Programación Avanzada

UNIDAD 5: PROGRAMACIÓN ORIENTADA A OBJETOS

Unidad 5

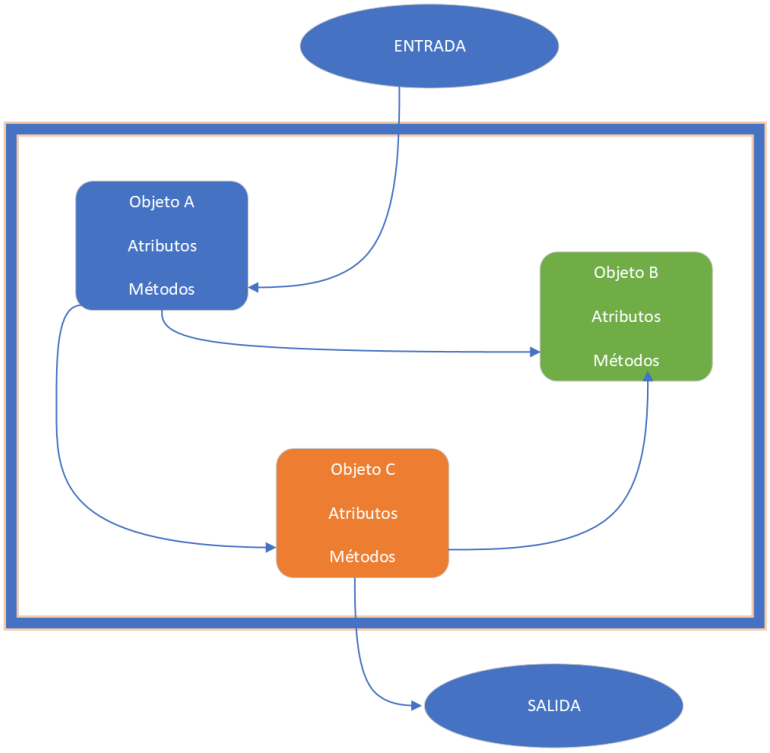
Programación Orientada a Objetos. Clases y Objetos. Instanciación de objetos. El objeto this. El constructor. Extensión y herencia. Visibilidad y encapsulación

¿QUÉ ES LA POO?

La Programación Orientada a Objetos (POO) es un **paradigma de programación**.

Se basa en el concepto de **clases** y **objetos**.

Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos.



Reseña Histórica de la POO

Años 60: Simula. (Noreweigan Computing System):

- Inspirado por los problemas involucrados en la simulación de sistemas de la vida real.

Años 70: Smalltalk (Xerox PARC):

- Alan Kay y su equipo de investigación desarrollan Smalltalk basándose en los conceptos que introducía Simula.

Años 80: Smalltalk 80 y C++ (Laboratorios Bells)

- Casi contemporáneo al trabajo de Kay desde los laboratorios Bell desarrollan C++ como extensión del lenguaje C.
- Comienzan proyectos similares, como Objective-C, Eiffel, Actor, entre otros.

Reseña Histórica de la POO

Años 90: Java (Sun Microsystems)	2002: C# y .NET (Microsoft)	Recientemente
• Es el resultado de la búsqueda de una plataforma independiente del hardware adoptando los conceptos de OO.	• Microsoft lanza su plataforma .NET para el desarrollo de aplicaciones multiplataforma en respuesta a la popularidad de JAVA. Junto con la plataforma aparece el lenguaje C#.	• Ruby, Scala, Phyton, etc.

Ventajas de la POO

- **Reutilización de Código:** La POO facilita la reutilización de código a través de conceptos como la herencia y la composición. Esto permite aprovechar clases existentes para crear nuevas clases, lo que ahorra tiempo y esfuerzo.
- **Organización del Código:** La POO proporciona una estructura organizada para el código al dividirlo en ***clases*** y ***objetos***. Esto mejora la modularidad y facilita la comprensión y mantenimiento del software.
- **Abstracción:** Permite la abstracción de detalles complejos, centrándose en la representación de objetos del mundo real. Esto facilita la comprensión del sistema y la resolución de problemas.
- **Mantenibilidad:** La modularidad y la estructura organizada de la POO facilitan la identificación y corrección de errores. Los cambios y actualizaciones se pueden realizar de manera más eficiente sin afectar otras partes del sistema.
- **Escalabilidad:** La POO facilita la escalabilidad del software al permitir la adición de nuevas funcionalidades mediante la creación de nuevas clases o la extensión de clases existentes.

