

Tema 2. Programación Orientada a Objetos

- a. Clases, objetos y estructuras de datos abstractos
- b. Encapsulación y control de acceso
- c. Métodos y miembros/atributos de la clase
- d. El operador de resolución de ámbito (::)

A. Estructuras

Repaso

Estructuras

- ▶ En C++, se pueden crear tipos de datos que agrupan datos en estructuras usando la palabra reservada **struct**.
- ▶ Para definir una estructura hay que definir un nuevo tipo de datos y declarar una variable de ese tipo.
- ▶ Los datos agrupados se denominan miembros y los miembros corresponden a diferentes variables(y posiblemente tipos)

```
struct TipoEstructura{  
    tipo miembro1;  
    tipo miembro2;  
    //...  
    tipo miembroN;  
} [lista_variables] ;
```

Estructuras

Palabra reservada para definir una estructura

Identificador de la estructura

```
struct TipoEstructura{  
    tipo miembro1;  
    tipo miembro2;  
    //...  
    tipo miembroN;  
} [lista_variables] ;
```

Miembros de la estructura

```
struct Contacto{  
    string nombre;  
    string apellidos;  
    string direccion;  
    string empresa;  
    long telefono;  
    int fechaNacimiento[3];  
} miContacto, libreta[30];
```

```
struct Fruta{  
    string nombre;  
    string variedad;  
    string color;  
    string forma;  
    double peso;  
    int vencimiento[3];  
} producto1[30], producto2[17];
```

Estructuras

```
typedef struct {  
    tipo miembro1;  
    tipo miembro2;  
    //...  
    tipo miembroN;  
} TipoEstructura ;
```

- ▶ C++ también permite definir a la estructura como un nuevo tipo de variable.
- ▶ Para ello ha de utilizarse la palabra reservada ***typedef*** al definir la estructura.
- ▶ En lugar de declarar variables, el nombre de la estructura debe escribirse al final de la definición de la estructura.
- ▶ Gracias a esto se puede utilizar el nombre de la estructura para inicializar en cualquier momento del programa (definiéndola en la static).

```
typedef struct {  
    int dia;  
    int mes;  
    int anho;  
} Fecha;
```

```
typedef struct {  
    unsigned short dia;  
    unsigned short mes;  
    unsigned short anho;  
} Fecha;
```

```
typedef struct {  
    string calle;  
    unsigned short numero;  
    unsigned short planta;  
    unsigned char puerta;  
    unsigned short codigoPostal;  
} Domicilio;
```

```
Fecha miCumple = {25, 12, 2001};
```

Estructuras. Acceso

```
typedef struct {  
    string nombre;  
    string apellidos;  
    string direccion;  
    string empresa;  
    long telefono;  
    int fechaNacimiento[3];  
} Contacto;
```

```
Contacto miContacto;  
miContacto.nombre = "Carlo";  
miContacto.apellidos = "Ancelotti";  
miContacto.empresa = "Real Madrid";
```

```
Contacto libreta[30];  
libreta[0].nombre = "Miguel";  
libreta[0].apellidos = "Almirón";  
libreta[0].empresa = "Newcastle FC";
```

- ▶ El acceso a los miembros de la estructura se realiza mediante el operador .
- ▶ El acceso permite leer/escribir el dato, esto significa que es compatible con todas las operaciones que permita el tipo que define al miembro.
- ▶ Mediante el acceso . también es posible asignar valores a los miembros de una estructura.

Estructuras. Acceso

```
typedef struct {  
    string nombre;  
    string apellidos;  
    string direccion;  
    string empresa;  
    long telefono;  
    int fechaNacimiento[3];  
} Contacto;
```

- ▶ Ya que una estructura puede contener a un miembro de cualquier tipo, una estructura también puede contener otra estructura.
Por ejemplo, en lugar de que sea un vector de 3 enteros, la fecha puede cambiarse a una estructura de la siguiente forma:

```
typedef struct {  
    unsigned short int dia;  
    unsigned short int mes;  
    unsigned short int anho;  
} Fecha;
```

```
Contacto miContacto;  
miContacto.nombre = "Carlo";  
miContacto.apellidos = "Ancelotti";  
miContacto.empresa = "Real Madrid";
```

Estructuras

```
typedef struct {  
    char nombre[10];  
    char apellidos[30];  
    char direccion[40];  
    char empresa[20];  
    long telefono;  
    Fecha fechaNacimiento;  
    Fecha fechaFinalContrato;  
} Contacto;
```

```
Contacto miContacto;  
miContacto.nombre = "Carlo";  
miContacto.apellidos = "Ancelotti";  
miContacto.empresa = "Real Madrid";
```

- Por supuesto, es posible reutilizar las estructuras y tener más de una estructura del mismo tipo dentro de otra.

