

Decimoséptima Sesión

Metodologías y Técnicas de Programación II

Programación Orientada a Objeto (POO) C++

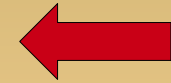
La Herencia

Estado del Programa

Introducción a la POO

Historia de la Programación
Conceptos de POO

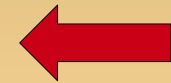
C++
Mi primera Clase



Repaso de Conceptos

Estándares de Programación
Punteros y Memoria

E/S
Control y Operadores



Clases y Objetos en C++

Uso y aplicación
Constructores
Constantes e "inline"

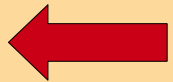
Funciones Amigas
Sobrecarga de Funciones



Sobrecarga

De Operadores

Creación Dinámica de Objetos



Herencia.

Tipos de Visibilidad

Herencia Múltiple



Polimorfismo

Funciones Virtuales

Polimorfismo y Sobrecarga.

Plantillas

Contenedores

Iteradores

17.1 Conceptos Vistos

Conceptos Previos

El concepto fundamental en la reutilización.

Todo lo que hemos visto va encaminado a poder programar de forma más eficiente:

- Mejoramos nuestra capacidad para analizar un problema.
- Podemos basarnos en código hecho (y probado).
- Reutilizo código o ideas anteriores.

Encapsulamiento:

- Oculto complejidad que no interesa como cliente de una clase.
- Impido el acceso directo a los mecanismos internos.

Las clases vienen ya con mecanismos para inicializar o finalizar.

Puedo sobrecargar constructores o funciones o métodos.

Puedo sobrecargar hasta los operadores.

Un código más legible también es más reutilizable.

17.2 Herencia

Herencia y Composición

Necesitamos algo más que copiar código y reutilizarlo para que la programación orientada a objeto sea algo Revolucionario.

Veremos cómo reutilizar código creando nuevas clases, pero en lugar de hacerlo desde cero nos basaremos en clases que alguien ha implementado y probado con anterioridad.

Composición: Creamos objetos de la clase existente dentro de la nueva clase que estamos creando.

La nueva clase **está compuesta** por objetos de clases existentes.

Herencia: Se toma la forma de la clase existente y se añade código sin modificar la clase existente.

La nueva clase **se crea como un tipo** de la existente.

Casi todo el trabajo lo realiza el compilador.

La herencia es uno de los pilares fundamentales de la POO.

17.2 Herencia

Sintaxis de la Composición

El concepto de composición ya lo hemos estado usando. Nuestras clases contenían atributos, que era de los tipos predefinidos.

```
class Coche
{
private:
    Rueda      a_[4];
    Motor      motor_;
    Direccion  dir_;
public:
    Rueda(): motor_(0), dir_(0)
    {};
    arrancar()
    {
        motor_.encender();
        dir_.centrar();
    }
}
```

```
int main()
{
    Coche      c1;

    c1.arrancar();    // Bien.

    c1.a_[0].inflar(); // Mal. Privado
}
```

17.2 Herencia

Herencia

La herencia es una propiedad de la Programación Orientada a Objeto.

Nos permite:

- Crear nuevas clases a partir de clases existentes.
- Conservando las propiedades de la clase original.
- Añadir nuevos métodos y atributos a la clase original.

La nueva clase obtenida se conoce como **clase derivada**, y las clases a partir de las cuales se deriva, **clases base**. Una clase derivada puede a su vez servir de base para otra clase diferente.

Cuando una clase deriva de más de una clase base: Herencia Múltiple.

Este tipo de relaciones nos permite crear Jerarquías de Clases.

Las jerarquías de clases se usan especialmente en la resolución de problemas complejos, es difícil que tengamos que recurrir a ellas para resolver problemas sencillos.

17.2 Herencia

¿Cuándo utilizar la herencia?

1. Cuando hay suficientes similitudes.

Todas las características de la clase existente o la mayoría de ellas, son adecuadas para la nueva clase.

