FACULTAD DE INFORMÁTICA

Herencia

RESUMEN

Programación orientada a objetos — Unidad 6

Autor: Luis Hernández Yáñez

Introducción

Mecanismo exclusivo y fundamental de la POO.

La herencia no está contemplada en la programación basada en tipos (TAD).

Es el principal mecanismo que ayuda a fomentar y facilitar la reutilización del software:
Las clases como componentes software reutilizables.

Si se necesita una nueva clase de objetos y se detectan *suficientes* similitudes con otra clase ya desarrollada, se toma esa clase existente como punto de partida para desarrollar la nueva:

- ✓ Se adoptan automáticamente características ya implementadas
  ⇒ Ahorro de tiempo y esfuerzo
- ✓ Se adoptan automáticamente características ya probadas
  ⇒ Menor tiempo de prueba y depuración

Se puede partir de varias clases (herencia múltiple).

Herencia

A partir de una clase base (superclase)
se deriva una subclase (clase derivada)

La subclase adopta automáticamente todos los atributos y métodos de la superclase (aunque puede no tener acceso directo a ellos).

La subclase puede añadir nuevos atributos y métodos propios, así como redefinir métodos heredados.

La herencia refleja un proceso de especialización (tipos/subtipos).

```
class Subclase : Superclase { ...
```

En la implementación de la subclase (en las funciones miembro definidas en la propia subclase) *no se tiene acceso a lo privado de la superclase*, a pesar de que se trate de miembros que se heredan de la misma forma que los públicos.

Modos de derivación: público / privado

Herencia: modo de derivación público

Herencia con modo de derivación público:

Palabra reservada **public** precediendo el nombre de la superclase.

Los miembros heredados mantienen su modo de acceso:

- ✓ Los que son públicos en la superclase siguen siendo públicos (y accesibles) en la subclase.
- ✓ Los que son privados en la superclase siguen siendo privados en la subclase y se ocultan, resultando inaccesibles en ésta.

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

A los objetos de la subclase se les puede seguir pasando mensajes que correspondan a métodos públicos heredados.

Es, con diferencia, el modo de derivación más habitual.

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

Herencia: modo de derivación privado

Herencia con modo de derivación privado (el predeterminado):

Palabra reservada **private** precediendo el nombre de la superclase.

Todos los miembros heredados se convierten en privados:

- ✓ Los que son públicos en la superclase pasan a ser privados en la subclase, pero no se ocultan y se pueden acceder en ésta.
- ✓ Los que son privados en la superclase siguen siendo privados en la subclase y se ocultan, resultando inaccesibles en ésta.

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 pasar mensajes que correspondan a métodos públicos heredados (se han convertido en privados en la subclase).

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


Los Alumnos son Personas

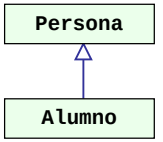
```
class Persona {
public: ...
  // miembros públicos
private: ...
  // miembros privados
};

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

Superclase (clase base)

Modo de derivación público

Subclase (clase derivada)

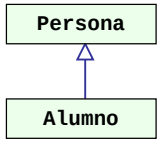


Sin tener que hacerlo explícito (sin tener que volver a declararlos):

- ✓ 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.

Sin embargo, las funciones miembro que se definan en la clase **Alumno** no podrán acceder a los miembros privados heredados.

class Alumno : public Persona { ...

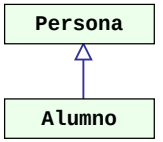



```
clase Persona
privado: _nif  _nombre  _apellidos  _edad
público: nif(string)   nombre(string)
         apellidos(string)   edad(int)
         nif()   nombre()   apellidos()
         edad()   leer()   mostrar()
         nombreCompleto()   felizCumple()

clase Alumno: características heredadas
privado: _nif  _nombre  _apellidos  _edad   (no accesibles)
público: nif(string)   nombre(string)
         apellidos(string)   edad(int)
         nif()   nombre()   apellidos()
         edad()   leer()   mostrar()
         nombreCompleto()   felizCumple()
```

Más adelante veremos cómo afecta la herencia a los constructores, destructores y operadores de asignación.

class Alumno : private Persona { ...




```
clase Persona
privado: _nif  _nombre  _apellidos  _edad
público: nif(string)   nombre(string)
         apellidos(string)   edad(int)
         nif()   nombre()   apellidos()
         edad()   leer()   mostrar()
         nombreCompleto()   felizCumple()

clase Alumno: características heredadas
privado: _nif  _nombre  _apellidos  _edad   (no accesibles)
         nif(string)   nombre(string)
         apellidos(string)   edad(int)
         nif()   nombre()   apellidos()      } (accesibles)
         edad()   leer()   mostrar()
         nombreCompleto()   felizCumple()
público: nada
```

Más adelante veremos cómo afecta la herencia a los constructores, destructores y operadores de asignación.

Miembros protegidos (modo de derivación público)

```
clase Persona
protegido:
   _nif        _nombre
   _apellidos  _edad
público:
   nif(string)   nombre(string)
   apellidos(string)   edad(int)
   nif()   nombre()   apellidos()
   edad()   leer()   mostrar()
   nombreCompleto()
   felizCumple()

class Alumno : public Persona
...
```

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

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

Miembros protegidos (modo de derivación privado)

