

The background features several large, stylized geometric shapes in the corners. These include dark gray and light gray triangles and chevrons, some with thin black outlines, creating a modern, abstract frame around the central text.

# **Clase 03**

# **Programación**

# **orienta a objetos**

# ¿Qué es un PARADIGMA DE PROGRAMACIÓN?


Un paradigma es una teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento.

Un paradigma de programación define la forma, metodología o estilo con el que se resolverá un problema utilizando un lenguaje de programación.



# ¿Qué es la PROGRAMACIÓN ORIENTADA A OBJETOS?

Es un paradigma de programación que propone resolver problemas a través de identificar objetos de la vida real, sus atributos (datos), su comportamiento (acciones) y las relaciones de colaboración entre ellos.



# Pilares de la programación orientada a objetos

ENCAPSULATION



ABSTRACTION



INHERITANCE



POLYMORPHISM





# ¿Qué significa ABSTRACCIÓN?

La habilidad de abordar un concepto mientras se ignoran algunos de sus detalles.

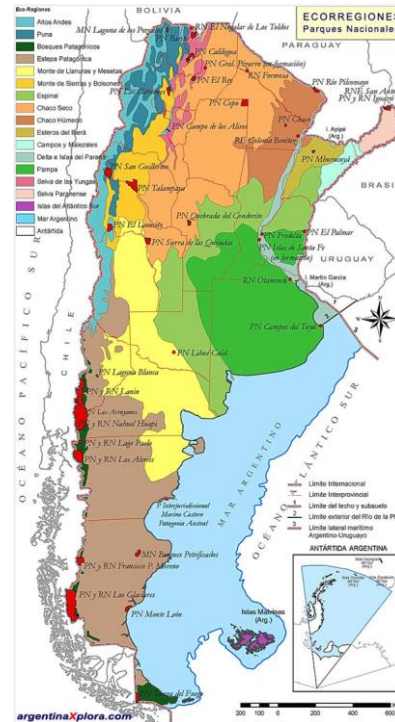
*Nos permite obtener una vista más simple de algo complejo, definiendo distintos niveles de detalle.*

*Hago foco en lo que me importa, descarto lo que no es relevante.*

# Niveles de abstracción



©Editorial Stella. Prohibida su venta bajo cualquier medio.



argentinaXplora.com



# ¿Cómo aplicamos la abstracción?

En análisis/diseño orientado a objetos, consiste en:

- ❖ Identificar las entidades que forman parte de nuestro contexto de negocio o problema a resolver.
- ❖ Definir las características esenciales de una entidad que la distinguen de otros tipos de entidades.
- ❖ Descartar las características que no sean relevantes, conservando aquellas que sean importantes en el contexto del problema.

# Classes



# ¿Qué es una CLASE?

Una clase es una **descripción** de un **conjunto de objetos** que comparten los mismos **atributos, métodos, relaciones y semántica** en un determinado **contexto**.

*Una clase es una implementación de una abstracción.*

# Composición de una Clase

## Atributos

Representan **características** que son compartidas por todos los objetos de una clase. Definen el rango de valores que puede tomar cada una de las propiedades de un objeto.

Utilizar notación *lowerCamelCase* y *sustantivos*.

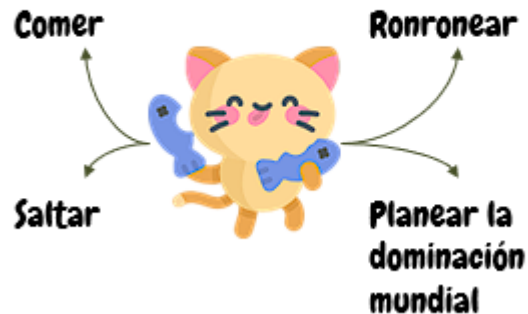


## Métodos

Un método es la **implementación de una operación**.

Una operación es una **abstracción de algo que puede hacer un objeto** y que es compartido por todos los objetos de esa clase.

Utilizar notación *UpperCamelCase* y *verbos*.



# Sintaxis de CLASE

```
//Modificador - Palabra Reservada Class - Identificador
3 referencias
public class Mascota
{
    //Atributos

    //Modificador - Tipo - Identificador
    public string nombre;
    public int edad;
    public string raza;
    public string especie;
    public bool hambre;
    //Metodos
    0 referencias
    public string Saludar()
    {
        return $"Hola mi nombre es {this.nombre} soy un {this.especie} y tengo {this.edad} años";
    }
    0 referencias
    public static void Alimentar(Mascota mascota)
    {
        if (!mascota.hambre)
        {
            mascota.hambre = true;
        }
    }
}
```

## Modificadores de CLASE

Nombre	Descripción
abstract	Indica que la clase no podrá instanciarse.
internal (*)	Accesible en todo el proyecto (Assembly).
public (*)	Accesible desde cualquier proyecto.
private (*)	Accesor por defecto.
sealed	Indica que la clase no podrá heredar.

