Grado en Ingeniería Información

PROGRAMACIÓN II - Sesión 9

Tema 5.

Herencia

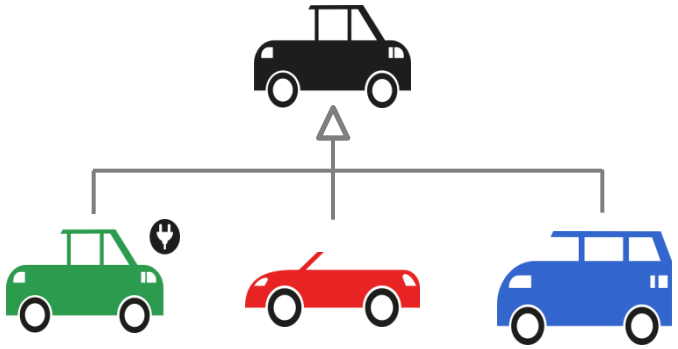
Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

UNIVERSIDAD NEBRIJA

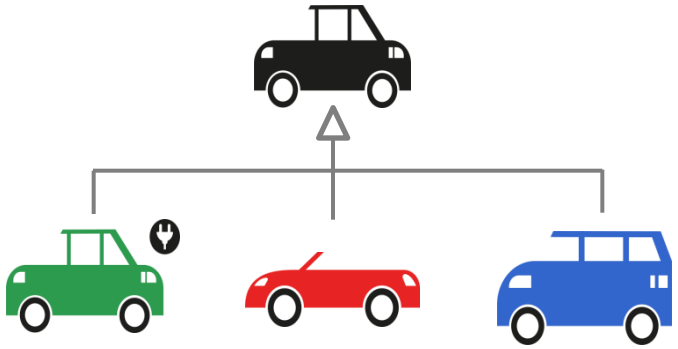
T5. Herencia



ÍNDICE

UNIVERSIDAD NEBRIJA

T5. Herencia



5.1. Herencia y Composición

5.2. Jerarquía de clases

5.3. Sintaxis en C++

5.4. Control de acceso en la herencia

5.5. Miembros que no se hereden automáticamente

5.6. Redefinición de métodos

5.7. Ejecución en los pases de mensajes con herencia

5.8. Herencia múltiple

ÍNDICE

Miembros que no se heredan automáticamente

NO se HEREDAN:

- **Constructores.**
- **Destructores.**
- El Operador de asignación (=) sobrecargado.
- Las funciones/clases ***friend***.

Miembros que no se heredan automáticamente

- Las clases derivadas **no heredan los constructores ni los destructores** de la clase base.
- Los miembros de la clase base **deben ser inicializados**:
  - **Si hay parámetros**, el constructor de la clase base debe ser **llamado de forma explícita** por el constructor de la clase derivada utilizando los parámetros para la inicialización.
  - **Si no hay parámetros** (constructor por defecto) **no es necesario hacer la llamada de forma explícita**.

¿Cuál es el orden de ejecución de los métodos constructores y destructores en las clases derivadas?

Al instanciar una clase derivada:

1.- Se ejecuta el **constructor** de la **clase base**

2.- Después, se ejecuta el **constructor** de la **clase derivada**

Al terminar el ciclo de vida de dicha clase:

1.- Se ejecuta el **destructor** de la **clase derivada**

2.- Después, se ejecuta el **destructor** de la **clase base**

Miembros que no se heredan automáticamente

Secuencia de ejecución válida para toda la jerarquía de clases

Al instanciar una clase derivada:

1.- Se ejecuta el **constructor** de la **clase base**

2.- Si existen objetos miembros, sus constructores se ejecutan después del constructor de la clase base.

3.- Por último, se ejecuta el **constructor** de la **clase derivada**

Al terminar el ciclo de vida de dicha clase, los destructores se llaman en orden inverso a la derivación:

1.- Se ejecuta el **destructor** de la **clase derivada**

2.- Después los destructores de los objetos miembros.

