FACULTAD DE INFORMÁTICA

Más sobre herencia

RESUMEN

Programación orientada a objetos — Unidad 7

Autor: Luis Hernández Yáñez

Subclase **Alumno** con atributo de la clase **Persona**

El archivo **alumno.h**

```
#ifndef alumno_h
#define alumno_h

#include <iostream>
#include <string>
using namespace std;
#include "persona.h" // Inclusión de la clase Persona

// Todos los métodos implementados fuera
class Alumno : public Persona {
public:
  // Constructor (predeterminado):
  Alumno(Persona = Persona(), string = "", int = 0,
         string = "", string = "", int  = 1);
  Alumno(const Alumno&); // Constructor de copia
  Alumno& operator=(const Alumno&); // Copia
  ~Alumno(); // Destructor
```

(*continúa*)

Subclase **Alumno** con atributo de la clase **Persona**

```
  void curso(int);
  void profesor(Persona);
  int curso() const;
  Persona profesor() const;
  void leer(); // No pide los datos del profesor
  void mostrar() const;
private:
  int _curso;
  Persona _profesor; // Atributo de clase Persona
};

#endif
```


Subclase **Alumno** con atributo de la clase **Persona**

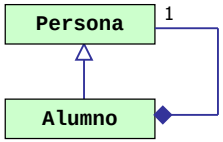
El archivo **alumno.cpp**

```
#include "alumno.h"

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

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

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

Alumno::~Alumno() { }
```

(*continúa*)

Subclase **Alumno** con atributo de la clase **Persona**

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

void Alumno::profesor(Persona profe)
{ _profesor = profe; }

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

Persona Alumno::profesor() const { return _profesor; }

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

void Alumno::leer() {
  Persona::leer();
  cout << "Curso: ";
  cin >> _curso;
}
```



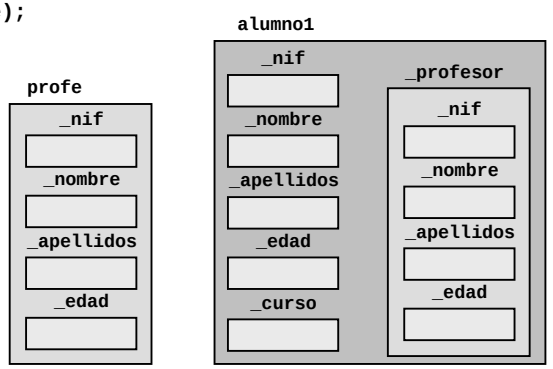
La clase **Alumno** es subclase de **Persona** y también contiene una **Persona** (el profesor)

>>> alumno.cpp

Subclase **Alumno** con atributo de la clase **Persona**

Prueba de la subclase **Alumno**

