

# Clases

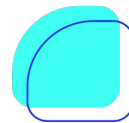
# Limitaciones de los objetos

A medida que desarrollamos sistemas con cientos de objetos, nos encontramos con que **tratar cada objeto individualmente tiene limitaciones.**

Cada objeto tiene características y comportamientos únicos que lo hacen singular. Sin embargo, surge la **necesidad de agrupar estos objetos en base a las similitudes en sus características y acciones compartidas.**

Esta agrupación permite gestionar eficientemente **conjuntos de objetos relacionados y aplicar operaciones comunes** de manera coherente y estructurada.

El proceso de agrupar objetos de acuerdo a sus similitudes se conoce también como **clasificación.**



## Ejemplo

Retomemos el ejemplo del manual anterior:

Se observa que el objeto **Superman** y el objeto **OmniMan** tienen el comportamiento de **atacar** que resulta **muy similar**. Podríamos reconocer incluso que, en ambos casos, hay **código duplicado**.

Entonces, se puede clasificar tanto a Superman como a OmniMan como **SuperHéroes** y decir que **todos los objetos de esa clase tienen la capacidad de atacar**.

```
// Objeto para representar a Superman
const superman = {
  nombre: "Superman",
  poder: 100,
  defensa: 80,
  salud: 100,
  atacar: function(oponente) {
    // Daño aleatorio entre 10 y 20
    const damage = Math.floor(Math.random() * (20 - 10 + 1)) + 10;
    oponente.salud -= damage;
  }
};
```

```
// Objeto para representar a Omni-Man
const omniMan = {
  nombre: "Omni-Man",
  poder: 90,
  defensa: 85,
  salud: 120,
  atacar: function(oponente) {
    // Daño aleatorio entre 15 y 25
    const damage = Math.floor(Math.random() * (25 - 15 + 1)) + 15;
    oponente.salud -= damage;
  }
};
```

# Clasificación de objetos

Es un proceso natural e intuitivo que se realiza cuando se reconocen las **características y comportamiento que comparten distintos objetos** en el sistema.

Veamos un ejemplo en la próxima pantalla.



### Ejemplo:

En la programación orientada a objetos cuando hablamos de **Superman y Iron Man** reconocemos **objetos individuales** con características únicas.

Pero al pensar el concepto de **superhéroe** con las **características y comportamientos compartidos que los definen**, se está hablando de **Clases**.



- **Superman** es un objeto.
- **Iron Man** es un objeto.
- Ambos pertenecen a la clase **Superhéroe**.

# Clase

En el contexto de la programación orientada a objetos, una clase es un **plano o molde que define las características y comportamientos de un conjunto de objetos similares**.

Proporciona la **estructura y las funcionalidades comunes** que compartirán todas sus instancias.

```
class SuperHeroe{  
    constructor(nombre, poder, defensa, salud) {  
        this.nombre = nombre;  
        this.poder = poder;  
        this.defensa = defensa;  
        this.salud = salud;  
    }  
  
    atacar(oponente) {  
        const damage = /* Calculo del ataque */  
        oponente.salud -= damage;  
    }  
}
```



## Ventajas de utilización de Clases en la POO

- **Reutilización de código:** Las clases permiten encapsular funcionalidades comunes en un solo lugar, esto facilita la reutilización del código en diferentes partes del programa.
- **Abstracción:** Las clases permiten abstraer conceptos del mundo real (como un empleado, un vehículo o una transacción) de manera que se refleje su estructura y comportamiento esencial.
- **Modularidad:** Las clases promueven la organización del código en módulos cohesivos y fácilmente mantenibles. Facilitan la colaboración entre equipos y el desarrollo incremental de *software*.



Más detalles sobre este concepto en el siguiente *slide*.

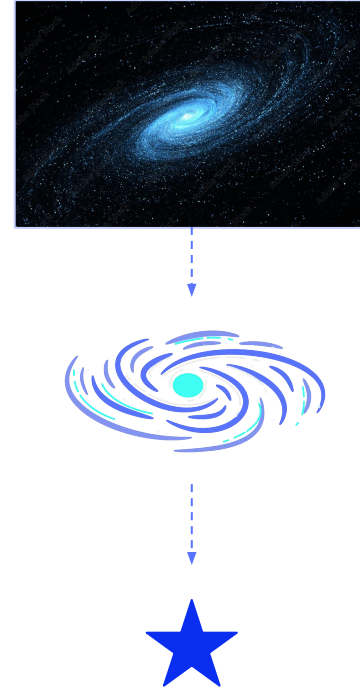


## Abstracción

El concepto de *clase* se funda en una capacidad inherente a todos, el **proceso de abstracción**: consiste en **simplificar la complejidad ignorando los detalles irrelevantes y enfocándose en lo esencial**.

