print, y no es necesario poner un punto y coma al final del comando.

COBOL:

“El lenguaje por excelencia del mundo de la gestión y de los negocios hasta hace muy poco tiempo” (Joyanes, 2008, p.42).

Ejemplo de Código:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PRUEBA-COBOL.
DATA DIVISION.
WORKING-STORAGE SECTION.
77 A PIC X(30).
PROCEDURE DIVISION.
DISPLAY 'Inserta un numero'.
ACCEPT A.
DISPLAY 'El numero insertado es: '
DISPLAY A.
STOP RUN.
```

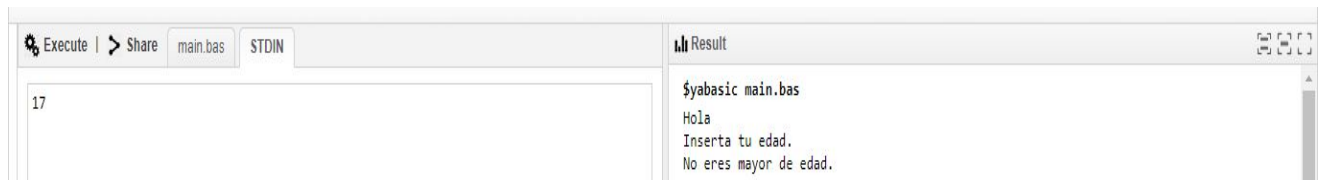
En COBOL el código se escribe en divisiones, como la de identificación, la de datos o la de procedimientos. En este programa la división de identificación es usada para darle un nombre al programa, la de datos es para declarar variables, como la variable A en el caso de este programa, y en la de procedimientos es donde se escribe el código principal. En COBOL se utiliza “DISPLAY” para imprimir en pantalla y “ACCEPT” es utilizado para la entrada de datos, además, el equivalente al punto y coma en este lenguaje es el del punto.

BASIC:

Fue creado en 1964 por John George Kemeny y Thomas Eugene Kurtz.

Ejemplo de Código:

```
PRINT "Hola"
INPUT "Inserta tu edad."
" E$
IF E<18 THEN
PRINT "No eres mayor de edad."
END IF
IF E>=18 THEN
Print"Eres mayor de edad."
END IF
```



Este código te pide que insertes tu edad, y dependiendo de ésta el programa indicará si eres o no mayor de edad. Hay que tener en cuenta que cada if debe de cerrarse con un end if. Para realizar saltos de línea al imprimir texto se puede simplemente oprimir enter para ir a la siguiente línea, siempre y cuando el salto se de dentro de las comillas.

Pascal:

“Aunque ya está muy en desuso en la enseñanza de la programación el lenguaje Pascal, su sintaxis y estructura sigue siendo un modelo excelente de aprendizaje” (Joyanes, 2008, p. xxvi).

Ejemplo de Código:

```
program EjemploPascal;
var A,B:integer;
begin
  writeln ('Inserta el valor de la base del rectangulo');
  readln(B);
  writeln ('Inserta el valor de la altura del rectangulo');
  readln(A);
  writeln ('El area del rectangulo es: ',(A*B));
end.
```



En este programa se implementa una aplicación que lee dos variables, calcula el resultado y lo imprime en pantalla. En pascal se utiliza writeln para imprimir en pantalla y readln para la entrada de datos.

C:

Fue diseñado en 1972, Bardenas (2001), afirma lo siguiente: “Aunque el lenguaje C es considerado un lenguaje de alto nivel, muchos autores consideran que es de nivel medio, pues es posible trabajar con él como si se tratara de un ensamblador” (p.5).

Ejemplo de Código:

```
#include <stdio.h> //Se incluye el archivo de cabecera que contiene las funciones estándar de C.