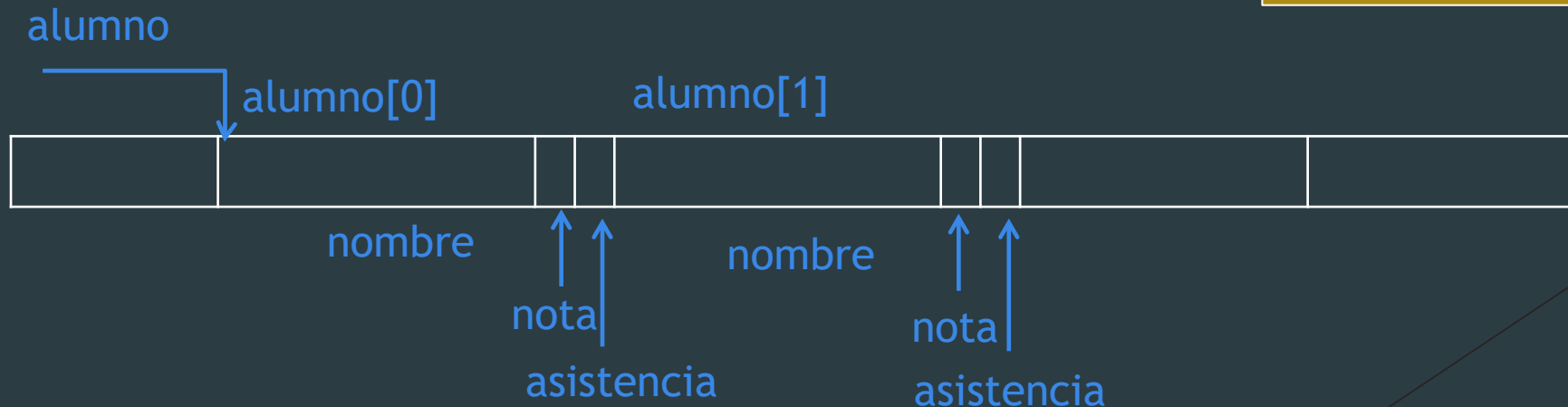
```
Contacto libreta[30];  
libreta[0].nombre = "Miguel";  
libreta[0].apellidos = "Almirón";  
libreta[0].empresa = "Newcastle FC";  
libreta[0].fechaNacimiento = {10, 2, 1994};
```

```
libreta[1].nombre = "Kyllian";  
libreta[1].apellidos = "Mbappe";  
libreta[1].empresa = "Real Madrid FC";  
libreta[1].fechaFinalContrato = {99, 13, 2024};
```


Estructuras en memoria

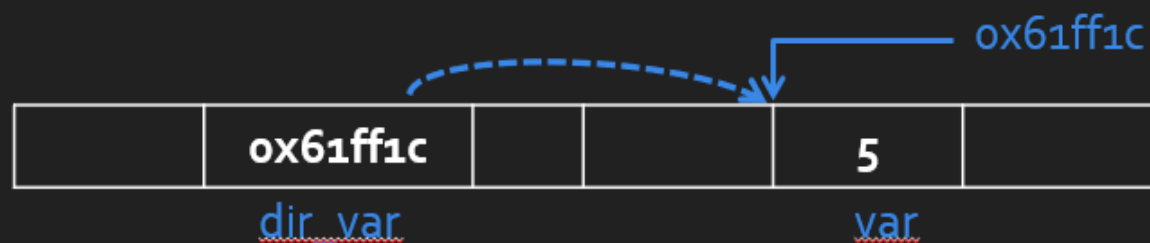
- ▶ Las estructuras se crean con los miembros uno al lado del otro.
- ▶ Con esto, es más fácil para el computador de modificar/crear/etc un conjunto de datos relacionados
- ▶ También es posible referirse a la dirección de memoria de una estructura

```
typedef struct{  
    string nombre;  
    float nota;  
    float asistencia;  
} tFicha;  
tFicha alumno[100];  
alumno[0].nombre = "Fede";  
alumno[0].nota = 10.0f;  
alumno[0].asistencia = 69.99f;
```



Punteros

```
int var = 5;  
int* dir_var;  
dir_var = &var;
```



Segmento de memoria de n bytes

- El puntero siempre *apunta* al comienzo de la variable.
- Esto permite modificar directamente una dirección de memoria sin crear copias: mejorando el rendimiento y la rapidez.

Punteros a estructuras



- Los punteros a estructuras se declaran igual que los punteros a otros tipos de datos.

```
Cuenta cuentaUsuario, *punteroCuenta;  
punteroCuenta = &cuentaUsuario;
```

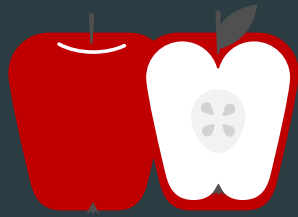
- La diferencia de uso erradica en que para referirse a un miembro de una estructura apuntada por un puntero hay que utilizar el operador -> o bien indireccionar el puntero (* estructura).

```
punteroCuenta->nombre = "Maria";
```

```
(* punteroCuenta).nombre = "Maria";
```

Propiedades de un objeto real

- ▶ ¿Estructuras, para qué?
- ▶ La descripción de objetos y cosas es más sencilla al enfocarse en los detalles que diferencian un objeto de otro.



- ▶ Color: Rojo
- ▶ Forma: Trapezoidal
- ▶ Peso: 136 g
- ▶ Nombre común: Manzana
- ▶ ...



- ▶ Color: Verde
- ▶ Forma: Trapezoidal
- ▶ Peso: 145 g
- ▶ Nombre común: Manzana Verde
- ▶ ...



- ▶ Color: Verde-Amarillo
- ▶ Forma: ¿Unión de Elipses?
- ▶ Peso: 87 g
- ▶ Nombre común: Pera
- ▶ ...