Desventajas de la POO

- Cambio en la forma de pensar de la programación tradicional, a la orientada a objetos
- La ejecución de programas orientada a objetos es más lenta
- Complejidad para adaptarse
- Dificultad en la abstracción

POO

EN EL PARADIGMA DE OBJETOS, SÓLO HAY **OBJETOS** Y **MENSAJES** (QUE TAMBIÉN SON OBJETOS).

PROGRAMA ORIENTADO A OBJETOS

Un programa en POO es un conjunto de objetos que colaboran enviándose mensajes

CONCEPTOS GENERALES

PROGRAMACIÓN ORIENTADA A OBJETOS

Clase

Una clase es una plantilla mediante la cual se crean los diferentes objetos requeridos para la solución del problema.

Una clase se compone de:

- **Información:** campos (atributos, propiedades)
- **Comportamiento:** métodos (operaciones, funciones)

Ejemplo:

- Clase VEHICULO
- Clase PERSONA

Objeto

Un objeto **es una instancia** de una clase. Por lo tanto, los objetos hacen uso de los ***Atributos*** (variables) y ***Métodos*** (Funciones y Procedimientos) de su correspondiente clase.

Es una variable de tipo clase.

- Por ejemplo
  - el objeto **Ferrari** es un objeto de la clase: **vehículo**.
  - El objeto **Ana** es un objeto de la clase **persona**

Un objeto permite **modelar** entidades del mundo real

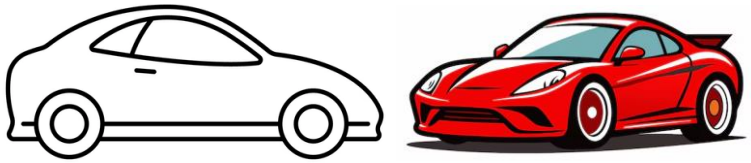
Clases y Objetos

Definición: En POO, una **clase** es un plano para crear objetos, mientras que un **objeto** es una instancia específica de una clase. Las clases definen atributos y métodos comunes para los objetos.

Ejemplos:

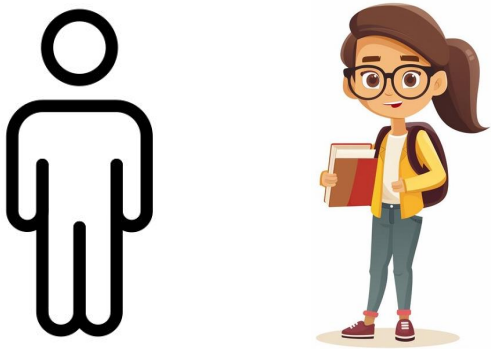
Una clase '**Vehículo**' puede tener

- *atributos* como 'color' y 'modelo', y
- *métodos* como 'arrancar' y parar'."



Una clase '**Persona**' puede tener

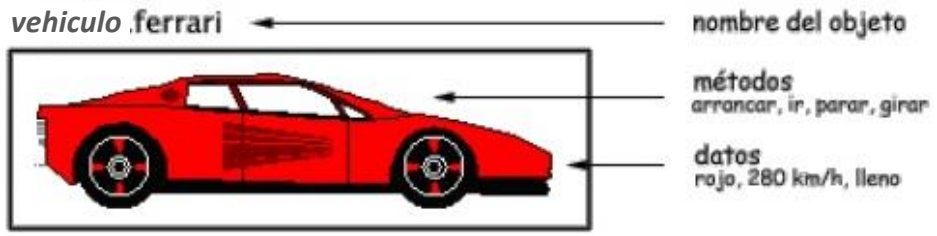
- *atributos* como 'nombre', 'apellido´, 'edad', 'genero'
- *métodos* como 'esMayorEdad'



Clase y Objeto: Ejemplo

Clase:
VEHICULO

- **Objeto**: *Ferrari*



Objetos: Identificación

La primera tarea a la que se enfrenta un programador en POO es la identificación de los objetos inmersos en el problema a solucionar.

