Introducción a la programación orientada a objetos

1. Introducción a la programación orientada a objetos
2. Las clases
3. El tipo Struct
4. Diferencias entre Class y Struct
5. Pilares de la Programación Orientada a Objetos
6. Características fundamentales de las clases en C++
7. Representaciones Gráficas
8. Construcción de una clase
9. Atributos
10. Métodos. Implementación de los métodos de una clase
11. Creación de objetos
12. Paso de mensajes
13. Resumen

Introducción

Un programa se encarga de procesar información. El usuario se encarga de dos cosas fundamentales:

- Los datos que se procesan (datos)
- Las operaciones que procesan esos datos (funciones)

Podemos ligar datos con operaciones

Las operaciones se aplican sobre unos datos, más exactamente, sobre unos tipos de datos. Por ejemplo, si tenemos la operación de multiplicar complejos, no tiene sentido multiplicar cadenas.

Si tenemos un array de enteros, podemos recorrerlo, buscar un elemento, etc..

Si tenemos una pantalla, podemos escribir en ella, borrar, etc..

Programación orientada a objetos

Aparece otro paradigma de programación; reconocemos los datos sobre los que vamos a trabajar y definimos esos datos por las operaciones que se pueden realizar sobre ellos.

➡ En la programación orientada a objetos, lo importante son los *objetos*.

Objeto : entidad compuesta de unos datos y las operaciones que realizamos sobre esos datos.

Clase : Los datos y las operaciones comunes a un conjunto de objetos forman un conjunto que se conoce como *clase*.

Las clases

- Las clases C++ permiten definir nuevos tipos de datos,
- Cada clase es un nuevo tipo,
- Cada elemento de la clase se caracteriza por ciertos valores y las operaciones disponibles para crear dichos elementos, modificarlos y destruirlos.

Podemos establecer un paralelismo entre los tipos de datos vistos hasta el momento (int, char, arrays, estructuras,…) y las clases:

1. Ambos son tipos

El tipo **int** está definido por el lenguaje y las operaciones que pueden realizarse están completamente determinadas.	Una clase **CL** no está definida por el lenguaje, por lo que debe ser definida por el programador. Esta definición debe contener dos cosas: *datos* y *operaciones*.

Las clases

2. Pueden declararse elementos de ese tipo

Podemos declararnos elementos de tipo **int**, es lo que llamamos **variables**.	Podemos declararnos elementos de la clase **CL**, en éste caso los llamaremos **objetos** o **instancias** de la clase **CL**.

Esto implica que podemos tener muchos objetos de una misma clase, igual que podíamos definir muchas variables de un tipo.

Pero una clase no es tan simple como un tipo de datos

Una clase es un conjunto de **datos** y un conjunto de **operaciones**, que como hemos dicho, deben ir unidas.

Miembros
- Datos — **Atributos**
- Operaciones — **Métodos**

El tipo Struct

Hemos visto anteriormente cómo implementar un tipo de dato Cliente por medio de una estructura, y unas operaciones para manejarla:

```
struct cliente
    {
        char nombre[30];
        char calle[30];
        double importe_factura;
        char tlf[20] ;
        int edad;
    } ;

cliente c;
void visualizar_datos_cliente ( cliente c );
void modificar_edad ( cliente & c );
void hacer_descuento ( cliente & c, double 0,5 );
```

Miembros

No existe una relación explícita entre las funciones y los datos con los que se trabaja

En C++ podemos asociar las operaciones con los datos simplemente incluyendo las funciones dentro de la definición de la estructura

El tipo Struct

PÚBLICO

```
struct cliente
{
   char nombre[30];
   char calle[30];
   double importe_factura;
   char tlf[20] ;
   int edad;

   void visualizar_datos_cliente ( );
   void modificar_edad ( );
   void hacer_descuento (double);
} ;

cliente c;
```

En C++ podemos asociar las operaciones con los datos simplemente incluyendo las funciones dentro de la definición de la estructura

Las funciones declaradas de esta forma se llaman **métodos** o **funciones miembro.**

Para ejecutar éstos métodos, se ha de utilizar una variable de esa estructura con el operador punto . (acceso habitual a miembros)

```
c.edad = 23 ;
c.visualizar_datos( );
c.hacer_descuento(0.5);
```


Clases y miembros

La estructura anterior proporciona un conjunto de operaciones adecuadas sobre el tipo Cliente, pero no se especifica que esas funciones deban ser las únicas que accedan a los datos del tipo.

