

Herramientas de C++: Herencia y composición

Informática II - R2004
2017

Reutilización del código

“Una de las características más importantes de C++ es la reutilización de código.
Pero para ser revolucionario, necesita ser capaz de hacer algo más que copiar
código y modificarlo” [BruceEckel]



Composición - objetos como miembros de clases

De la misma manera que las clases tienen miembros de diferentes tipos, pueden tener también objetos como miembros. Por ejemplo, en la siguiente clase:

```
class sueldo {  
    private:  
        float cantidad;  
        char moneda;  
  
    public:  
        sueldo();  
        sueldo(int, char);  
  
        ...  
}
```

```
class empleado {  
    private:  
        std::string nombre;  
        sueldo salario;  
  
    public:  
        empleado();  
        empleado (std::string , sueldo);  
}
```


Se dice que el objeto empleado TIENE un nombre (un objeto de tipo string) y un salario (un objeto de tipo sueldo)

Constructores en las clases compuestas

Al instanciar el objeto compuesto, se invoca el constructor por defecto de los objetos miembro A MENOS que en el constructor de la clase compuesta se indique otra cosa mediante una LISTA INICIALIZADORA:

```
empleado::empleado( std::string nom , sueldo sal ) : salario(sal)
{
    nombre = nom;
}
```

En este ejemplo se llamaría al constructor POR DEFECTO del objeto string nombre, y al constructor parametrizado del objeto sueldo salario.



Herencia

Otra forma de reutilizar funcionalidades de otras clases para crear nuevas es la HERENCIA. A diferencia de la composición, en la que un objeto TIENE otros objetos, en la herencia vamos a decir que el objeto ES otro objeto, con más funcionalidades:

Por ejemplo:

- Un Empleado ES una Persona (tiene nombre, apellido, DNI, etc.) que además tiene un salario, una tarea, un empleo y un horario.
- Un string ES un vector que contiene solo caracteres ascii y un caracter especial para indicar el final de la cadena.

En estos casos la clase derivada HEREDA de una clase base ciertas funcionalidades y agrega otras

Herencia - sintaxis

```
class clase_derivada : [modo_derivación] clase_base
```

```
{
```



public, private o PROTECTED:

```
...
```

Define la VISIBILIDAD de los miembros de la clase
base en la clase derivada

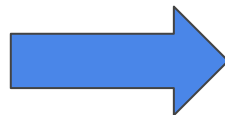
```
}
```



Sintaxis

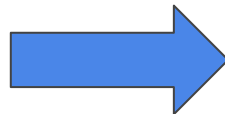
De esta manera, un empleado que ES una persona podría declararse como:

```
class Persona {  
    private:  
        std:string nombre;  
        long DNI;  
    public:  
        Persona();  
    ...  
}
```



Clase BASE

```
class empleado : public Persona {  
    private:  
        sueldo salario;  
    public:  
        empleado();  
        empleado (std:string , long , sueldo);  
    ...  
}
```




Clase DERIVADA



Modos de Derivación

```
class clase_derivada : [modo_derivación] clase_base
```

Especificador de acceso a miembros de la clase base	Tipo de herencia	
	herencia public	
public	public en la clase derivada. Puede ser utilizado directamente por las funciones miembro, las funciones friend y las funciones no miembro.	 No lo usamos en info 2
protected	protected en la clase derivada. Puede ser utilizado directamente por las funciones miembro y las funciones friend .	
private	Oculto en la clase derivada. Puede ser utilizado por las funciones miembro y las funciones friend a través de las funciones miembro public o protected de la clase base.	

¿Que se hereda?

Las CD heredan **TODOS los atributos y métodos** **excepto**:

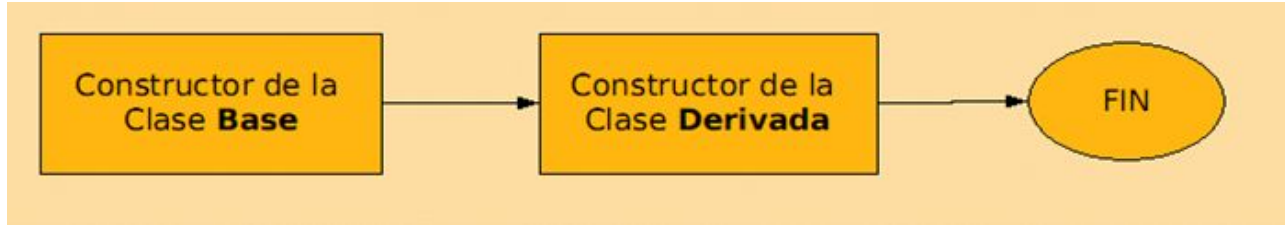
- Constructores y Destructor
- Operador de asignación.

