



El empleo
es de todos

Mintrabajo

APLICAR PROGRAMACIÓN ORIENTADA A OBJETO CON C#

José Fernando Galindo Suárez





PROGRAMACIÓN ORIENTADA A OBJETO CON C#

2

PROGRAMACIÓN ORIENTADA A OBJETO CON C#

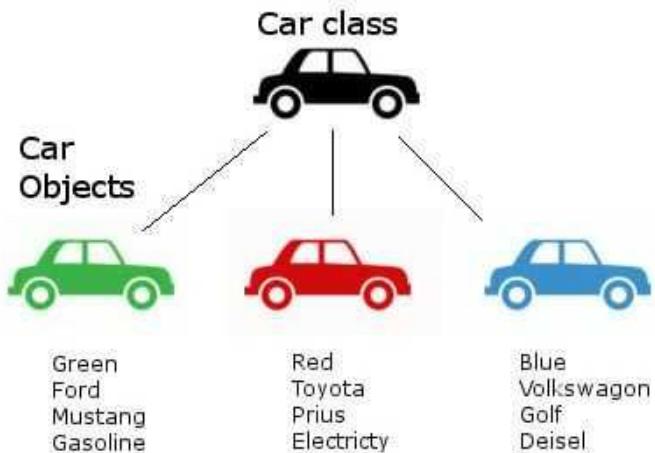
OBJETIVOS

Después de completar esta lección usted estará en la capacidad de:

- **Entender la programación orientada a objetos.**
- **Definir los conceptos básicos de la programación basada en objetos**
 - **Clases, objetos**
 - **Miembros (atributos, métodos)**
 - **Abstracción y ocultación de información**
- **Declarar una clase con características(atributos) y comportamientos (métodos).**
- **Crear objetos de una clase, modificando o restringiendo el acceso en su estado y comportamiento.**



¿Qué es un objeto?



- **Los objetos son representaciones (simples/complejas) (reales/imaginarias) de cosas: reloj, avión, coche.**
- **No todo puede ser considerado como un objeto, algunas cosas son simplemente características atributos de los objetos: color, velocidad, nombre**

¿Qué es un objeto?



Abstracción funcional

Hay cosas que sabemos que los coches hacen pero no cómo lo hacen:

- avanzar
- parar
- girar a la derecha
- girar a la izquierda

Abstracción de datos

Un coche tiene además ciertos atributos:

- color
- velocidad
- tamaño
- etc.

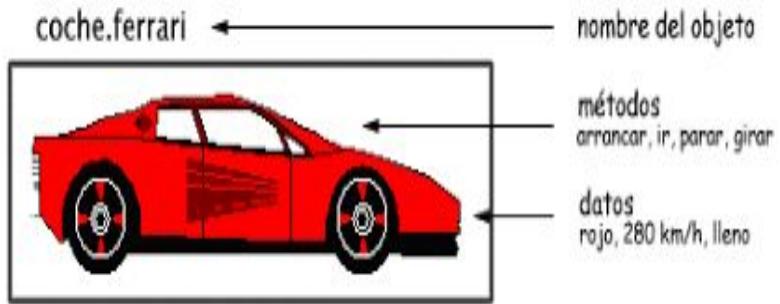
¿Qué es un objeto?



Es una forma de agrupar un conjunto de datos (estado) y de funcionalidad (comportamiento) en un mismo bloque de código que luego puede ser referenciado desde otras partes de un programa

La clase a la que pertenece el objeto puede considerarse como un nuevo tipo de datos

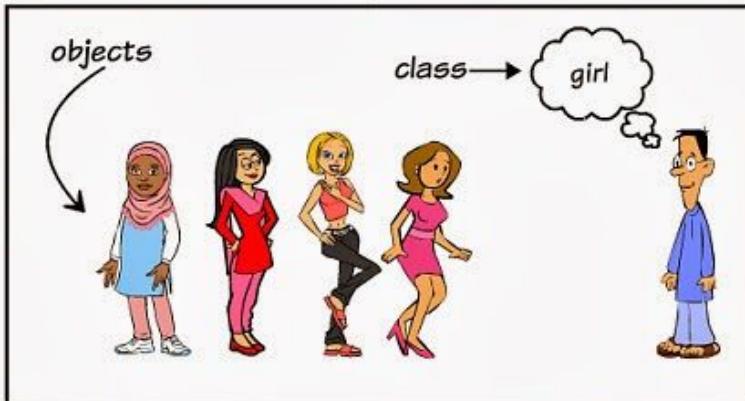
¿Qué es un objeto?



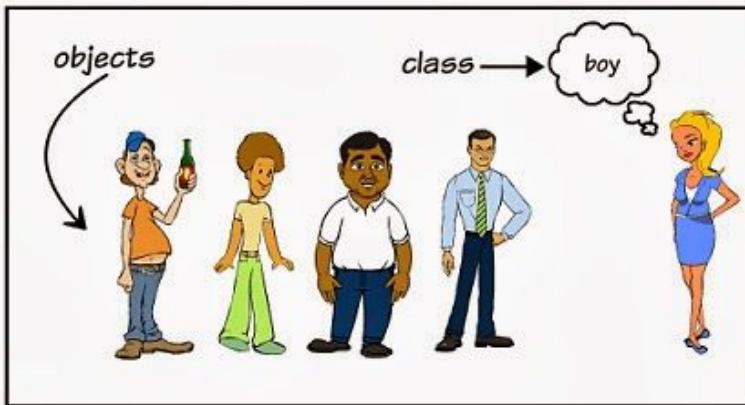
Los objetos permiten tener un control total sobre ‘quién’ o ‘qué’ puede acceder a sus variables y métodos, es decir, pueden tener componentes públicos, a los que podrán acceder otros objetos, o componentes privados, a los que únicamente puede acceder él.

Los componentes pueden ser tanto las variables como los métodos de ese objeto.

¿Qué es un objeto?



La programación orientada a objetos (Object Oriented Programming OOP) es un modelo de lenguaje de programación organizado por objetos constituidos por datos y funciones, entre los cuales se pueden crear relaciones como herencia, cohesión, abstracción, polimorfismo y encapsulamiento.



Esto permite que haya una gran flexibilidad y se puedan crear objetos que pueden heredarse y transmitirse sin necesidad de ser modificados continuamente.

JERARQUÍA DE COMPOSICIÓN



El objeto está compuesto por otros objetos con comportamientos distintos.

Esto sirve para representar uno varios objetos que están dentro de otro que los contiene.

DEFINICIÓN DE ABSTRACCIÓN



Nos da una visión simplificada de una realidad de la que sólo consideramos determinados aspectos esenciales:

- ¿qué entendemos por ... ?
- ¿... color de un semáforo?
- ¿... estado de una cuenta bancaria?
- ¿... estado de una bombilla?
- ¿qué necesitamos conocer de un coche para utilizarlo?

DEFINICIÓN DE ABSTRACCIÓN



La abstracción como técnica de programación



La programación es una tarea compleja

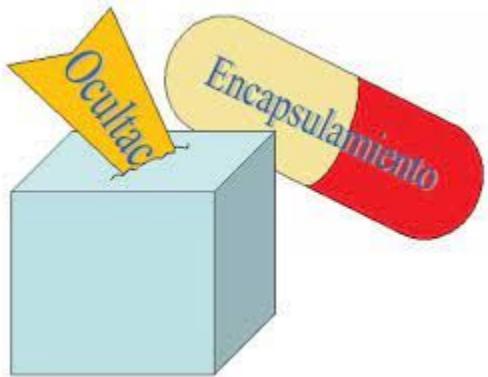
.

Mediante la abstracción es posible elaborar software que permita solucionar problemas cada vez más grandes.

DEFINICIÓN DE ENCAPSULAMIENTO



Proceso de ocultamiento de todos los detalles de una entidad que no contribuyen a sus características esenciales.



Abstracción → nos centramos en la visión externa.
Encapsulamiento → nos centramos en la visión interna.
El acceso a los datos y las operaciones se realiza mediante una interfaz bien definida.

MODIFICADORES DE ACCESO



Accesibilidad

- **public:** Un método o campo público puede ser accedido desde fuera de la clase.
- **private:** El modificador de acceso privado especifica campos y métodos de una clase que no son accesible fuera de la unidad donde se declara la clase.
- **protected:** El modificador de acceso protegido se utiliza para indicar métodos y campos con visibilidad sólo en la clase actual y sus clases derivadas (o subclases)

Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◊	◊	protected
~	△	▲	package private
+	○	●	public

DEFINICIÓN DE CLASES

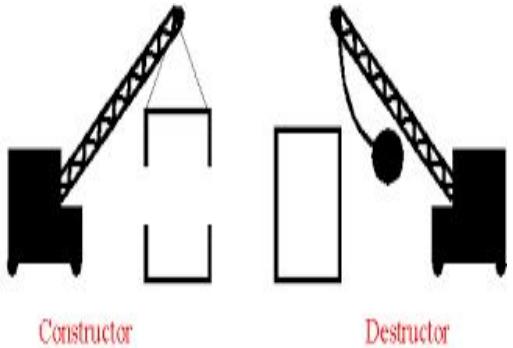


Una *Clase* es una estructura de software que especifica un determinado tipo de objeto.

Define la estructura y el comportamiento de los objetos que pertenecen a dicha clase.

- Después de declarar una clase pueden crear cualquier número de objetos de esa clase.
- Se dice que los objetos son *instancias* de las clases.

Constructor y destructor de una clase



Clase Construida
Press any key to continue . . .
Clase Destruida

- Un constructor es un método especial de una clase que tiene por objetivo inicializar campos de la clase (normalmente otros objetos).
- El destructor es otro método de la clase que será el último que se ejecute y tiene por objetivo liberar espacio de campos de la clase.

```
using System;
namespace Constructor
{
    /// <summary>
    /// Description of Clase1.
    /// </summary>
    public class Clase1
    {
        public Clase1()
        {
            Console.WriteLine("Clase Construida");
        }
        ~Clase1()
        {
            Console.WriteLine("Clase Destruida");
        }
    }
}
```

```
using System;
namespace Constructor
{
    class Program
    {
        public static void Main(string[] args)
        {
            Clase1 c1=new Clase1();
            c1=null;
            GC.Collect();

            // TODO: Implement Functionality Here

            Console.WriteLine("Press any key to continue . . .");
            Console.ReadKey(true);
        }
    }
}
```

DEFINICIÓN DE HERENCIA

Proceso mediante el cual una clase adquiere las propiedades de otra clase



- Permite definir una nueva clase o subclase a partir de otra clase o superclase.
- Una subclase incluye todo el comportamiento y especificación de sus antecesores.
- Las subclases redefinen la estructura y el comportamiento de sus superclases.
- La herencia permite reutilizar código

DEFINICIÓN DE POLIMORFISMO



El polimorfismo se refiere al hecho que un método adopte múltiples formas.

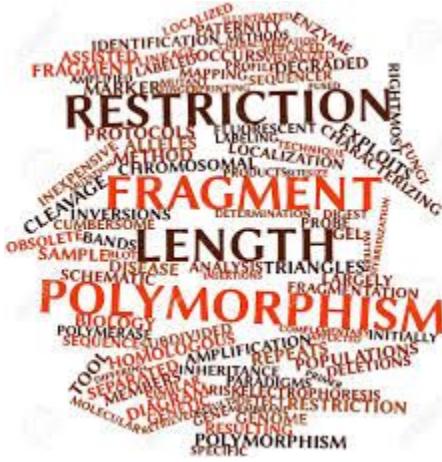
Esto se consigue por medio de la sobrecarga de métodos:
un mismo nombre de método para distintas funcionalidades.

$a = \text{Sumar}(c,d);$ $a = \text{Sumar}(c,d,5);$

Sobrecarga de operadores:

un mismo operador con distintas funcionalidades.

DEFINICIÓN DE POLIMORFISMO



En la sobrecarga de funciones se desarrollan distintas funciones con un mismo nombre pero distinto código.