int main()
{
int a,b; //Se declaran dos variables de tipo int.
//Se escriben dos instrucciones donde se pide al usuario insertar un número para a y para b.
    printf("Inserta el primer numero: ");
    scanf("%i",&a);
    printf("Inserta el segundo numero: ");
    scanf("%i",&b);
if (a>b){//Si a es mayor que b, se imprimirá un texto en pantalla diciendo que lo es.
    printf("El mayor de los dos numeros es: %i",a);
}
else if (b>a){//Si b es mayor que a, se realiza lo mismo, pero indicando que b es mayor.
    printf("El mayor de los dos numeros es: %i",b);
}
else{//Si a y b son iguales, se imprime un texto que indica que lo son.
    printf("Los dos numeros son iguales");
}
    return 0;
}
```

2.- Definición de lenguaje orientada a objetos

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

C++

El lenguaje C++ fue desarrollado a principios de los años ochenta por Bjarne Stroustrup, siendo diseñado como un mejor C. Joyanes (2000) menciona que:

“Las características más notables que han ido incorporándose a C++ son: herencia múltiple, genericidad, plantillas, funciones virtuales, excepciones, etc. C++ ha ido evolucionando año

a año y como su autor ha explicado: “evolucionó siempre para resolver problemas encontrados por los usuarios y como consecuencia de conversaciones entre el autor, sus amigos y sus colegas.” (p.40).

Java

En 1991, un pequeño grupo de ingenieros solares llamado “Green Team” creía que la siguiente ola en la computación sería la unión de aparatos de consumidores digitales y computadoras. Liderados por James Gosling, el equipo trabajó contrarreloj y creó el lenguaje Java. Barnes (2007) menciona lo siguiente:

“En los últimos años, Java ha sido ampliamente usado en la enseñanza de programación y esto se debe a varios motivos. Uno de ellos es que Java tiene muchas características que facilitan la enseñanza: tiene una definición relativamente clara, el compilador realiza extensos análisis estadísticos fáciles de enseñar y tiene un modelo de memoria muy robusto que elimina la mayoría de los errores “misteriosos” que surgen cuando se comprometen los límites de los objetos o el sistema de tipos. Otro motivo es que Java se ha vuelto muy importante comercialmente.” (p.xix).

Python

Pérez (2014) expone los siguientes puntos acerca de la historia de python:

- Python fue creado por Guido van Rossum, un programador holandés a finales de los 80 y principio de los 90 cuando se encontraba trabajando en el sistema operativo Amoeba.
- El 16 de octubre del 2000 se lanza Python 2.0 que contenía nuevas características como completa recolección de basura y completo soporte a Unicode.
- A Guido van Rossum le fue otorgado el Free Software Award (Premio del Software Libre) en el 2001, por sus trabajos en la creación y desarrollo del lenguaje Python.
- En el 2005 fue contratado por Google, donde trabaja en la actualidad, aunque sigue liderando los esfuerzos en el desarrollo del Python.

C#

- Fue desarrollado por un equipo de Microsoft liderado por Anders Hejlsberg.
- El proceso de desarrollo inició en enero de 1999.
- La primera versión de C# fue publicada en el año 2000 como una parte integral del marco de referencia .NET de Microsoft.

3.- Cuadro comparativo de estos dos paradigmas

Paradigma Procedural	Paradigma Orientada a Objetos
*Los atributos y comportamientos normalmente están separados.	*Los atributos y comportamientos son almacenados dentro de un objeto.
*Separa los datos del sistema de los	*Los datos y las operaciones que

operadores que los manipulan.	manipulan los datos están encapsulados en el objeto.
*Se enfatiza en los algoritmos.	*Se enfatiza en los datos.
*Intenta ajustar el problema al enfoque procedimental.	*Intenta ajustar el lenguaje al problema
*Útil para programas que realizan tareas sencillas.	*Útil para programas más complejos.
*No corresponde con los modelos de las cosas del mundo real.	*Los objetos se comportan como modelos del mundo real.

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 como implementar:

Una clase. Código hecho por: Koffman (2006, Pp.76-77)