2.- Finalmente, se ejecuta el **destructor** de la **clase base**

Miembros que no se heredan automáticamente

```
#ifndef CLASES_H
#define CLASES_H

#include <iostream>
#include <cstring>
using namespace std;

class CBase
{   private:
    //protected:
        int b1;
        int b2;
    public:
        int b3;
        CBase():b1(0), b2(1), b3(2) {}
        CBase(int n1, int n2, int n3);
        void metodoB1(int a);
        void metodoB2() const;
};

class CDerivada : public CBase
{   private:
        float f1;
        char c1;
    public:
        CDerivada ();
        CDerivada (float d, char e);
        CDerivada (int a, int b, int c, float d, char e);
        float metodoD1();
        void metodoD2 (char c);
        void metodoD3() const;
};

#endif // CLASES_H
```

Miembros que no se heredan automáticamente

```cpp
#include <iostream>
#include <cstring>

#include "Clases.h"

using namespace std;

//---------------CLASE BASE--------------------
CBase::CBase(int n1, int n2, int n3){
	b1 = n1;
	b2 = n2;
	b3 = n3;
}

void CBase::metodoB1(int a){
	b1 = b2 = b3 = a;
}

void CBase::metodoB2() const{
	cout << "\n\t\tCLASE BASE" << endl;
	cout << "\n\t\tB1: " << b1 << " B2: " << b2 << " B3: " << b3 << endl << endl;

}
```

Miembros que no se heredan automáticamente

```
//----------------CLASE DERIVADA----------------------
// Ejemplo nº1 - caso base
CDerivada::CDerivada (){
// Ejemplo nº1 - caso para inicializar con un valor
   //  metodoB1(9);
// Ejemplo nº1 - caso para inicializar con valores
//     CDerivada::CDerivada ():CBase(9,7,6){
     f1=1.1;
     c1='a';
}


CDerivada::CDerivada (float d, char e){//:CBase(){
     f1=d;
     c1=e;
}

CDerivada::CDerivada (int a, int b, int c, float d, char e):CBase(a, b, c), f1(d), c1(e){
 //  f1=d;
 //  c1=e;
}

float CDerivada::metodoD1(){
     return f1;
}
```

Miembros que no se heredan automáticamente

```
float CDerivada::metodoD1(){
     return f1;
}

void CDerivada::metodoD2 (char c){
    c1 = c;
    f1 = 8.5;

 //    b1 = 10;       // PERMITIDO cuando son protected
 //    b2 = 15;     // PERMITIDO cuando son protected
    b3 = 20;
}

void CDerivada::metodoD3() const {
     cout << "\n\n\tDATOS DE LAS CLASES" << endl;
     cout << "\t====================" << endl;
     metodoB2();
     cout << "\n\t\tCLASE DERIVADA" << endl;
     cout << "\n\t\tF1: " << f1 << " C1: " << c1 << endl << endl;
}
```

Miembros que no se heredan automáticamente

```cpp
#include <iostream>
#include <cstring>

#include "Clases.h"

using namespace std;

int main ()
{
    cout << "\n\n\tEJEMPLO TEORIA Num. 1" << endl;
    CDerivada od1 (5.2, 'm');

    od1.metodoD3();

    CDerivada od2;

    od2.metodoD3();
```

```
DATOS DE LAS CLASES
===================

        CLASE BASE

        B1: 0 B2: 1 B3: 2


        CLASE DERIVADA

        F1: 5.2 C1: m


DATOS DE LAS CLASES
===================

        CLASE BASE

        B1: 0 B2: 1 B3: 2


        CLASE DERIVADA

        F1: 1.1 C1: a
```

Miembros que no se heredan automáticamente

```cpp
#include <iostream>
#include <cstring>

#include "Clases.h"

using namespace std;

int main ()
{
    cout << "\n\n\tEJEMPLO TEORIA Num. 1" << endl;
    CDerivada od1 (5.2, 'm');

    od1.metodoD3();

    CDerivada od2;

    od2.metodoD3();
```

```
DATOS DE LAS CLASES
===================

        CLASE BASE

        B1: 0 B2: 1 B3: 2


        CLASE DERIVADA

        F1: 5.2 C1: m


DATOS DE LAS CLASES
===================

        CLASE BASE

        B1: 9 B2: 9 B3: 9


        CLASE DERIVADA

        F1: 1.1 C1: a
```