- Las funciones que comparten un mismo nombre deben tener una relación en cuanto a su funcionalidad.
- Aunque comparten el mismo nombre, debentener distintos parámetros. Éstos pueden diferir en :
 - El número
 - El tipo
 - El orden
- El tipo del valor de retorno de una función no es válido como distinción.

RELACIONES ENTRE CLASE



**PRÁCTICA EN EL AMBIENTE
DE APRENDIZAJE**

EJEMPLO DE CLASES Y ARREGLOS DE CLASES



1	Juan	25	0
2	Pedro	3	1
3	Maria	4,5	2
4	Carlos	4	3
.	.	.	.

```
class Program
{
    public class Notas{ ①
        int ID;
        String Nombre;Decimal Nota; ②
        public void setId(string datos){ ③
            string[] words = datos.Split(';');
            ID=int.Parse(words[0]);
            Nombre=words[1];
            Nota=Convert.ToDecimal( words[2]);
        }
        ④ public void getNota(){Console.WriteLine("{0} {1} {2}",ID,NOMBRE,Nota);}
    }
    public static void Main(string[] args){
        Notas[] ar=new Notas[4]; ⑤
        ar[0]=new Notas(); ⑥
        ar[0].setId("1;Juan;2.5"); ⑦
        ar[1]=new Notas();
        ar[1].setId("2;Pedro;3");
        ar[2]=new Notas();
        ar[2].setId("3;Maria;4,5");
        ar[3]=new Notas();
        ar[3].setId("4;Carlos;4");
        for(int i=0;i<ar.Length;i++) ⑧
            ar[i].getNota(); ⑨
        Console.WriteLine("Oprima una tecla para continuar");
        Console.ReadLine();
    }
}
```

EJEMPLO DE CLASES Y ARREGLOS DE CLASES



```
class Program
{
    public class Notas{
        int ID;
        String Nombre;Decimal Nota;
        public void setId(string datos){
            string[] words = datos.Split(';');
            ID=int.Parse(words[0]);
            Nombre=words[1];
            Nota=Convert.To
        }
        public void getNota();
    }
    public static void Ma
        Notas[] ar=new Notas[4];
        ar[0]=new Notas();
        ar[0].setId("1;Juan");
        ar[1]=new Notas();
        ar[1].setId("2;Pedro");
        ar[2]=new Notas();
        ar[2].setId("3;Maria;4,5");
        ar[3]=new Notas();
        ar[3].setId("4;Carlos;4");
        for(int i=0;i<ar.Length;i++)
            ar[i].getNota();

        Console.WriteLine("Oprima una tecla para continuar");
        Console.ReadLine();
    }
}
```

1 Juan 25 0
2 Pedro 3 1
3 Maria 4,5 2
4 Carlos 4 3

Oprima una tecla para continuar

EJEMPLO DE CLASES GENÉRICAS EN C#



```
using System;
namespace ClasesGenericas
{
    class Program
    {
        public static Object[] objetos=new Object[4]; 1
        public static int i=0; 2

        public static Object getDatos(int i){ 4
            return (String) objetos[i];
        }
        public static void Agregar(Object obj){ 3
            objetos[i++]=obj;
        }
        public static void Main()
        {
            Agregar("Pedro"); 5
            Agregar("Juan");
            Agregar("Maria");
            for(int ii=0;ii<i;ii++) 6
                Console.WriteLine("{0}",getDatos(ii));
            Console.Write("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

```
Pedro
Juan
Maria
Press any key to continue . . .
```



EJEMPLO DE CLASES SELLADAS EN C#



```
using System;
namespace ClasesSelladas
{
    public class Parte1{ 1
        protected int partes;
        public void setPartes(int n){
            partes=n;
        }
    }
    public sealed class Parte2:Parte1{ 2
        public int getPartes(){
            return partes;
        }
    }
    public class Parte3:Parte2{ 3
        public int getPartes(){
            return partes;
        }
    }
    class Program
    {
        public static void Main(string[] args)
        {
            Console.Write("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

**SEALED NO PERMITE QUE
SE HEREDE LA CLASE Parte2**



EJEMPLO DE GETTER Y SETTER EN C#



```
class Student
{
    public int Age { get; set; }
}
```

```
class Student
{
    private int _age;

    public int GetAge()
    {
        return _age;
    }

    public void SetAge(int age)
    {
        _age = age;
    }
}
```

```
public int Age { get; private set; }
```

int Age {get; }

```
private int yearBorn;
public int Age
{
    get { return DateTime.Now.Year - yearBorn; }
    set { yearBorn = value; }
}
```

```
Person p = new Person();
p.Name = "Geetha";
p.Age = 1986;
Console.WriteLine("Name = {0}",p.Name);
Console.WriteLine("Age = {0}",p.Age);
```

EJEMPLO DE DELEGADOS EN C#



```
using System;

namespace FuncionesAnonimas
{
    public class CMuestra{ ③
        public static void Muestra(string cual){
            Console.WriteLine("Estoy mostrando esto:{0}",cual);
        }
    }
    public class COoperaciones{ ④
        public static int Suma(int numero1,int numero2){
            return (numero1+numero2);
        }
    }
    class Program
    {
        public static void Main(string[] args)
        {
            var Donde=new DMuestra(CMuestra.Muestra); ⑤
            var Donde1=new DOtraMuestra(COoperaciones.Suma); ⑥
            Donde("Hola Mundo"); ⑦
            Console.WriteLine("La suma es={0}",Donde1(8,5));
            // TODO: Implement Functionality Here

            Console.Write("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
        public delegate void DMuestra(string que); ⑧
        public delegate int DOtraMuestra(int n1,int n2); ⑨
    }
}
```



```
Estoy mostrando esto:Hola Mundo
La suma es=13
Press any key to continue . . .
```

EJEMPLO DE FUNCIONES LAMBDA EN C#



```
using System;
namespace FuncionesAnonimas
{
    public class CMuestra{
        public static void Muestra(string cual){
            Console.WriteLine("Estoy mostrando esto:{0}",cual);
        }
    }
    public class COoperaciones{
        public static int Suma(int numero1,int numero2){ ①
            return (numero1+numero2);
        }
    }
    class Program
    {
        public static void Main(string[] args)
        {
            var Donde=new DMuestra(CMuestra.Muestra); ②
            Suma Donde1=new Suma((n1, n2) => n1+n2); ③
            Donde("Hola Mundo");
            Console.WriteLine("La suma es={0}",Donde1(8,5)); ④
            // TODO: Implement Functionality Here

            Console.Write("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
        public delegate void DMuestra(string que);
        public delegate int Suma(int nu1,int nu2); ⑤
    }
}
```



```
Estoy mostrando esto:Hola Mundo
La suma es=13
Press any key to continue . . .
```

RELACIONES ENTRE CLASE

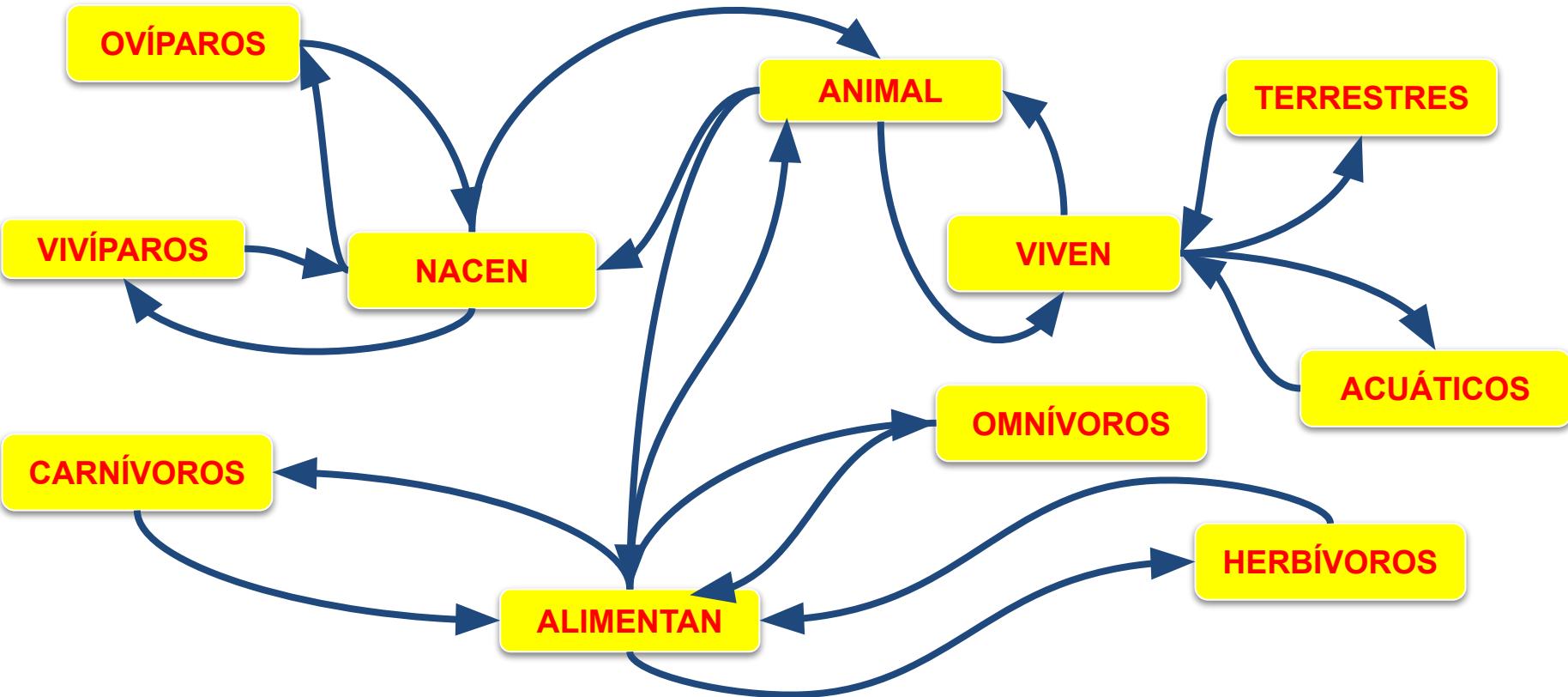


**PRÁCTICA EN EL AMBIENTE
DE APRENDIZAJE**

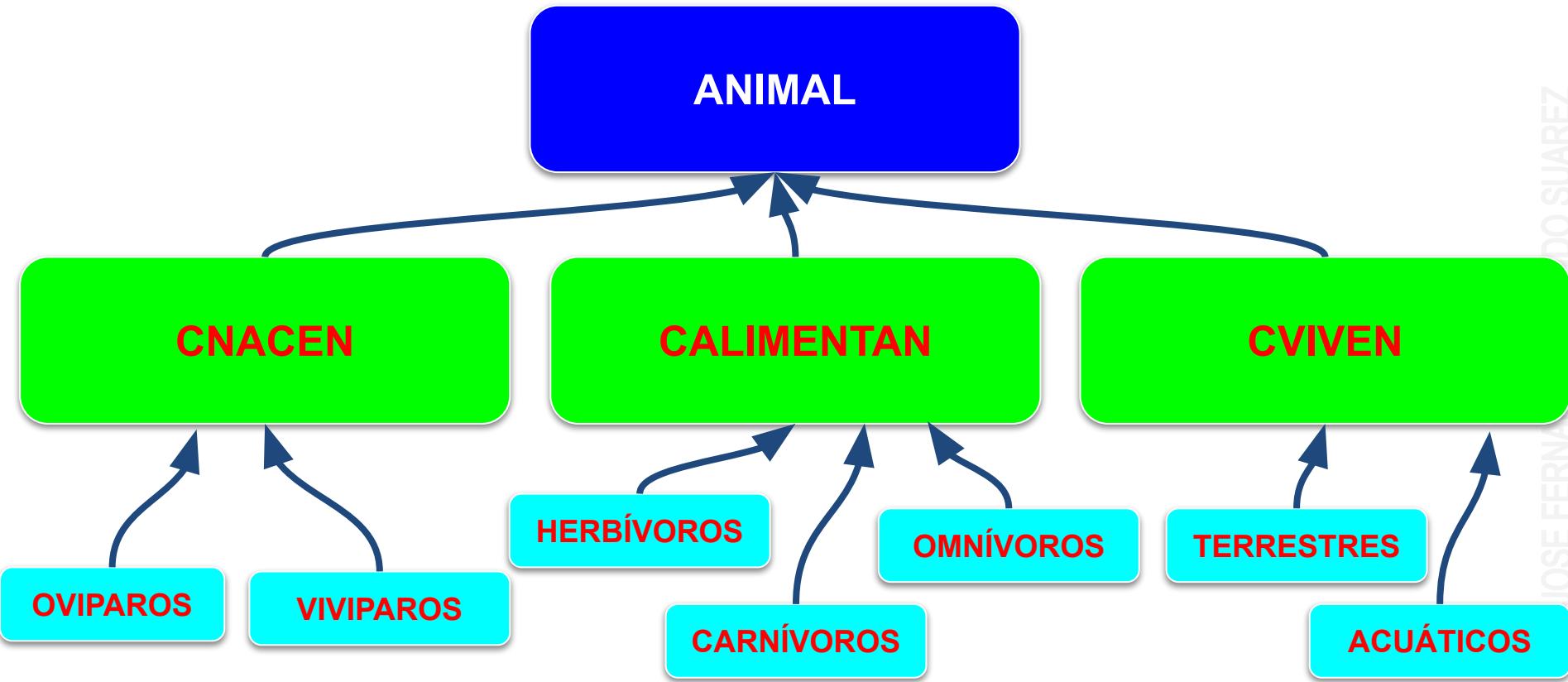
HERENCIA ¿ES UN?



