Universidad Autónoma de Ciudad Juárez

División Multidisciplinaria Ciudad Universitaria

Cuadro comparativo de lenguajes de Programación Orientada a Objetos

Programación II

Docente: Alan Ponce

Alumno 169840

Héctor Gerardo Sánchez Quiroga

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

Indicar lenguajes de programación y ejemplos de código

El lenguaje procedural, es aquel que se basa en instrucciones, condiciones y pasos. Las instrucciones transforman el programa hasta que se puede llegar a una solución que resolvería la problemática. Este tipo de programación es parecido a los algoritmos, porque se sigue cada instrucción, este algoritmo debería de resolver la problemática. (Loopa, 2016)

A continuación se observa una calculadora en escrito en Fortran, Pascal y C:

Fortran:

```
program Calc
   real :: a, b, additionAnswer, subtractionAnswer, multiplicationAnswer, divisionAnswer
   print *, "George Gibson's Calc"
   print *, "Type the first number: "
   read *, a
   print *, "Type the second number: "
   read *, b
   additionAnswer = a + b
   subtractionAnswer = a - b
   multiplicationAnswer = a * b
   divisionAnswer = a / b
   print *, a, " + ", b, " = ", additionAnswer
   print *, a, " - ", b, " = ", subtractionAnswer
   print *, a, " * ", b, " = ", multiplicationAnswer
   print *, a, " / ", b, " = ", divisionAnswer
   read *
end program Calc
```

Pascal:

```
Program Basic_Calculator;
Uses Crt;
Var
 Operator: Char;
 Opt1,Opt2 : Integer;
 Sum,Subtract, Multiply: Integer;
 Divide : Integer;
 Result : Integer;
Begin
 clrscr;
 Opt1 := 0; Opt2 := 0;
```

```
 write('Simple Math Calculator');
 writeln; writeln;
 write('Enter Two Numbers : ');
 readln(Opt1,Opt2);
 write('Select Operators +,-,* and / : ');
 readln(Operator);
 if (Operator = '+') then
Begin
 Sum := (Opt1 + Opt2);
 Result := Sum;
End
else if (Operator = '-') then
Begin
 Subtract := (Opt1 - Opt2);
 Result := Subtract;
End
else if (Operator = '*') then
Begin
 Multiply := (Opt1 * Opt2);
 Result := Multiply;
End
else if (Operator = '/') then
Begin
 Divide := (Opt1 DIV Opt2);
 Result := Divide;
End;
 writeln;
 writeln;
 Write('The result is ' ,Result,'.');
 readln;
End.
```

C:

```
# include <stdio.h>
int main() {
    char operator;
    double firstNumber,secondNumber;
    printf("Enter an operator (+, -, *,): ");
    scanf("%c", &operator);
    printf("Enter two operands: ");
    scanf("%lf %lf",&firstNumber, &secondNumber);
    switch(operator)
    {
        case '+':
                        printf("%.1lf + %.1lf = %.1lf",firstNumber, secondNumber, firstNumber +
secondNumber);
```

```
            break;
        case '-':
                        printf("%.1lf - %.1lf = %.1lf",firstNumber, secondNumber, firstNumber -
secondNumber);
            break;
        case '*':
                       printf("%.1lf * %.1lf = %.1lf",firstNumber, secondNumber, firstNumber *
secondNumber);
            break;
        case '/':
                        printf("%.1lf / %.1lf = %.1lf",firstNumber, secondNumber, firstNumber /
secondNumber);
            break;
        // operator doesn't match any case constant (+, -, *, /)
        default:
            printf("Error! operator is not correct");
    }
    return 0;
}
```

Como se observa las diferencias primarias es primero en las palabras reservadas que comparten los tres lenguajes, como en el caso de Fortran utiliza real para referenciar a los números reales, mientras que en C, se utiliza float, o double para los reales con parte decimal más longeva. Fortran no utiliza ; para terminar las declaraciones. Mientras que Pascal y C, si la utilizan. Fortran tampoco puede utilizar punteros, ni recursividad. Pascal utiliza Begin/End. Pascal primero pone la variable luego la declara ejemplo:
Resultado : Integer. Pero estos lenguajes siguen teniendo una similitud entre ellos, en sí al comparar los códigos son muy parecidos, en cierto modo sólo la sintaxis y unos extras como los anteriores mencionados son los que cambian.

