Universidad Autónoma de Ciudad Juárez

División Multidisciplinaria Ciudad Universitaria

Comparative table of Object Oriented Programming languages

Programming II

Teacher: Alan Ponce

Tanya Michelle Rincon Tarango 169870

Software Engineering

March 08, 2019

Introduction

Procedural Language

Series of obligatory steps. It is always the next step. Input - Process - Output.
It is governed by two basic concepts for the construction of programs: the structure and the module.

- **C**

```
#include <stdio.h>
int main()
{
   // printf() displays the string inside quotation
   printf("Hello, World!");
   return 0;
}
```

- **Pascal**

```
program ProjectHelloWorld;
begin
    writeln('Hello, World!');
end.
```

- **Fortran**

```
program helloworld
    print *, "Hello world!"
end program helloworld
```

- **Basic**

```
PRINT "Hello, World!"
END
```

- **Cobol**

```
IDENTIFICATION DIVISION.
      PROGRAM-ID. hello-world.
      PROCEDURE DIVISION.
          DISPLAY "Hello, world!"
          .
```

- **Algol**

BEGIN DISPLAY("HELLO WORLD!") END.

Object Oriented Languages

The emulation of reality, is handled as a structure of data that are stored in a logical way representing reality, each representation within the program is called object.
As in real life, each object has status and behavior, and these same can relate to each other.

- **C++**

It begins in 1972 with the design of C by the Bell laboratories of AT & T as a programming language of procedural type. In 1980, C ++ emerged in the same way by the Bell laboratories of AT & T, to which the use of virtual classes and functions was added. Start using the multiple inheritance in the objects by combining the procedural programming of C with the object-oriented programming. Years later the incorporation of the STL library (Standar Template Library) had a great evaluation. In 1998 the language is approved and formalized.

```
#include <iostream>

int main()
{
   std::cout << "Hello, world!\n";
   return 0;
}
```

- **Java**

In 1991 OAK is created whose name changed to Java. It was intended to offer a programming language relatively similar to C ++ but with a virtual machine.
In 1992, low-level prototypes of what is now Java are presented and between 1993 and 1994 a functional Java prototype is presented.
In 1995 the Alpha version of Java is presented and a year later the JFK 1.0 is released.

```
class HelloWorldApp {
   public static void main(String[] args) {
      System.out.println("Hello World!"); // Prints the string to the console.
   }
}
```

- **Python**

Implemented by Guido Van Rossum in 1989, but it was until 1991 when the public version 0.9.0 was released. Version 1.0 was published in 1994, version 2.0 in 2000 and version 3.0 in 2018. Until 2018 Guido Van Rossum announced that he would stop directing the development of Python, which would be entrusted to members of the Python Software Foundation.

```
print("Hello World")
```

- **C#**

Version 1.0 of C # was very similar to java, which tried to make the OOP language modern and simple. In 2003 the version 1.2 of C # was included in Visual Studio, the most notable of the version was the incorporation of foreach loop. In 2005, together with VS 2005, C # 2.0 was launched, which handled static classes and groups of methods. In 2007 together with VS 2008, version 3.0 was launched in which the object initializers and collections of these began. In 2012 version 5.0 was concentrated on the async model and await as call attributes.
The most recent version is the 7.0 throw expressions are noted and the keyword out that returns through tuples, we try to concentrate on the cloud and portability.

```
using System;

class Program
{
   static void Main(string[] args)
   {
      Console.WriteLine("Hello, world!");
   }
}
```

Procedural Paradigm	Object Oriented Paradigm
● It focuses on procedure (function). ● Based on instruction for the computer to do something. ● Uses procedures. ● Uses records. ● Uses modules. ● Uses procedure calls. ● Is not able for reused and recycled	● That uses classes and objects to create models ● Based on the real world environment. ● Easy Understand the program for other developers. ● Uses methods. ● Uses objects. ● Uses messages. ● Uses class.

● It focuses on process rather than data. ● Dividing the program into functions. ● How not uses objects not uses inheritance. ● It's sequences ● Workflow in the program ● the next step does not start without the previous one ending ● It is mainly based on procedures ● Provide clearer understanding of the code to the programmer	● Is able for reused and recycled ● Reuse the code and the design. ● Reduces the risks of software. ● Uses the polymorphism and inheritance. ● Through a base class you can perform several tasks in secondary class ● captures the behaviors of an object ● Memory management ● It keeps internal processes hidden and only shows the necessary