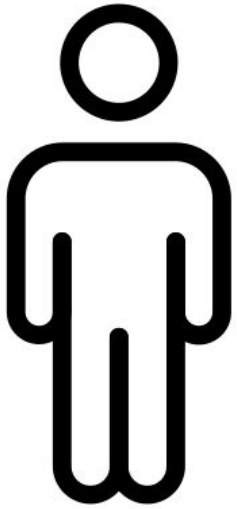
Los objetos generalmente se ubican en las siguientes categorías:

- Cosas tangibles: ***árbol, auto, etc.***
- Cosas intangibles: ***emociones, marca***
- Organizaciones o entidades: ***universidad, empresa de transporte***
- Roles: ***alumno, profesor, etc.***

Objetos y Atributos

Sea la clase PERSONA y el objeto ANA

- El **nombre** de Ana es Ana María
- El **apellido** de Ana es Rodriguez
- La **edad** de Ana es 27
- El **género** de Ana es Femenino



«Atributos, determinan el estado interno de un objeto»

Objetos y Comportamientos

Sea la clase PERSONA y el objeto ANA

- Ana es mayor de edad
- Ana habla
- Ana compra tickets
- Ana viaja en avión
- Ana cumple años

«Comportamiento, determina el protocolo del objeto»

Características de la POO

- ABSTRACCIÓN
- ENCAPSULAMIENTO
- MENSAJES
- HERENCIA
- POLIMORFISMO

Abstracción

- ABSTRACCIÓN
- ENCAPSULAMIENTO
- MENSAJES
- HERENCIA
- POLIMORFISMO

Es una de las principales características a tener en cuenta ya que permite vislumbrar los diferentes agentes u objetos implicados en un problema.

Captar los ***atributos*** y ***métodos*** que conforman cada objeto y la relación que existen entre ellos.

Resolver el problema en subproblemas donde cada objeto se haga cargo de cada subproblema.

La comunicación entre objetos genera la solución general a todo el problema (Divide y vencerás)

ABSTRACCIÓN

ENCAPSULAMIENTO

MENSAJES

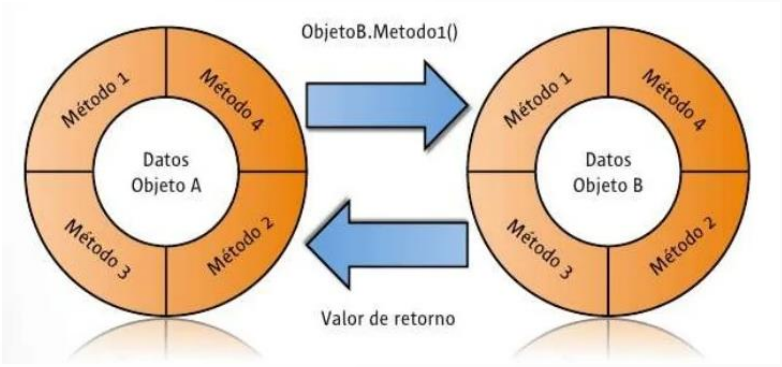
HERENCIA

POLIMORFISMO

Encapsulamiento

Permite la **ocultación** de la información es decir permite asegurar que el contenido de un objeto se pueda ocultar del mundo exterior dejándose ver lo que cada objeto necesite hacer público.