2. En la nueva clase se ampliará y/o redefinirá el conjunto de características.

La nueva clase definida a partir de la clase existente, adopta todos los miembros de la clase existente:

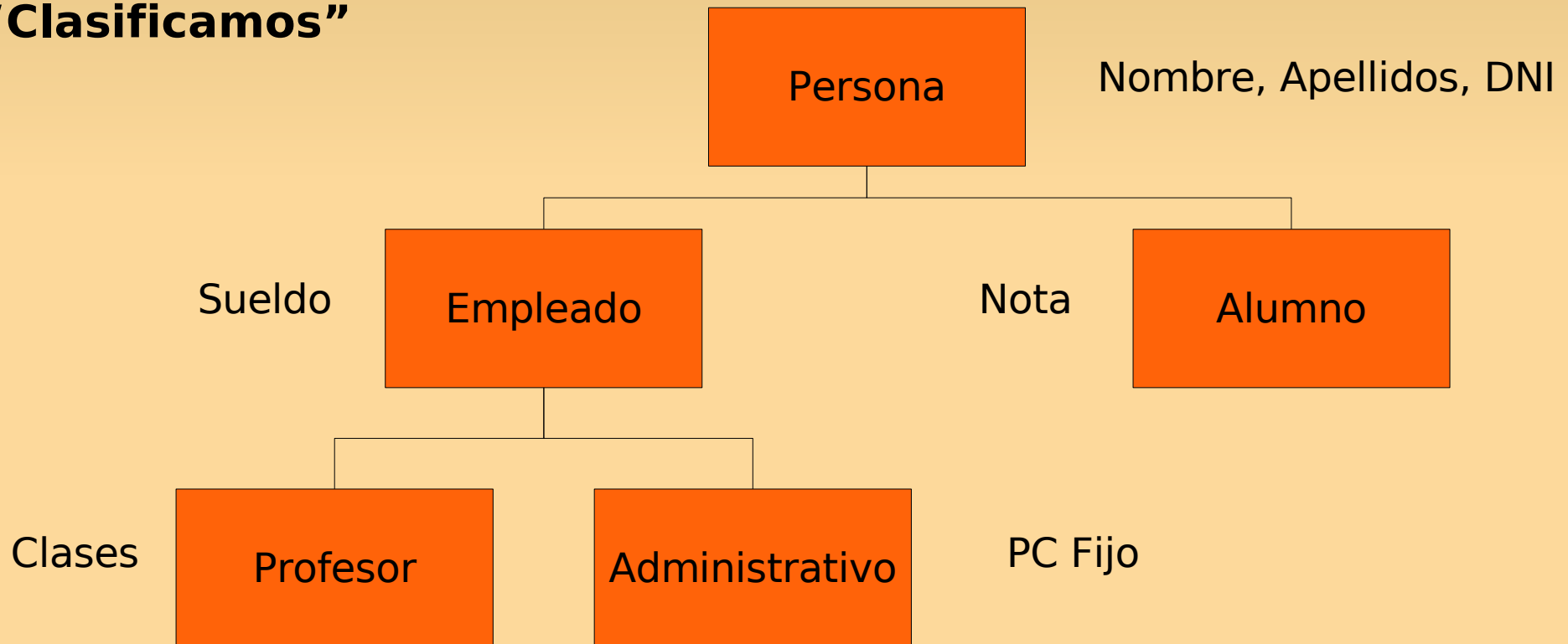
- atributos
- métodos

17.2 Herencia

Jerarquías de Clases

Las clases bases son las más genéricas. Vamos especializando según derivamos.

“Clasificamos”



17.2 Herencia

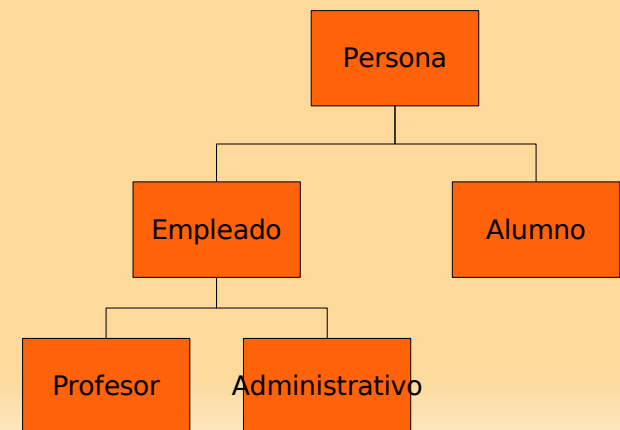
Jerarquías de Clases

Cada vez que creamos un objeto de la clase Administrativo estamos creando además un objeto Empleado y un objeto Persona.

