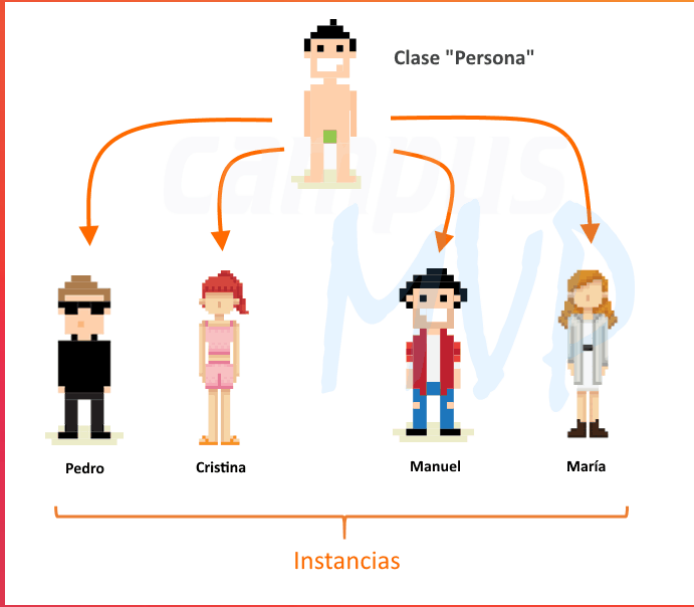
UNIVERSIDAD PRIVADA FRANZ TAMAYO
DEFENSA HITO 2 - TAREA FINAL
Estudiante: KEVIN JAVIER SANGA ORTIZ
Asignatura: ESTRUCTURA DE DATOS
Carrera: INGENIERÍA DE SISTEMAS
Paralelo: EDD (1)
Docente: Lic. William Barra Paredes
fecha: 30/03/2022
GITHUB:
https://github.com/kevinSanga/ESTRUCTURA_DE_DATOS_I/tree/main/HITO_2

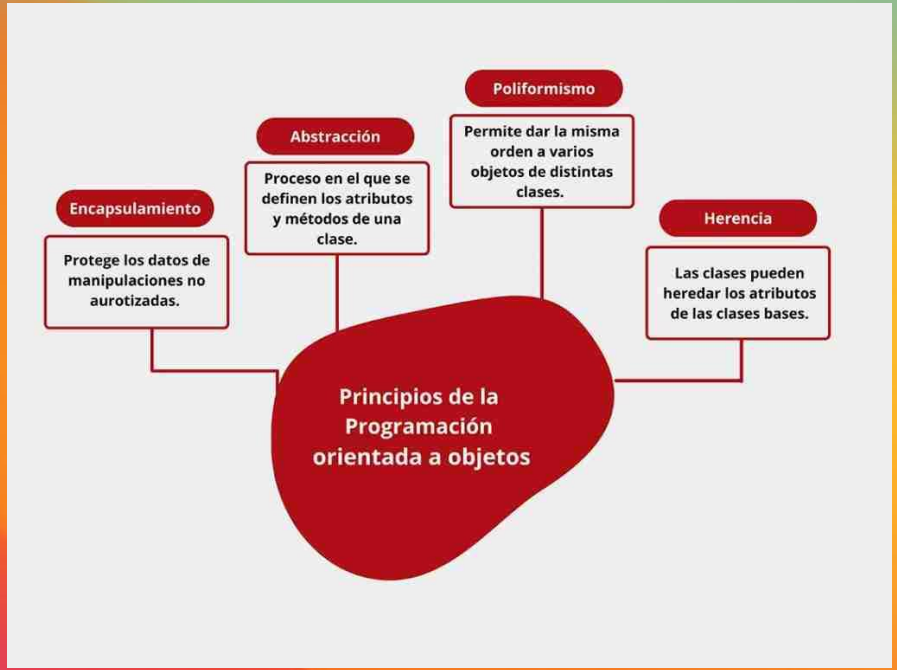
¿A que se refiere cuando se habla de POO?

es un paradigma de programación, esto es, un modelo o un estilo de programación que proporciona unas guías acerca de cómo trabajar con él y que está basado en el concepto de clases y objetos.



¿Cuáles son los 4 componentes que componen POO?

- **FUNDAMENTALES los cuales son:**
- Encapsulación.
- Abstracción.
- Herencia.
- Polimorfismo.



¿Cuáles son los pilares de POO?.

- Pilares de la POO. La programación orientada a objetos como paradigma, se basa en cuatro pilares fundamentales: abstracción , encapsulamiento, polimorfismo y herencia.

¿Qué es Encapsulamiento?