Ejemplo: Una persona desea llevar su auto descompuesto para que sea arreglado por un mecánico



ABSTRACCIÓN

ENCAPSULAMIENTO

MENSAJES

HERENCIA

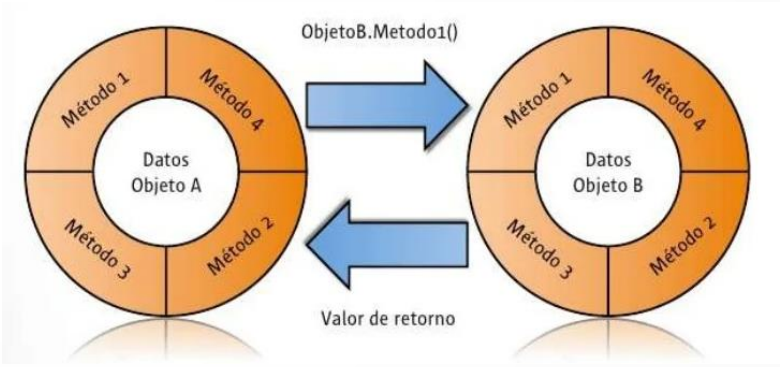
POLIMORFISMO

Mensajes

Un objeto sin comunicación con el mundo exterior no es de utilidad. Los objetos **deben** relacionarse.

Los objetos **interactúan** entre ellos mediante **mensajes**.

Cuando un objeto A quiere que otro objeto B ejecute una de sus funciones o procedimientos (métodos de B), el objeto A manda un mensaje al objeto B.

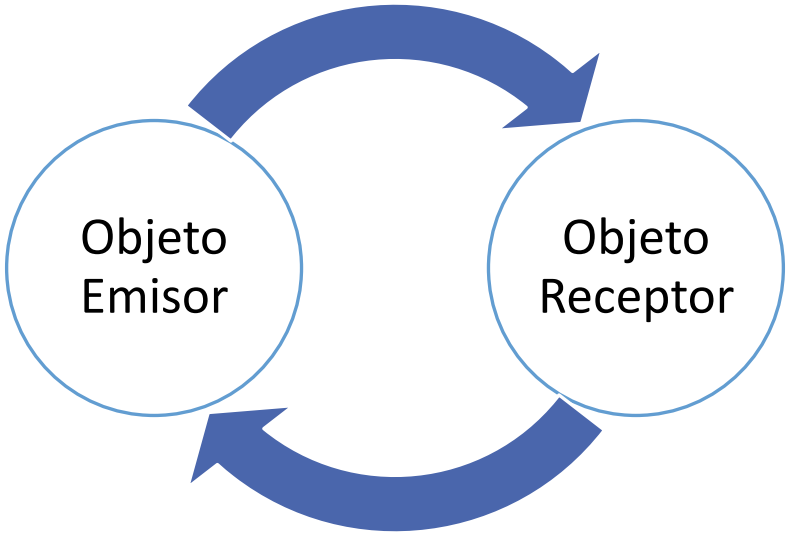


Mensajes

En un mensaje siempre hay un receptor, lo cual no ocurre en una llamada a procedimiento.

Un mensaje consta de 3 partes:

1. **Identidad del receptor**: Nombre del objeto que contiene el método a ejecutar.
2. **Nombre del método a ejecutar**: Solo los métodos declarados públicos.
3. **Lista de Parámetros** que recibe el método (cero o más parámetros)



Herencia

La herencia en POO permite que se defina una jerarquía entre clases y poder compartir atributos y métodos comunes puedan ser reutilizados.

Esta propiedad permite la creación de nuevas clases a partir de una ya existente. La base es la clase existente que sirve de modelo y hereda sus características a las derivadas, las nuevas clases formadas.