Siempre podemos extender nuestra jerarquía. Si tenemos una clase Becario que no es ni un Estudiante ni un Empleado podemos derivarlo directamente de persona.

Si consideramos que es un Estudiante y además un Empleado podemos derivarlo de los dos: **Herencia Múltiple**.

Podemos **aplicar procedimientos genéricos** a una clase concreta. Subida de sueldo general a todos los empleados independientemente de su puesto principal.



17.2 Herencia

Sintaxis de la Herencia en C++

```
class <clase_derivada> :  
    [public|private] <base1> [, [public|private] <base2> ] {};
```

```
class AlumnoM2 : public Alumno  
{  
    ....  
    ....  
}
```

```
class Clase_derivada : public ó private Clase_Base
```

17.2 Herencia

Ejemplo de Herencia

```
class Persona
{
    private:
        char * nif;
        int edad;
        char * nombre, *apellidos;
    public:
        Persona(char*,int=0,char*,char*);
        Persona & operator=(Persona &);
        ~Persona();
        void medad(int);
        void mnombre(char *);
        char * mnombre() ;
        void mostrar() ;
        char * nombreCompleto() ;
        void felizCumple();
        void leer();
};
```

Si queremos un Alumno, sabiendo que contamos con la clase Persona.

Los alumnos tienen NIF, nombre, apellidos y edad.

Los **atributos** de Persona nos valen para Alumno.

Los **métodos** de Persona nos valen para Alumno.

17.2 Herencia

Ejemplo de Herencia

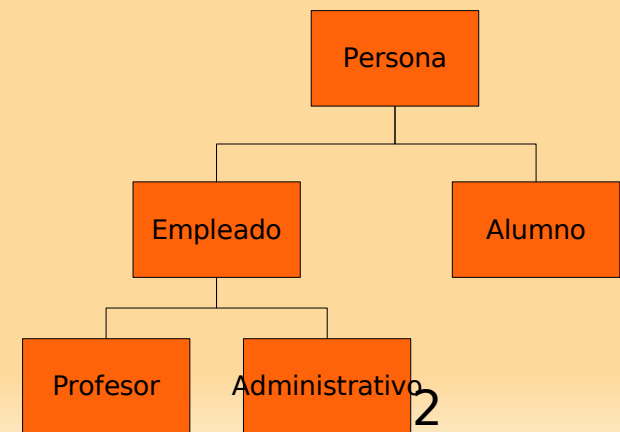
Resulta adecuado aplicar el mecanismo de herencia y crear una nueva clase Alumno a partir de la clase ya existente Persona.

Los alumnos son personas

Clase base: La clase base es la clase ya creada, de la que se hereda. También se la denomina clase madre o superclase.

Clase derivada: es la clase que se crea a partir de la clase base. Se dice que es la clase que hereda. También se la denomina clase hija o subclase.

La clase Persona es la clase base
La clase Alumno es la clase derivada.

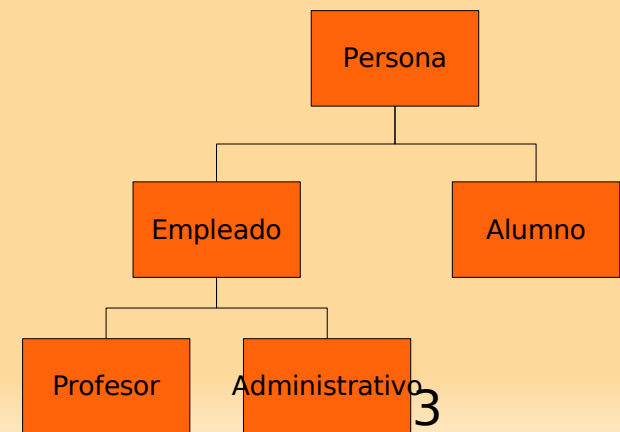
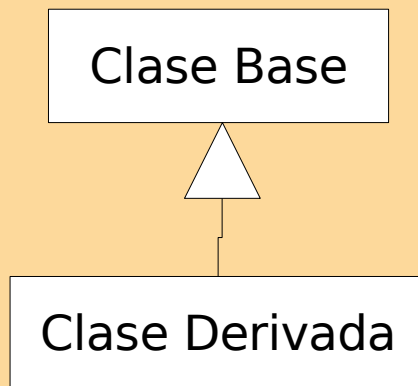


