

### Contenidos

1. Miembros que no se heredan automáticamente.
2. Constructor de la clase derivada.
3. Destructor de la clase derivada.
4. Operador de asignación de la clase derivada.
5. Modos de derivación.

### Miembros que no se heredan automáticamente

La clase derivada hereda todos los atributos y todos los métodos, **excepto:**

- Constructores
- Destructor
- Operador de asignación

**Constructores y destructor:** Si en la subclase no están definidos, se crea un constructor por defecto y un destructor, aunque sin que hagan nada en concreto.

**Operador de asignación:** Si en la subclase no está definido, se crea uno por defecto que se basa en el operador de asignación de la superclase.

Se deben crear los constructores y el destructor en la subclase.  
Se debe definir el operador de asignación en la subclase.

## Herencia (II)

### Constructor de la clase derivada: un ejemplo de constructores

```
class ClaseBase
{
protected:
    int b1 ;
    int b2 ;
public:
    int b3 ;
    ClaseBase (int a=0, int b=0, int c=0 );
    ...
};
```

```
ClaseBase :: ClaseBase (int a, int b, int c )
{
    b1= a;
    b2 = b;
    b3 = c;
};
```

```
class ClaseDerivada : public ClaseBase
{
private:
    float s1 ;
    char s2 ;
public:
    ClaseDerivada ( float d, char e );
};
```

```
ClaseDerivada :: ClaseDerivada ( float d, char e )
{
    s1 = d;
    s2 = e;
};
```

```
ClaseDerivada d( 8.2, 'f' );
```

3

## Herencia (II)

### Constructor de la clase derivada

```
class ClaseBase
{
protected:
    int b1 ;
    int b2 ;
public:
    int b3 ;
    ClaseBase ( int a=0, int b=0, int c=0 );
    ...
};
```

Cuando se llama al constructor de la clase derivada, el compilador ejecuta el constructor por defecto de la clase Base, a no ser que especifiquemos uno en concreto.

```
ClaseDerivada d( 8.2, 'f' );
```

```
class ClaseDerivada : public ClaseBase
{
private:
    float s1 ;
    char s2 ;
public:
    ClaseDerivada ( float d, char e );
};
```

d	b1	0	s1	8.2
	b2	0	s2	f
	b3	0		

4

## Herencia (II)

### Constructor de la clase derivada

```
class ClaseDerivada : public ClaseBase
{
private:
    float s1 ;
    char s2 ;
public:
    ClaseDerivada (int a, int b, int c,float d, char e );
};
```

Otra opción, si queremos que los atributos heredados tengan un valor concreto, es llamar al constructor de Clase Base con dichos valores

```
ClaseDerivada :: ClaseDerivada (int a, int b, int c, float d, char e ) : ClaseBase(a,b,c)
{
    s1 = d;
    s2 = e;
};
```

Paso de argumentos al constructor de la superclase

```
ClaseDerivada d( 6,7,4,8.2, 'f' );
```

b1	6	s1	8.2
b2	7	s2	f
b3	8		

d

5

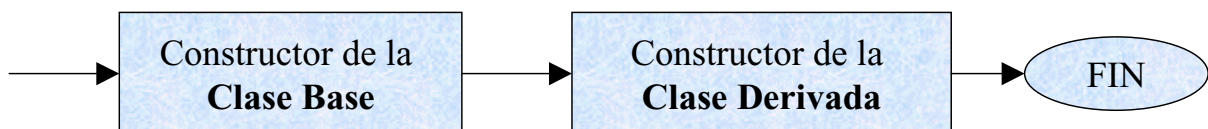
## Herencia (II)

### Constructor

Cuando se crea un objeto de la clase derivada, ocurre lo siguiente:

1º Se invoca el constructor de la superclase. La invocación del constructor de la superclase se realiza con los argumentos que se especifiquen. Si no hay argumentos, se usa el constructor predeterminado.

2º Se ejecuta el constructor de la clase derivada.

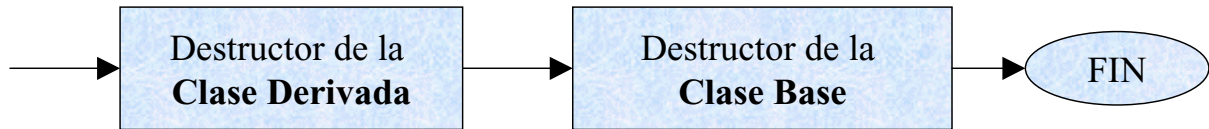


6

## Destructor

Cuando se destruye un objeto de la clase derivada, ocurre lo siguiente:

- 1º Se invoca al destructor de la clase derivada,
- 2º Se invoca el destructor de la superclase.



