

CONTENIDOS

1. Introducción
2. Cuándo se aplica el mecanismo de herencia
3. Un ejemplo de herencia
4. Terminología
5. Clase derivada. Creación de una clase derivada
6. Miembros que no se heredan automáticamente
7. Acceso a miembros heredados

Introducción

La **herencia** es un potente mecanismo para definir una nueva clase a partir de otra.

La nueva clase puede añadir características sin tener que reprogramar toda la clase de nuevo.

La **herencia** permite:

- Adoptar automáticamente características ya implementadas.

Ahorro de tiempo y esfuerzo

- Adoptar automáticamente características ya probadas.

Menor tiempo de prueba y depuración

¿Cuándo se aplica el mecanismo de 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

Un ejemplo de herencia

```
class Persona
{
  private:
    char * nif;
    int edad;
    char * nombre, *apellidos;
  public:
    Persona(char * , int = 0, char *, char * );
    Persona & operator=( Persona &);
    ~Persona(); // Destructor
    void medad(int);
    void mnombre(char *);
    char * mnombre() ;
    void mostrar() ;
    char * nombreCompleto() ;
    void felizCumple(); // El día del cumpleaños
    void leer(); // Lectura de los datos de la persona
};
```

Supongamos que queremos implementar una clase

Alumno:

→ Los alumnos tienen NIF, nombre y apellidos.

Los atributos de la clase **Persona** son adecuados para la clase **Alumno**.

→ Los métodos de la clase **Persona** también son adecuados para la clase **Alumno**.

Herencia (I)

Un 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

Terminología

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.

Herencia (I)

Clase derivada: la nueva clase que hereda

- ◆ 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.

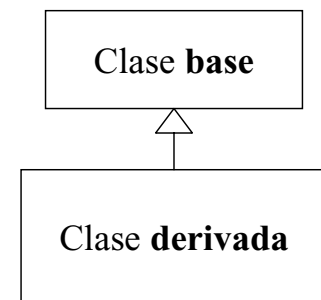


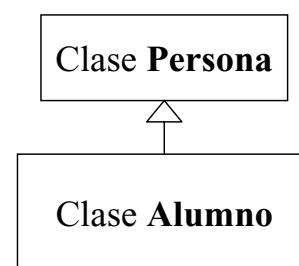
Diagrama UML que muestra la relación de herencia.

El proceso de herencia no afecta de ningún modo a la **clase base**.

Herencia (I)

Clase derivada: la clase Alumno

- La **clase Alumno** hereda todos los miembros privados de la clase **Persona**, es decir incorpora los miembros privados sin tener que volver a declararlos.
- La **clase Alumno** hereda los miembros públicos de la clase **Persona**, incorpora los miembros públicos sin tener que volver a declararlos.
- Además, la clase **Alumno** puede definir atributos adicionales, por ejemplo, grupo del alumno.
- La **clase Alumno** puede redefinir métodos heredados de la clase **Persona** (Por ejemplo mostrar ())
- La **clase Alumno** puede anular características heredadas de la clase **Persona**.



7

Herencia (I)

Creación de la clase derivada: sintaxis

Cuando se crea una clase (clase derivada) a partir de otra ya existente (clase base), se utiliza la siguiente sintaxis:

```
class Subclase : public Superclase
{
    // lista de miembros públicos y privados
};
```

Ha de indicarse la clase de la que se parte

Subclase adopta todos los miembros públicos y privados.

Subclase puede definir miembros adicionales, así como redefinir los heredados.

8

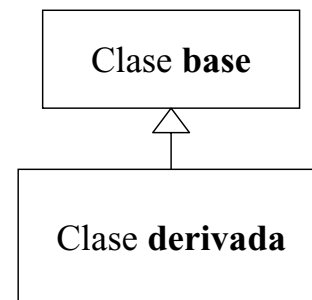
Herencia (I)

Creación de la clase derivada: sintaxis

Muy importante

En las funciones miembro definidas en la subclase, no se tiene acceso a lo privado de la superclase, a pesar de ser miembros que se heredan.