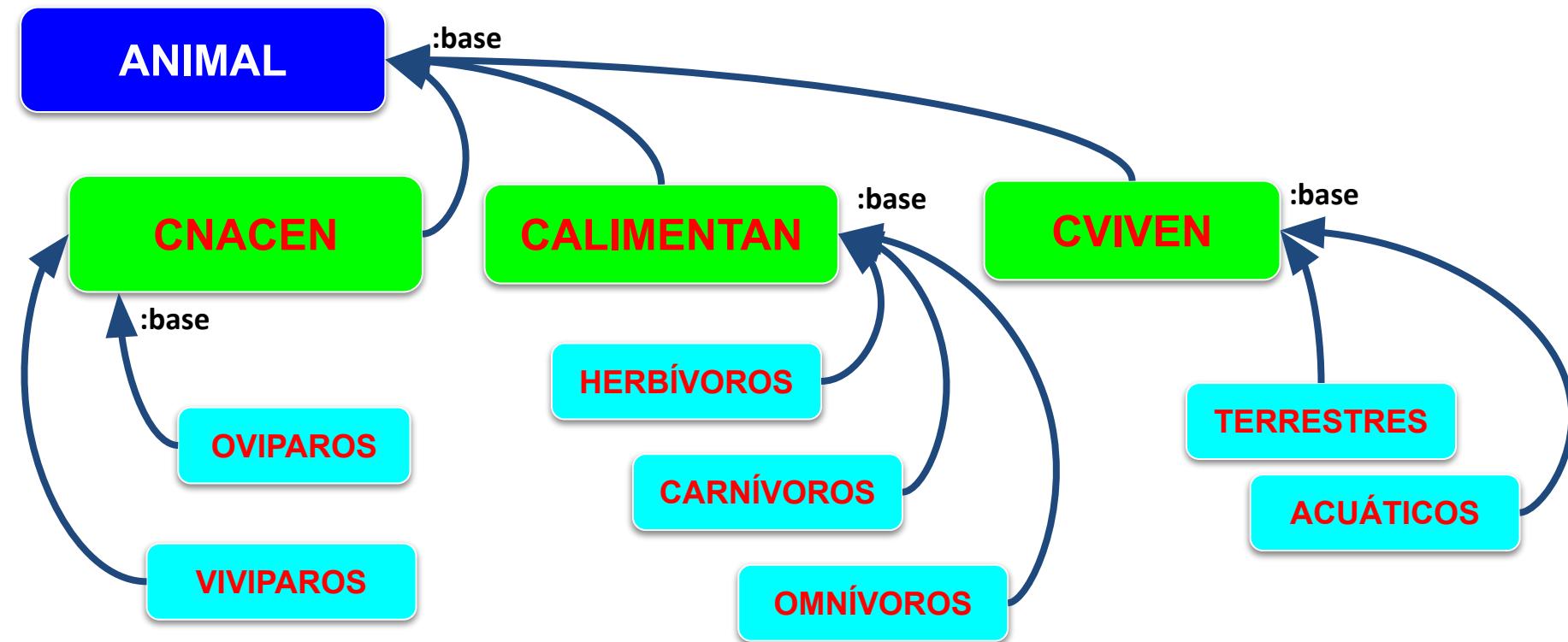
HERENCIA ¿ES UN?



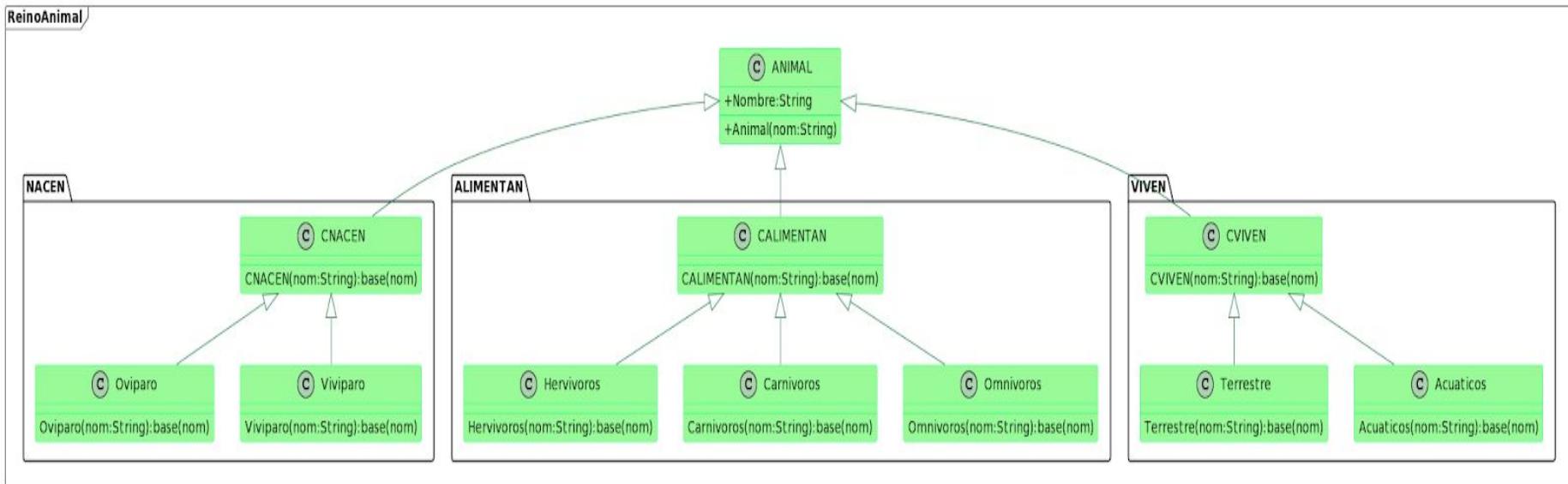
HERENCIA ¿ES UN?



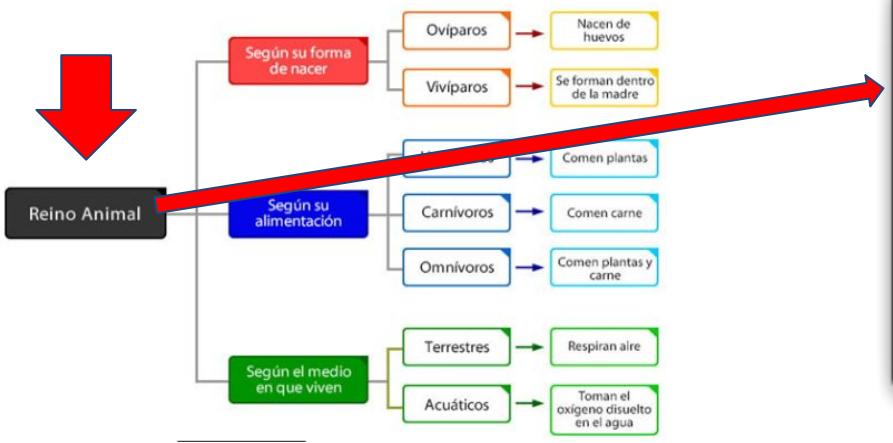
HERENCIA LLAMANDO AL CONSTRUCTOR DE LA SUPERCLASE



HERENCIA ¿ES UN?



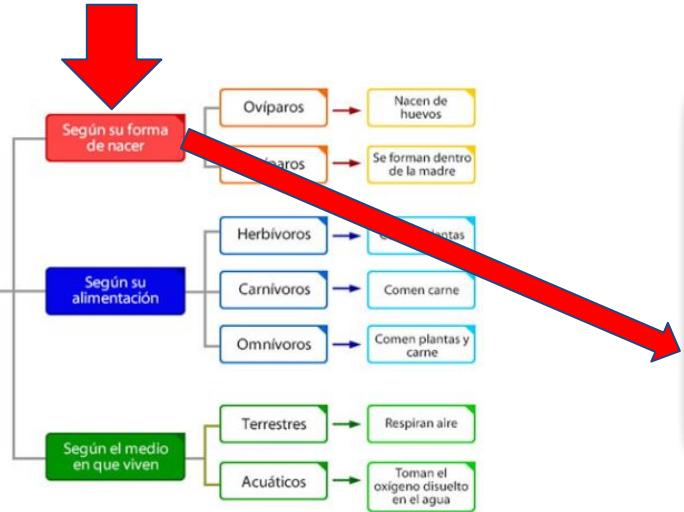
DEFINICIÓN DE CLASE ANIMAL



```
public class Animal{  
    public string Nombre;  
    public Animal(String nom){  
        Nombre=nom;  
    }  
}
```



DEFINICIÓN DE CLASE CNACEN CON HERENCIA



```

public class CNACEN:Animal{
    public CNACEN(String nom):base(nom){
    }
}
  
```



DEFINICIÓN DE CLASE OVÍPARO CON HERENCIA



```
public class Oviparo:CNACEN{  
    public Oviparo(String nom):base(nom){  
    }  
    public void getNombre(){  
        Console.WriteLine(Nombre);  
    }  
}
```



DEFINICIÓN DE CLASE VIVÍPARO CON HERENCIA



```
public class Viviparo:CNACEN{
    public Viviparo(String nom) :base(nom){
    }
    public void getNombre(){
        Console.WriteLine(Nombre);
    }
}
```



DEFINICIÓN DE CLASE CALIMENTAN CON HERENCIA



```
public class CALIMENTAN:Animal{  
    public CALIMENTAN(String nom):base(nom){  
    }  
}
```



DEFINICIÓN DE CLASE HERVIVOROS CON HERENCIA



```
public class Hervivoros:CALIMENTAN{
    public Hervivoros(String nom):base(nom){
    }

    public void getNombre(){
        Console.WriteLine(Nombre);
    }
}
```



DEFINICIÓN DE CLASE CARNÍVOROS CON HERENCIA



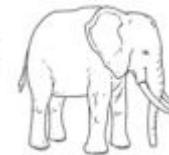
```
public class Carnivoros:CALIMENTAN{  
    public Carnivoros(String nom):base(nom){  
    }  
    public void getNombre(){  
        Console.WriteLine(Nombre);  
    }  
}
```



