## Universidad Autónoma de Ciudad Juárez

## División Multidisciplinaria Ciudad Universitaria

## 

## Cuadro comparativo de lenguajes de Programación Orientada a Objetos

## Programación II

## Docente: Alan Ponce

## 

## 

## Alumno Luis Ivan Valdez Anchondo 159949

## Licenciatura en Ingeniería de Software

## 08 de Marzo de 2019

# Introducción

El objetivo de este trabajo es definir los lenguajes orientados a objetos y procedural, ver sus características y sus diferencias, así como también ejemplos de códigos en varios lenguajes para ver como cambia la sintaxis en cada uno de ellos para ver cómo se implementa una clase, cómo implementan la herencia, el polimorfismo y el encapsulamiento.

**Programación procedural**

La programación procedural es un estilo que se basa en estructurar el código en componentes como procedimientos, subrutinas o funciones. En el caso de que esta técnica se aplique en lenguajes de alto nivel, recibirá el nombre de programación funcional. Esta técnica consiste en basarse de un número muy bajo de expresiones repetidas, englobadas todas en un procedimiento o función y llamarlo cada vez que tenga que ejecutarse.

Esta técnica de programación ofrece muy buena respuesta con relación al tamaño de los programas, y en bruto casi no se nota en la velocidad de ejecución de los mismos,aunque es muy complicado conseguir una programación por procedimientos pura. Algunos lenguajes de programación procedural son:

* **C.-** es un lenguaje orientado a la implementación de Sistemas operativos, concretamente Unix. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

*/\** *ejemplo.c* *(c) Envite, 2004* *para el wikilibro "Programación en C (fundamentos)"* *bajo licencia FDL* *\*/*

#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 multiplicación: %i**\n**",resultado);

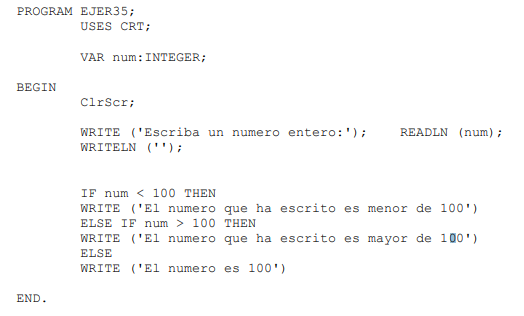
resultado=5/2; */\*Realizamos una división entera.\*/*

printf("Resultado de la división:%i**\n**",resultado);

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

}

* **Pascal:** se caracteriza por ser un lenguaje de programación estructurado fuertemente tipado. Esto implica que el código está dividido en porciones que son más fáciles de leer a las cuales llamamos procedimientos o funciones . De esta forma Pascal facilita la utilización de la programación estructurada en oposición al antiguo estilo de programación monolítica. El tipo de dato de todas las variables debe ser declarado previamente para que su uso quede habilitado.



**Programación orientada a objetos**

La programación orientada a objetos es una forma de programar que se acerca a cómo expresamos las cosas en la vida real a diferencia de otros tipos de programación. La POO nos hace pensar las cosas de manera distinta con el fin de escribir nuestros programas en términos de objetos, propiedades, métodos, entre otras cosas. Está basada en varias técnicas como la herencia, el polimorfismo, el encapsulamiento y la composición.

