Universidad Autónoma de Entre Ríos | FCyT Facultad de Ciencia y Tecnología

Clases y Objetos

Programación Orientada a Objetos

Lic. Leonardo Brambilla

Programación Orientada a Objetos (POO)

POO

Es un paradigma (modelo de trabajo) de programación que agrupa los datos y los procedimientos para manejarlos en una única entidad: **el objeto**

Objeto

Un objeto es una unidad que engloba en sí mismo variables y funciones necesarios para el tratamiento de esos datos.

Método

Función perteneciente a determinado objeto.

Atributo

Variable perteneciente a determinado objeto.

Definiciones (POO)

Clase

Se puede considerar como un patrón para construir objetos.

Interfaz

Es la parte del objeto que es visible para el resto de los objetos. Es decir, es el conjunto de métodos y atributos que dispone un objeto para comunicarse con él.

Herencia

Capacidad de crear nuevas clases basándose en clases previamente definidas de las que se aprovechan ciertos datos y métodos, se desechan otros y se añaden nuevos.

Jerarquía

Orden de subordinación de un sistema de clases.

Polimorfismo

Propiedad según la cual un mismo objeto puede considerarse como perteneciente a distintas clases.

Constructores (POO)

Son métodos especiales que sirven para inicializar un objeto de una determinada clase al mismo tiempo que se declara.

Si no definimos un constructor el compilador creará uno por defecto, sin parámetros.

Si definimos un constructor que requiere argumentos, es obligatorio suministrarlos. Será llamado siempre que se declare un objeto de esa clase.

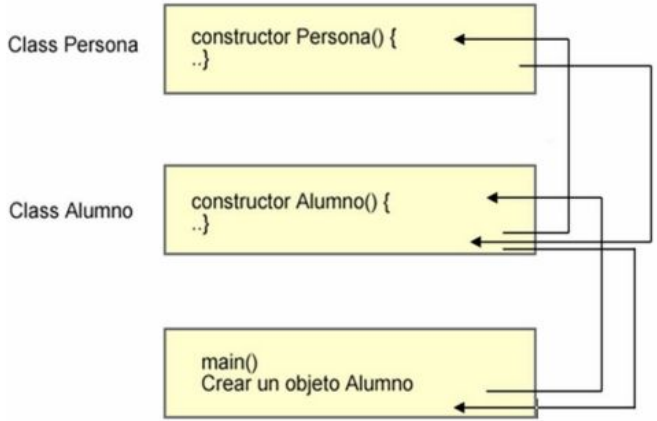
Encadenamiento de constructores

Aseguramos que se creen los objetos

teniendo al menos un estado mínimo.

```
Alumno::Alumno(long dni) : Persona(dni) {

}
```



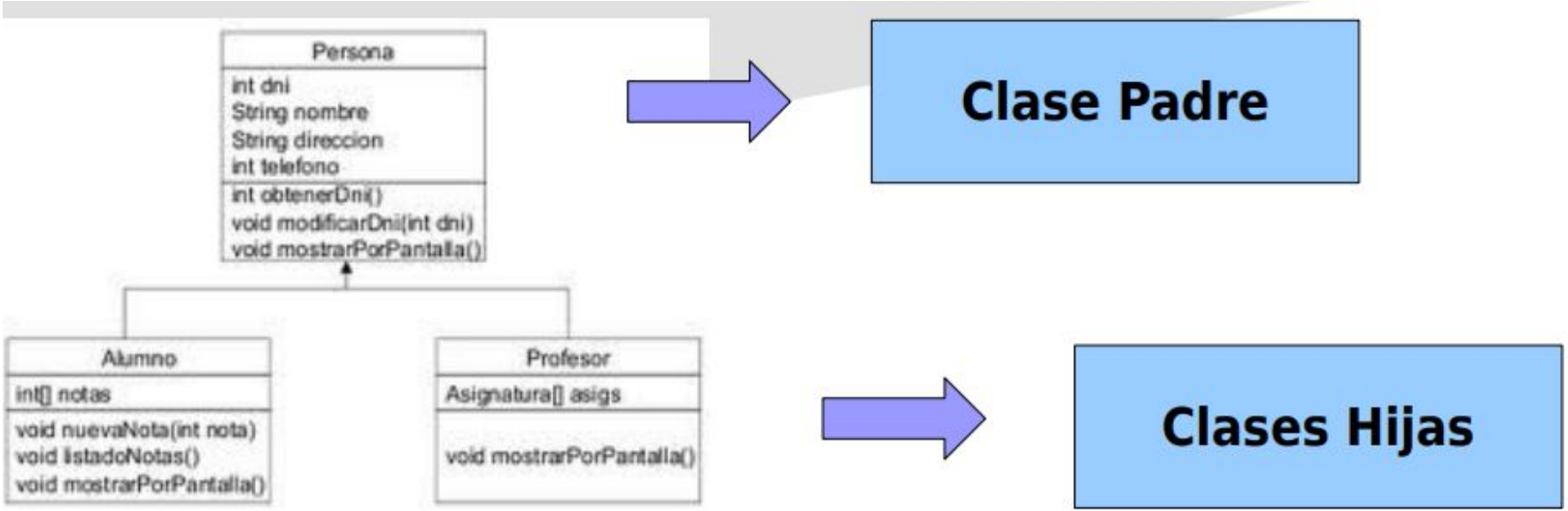
Herencia (POO)

La reusabilidad puede lograrse mediante **herencia**.