Object Oriented Programming Languages

C++

- *A class*

```
//Definición de la clase persona -> archivo "Persona.h"
//siempre es buena idea usar macros del preprocesador para evitar compilar varias veces el mismo archivo

#ifndef PERSONA_H
#define PERSONA_H

#include <string>
	class Persona //Declaramos la clase con el nombre Persona
	{
		private:    //a partir de aquí todos los miembros serán privados
//los datos miembro pueden ser cualquier tipo de dato, incluso otras clases como string
			std::string nombre;
			int edad;
			float estatura;
			float peso;
				//métodos privados
			float aumentaEstatura(float metros){return estatura += metros};
		//función inline
			float aumentaPeso(float kilogramos){return peso += kilogramos};

			public:     //a apatir de aquí todas las declaraciones serán de acceso público
			Persona(const std::string& nombre,int edad, float peso, float estatura); // Constructor
			void saluda();
			int cumpleAnios();
	};
#endif
```

- inheritance

```
// 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;
};
```

- Polymorphism

```
# include <iostream>
using namespace std;
class Animal
{
public:
virtual void come() { cout << "Yo como como un animal genérico.\n"; }
};

class Lobo : public Animal
{
public:
void come() { cout << "¡Yo como como un lobo!\n"; }
};

class Pez : public Animal
```

```
{
public:
void come() { cout << "¡Yo como como un pez!\n"; }
};

class OtroAnimal : public Animal
{
};

int main()
{
Animal *unAnimal[4];
unAnimal[0] = new Animal();
unAnimal[1] = new Lobo();
unAnimal[2] = new Pez();
unAnimal[3] = new OtroAnimal();

for(int i = 0; i < 4; i++) {
unAnimal[i]->come();
}

for (int i = 0; i < 4; i++) {
delete unAnimal[i];
}
return 0;
}
```

- Encapsulation

```
class TObjGraf {

public:
  int       X;        // Propiedades
  int       Y;
  TColor    Color;
  TPaintBox *PaintBox;

  void Mostrar (void); // Métodos
};
void __fastcall TPpalFrm::FormCreate(TObject *Sender)
{
  ...
  int ValorY;
```

```
   ...
   ObjGraf->X = 5;
   ValorY = ObjGraf->Y;
   ObjGraf->Mostrar();   //Equivalente a (*Obj).Mostrar();
}
```

Java

- A class

```
package misClases; //Se le declara un paquete

public class Animal
{
   private String raza;
   private String nombre;
   private int edad;

   public Animal(String nuevoNombre)
   {
      nombre = nuevoNombre; //Se le da un nombre al animal
   }
   //Método para obtener la edad del animal
   public int getEdad()
   {
      return edad;
   }
   //Método para establecer la edad del animal
   public void setEdad(int nuevaEdad)
   {
      edad = nuevaEdad;
   }
   //Método para obtener el nombre del animal
   public String getNombre()
   {
      return nombre;
   }
}
```

- inheritance

```
//Se utiliza la palabra: extends
```

```
//Código de la clase Persona ejemplo aprenderaprogramar.com

public class Persona {

    private String nombre;

    private String apellidos;

    private int edad;

    //Constructor

    public Persona (String nombre, String apellidos, int edad) {

        this.nombre = nombre;

        this.apellidos = apellidos;

        this.edad = edad;                  }

    //Métodos

    public String getNombre () { return nombre;  }

    public String getApellidos () { return apellidos;  }

    public int getEdad () { return edad;   }

} //Cierre de la clase


//Código de la clase profesor, subclase de la clase Persona ejemplo
aprenderaprogramar.com

public class Profesor extends Persona {

    //Campos específicos de la subclase.

    private String IdProfesor;
```

```
    //Constructor de la subclase: incluimos como parámetros al menos los del constructor de la superclase

    public Profesor (String nombre, String apellidos, int edad) {

        super(nombre, apellidos, edad);

        IdProfesor = "Unknown";   } //Cierre del constructor

    //Métodos específicos de la subclase

    public void setIdProfesor (String IdProfesor) { this.IdProfesor = IdProfesor;   }

    public String getIdProfesor () { return IdProfesor;   }

    public void mostrarNombreApellidosYCarnet() {

        // nombre = "Paco"; Si tratáramos de acceder directamente a un campo privado de la superclase, salta un error

        // Sí podemos acceder a variables de instancia a través de los métodos de acceso públicos de la superclase

        System.out.println ("Profesor de nombre: " + getNombre() + " " +  getApellidos() +

         " con Id de profesor: " + getIdProfesor() ); }

} //Cierre de la clase
```