```

/** La clase reloj representa la hora del día. */
class Clock { public: // La interfaz se define aquí.
/** Ajusta el reloj. */
void set_clock(int hr, int min, int sec);
/** Consigue la hora actual. */
int get_hours() const;
/** Consigue el minuto actual. */
int get_minutes() const;
/** Consigue el segundo actual */
int get_seconds() const;
/** Hace que el reloj avance un segundo. */
void tick();
private:
// Aquí se definen los detalles de implementación
};

```

Herencia (Fuente del código en la bibliografía)

```
#include <iostream>
```

```

using namespace std;

class Vehiculo { //Clase base
public:
    void avanza() {}
};

class Coche : public Vehiculo { //Clase derivada Coche
public:
    void avanza(void) //Se crea el método avanza para la clase Coche
    { cout << "Avanza coche." << endl; } //Imprime el mensaje en pantalla
};

class Moto: public Vehiculo { //Clase derivada Moto
public:
    void avanza(void) //Se crea el método avanza para la clase Moto
    { cout << "Avanza moto." << endl; } //Imprime el mensaje en pantalla
};

int main()
{
    Moto t; //Se declara una variable de tipo Moto
    Coche c; //Se declara una variable de tipo Coche

    t.avanza(); /*La variable tipo Moto realizará el método avanza, el cual imprime un
mensaje en pantalla*/
    c.avanza(); /*La variable tipo Coche realizará el método avanza, el cual imprime un
mensaje en pantalla*/

    return 0;
}

```

Polimorfismo (Fuente del código en la bibliografía)

```

#include <iostream>
#include <string>
using namespace std;
class Mascota
{
public:
    Mascota(string nombre, int patas); /*Se declaran las variables nombre y patas como
públicas*/
    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 /*Se declara la clase perro como una subclase de
Mascota*/
{
public:
    Perro(string nombre);
    string hablar(); //Se define al método hablar como un string
};

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

string Perro::hablar()
{
    return "GUAU"; //Se define que al llamar al método, éste va a imprimir un texto.
}

class Gato : public Mascota /*Se define la clase gato como una subclase de
Mascota.*/
{
public:
    Gato(string nombre);

    string hablar(); //Se define al método hablar como un string
};

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

string Gato::hablar()
{
    return "MIAU"; //Se define que al llamar al método, éste va a imprimir un texto.
}
/*Dependiendo de que si la clase que llama al método main es perro o gato, se
imprime en pantalla un mensaje diferente para cada uno.*/

```

Encapsulamiento (Fuente del código en la bibliografía)

```

class TObjGraf {

```

```

public:
    int    X;        // Aquí se declaran las propiedades
    int    Y;
    TColor  Color;
    TPaintBox *PaintBox;

    void Mostrar (void); // Aquí se declaran los métodos
};

```

Java

Incluir código de como implementar:

Una clase (Fuente del código en la bibliografía)

```

class Vehiculo{//Se declara la clase vehículo
    int pasajeros; //número de pasajeros
    int capacidad; //capacidad en galones
    int mpg;       //consumo de combustible en millas por galón
}

```

Herencia (Fuente del código en la bibliografía)

//Clase para objetos de dos dimensiones

```

class DosDimensiones{
    double base;
    double altura;

    void mostrarDimension(){
        System.out.println("La base y altura es: "+base+" y "+altura);
    }
}

```

//Una subclase de DosDimensiones para Triangulo

```

class Triangulo extends DosDimensiones{//Para declarar subclases se utiliza "extends".
    String estilo;

    double area(){
        return base*altura/2;
    }
}

```

```

void mostrarEstilo(){
    System.out.println("Triangulo es: "+estilo);
}
}

class Lados3{
    public static void main(String[] args) {

        //Se declaran dos variables de tipo Triangulo.
        Triangulo t1=new Triangulo();
        Triangulo t2=new Triangulo();