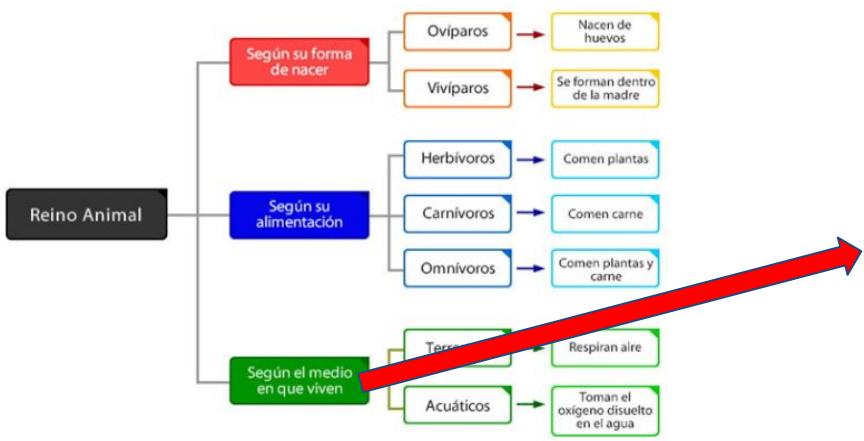
DEFINICIÓN DE CLASE OMNÍVOROS CON HERENCIA



```
public class Omnivoros : CALIMENTAN{  
    public Omnivoros(String nom) : base(nom){  
    }  
    public void getNombre(){  
        Console.WriteLine(Nombre);  
    }  
}
```



DEFINICIÓN DE CLASE CVIVEN CON HERENCIA

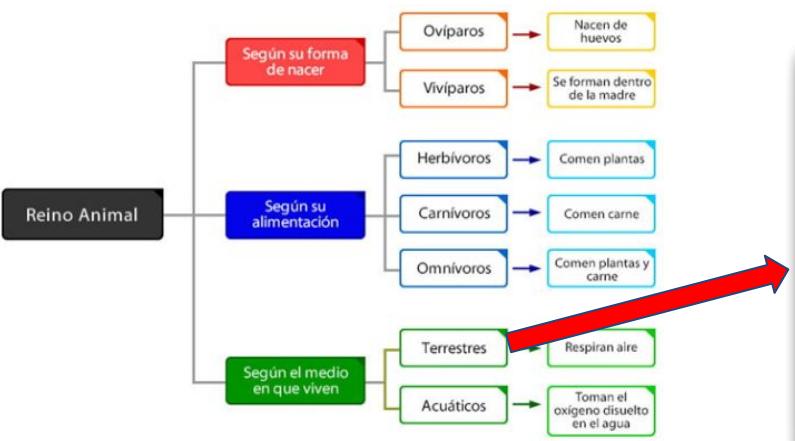
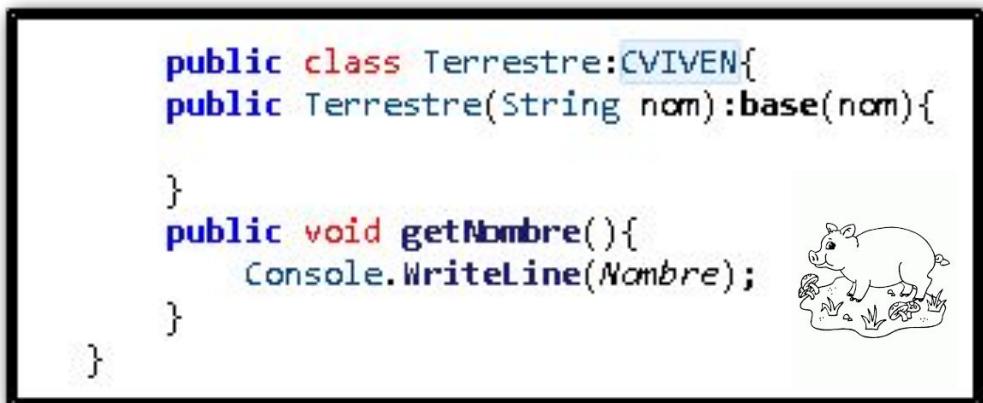


```

public class CVIVEN:Animal{
    public CVIVEN(String nom):base(nom){
        }
    }
  
```



DEFINICIÓN DE CLASE TERRESTRE CON HERENCIA

```

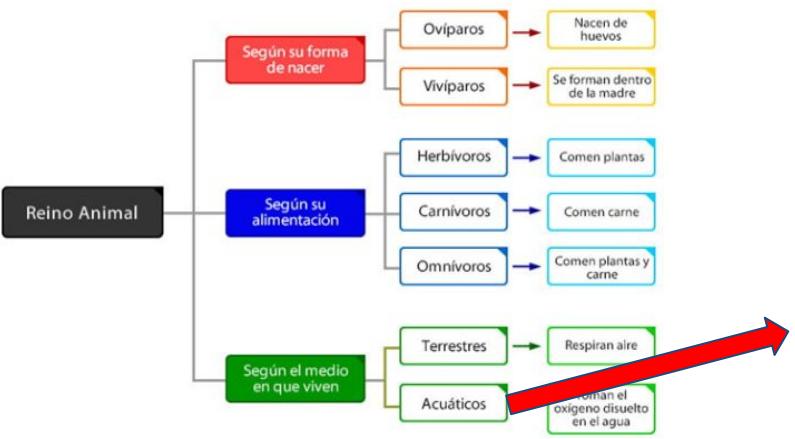
public class Terrestre:CVIVEN{
    public Terrestre(String nom):base(nom){
    }

    public void getNombre(){
        Console.WriteLine(Nombre);
    }
}
  
```

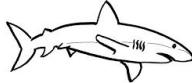
The code defines a class named `Terrestre` that implements the interface `CVIVEN`. It has a constructor that takes a name and a method `getNombre` that prints the name to the console. A red arrow points from the `Terrestres` node in the classification tree to the `Terrestre` class definition.



DEFINICIÓN DE CLASE ACUÁTICOS CON HERENCIA



```
public class Acuaticos:CVIVEN{
    public Acuaticos(String nom):base(nom){
    }
    public void getNombre(){
        Console.WriteLine(Nombre);
    }
}
```



INSTANCIAR CLASES CON HERENCIA

```
public static void Main(string[] args)
{
    // TODO: Implement Functionality Here
    Oviparo mioviparo=new Oviparo("Nacen de huevos"); 1
    mioviparo.getNombre();
    Viviparo miviviparo=new Viviparo("Nacen dentro del vientre de la madre");
    miviviparo.getNombre(); 2
    Hervivoros miHervivoros=new Hervivoros("Comen plantas");
    miHervivoros.getNombre(); 3
    Carnivoros miCarnivoros=new Carnivoros("Comen carne");
    miCarnivoros.getNombre(); 4
    Omnivoros miOmnivoros=new Omnivoros("Comen carne y plantas");
    miOmnivoros.getNombre(); 5
    Terrestre miTerrestre=new Terrestre("Viven en la tierra");
    miTerrestre.getNombre(); 6
    Acuaticos miAcuaticos=new Acuaticos("Viven en el agua");
    miAcuaticos.getNombre(); 7

    Console.WriteLine("Press any key to continue . . . ");
    Console.ReadKey(true);
}
```



INSTANCIAR CLASES CON HERENCIA

Nacen de huevos 1

Nacen dentro del vientre de la madre 2

Comen plantas 3

Comen carne 4

Comen carne y plantas 5

Viven en la tierra 6

Viven en el agua 7

Press any key to continue . . . -



¿QUÉ PODEMOS HACER PARA OBTENER ESTA SALIDA?



Nacen de huevos
Nacen dentro del vientre de la madre
Comen plantas
Comen carne
Comen carne y plantas
~~Viven en la tierra~~
Viven en el agua los Acuaticos
Oprima una tecla para continuar...



Nota:

Utilice un atributo en la clase Animal que almacene el tipo de animal y se muestre desde la clase Acuáticos como se muestra en la figura.

Ejemplo: Acuaticos miacuaticos=new Acuaticos("Viven en el agua", "Acuaticos");



RELACIONES ENTRE CLASE

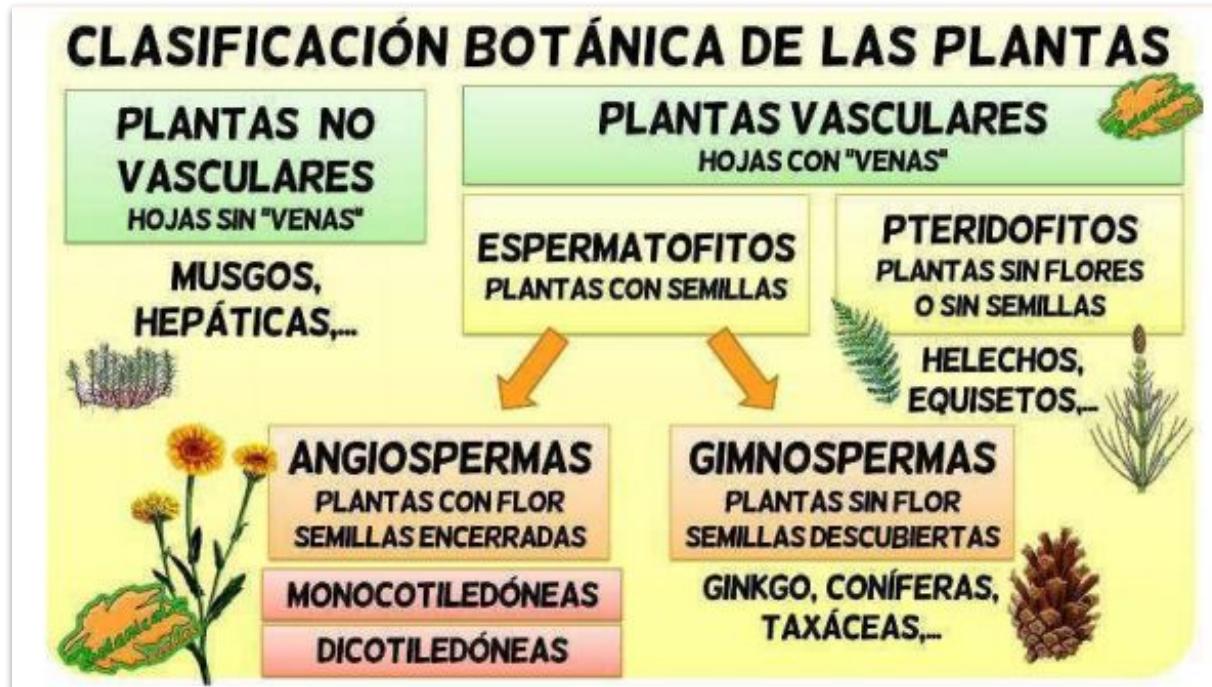


EVIDENCIA DE APRENDIZAJE

TAREA PARA REALIZAR AUTÓNOMAMENTE

Realizar las clases aplicando herencia

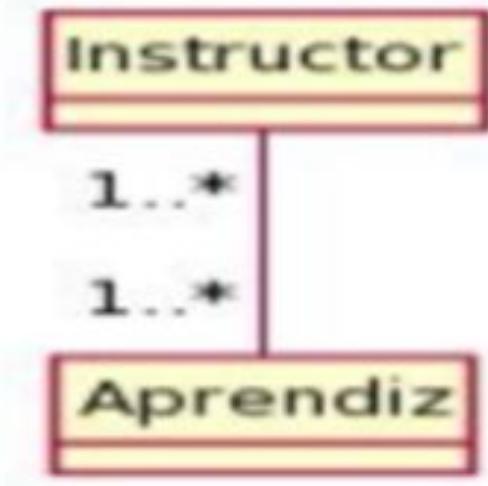
E
V
I
D
E
N
C
I
A



Nota:

Construya las clases de acuerdo a la gráfica presentada y utilice herencia, encapsulamiento y constructores.
No olvide guardar una copia en su portafolio de evidencias.

RELACIONES ENTRE CLASE ASOCIACIÓN



- La asociación es una relación donde los objetos tienen su propio ciclo de vida y no hay propietario.
- La frase para comprobar una relación de este tipo es A es una parte de B.

RELACIONES ENTRE CLASE

DEPENDENCIA



- La dependencia entre dos clases declara que una de ellas necesita conocer acerca de la clase a la que utilizará.