Miembros que no se heredan automáticamente

```cpp
#include <iostream>
#include <cstring>

#include "Clases.h"

using namespace std;

int main ()
{
    cout << "\n\n\tEJEMPLO TEORIA Num. 1" << endl;
    CDerivada od1 (5.2, 'm');

    od1.metodoD3();

    CDerivada od2;

    od2.metodoD3();
```

```
DATOS DE LAS CLASES
===================

        CLASE BASE

        B1: 0 B2: 1 B3: 2


        CLASE DERIVADA

        F1: 5.2 C1: m


DATOS DE LAS CLASES
===================

        CLASE BASE

        B1: 9 B2: 7 B3: 6


        CLASE DERIVADA

        F1: 1.1 C1: a
```

Miembros que no se heredan automáticamente

```
#ifndef CLASES_H
#define CLASES_H

#include <iostream>
#include <cstring>
using namespace std;

class CBase
{   //private:
    protected:
        int b1;
        int b2;
    public:
        int b3;
        CBase():b1(0), b2(1), b3(2) {}
        CBase(int n1, int n2, int n3);
        void metodoB1(int a);
        void metodoB2() const;
};

class CDerivada : public CBase
{   private:
        float f1;
        char c1;
    public:
        CDerivada ();
        CDerivada (float d, char e);
        CDerivada (int a, int b, int c, float d, char e);
        float metodoD1();
        void metodoD2 (char c);
        void metodoD3() const;
};

#endif // CLASES_H
```

Miembros que no se heredan automáticamente

```
void CDerivada::metodoD2 (char c){
    c1 = c;
    f1 = 8.5;

    b1 = 10;       // PERMITIDO cuando son protected
    b2 = 15;       // PERMITIDO cuando son protected
    b3 = 20;
}

void CDerivada::metodoD3() const {
     cout << "\n\n\tDATOS DE LAS CLASES" << endl;
     cout << "\t====================" << endl;
     metodoB2();
     cout << "\n\t\tCLASE DERIVADA" << endl;
     cout << "\n\t\tF1: " << f1 << " C1: " << c1 << endl << endl;
}
```

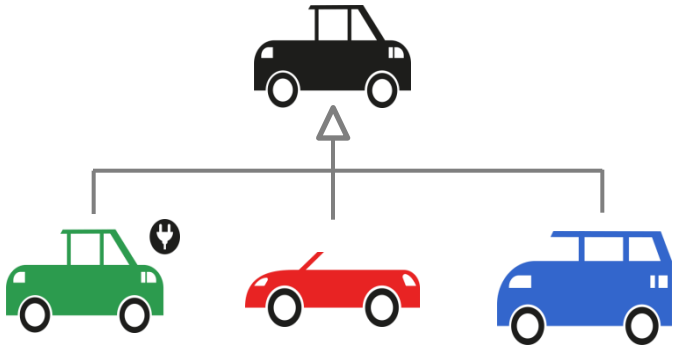
Miembros que no se heredan automáticamente

```
cout << "\n\n\tEJEMPLO TEORIA Num. 2" << endl;
CDerivada od3(11, 22, 33, 15.25, 'k');

od3.metodoD3();

od3.metodoD2('A');

od3.metodoD3();
```

```
DATOS DE LAS CLASES
==================

        CLASE BASE

        B1: 11 B2: 22 B3: 33


        CLASE DERIVADA

        F1: 15.25 C1: k


DATOS DE LAS CLASES
==================

        CLASE BASE

        B1: 10 B2: 15 B3: 20


        CLASE DERIVADA

        F1: 8.5 C1: A
```

UNIVERSIDAD NEBRIJA

T5. Herencia



5.1. Herencia y Composición

5.2. Jerarquía de clases

5.3. Sintaxis en C++

5.4. Control de acceso en la herencia

5.5. Miembros que no se hereden automáticamente

5.6. **Redefinición de métodos**

5.7. Ejecución en los pases de mensajes con herencia

5.8. Herencia múltiple

ÍNDICE

Redefinición o Superposición de Métodos es la **declaración**, en la **clase derivada**, de una **función miembro** con el **mismo nombre** que en la **clase base**.