- Polymorphism

```
public abstract class Animal implements IAnimal {

 private String nombre;

 /**
  * Constructor de la clase Animal
  * @param nombre
  */
 public Animal (String nombre){
  this.nombre=nombre;
```

```
  System.out.println("Constructor Animal, " +
    "nombre del animal : "+this.nombre);
 }

 /**
  * Retorna el valor de nombre
  * @return
  */
 public String getNombre(){
  return nombre;
 }


 /**
  * Metodo Abstracto tipoAnimal, la implementación depende
  * de las clases concretas que extiendan la clase Animal
  */
 public abstract void tipoAnimal();

}

public class Gato extends Animal{

   /**
    * Constructor explicito clase Gato
    * @param nombre
    */
    public Gato(String nombre) {
  super(nombre);//envia el parametro a el constructor de la clase padre
 System.out.println("Constructor Gato, nombre : "+nombre);
    }

    public void tipoAnimal() {
 System.out.println("Tipo Animal : Es un Gato");
    }

    public void comunicarse(){
  System.out.println("Metodo comunicarse : El gato maulla... Miau Miau");
    }
}

public class Test {
```

```
 public static void main (String[] arg){

 /**Creamos anim, un objeto Perro de tipo Animal*/
 Animal anim= new Perro("goliath") ;
 anim.tipoAnimal();
 anim.comunicarse();
 System.out.println();

 /**Creamos perro, un objeto Perro de tipo Perro*/
 Perro perro=new Perro("hercules");
 perro.tipoAnimal();
 System.out.println();

 /**Creamos animalPolimorfico, un objeto perro de tipo Animal
  * asignamos una referencia ya existente*/
 Animal animalPolimorfico=perro;
 animalPolimorfico.tipoAnimal();
 System.out.println();

 /**reasignamos la referencia del objeto anim a el objeto perro
  * esto es valido ya que ambos son de tipo Perro*/
 perro=(Perro) anim;
 perro.tipoAnimal();
 System.out.println();
 }
}
```

- Encapsulation

```
public class MiClase{
        private int tipo;
                public void setTipo(int t){
                        tipo = t;
                }
        public int getTipo() {
                return tipo;
        }
}

class AccesoIndirecto {
        public static void main (String[] args) {
                MiClase mc = new MiClase();
```

```
                mc.setTipo(5);
                System.out.println("El tipo es: "+mc.getTipo());
        }
}
```

Python

- A class

```
class Perro:
    def __init__(self, nombre):
        self.nombre = nombre
        self.trucos = []    # crea una nueva lista vacía para cada perro

    def agregar_truco(self, truco):
        self.trucos.append(truco)

>>> d = Perro('Fido')
>>> e = Perro('Buddy')
>>> d.agregar_truco('girar')
>>> e.agregar_truco('hacerse el muerto')
>>> d.trucos
['girar']
>>> e.trucos
['hacerse el muerto']
```

- inheritance

```
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)

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
```

- Polymorphism

```
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
```

```
>>> frecuencias(["peras", "manzanas", "peras", "manzanas", "uvas"])
{'uvas': 1, 'peras': 2, 'manzanas': 2}
>>> frecuencias((1,3,4,2,3,1))
{1: 2, 2: 1, 3: 2, 4: 1}
>>> frecuencias("Una frase")
{'a': 2, ' ': 1, 'e': 1, 'f': 1, 'n': 1, 's': 1, 'r': 1, 'U': 1}
>>> ran = xrange(3, 10, 2)
>>> frecuencias(ran)
{9: 1, 3: 1, 5: 1, 7: 1}
>>> frecuencias(4)
Traceback (most recent call last):
    File "<pyshell\#0>", line 1, in <module>
        frecuencias(4)
    File "frecuencias.py", line 12, in frecuencias
        for v in seq:
    TypeError: 'int' object is not iterable
```

- Encapsulation

```
#!/usr/bin/env python
# Encapsulacion en python
class Automotor(object) :
  """Clase de la cual heredan las demas y que es del nuevo estilo en python"""

  def __init__(self, ensambladora, nombre) :
    self.setEnsambladora(ensambladora)
    self.setNombre(nombre)
    print "Automotor", nombre, "Creado!!!"

  def setEnsambladora(self, ensambladora) :
    """Determina la ensambladora"""
    self.__ensambladora = ensambladora

  def getEnsambladora(self) :
    """Retorna la ensambladora"""
    return self.__ensambladora

  def setNombre(self, nombre) :
    """Determina el nombre del automotor"""
    self.__nombre = nombre

  def getNombre(self) :
    """Retorna el nombre del automotor"""
    return self.__nombre

  #Determinamos que el unico acceso a las variables es por los metodos
  ensambladora = property(getEnsambladora, setEnsambladora)
  nombre = property(getNombre, setNombre)


class Carro(Automotor) :
  """Esta clase hereda de Automotor y sus metodos"""
  pass

class Motocicleta(Automotor) :
  """Esta clase hereda de Automotor y sus metodos, pero es diferente a Carro"""
  pass

if __name__ == '__main__':
  carro = Carro("Toyota", "Celica")
```

```
moto = Motocicleta("Auteco", "Pulsar")

print "\n\n"
print "El carro es", carro.getNombre()
print "La moto es", moto.getNombre()
```

