

The slide features a white background with decorative geometric elements in the corners. These include dark gray and light gray triangles, some with double outlines, pointing towards the center. The main title is centered in a large, bold, black sans-serif font.

CLASE 09

POLIMORFISMO

Programación y Laboratorio II

CONTENIDO

HERENCIA

- Repaso Herencia

POLIMORFISMO

- ¿Qué es el polimorfismo?
- Herencia polimorfica
- Herencia no-polimorfica
- Sobreescritura de miembros heredados de Object

CLASES ABSTRACTAS

- ¿Qué es una clase abstracta?
- Miembros abstractos
- Implementación de miembros abstractos

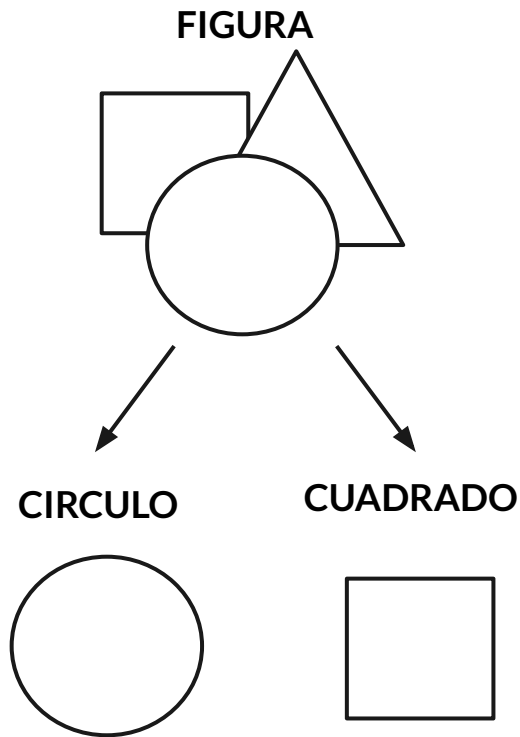


01.

HERENCIA

HERENCIA

- Una clase derivada es una especialización de la clase base. La clase derivada amplía la funcionalidad de la clase base.
- La clase derivada obtiene todos los miembros de la clase base, salvo sus constructores
- Una clase derivada sólo puede tener una clase base directa, pero la herencia es transitiva. Si C deriva de B y B se deriva de A, C hereda los miembros declarados en B y A






02.

POLIMORFISMO



POLIMORFISMO

es una palabra griega que significa
"con muchas formas"

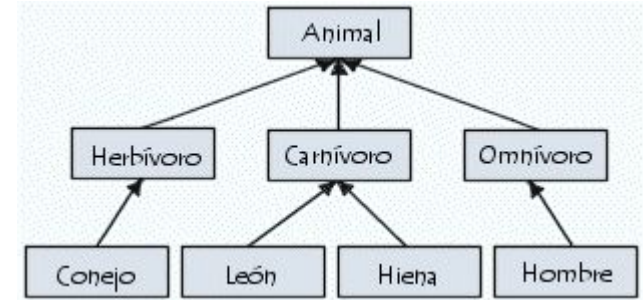


POLIMORFISMO

El polimorfismo es la habilidad de los objetos de responder al mismo mensaje de distintas formas.

Mensaje: cuando un objeto recibe una invocación de uno de los servicios que ofrece en forma de métodos

Las clases derivadas pueden proporcionar su propia definición e implementación de miembros heredados.



HERENCIA NO-POLIMORFICA

La herencia no-polimórfica nos permite redefinir un método de la clase base pero sin que se aplique polimorfismo.

El CLR ejecutará la implementación correspondiente al tipo de la referencia que apunta al objeto en memoria, sin importar el tipo del objeto en memoria.

La palabra reservada **NEW** oculta la implementación de la clase base

```
1  public class Animal
2  {
3      public string EmitirSonido()
4      {
5          return "¡Roar!";
6      }
7  }
8
9  public class Perro : Animal
10 {
11     public string EmitirSonido()
12     {
13         return "¡Woof!";
14     }
15 }
16
17 public class Gato : Animal
18 {
19     public string EmitirSonido()
20     {
21         return "¡Miau!";
22     }
23 }
```


HERENCIA POLIMORFICA

La herencia polimórfica nos permite redefinir un método de la clase base aplicando polimorfismo.