```
clase Persona
protegido:
   _nif        _nombre
   _apellidos  _edad
público:
   nif(string)   nombre(string)
   apellidos(string)   edad(int)
   nif()   nombre()   apellidos()
   edad()   leer()   mostrar()
   nombreCompleto()
   felizCumple()

class Alumno : private Persona
...
```

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

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

Herencia en C++: accesibilidad en la derivación

En este curso no usaremos miembros protegidos. Los atributos siempre irán en la sección **private**.

	Miembros de la superclase		
	públicos	protegidos	Privados
En la subclase			
Con derivación pública:	públicos	protegidos	privados (ocultos)
Accesibles a las funciones miembro	SI	SI	NO
Accesibles desde fuera de los objetos	SI	NO	NO
Con derivación privada:	privados	protegidos	privados (ocultos)
Accesibles a las funciones miembro	SI	SI	NO
Accesibles desde fuera de los objetos	NO	NO	NO

Miembros que no se heredan

Los constructores, el destructor y el operador de asignación no se heredan.

Cuando se crea un ejemplar de la subclase se invocan automáticamente todos los constructores de las superclases, empezando por el de la más alta en la jerarquía; finalmente se ejecuta el constructor propio de la subclase.

Cuando se destruye un ejemplar de la subclase se ejecuta primero el destructor propio de la subclase y luego se invocan automáticamente todos los destructores de las superclases, empezando por el de la clase madre.

En el constructor de la subclase se pueden indicar datos con los que se debe ejecutar el constructor de la superclase (cuando le toque).

Y en el constructor de copia y el operador de asignación de la subclase se pueden invocar, respectivamente, el constructor de copia y el operador de asignación de la superclase.

La subclase Alumno

```
// Clase Alumno - Archivo de cabecera "alumno.h"

#ifndef alumno_h // Evitar inclusiones múltiples
#define alumno_h

#include <iostream>
#include <string>
using namespace std;
#include "persona.h"

class Alumno : public Persona {
public:
  Alumno(string = "", int = 0, string = "", string = "",
         int = 1); // NIF, edad, nombre, apellidos y curso
  Alumno(const Alumno&); // Constructor de copia
  Alumno& operator=(const Alumno&); // Copia
  ~Alumno(); // Destructor
```

Su propia FCO

(*continúa*)

La subclase Alumno

```
  void curso(int); // Mutador
  int curso() const; // Accedente
  void mostrar() const; // Método redefinido
private:
  int _curso; // Nuevo atributo
};

#endif
```

Nuevos atributo y métodos

Método heredado redefinido

La subclase Alumno

```
// Clase Alumno - Implementación "alumno.cpp"

#include "alumno.h"

Alumno::Alumno(string nif, int edad, string nombre,
               string apellidos, int curso)
  : _curso(curso), Persona(nif, edad, nombre, apellidos)
  { }
```

De esta forma se indican los argumentos con los que se deberá ejecutar el constructor de la superclase (cuando le toque)

```
Alumno::Alumno(const Alumno& otro) : Persona(otro) {
  _curso = otro._curso;
}
```

De esta forma se fuerza la ejecución del constructor de copia de la superclase

(*continúa*)

La subclase Alumno

```
Alumno& Alumno::operator=(const Alumno& otro) {
  Persona::operator=(otro);
  _curso = otro._curso;
  return *this;
}

Alumno::~Alumno() { }

void Alumno::curso(int num) { _curso = num; }

int Alumno::curso() const { return _curso; }

void Alumno::mostrar() const {
  Persona::mostrar();
  cout << "Curso: " << _curso << endl;
}
```

Así se fuerza la ejecución del operador de asignación de la superclase

Los métodos redefinidos todavía están accesibles cualificando con el nombre de la superclase

Prueba de la subclase **Alumno**

```
#include "alumno.h"

int main() {
  Alumno alumno1("223344D", 21, "Jorge", "Casa Ros", 2);
```

1. Se ejecuta el constructor de la superclase **Persona**, usando los argumentos que se indican en el *inicializador* del constructor de la subclase (**Persona("223344D",21,"Jorge","Casa Ros")**).
   Los atributos heredados **_nif**, **_edad**, **_nombre** y **_apellidos** quedan inicializados con los valores proporcionados.
2. Se ejecuta el constructor propio de la subclase, que establece el último argumento proporcionado como valor del atributo **_curso**.

(*continúa*)

Prueba de la subclase **Alumno**

```
  Alumno alumno2(alumno1);
```

1. Se ejecuta el constructor de copia de **Alumno**, que lo primero que hace es forzar la ejecución del constructor de copia de la superclase **Persona**, pasándole como argumento **alumno1** (un alumno es una persona).
   Los atributos heredados **_nif**, **_edad**, **_nombre** y **_apellidos** quedan creados con los mismos valores que los de **alumno1**.
2. Se termina de ejecutar el constructor de copia propio de la subclase, que copia el atributo **_curso** de **alumno1**.

No se ejecuta automáticamente ningún constructor.
Se hace sólo lo que diga el propio constructor de copia.

(*continúa*)

Prueba de la subclase **Alumno**

```
  Alumno alumno3;
  // Se ejecuta el constructor predeterminado

  alumno3 = alumno2;
```

1. Se ejecuta el operador de asignación de **Alumno**, que lo primero que hace es forzar la ejecución del operador de asignación de la superclase **Persona**, pasándole como argumento **alumno1** (un alumno es una persona).
   Los atributos heredados **_nif**, **_edad**, **_nombre** y **_apellidos** quedan con los mismos valores que los de **alumno1**.
2. Se termina de ejecutar el operador de asignación propio de la subclase, que copia el atributo **_curso** de **alumno1**.

No se ejecuta automáticamente nada

(*continúa*)

Prueba de la subclase **Alumno**

```
  alumno3.mostrar();
  alumno3.felizCumple();
```

```
223344D
Jorge Casa Ros
Edad: 21
Curso: 2
Registro actualizado:
223344D
Jorge Casa Ros
Edad: 22
```

¿Un **Alumno** o simplemente una **Persona**?

```
  return 0;
}
```

Para cada objeto, se ejecuta primero el destructor de **Alumno** y luego el destructor de **Persona**