        //Se le asignan valores a los atributos de base, altura y estilo a t1.
        t1.base=4.0;
        t1.altura=4.0;
        t1.estilo="Estilo 1";
        //Se le asignan valores a los atributos de base, altura y estilo a t2.
        t2.base=8.0;
        t2.altura=12.0;
        t2.estilo="Estilo 2";

        System.out.println("Información para T1: ");
        t1.mostrarEstilo();
        t1.mostrarDimension();
        System.out.println("Su área es: "+t1.area());

        System.out.println();

        System.out.println("Información para T2: ");
        t2.mostrarEstilo();
        t2.mostrarDimension();
        System.out.println("Su área es: "+t2.area());

    }
}

```

Polimorfismo (Fuente del Código en la bibliografía)

```

class Doctor {
    public void treatPatient () {
        // método treatPatient
    }
}
class Cirujano extends Doctor {
    public void treatPatient () {
        // método treatPatient
    }
}
class Hospital {
    public static void main (String args []) {
        Doctor doctorObj = new Doctor ()
// El método treatPatient en la clase Doctor se ejecutará
        doctorObj.treatPatient ();

        Cirujano cirujanoObj = new cirujano ();
        // El método treatPatient en la clase Surgeon se ejecutará
        surgeonObj.treatPatient ();
    }
}

```

El método treatPatient debe de ocasionar una salida diferente para la clase Doctor y la clase Cirujano, el programa no especifica una salida pero como ejemplo podemos decir que el tratamiento del doctor es administrar una medicina, y el del cirujano es operar al paciente.

Encapsulamiento (Fuente del Código en la bibliografía)

```

public class Persona{
    //Atributos
    private int altura; //Se declara la variable altura como privada para encapsularla.
    //Métodos
    public int getAltura(){/*El getter va a ser el que acceda a los atributos en forma de sólo
lectura.*/
        return this.Altura;
    }
    public void setAltura(int unaAltura){/*El setter accede a los atributos en forma de
escritura, este establece el valor de un atributo.*/
        this.altura = unaAltura;
    }}

```

Python

Incluir código de como implementar:

Una clase (Fuente del Código en la bibliografía)

```

class Person:

    def __init__(self, n, s):

        self.name = n

        self.school = s

    def print_name(self):

        print self.name

    def print_school(self):

        print self.school

jorge = Person('Jorge', 'Universidad de la vida')

jorge.print_name()

jorge.print_school()

```

Herencia (Fuente del Código en la bibliografía)

class Vehiculo:

```

def __init__(self, nombre, color):
    self.__nombre = nombre      # __name es privado
    self.__color = color

def getColor(self):             # getColor() funcion accesible a la clase Auto
    return self.__color

def setColor(self, color):      # setColor es accesible fuera de la clase
    self.__color = color

def getNombre(self):            # getNombre() es accesible fuera de la clase
    return self.__nombre

```

class Auto(Vehiculo):

```

def __init__(self, nombre, color, modelo):
    # Llamada al constructor para establecer nombre y color
    super().__init__(nombre, color)
    self.__modelo = modelo

def getDescripcion(self):

```

```

        return self.getNombre() + self.__modelo + " de color " + self.getColor()

# En el método getDescripcion podemos llamar a getNombre() y getColor porque
# son accesibles al pasar la herencia a la clase Derivada (Auto)

c = Auto("Ford Mustang", "rojo", "GT350")
print(c.getDescripcion())
print(c.getNombre()) # La clase auto no tiene un método getNombre() pero es accesible de
la clase heredada Vehiculo

```

Polimorfismo (Fuente del Código en la bibliografía)

```

class A():

    def __init__(self):
        self.__x = 1

    def m1(self):
        print("m1 de A") #Se define como el comportamiento de A ante self

class B(A):

    def __init__(self):
        self.__y = 1

    def m1(self):
        print("m1 de B") #Se define como el comportamiento de B ante self

c = B()
c.m1()

```

Encapsulamiento (Fuente del Código en la bibliografía)