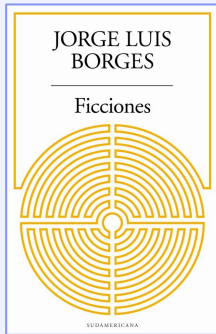
En POO, se abstraen características comunes en una clase para **modelar un concepto**.

En programación orientada a objetos, la abstracción es fundamental para **crear modelos que representen conceptos del mundo real de manera simplificada y coherente**. Esta representación permite desarrollar sistemas más estructurados y fáciles de entender y mantener.





### Ejemplo:



Un ejemplo que define la importancia del proceso de abstracción lo describe el cuento *“Funes el memorioso”*, de Jorge Luis Borges.

Allí, Borges describe a Funes, un personaje que no puede olvidar ningún detalle y esto le impide tener la capacidad de abstraer conceptos.

La programación **requiere la habilidad de abstraer para manejar la complejidad**. En definitiva, el proceso de abstracción conlleva **olvidar la diferencia entre distintos objetos y centrarse en lo esencial**.



# Características y comportamiento en las Clases

Las clases **encapsulan las características y comportamientos** de objetos similares o del mismo tipo.

## Características

- Definen los **atributos** o propiedades de los objetos.
- Representan los **datos específicos** que cada objeto puede contener.

## Comportamiento

- Define los **métodos que especifican las acciones** que los objetos pueden realizar.
- Incluye **funciones y procedimientos** que manipulan los datos asociados a los objetos.

**Todos los objetos que pertenecen a una clase comparten sus características y su comportamiento.**

# Tipos de datos primitivos

Son los **tipos de datos básicos** integrados en el lenguaje de programación.

Tipo de dato	Representa
Entero (int)	Números enteros.
Punto flotante (float)	Números con decimales.
Cadena (string)	Texto.
Booleano (bool)	Valores de verdad (True o False).

Las **variables de los lenguajes** dinámicamente tipados como Python o JavaScript, o de tipado estático como C# o Java, en un momento dado, alojan un **tipo de dato que identifica el conjunto de valores posibles y las operaciones que se pueden realizar sobre esos valores.**



# Tipos de datos Clase

En términos de un lenguaje de programación, **una clase corresponde a un tipo de dato personalizado** definido por el usuario. Esto permite **crear variables u objetos de ese tipo**,

con atributos que representan sus datos y métodos que definen sus comportamientos, extendiendo así las capacidades del lenguaje más allá de los tipos de datos primitivos.

```
const superman = new Personaje("Superman", 100, 80, 100);  
const batman = new Personaje("Batman", 80, 90, 90);  
  
superman.atacar(batman);
```

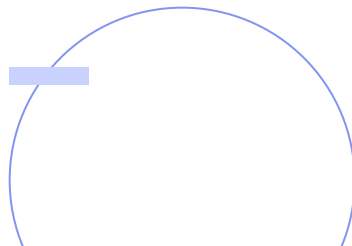
## Además de extender el lenguaje con tipos de datos personalizados, las clases:

- **Permiten declarar variables de ese tipo:**

Una vez definida una clase, se pueden declarar variables u objetos de ese tipo en el código, lo cual permite manejar conjuntos de datos complejos de manera estructurada.

- **Se instancian generalmente con new:** Para utilizar una clase y crear un objeto basado en ella, típicamente se utiliza la palabra clave new, que reserva memoria dinámica para almacenar el objeto y llama al constructor de la clase para inicializarlo.

- **Utilizan memoria dinámica:** Los objetos creados a partir de clases utilizan memoria dinámica, lo que permite una gestión eficiente de los recursos y una asignación flexible de memoria según sea necesario durante la ejecución del programa.



# Clases de negocios o dominio

En el desarrollo de *software* orientado a objetos, las expresiones “clases de negocio” o “*domain model*” se refieren al **conjunto de clases que representan los conceptos y reglas de negocio** que son fundamentales para una aplicación.

Esta capa se centra en **modelar los objetos y comportamientos que son específicos del dominio o sector al que pertenece la aplicación.**

Estas clases son la protagonistas en el diseño de *software* orientado a objetos. Representan **la esencia del negocio** y facilitan la construcción de aplicaciones robustas y adaptativas.