Propiedades de un objeto

- ▶ Agrupando estas propiedades definimos esta estructura simple.
- ▶ Al tener, por ejemplo, la necesidad de guardar en un ordenador estas estructuras es posible usar C++ (y muchos otros lenguajes), de tal forma a que las propiedades de una estructura se “mantengan unidas”.
- ▶ Porque, por ejemplo, si queremos guardar los datos de contacto de personas que conozcamos en un evento es más fácil agruparlos datos en propiedades que tenerlos todos juntos de la siguiente forma:

```
string nombres[100], apellidos[100] ;  
string  empresa[100];  
unsigned long numeroMovil[100];  
  
unsigned short diaCumple[100];  
unsigned short mesCumple[100], anhoCumple[100];
```



```
typedef struct{  
    int dia, mes, anio;  
} Fecha;  
typedef struct{  
    string nombre, apellido, empresa;  
    unsigned long numeroMovil;  
    Fecha fechaNacimiento;  
} Contacto;  
  
Contacto agenda[100];
```

- ▶ Pero las estructuras tienen algunas desventajas importantes como seguridad limitada, manejo de datos, funcionalidad, reutilización de código. Estas limitaciones pueden ser solucionadas Orientando la Programación para describir Objetos.

A. Clases y Objetos



Según René Magritte
en La traición de las
Imágenes:
“Esto no es una pipa”

¿Qué es programar?

- ▶ Crear una serie de instrucciones lógicas que permite a un computador realizar cálculos y tomar decisiones lógicas y deterministas.

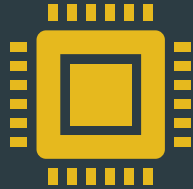
¿Cómo se programa?

- ▶ A través de los años, los programadores diseñaron y estructuraron diferentes formas o modelos a seguir para escribir programas.
- ▶ Inicialmente los programas se realizaban línea por línea, bajo el pilar fundamental de las estructuras de secuencia, condiciones e iteraciones. A este paradigma de la programación se le ha denominado **Programación Estructurada**.
- ▶ Con el paso del tiempo, la programación se ha hecho más compleja y de escritura de mayor nivel (mejor abstracción del software). Este tipo de programación permitía dividir el programa en módulos y sub-módulos específicos caracterizados por la independencia: La **Programación Modular**.

¿Cómo se programa?



Programación Funcional:
La evolución natural de la programación modular consistió en reutilizar los módulos mediante llamadas y accesos con elementos de entrada y de salida de forma a interconectar los módulos. Se trabajaba de la misma forma que una función matemática $y = f(x)$.



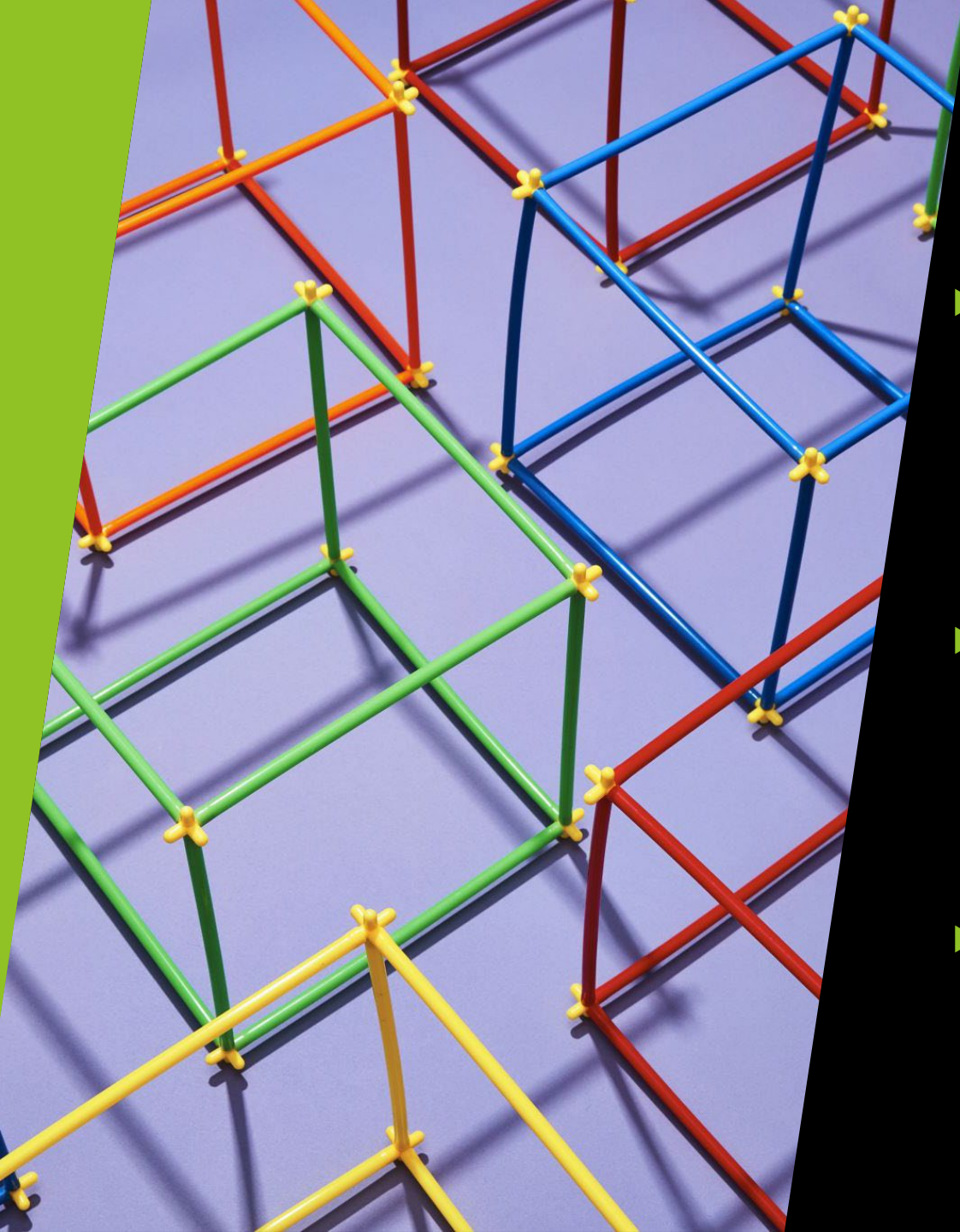
Con el fin de simplificar aún más el desarrollo de software, se popularizó en los '90 un paradigma de programación que se enfocaba en agrupar los comportamientos similares para definir **objetos** similares.