```
#include "persona.h"
#include "alumno.h"
int main()
{
  Persona profe("445566F", 33, "LUIS", "HERNANDEZ YAÑEZ");
  Alumno alumno1(profe);
  alumno1.leer();
  alumno1.mostrar();
  return 0;
}
```



Relaciones entre clases

En unidades anteriores vimos cómo se establecen las relaciones de clientelismo entre las clases (*una clase hace uso de objetos de otra*).

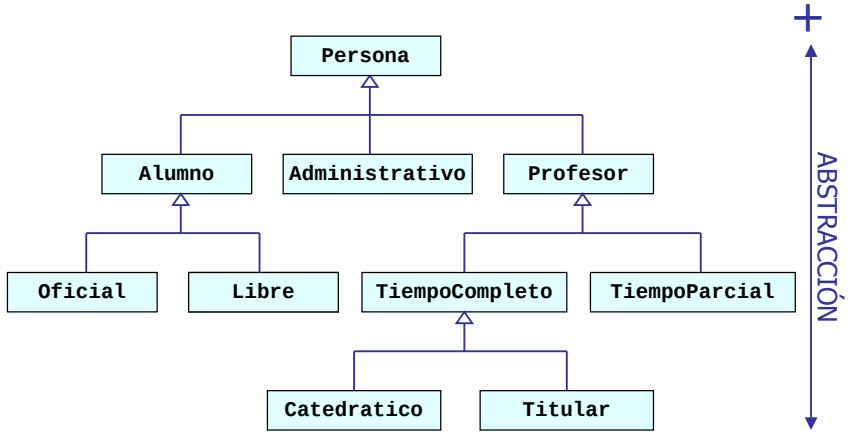
Hemos visto que también se puede establecer una relación de herencia entre las clases.

- ✓ Clientelismo: la clase A es cliente de la clase B si en la implementación de la clase A se hace uso de objetos de la clase B.
- ✓ Herencia: la clase A desciende (se deriva) de la clase B (mediante derivación directa -A es subclase de B- o mediante una cadena de derivación -A es subclase de una subclase de B, por ejemplo-).

Las relaciones entre clases establecen dependencias entre ellas que se han de contemplar en la reutilización: para utilizar una clase C se necesita no sólo la implementación de dicha clase, sino también la de todas aquellas otras de las que dependa (por herencia o por clientelismo).

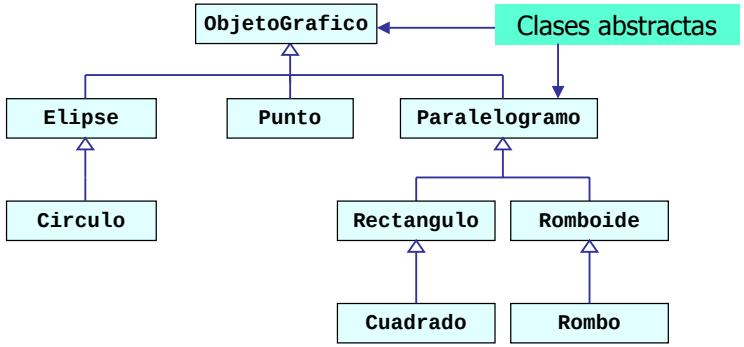
Jerarquía(s) de clases

Utilizando diagramas de UML:



Múltiples jerarquías

Clases abstractas



En el programa de dibujo sólo se van a crear objetos gráficos concretos: puntos, elipses, círculos, cuadrados, rectángulos, romboides o rombos.

Modelan el comportamiento común de todas sus subclases.

Ejecución de los pasos de mensajes con herencia

```
Alumno unAlumno; ...    unAlumno.mostrar();
```

El método **mostrar()** está definido en la clase **Alumno**, por lo que se ejecuta dicho método (el mensaje **mostrar()** se vincula con el método **mostrar()** de la clase **Alumno**).

```
Alumno unAlumno; ... unAlumno.felizCumple();
```

El método **felizCumple()** NO está definido en la clase **Alumno**, por lo que se busca en la superclase (**Persona**); allí sí se encuentra, ejecutándose ese método (el mensaje **felizCumple()** se vincula con el método **felizCumple()** de la clase **Persona**).

```
Alumno unAlumno; ... unAlumno.pasarCurso(); // Error
```

No hay ningún método **pasarCurso()** definido en la clase **Alumno** ni en su superclase **Persona**, por lo que se concluye que el objeto **unAlumno** no entiende el mensaje **pasarCurso()** (el mensaje **pasarCurso()** no se puede vincular con ningún método).

Problemas con la vinculación de mensajes

```
int main()
{
  Persona p("445566F", 33,
    "LUIS", "HERNANDEZ YANEZ");
  Alumno al(p, "223344G", 21,
    "ROSA", "RATO PALOMO", 2);
  al.mostrar();
  cout << endl;
  al.felizCumple();

  return 0;
}
```

```
223344G
ROSA RATO PALOMO
Edad: 21
curso: 2
Profesor:
445566F
LUIS HERNANDEZ YANEZ
Edad: 33