Un tipo ***struct*** no evita que se acceda al dato edad y se actualice a 200.

Para que los MIEMBROS, sean privados, utilizamos la palabra ***class*** en lugar de ***struct***.

PRIVADO

```
class  Cliente
{
   char nombre[30];
   char calle[30];
   double importe_factura;
   char telefono[20] ;
   int edad;

   void visualizar_datos_cliente ( );
   void modificar_edad ( );
   void hacer_descuento (double);
} ;

cliente c;
```

Clases y miembros

```
class Cliente
{
   char nombre[30];
   char calle[30];
   double importe_factura;
   char telefono[20] ;
   int edad;
public:
   void visualizar_datos_cliente ( );
   void modificar_edad ( );
   void hacer_descuento (double );
} ;

cliente c;
```

public: Especificador de acceso

El identificador ***public*** separa la clase en dos partes:

- parte privada: donde hemos declarado los datos de la clase.
- parte pública: donde declaramos las funciones de la clase.

Solo podemos acceder a los miembros privados a través de las funciones públicas.

Diferencias entre class y struct

En una clase ***class***, los miembros son privados por defecto, mientras que en una estructura ***struct***, los miembros son públicos por defecto.

```
class Cliente
{
   char nombre[30];
   char calle[30];
   double importe_factura;
   char telefono[20] ;
   int edad;
public:
   void visualizar_datos_cliente ( );
   void modificar_edad ( );
   void hacer_descuento ( double );
} ;
```

Con ésta definición del tipo Cliente, se puede acceder a la parte pública, pero no a la privada.

```
void main()
{
   cliente c;
   c.visualizar_datos_cliente( ) ;
   c.edad = 8 ;
   ...
```

Error de compilación.

Pilares de la programación orientada a objetos

Ocultación de la información

Solo se puede acceder a los datos por medio de las funciones públicas.

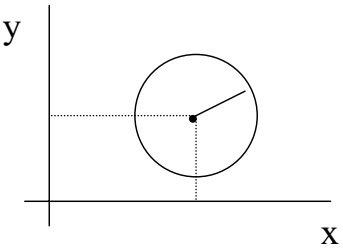
Los datos están ocultos y eso asegura que no se puedan modificar por funciones externas al objeto.

Encapsulamiento de la información

Encapsulación de código y datos.

Ejemplo:

```
class  Circulo
{
 private :
   float coor_x;
   float coor_y;
   float radio;
public :
   void inicializar ( float a, float b, float  r)
      {  coor_x = a;
         coor_y = b;
         radio = r;
      }
   float  visualizar_radio ( )
      {
        return radio;
      }
   float calcular_area (  ) ;
} ;
```



La clase círculo tiene 6 miembros:
- 3 atributos privados
- 3 métodos o funciones miembro

Los métodos pueden implementarse dentro de la clase: se llaman **funciones insertadas**

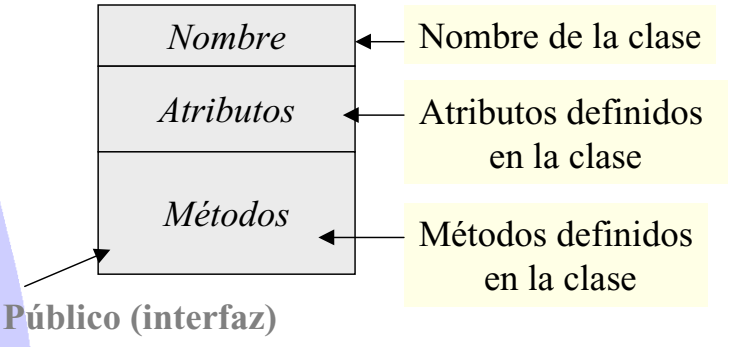
También se puede implementar los métodos fuera de la clase