2.- Definición de lenguaje orientada a objetos

Indicar lenguajes de programacion y una resena de creacion y evolucion (C++, Java, Python, C#)

El fundamento principal del paradigma orientado a objetos, es que combina en una sola unidad los datos y las funciones que operan estos datos. Esta unidad se llama objeto. Se pueden crear relaciones entre los objetos. Las funciones del objeto es usualmente la única forma para poder acceder a los datos de los objetos. Si quieres leer datos de un objeto, llamas una función del objeto que da entrada a los datos y luego regresa los valores. Esto hace que no se pueda acceder a los datos directamente. (Lafore, 2005)

C++

Fue creado por Bjarne Stroustrup, y su objetivo era ayudar a implementar proyectos de simulaciones en un paradigma orientado a objetos y de una manera eficiente. C++ es un híbrido ya que contiene la funcionalidad de C, esto significa que tiene todas sus características, como seria estar muy cercano a la programación maquina. Aparte de esto como es un lenguaje Orientado a Objetos, también posee las características como abstracción de datos, encapsulamiento, herencia y polimorfismo

C++ primero surgió como una idea de Stroustrup en la que se quería añadir la programación orientada a objetos a C, esto sería conocido como “C with Classes”. Este lenguaje incluye clases, herencia básica, entre otras características, aparte de las que ya eran propias del lenguaje de C. El primer compilador que se origina para “C with Classes” fue Cfront, este compilador lo que hacía era traducir el código de “C with Classes” a código en C. Luego lo compilaba. Para 1983, se originó C++, que contiene características como funciones virtuales y sobrecarga de funciones. En 1989, incluye miembros protegidos y miembros estáticos, además de herencia para varias clases. (Kirch-Prinz & Prinz, 2002) (History, N.d.)

Java

Java fue creado por James Gosling, Patrick Naughton, Chris Warth, Ed Frank, y Mike Sheridan en Sun Microsystems, Inc. En 1991. Java está relacionado con C++, Java hereda mucho de estos dos lenguajes, como su sintaxis, y también algunas características de programación orientada a objetos de C++. Su primera meta era que Java fuera un lenguaje que fuera independiente a la plataforma. Que podía ser usado para crear software embebido en varios electrodomésticos. Pero Java está siendo diseñado en el momento justo que la “World Wide Web” está ascendiendo, porque la Web también requiere de programas portables.

Para 1993, los miembros del equipo notaron que los problemas de portabilidad que se tenían al crear código para controladores embebidos, era el mismo que se podía aplicar en el internet a gran escala. Esto hizo que se enfocara a trabajar en la programación en el Internet. (Schildt, 2007)

Python

Fue creado en 1980 por Guido Van Rossum, como un sucesor al lenguaje ABC que era capaz de manejo de excepciones y la interfaz con el OS Amoeba. Se enfoca en la legibilidad del código y la sintaxis que utiliza hace que se requerían menos líneas de código que en lenguajes como C++ o Java. Python maneja diferentes paradigmas, como OO, Imperativo, y funcional. Tiene un manejo de memoria automático. (Venners, 2013)

C#

Desarrollado en 2002 por Anders Hejlsberg. Este lenguaje es para uso general, utiliza el paradigma orientado a objetos. Con el paso del tiempo ha mejorado como para el 2007 se implementó el uso de expresiones lambda, árboles de expresiones y métodos parciales. Y actualmente la versión más actual contiene tuplas, coincidencias de patrones, decomposición, y funciones locales. (Srivastava, 2017)

3.- Cuadro comparativo de estos dos paradigmas

Paradigma Procedural	Paradigma Orientada a Objetos
Programa está dividido en funciones	Programa está dividido en objetos
La ejecución es paso por paso	La ejecución es simultáneamente
No hay concepto de herencia	La herencia se puede alcanzar en tres modos: Pública, privada y protegida
No hay forma de esconder los datos	Los datos pueden estar escondidos en tres modos: Públicos, Privados y Protegidos
Se debe de alterar todo el programa para implementar una modificación	Los objetos pueden ser creados con facilidad y pueden ser modificados fácilmente.