Por ejemplo, es útil crear una sola vez código que defina el comportamiento de una pelota de voleibol y una de baloncesto, que comparten muchas características. Mientras que, por ejemplo, las características compartidas de estos dos **objetos** con una silla son casi nulas.



Partiendo del término de “objeto” se ha desarrollado lo que hoy se conoce como **Programación Orientada a Objetos**.



Programación Orientada a Objetos

- ▶ Entonces la POO no es otra cosa que un **paradigma** que busca agrupar las informaciones de objetos que compartan campos, atributos o propiedades mediante el uso de técnicas de programación ya existentes como funcionales o estructuradas y técnicas específicas definidas basadas en la definición de objetos.
- ▶ Así también POO permitiría crear no solo objetos parecidos, sino también dos objetos idénticos pero diferenciados por su identidad. Ejemplo: POO se puede utilizar para crear dos sillas de características iguales, pero diferenciadas por su existencia misma.
- ▶ A la **plantilla de código que genera los objetos se denomina Clase**, así una clase Silla agruparía todos los atributos, propiedades y métodos que posee una silla real.

Clases y Objetos

- La analogía con el mundo real es que las Clases son organizaciones imaginarias y los objetos son elementos reales:

| |
|------|
| Pipa |
| |
| |
| |



Clases y Objetos

Clase

| |
|--|
| Identificador: Pipa |
| <i>Atributos: Características de las pipas</i> |
| <i>Métodos: Funciones de las pipas</i> |

Objeto

| |
|---|
| <code>pipaMagritte</code> : Pipa |
| <i>Características específicas del objeto real de la clase Pipa</i> |

Clases

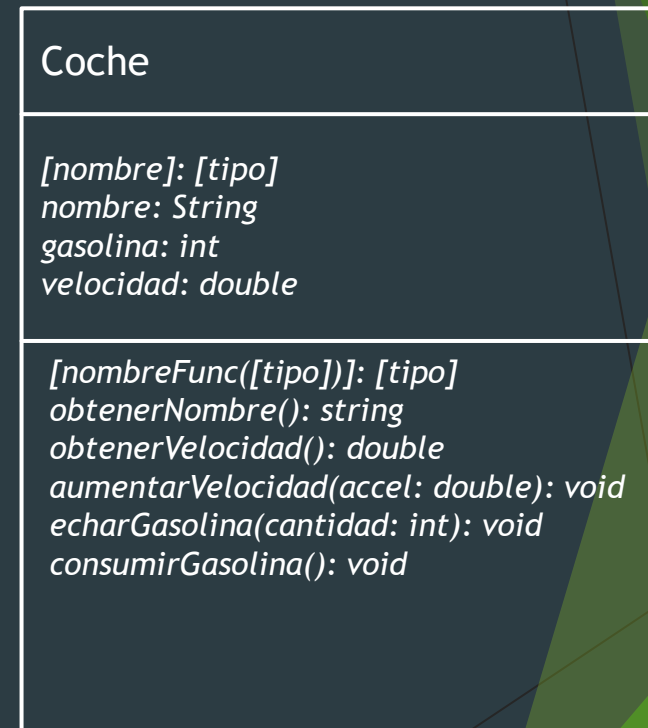
- Para describir una clase recurrimos en primera instancia a un diagrama llamado Diagrama UML con tres secciones: Identificador, Atributos y Métodos.

Identificador: Nombre de la clase. Se escribe según PascalCase: Coche, TelefonoMovil, CuentaDeBanco

Atributos: características que describen el objeto y forman parte siempre del mismo.
snake_case: nombre, numero_de_telefono.
Seguidamente el tipo: string, long.

Métodos: (funciones miembro) operaciones que describen el comportamiento de los objetos en el mundo y cómo se va a realizar la interacción humano-objeto. Se escribe de la misma forma en que se define una función.

- Identificador camelCase: getNombre
- Parámetros (tipo1, tipo2): (int, float*, ...)
- Resultados tipo: void, double, ...



Objetivo: Combinar (encapsular) en una sola entidad tanto los datos como las funciones que operan (manipulan) los datos.