17.2 Herencia

Clase Base y Clase Derivada

- La clase derivada hereda todas las características de la clase base.
- La clase derivada puede definir características adicionales.
- La clase derivada puede redefinir características heredadas de la clase base.
- La clase derivada puede anular características heredadas de la clase base.

El proceso de herencia no afecta a la clase base!!



17.2 Herencia

Clase Base y Clase Derivada

La clase derivada:

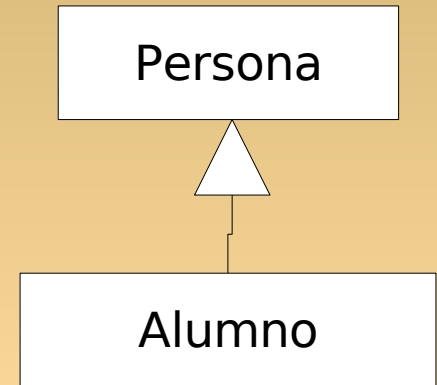
ii No puede acceder a los miembros privados de la clase Base !!

Las funciones miembros o métodos de Alumno no se tiene acceso a los atributos privados de Persona.

No podemos hacer esto:

```
Alumno::pasar_curso()  
{  
    .....  
    edad_ = edad + 1;  
    .....  
}
```

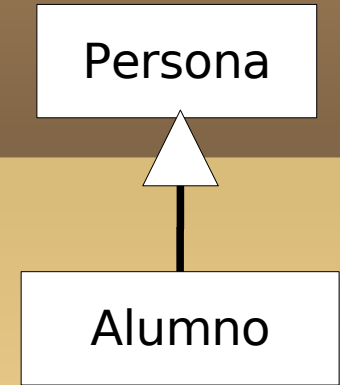
Violaría las normas de encapsulamiento.



17.2 Herencia

Control de Acceso en la Herencia

```
class <clase_derivada> :  
[public|private] <base1> [, [public|private] <base2>] {};
```



Para cada clase base podemos definir dos tipos de acceso, public o private. Si no se especifica ninguno de los dos, por defecto se asume que es private.

public: los miembros heredados de la clase base conservan el tipo de acceso con que que fueron declarados en ella.

private: todos los miembros heredados de la clase base pasan a ser miembros que privados en la clase derivada.

Dentro de la clase podemos declarar nuestros atributos y métodos como:

public: Se puede acceder a ellos desde fuera de la clase.

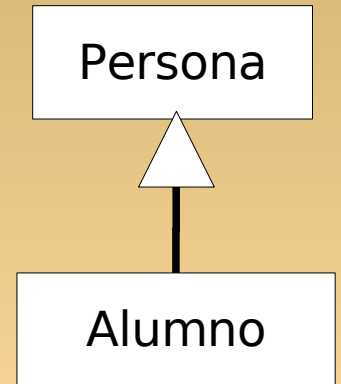
private: No se puede acceder a ellos desde fuera.

protected: No se puede acceder a ellos a no ser que sea una clase derivada.