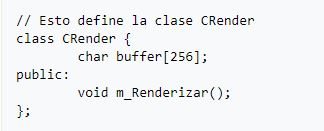
* **C++.-** Es un lenguaje de programación creado con la intención de extender al lenguaje de programación C, con mecanismos que permiten la manipulación de objetos, desde el punto de vista de un lenguaje orientado a objetos C++ es un lenguaje híbrido. Este lenguaje tiene su origen por finales de los años 70’s, fue diseñado aproximadamente en 1979 por Bjarne Stroustrup quien se inspiró en el lenguaje Simula que era utilizado para hacer simulaciones y se considera como el primer lenguaje en permitir la programación orientada a objetos, Bjarne se dio cuenta de que la funcionalidad de Simula era muy útil para desarrollar software pero era muy lento para un uso práctico. Bjarne en un principio llamó su lenguaje “C with classes” y su principal objetivo era agregar la POO al lenguaje C. En el año 1983 el nombre del lenguaje fue cambiado a “C++” ya que él veía a su lenguaje como una mejora de C y el ++ es un incrementador de variable.
* **Java.-** Es un lenguaje de programación orientado a objetos que fue creado por Sun MicroSystems a principios de los 90’s. En el 1995 se creó la empresa javasoft que era para dedicarse al desarrollo de productos basados en tecnología java, y así ellos trabajaban con terceros para desarrollar aplicaciones, herramientas, sistemas de plataforma y servicios para de esa manera aumentar la capacidad del lenguaje. El punto principal para el éxito de java es la portabilidad que tiene, ya que en java es posible escribir programas que se ejecuten de manera satisfactoria en varias plataformas de diferentes arquitecturas gracias a la JVM.
* **Python.-** Es un lenguaje cuya filosofía es hacer que la sintaxis del código sea legible, Python fue creado por Guido Van Rossum a finales de los 80’s, esto paso en una navidad de 1989 cuando Guido trabajaba en el CWI que es un centro de investigación holandés, el comenzó a desarrollar su proyecto como un pasatiempo queriendole dar continuidad a el lenguaje ABC que fue desarrollado en el CWI como una alternativa para BASIC. En 1991 Guido publicó el código de la versión 0.9.0 en alt.sources, en esta versión ya se tenían disponibles clases con herencias, manejo de excepciones, funciones y tipos modulares. La última gran actualización de Python fue en 2008 con Python 3.0 que fue diseñado para rectificar fallas fundamentales en el diseño del lenguaje.
* **C#.-** Es un lenguaje de programación orientado a objetos que fue desarrollado y estandarizado por microsoft en abril de 1999 por un equipo formado por Anders Hejlsberg. C# es un híbrido que fusiona C++ y Java, principalmente en la capacidad de combinar operadores de C++ pero sin incorporar la herencia múltiple y la plena orientación a objetos de Java.

|  |  |
| --- | --- |
| Paradigma Procedural | Paradigma Orientada a Objetos |
| Está basado en funciones | Está basado en objetos |
| El programa se divide en muchas funciones | El programa simula aspectos del mundo real |
| Tiene funciones con datos locales | Tiene objetos con propiedades y procedimientos |
| Puede llamar a nuevas funciones y procedimientos | Puede llamar a los métodos de los objetos |
| Favorece a la reutilización | Es totalmente reutilizable, utiliza técnicas como la herencia. |
| Variables globales sin protección | Tiene encapsulamiento, que oculta la complejidad interna y los detalles de los datos y las operaciones. |
| La modularidad depende de la función contenedora | Tiene una mayor modularidad |

# Lenguajes de Programación Orientada a Objetos

### **C++**

**Una clase**



**Herencia**

// Clase base Persona:

**class** Persona {

**public**:

Persona(**char** \*n, **int** e);

**const** **char** \*LeerNombre(**char** \*n) **const**;

**int** LeerEdad() **const**;

**void** CambiarNombre(**const** **char** \*n);

**void** CambiarEdad(**int** e);

**protected**:

**char** nombre[40];

**int** edad;

};

// Clase derivada Empleado:

**class** Empleado : **public** Persona {

**public**:

Empleado(**char** \*n, **int** e, **float** s);

**float** LeerSalario() **const**;

**void** CambiarSalario(**const** **float** s);

**protected**:

**float** salarioAnual;

};

**Polimorfismo**

#include*<iostream>*

#include *<math.h>*

using namespace std;

class Figura {

private:

float base;

float altura;

public:

void captura();

virtual unsigned float perimetro()=0;

virtual unsigned float area()=0;

};

class Rectangulo: public Figura {

public:

void imprime();

unsigned float perimetro(){**return** 2\*(base+altura);}

unsigned float area(){**return** base\*altura;}

};

class Triangulo: public Figura {

public:

void muestra();

unsigned float perimetro(){**return** 2\*sqrt(altura^2+(base/2)^2)+base;} *//Usando pitágoras*

unsigned float area(){**return** (base\*altura)/2;}

};

void Figura::captura()

{

cout << "CALCULO DEL AREA Y PERIMETRO DE UN TRIANGULO ISÓSCELES Y UN RECTANGULO:" << endl;

cout << "escribe la altura: ";

cin >> altura;

cout << "escribe la base: ";

cin >> base;

cout << "EL PERIMETRO ES: " << perimetro();

cout << "EL AREA ES: " << area();

getchar();