Clases y Objetos

- ▶ La clase define de forma generalizada, el objeto es una **instancia** específica, nombrada y alojada en memoria tal como cualquier tipo primitivo.
- ▶ Los objetos son **instancias** de las clases.
- ▶ Los tipos primitivos no tienen atributos ni métodos, por lo que no son Clases.
- ▶ Las estructuras, por otro lado, tienen miembros y soportan métodos, pero no permiten implementar muchas de las técnicas que se aprenderán en este curso.

Coche

[nombre]: [tipo]
nombre: String
gasolina: int
velocidad: double

[nombreFunc([tipo])]: [tipo]
obtenerNombre(): string
obtenerVelocidad(): double
aumentarVelocidad(accel: double): void
echarGasolina(cantidad: int): void
consumirGasolina(): void

Mazda rx-8: Coche

nombre = Mazda rx-8
gasolina = 50
velocidad = 210,0

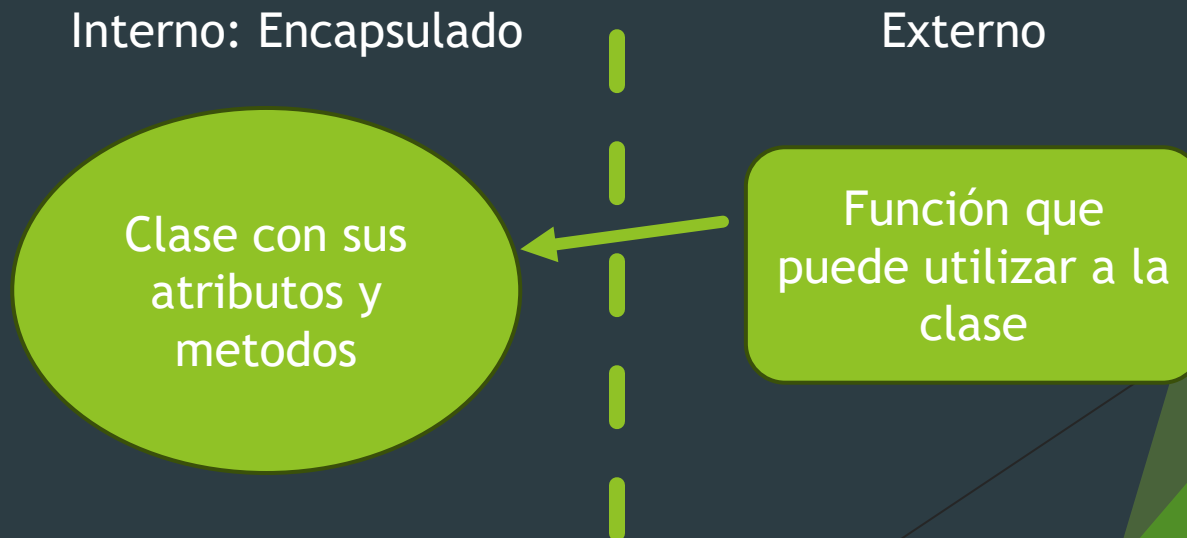
Ferrari Enzo: Coche

nombre = Ferrari Enzo
gasolina = 110
velocidad = 366,0

Clases. Encapsulación

- ▶ El paradigma por seguir para definir clases se basa en la encapsulación de datos.
- ▶ La clase debe definir absolutamente todos los datos necesarios para su uso, pero no así los datos y funciones que dependan de la clase. “Las clases encapsulan o envuelven exclusivamente todos los datos necesarios para su funcionamiento.” El acceso a los atributos se hará mediante funciones o métodos específicos.
- ▶ Ejemplo, la clase coche debe definir el atributo **gasolina** y la función **acelerar**, pero no debe definir la variable **destino** o la función **calcularRutaA**, debido a que el coche funciona con o sin un destino asociado.

| Coche |
|---|
| <i>[nombre]: [tipo]</i> <i>nombre: String</i> <i>gasolina: int</i> <i>velocidad: double</i> |
| <i>[nombreFunc([tipo])]: [tipo]</i> <i>obtenerNombre(): string</i> <i>obtenerVelocidad(): double</i> <i>aumentarVelocidad(accel: double): void</i> <i>echarGasolina(cantidad: int): void</i> <i>consumirGasolina(): void</i> |



Ejercicios



Haga un diagrama que defina a la Clase: Mesa junto con sus atributos y métodos.



Haga un diagrama que defina a una clase que pueda describir un objeto cualquiera de un hobby suyo. Ejemplo: Lienzo, Pala de padel, Club de Fútbol, Indique por lo menos dos atributos y dos métodos.

Clase o Estructura?

- ▶ A pesar de la similitud, las clases son mucho más complejas que las estructuras y deben ser utilizadas bajo la premisa de que las funciones miembro o métodos son realmente necesarios de describir.
- ▶ La ficha de un alumno podría ser descrita con tipos primitivos simples y la ficha por sí misma no realiza ninguna operación o método interno para su funcionamiento correcto.
- ▶ Por otro lado, un coche necesita ser definido en una clase ya que un coche que no interactúa con el mundo (no se mueve, no consume gasolina para mantener el motor encendido, etc..) no es un coche real.
- ▶ En definitiva, las estructuras deben usarse cuando sus miembros no varían internamente a la estructura misma, mientras que las clases deben usarse cuando sus atributos pueden obtenerse o modificarse **interna o externamente** mediante funciones específicas.