Decimos que el encapsulamiento en la programación orientada a objetos es cuando limitamos el acceso o damos un acceso restringido de una propiedad a los elementos que necesita un miembro y no a ninguno más.

un ejemplo

ejemplo muy común de encapsulamiento son los getters y setters de las propiedades dentro de una clase. Por defecto nos dan el valor “normal” pero podemos modificarlos para que cambie.

```
private decimal _velocidadActual { get; set; }
public decimal VelocidadActual
{
    get{
        return _velocidadActual + 2;
    }
    set{
        _velocidadActual = value;
    }
}
```

¿Qué es Abstracción?

se refiere a la capacidad de un lenguaje de programación para representar y manipular conceptos abstractos en el código.

Ejemplo

```java
abstract class Forma {
    // Atributos
    protected String color;

    // Constructor
    public Forma(String color) {
        this.color = color;
    }

    // Métodos abstractos
    public abstract double area();
    public abstract double perimetro();
}

class Rectangulo extends Forma {
    // Atributos
    private double largo;
    private double ancho;

    // Constructor
    public Rectangulo(double largo, double ancho, String color) {
        super(color);
        this.largo = largo;
        this.ancho = ancho;
    }

    // Implementación de métodos abstractos
    public double area() {
        return largo * ancho;
    }

    public double perimetro() {
        return 2 * (largo + ancho);
    }
}

class Circulo extends Forma {
    // Atributos
    private double radio;

    // Constructor
    public Circulo(double radio, String color) {
        super(color);
        this.radio = radio;
    }

    // Implementación de métodos abstractos
    public double area() {
        return Math.PI * Math.pow(radio, 2);
    }

    public double perimetro() {
        return 2 * Math.PI * radio;
    }
}

// Atributos
private double lado1;
private double lado2;
private double lado3;

// Constructor
public Triangulo(double lado1, double lado2, double lado3, String color)
    super(color);
    this.lado1 = lado1;
    this.lado2 = lado2;
    this.lado3 = lado3;
}

// Implementación de métodos abstractos
public double area() {
    double s = (lado1 + lado2 + lado3) / 2;
    return Math.sqrt(s * (s - lado1) * (s - lado2) * (s - lado3));
}

public double perimetro() {
    return lado1 + lado2 + lado3;
}
```

¿Qué es Herencia?

La herencia permite que se puedan definir nuevas clases basadas de unas ya existentes a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación. Si una clase deriva de otra, esta hereda sus atributos y métodos y puede añadir nuevos atributos, métodos o redefinir los heredados.

Ejemplo:

```
//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());

    }
}
```

¿Qué es polimorfismo?

es la capacidad que tienen ciertos lenguajes para hacer que, al enviar el mismo mensaje (o, en otras palabras, invocar al mismo método) desde distintos objetos, cada uno de esos objetos pueda responder a ese mensaje (o a esa invocación) de forma distinta.

Ejemplo:

```
class Animal {
  public void makeSound() {
    System.out.println("Grr...");
  }
}
class Cat extends Animal {
  public void makeSound() {
    System.out.println("Meow");
  }
}
class Dog extends Animal {
  public void makeSound() {
    System.out.println("Woof");
  }
}
```

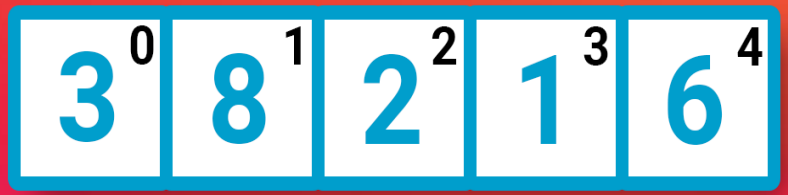
```
public static void main(String[ ] args) {
  Animal a = new Dog();
  Animal b = new Cat();
}
```

```
a.makeSound();
//Outputs "Woof"

b.makeSound();
//Outputs "Meow"
```

Que es un ARRAY?

Los arrays son variables estructuradas, donde cada elemento se almacena de forma consecutiva en memoria. Las cadenas de caracteres son declaradas en C como arrays de caracteres y permiten la utilización de un cierto número de notaciones y de funciones especiales.



¿Qué son los paquetes en JAVA?

Los paquetes son el **mecanismo que usa Java para facilitar la modularidad del código**. Un paquete puede contener una o más definiciones de interfaces y clases, distribuyéndose habitualmente como un archivo

¿Cómo se define una clase main en JAVA?

El método main de momento lo situaremos en una clase independiente destinada exclusivamente a contener este método, aunque esto no es obligatorio: la clase con el método main podría tratarse como una clase más y el método main como un método más. Nosotros preferiremos diferenciarlo por motivos didácticos.

Ejemplo:

```
// Clase principal iniciadora del programa ejemplo aprenderaprogramar.com
public class TestDeposito {
   public static void main (String [ ] args) {

      //Aquí las instrucciones de inicio y control del programa

      System.out.println ("Empezamos la ejecución del programa");

   } //Cierre del main
} //Cierre de la clase
```