



Programación y Laboratorio II

Clase 02

Programación orientada a objetos

Programación orientada a objetos

- ¿Qué es un paradigma de programación?
- ¿Qué es la programación orientada a objetos?
- Pilares de la programación orientada a objetos.

Abstracción

- ¿Qué significa abstracción?
- Niveles de abstracción.
- ¿Cómo aplicamos abstracción?

Clases

- ¿Qué es una clase?
- Composición de una clase.
- ¿Qué es un constructor?
- Constructores no-estáticos y estáticos.
- Métodos de acceso.
- Diagramas de clases (UML).
- Modificadores de acceso.
- Contexto de negocio
- Identificadores de las clases.

Objetos

- ¿Qué es un objeto?
- Instanciar un objeto.
- Características de un objeto.
- Destrucción de un objeto.
- Ciclo de vida de un objeto.
- ¿Qué es el estado de un objeto?



01.

Programación orientada a objetos

Principio DRY

“Toda pieza de conocimiento debe tener una representación única, inequívoca y fidedigna dentro de un sistema.” - The pragmatic programmer

Pieza de conocimiento

Funcionalidad precisa dentro del contexto de negocio o un algoritmo concreto.

Única

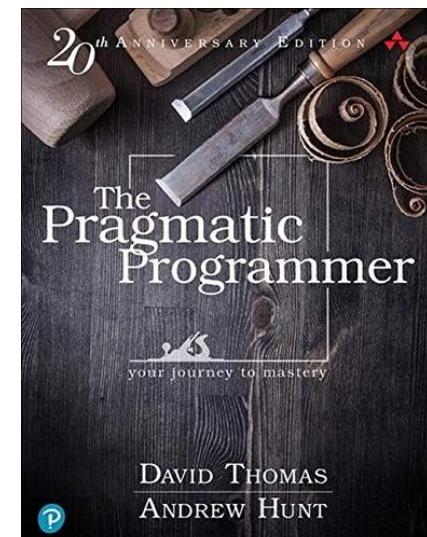
No debe existir otra representación de la misma pieza de conocimiento.

Inequívoca

Solamente puede ser interpretada, entendida o explicada de una manera.

Fidedigna

Debemos poder confiar que es correcta.



Principio DRY

"Many people took it [the DRY principle] to refer to code only: they thought that DRY means "don't copy-and-paste lines of source." [...] DRY is about the duplication of knowledge, of intent. It's about expressing the same thing in two different places, possibly in two totally different ways."

The Pragmatic Programmer - Dave Thomas



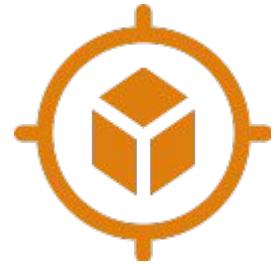
Cuando ocurra un cambio no deberíamos necesitar actualizar múltiples cosas en paralelo.



¿Qué es un PARADIGMA DE PROGRAMACIÓN?

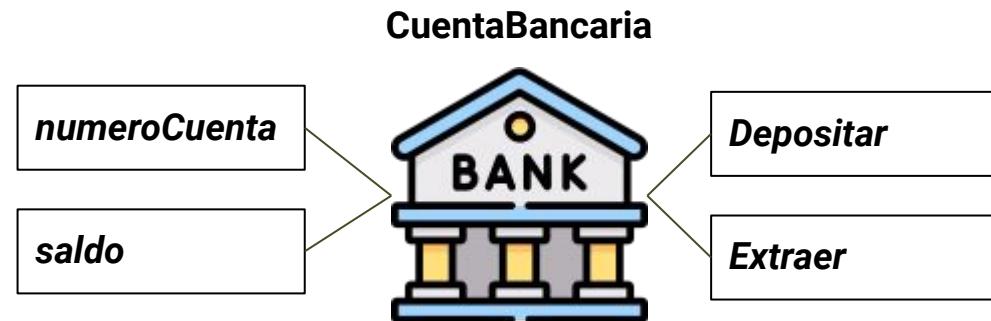
Un **paradigma** es una teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento.