## Operador de asignación

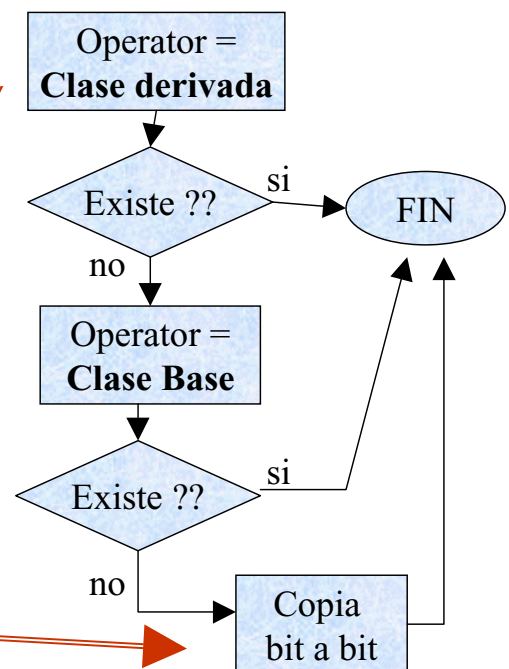
El operador de asignación se comporta de la siguiente manera:

Se invoca el operador de asignación de la clase **derivada**.

Se invoca al operador de asignación de la **superclase**, que afecta a los atributos heredados.

El resto de atributos se copian **bit a bit**.

Si no existe ninguno de los dos, la copia se realiza **bit a bit**.



## Operador de asignación

```
ClaseBase& ClaseBase :: operator = ( ClaseBase c )  
{  
    b1 = c.b1;  
    b2 = c.b2;  
    b3 = c.b3;  
    return *this;  
};
```

```
ClaseDerivada& ClaseDerivada :: operator = ( ClaseDerivada d )  
{  
    s1 = d.s1;  
    s2 = d.s2;  
    return *this;  
};
```

```
ClaseDerivada ob1 ( 6,7,4,8.2, 'f' );  
ClaseDerivada ob2 ( 0,0,0, 0, 'h' );  
  
ob2 = ob1 ;
```

¿Qué método se ejecuta?

**Incorrecto !!**

ob1	b1	6	s1	8.2
	b2	7	s2	f
	b3	8		
ob2	b1	0	s1	8.2
	b2	0	s2	f
	b3	0		

## Operador de asignación

Para arreglar el problema, y que la copia se realice correctamente, podemos modificar el operador de asignación de la clase derivada:

```
ClaseDerivada& ClaseDerivada :: operator = ( ClaseDerivada d )  
{  
    b1 = d.b1;  
    b2 = d.b2;  
    b3 = d.b3;  
    s1 = d.s1;  
    s2 = d.s2;  
    return *this;  
};
```

¿Y si tenemos 40 atributos?  
¿Y si añadimos más atributos en la clase base?

Se puede hacer que la copia de los atributos heredados lo realice la superclase: para ello tenemos que invocar al operador de asignación de la superclase desde la clase derivada.

### Operador de asignación

Las dos implementaciones siguientes tienen el mismo efecto.

```
ClaseDerivada& ClaseDerivada :: operator = ( ClaseDerivada d )  
{  
    ClaseBase :: operator = ( d );  
    s1 = d.s1;  
    s2 = d.s2;  
    return *this;  
};
```

Se ejecuta el operador de asignación de la superclase

```
ClaseDerivada& ClaseDerivada :: operator = ( ClaseDerivada d )  
{  
    b1 = d.b1;  
    b2 = d.b2;  
    b3 = d.b3;  
    s1 = d.s1;  
    s2 = d.s2;  
    return *this;  
};
```

ob1	b1	6	s1	8.2
	b2	7	s2	f
	b3	8		
ob2	b1	6	s1	8.2
	b2	7	s2	f
	b3	8		

### Modos de derivación

El modo de derivación o modo de acceso determina cómo será el acceso a los miembros que se heredan desde la subclase.

Es decir, tenemos un mecanismo para restringir el acceso a los atributos y a los métodos que se heredan en la subclase.

Para indicar el modo de derivación tenemos la siguiente sintaxis:

```
class Subclase : <modo_derivacion> Superclase  
{  
    // lista de miembros públicos y privados  
};
```

## Herencia (II)

### Modos de derivación

Disponemos de varios modos:

**Derivación Pública:** en la subclase, cada miembro heredados mantiene el modo de acceso establecido en la superclase.

```
class Subclase : public Superclase  
{ ... };
```

**Derivación Privada:** todos los miembros heredados se convierten en privados en la subclase.

```
class Subclase : private Superclase  
{ ... };
```

Por defecto, si no se indica nada, se aplica el modo de derivación privado.