Table created with help from: (Khillar, 2018)
Link: http://www.differencebetween.net/technology/difference-between-oop-and-pop/

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 codigo de como implementar:

Una clase

```
#include <iostream>
using namespace std;
class Rectangle {
    int width, height;
  public:
    void set_values (int,int);
    int area() {return width*height;}
};
void Rectangle::set_values (int x, int y) {
  width = x;
  height = y;
}
int main () {
  Rectangle rect;
  rect.set_values (3,4);
  cout << "area: " << rect.area();
  return 0;
}
```

Como se observa la clase rectangulo, tiene valores como son el de altura y ancho. Aparte de esto tiene dos métodos que son el de establecer los valores, y el área. Estos dos métodos son propios de la clase rectangulo. Y se pueden crear diferentes “rectángulos”

Herencia

```
#include <iostream>
using namespace std;
// Base class
class Shape {
   public:
      void setWidth(int w) {
         width = w;
      }
      void setHeight(int h) {
         height = h;
      }
```

```
  protected:
    int width;
    int height;
};
// Derived class
class Rectangle: public Shape {
  public:
    int getArea() {
      return (width * height);
    }
};
int main(void) {
  Rectangle Rect;
  Rect.setWidth(5);
  Rect.setHeight(7);
  // Print the area of the object.
  cout << "Total area: " << Rect.getArea() << endl;
  return 0;
}
```

Un ejemplo parecido al anterior solamente que se tiene una clase base, en las que la clase derivada rectángulo hereda las variables como altura y ancho. Además de esto también hereda los sets que nos servirán para los valores anteriormente mencionados

Poliformismo

```
#include <iostream>
using namespace std;
class Shape {
  protected:
    int width, height;
  public:
    Shape( int a = 0, int b = 0){
      width = a;
      height = b;
    }
    int area() {
      cout << "Parent class area :" <<endl;
      return 0;
    }
};
class Rectangle: public Shape {
  public:
    Rectangle( int a = 0, int b = 0):Shape(a, b) { }
    int area () {
      cout << "Rectangle class area :" <<endl;
      return (width * height);
    }
};
```

```
class Triangle: public Shape {
  public:
    Triangle( int a = 0, int b = 0):Shape(a, b) { }
    int area () {
      cout << "Triangle class area :" <<endl;
      return (width * height / 2);
    }
};
// Main function for the program
int main() {
  Shape *shape;
  ...
  return 0;
}
```

Como se observa en el polimorfismo la clase base que es figura puede dar a dos clases diferentes que en este caso serían el rectángulo y el triángulo. Y cada una de estas clases se implementan de forma diferente que en el ejemplo seria el área de la figura. Realizan acciones diferentes pero siguen siendo de la misma clase base.

Encapsulamiento

```
#include<iostream>
using namespace std;
class ExampleEncap{
private:
  /* Since we have marked these data members private,*/
  int num;
  char ch;
public:
  /* Getter functions to get the value of data members.*/
  int getNum() const {
    return num;
  }
  char getCh() const {
    return ch;
  }
  /* Setter functions, they are called for assigning the values
   * to the private data members.
   */
  void setNum(int num) {
    this->num = num;
  }
  void setCh(char ch) {
    this->ch = ch;
  }
};
int main(){
  ExampleEncap obj;
```

```
  obj.setNum(100);
  obj.setCh('A');
  cout<<obj.getNum()<<endl;
  cout<<obj.getCh()<<endl;
  return 0;
}
```

Cómo num y ch, han sido declarados como privados se necesitará el apoyo del getter y setter para poder asignar los valores y obtener los valores. Esto es una forma de encapsulamiento ya que previene que se maneje directamente con los datos.

Java

Incluir codigo de como implementar:

Una clase

```
class Lamp {
  // instance variable
  private boolean isOn;
  // method
  public void turnOn() {
    isOn = true;
  }
// method
  public void turnOff() {
      isOn = false;
  }
}
```

Al implementar una clase es muy parecido a lo que se hace en C++, también se declaran privadas o públicas y se ingresan los métodos y las variables que tienen la clase.

Herencia

```
class Teacher {
  String designation = "Teacher";
  String collegeName = "Beginnersbook";
  void does(){
      System.out.println("Teaching");
  }
}
public class PhysicsTeacher extends Teacher{
  String mainSubject = "Physics";
  public static void main(String args[]){
      PhysicsTeacher obj = new PhysicsTeacher();
      System.out.println(obj.collegeName);
      System.out.println(obj.designation);
      System.out.println(obj.mainSubject);
      obj.does();
  }
}
```

De la clase base Teacher se heredó los valores y los métodos para poder crear la clase Physics Teacher, y esta clase PhysicsTeacher comparte los valores y métodos de Teacher pero también tiene una variable para el que es mainSubject.