**return** 0;

}

**Encapsulamiento**

**#include <iostream>**

**#include <locale>**

**using namespace std;**

**const double PI = 3.141592;**

**class Figura{**

**protected:**

**float x;**

**public:**

**Figura(float cx=0){**

**x = cx;**

**}**

**virtual float Perimetro() = 0;**

**virtual float Area() = 0;**

**};**

***// Clases derivadas***

**class Circulo: public Figura{**

**public:**

**Circulo(float radio){**

**x = radio;**

**}**

**float Perimetro(){**

**return 2\* PI \* x;**

**}**

**float Area(){**

**return PI \* x \* x;**

**}**

**};**

**class Cuadrado: public Figura**

**{**

**public:**

**Cuadrado(float lado){**

**x = lado;**

**}**

**float Perimetro(){**

**return 4 \* x;**

**}**

**float Area(){**

**return x \* x;**

**}**

**};**

**int main(){**

**setlocale(LC\_ALL, "");**

**float l, r;**

**cout << "Entre el lado del cuadrado" << endl;**

**cin >> l;**

**cout << "Entre el radio del círculo" << endl;**

**cin >> r;**

**Cuadrado cuad1(l);**

**Circulo Circ1(r);**

**cout << "El perimetro del circulo es:" << Circ1.Perimetro() << endl;**

**cout << "El área del circulo es:" << Circ1.Area() << endl;**

**cout << "El perimetro del cuadrado es:" << cuad1.Perimetro() << endl;**

**cout << "El area del cuadrado es:" << cuad1.Area() << endl;**

**}**

### **Java**

**Una clase**

**//Le damos un nombre "MiClase" a la clase**

**public class MiClase**

**{**

**//Atributos de la clase**

**private String atributo1;**

**private int atributo 2;**

**private float atributo 3;**

**//Constructor con el mismo nombre de la clase**

**public MiClase(){}**

**//Métodos de la clase**

**public void metodo1()**

**{**

**//Método vacio**

**}**

**public String metodo2()**

**{**

**return "metodo2";**

**}**

**}**

**Herencia**

**//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{**

**String estilo;**

**double area(){**

**return base\*altura/2;**

**}**

**void mostrarEstilo(){**

**System.out.println("Triangulo es: "+estilo);**

**}**

**}**

**class Lados3{**

**public static void main(String[] args) {**

**Triangulo t1=new Triangulo();**

**Triangulo t2=new Triangulo();**

**t1.base=4.0;**

**t1.altura=4.0;**

**t1.estilo="Estilo 1";**

**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**

**public class Polimorfismo {**

**public static void main(String[] args) {**

***// Creamos una variable del tipo MiClaseA, que sera un array de 3 elementos***

**MiClaseA [] misClases=new MiClaseA[3];**

**misClases[0]=new MiClaseA("Esther");**

***// Asignamos a la variable misClases que son del tipo MiClaseA un objeto del***

***// tipo MiClaseB, ya que hereda de MiClaseA***

**misClases[1]=new MiClaseB("Juan", "Azul");**

**misClases[2]=new MiClaseA("Rosa");**

**for(MiClaseA e:misClases) {**

***// ejecutara la función info() de la clase que haya sido instanciada.***

***// Esto se llama polimorfismo***

**System.out.println(e.info());**

**}**

**}**

**}**

**class MiClaseA {**

**private String name;**

***// constructor***

**public MiClaseA(String name) {**

**this.name=name;**

**}**

**public String info() {**

***// Devolvemos el nombre***

**return this.name;**

**}**

**}**

**class MiClaseB extends MiClaseA {**

**private String color;**

***// constructor***

**public MiClaseB(String name, String color) {**

***// ejecutamos el constructor de la superclase (MiClaseA) enviandole el nombre***

**super(name);**

***// guardamos el nombre del color en la variable color de MiClaseB***

**this.color=color;**

**}**

**public String info() {**

***// devolvemos el contenido de MiClaseA.info() mas la variable color***

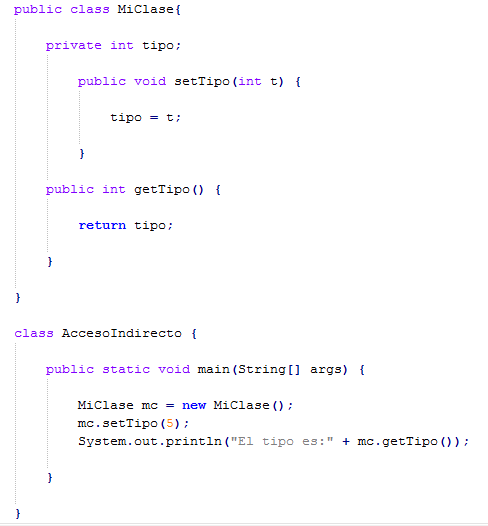