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

Encapsulamiento



Abstraccion



Herencia



Polimorfismo





02. Abstracción



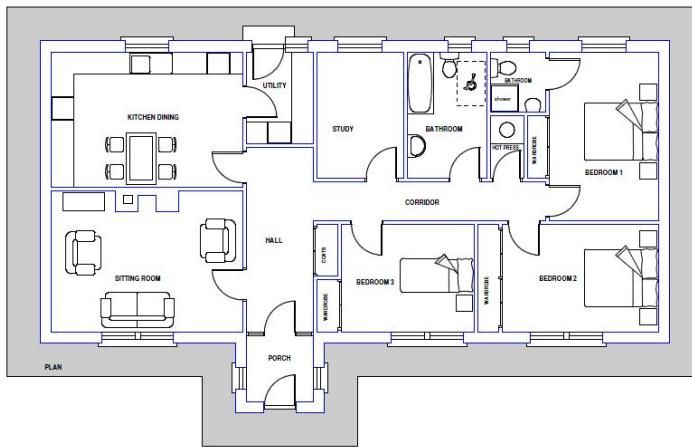
¿Qué significa ABSTRACCIÓN?

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

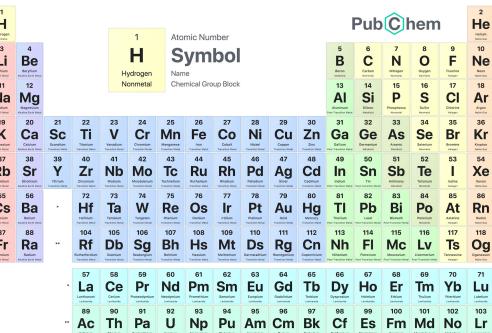
*Nos permite obtener una **vista más simple de algo complejo**, definiendo **distintos niveles de detalle**.*

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

Niveles de abstracción



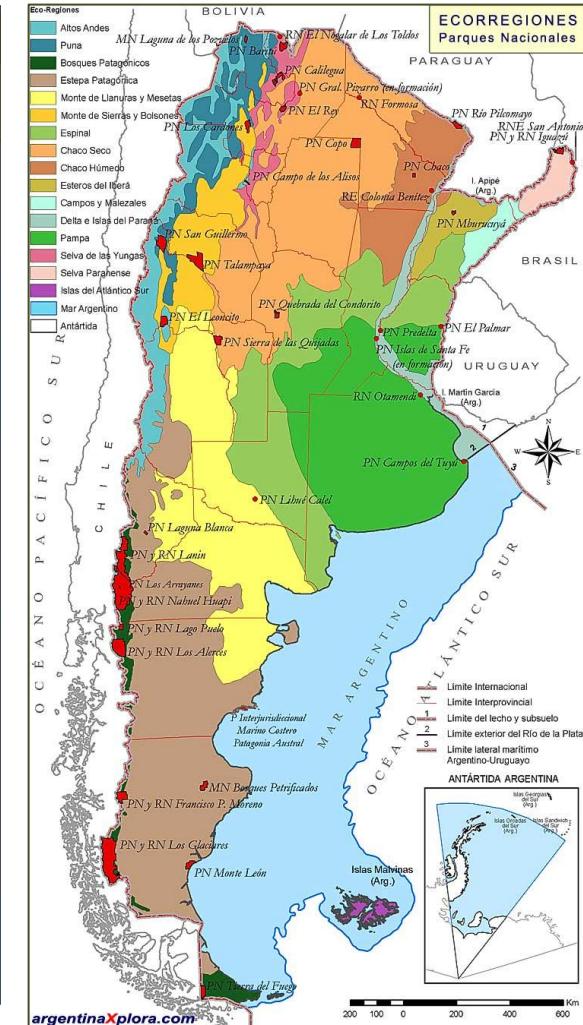
PERIODIC TABLE OF ELEMENTS



The table is a standard periodic table where each element is represented by a small square containing its atomic number (top left), symbol (top right), name (bottom left), and a color-coded block indicating its chemical group (bottom right). The groups are color-coded as follows: Ia (pink), IIa (light blue), IIIa (light green), IVa (yellow), Va (orange), VIa (red), VIIa (purple), VIIIA (dark purple), IA (light blue), IIA (light green), IIIA (yellow), IVA (orange), VA (red), VIA (purple), VIIIA (dark purple), and the transition metals (various shades of grey).