17.2 Herencia

Control de Acceso en la Herencia

```
class <clase_derivada> :  
[public|private] <base1> [, [public|private] <base2>] {};
```



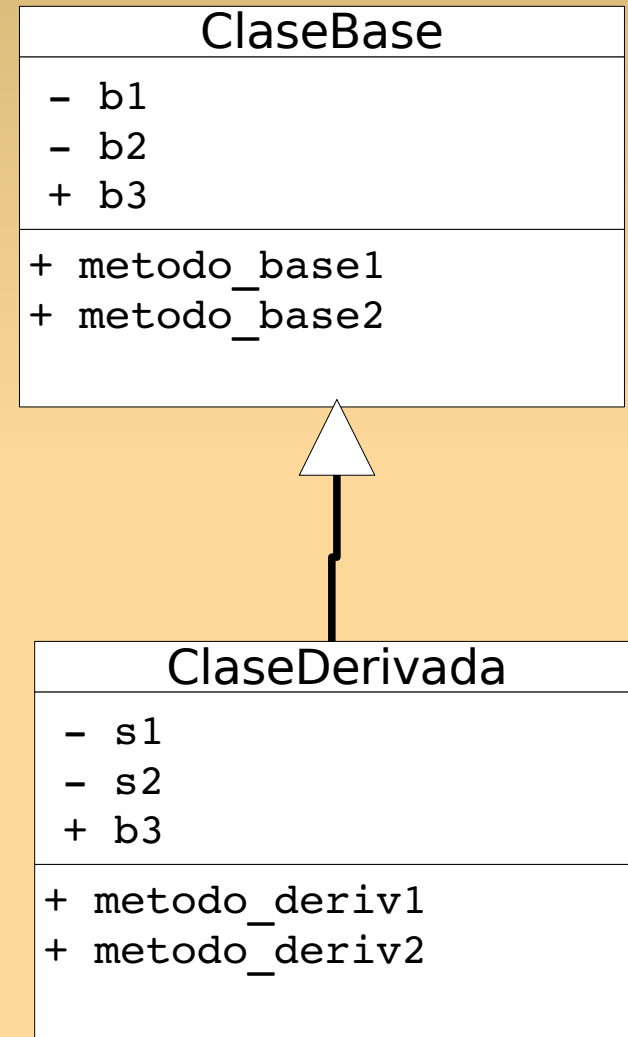
Tipo de Dato Clase Base	Herencia con public	Herencia con private	Otros
Private	No accesible directamente	No accesible directamente	No accesible
Protected	Protected	Private	No accesible
Public	Public	Private	Accesible (. ó ->)

17.2 Herencia

Ejemplo de control de acceso en la Herencia

```
class ClaseBase
{
    private:
        int b1 ;
        int b2 ;
    public:
        int b3 ;
        void metodo_base1 (int a);
        void metodo_base2 ();
};

class ClaseDerivada : public ClaseBase
{
    private:
        float s1 ;
        char s2 ;
    public:
        float metodo_deriv1 ();
        void metodo_deriv2 (char x);
};
```



17.2 Herencia

Ejemplo de control de acceso en la Herencia

```
class ClaseBase
{
    private:
        int b1 ;
        int b2 ;
    public:
        int b3 ;
        void metodo_base1 (int a);
        void metodo_base2 ();
};
class ClaseDerivada : public ClaseBase
{
    private:
        float s1 ;
        char s2 ;
    public:
        float metodo_deriv1 ();
        void metodo_deriv2 (char x);
};
```

Utilicemos objetos de las clases como hemos hecho hasta ahora:

```
ClaseBase objb1, objb2;
ClaseDerivada oder1, oder2;
objb1.metodo_base2( );
objb2.b3 = 8;
```

```
cout <<oder1.metodo_deriv ( );
oder2.metodo_base2( );
oder1.b3 = 5;
oder1.b1 = 7; //¿error?
```

Al heredar, desde la ClaseDerivada tampoco se se tiene acceso a los miembros privados de ClaseBase : b1 y b2

17.2 Herencia

Ejemplo de control de acceso en la Herencia

```
class ClaseBase
{
    public:
        int b1 ;
        int b2 ;
        int b3 ;
        void metodo_base1 (int a);
        void metodo_base2 ();
};

class ClaseDerivada : public ClaseBase
{
    private:
        float s1 ;
        char s2 ;
    public:
        float metodo_deriv1 ();
        void metodo_deriv2 (char x);
};
```

Si queremos acceder desde la clase derivada podemos declarar b1 y b2 también como públicos.