```

class CajaDeSeguridad:
    """Clase que incluye un atributo "escondido."""
    __contraclave = "123qwe"

    def seguro(self, clave):
        if self.__contraclave == clave:
            print("Acceso concedido.")
        else:
            print("Acceso denegado.")

```

C#

Incluir código de cómo implementar:

Una clase (Fuente del Código en la bibliografía)

//La manera en la que se declara una clase es muy similar a java

```
public class Customer {  
    // Aquí van los campos, propiedades, métodos y eventos  
}
```

Herencia (Fuente del Código en la bibliografía)

```
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());  
    }  
}
```

Polimorfismo (Fuente del Código en la bibliografía)

```
using System;
```

```
using System.Collections.Generic;
```

```
public class Shape  
{  
    // Unos cuantos miembros de ejemplo
```

```

    public int X { get; private set; }
    public int Y { get; private set; }
    public int Height { get; set; }
    public int Width { get; set; }

    // Método virtual
    public virtual void Draw()
    {
        Console.WriteLine("Performing base class drawing tasks");
    }
}

class Circle : Shape
{
    public override void Draw()
    {
        // Código para dibujar un círculo
        Console.WriteLine("Drawing a circle");
        base.Draw();
    }
}

class Rectangle : Shape
{
    public override void Draw()
    {
        // Código para dibujar un rectángulo
        Console.WriteLine("Drawing a rectangle");
        base.Draw();
    }
}

class Triangle : Shape
{
    public override void Draw()
    {
        // Código para dibujar un triángulo
        Console.WriteLine("Drawing a triangle");
        base.Draw();
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Un rectángulo, triángulo y círculo pueden ser usados
        // donde sea que se espere una clase. No se requiere un casteo
    }
}

```



```

// porque una conversión implícita existe desde una clase
// derivada a su clase base.
    var shapes = new List<Shape>
    {
        new Rectangle(),
        new Triangle(),
        new Circle()
    };

// El método visual "Draw" es invocado en cada una de
// clases derivadas, no en la clase base.
foreach (var shape in shapes)
{
    shape.Draw();
}

// Mantiene la consola abierta en modo de depuración.
Console.WriteLine("Press any key to exit.");
Console.ReadKey();
}
}

```

Encapsulamiento (Fuente del Código en la bibliografía)

```

namespace Encapsulamiento
{
    class Persona
    {
        //Primero se definen los atributos
        private string nombre;
        private string apellidos;
        private int clave;

        //Los métodos de encapsulamiento NO pueden llamarse igual
        //que los atributos, por eso la diferencia está en la mayúscula
        public string Nombre
        {
            //GET es el método que se utiliza para capturar un valor.
            get { return nombre; }
            //SET es el método que se utiliza para asignar un valor al atributo.
            set { nombre = value; }
        }

        public string Apellidos
        {
            get { return apellidos; }
            set { apellidos = value; }
        }
    }
}

```