Niveles de abstracción

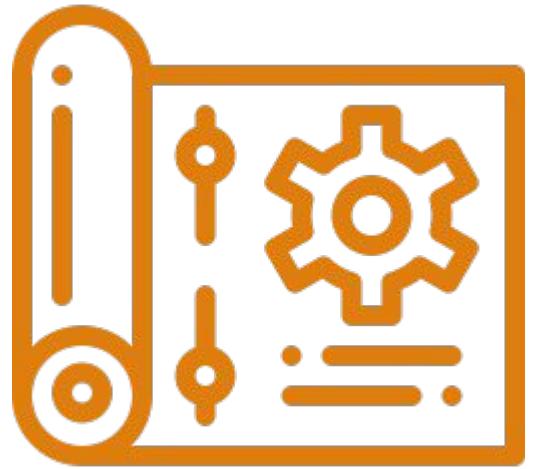


© Editorial Stella. Prohibida su venta bajo cualquier medio.

¿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.



03. Clases



¿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.

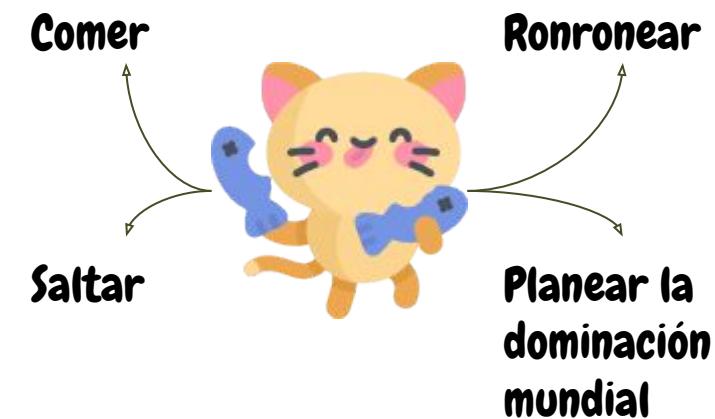
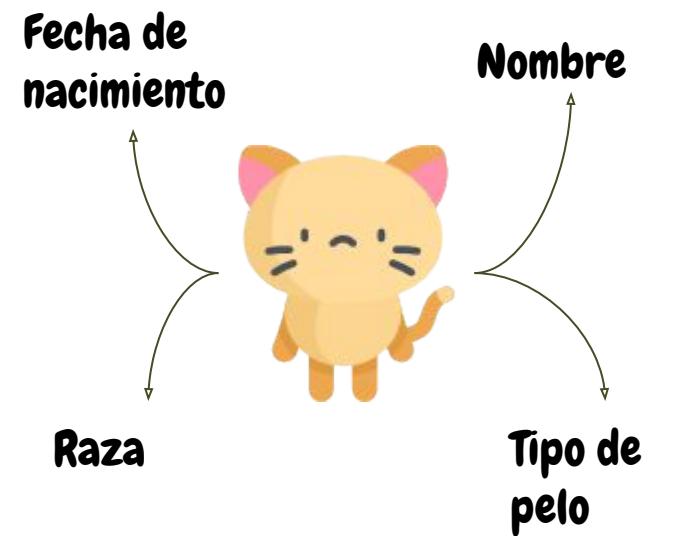
Utilizar notación **lowerCamelCase** y **sustantivos**.

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**.



Composición de una clase

Relaciones

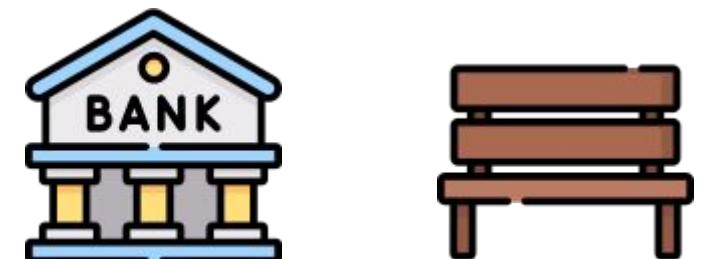
Las clases se conectan entre sí a través de relaciones y **colaboran** para realizar un comportamiento mayor.

Los objetos de clases relacionadas pueden interactuar entre sí.



Semántica

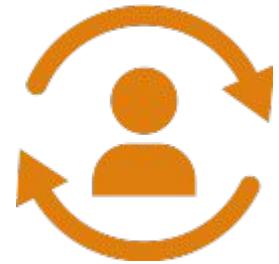
La semántica es el **significado** que se le da a la clase y a sus relaciones dentro del contexto de negocio.