(\*) Modificadores de visibilidad

# Sintaxis Atributos

```
//Modificador - Tipo - Identificador
public string nombre;
public int edad;
public string raza;
public string especie;
public bool hambre;
```

## Modificadores de Atributos

Nombre	Puede ser accedido por...
private (*)	Los miembros de la misma clase.
protected	Los miembros de la misma clase y clases derivadas o hijas.
internal	Los miembros del mismo proyecto.
internal protected	Los miembros del mismo proyecto o clases derivadas.
public	Cualquier miembro. Accesibilidad abierta.

# Sintaxis Métodos

```
//Metodos

//Modificador - Retorno - Identificador - Parametro de entrada
0 referencias
public string Saludar()
{
    return $"Hola mi nombre es {this.nombre} soy un {this.especie} y tengo {this.edad} años";
}

//Modificador - Static - Retorno - Identificador - Parametro de entrada
0 referencias
public static void Alimentar(Mascota mascota)
{
    if (!mascota.hambre)
    {
        mascota.hambre = true;
    }
}
```

## Modificadores de Métodos

Nombre	Descripción
abstract	Sólo la firma del método, sin implementar.
extern	Firma del método (para métodos externos).
internal (*)	Accesible desde el mismo proyecto.
override	Reemplaza la implementación del mismo método declarado como virtual en una clase padre.
public (*)	Accesible desde cualquier proyecto.
private (*)	Sólo accesible desde la clase.
protected (*)	Sólo accesible desde la clase o derivadas.
static	Indica que es un método de clase.
virtual	Permite definir métodos, con su implementación, que podrán ser sobrescritos en clases derivadas.

(\*) Modificadores de visibilidad



# **NameSpaces**



# NameSpace

- ❖ Es una agrupación lógica de clases y otros elementos.
- ❖ Toda clase esta dentro de un NameSpace.
- ❖ Proporcionan un marco de trabajo jerárquico sobre el cuál se construye y organiza todo el código.
- ❖ Su función principal es la organización del código para reducir los conflictos entre nombres.
- ❖ Esto hace posible utilizar en un mismo programa componentes de distinta procedencia.





# Directivas







# Directivas

- ❖ Son elementos que permiten a un programa identificar los NameSpaces que se usarán en el mismo.
- ❖ Permiten el uso de los miembros de un NameSpace sin tener que especificar un nombre completamente cualificado.

C# posee dos directivas de NameSpace:

- ✓ Using
- ✓ Alias

# Objetos

# ¿Qué es un OBJETO?

Los objetos son **instancias** de una clase.

Una instancia es **una manifestación concreta** de algo.

*Las clases son el molde o plano a partir de las cuales se crean los objetos.*



The background features several large, stylized geometric shapes in the corners. In the top-left and bottom-right, there are dark gray shapes with a white outline and a thin black border. In the top-right and bottom-left, there are light gray shapes. The word "Constructores" is centered in a bold, black, sans-serif font.

# **Constructores**

# ¿Qué es un **CONSTRUCTOR**?

Un **constructor** es un método especial cuya función es darle un valor inicial a los atributos de un objeto para asegurar el correcto funcionamiento del mismo.

# Tipos de constructores

## De Instancia/No estáticos

- Estos se utilizan para inicializar objetos al momento de su creación.
- En C#, la única forma de crear un objeto es mediante el uso de la palabra reservada `new` para adquirir y asignar memoria.
- Aunque no se escriba ningún constructor, existe uno por defecto que se usa cuando se crea un objeto a partir de un tipo referencia.
- Los constructores llevan el mismo nombre de la clase.

## Estáticos

- Son invocados por el entorno de ejecución (CLR) una única vez.
  - En la primera interacción con la clase.
  - Siempre es el primer constructor en invocarse.
- **No** pueden ser invocados de otra forma.
  - No tienen modificador de acceso.
  - No tienen parámetros de entrada.
- Al igual que los métodos estáticos, sólo permiten trabajar con otros miembros que también sean estáticos.