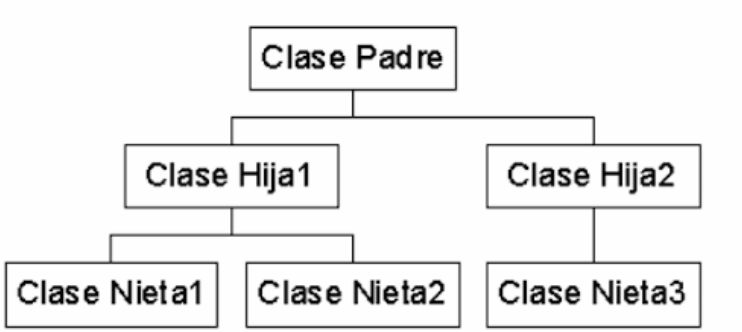


Imagen 1: Ejemplo de árbol de herencia

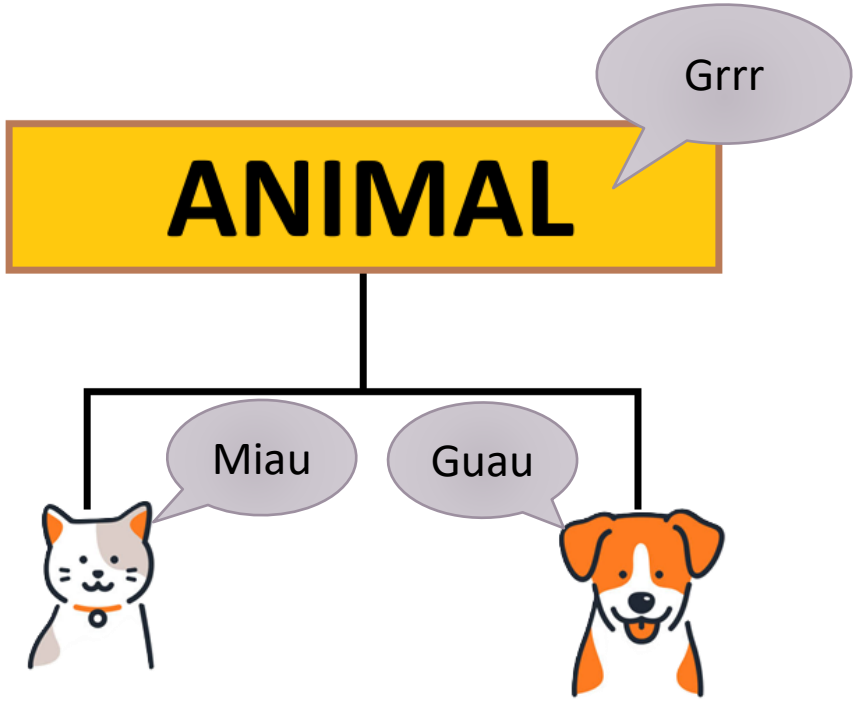
Tipos de herencia

- **Herencia simple:** Una clase posee una sola superclase directa. El gráfico de herencia es un árbol
- **Herencia múltiple:** Una clase posee varias superclases directas. El gráfico de herencia no es un árbol

Polimorfismo

Este permite a una operación o función tener el **mismo nombre** en **clases diferentes** y **actuar** de modo **distinto** en cada una de ellas.