Por lo general se denomina a esta relación como la más débil. El tiempo establecido para esta relación es corto.

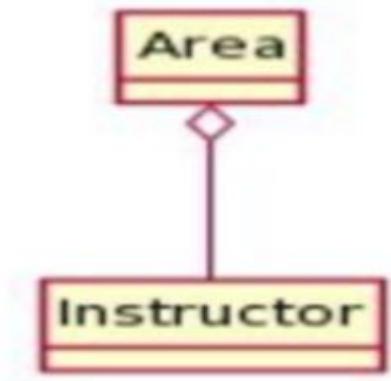
El conocimiento entre las clases es de una sola vía, esto significa que solamente la clase que utiliza a la otra tiene conocimiento de ella.

La frase para determinar esta relación es clase A *utiliza a la clase B*.

RELACIONES ENTRE CLASE AGREGACIÓN



- La agregación es una forma especializada de asociación donde todos los objetos tienen su propia existencia, pero hay propiedad y los objetos secundarios no pueden pertenecer a otro objeto principal.

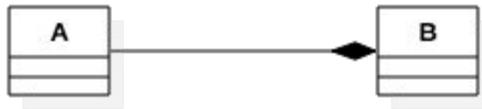


Un ejemplo: Área e Instructor. Un solo Instructor no puede pertenecer a múltiples Áreas, pero si eliminamos el Área, el objeto Instructor no se destruirá. Podemos pensarla como una relación “tiene una”.

RELACIONES ENTRE CLASE COMPOSICIÓN



COMPOSICIÓN



- La composición es una forma especializada de Agregación donde el objeto secundario no tiene su ciclo de vida propio y si se elimina el objeto principal, también se eliminarán todos los objetos secundarios.

La relación entre Preguntas y Opciones. Las preguntas individuales pueden tener múltiples opciones y la opción no puede pertenecer a otra pregunta.

RELACIONES ENTRE CLASE

AGREGACION



```
using System;
namespace RelacionComposicion
{
    class Factura{...} ①
    class FacturaDetalle{...} ②

    class Program{
        public static void Main(string[] args)
        {
            Factura fac=new Factura(); ③
            fac.addDetalle(new FacturaDetalle("Maria")); ④
            fac.addDetalle(new FacturaDetalle("Pablo"));
            fac.addDetalle(new FacturaDetalle("Luis"));
            fac.addDetalle(new FacturaDetalle("Elsa"));
            fac.addDetalle(new FacturaDetalle("Rosa"));
            fac.mostrarDetalle(); ⑤
            // TODO: Implement Functionality Here

            Console.WriteLine("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

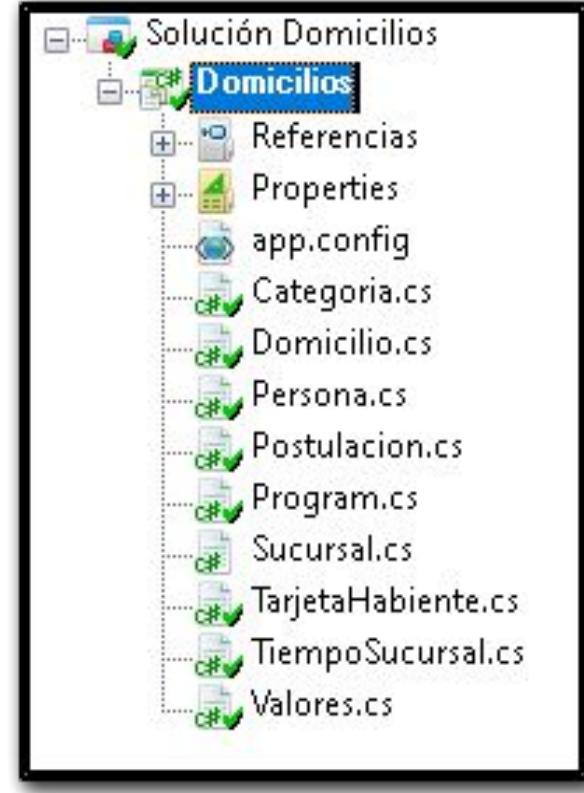
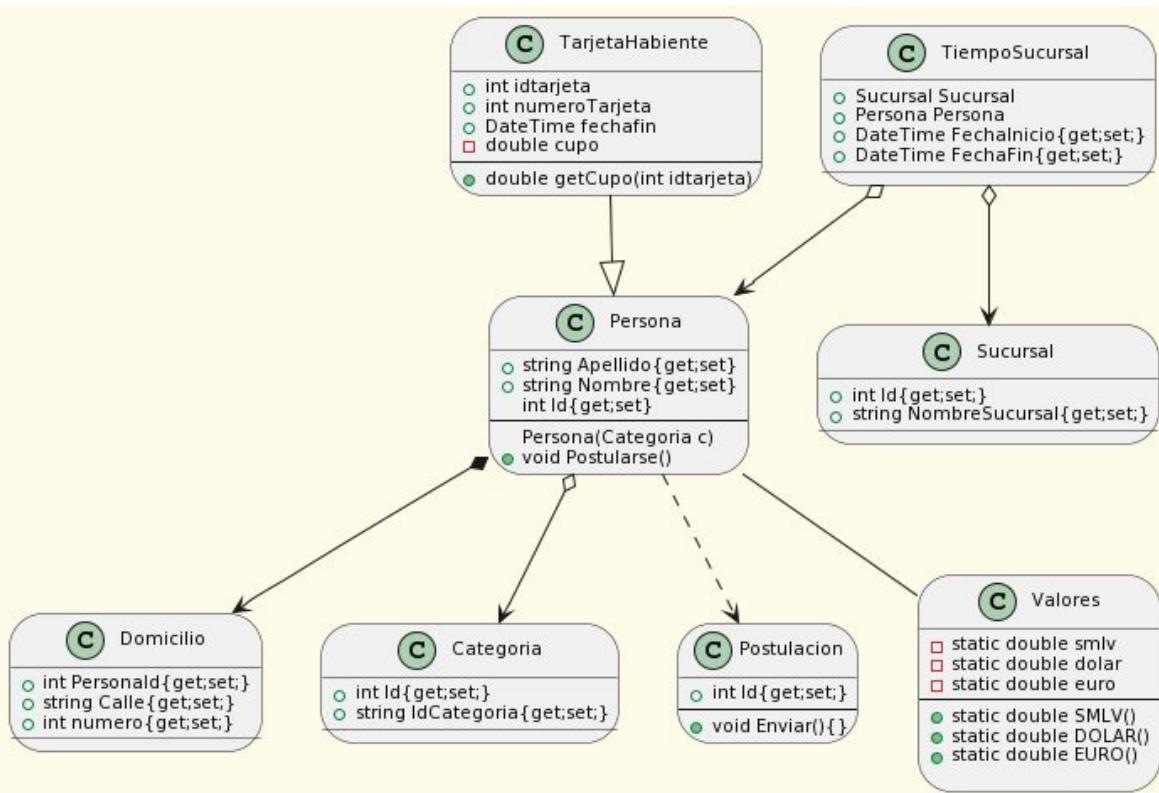
```
class Factura{
    1   public int i=0;
    private Object[] colDetalle=new Object[1000];
    public Factura(){
        2   i=0;
    }
    3   public void addDetalle(FacturaDetalle detalle){
        colDetalle[i++]=detalle.Nombre; ⑤
    }
    6   public void mostrarDetalle(){
        for(int ele=0; ele<this.i;ele++)
            Console.WriteLine("{0}",colDetalle[ele]);
    }
}
class FacturaDetalle{ ⑦
    public string Nombre;
    public FacturaDetalle(string Que){ ⑧
        Nombre=Que; ⑨
    }
}
```

RELACIONES ENTRE CLASE

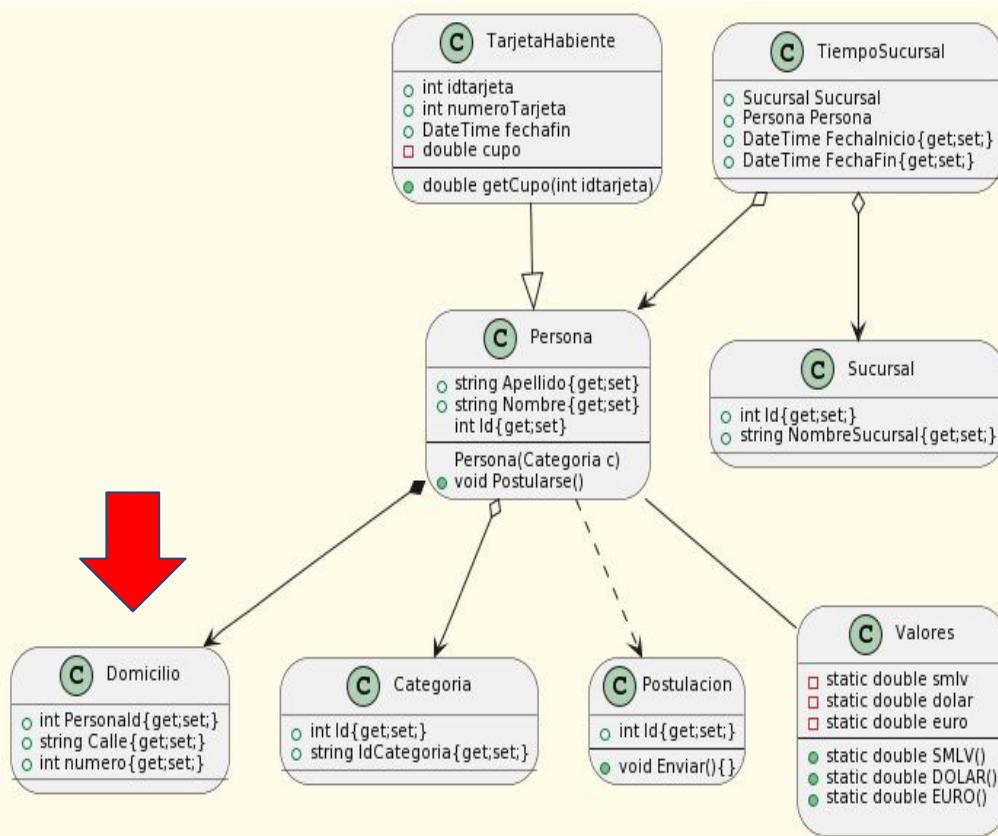


**PRÁCTICA EN EL AMBIENTE
DE APRENDIZAJE**

RELACIONES ENTRE CLASE

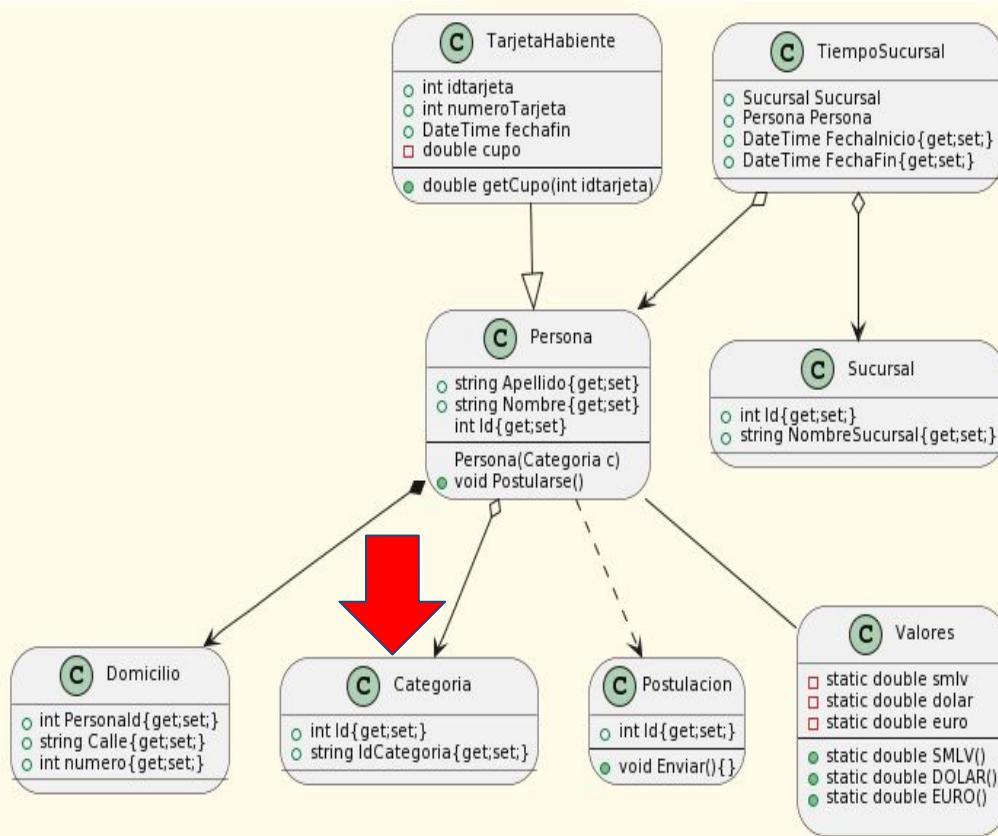


RELACIONES ENTRE CLASE



