



## Constructores y Destrucciones

---

- Ya se ha apuntado que C++ no permite crear objetos sin dar un valor inicial apropiado a todas sus variables miembro. Esto se hace por medio de unas funciones llamadas **constructores**, que se llaman automáticamente **siempre que se crea un objeto** de una clase.
- El **nombre del constructor** es siempre el **nombre de la clase**. Los constructores se caracterizan porque *se declaran y definen sin valor de retorno*, ni siquiera **void**. C++ utiliza las capacidades de sobrecarga de funciones de para que una clase tenga **varios constructores**.



# Constructores y Destructores

---

Ejemplo:

```
class Cuenta {
    // Variables miembro
private:
    double Saldo; // Saldo Actual de la
    cuenta
    double Interes; // Interés aplicado
public:
    // Constructor
Cuenta(double unSaldo, double unInteres);
    // Acciones básicas
    double GetSaldo()
    { return Saldo; }
    double GetInteres()
    { return Interes; }
    void SetSaldo(double unSaldo)
    { Saldo = unSaldo; }
    void SetInteres(double unInteres)
    { Interes = unInteres; }
    void Ingreso(double unaCantidad)
    { SetSaldo( GetSaldo() + unaCantidad ); }
};
```



## Constructores y Destructores

---

- La definición del **constructor** de la clase **Cuenta** pudiera ser:

```
Cuenta::Cuenta(double unSaldo, double
    unInteres)
{ //Hace llamadas los otros métodos
    SetSaldo(unSaldo);
    SetInteres(unInteres);
}
```

- Como el **constructor** es una **función miembro**, tiene **acceso directo a las variables miembro privadas**. Luego el **constructor** también podría definirse del siguiente modo:

```
Cuenta:: Cuenta(double unSaldo, double
    unInteres)
{ //Asigna a los datos el valor de los
    parametros
    Saldo = unSaldo;
    Interes = unInteres;
}
```



## Constructores y Destructores

---

- La llamada al **constructor** se puede hacer explícitamente en la forma:  
Cuenta c1 = Cuenta(500,10);  
o bien, de una forma implícita, más abreviada, permitida por C++:  
Cuenta c1(500, 10);
- Se llama **constructor por defecto** a un constructor que no necesita que se le pasen parámetros o argumentos para inicializar las variables miembro de la clase.
- El **constructor por defecto es necesario** si se quiere hacer una declaración en la forma:  
Cuenta c1;  
y también cuando se quiere crear un **vector de objetos**, por ejemplo:  
Cuenta cuentas[100];



## Constructores y Destructores

---

- Se llama **constructor de oficio** al **constructor por defecto** que define automáticamente el compilador si el usuario no define ningún constructor. Si bien todo **constructor de oficio** es **constructor por defecto** (ya que no tiene argumentos), lo contrario no es cierto: el programador puede definir **constructores por defecto** que no son **de oficio**.
- Importante: el compilador sólo crea un **constructor de oficio** en el caso de que el programador no haya definido **ningún constructor**.
- En caso de que sólo se haya definido un **constructor con argumentos** y se necesite un **constructor por defecto** para crear, por ejemplo un **vector de objetos**, el compilador no crea este **constructor por defecto** sino que da un mensaje de error.

## Constructores y Destructores

---

- Un caso particular se produce cuando se **crea un objeto** inicializándolo a partir de **otro objeto de la misma clase**. Por ejemplo, C++ permite crear tres objetos **c1**, **c2** y **c3** de la siguiente forma:

```
Cuenta c1(1000,8.5);
```

```
Cuenta c2 = c1;
```

```
Cuenta c3(c1) (ó Cuenta c3=Cuenta(c1));
```

- En la primera sentencia se crea un objeto **c1** con un saldo de 1000 y un interés del 8.5%. En la segunda se crea un objeto **c2** a cuyos atributos se les asignan los mismos valores que tienen en **c1**. La tercera sentencia es equivalente a la segunda: **c3** se inicializa con los valores de **c1**.
- En las sentencias anteriores se han creado tres objetos y por definición se ha tenido que llamar tres veces a un constructor: en la primera sentencia se ha llamado al **constructor con argumentos** definido en la clase, pero en la segunda y en la tercera se ha llamado a un constructor especial llamado **constructor de copia (copy constructor)**.
- Por definición, el **constructor de copia** tiene **un único argumento** que es una **referencia constante a un objeto de la clase**. Su declaración sería pues como sigue:

## Constructores y Destructores

---

```
Cuenta::Cuenta(const Cuenta& c2){  
    Saldo    = c2.Saldo;  
    Interes  = c2.Interes;}
```

- o Las sentencias anteriores de declaración de los objetos **c2** y **c3** funcionarían correctamente aunque no se haya declarado y definido en la clase **Cuenta** ningún constructor de copia. Esto es así porque el compilador de C++ proporciona también un **constructor de copia de oficio**, cuando el programador no lo define. El **constructor de copia de oficio** se limita a realizar una **copia bit a bit** de las variables miembro del objeto original al objeto copia. En este caso, eso es perfectamente correcto y es todo lo que se necesita. Otras veces, esta copia bit a bit no da los resultados esperados. Entonces el programador debe preparar su propio **constructor de copia** e incluirlo en la clase como un constructor sobrecargado más. Dos casos muy importantes en los que se utiliza el constructor de copia:
  1. Cuando a una función se le pasan objetos como **argumentos por valor**, y
  2. Cuando una función tiene un **objeto como valor de retorno**.