Registro actualizado:
223344G
ROSA RATO PALOMO
Edad: 22
```

```
void Persona::felizCumple() {
  _edad++;
  cout << "Registro actualizado:" << endl;
  mostrar();
}
```

_edad incrementada, pero se muestra información de **Persona**

Solución: métodos virtuales

>>> prueba.cpp

Compatibilidad entre objetos de distintas clases

Todos los alumnos son personas,
pero no todas las personas son alumnos

A un identificador de la clase **Persona** (superclase) se le puede asignar un objeto de la clase **Alumno** (subclase), pero a un identificador de la clase **Alumno** (subclase) **no** se le puede asignar un objeto de la clase **Persona** (superclase)

```
Persona unaPersona;
Alumno unAlumno;
...
unaPersona = unAlumno; // Correcto:
                       // todos los alumnos son personas
unAlumno = unaPersona; // INCORRECTO:
                       // no todas las personas son alumnos
```

A un identificador de una clase sólo se le pueden asignar objetos de esa clase o de cualquiera de sus subclases

Más problemas con la vinculación de mensajes

```
int main()
{
  Persona p("445566F", 33, "LUIS", "HERNANDEZ YANEZ");
  Alumno al(p, "223344G", 21, "ROSA", "RATO PALOMO", 2);
  al.mostrar();
  cout << endl;
  p = al;
  // ahora p es igual
  // a al (el alumno)
  p.mostrar();

  return 0;
}
```

```
223344G
ROSA RATO PALOMO
Edad: 21
curso: 2
Profesor:
445566F
LUIS HERNANDEZ YANEZ
Edad: 33

223344G
ROSA RATO PALOMO
Edad: 21
```

Se muestra información de **Persona**

Solución: métodos virtuales

Una lista de **Personas** y **Alumnos**

```
#ifndef listaper_h
#define listaper_h
#include "persona.h"

class Lista {
public:
  Lista();
  Lista(const Lista&);
  Lista& operator=(const Lista&);
  ~Lista();
  bool llena() const;
  bool vacia() const;
  bool insertar(Persona);
  bool recuperar(int, Persona&) const;
  void mostrar() const;
private:
  enum { MAX = 100 };
  Persona _array[MAX];
  int _cont;
};
#endif
```

Nada impide insertar alumnos
`miLista.insertar(alumno);`

No cambia NADA respecto de la implementación de la Unidad 5

`>>> listaper.h`

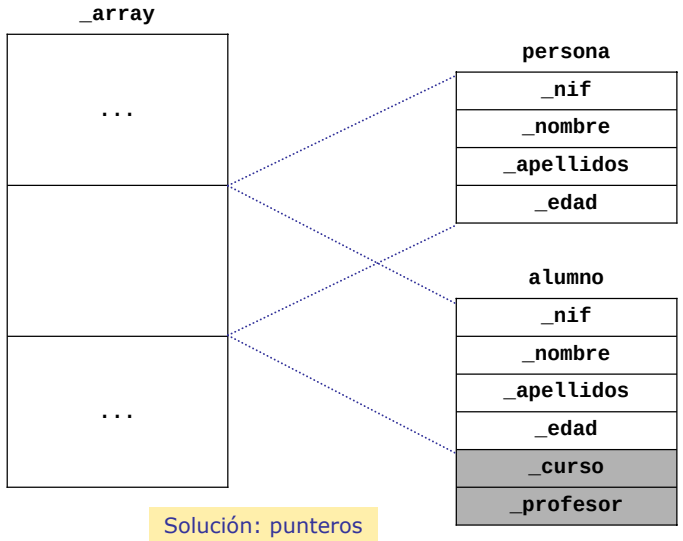
Una lista de **Personas** y **Alumnos**

```
1 - Insertar persona
2 - Insertar alumno
3 - Listar
0 - Salir
Opción: 2

NIF: 765983P
Nombre: Juan
Apellidos: García Santos
Edad: 19
Curso: 2
1 - Insertar persona
2 - Insertar alumno
3 - Listar
0 - Salir
Opción: 3

Elementos de la lista:
765983P
Juan García Santos
Edad: 19
```

Se lee (e inserta) un alumno

Pero se muestra información de persona

Una lista de **Personas** y **Alumnos**



Solución: punteros