C#

- A class and inheritance

```
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();
    }
}
```

- Polymorphism

```
public class BaseClass
{
    public virtual void DoWork() { }
    public virtual int WorkProperty
    {
        get { return 0; }
    }
}
public class DerivedClass : BaseClass
{
    public override void DoWork() { }
    public override int WorkProperty
    {
        get { return 0; }
    }
}
```

- Encapsulation

```
sing System ;

class clase_cliente {
  // Definición de variables de instancia.
  private string is_nombre ;
  private string is_ape1 ;
  private string is_ape2 ;

  // Definición de métodos públicos (interfaz pública de la clase).
  public void nombre(string nombre, string ape1, string ape2) {
```

```
        is_nombre = nombre ;
        is_ape1 = ape1 ;
        is_ape2 = ape2 ;
  }

  public string nombre() {
  return nombre_completo() ;
  }

  // Definición de métodos de soporte.
  private string nombre_completo() {
        return "Nombre: " + is_nombre + " | Primer ape.: " + is_ape1
        + " | Segundo ape.: " + is_ape2 ;
  }
}

class Ejemplo6 {
  // Procedimiento principal.
  public static void Main() {
        // Definimos un objeto cliente a partir de una clase.
        clase_cliente objeto_cliente ;

        // Creamos un objeto a partir de la clase cliente.
        objeto_cliente = new clase_cliente() ;

        // Usamos m,todo para asignar nombre de cliente.
        objeto_cliente.nombre("Angel Luis","Garcia","Garcia") ;

        // Usamos m,todo para mostrar el nombre completo.
        Console.WriteLine(objeto_cliente.nombre()) ;
  }
}
```

Conclutions

After conducting research between procedural paradigm and object-oriented paradigm it was easier to find the difference between using only one algorithm and the behavior of an object, but both paradigms have in common that they look for the computer to perform specific tasks.

For personal and practical tastes, I am more oriented to the python-oriented object language because of its simplicity feature when writing a set of code lines. On the side of personal comfort I prefer the language of C ++ programming, but in the sense of greater evaluation and growth, both work and practice, Java is better and is more directed to the future of work. Similarly using a code as simple as the "Hello World" was relatively easy to notice the simplicity of each programming language, and how a simple and easy task can occupy 3 or up to 7 lines of code.

Bibliography

Alex Rodriguez. (2019). Ejemplo de herencia en Java. Uso de palabras clave extends y super. Constructores con herencia.. 7 de marzo de 2019, de Aprende a programar Sitio web: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=653:ejemplo-de-herencia-en-java-uso-de-palabras-clave-extends-y-super-constructores-con-herencia-cu00686b&catid=68&Itemid=188

Curso de C++. (N/A). 36 Herencia. 7 de marzo de 2019, de Curso de C++ Sitio web: http://c.conclase.net/curso/index.php?cap=036

Curso de C++ Builder. (N/A). Programación Orientada a Objetos en C++. 7 de marzo de 2019, de Curso de C++ Builder Sitio web: https://elvex.ugr.es/decsai/builder/intro/5.html#ENCAPSULAMIENTO

Nancy Herrera. (2012). Implementación de polimorfismo en funciones y funciones virtuales de objetos. 7 de marzo de 2019, de Blogspot Sitio web: http://tecnicasdeprogra.blogspot.com/2012/05/implementacion-de-polimorfismo-en.html

N/A. (2018). Clases (Guía de programación de C#). 7 de marzo de 2019, de Microsoft Sitio web: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/classes

N/A. (2015). Polimorfismo (Guía de programación de C#). 7 de marzo de 2019, de Microsoft Sitio web: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/polymorphism

Python Software Foundation. (2017). Tutorial de Python. 7 de marzo de 2019, de Python Software Foundation Sitio web: http://docs.python.org.ar/tutorial/3/classes.html

Uniwebuniversidad. (2014). Herencia. 7 de marzo de 2019, de Uniwebuniversidad Sitio web: https://uniwebsidad.com/libros/algoritmos-python/capitulo-15/herencia