Un comportamiento definido en una superclase es heredado por sus subclases.

Las subclases extienden la funcionalidad heredada

Permite definir la mayor cantidad de funcionalidades y atributos y luego reutilizarlas.



Sobreescritura de métodos(POO)

Consiste en reescribir la implementación de un método de instancia, con su mismo nombre y argumentos (firma del método).

Si una clase hija sobre-escribe un método de su clase padre, entonces ocultará al mismo.

```cpp
#ifndef PERSONA_H
#define PERSONA_H


class Persona
{
private:
    long dni;
    char * nombre;
public:
    Persona(long dni);
    long getDni();
    char * getNombre();
    void setNombre(char * nombre);
    void virtual mostrarInfo();
};

#endif // PERSONA_H
```

```cpp
#ifndef ALUMNO_H
#define ALUMNO_H
#include "persona.h"

class Alumno : public Persona
{
public:
    Alumno(long dni);
    void mostrarInfo();
};

#endif // ALUMNO_H
```

Downcasting y Upcasting de Objetos (POO)

Un objeto de un tipo se puede tratar como objeto de otro tipo siempre que la clase fuente y la clase destino estén relacionadas por herencia. En c++ un puntero puede ser tipado por su clase general pero apuntar a una subclase.

Upcasting: conversión de clase derivada a la clase base, se hace implícitamente. Es posible asignar a un puntero de tipo clase_padre la dirección de memoria de un objeto de tipo clase_hija.

```
Persona * p = new Alumno(555555);
p->mostrarInfo();
return 0;
```

Downcasting: conversión de clase base a la clase derivada. Es explícito

```
Persona * p = new Alumno(555555);
Alumno * a = (Alumno *) p;
a->mostrarInfo();
```

Métodos y Clases abstract (POO)

El calificador **abstract** condiciona el diseño de una jerarquía de herencia.

Clases Abstract:

- No pueden ser instanciadas.

Métodos Abstract:

- No pueden tener implementación.
- Deben ser implementados para las clases no abstractas que extiendan de su clase.

Polimorfismo (POO)

Una variable de referencia cambia el comportamiento según el tipo de objeto al que apunta.

- Una variable trata a objetos de una clase como objetos de una superclase y se invoca dinámicamente el método correspondiente (***binding dinámico***)

- Si tenemos un método que espera como parámetro una variable de clase X, podemos invocarlo usando subclases pasando como parámetros referencias a objetos instancia de subclases de X.

Miembros virtuales

Los miembros virtuales son aquellos que están definidos en la clase madre, y se re definirán en las clases descendientes.

Sobrecarga de Operadores (POO)

Redefinir algunos de los operadores existentes en C++, para su funcionamiento con alguna clase específica.

- definir operaciones matemáticas con elementos tales como vectores y matrices.
- sobrecargar los operadores de los flujos de entrada y salida (>> y <<), de manera que puedan imprimir o leer estructuras o clases complejas con una sentencia estándar.

```
+ - * / % ^ & | ~  && || !
= += -= *= /= ^= &= |=
> < >= <= == != -> ->*
<< >> <<= >>= ++ -- [] ()
new new[] delete delete[]
```

- Los operadores :: (resolución del alcance),. (acceso de miembros),. * (acceso de miembros a través del puntero a miembro) y ?: (condicional ternario) no se pueden sobrecargar.
- No es posible cambiar la precedencia, la agrupación o el número de operandos de los operadores.

Sobrecarga de Operadores (POO)

Un operador sobrecargado puede ser miembro o friend de la clase para la que se define:

El que se defina de una forma u otra es en ocasiones cuestión de conveniencia o incluso de preferencia personal, mientras que en otros casos la decisión está impuesta.

Habitualmente:

- Se suelen declarar miembros de la clase los operadores unarios (es decir, aquellos que actúan sobre un único objeto), o los que modifican el primer operando, como sucede con los operadores de asignación.

- Por el contrario, los operadores que actúan sobre varios objetos sin modificarlos (por ejemplo los operadores aritméticos y lógicos) se suelen declarar como friends.

Sobrecarga de Operadores (POO)

cadena& operator= (const cadena&);

```
// operador de asignación sobrecargado (=)
cadena& cadena::operator= (const cadena& cd)
{
        if(*this != cd) {
                nchar = cd.nchar;
                delete [] pstr;
                pstr = new char[nchar + 1];
                strcpy(pstr, cd.pstr);
                cout << "Operador =" << endl;
        }
        return *this;
}
```

La definición del operador inserción (<<) sobrecargado es muy sencilla, pues lo único que se hace es insertar en el flujo de salida la cadena de caracteres estándar a la que apunta la variable miembro pstr.

friend cadena operator+ (const cadena&, const cadena&);

```
// operador de inserción en ostream
ostream& operator<< (ostream& co, const cadena& cad) {
        co << cad.pstr;
        return co;
}
```