El runtime ejecutará la implementación correspondiente al tipo real del objeto en memoria, independientemente del tipo de la referencia.

La palabra reservada **VIRTUAL** se usa para declarar un método que pueda ser invalidado / redefinido / sobrescrito por una clase derivada.

La palabra reservada **OVERRIDE** se usa para invalidar / redefinir / sobrescribir un método virtual de la clase base.

```
1 public class Animal
2 {
3     public virtual string EmitirSonido()
4     {
5         return "¡Woof!";
6     }
7 }
8
9 public class Gato : Animal
10 {
11     public override string EmitirSonido()
12     {
13         return "¡Miau!";
14     }
15 }
16
17 static void Main(string[] args)
18 {
19     Animal animal = new Animal();
20     Animal perro = new Perro();
21     Animal gato = new Gato();
22
23     Console.WriteLine($"Animal: {animal.EmitirSonido()}");
24     Console.WriteLine($"Perro: {perro.EmitirSonido()}");
25     Console.WriteLine($"Gato: {gato.EmitirSonido()}");
26 }
```

SOBREESCRITURA DE EQUIVALENCIAS

De **object** heredamos los métodos **Equals** y **GetHashCode**.

Por defecto: Dos objetos son iguales si tienen la misma dirección de memoria.

Ambos métodos pueden redefinirse en las clases derivadas con una nueva implementación.

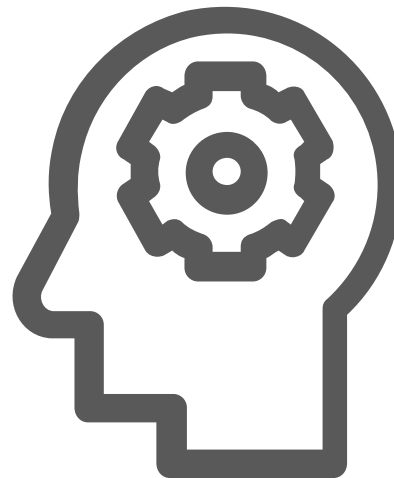
Un **hashcode** es un valor numérico que se utiliza para identificar y comparar objetos, por ejemplo en las colecciones **HashTable** y **HashSet**.

Dos objetos iguales deberían retornar el mismo hashcode.

```
1 public class Persona
2 {
3     private string nombre;
4     private string dni;
5
6     public static bool operator ==(Persona p1, Persona p2)
7     {
8         if (p1 is not null && p2 is not null)
9         {
10             return p1.nombre == p2.nombre && p1.dni == p2.dni;
11         }
12         return false;
13     }
14
15     public override bool Equals(object obj)
16     {
17         Persona persona = obj as Persona;
18         return persona is not null && this == persona;
19     }
20     public override int GetHashCode()
21     {
22         return (nombre, dni).GetHashCode();
23     }
24 }
```

EJERCICIOS

- I01 - Sobre-sobrescribiendo esas advertencias PARTE I





03.


CLASES ABSTRACTAS



CLASES ABSTRACTAS

Las clases abstractas no se pueden instanciar.

Su propósito es proporcionar una definición común
que modele una jerarquía de herencia.



CLASES ABSTRACTAS

Los **miembros abstractos** no tienen implementación.

Las clases derivadas de la clase abstracta **DEBEN** dar una implementación a todos los miembros abstractos.

Las clases abstractas son las únicas que pueden contener miembros abstractos, por lo que las declaraciones de métodos abstractos sólo se permiten en clases abstractas.

La implementación la proporciona un método **override** que es miembro de una clase no abstracta.

```
1 public abstract class Figura
2 {
3     public abstract float CalcularArea();
4 }
5 public class Cuadrado : Figura
6 {
7     int lado;
8
9     public override float CalcularArea(){
10         return lado * lado;
11     }
12 }
```

EJERCICIOS

- I01 - Sobre-sobrescribiendo esas advertencias PARTE II
- I02 - Calculadora de formas