***// de la clase MiClaseB***

**return super.info() + " (" + this.color + ")";**

**}**

**}**

**Encapsulamiento**

****

### **Python**

**Una clase**

**class Objeto():**

**color = "verde"**

**tamanio = "grande"**

**aspecto = "feo"**

**antenas = Antena()**

**ojos = Ojo()**

**pelos = Pelo()**

**def flotar(self):**

**pass**

**Herencia**

**class Persona:**

***'''Clase base para creación de datos personales.'''***

**def \_\_init\_\_(self):**

***'''Genera una clave única a partir de una estampa de tiempo y la relaciona con el atributo \_\_clave.'''***

**from time import time**

**self.\_\_clave = str(int(time() / 0.017))[1:]**

**@property**

**def clave(self):**

***'''Regresa el valor del atributo "escondido" \_\_clave.'''***

**return self.\_\_clave**

**@property**

**def nombre(self):**

***'''Regresa una cadena de caracteres a partir de la lista contenida en lista\_nombre.'''***

**return " ".join(self.lista\_nombre)**

**@nombre.setter**

**def nombre(self, nombre):**

***'''Debe ingresarse una lista o tupla con entre 2 y 3 elementos.'''***

**if len(nombre) < 2 or len(nombre) > 3 or type(nombre) not in (list, tuple):**

**raise ValueError("Formato incorrecto.")**

**else:**

**self.lista\_nombre = nombre**

**class Estudiante(Persona):**

***'''Clase que hereda a Persona.'''***

**tira\_de\_materias = []**

**def inscripcion(self, materia):**

***'''Añade elementos a tira\_de\_materias.'''***

**self.tira\_de\_materias.append(materia)**

**bc2186 = Estudiante()**

**bc2186.clave**

**Polimorfismo**

**#!/usr/bin/python**

**# Nombre de Fichero : polimorfismo.py**

**class Gato:**

**"Clsase Gato"**

**def \_\_init\_\_(self):**

**self.pos = 0**

**def trepar(self):**

**pass**

**def caminar(self):**

**self.pos = self.pos + 10**

**def maullar(self):**

**print "Miu Miu"**

**class Perro:**

**"Clase Perro"**

**def \_\_init\_\_(self):**

**self.pos = 0**

**def caminar(self):**

**self.pos = self.pos +4**

**def ladrar(self):**

**print "Gua Gua"**

**def pasearMascota(mascota):**

**for i in range(5):**

**mascota.caminar()**

**print "La mascota quedo en " + str(mascota.pos)**

**perro=Perro()**

**gato=Gato()**

**pasearMascota(perro)**

**pasearMascota(gato)**

**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#**

**Una clase**

**//[access modifier] - [class] - [identifier]**

**public class Customer**

**{**

**// Fields, properties, methods and events go here...**

**}**

**Herencia**

**using System;**

**public class Person**

**{**

**// Constructor that takes no arguments:**

**public Person()**

**{**

**Name = "unknown";**

**}**

**// Constructor that takes one argument:**

**public Person(string name)**

**{**

**Name = name;**

**}**

**// Auto-implemented readonly property:**

**public string Name { get; }**

**// Method that overrides the base class (System.Object) implementation.**

**public override string ToString()**

**{**

**return Name;**

**}**

**}**

**class TestPerson**

**{**

**static void Main()**

**{**

**// Call the constructor that has no parameters.**

**var person1 = new Person();**

**Console.WriteLine(person1.Name);**

**// Call the constructor that has one parameter.**

**var person2 = new Person("Sarah Jones");**

**Console.WriteLine(person2.Name);**

**// Get the string representation of the person2 instance.**

**Console.WriteLine(person2);**

**Console.WriteLine("Press any key to exit.");**

**Console.ReadKey();**

**}**

**}**

**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();**

**}**

**}**

**class Triangle : Shape**

**{**

**public override void Draw()**

**{**

**// Code to draw a triangle...**

**Console.WriteLine("Drawing a triangle");**

**base.Draw();**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**// Polymorphism at work #1: a Rectangle, Triangle and Circle**