Mal asunto:

También se puede acceder desde cualquier lado de cualquier programa:

¡¡Perdemos el Encapsulamiento!!

17.2 Herencia

Ejemplo de control de acceso en la Herencia

```
class ClaseBase
{
    protected:
        int b1 ;
        int b2 ;
    public:
        int b3 ;
        void metodo_base1 (int a);
        void metodo_base2 ();
};

class ClaseDerivada : public ClaseBase
{
    private:
        float s1 ;
        char s2 ;
    public:
        float metodo_deriv1 ();
        void metodo_deriv2 (char x);
};
```

Protected:

Calificador que permite acceso desde las clase derivadas.

ClaseBase
b1
b2
+ b3
+ metodo_base1
+ metodo_base2

```
void metodo_deriv2(char x)
{
    s1 = 3.6;
    s2 = x;
    b1 = 4; // ¡¡Ahora sí!!
    b3 = 2;
}
```

17.2 Herencia

Resumen de Control de Acceso en la Herencia

```
class <clase_derivada> :  
[public|private] <base1> [, [public|private] <base2>] {};
```

Los miembros de una clase pueden ser:

private:

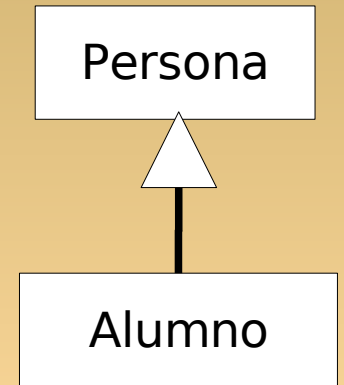
Miembros privados. Protegidos de todo acceso fuera del ámbito de la clase.

protected:

Miembros protegidos. Protegidos de todo acceso fuera del ámbito de su clase o de sus clases derivadas.

public:

Miembros públicos. Accesibles desde cualquier ámbito.



17.2 Herencia

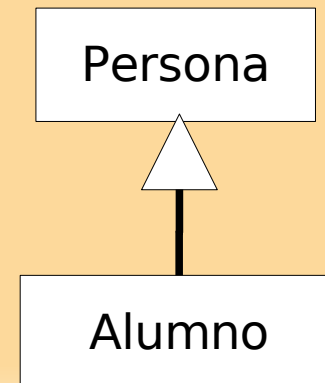
Modo de Derivación Público

```
class <clase_derivada> : public <clase_base> {};
```

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.



17.2 Herencia

Modo de Derivación Privado

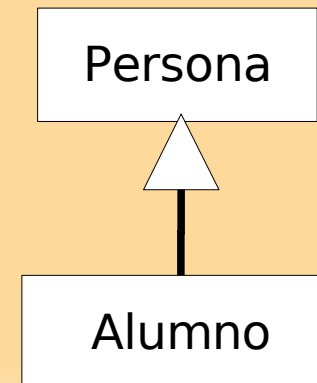
```
class <clase_derivada> : [private] <clase_base> {};
```

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.

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



17.2 Herencia

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 Alumno 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.

17.2 Herencia

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.**

17.2 Herencia

Ejemplo derivación o herencia pública

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

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

Privado en Alumno

Inaccesible desde sus funciones miembro.
Inaccesible desde fuera.

Público en Alumno

Accesible desde sus funciones miembro.
Accesible desde fuera.

17.2 Herencia

Ejemplo derivación o herencia pública (protected)

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

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

Protegido en Alumno

Accesible desde sus funciones miembro.
Inaccesible desde fuera.

Público en Alumno

Accesible desde sus funciones miembro.
Accesible desde fuera.

17.2 Herencia

Ejemplo derivación o herencia privada

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

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

Privado en Alumno

Inaccesible desde sus funciones miembro.
Inaccesible desde fuera.

Privado en Alumno

Accesible desde sus funciones miembro.
Inaccesible desde fuera.

17.2 Herencia

Ejemplo derivación o herencia privada (protected)

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

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

Protegido en Alumno

Accesible desde sus funciones miembro.
Inaccesible desde fuera.

Privado en Alumno

Accesible desde sus funciones miembro.
Inaccesible desde fuera.