```
using System;  
  
namespace Domicilios  
{  
  
    public class Domicilio  
    {  
  
        public int PersonaId{get;set;}  
        public string Calle{get;set;}  
        public int numero{get;set;}  
  
    }  
}
```

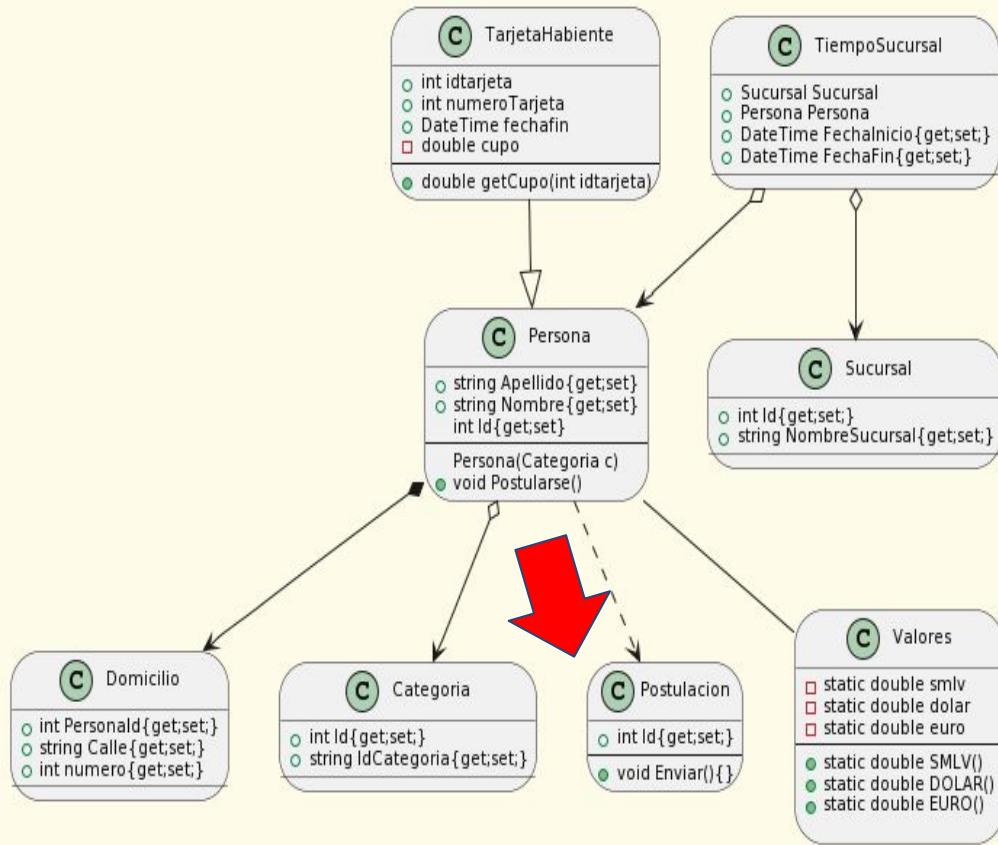
RELACIONES ENTRE CLASE



```
using System;

namespace Domicilios
{
    public class Categoría
    {
        public int Id{get;set;}
        public string IdCategoria{get;set;}
        public Categoría()
        {
            Console.WriteLine("Creando Categoría...");
        }
        ~Categoría(){Console.WriteLine("Saliendo de Categoría");}
    }
}
```

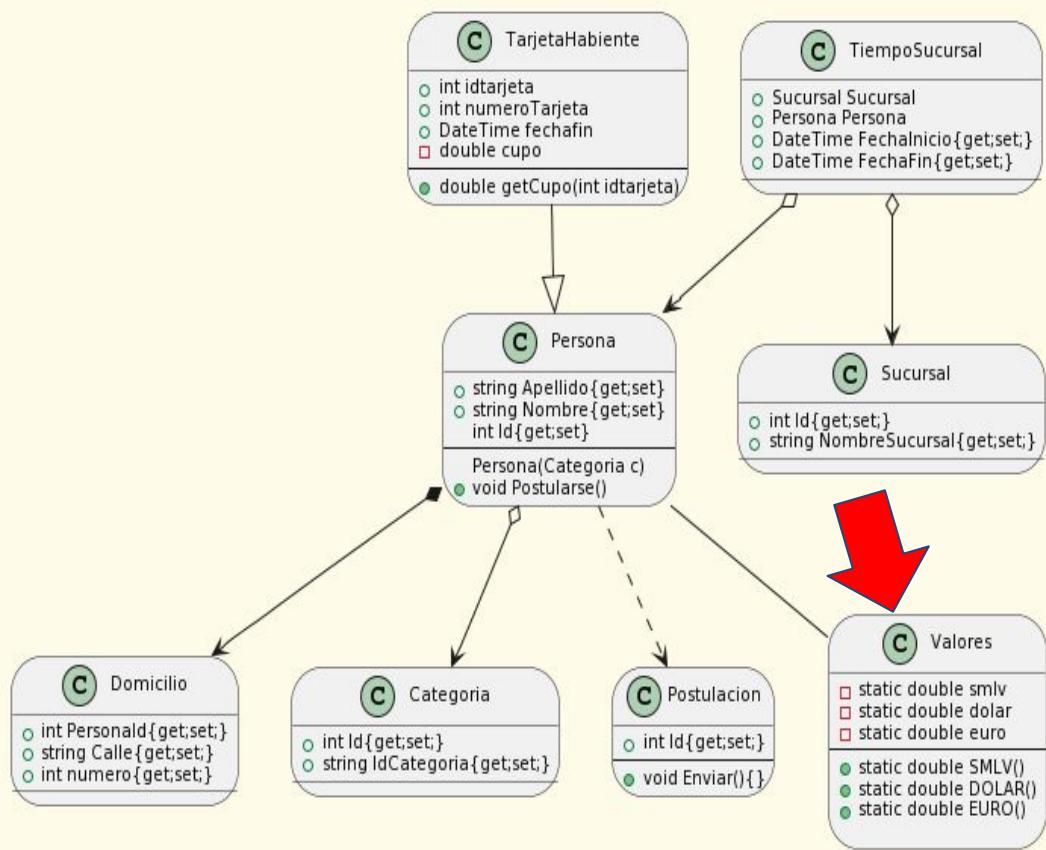
RELACIONES ENTRE CLASE



```
using System;
namespace Domicilios
{
    public class Postulacion
    {
        public int Id{get;set;}
        1 public Postulacion(){Console.WriteLine("Creando Postulacion");}
        2 ~Postulacion(){Console.WriteLine("Saliendo Postulacion");}
        3 public void Enviar(){Console.WriteLine("Enviando Postulacion");}
    }
}
```

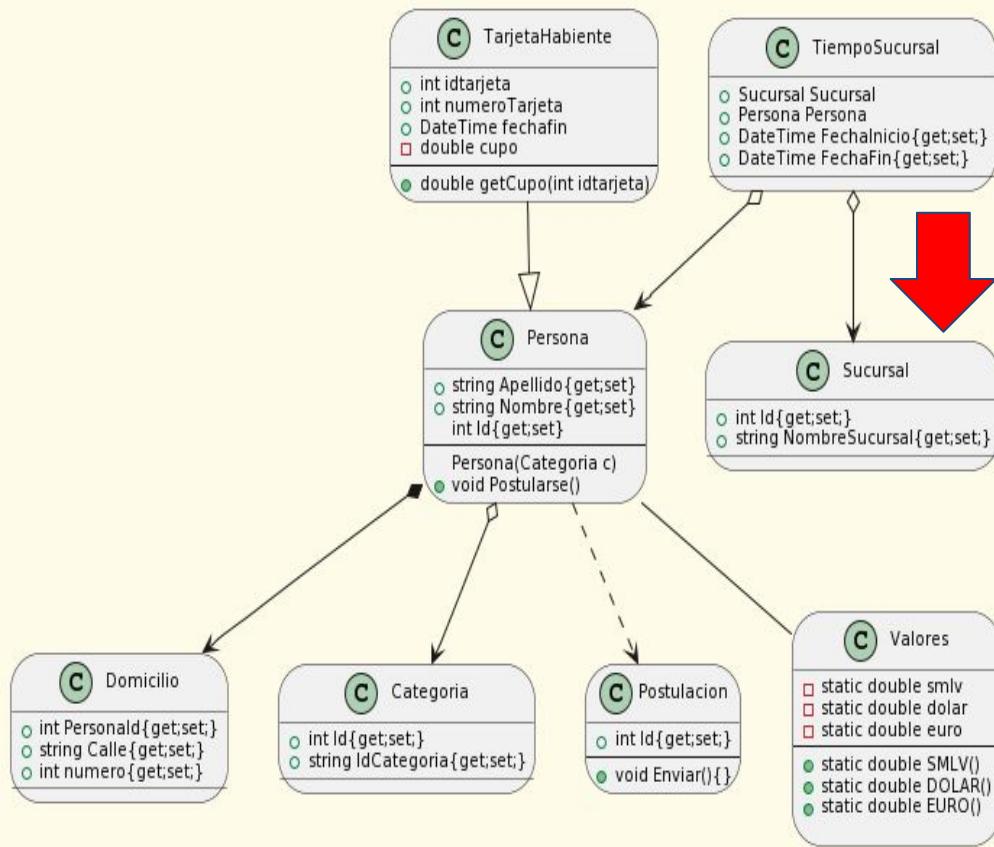
The code snippet shows a C# class definition named `Postulacion` within the `Domicilios` namespace. The class has a constructor, a destructor, and a method `Enviar()`. Three numbered steps (1, 2, 3) are highlighted in red, likely indicating specific points of interest or sequence in the application logic.

RELACIONES ENTRE CLASE



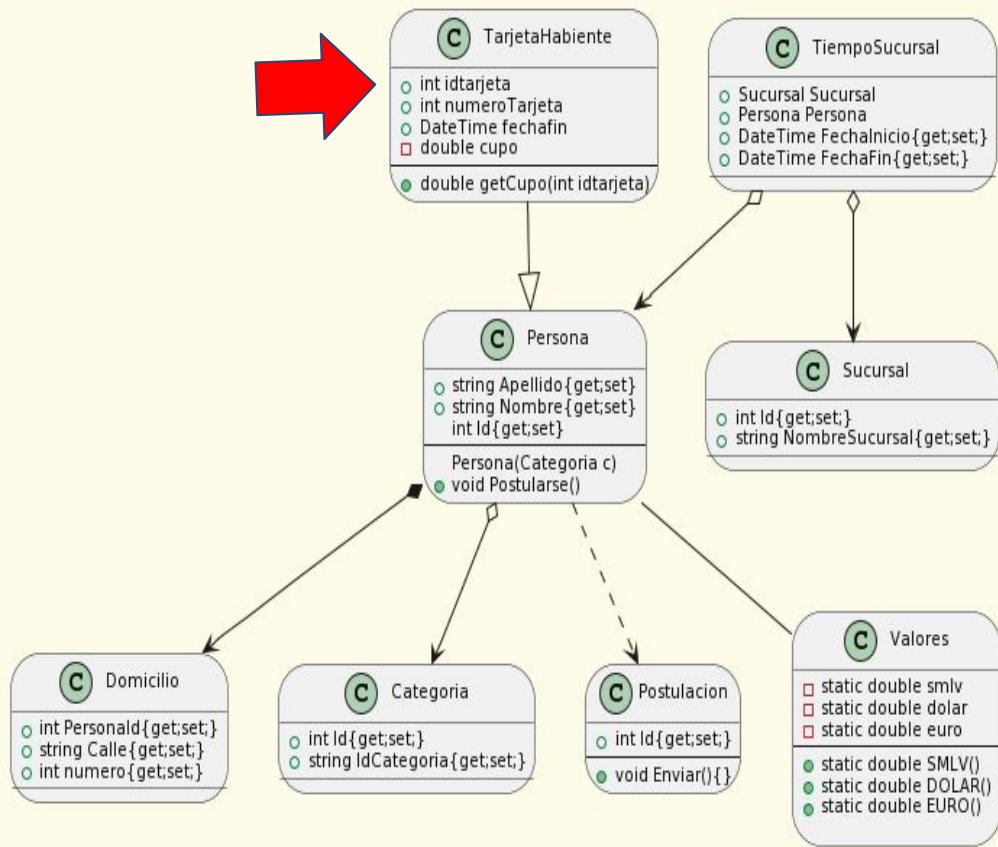
```
using System;
namespace Domicilios
{
    public static class Valores
    {
        private static double dolar=3478.0;
        private static double euro=4300.0;
        private static double smtv=978000.0;
        public static double SMLV(){return smtv;}
        public static double EURO(){return euro;}
        public static double DOLAR(){return dolar;}
    }
}
```

RELACIONES ENTRE CLASE