¿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.



¿Qué son los MÉTODOS DE ACCESO?

Los **métodos de acceso** permiten consultar o modificar el valor los atributos de un objeto de forma segura (sin romper el encapsulamiento).

Si el método es para consultar se dice que es un **método getter**.

Si el método es para modificar o asignar un nuevo valor, se dice que es un **método setter**.

Métodos de acceso: getter y setter

```
1  public class Persona
2  {
3      private string nombre;
4      private string apellido;
5
6      public string GetNombre()
7      {
8          return nombre;
9      }
10
11     public void SetNombre(string nombre)
12     {
13         if(!string.IsNullOrWhiteSpace(nombre))
14         {
15             this.nombre = nombre.Trim();
16         }
17     }
18 }
```

¿Qué son las PROPIEDADES?

Una **propiedad** es un miembro que proporciona un mecanismo flexible para leer, escribir o calcular el valor de un atributo.

No son más que otra forma de escribir métodos de acceso (getters y setters).

Nos ayudan a aplicar el encapsulamiento:

- *Habilitan una forma segura de obtener y modificar el estado de un objeto.*
- *Aportan al ocultamiento de los detalles de la implementación.*

Descriptores de acceso



```
1 private int totalGoles;
2
3 public int GetTotalGoles()
4 {
5     return totalGoles;
6 }
7
8 public void SetTotalGoles(int value)
9 {
10    totalGoles = value;
11 }
```



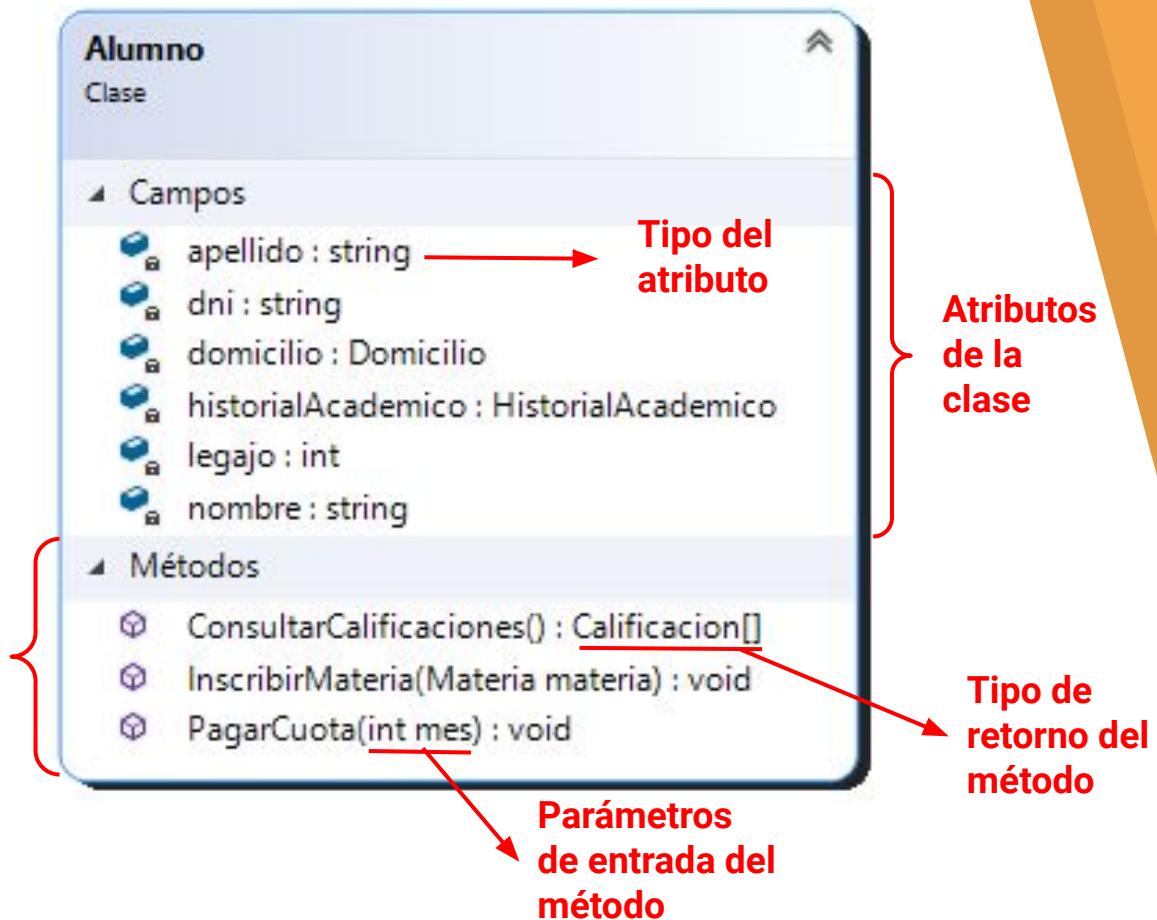
```
1 private int totalGoles;
2
3 public int TotalGoles
4 {
5     get
6     {
7         return totalGoles;
8     }
9     set
10    {
11        this.totalGoles = value;
12    }
13 }
```