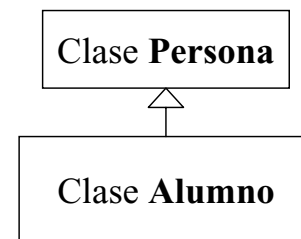
```
class Subclase : public Superclase
{
    // lista de miembros públicos y privados
};
```



Herencia (I)

Un ejemplo de herencia

```
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
{
private:
    char curso[5 ] ;
    char * titulacion ;
public:
    void actualizar();
};
```

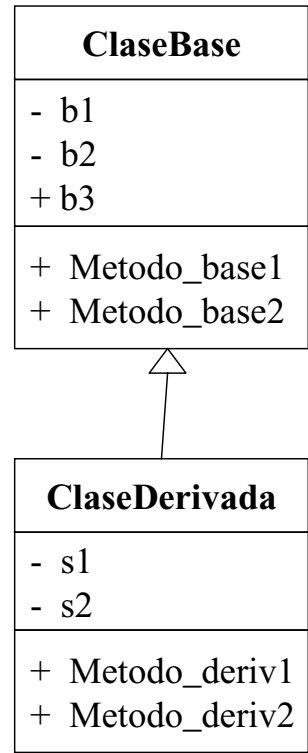
El metodos actualizar() de la clase **Alumno**, no tiene acceso a los atributos privados heredados (nif, edad, nombre, apellidos)

Herencia (I)

Ejemplo

```
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);
};
```



Herencia (I)

Ejemplo

```
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);
};
```

ClaseBase:

Privado:

b1, b2

Público:

b3, metodo_base1(int) ,
metodo_base2()

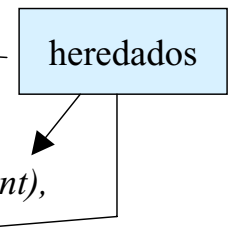
ClaseDerivada:

Privado:

b1, b2 ,
s1, s2

Público:

b3, metodo_base1(int),
metodo_base2(),
metodo_deriv1(),
metodo_deriv2 (char).



Herencia (I)

Ejemplo

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

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

Con las declaraciones anteriores, podremos definir objetos de las dos 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
...
```



Herencia (I)

Acceso a miembros heredados

```
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);
};
```

Fuera de la **ClaseBase** tampoco se puede acceder a los atributos **b1** y **b2**

Fuera de la **ClaseDerivada** no se puede acceder a los atributos **s1** y **s2**

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

Acceso a miembros heredados

Si queremos acceder a los miembros privados de `ClaseBase` desde los métodos de `ClaseDerivada` una solución sería declarar `b1` y `b2` como públicos:

```
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);
};
```

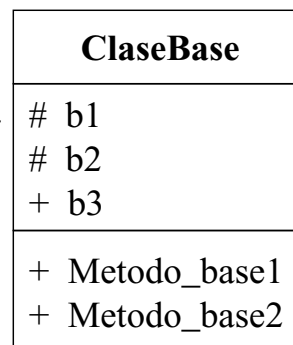
Inconveniente: Desde cualquier sitio se puede acceder a `b1` y `b2`, por tanto se pierde la propiedad de **ocultación de la información**.

Acceso a miembros heredados

Para solucionar éste problema, basta con definir los miembros `b1` y `b2` como miembros protegidos (`protected`).

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

Protected
Calificador que permite acceso desde clases derivadas



Representación en UML del nivel de acceso protected

Herencia (I)

Ejemplo

```
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);
};
```

Ahora sí que se puede acceder a los atributos b1 y b2 desde los métodos de la clase derivada.

```
...
ClaseBase ob1, ob2;
ClaseDerivada oder1, oder2;
oder1.b3 = 5;
...
void metodo_deriv2 (char x)
{
    s1 = 3.6;
    s2 = x;
    b1 = 4; // Permitido !!
    b3 = 2;
}
...
```

Herencia (I)

En resumen ...

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.

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.