```
using System;
namespace Domicilios
{
    public class Sucursal
    {
        public int Id{get;set;}
        public string NombreSucursal{get;set;}
        public static int paso;
        public static int cantidad;
        public Sucursal(){Console.WriteLine("Creando Sucursal");}
    }
}
```

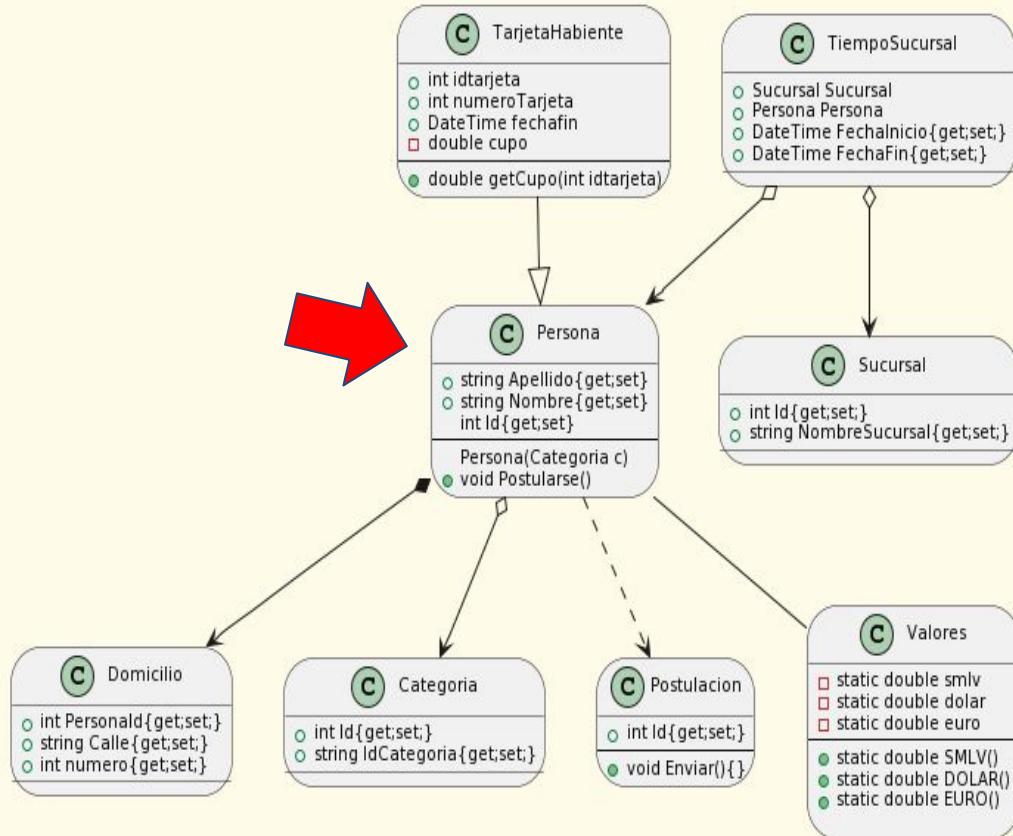
RELACIONES ENTRE CLASE



```
using System;
namespace Domicilios
{
    public class TarjetaHabiente:Persona
    {
        private int idtarjeta;
        private int numerotarjeta;
        private DateTime fechafin;
        private double cupo;
        public double getCupo(int idtarjeta){return 0;}
    }
}
```

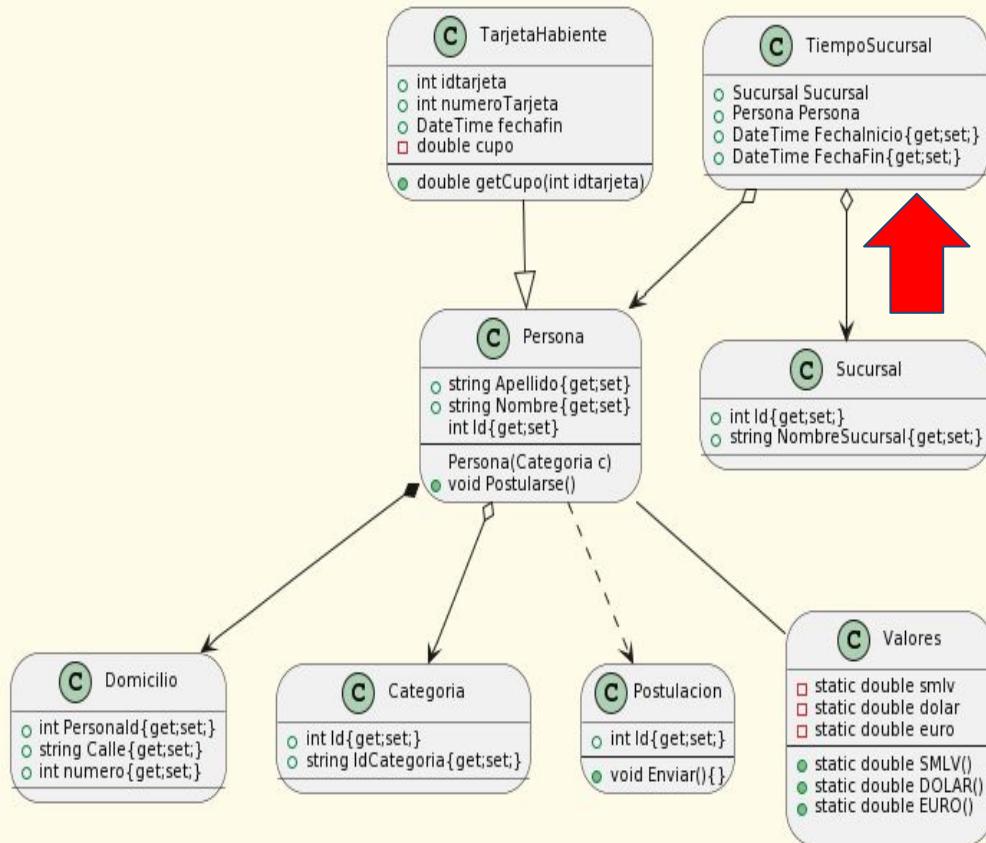
1

RELACIONES ENTRE CLASE



```
using System;
namespace Domicilios
{
    public class Persona
    {
        public string Apellido{get;set;}
        public string Nombre{get;set;}
        public int Id{get;set;}
        public Domicilio dom=new Domicilio(); ①
        public Categoría cat; ②
        public Persona(Categoría c)
        {
            cat=c; ③
        }
        public Persona():Console.WriteLine("Creando Persona");
        Console.WriteLine("SMLV={0}",Valores.SMLV());
        Console.WriteLine("DOLAR={0}",Valores.DOLAR());
        Console.WriteLine("EURO={0}",Valores.EURO());
        ~Persona():Console.WriteLine("Saliendo Persona");
        public void Postularse(){ ⑦
            var p= new Postulacion();
            p.Envíar();
            p=null;
            GC.Collect(); ⑨
        }
    }
}
```

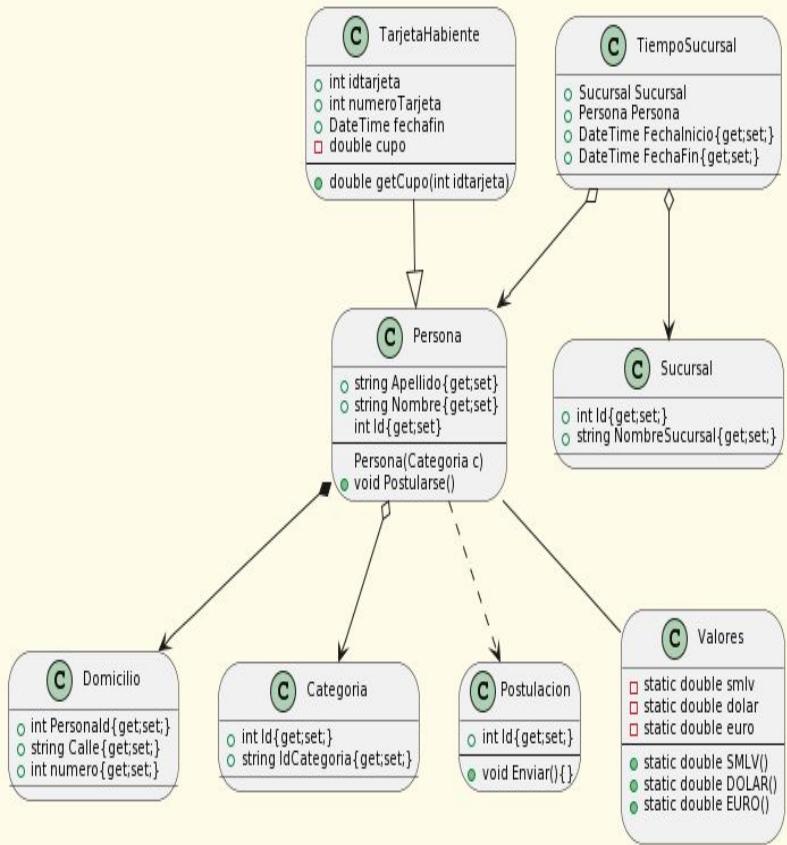
RELACIONES ENTRE CLASE



```
using System;
namespace Domicilios
{
    public class TiempoSucursal
    {
        public Sucursal sucursal; 1
        public Persona persona;

        public DateTime FechaInicio{get;set;}
        public DateTime FechaFin{get;set;}
        public TiempoSucursal()
        {
            sucursal=new Sucursal();
            persona=new Persona();
            Console.WriteLine("Creando TiempoSucursal");
        }
    }
}
```

RELACIONES ENTRE CLASE



```
using System;
namespace Domicilios
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("*****");
            var ts=new TiempoSucursal();
            ts.sucursal.Id=3;
            ts.sucursal.NombreSucursal="La Campiña";
            ts.persona.Postularse();
            ts.persona.dom.Calle="Calle 152B";
            ts.FechaInicio=DateTime.Now;
            ts=null;
            Console.WriteLine("*****");
            var p=new Persona(new Categoría());
            p.Postularse();
            p=null;
            GC.Collect();
            Sucursal.paso=3;
            Sucursal.cantidad=4;
            Console.WriteLine("Paso="+Sucursal.paso);
            Console.WriteLine("*****");
            Console.WriteLine("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

Detailed description: This C# code shows the creation and manipulation of objects from the classes defined in the UML diagram. It starts by creating a **TiempoSucursal** object (line 1). It then sets its **sucursal** attribute to a **Sucursal** object with Id 3 and NombreSucursal "La Campiña" (lines 2-3). It calls the **Postularse** method on the **Persona** object (line 4). It sets the **Calle** attribute of the **dom** (domicilio) of the **Persona** to "Calle 152B" and the **FechaInicio** to the current date and time (line 5). It then creates a new **Persona** object (line 6), sets its **Categoría** (line 7), and calls its **Postularse** method (line 8). Finally, it prints the value of the **paso** attribute of the **Sucursal** object (line 9) and waits for user input (line 10).