Diagramas de clases

UML es un lenguaje de modelado que es un estándar a la hora de construir planos de software.

Entre sus herramientas se encuentran los **diagramas de clase**, que permiten modelar clases y sus relaciones.

Métodos
de la
clase



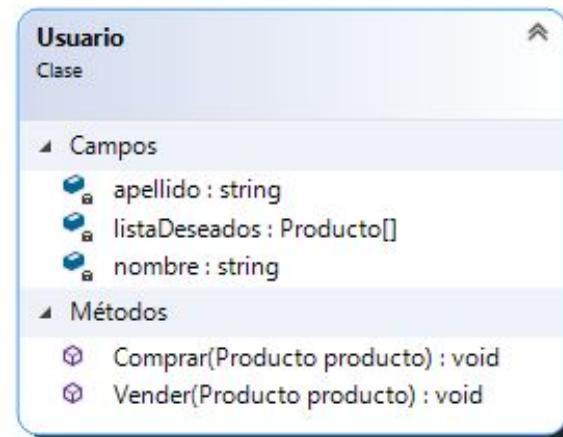
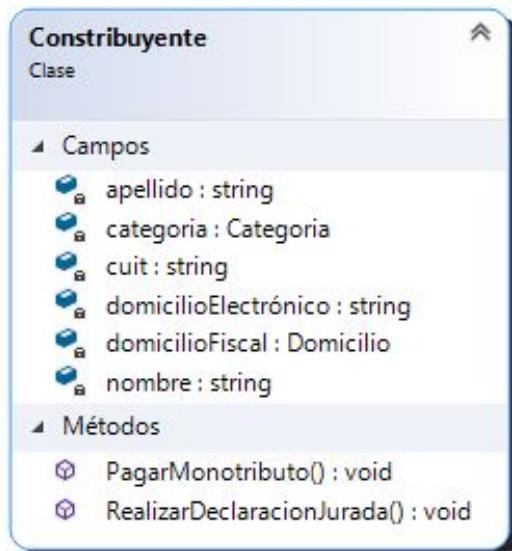
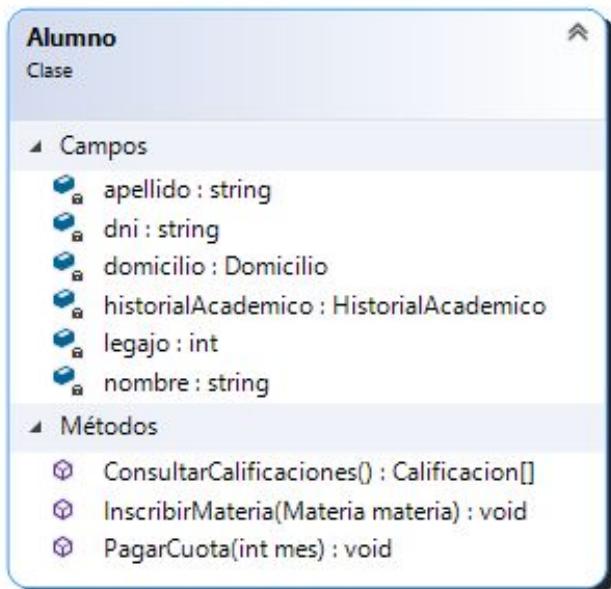
Modificadores de acceso

Todos los tipos de dato y sus miembros tienen un nivel de accesibilidad.