# Ejemplo constructor de instancia

```
0 referencias
internal class Program
{
    0 referencias
    static void Main(string[] args)
    {
        //Constructor por defecto
        Persona persona = new Persona();

        Console.ReadKey();
    }
}

2 referencias
public class Persona
{
    public int edad;
    public string nombre;
    public string apellido;
}
```

```
0 referencias
internal class Program
{
    0 referencias
    static void Main(string[] args)
    {
        //Si una declaracion explicita de constructor, se anula el por defecto
        Persona persona = new Persona();

        Console.ReadKey();
    }
}

3 referencias
public class Persona
{
    public int edad;
    public string nombre;
    public string apellido;
    1 referencia
    public Persona (string nombre, string apellido, int edad)
    {
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
    }
}
```

# Ejemplo constructor de instancia

```
0 referencias
internal class Program
{
    0 referencias
    static void Main(string[] args)
    {
        //Constructor explicito
        Persona persona = new Persona("Jose", "Perez", 50);

        Console.ReadKey();
    }
}

3 referencias
public class Persona
{
    public int edad;
    public string nombre;
    public string apellido;
    1 referencia
    public Persona (string nombre, string apellido, int edad)
    {
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
    }
}
```



# Declarar e instanciar varios objetos.

```
0 referencias
static void Main(string[] args)
{
    Mascota perro = new Mascota();
    perro.especie = "Perro";
    perro.raza = "Labrador";
    perro.nombre = "Perrito Malvado";
    perro.edad = 3;

    Mascota gato = new Mascota();
    gato.especie = "Gato";
    gato.raza = "Persa";
    gato.nombre = "Bola de nieve";
    gato.edad = 1;

    Console.WriteLine(perro.Saludar());
    Console.WriteLine(gato.Saludar());
    Console.ReadKey();
}
```

- ✓ `Mascota()` es el constructor del objeto y no el tipo de objeto.
- ✓ Una vez inicializado el objeto se puede utilizar para manipular sus atributos y llamar a sus métodos.

C:\E:\Facultad\5-Ayudantia\2022 - DIV E\Proyectos C#\Clase\EjerciciosClase\EjerciciosClase\bin\Debug\net5.0\EjerciciosClase.exe

```
Hola mi nombre es Perrito Malvado soy un Perro y tengo 3 años
Hola mi nombre es Bola de nieve soy un Gato y tengo 1 años
```

# Características de los objetos

## Los objetos viven en memoria

En C#, los objetos se crean a partir de clases, las cuales son **tipos de referencia**.

Por consecuencia, se almacenan en el **sector heap** de la memoria.

Si decimos que los objetos existen como bloques de memoria, entonces los objetos existen únicamente en **tiempo de ejecución**.

# Características de los objetos

## Los objetos tienen identidad

La **identidad** es la propiedad que permite diferenciar a un objeto y distinguirse de otros.

Los objetos de un mismo tipo (misma clase) tienen las **mismas propiedades** pero almacenan **valores independientes**.

*Por defecto, dos objetos son el mismo si tienen la misma referencia a memoria. Es decir, son la misma instancia.*

# Características de los objetos

## Los objetos se comunican

Si están relacionados, los objetos pueden enviar y recibir **mensajes** de otros objetos (comunicarse/interactuar).

El **comportamiento** de un objeto son las acciones que puede realizar al recibir un mensaje de otro objeto.

# Destrucción de un objeto

El tiempo de vida de una variable local está vinculado al ámbito en el que está declarada.

- Tiempo de vida corto (en general).
- Creación y destrucción deterministas.

El tiempo de vida de un objeto **no está vinculado a su ámbito**.

- Tiempo de vida más largo.
- Destrucción no determinista.

Los objetos se destruyen por un proceso conocido como **recolección de basura**.

En este proceso, un programa (**Garbage collector**) busca objetos inalcanzables y los destruye. Los convierte de nuevo en memoria binaria no utilizada.

# Ciclo de vida de un objeto



## CREACIÓN (Instanciar)

El **operador new** lo único que hace es reservar memoria binaria sin inicializar.

El **constructor** inicializará el estado del objeto en valores seguros (y sólo eso).



## UTILIZACIÓN

Una vez instanciado el objeto se pueden invocar sus métodos y atributos a partir de la referencia.



## DESTRUCCIÓN

El **Garbage Collector** es un programa que forma parte del **CLR**.

Será el **encargado de liberar memoria** sin intervención de los programadores.

*En general, liberará memoria de objetos sin referencia.*

# ¿Qué es el ESTADO de un objeto?

El estado de un objeto son los valores que toman sus atributos en un determinado momento.

*¡Como si le tomaramos una foto!*