**// can all be used whereever a Shape is expected. No cast is**

**// required because an implicit conversion exists from a derived**

**// class to its base class.**

**var shapes = new List<Shape>**

**{**

**new Rectangle(),**

**new Triangle(),**

**new Circle()**

**};**

**// Polymorphism at work #2: the virtual method Draw is**

**// invoked on each of the derived classes, not the base class.**

**foreach (var shape in shapes)**

**{**

**shape.Draw();**

**}**

**// Keep the console open in debug mode.**

**Console.WriteLine("Press any key to exit.");**

**Console.ReadKey();**

**}**

**}**

**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 conclusión, la programación orientada a objetos y la programación procedural ambas tienen su ventajas y desventajas, cada uno tiene su propósito y no podemos decir que un paradigma es mejor que otro, para algunas cosas nos es más sencillo utilizar uno o simplemente nos conviene mejor más usar uno debido a la tarea que se va a realizar. También al conocer mejor la historia de creación de algunos de los lenguajes orientados a objetos uno se puede dar cuenta que se desarrollaron para cubrir una necesidad de mejorar la manera en que se desarrolla el software, al abstraer objetos de la vida real en objetos de programación las posibilidades de desarrollar softwares más complejos aumentaron y con ello el desarrollo de software fue adquiriendo mayor importancia en varios aspectos de nuestra vida cotidiana.

# 

# Bibliografía

Alvarez, M. A. (2001, Julio 24). *Qué es la programación orientada a objetos*. Retrieved from Desarrolloweb: <https://desarrolloweb.com/articulos/499.php>

*C (lenguaje de programación)*. (2019, Marzo 08). Retrieved from Wikipedia: [https://es.wikipedia.org/wiki/C\_(lenguaje\_de\_programación)](https://es.wikipedia.org/wiki/C_(lenguaje_de_programaci%C3%B3n))

*C++*. (2019, Marzo 05). Retrieved from Wikipedia: https://es.wikipedia.org/wiki/C%2B%2B

Diaz, C. (n.d.). *Encapsulación en C#*. Retrieved from francysystem: <https://francysystem.blogspot.com/2015/02/encapsulacion-en-c.html>

Gomez, D. (2016, Diciembre 01). *Historia del lenguaje C++*. Retrieved from michelletorres.mx: <https://blog.michelletorres.mx/historia-del-lenguaje-c/>

González, J. D. (2018, febrero 14). *Objetos, clases y constructores en Java.* Retrieved from programarya: <https://www.programarya.com/Cursos/Java/Objetos-y-Clases>

*Historia del lenguaje Java*. (n.d.). Retrieved from cad: <http://www.cad.com.mx/historia_del_lenguaje_java.htm>

Pablo, J. (2018, Marzo 02). *La historia de Python*. Retrieved from Platzi: <https://platzi.com/blog/historia-python/>

*Programación en C*. (2018, Septiembre 11). Retrieved from Wikibooks: [https://es.wikibooks.org/wiki/Programación\_en\_C/Ejemplos](https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C/Ejemplos)

*Programación en C++*. (2018, Diciembre 10). Retrieved from Wikibooks: [https://es.wikibooks.org/wiki/Programación\_en\_C%2B%2B/Objetos\_y\_Clases](https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_C%2B%2B/Objetos_y_Clases)

*Programación Orientada a Objetos con C++*. (n.d.). Retrieved from LabSCN: <https://labscn-unalmed.github.io/programacion-R/clases/clase-03.html>

*Universidad de Oviedo*. (n.d.). Retrieved from Uniovi: <http://www.atc.uniovi.es/telematica/2ac/Transparencias/T02-Programacion-Procedural.pdf>

Wagner, B. (2018, Agosto 20). *Clases (Guía de programación de C#)*. Retrieved from Microsoft: <https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/classes>