El **nivel de accesibilidad** controla bajo qué condiciones puede ser usado un elemento desde otras partes del código o desde otros proyectos.

Modificadores de acceso		
Ícono	Modificador	Descripción
Sin ícono	<i>public</i>	Acceso público, desde cualquier componente.
	<i>private</i>	Accesible sólo desde la misma clase.
	<i>protected</i>	Accesible sólo desde la misma clase o sus derivadas.
	<i>internal</i>	Accesible sólo desde el mismo proyecto.

Contexto de negocio / Dominio



Identificadores de clases

A la hora de nombrar clases debemos seguir las siguientes convenciones:

Grafía pascal

La primera letra del identificador y la primera letra de las siguientes palabras concatenadas están en mayúsculas (UpperCamelCase).

Uso de sustantivos

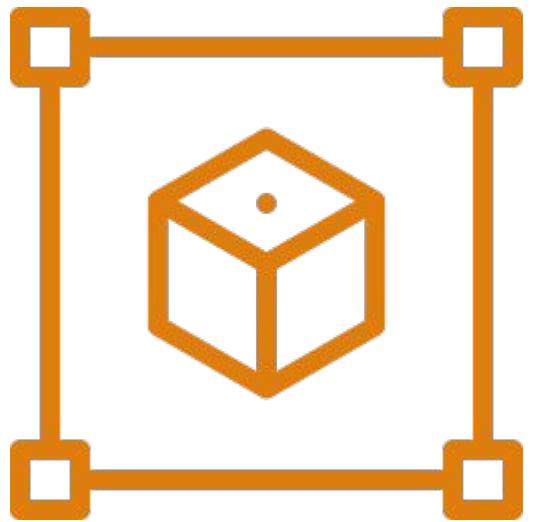
Los nombres de clases se escriben con sustantivos, ya que representan a objetos.

Nombres descriptivos

Como todo identificador, debe describir con la mayor exactitud posible a lo que representa.

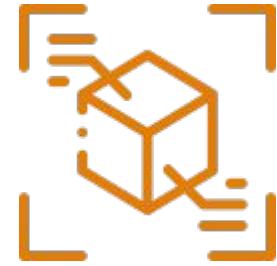
Ejemplos

HistoriaClinica, Mascota, JugadorDeFutbol, LibroDiario, Transferencia, DocenteAdjunto, ExamenFinal.



04.

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.



Instancias de una clase

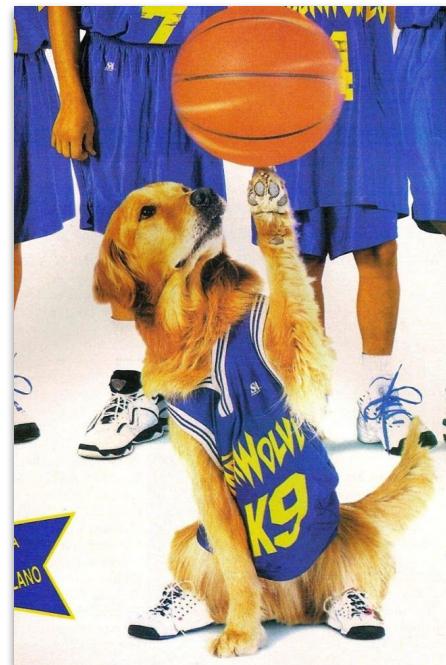
Mascota

Clase

▲ Campos

- especie : string
- fechaNacimiento : DateTime
- nombre : string

{



Instanciar un objeto

Para instanciar un objeto se utiliza el **operador new**.

`Gato felix;` → Declaración de una variable.

`felix = new Gato();`

→ Instancio el objeto y asigno a la variable.

Una vez instanciado, puedo invocar sus atributos y métodos a través de la referencia.

`felix.Comer();` → Invocación a método.

`Console.WriteLine(felix.fechaNacimiento);`

→ Lectura de atributo.

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 tomáramos una foto!



Ejercicios



- I01 - Creo que necesito un préstamo
- I03 - El ejemplo universal
- I04 - Invento argentino

https://codeutnfra.github.io/programacion_2_laboratorio_2_apuntes/

Tarea



- I02 - ¿Vos cuántas primaveras tenés?
- I05 - Prueba de geometría
- A02 - La veterinaria

https://codeutnfra.github.io/programacion_2_laboratorio_2_apuntes/