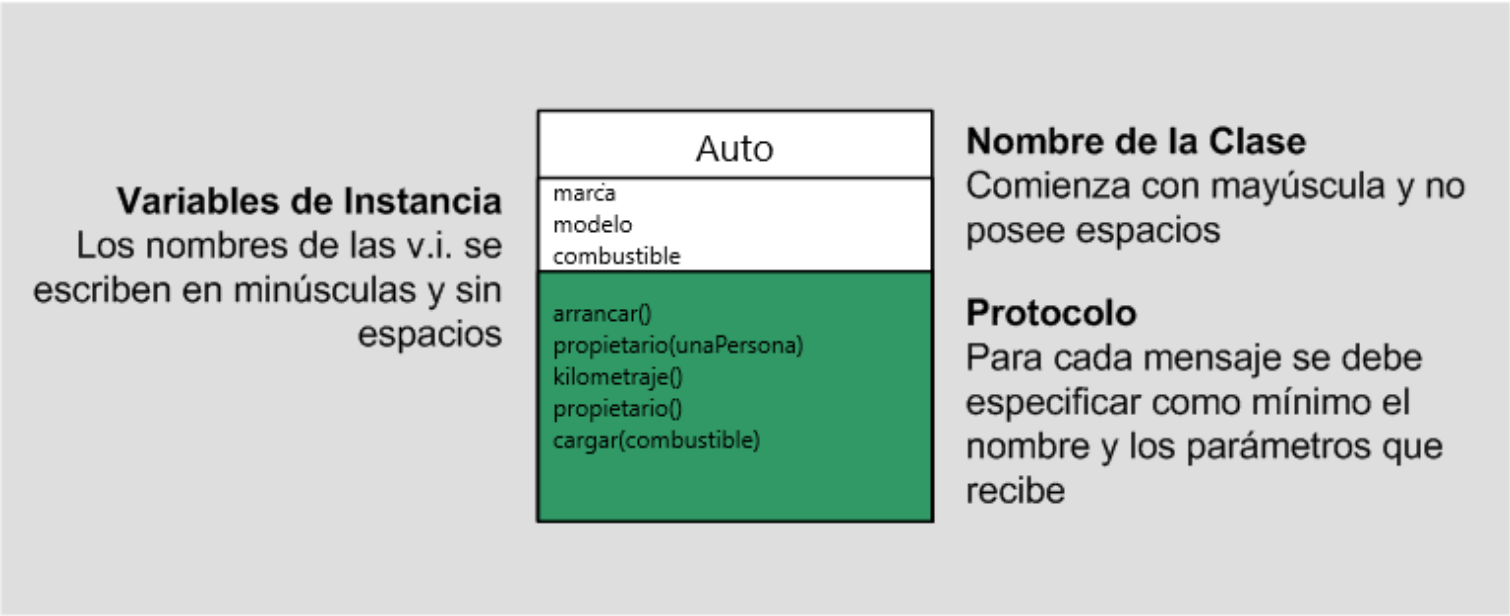
El polimorfismo implica la capacidad de una operación de ser interpretada sólo por el propio objeto que lo invoca. Es decir a pesar de que la función comparta el mismo nombre, tendrá un significado diferente para cada clase y por ende podrá hacer las operaciones que le fueron indicadas en su definición a pesar de ser diferentes.



ESPECIFICACIÓN DE CLASES

Las clases se especifican por medio de un nombre, el estado o estructura interna que tendrán sus instancias y los métodos asociados que definen el comportamiento

Gráficamente:



Definición de una clase en C++

```
1  class Persona //Declaramos la clase con el nombre Persona
2  {
3  private:     //a partir de aquí todos los miembros serán privados
4  //los datos miembro pueden ser cualquier tipo de dato, incluso otras clases como string
5    std::string nombre;
6    int edad;
7    float estatura;
8    float peso;
9  //métodos privados
10   float aumentaEstatura(float metros){return estatura += metros}; //función inline
11   float aumentaPeso(float kilogramos){return peso += kilogramos};
12
13 public:      //a patir de aquí todas las declaraciones serán de acceso público
14   Persona(const std::string& nombre,int edad, float peso, float estatura); // Constructor
15   void saluda();
16   int cumpleAnios();
17 };
18
```

atributos

métodos

Formas de Conocimientos

Para que un objeto conozca a otro lo debe poder nombrar. Decimos que se establece una ligadura (binding) entre un nombre y un objeto.

Algunas de las formas de conocimiento o tipos de relaciones entre objetos son

- **Conocimiento Interno:** Variables de instancia.
- **Conocimiento Externo:** Parámetros.

Además, existe una forma de conocimiento especial: las **seudo variables**.

Variables de Instancia

Define una relación entre un objeto y sus atributos.

Se definen **explícitamente** como parte de la estructura de la clase.

La relación dura tanto tiempo como viva el objeto, o se cambie explícitamente.

```
1  class Persona //Declaramos la clase con el nombre Persona
2  {
3  private:     //a partir de aquí todos los miembros serán privados
4  //los datos miembro pueden ser cualquier tipo de dato, incluso otras clases como string
5    std::string nombre;
6    int edad;
7    float estatura;
8    float peso;
```

Parámetros

Se refiere a los parámetros de un mensaje.