## Herencia (II)

### Modo de derivación Público

```
class Subclase : public Superclase  
{ ... };
```

Los miembros heredados mantienen su modo de acceso:

- ➔ Los que son públicos en la superclase, siguen siendo públicos en la subclase y por tanto accesibles.
- ➔ Los que son privados en la superclase, siguen siendo privados en la subclase e inaccesibles.

En las funciones miembro de la subclase no se tiene acceso a los miembros privados heredados.

## Herencia (II)

### Modo de derivación Privado

```
class Subclase : private Superclase  
{ ... };
```

Todos los miembros heredados se convierten en privados:

- Los que son públicos en la superclase, pasan a ser privados en la subclase, pero son accesibles en la subclase.
- Los que son privados en la superclase, siguen siendo privados en la subclase e inaccesibles.

En las funciones miembro de la subclase no se tiene acceso a los miembros privados heredados, pero sí a los miembros públicos heredados, aunque se hayan convertido en privados.

## Herencia (II)

### Modo de derivación Privado

```
class Subclase : private Superclase  
{ ... };
```

#### Importante:

A los objetos de la subclase, NO se les puede enviar mensajes que correspondan a métodos públicos heredados (se han convertido en privados).

## Herencia (II)

### Ejemplo: modo de derivación público

```
class Persona
{
  private:
    // miembros privados
  public:
    // miembros públicos
};
```

```
class Alumno : public Persona
{
  private:
    ...
  public:
    ....
};
```

- La clase Alumno incorpora los miembros privados de Persona. Los miembros privados heredados siguen siendo privados.
- La clase alumnos incorpora los miembros públicos de Persona. Los miembros públicos heredados siguen siendo públicos.
- Las funciones miembro que se definan en la clase alumno, no podrán acceder a los miembros privados heredados.

## Herencia (II)

### Ejemplo: modo de derivación privado

```
class Persona
{
  private:
    // miembros privados
  public:
    // miembros públicos
};
```

```
class Alumno : private Persona
{
  private:
    ...
  public:
    ....
};
```

- Todos los miembros privados y públicos heredados pasan a ser privados:
  - ⇒ Los miembros privados heredados: privados y No accesibles
  - ⇒ Los miembros públicos heredados: privados pero accesibles.

## Herencia (II)

### Modo de derivación público: acceso a lo heredado

```
class Persona
{
  private:
    char * nif;
    int edad;
    char * nombre, *apellidos;
  public:
    Persona(char *, int = 0, char *, char * );
    ...
    char * mnombre() ;
    void mostrar() ;
    ...
    void leer();
};
```

Privado en **Alumno**.  
Inaccesible desde sus funciones miembro.  
Inaccesible desde fuera de los objetos.

Público en **Alumno**.  
Accesible en sus funciones miembro.  
Accesible desde fuera de los objetos.

```
class Alumno : public Persona
{ ... };
```

## Herencia (II)

### Modo de derivación privado: acceso a lo heredado

```
class Persona
{
  private:
    char * nif;
    int edad;
    char * nombre, *apellidos;
  public:
    Persona(char *, int = 0, char *, char * );
    ...
    char * mnombre() ;
    void mostrar() ;
    ...
    void leer();
};
```

Privado en **Alumno**.  
Inaccesible desde sus funciones miembro.  
Inaccesible desde fuera de los objetos.

Privado en **Alumno**.  
Accesible en sus funciones miembro.  
Inaccesible desde fuera de los objetos.

```
class Alumno : private Persona
{ ... };
```

### Modo de derivación público: acceso a lo heredado

```
class Persona
{
  protected:
  char * nif;
  int edad;
  char * nombre, *apellidos;
  public:
  Persona(char *, int = 0, char *, char * );
  ...
  char * mnombre() ;
  void mostrar() ;
  ...
  void leer();
};
```

Protegido en **Alumno**.  
Accesible desde sus funciones miembro.  
Inaccesible desde fuera de los objetos.

**Público** en **Alumno**.  
Accesible en sus funciones miembro.  
Accesible desde fuera de los objetos.

```
class Alumno : public Persona
{ ... };
```

### Modo de derivación privado: acceso a lo heredado

```
class Persona
{
  protected:
  char * nif;
  int edad;
  char * nombre, *apellidos;
  public:
  Persona(char *, int = 0, char *, char * );
  ...
  char * mnombre() ;
  void mostrar() ;
  ...
  void leer();
};
```

Protegido en **Alumno**.  
Accesible desde sus funciones miembro.  
Inaccesible desde fuera de los objetos.

**Privado** en **Alumno**.  
Accesible en sus funciones miembro.  
Inaccesible desde fuera de los objetos.

```
class Alumno : private Persona
{ ... };
```