```
}
```

```
public int Clave  
{  
  get { return clave; }  
  set { clave = value; }  
}  
}  
}
```

Conclusiones

Es importante conocer la sintaxis y el funcionamiento de distintos lenguajes de programación, pues si se tiene el conocimiento base de conceptos como la herencia o el polimorfismo en el caso de los lenguajes orientados a objetos, lo único que queda es saber la manera en la que se implementan en cada lenguaje. También es importante reconocer los puntos fuertes de cada paradigma de programación y tener una buena cantidad de conocimiento de cada uno, puesto que aunque haya gente que piense que uno es completamente superior al otro, la verdad es que lo mejor es saber cuando es más útil el implementar uno sobre otro para ahorrar tiempo y código. Por ejemplo, puede que varias herramientas sirvan ocasionalmente para aflojar un tornillo aunque no hayan sido creadas para tal tarea, pero es una pérdida de tiempo y esfuerzo realizar la tarea con estas herramientas en vez de usar un destornillador, el cual se especializa en ese tipo de tareas.

Pude notar que las clases, la herencia, el polimorfismo y la encapsulación en cada uno de los distintos lenguajes orientados a objetos comparten similitudes en la manera en la que se realizan, también noté que la sintaxis de C# se asemeja a Java y a C++, parece como si este lenguaje hubiera tomado a los otros dos como referencia.

Personalmente, el realizar esta investigación me ayudó a comprender mejor a las bases de la POO, ya contaba con conocimiento acerca de cada concepto, pero ver cómo se implementa el código en la práctica me hizo entender de una manera más eficaz estos conceptos.

Bibliografía:

Luis Joyanes Aguilar. (2008). FUNDAMENTOS DE PROGRAMACIÓN. Algoritmos, estructura de datos y objetos. Madrid: McGraw-Hill.

Carpio, J. B. (2001). Curso práctico de programación en C y C++ (No. 8). Universitat Jaume I.

Joyanes Aguilar, L., Sánchez García, L., & Zahonero Martínez, I. (2007). Estructura de datos en C++. McGraw-Hill.

Joyanes, A. (2000). Programación en C++. Algoritmos, estructuras de datos y objetos.

Barnes, D. J., Kölling, M., & Brenta, B. I. (2007). *Programación orientada a objetos con Java*. Pearson Educación.

Historia breve de java:

<https://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html>

Línea del tiempo de java: <http://oracle.com.edgesuite.net/timeline/java/>

Challenger-Pérez, I., Díaz-Ricardo, Y., & Becerra-García, R. A. (2014). El lenguaje de programación Python. *Ciencias Holguín*, 20(2).

Historia de C#: <https://bytescout.com/blog/2016/02/c-brief-history-of-microsofts-premier.html>

Weisfeld, M. (2008). *The object-oriented thought process*. Pearson Education.

Koffman, E. B., & Wolfgang, P. A. (2006). *Objects, abstraction, data structures and design using C++*. John Wiley & Sons.

Ejemplo de herencia en C++:

http://jbgarcia.webs.uvigo.es/assignaturas/TO/cursilloCpp/15_herencia_simple.html

Ejemplo de polimorfismo en C++:

<https://poesiabinaria.net/2011/11/polimorfismos-enrevesando-la-herencia-entre-clases-c/>

Ejemplo de encapsulamiento en C++: <https://elvex.ugr.es/decsai/builder/intro/5.html>

Ejemplo de clase en Java: <https://javadesdecero.es/poo/que-es-una-clase-ejemplos/>

Ejemplo de herencia en Java: <https://javadesdecero.es/poo/herencia-java-tipos-ejemplos/>

Ejemplo de polimorfismo en Java: <https://guru99.es/java-inheritance-polymorphism/>

Ejemplo de encapsulamiento en Java:

<http://labojava.blogspot.com/2012/05/introduccion-oopencapsulamiento.html>

Ejemplo de clase en Python:

https://programacion.net/articulo/como_funcionan_las_clases_y_objetos_en_python_1505

Ejemplos de herencia y polimorfismo en Python:

<http://www.pythondiario.com/2016/10/herencia-y-polimorfismo-en-python.html>

Ejemplo de encapsulamiento en Python:

<https://pythonista.io/cursos/py111/interfaces-implementaciones-y-encapsulamiento>

Ejemplo de clase en C#:

<https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/classes>

Ejemplo de herencia en C#:

<https://docs.microsoft.com/es-es/dotnet/csharp/tutorials/inheritance>

Ejemplo de polimorfismo en C#:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/poly-morphism>

Ejemplo de encapsulamiento en C#:

<http://undiaparahablar.blogspot.com/2010/02/polimorfismo.html>