El nombre de la relación se define explícitamente en el nombre del método.

La relación de conocimiento dura el tiempo que el método se encuentra activo.

La ligadura entre el nombre y el objeto no puede alterarse durante la ejecución del método.

```
13  public:      //a patir de aquí todas las declaraciones serán de acceso público
14    Persona(const std::string& nombre,int edad, float peso, float estatura); // Constructor
```

```
46  main(){
47      Persona a;
48      string nom="Ana";
49      Persona emple(nom,22,52,1.56);
```

Constructores

Los constructores son **funciones miembro especiales** que sirven para inicializar un objeto de una determinada clase al mismo tiempo que se declara.

Los constructores son especiales por varios motivos:

- Tienen el mismo nombre que la clase a la que pertenecen.
- No tienen tipo de retorno, y por lo tanto no retornan ningún valor.
- No pueden ser heredados.
- Por último, deben ser **públicos**, no tendría ningún sentido declarar un constructor como privado, ya que siempre se usan desde el exterior de la clase, ni tampoco como protegido, ya que no puede ser heredado.

Constructores

```
1  Persona::Persona(){
2      this -> nombre = "";
3      this -> edad = 0;
4      this -> peso = 0.0;
5      this -> estatura = 0.0;
6  }
7
8  Persona::Persona(const string& nombre,int edad, float peso, float estatura){
9      this -> nombre = nombre;
10     this -> edad = edad;
11     this -> peso = peso;
12     this -> estatura = estatura;
13 }
14
```

Destructores

Los **destructores** son **funciones miembro especiales** que sirven para eliminar un objeto de una determinada clase.

Al igual que los constructores, los destructores también tienen algunas características especiales:

- También tienen el mismo nombre que la clase a la que pertenecen.
- No tienen tipo de retorno, y por lo tanto no retornan ningún valor.
- No tienen parámetros.
- No pueden ser heredados.
- Deben ser **públicos**, no tendría ningún sentido declarar un destructor como privado, ya que siempre se usan desde el exterior de la clase, ni tampoco como protegido, ya que no puede ser heredado.
- No pueden ser **sobrecargados**, lo cual es lógico, puesto que no tienen valor de retorno ni parámetros, no hay posibilidad de sobrecarga.

La Seudo Variable: THIS

Para mandarle un mensaje a un objeto hay que poder nombrarlo.

¿Cómo hace un objeto para mandarse un mensaje a sí mismo?

Seudo variable: como una variable ordinaria pero:

- No se declara
- No puede modificarse

C++: this

Java y C# : this

Smalltalk : self

Con esta seudo variable el objeto puede hacer referencia a sí mismo.

La seudo variable This

```
1  Persona::Persona(){
2      this -> nombre = "";
3      this -> edad = 0;
4      this -> peso = 0.0;
5      this -> estatura = 0.0;
6  }
7
8  Persona::Persona(const string& nombre,int edad, float peso, float estatura){
9      this -> nombre = nombre;
10     this -> edad = edad;
11     this -> peso = peso;
12     this -> estatura = estatura;
13 }
14
```

Get y Set

Los métodos **get** y **set** son patrones de programación orientada a objetos (POO) que permiten controlar el acceso a los atributos privados de una clase.

Los getters (o "obtener") devuelven el valor de un atributo, mientras que los setters (o "establecer") lo modifican, a menudo con lógica adicional como validaciones.

Este mecanismo es una parte fundamental de la encapsulación, ya que protege los datos internos del objeto de un acceso directo y no controlado.

get y set

- **Métodos get (Getter)**
  - **Función:** Obtiene y devuelve el valor de un atributo privado.
  - **Ejemplo de nombre:** getNombre() para un atributo nombre.
  - **Lógica:** Retorna el valor del atributo.
- **Métodos set (Setter)**
  - **Función:** Asigna un nuevo valor a un atributo privado.
  - **Ejemplo de nombre:** setNombre() para un atributo nombre.
  - **Lógica:** Recibe un parámetro y asigna su valor al atributo. Se pueden incluir validaciones para asegurar que el valor es válido antes de asignarlo.