Clase. Acceso

- ▶ Para modificar externamente un atributo (clase) o un miembro (estructura) se puede acceder a la variable específica de la misma forma:
objeto.atributo1 estructura.miembro1
- ▶ Pero hay veces que no se desea que los atributos de una clase puedan modificarse desde cualquier parte del programa. Por ejemplo, en el mundo real no podemos modificar directamente la velocidad a la que va un coche.
- ▶ La velocidad de un coche puede ser del tipo float, pero su valor siempre depende de otras variables tanto internas como externas, que hacen que esta variable no se pueda cambiar de forma directa.
- ▶ Entonces es necesario crear una clase cuando existen atributos y métodos que no puedan ser accedidos directamente. Esto es conocido como **data hiding** que sigue a la definición de **encapsulamiento**. La variable velocidad de un coche es una variable interna que no debe ser modificada desde afuera de la clase.

Clase. Miembros privados y públicos

- ▶ Se define formalmente a los atributos **privados** y funciones **privadas** a aquellos elementos de una clase que no pueden ser *accedidos* por fuera de la clase debido a su encapsulamiento.
- ▶ A los elementos que pueden ser *accedidos* desde cualquier parte del programa se los denomina miembros **públicos**.
- ▶ Para diferenciarlos en el diagrama, se agrega el símbolo - o + según el *acceso* a elementos sea privado o público.
- ▶ En la gran mayoría de las veces, los atributos son privados y los métodos públicos.

Coche

- *nombre: String*
- *gasolina: int*
- *velocidad: double*
+ *matricula: string*

+ *obtenerNombre(): string*
+ *definirNuevoNombre(nuevo: string): void*
+ *obtenerGasolina():int*
+ *echarGasolina(cant: double):void*
+ *pisarElAcelerador(fuerza: double): void*
+ *pisarElFreno(fuerza: double):void*
- *consumirGasolina(cant: double): void*
- *girarRuedas(velocidad: float): void*

Ejercicios

1. Modifique las 2 clases creadas en el ejercicio anterior tal que los diagramas presenten los atributos y métodos con el acceso que mejor se adecue.

Clases. Definición.

- ▶ En C++, existen varias herramientas que nos ayudan a crear/definir las clases.
- ▶ Principalmente, se utilizará la palabra reservada `class` para definir clases.
- ▶ Para definir una clase hacemos uso de 2 ficheros fundamentales, el fichero de cabecera(header) y el fichero de origen (implementación).
- ▶ **El header traduce el diagrama de la clase diseñada a código compilable en C++.**
- ▶ Pero la implementación se hará en el fichero de implementación.
- ▶ La normativa de C++ indica que han de crearse 2 ficheros con el mismo nombre con diferente extensión: un header `clase.h` y una implementación `clase.cpp`

Clases. Header.

```
#include <iostream>
using namespace std;
```

Coche

- *nombre: string*
- *gasolina: double*
- *velocidad: double*
+ *matricula: string*

+ *obtenerNombre():String*
+ *definirNombre(string):void*
+ *obtenerGasolina():int*
+ *echarGasolina(double):void*
+ *pisarElAcelerador(double): void*
+ *pisarElFreno(double):void*
- *consumirGasolina(double): void*
- *girarRuedas(float): void*

```
class Coche
{
private:
```

```
    string nombre;
    double gasolina;
    double velocidad;
```

```
    void consumirGasolina(double consumicion);
    void girarRuedas(float angulo);
```

```
public:
```

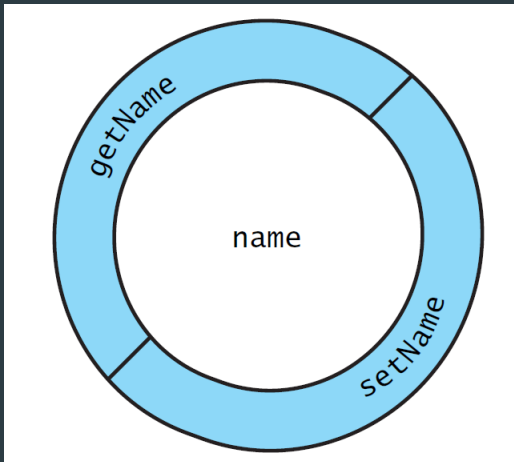
```
    string matricula;
```

```
    string obtenerNombre();
    void definirNombre(string nuevo_nombre);
    double obtenerGasolina();
    void echarGasolina(double litros);
    void pisarElAcelerador(double fuerza);
    void pisarElFreno(double fuerza);
```

```
};
```

Clases. Header.

- ▶ Cuando hablamos de modificadores/selectores de atributos, en POO hablamos de getters y setter.
- ▶ Casi siempre todos los atributos de una clase deben poder accederse para lectura y modificación mediante funciones miembro.
- ▶ Por convención, se utilizan getName y setName como nombres, así respetamos la encapsulación de datos en POO:



```
#include <iostream>
using namespace std;

class Coche
{
private:
    string nombre;
    double gasolina;
    double velocidad;

    void consumirGasolina(double consumicion);
    void girarRuedas(float angulo);

public:
    string matricula;

    string getNombre();
    void setNombre(string nuevo_nombre);
    double getGasolina();
    void setGasolina(double litros);
    void pisarElAcelerador(double fuerza);
    void pisarElFreno(double fuerza);
};
```

Clases. Header.