Poliformismo

```
class Vehicle{
   public void move(){
   System.out.println(“Vehicles can move!!”);
   }
}
class MotorBike extends Vehicle{
   public void move(){
   System.out.println(“MotorBike can move and accelerate too!!”);
   }
}
class car extends Vehicle{
   public void move(){
   System.out.println("Car can move and accelerate too")
}
}
class Test{
   public static void main(String[] args){
   Vehicle vh=new MotorBike();
   vh.move();    // prints MotorBike can move and accelerate too!!
   vh=new Vehicle();
   vh.move();    // prints Vehicles can move!!
   vh=new Car();
   vh.move();    // prints car can move and accelerate too
   }
}
```

En java se observa que también de la clase Vehicle se obtiene el método de Move(). Este método funciona en ambas clases ya sea en car o MotorBike y cada una genera una impresión diferente.

Encapsulamiento

```
class EncapsulationDemo{
   private int ssn;
   private String empName;
   private int empAge;
   //Getter and Setter methods
   public int getEmpSSN(){
      return ssn;
   }
   public String getEmpName(){
      return empName;
   }
   public int getEmpAge(){
      return empAge;
```

```
   }
   public void setEmpAge(int newValue){
      empAge = newValue;
   }
   public void setEmpName(String newValue){
      empName = newValue;
   }
   public void setEmpSSN(int newValue){
      ssn = newValue;
   }
}
public class EncapsTest{
   public static void main(String args[]){
       EncapsulationDemo obj = new EncapsulationDemo();
       obj.setEmpName("Mario");
       obj.setEmpAge(32);
       obj.setEmpSSN(112233);
       System.out.println("Employee Name: " + obj.getEmpName());
       System.out.println("Employee SSN: " + obj.getEmpSSN());
       System.out.println("Employee Age: " + obj.getEmpAge());
   }
}
```

Al igual que en C++ se necesitan los getter y setter para obtener y modificar los datos de las variables.

Python

Incluir codigo de como implementar:

Una clase

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Las clases en python son un poco diferentes en la sintaxis, que las clases en C++ o Java. Pero estas siguen dando la misma idea de tener métodos y datos, también identidad.

Herencia

```
class Person:

    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last
```

```
    def Name(self):
        return self.firstname + " " + self.lastname

class Employee(Person):

    def __init__(self, first, last, staffnum):
        Person.__init__(self,first, last)
        self.staffnumber = staffnum

    def GetEmployee(self):
        return self.Name() + ", " +  self.staffnumber

x = Person("Marge", "Simpson")
y = Employee("Homer", "Simpson", "1007")

print(x.Name())
print(y.GetEmployee())
```

En este ejemplo se puede observar que el empleado es una clase derivada de la clase Persona, entonces se puede decir que todos los empleados son personas, pero no por esto signifique que todos las personas son empleados.

Poliformismo

```
class Bird:
  def intro(self):
    print("There are many types of birds.")

  def flight(self):
    print("Most of the birds can fly but some cannot.")

class sparrow(Bird):
  def flight(self):
    print("Sparrows can fly.")

class ostrich(Bird):
  def flight(self):
    print("Ostriches cannot fly.")
```

Todas las clases de Ostrich, y Sparrow provienen de Bird, y estas contienen métodos como sería el de flight, y muestra que una puede volar mientras que la otra carece de esa habilidad.

Encapsulamiento

```
class Robot(object):
  def __init__(self):
    self.a = 123
    self._b = 123
    self.__c = 123

obj = Robot()
```

```
print(obj.a)
print(obj._b)
print(obj.__c)
```

Python no tiene private, asi que lo que hace para encapsular los datos es utilizar un _ cuando una clase no debe ser accedida directamente.

C#

Incluir codigo de como implementar:

Una clase

```
// C# program to illustrate the
// Initialization of an object
using System;
// Class Declaration
public class Dog {
    // Instance Variables
    String name;
    String breed;
    int age;
    String color;
    // Constructor Declaration of Class
    public Dog(String name, String breed,
               int age, String color)
    {
        this.name = name;
        this.breed = breed;
        this.age = age;
        this.color = color;
    }
    // method 1
    public String getName()
    {
        return name;
    }
    // method 2
    public String getBreed()
    {
        return breed;
    }
    // method 3
    public int getAge()
    {
        return age;
    }
    // method 4
```

```
        public String getColor()
        {
            return color;
        }
        public String toString()
        {
            return ("Hi my name is " + this.getName()
                    + ".\nMy breed, age and color are " + this.getBreed()
                    + ", " + this.getAge() + ", " + this.getColor());
        }
    // Main Method
    public static void Main(String[] args)
        {

            // Creating object
            Dog tuffy = new Dog("tuffy", "papillon", 5, "white");
            Console.WriteLine(tuffy.toString());
        }
    }
     Igual que C++
    Herencia
using System;

namespace Tutlane
{
   public class User
        {
      public string Name;
      private string Location;
      public User()
        {
        Console.WriteLine("Base Class Constructor");
        }
      public void GetUserInfo(string loc)
        {
        Location = loc;
        Console.WriteLine("Name: {0}", Name);
        Console.WriteLine("Location: {0}", Location);
        }
        }
   public class Details : User
        {
      public int Age;
      public Details()
        {
        Console.WriteLine("Child Class Constructor");
        }
```

```
        public void GetAge()
            {
            Console.WriteLine("Age: {0}", Age);
            }
            }
    class Program
            {
        static void Main(string[] args)
            {
            Details d = new Details();
            d.Name = "Suresh Dasari";
            // Compile Time Error
            //d.Location = "Hyderabad";
            d.Age = 32;
            d.GetUserInfo("Hyderabad");
            d.GetAge();
            Console.WriteLine("\nPress Any Key to Exit..");
            Console.ReadLine();
            }
            }
}
```