Es fundamental **comprender y modelar adecuadamente esta capa** para asegurar que la aplicación cumpla con los requisitos y expectativas del negocio de manera efectiva.



## Definiciones para recordar

- Una clase se comporta como un **tipo de datos**.
- Una clase es una **fábrica de objetos**.
- Un **objeto es una instancia** de una clase.
- Las clases **encapsulan datos y comportamientos** relacionados.
- Los objetos pueden interactuar entre sí a través de **métodos definidos en sus clases**.
- Las clases **proporcionan una estructura** para organizar y reutilizar código de manera eficiente.



# Ejemplos de clases

En las pantallas siguientes, con la **ayuda de ChatGPT**, se desarrollarán clases, en los siguientes lenguajes:

- Java.
- C#.
- JavaScript.
- Python.






## Clase en Java

¿Podrías programar, en JAVA, la clase Personaje que tiene nombre, defensa, poder y salud? Mostrar, directamente, el código de la clase.



Por supuesto, aquí tienes una clase `Personaje` en Java con los atributos `nombre`, `defensa`, `poder` y `salud`.

java

 Copiar código

```
public class Personaje {  
    private String nombre;  
    private int defensa;  
    private int poder;  
    private int salud;  
  
    // Constructor  
    public Personaje(String nombre, int defensa, int poder, int salud) {  
        this.nombre = nombre;  
        this.defensa = defensa;  
        this.poder = poder;  
        this.salud = salud;  
    }  
}
```




## Clase en C#

¿Podrías programar, en C#, la clase `Personaje` que tiene nombre, defensa, poder y salud?Mostrar directamente el código de la clase.



Claro, aquí tienes un ejemplo de una clase `Personaje` en C# que incluye las propiedades `Nombre`, `Defensa`, `Poder` y `Salud`:

csharp

 Copiar código

```
public class Personaje
{
    // Propiedades
    public string Nombre { get; set; }
    public int Defensa { get; set; }
    public int Poder { get; set; }
    public int Salud { get; set; }

    // Constructor por defecto
    public Personaje() {}

    // Constructor con parámetros
    public Personaje(string nombre, int defensa, int poder, int salud)
```



## Clase en JavaScript

¿Podrías programar, en JavaScript, la clase Personaje que tiene nombre, defensa, poder y salud? Mostrar directamente el código de la clase.



Claro, aquí tienes el código de la clase `Personaje` en JavaScript:

javascript

Copiar código

```
class Personaje {  
  constructor(nombre, defensa, poder, salud) {  
    this.nombre = nombre;  
    this.defensa = defensa;  
    this.poder = poder;  
    this.salud = salud;  
  }  
  
  recibirAtaque(poderAtaque) {  
    const danio = poderAtaque - this.defensa;  
    if (danio > 0) {  
      this.salud -= danio;  
      if (this.salud < 0) {  
        this.salud = 0;  
      }  
    }  
  }  
}
```

JS

## Clase en Python

¿Podrías programar, en Python, la clase `Personaje` que tiene nombre, defensa, poder y salud? Mostrar, directamente, el código de la clase.



Claro, aquí tienes el código de la clase `Personaje` en Python:

```
python Copiar código

class Personaje:
    def __init__(self, nombre, defensa, poder, salud):
        self.nombre = nombre
        self.defensa = defensa
        self.poder = poder
        self.salud = salud

    def __str__(self):
        return f"Nombre: {self.nombre}, Defensa: {self.defensa}, Poder: {self.poder}, Salud: {self.salud}"
```

Esta clase tiene un constructor (`__init__`) que inicializa los atributos `nombre`, `defensa`, `poder` y `salud`. Además, he añadido un método especial `__str__` para que puedas obtener una representación en cadena del objeto `Personaje` de manera sencilla.



# Conceptos avanzados

Esto es sólo el comienzo. Las clases permiten, además, varias características que son los **tres pilares de la programación orientada a objetos**.

## Encapsulamiento

Las clases permiten **ocultar detalles internos** de implementación y **exponer solo la interfaz pública**, lo que mejora la seguridad, reduce la complejidad y facilita el mantenimiento del código.

## Herencia

La herencia permite que una clase (subclase) **hereda atributos y métodos de otra clase** (superclase). Esto facilita la creación de **jerarquías de clases** y la definición de relaciones entre objetos.

## Polimorfismo

**Permite a objetos de diferentes clases responder a métodos de la misma manera**, lo que proporciona flexibilidad y permite el tratamiento genérico de objetos que comparten una interfaz común.