- ▶ Toda clase está conformada por su nombre, sus atributos y métodos. El acceso a los diferentes atributos y métodos puede ser tanto privado o público. Más adelante, también estudiaremos los accesos protegidos.
- ▶ El diagrama de clases representa enteramente al fichero de cabecera, el cual lleva por nombre: NombreClase.h (Corresponde a PascalCase)
- ▶ Los atributos se escriben con snake_case
- ▶ Los métodos se escriben con camelCase

| NombreClase |
|--|
| - <i>atributos privados</i> + <i>atributos públicos</i> |
| - <i>Métodos privados</i> + <i>Métodos públicos</i> |

```
<incluir librerías necesarias>

class NombreClase
{
private:
    <atributos privados>

    <métodos privados>

public:
    <atributos públicos>

    <métodos públicos>
};
```

Clases. Implementación. Resolución de Ámbito

- ▶ La implementación se lleva a cabo en el fichero.cpp.
- ▶ Para trabajar con todos los miembros de la clase, es necesario usar el operador de resolución de ámbito.
- ▶ Se debe **resolver el ámbito** usando el operador `::` para implementar las funciones.
- ▶ ¡Cuidado! Los atributos ya se encuentran definidos en este fichero.h (y son utilizables en el .cpp)

| Coche |
|--|
| ... |
| + <i>getNombre():string</i> + <i>setNombre (string):void</i> + <i>getGasolina():int</i> + <i>setGasolina(double):void</i> + <i>pisarElAcelerador(double): void</i> + <i>pisarElFreno(double):void</i> - <i>consumirGasolina(double): void</i> - <i>girarRuedas(float): void</i> |

```
#include "Coche.h"

string Coche::getNombre()
{
    return nombre;
}

void Coche::setNombre(string nuevo_nombre)
{
    nombre = nuevo_nombre;
}

double Coche::getGasolina()
{
    return fuel;
}

.....
```


Ejercicios

1. Cree en C++ el header y la implementación de la siguiente Clase

MesaCuadrada

- *patas: int*
- *lado: float*
- *color: string*

+ *getPatas():int*
+ *setPatas(new_patas: int):void*
+ *getLado():float*
+ *setLado(new_lado: float):void*
+ *getColor():string*
+ *setColor(new_color: string):void*

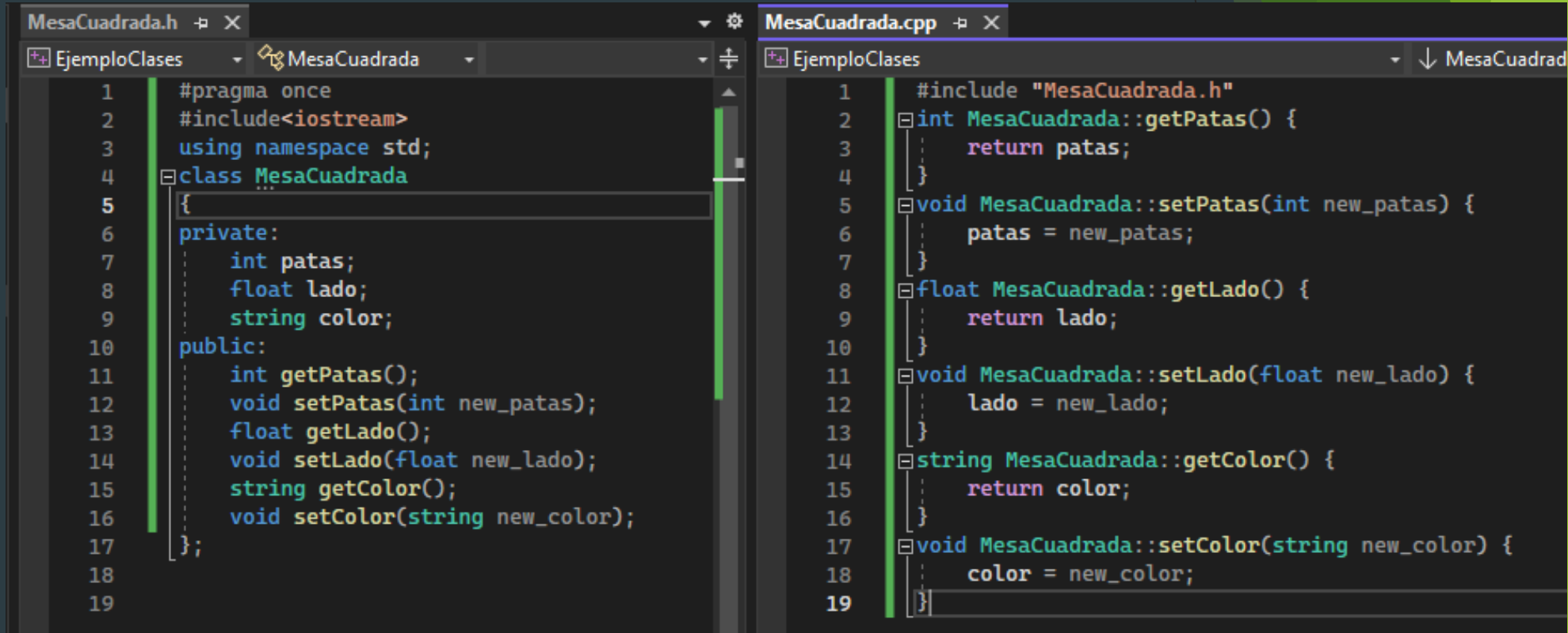
Ejercicios

1. Cree en C++ el header y la implementación de la siguiente clase

MesaCuadrada

- *patas*: int
- *lado*: float
- *color*: string

- + *getPatas()*:int
- + *setPatas(new_patas:int):void*
- + *getLado()*:float
- + *setLado(new_lado:float):void*
- + *getColor()*:string
- + *setColor(string new_color):void*