Características fundamentales de las clases en C++

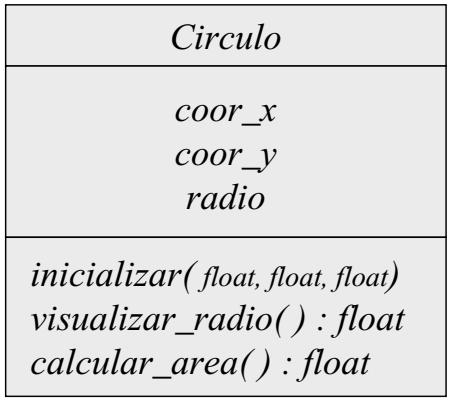
- Nombre de la clase. Sirve para identificar a todos los objetos que tengan unas determinadas características.
- Conjunto de atributos. Datos miembro. El valor de los atributos representan el **estado** de cada objeto.
- Conjunto de métodos. Funciones miembro. Permite que los objetos cambien de estado, dependiendo del estado anterior que tuviera el objeto.
- Niveles de acceso para proteger ciertos miembros de la clase. Normalmente, se definirán como ocultos (privados) los atributos y visibles (públicos) los métodos.

Representaciones gráficas

Para representar gráficamente las clases y objetos, vamos a utilizar diagramas similares a los de UML (Lenguaje de Modelado Unificado).



Ejemplo:



Métodos:
Si la función devuelve un valor, el tipo se pone al final precedido de **:**

Construcción de una clase

La definición de una clase es sintácticamente similar a una estructura (**struct**).

```
class  <nombre_de_la clase>
{
  <cuerpo de la clase>
 } ;
```

```
class  <nombre_de_la clase>
{
  private:
        <lista miembros privados (atributos y métodos)>

  public:
        <lista miembros públicos (atributos y métodos)>
 } ;
```

Especificadores de acceso

Ejemplo

Vamos a modelar una clase cuyos objetos sean las cuentas de un banco.

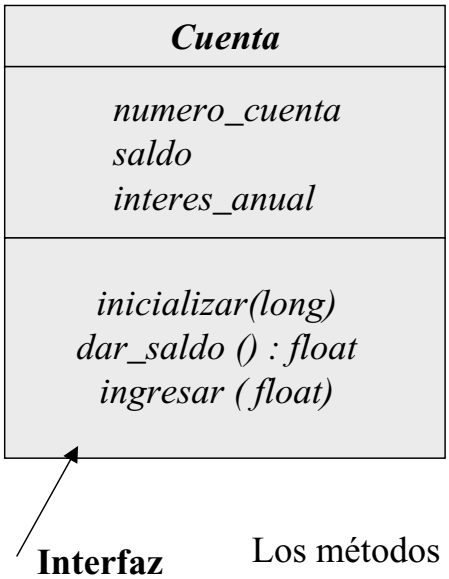
Cuenta
numero_cuenta *saldo* *interes_anual*
inicializar(long) *dar_saldo () : float* *ingresar (float)* *visualizar_datos()*

La clase se llama Cuenta.

¿Qué información ha de contener los objetos de la clase? Es decir, cada una de las cuentas.

¿Qué operaciones se van a necesitar para gestionar las cuentas?

Ejemplo




```
class Cuenta
{
  private:
     long int numero_cuenta;
     float saldo;
     float interes_anual;
  public:
     …
     ...

 } ;
```

Interfaz

Los métodos o funciones miembro, se declaran dentro de la clase.
La implementación de dichos métodos,
puede hacerse dentro (funciones insertadas) o fuera.

Atributos (datos miembro)

Los atributos describen el estado de un objeto.