Se crea la clase base user, a lo que luego se deriva una clase Details, y de esta clase se obtienen los datos.

Poliformismo

```
public class Drawing
{
    public virtual double Area()
    {
          return 0;
    }
}

public class Circle : Drawing
{
    public double Radius { get; set; }
    public Circle()
    {
        Radius = 5;
    }
    public override double Area()
    {
        return (3.14) * Math.Pow(Radius, 2);
    }
}
public class Square : Drawing
```

```
{
    public double Length { get; set; }
    public Square()
    {
        Length = 6;
    }
    public override double Area()
    {
        return Math.Pow(Length, 2);
    }
}
class Program
{
    static void Main(string[] args)
    {
        Drawing circle = new Circle();
        Console.WriteLine("Area :" + circle.Area());
        Drawing square = new Square();
        Console.WriteLine("Area :" + square.Area());
    }
}
```

Se tienen get y set para definir y obtener los valores de la clase base.

Encapsulamiento

```
using System;
using System.Text;

namespace Tutlane
{
   class User
        {
      private string location;
      private string name;
      public string Location
        {
        get
        {
           return location;
        }
        set
        {
        location = value;
        }
        }
      public string Name
        {
```

```
            get
            {
               return name;
            }
            set
            {
            name = value;
            }
            }
            }
    class Program
            {
        static void Main(string[] args)
            {
            User u = new User();
            // set accessor will invoke
            u.Name = "Suresh Dasari";
            // set accessor will invoke
            u.Location = "Hyderabad";
            // get accessor will invoke
            Console.WriteLine("Name: " + u.Name);
            // get accessor will invoke
            Console.WriteLine("Location: " + u.Location);
            Console.WriteLine("\nPress Enter Key to Exit..");
            Console.ReadLine();
            }
            }
}
```

Utiliza get y set para los datos, en el get obtiene los datos, mientras que en el Set los establece.

Conclusiones

La programación procedural puede ser una herramienta muy poderosa al momento de programar, pero cuando son proyectos que se requiere simular la realidad, la programación orientada a objeto tiene una ventaja masiva, su principal ventaja son los objetos. Estos objetos son instanciaciones de memoria, que tienen atributos, métodos y una identidad. Después se empiezan a conocer diferentes características como la encapsulación que en el caso de la procedural no sirve porque se utilizan los mismos datos para hacer la función. Mientras que en la OOP no se manejan los datos directamentes si no que se hace a través de interfaces. Con estos conocimientos adquiridos podemos notar que hay una gran diferencia entre los diferentes paradigmas de la programación y cada uno tiene su forma para ser utilizado.

References

History Of C# Programming Language. (n.d.). Retrieved from https://www.c-sharpcorner.com/blogs/history-of-c-sharp-programming-language

History of C. (n.d.). Retrieved from http://www.cplusplus.com/info/history/

Khillar, S. (2018, June 05). Difference Between. Retrieved from http://www.differencebetween.net/technology/difference-between-oop-and-pop/

Kirch-Prinz, U., & Prinz, P. (2002). *A complete guide to programming in C*. Jones and Bartlett.

Lafore, R. (2005). *Object-oriented programming in C , fourth edition*. SAMS.

Loopa. (2016, June 02). Paradigmas de Programación: Programación Imperativa y Programación Declarativa. Retrieved from https://medium.com/@Loopa/paradigmas-de-programación-programación-imperativa-y-programación-declarativa-4c4a4182fd87

Schildt, H. (2007). *Java: The complete reference*. McGraw-Hill.

Venners, B. (n.d.). ABC's Influence on Python. Retrieved from

https://www.artima.com/intv/pythonP.html