Constructores y destructor: Si en la **CD** no están definidos, se crean, como siempre, unos por defecto que no hacen nada. Por ello **DEBEN DEFINIRSE**

Operador de asignación: Si en la **CD** no está definido, se crea uno por defecto que se basa, si existe, en el operador de asig. de la **CB**. Por ello **DEBE DEFINIRSE**



Orden de ejecución de constructores y destructores



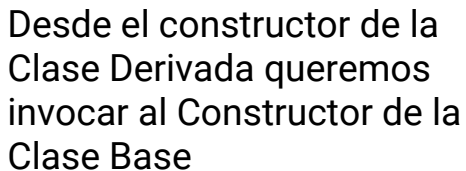
1. Se invoca al constructor de la CB con los argumentos que se especifiquen. Sino, se usa el constructor predeterminado.
2. Se ejecuta el constructor de la CD



1. Se invoca al destructor de la CD
2. Se invoca al destructor de la CB

Constructores CB y CD

Desde el constructor de la Clase Derivada queremos invocar al Constructor de la Clase Base



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

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

Constructores CB y CD

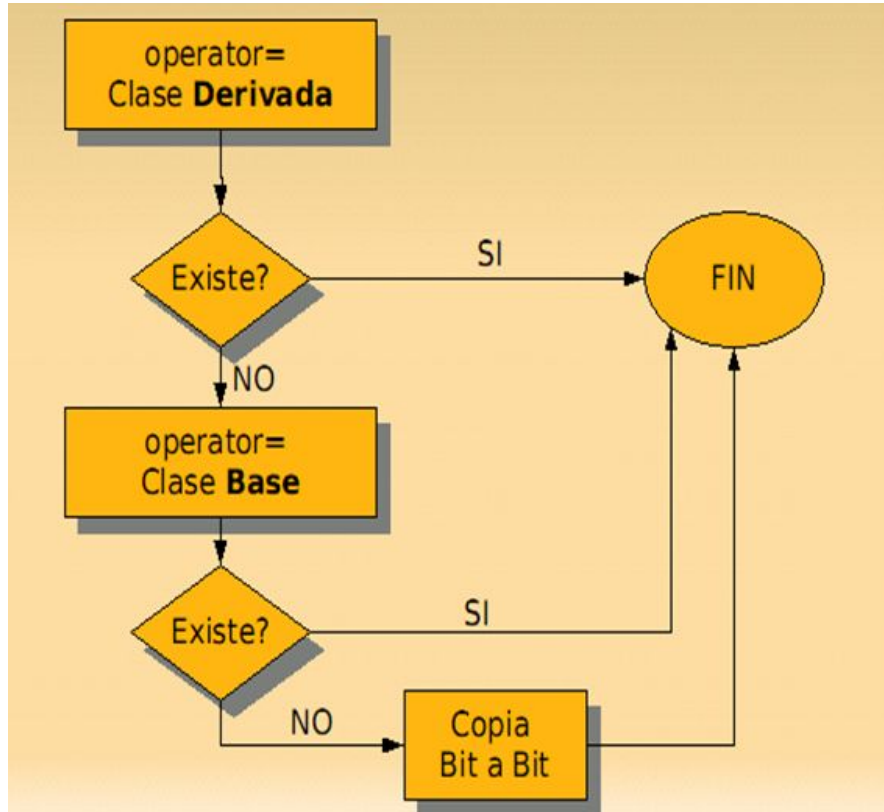
Correcta inicialización de objetos en la composición y la herencia:

Listas inicializadoras

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



Operador de asignación



Operador de asignación

```
class ClaseBase{
private:
    int b1 ;
    int b2 ;
public:
    int b3 ;
    ClaseBase(int=0,int=0,int=0);
    .....
    //métodos para acceder a b1 y b2.....
};

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

```
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 obj1 (6,7,4,8.2,'f');
```

```
ClaseDerivada obj2 (0,0,0,0,'h');
```

```
obj2 = obj1;
```

Operador de asignación Clase Derivada

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

Desde el operador de Asignación de la CD se invoca al operador de asignación de la CB



Herencia de operadores sobrecargados

```
class Persona
{
    protected:
        string nombre;
        long dni;

    public:
        Persona ();
        Persona (string, long);
        Persona (Persona &);
        Persona& operator=(Persona&);

    friend ostream& operator<< ( ostream& , Persona&);
};
```

```
class Estudiante : public Persona{
    private:
        long legajo;
        int notas[3];

    public:
        Estudiante ();
        Estudiante(long, int*)
        Estudiante (string, long , long , int*);

    friend ostream & operator<<( ostream&, Estudiante&);
};
```



Herencia de Operadores Sobrecargados

```
ostream & operator<< ( ostream & out , Estudiante &a)
{
    out << (Persona&) a; //convoco al operador de la CB casteando
    out << a.legajo << " - " << a.notas[0] << " - " << a.notas[1];
    out << endl;
    return out;
}
```