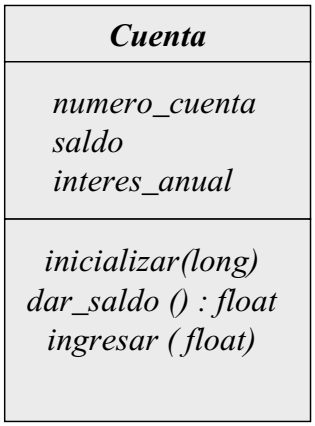
Un atributo consta de dos partes: nombre del atributo y valor

Los atributos de una clase pueden ser:

- de cualquier tipo básico (int, float, bool,…)
- de tipo compuesto (estructura)
- un puntero
- un array
- objetos de otra clase

```
 class Cuenta
{
  private:
     long int numero_cuenta;
     float saldo;
     float interes_anual;
  public:
          …..
}
```

Ejemplo




```
class Cuenta
{
   private:
      long int numero_cuenta;
      float saldo;
      float interes_anual;
   public:
      void inicializar(long int num)
        {
          numero_cuenta = num;
          saldo = 0;
          interes_anual = 0;
        }
      float dar_saldo ();
      void ingresar( float);
} ;
```

Función insertada, no acaba en **;**

Implementación de los métodos fuera de la clase

Prototipos

Métodos definidos interna y externamente

Los métodos se definen de la misma forma que las funciones normales. Lo normal es que dentro del cuerpo de la clase, solo aparezcan los prototipos, y la implementación de las funciones fuera. **Así los detalles de la implementación permanecen ocultos y la interfaz queda mucho más clara.**

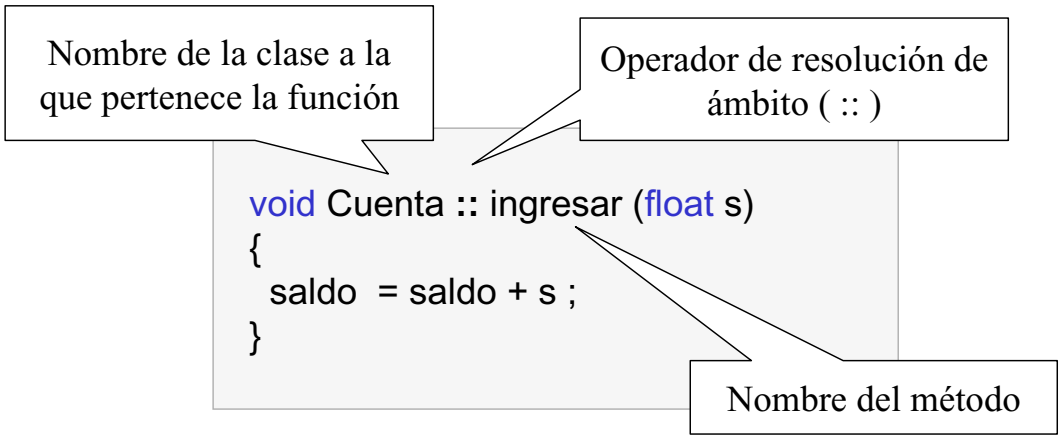
```
class Cuenta
{
   private:
      long int numero_cuenta;
      float saldo;
      float interes_anual;
   public:
      void inicializar(long int num);
      float dar_saldo ();
      void ingresar(float);
} ;
```

```
 ...
void Cuenta :: ingresar (float s)
{
  saldo  = saldo + s ;
}

float Cuenta :: dar_saldo ()
{
    return saldo ;
}
```