The image shows a C++ IDE with two files open: `MesaCuadrada.h` and `MesaCuadrada.cpp`. The `MesaCuadrada.h` file contains the class declaration for `MesaCuadrada`, including private attributes (`patas`, `lado`, `color`) and public methods (`getPatas`, `setPatas`, `getLado`, `setLado`, `getColor`, `setColor`). The `MesaCuadrada.cpp` file contains the implementation of these methods, including the inclusion of the header file and the definition of each method.

```
MesaCuadrada.h
1 #pragma once
2 #include<iostream>
3 using namespace std;
4 class MesaCuadrada
5 {
6 private:
7     int patas;
8     float lado;
9     string color;
10 public:
11     int getPatas();
12     void setPatas(int new_patas);
13     float getLado();
14     void setLado(float new_lado);
15     string getColor();
16     void setColor(string new_color);
17 };

MesaCuadrada.cpp
1 #include "MesaCuadrada.h"
2 int MesaCuadrada::getPatas() {
3     return patas;
4 }
5 void MesaCuadrada::setPatas(int new_patas) {
6     patas = new_patas;
7 }
8 float MesaCuadrada::getLado() {
9     return lado;
10 }
11 void MesaCuadrada::setLado(float new_lado) {
12     lado = new_lado;
13 }
14 string MesaCuadrada::getColor() {
15     return color;
16 }
17 void MesaCuadrada::setColor(string new_color) {
18     color = new_color;
19 }
```

Uso de Clases.



- ▶ Para el uso de las Clases basta con incluir el header en el programa principal, tal y como lo hacíamos para estructuras.
- ▶ Para modificar un dato siempre hacemos uso del **modificador** correspondiente (getter), lo mismo va para **seleccionar** un dato (setter).
- ▶ Declarar no es lo mismo que asignar.
- ▶ En este programa, solamente estamos declarando que existe un objeto llamado `mi_mesa` de la clase `MesaCuadrada`.

```
#include<iostream>
#include "MesaCuadrada.h"
using namespace std;

int main() {
    MesaCuadrada mi_mesa; // declaración
    mi_mesa.patas = 4; // ERROR!!!
    mi_mesa.setPatas(4); // setters
    mi_mesa.setColor("Blanca");
    mi_mesa.setLado(7.0);
    cout << "La mesa tiene:" << endl; // getters
    cout << mi_mesa.getPatas() << " patas" << endl;
    cout << mi_mesa.getLado() << "m. de lado" << endl;
    cout << "y es " << mi_mesa.getColor() << endl;
    return 0;
}
```

Uso de Clases.



- Agregar la función pública:
 - obtenerArea(): float

MesaCuadrada

- *patas: int*
- *lado: float*
- *color: string*

+ *getPatas():int*
+ *setPatas(new_patas: int):void*
+ *getLado():float*
+ *setLado(new_lado: float):void*
+ *getColor():string*
+ *setColor(new_color: string):void*
+ *obtenerArea(): float*

```
#include<iostream>
#include "MesaCuadrada.h"
using namespace std;

int main() {
    MesaCuadrada mi_mesa; // declaración
    mi_mesa.setPatas(4); // setters
    mi_mesa.setColor("Blanca");
    mi_mesa.setLado(7.0);
    cout << "La mesa tiene:" << endl; // getters
    cout << mi_mesa.getPatas() << " patas" << endl;
    cout << mi_mesa.getLado() << "m. de lado" << endl;

    cout << "tiene un área de: " << mi_mesa.obtenerArea() << endl;

    cout << "y es" << mi_mesa.getColor() << endl;
    return 0;
}
```

Ejercicios

1. Cree en C++ la siguiente Clase y escriba un programa que demuestre su funcionamiento

```
typedef struct {  
    char cara;  
    int valor;  
} Carta;
```

Mazo

- cartas[48]: Carta
- carta_superior: Carta*
+ numero_de_cartas: 48

+ llenarMazo(): int
+ getCarta(numero: int): Carta
+ robarCarta(): Carta*
+ mezclarMazo(): void

Ejercicio

- ▶ Se busca crear un sistema que permita establecer un ranking de los jugadores a nivel mundial. Para ello diseñe y cree una clase que tenga atributos y métodos relevantes según <https://www.fifaratings.com/players>
- ▶ Atributos: nombre, nacionalidad y ratings de POS ATT SKI MOV POW MEN DEF, GK y OVA
- ▶ El atributo de OVA es el único que no tiene un setter, su valor se calcula en el getter mediante:
 - ▶ La media de los 4 mejores ratings.
- ▶ Cree una lista con todos los nombres de los alumnos (y los profesores) de Fundamentos II/Prog. II y seleccione a los 11 mejores.