RELACIONES ENTRE CLASE

```

using System;
namespace Domicilios
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("*****");
            var ts=new TiempoSucursal(); 1
            ts.sucursal.Id=3;
            ts.sucursal.NombreSucursal="La Campiña"; 2
            3 ts.persona.Postularse();
            ts.persona.dom.Calle="Calle 152B";
            ts.FechaInicio=DateTime.Now;
            4 ts=null;
            Console.WriteLine("*****");
            var p=new Persona(new Categoria()); 5
            p.Postularse();
            p=null;
            GC.Collect(); 7
            Sucursal.paso=3; 8
            Sucursal.cantidad=4;
            Console.WriteLine("Paso="+Sucursal.paso); 9
            Console.WriteLine("*****");
            Console.WriteLine("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}

```

```

*****
Creando Sucursal
Creando Persona
SMLV=978000
DOLAR=3478
EURO=4300
Creando TiempoSucursal
Creando Postulacion
Enviando Postulacion
Saliendo Postulacion
*****
Creando Categoria...
Creando Postulacion
Enviando Postulacion
Paso=3
*****
Press any key to continue . . .
Saliendo Persona
Saliendo Postulacion
Saliendo de Categoria
Saliendo Persona

```

RELACIONES ENTRE CLASE



**PRÁCTICA EN EL AMBIENTE
DE APRENDIZAJE**

RELACIONES ENTRE CLASE

EJEMPLO

TABLA DE FÓRMULAS		
PERÍMETRO	ÁREAS	VOLUMENES
CUADRADO $P = L + L + L + L$	CUADRADO Lado $A = L \times L$	CUBO Lado $V = L^3$
RECTÁNGULO $P = b + b + h + h$	RECTÁNGULO Base Altura $A = b \times h$	PRISMA ab = área de la base Altura $V = ab \times h$
TRIÁNGULO $P = L + L + L$	TRIÁNGULO Báse Altura $A = \frac{b \times h}{2}$	PIRÁMIDE Báse Altura $V = \frac{ab \times h}{3}$
PENTÁГОNO $P = L \times 5$	PENTÁGOLO Lado apótema $A = \frac{p \times a}{2}$	CILINDRO Altura 2 Radio $V = \pi r^2 \times h$
CÍRCULO $C = \pi \times D$	CÍRCULO Radio $A = \pi \times r^2$	ESFERA $V = \frac{4}{3} \times \pi \times r^3$
$ab = \text{área de la base}$ $b = \text{base}$ $p = \text{perímetro}$ $a = \text{apotema}$ $L = \text{lado}$ $\pi = \text{Pi} = 3.1416$ $h = \text{altura}$ $D = \text{diámetro}$		

RELACIONES ENTRE CLASE EJEMPLO



TABLA DE FÓRMULAS

PERÍMETRO	ÁREAS	VOLUMENES
CUADRADO $P = L + L + L + L$	CUADRADO $A = L \times L$	CUBO $V = L^3$
RECTÁNGULO $P = b + b + h + h$	RECTÁNGULO $A = b \times h$	PRISMA $V = ab \times h$ ab = área de la base
TRIÁNGULO $P = L + L + L$	TRIÁNGULO $A = \frac{b \times h}{2}$ Base 3 CM Altura 2 CM	PIRÁMIDE $V = \frac{ab \times h}{3}$ ab = área de la base Base 7 CM Altura 3 CM
PENTÁГОNO $P = L \times 5$	PENTÁGOLO $A = \frac{p \times a}{2}$ Lado 8 CM apótema 2 CM	CILINDRO $V = \pi \times r^2 \times h$ Altura 30 CM Radio 2 CM
CÍRCULO $C = \pi \times D$	CÍRCULO $A = \pi \times r^2$	ESFERA $V = \frac{4}{3} \times \pi \times r^3$

using System;

namespace Figuras

{

class Program

{

protected Double radio_Circulo;

}

}

}

RELACIONES ENTRE CLASE

EJEMPLO



TABLA DE FÓRMULAS		
PERÍMETRO	ÁREAS	VOLUMENES
CUADRADO $P = L + L + L + L$	CUADRADO $A = L \times L$	CUBO $V = L^3$
RECTÁNGULO $P = b + b + h + h$	RECTÁNGULO $A = b \times h$	PRISMA $V = ab \times h$ ab = área de la base
TRIÁNGULO $P = L + L + L$	TRIÁNGULO $A = \frac{b \times h}{2}$ Base 3CM Altura 2CM	PIRÁMIDE $V = \frac{ab \times h}{3}$ ab = área de la base
PENTÁГОNO $P = L \times 5$	PENTÁGO $A = \frac{p \times a}{2}$ Lado 8CM apótema 2CM	CILINDRO $V = \pi \times r^2 \times h$ Altura 30CM Radio 2 CM
CÍRCULO $C = \pi \times D$	CÍRCULO $A = \pi \times r^2$	ESFERA $V = \frac{4}{3} \times \pi \times r^3$

ab = área de la base b = base p = perímetro a = apótema L = lado $\pi = \text{Pi} = 3.1416$ h = altura D = diámetro

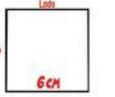
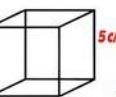
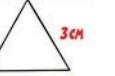
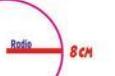
```
using System;

namespace Figuras
{
    /// <summary>
    /// Description of Figuras.
    /// </summary>
    public class Figuras
    {
        protected Double radio_Circulo{get;set;}
        public Figuras()
        {
        }
        }
    }
```

RELACIONES ENTRE CLASE

EJEMPLO

TABLA DE FÓRMULAS

PERÍMETRO	ÁREAS	VOLUMENES
CUADRADO  $P = L + L + L + L$	CUADRADO  $A = L \times L$	CUBO  $V = L^3$
RECTÁNGULO  $P = b + b + h + h$	RECTÁNGULO  $A = b \times h$	PRISMA  $V = ab \times h$ ab = área de la base h = altura
TRIÁNGULO  $P = L + L + L$	TRIÁNGULO  $A = \frac{b \times h}{2}$ Base 3 CM Altura 2 CM	PIRÁMIDE  $V = \frac{ab \times h}{3}$ ab = área de la base h = altura
PENTÁГОNO  $P = L \times 5$	PENTÁGOLO  $A = \frac{p \times a}{2}$ Lado 8 CM apótema 2 CM	CILINDRO  $V = \pi \times r^2 \times h$ r = radio h = altura
CÍRCULO  $C = \pi \times D$	CÍRCULO  $A = \pi \times r^2$	ESFERA  $V = \frac{4}{3} \times \pi \times r^3$
$ab = \text{área de la base}$ $b = \text{base}$ $p = \text{perímetro}$ $a = \text{apótema}$ $L = \text{lado}$ $\pi = \text{Pi} = 3.1416$ $h = \text{altura}$ $D = \text{diámetro}$		

```
using System;

namespace Figuras
{
    /// <summary>
    /// Description of Circulo.
    /// </summary>
    public class Circulo:Figuras
    {
        public Circulo(Double radio){
            radio_Circulo=radio;
        }
        public void getArea(){
            var miAC=new AreaCirculo(radio_Circulo);
            Console.WriteLine("El area del circulo de radio {0} es={1}",radio_Circulo,miAC.getAC());
        }
    }
}
```

RELACIONES ENTRE CLASE

EJEMPLO

TABLA DE FÓRMULAS		
PERÍMETRO	ÁREAS	VOLUMENES
CUADRADO $P = L + L + L + L$	CUADRADO Lado $A = L \times L$	CUBO Lado $V = L^3$
RECTÁNGULO 3cm 6cm $P = b + b + h + h$	RECTÁNGULO 3cm am educación 6cm Base $A = b \times h$	PRISMA $V = ab \times h$ ab = área de la base 3cm Altura
TRIÁNGULO 3cm $P = L + L + L$	TRIÁNGULO $A = \frac{b \times h}{2}$ Altura 2cm Base 3cm	PIRÁMIDE $V = \frac{ab \times h}{3}$ ab = área de la base 3cm Altura 3cm Base
PENTÁГОNO (AM) 8cm $P = L \times 5$	PENTÁGO $A = \frac{p \times a}{2}$ Lado 8cm apótema 2cm	CILINDRO $V = \pi \times r^2 \times h$ 30cm Altura 2cm Radio
CÍRCULO 8cm $C = \pi \times D$	CÍRCULO Radio 8cm $A = \pi \times r^2$	ESFERA $V = \frac{4}{3} \times \pi \times r^3$ 2cm

ab = área de la base b = base p = perímetro a = apótema L = lado $\pi = \text{Pi} = 3.1416$ h = altura D = diámetro

```
using System;

namespace Figuras
{
    /// <summary>
    /// Description of AreaCirculo.
    /// </summary>
    public class AreaCirculo
    {
        public Double radio;

        public AreaCirculo(Double radio){
            this.radio=radio;
        }

        public Double getAC(){
            return 3.1416*radio*radio;
        }
    }
}
```

RELACIONES ENTRE CLASE

EJEMPLO



TABLA DE FÓRMULAS

PERÍMETRO	ÁREAS	VOLUMENES
CUADRADO $P = L + L + L + L$	CUADRADO $A = L \times L$	CUBO $V = L^3$
RECTÁNGULO $P = b + b + h + h$	RECTÁNGULO $A = b \times h$	PRISMA $V = ab \times h$ ab = área de la base
TRIÁNGULO $P = L + L + L$	TRIÁNGULO $A = \frac{b \times h}{2}$ Base 3CM Altura 2CM	PIRÁMIDE $V = \frac{ab \times h}{3}$ ab = área de la base Base 7CM Altura 3CM
PENTÁГОNO $P = L \times 5$	PENTÁGOLO $A = \frac{p \times a}{2}$ Lado 8CM apótema 2CM	CILINDRO $V = \pi \times r^2 \times h$ Base 20CM Altura 30CM
CÍRCULO $C = \pi \times D$	CÍRCULO $A = \pi \times r^2$	ESFERA $V = \frac{4}{3} \times \pi \times r^3$
$a = \text{área de la base}$ $b = \text{base}$ $p = \text{perímetro}$ $a = \text{apótema}$ $L = \text{lado}$ $\pi = \text{Pi} = 3.1416$ $h = \text{altura}$ $D = \text{diámetro}$		

```
using System;
namespace Figuras
{
    class Program
    {
        public static void Main(string[] args)
        {

            // TODO: Implement Functionality Here
            var c=new Circulo(5.0);
            c.getArea();
            Console.WriteLine("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

El area del circulo de radio 5 es=78,54
Press any key to continue . . . -

DESCARGAR ARCHIVOS



COMPILADOR SHARPDEV

VERSIÓN PDF



CRÉDITOS



Realizado por el instructor José Fernando Galindo Suárez
jgalindos@sena.edu.co 2022





G R A C I A S

Línea de atención al ciudadano: 018000 910270
Línea de atención al empresario: 018000 910682



@SENAcomunica

www.sena.edu.co