Implementación de los métodos fuera de la clase

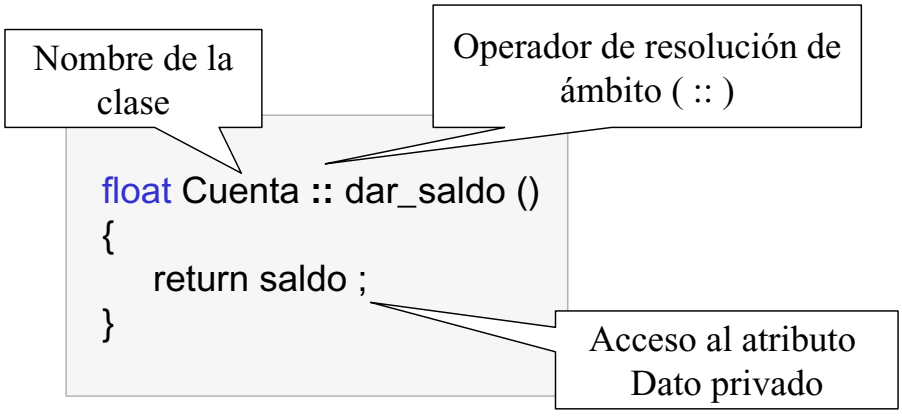
Esta es la opción más correcta. La implementación del método ingresar sería el siguiente:



Se pone el nombre de la clase delante del nombre del método, porque podemos tener varias clases con el mismo método.

Implementación de los métodos fuera de la clase

La implementación del método dar_saldo sería el siguiente:



En el código de los métodos se puede usar todo lo que está declarado en la clase: lo público y lo privado.

Creación de elementos de una clase: creación de objetos

La clase ya está terminada (declarada e implementada). Ya podemos crear objetos y trabajar con ellos.

La sintaxis es igual que para declarar variables de un cierto tipo.

```
class Circulo
{
 private :
   float coor_x;
   float coor_y;
   float radio;
 public :
   void inicializar ( );
   float  visualizar_radio ( );
   float calcular_area (  ) ;
} ;
```

<tipo_de_dato> <nombre_de_variable> ;
<Nombre de la clase> <nombre_del_objeto> ;

```
Circulo c1, c2, c3;
```

Objetos de la clase Círculo

Uso y manipulación de objetos: paso de mensajes

Una vez creados los objetos, podemos trabajar con ellos. Si queremos que un determinado objeto ejecute un método, utilizamos el operador punto (•) . A esto se le conoce como **paso de mensajes.**

Los objetos se manipulan mediante el paso de mensajes .

- Cada paso de mensaje provoca la ejecución del correspondiente método definido en la clase del objeto receptor.
- Cada objeto entiende tantos mensajes como métodos estén definidos en su clase.

```
void Circulo:: inicializar ( float a, float b, float  r)
    {  coor_x = a;
       coor_y = b;
       radio = r;
    }
```

Se pasa el mensaje inicializar al objeto c1

```
Circulo c1, c2, c3;
c1.inicializar (1, 2, 4);
c2.inicializar (0,0,5);
```

Los objetos c1, c2 tienen valores diferentes, tienen distinto **estado**.

Público y privado

- Los atributos deben ser privados.
- Los métodos o funciones miembro deben ser públicas.
- A través del objeto de una clase sólo se puede acceder a lo que está declarado como público en la clase (con el operador •).
- Lo privado solo puede ser accedido desde el código de los métodos.

```
class  Circulo
{
 private :
   float coor_x;
   float coor_y;
   float radio;
public :
   ....
} ;
```

```
Circulo c1, c2, c3;
c1.coor_x = 4;
c1.radio = 7 ;
```

Error

Los atributos coor_x y radio son privados

Resumen:

- Las instancias de un tipo cualquiera se denominan **variables**, mientras que las instancias de una clase se denominan **objetos**.
- Los atributos describen el estado de un objeto. Un atributo consta de dos partes: nombre del atributo y valor.
- Los métodos describen el comportamiento de los objetos de una clase. Representan las operaciones que se pueden realizar con los objetos de la clase. La ejecución de un método puede conducir a cambiar el estado del objeto.
- Los objetos se manipulan mediante el paso de mensajes.
- Para ejecutar un método asociado a un objeto, se realiza una acción que se conoce como "envío de mensajes".
- Las operaciones y valores visibles desde el exterior de la clase, es lo que se denomina "Interfaz" de la clase.