get y set

Ejemplo en C++

```cpp
// C++ programa para demostrar el uso de getters y setters

#include <iostream>
using namespace std;

// Definición de class Empleado
class Empleado {
private:
    // Defino el atributo privado salario
    int salario;

public:
    // Setter
    void setSalario(int s) { salario = s; }
    // Getter
    int getSalario() { return salario; }
};

int main()
{

    // crear objeto de la clase Empleado
    Empleado myObj;

    // set the salario
    myObj.setSalario(1000000);

    // get de salario y su muestra
    cout << "Salario es: " << myObj.getSalario();
    return 0;
}

```

Clases en C++. Ejemplo

[*] Persona.hpp | Persona.cpp | Clase1.cpp

```
1  //Definición de la clase persona -> archivo "Persona.h"
2
3  class Persona //Declaramos la clase con el nombre Persona
4  {
5  private:     //a partir de aquí todos los miembros serán privados
6  //los datos miembro pueden ser cualquier tipo de dato, incluso otras clases como string
7    string nombre;
8    int edad;
9    float estatura;
10   float peso;
11 //métodos privados
12   float aumentaEstatura(float metros){return estatura += metros;} //función inline
13   float aumentaPeso(float kilogramos){return peso += kilogramos;}
14
15 public:      //a patir de aquí todas las declaraciones serán de acceso público
16   Persona();
17   Persona(const std::string& nombre,int edad, float peso, float estatura); // Constructor
18   void saluda();
19   int cumpleAnios();
20 };
21
22
```

Clases en C++. Ejemplo

[*] Persona.hpp | Persona.cpp | Clase1.cpp

```cpp
Persona::Persona(){
  this -> nombre = "NN";
  this -> edad = 0;
  this -> peso = 0.0;
  this -> estatura = 0.0;
}

Persona::Persona(const string& nombre,int edad, float peso, float estatura){
  this -> nombre = nombre;
  this -> edad = edad;
  this -> peso = peso;
  this -> estatura = estatura;
}


// Saludo: la persona saluda y dice sus datos.
void Persona::saluda(){
  cout << "¡Hola! me llamo " << nombre
       << ", tengo " << edad << " años"
       << ", peso " << peso << " kilos"
       << " y mido " << estatura << " metros" << endl;
}
// Cumple años: refleja los cambios en los atributos de la persona al haber pasado un año
int Persona::cumpleAnios(){
  float aumento_peso = 0;
  float aumento_estatura = 0;

  if (edad <= 20){
    aumento_peso = 0.1;
    aumento_estatura = 0.1;
  }
```

Clases en C++. Ejemplo

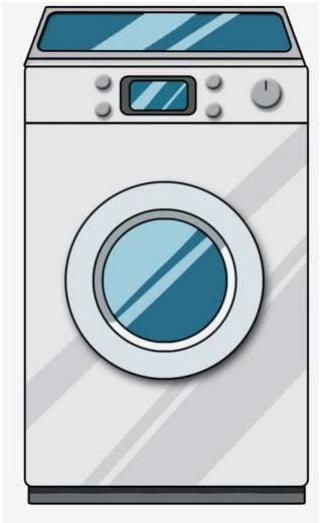
[*] Persona.hpp | Persona.cpp | [*] Clase1.cpp

```cpp
// Archivo fuente de implementación de la clase Persona

#include <iostream> // Incluimos iostream para usar std::cout

using namespace std; // utilizaremos el espacio de nombres std para cout y string

#include "Persona.h" // Incluimos la definición (declaración) de la clase
#include "Persona.cpp"

main(){
    Persona a;
    string nom="Ana";
    Persona emple(nom,22,52,1.56);
    emple.saluda();
    cout<<"ahora tengo: "<<emple.cumpleAnios()<<endl;
    a.saluda();
    return 0;
}

```

Defina la clase (atributos y métodos) para cada uno de los elementos e instancie 1 objeto de cada uno en C++

1

2



3

fin 1ra parte...