La **superposición** de un método produce que se **oculten todos los métodos con el mismo nombre en la clase base**, aunque estuvieran sobrecargados.

El acceso a cualquiera de los métodos de la **clase base** que han sido **redefinidos** se hace utilizando el nombre de la **clase base como especificador de ámbito**:

ClaseBase::metodoQueSea(…);

Redefinición de métodos

```
class Persona {
    private:
        string nif;
        int edad;
        string nombre, apellidos;
    public:
        Persona():nif(""), edad(0), nombre(""), apellidos(""){}
        Persona(string const &identif, int aa, string const &nom, string const &apel):
                nif(identif), edad(aa), nombre(nom), apellidos(apel) {}
        ...
        void mostrar() const;
        void okMatricula() const;
};


class Estudiante : public Persona
{
    private:
        string curso;
    public:
        Estudiante (string const &id, int a, string const &nom, string const &ape,
                    string const &cur):Persona(id, a, nom, ape), curso(cur) {}
        void mostrar() const;
};
```

SOLUCIÓN MENOS ADECUADA

```
void Persona::mostrar() const {

    cout << "\n\n\tEl nombre del alumno es: " << nombre << " " << apellidos << endl;
    cout << "\n\tSu NIF es: " << nif << " y su EDAD: " << edad << endl;
}


void Estudiante::mostrar() const {

    cout << "\n\n\tEl nombre del alumno es: " << nombre << " " << apellidos << endl;
    cout << "\n\tSu NIF es: " << nif << " y su EDAD: " << edad << endl;

    cout << "\n\n\tEsta matriculado en el " << curso << " curso. ";
    cout << endl << endl;
}
```

SOLUCIÓN MÁS ADECUADA

```
void Persona::mostrar() const {

    cout << "\n\n\tEl nombre del alumno es: " << nombre << " " << apellidos << endl;
    cout << "\n\tSu NIF es: " << nif << " y su EDAD: " << edad << endl;
}


void Estudiante::mostrar() const {

    Persona::mostrar();

    cout << "\n\tEsta matriculado en el " << curso << " curso. ";
    cout << endl << endl;
}
```

Uso del método ***mostrar*** según el objeto instanciado de cada clase

```
Persona pers ("123456789G", 44, "Juan","Garcia");
Estudiante estud ("123456789S", 20, "Eva","Sanz", "Tercero");

pers.mostrar ();
estud.mostrar ();

cout << endl << endl;
```

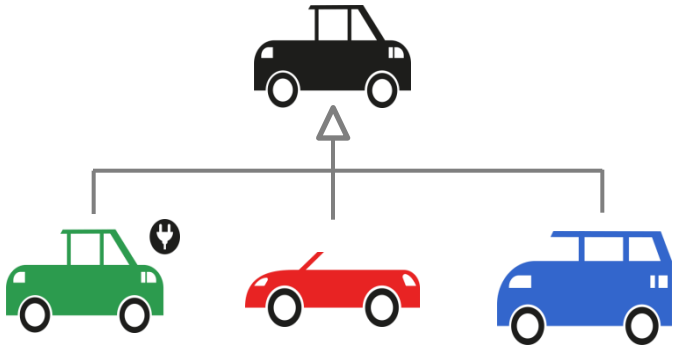
Uso del método ***mostrar*** según el objeto instanciado de cada clase

```
Persona pers ("123456789G", 44, "Juan","Garcia");
Estudiante estud ("123456789S", 20, "Eva","Sanz", "Tercero");

pers.mostrar ();
estud.mostrar ();

cout << endl << endl;
```

```
El nombre del alumno es: Juan Garcia

Su NIF es: 123456789G y su EDAD: 44


El nombre del alumno es: Eva Sanz

Su NIF es: 123456789S y su EDAD: 20


Esta matriculado en el Tercero curso.
```

UNIVERSIDAD NEBRIJA

T5. Herencia



5.1. Herencia y Composición

5.2. Jerarquía de clases

5.3. Sintaxis en C++

5.4. Control de acceso en la herencia

5.5. Miembros que no se hereden automáticamente

5.6. Redefinición de métodos

5.7. **Ejecución en los pases de mensajes con herencia**

5.8. Herencia múltiple

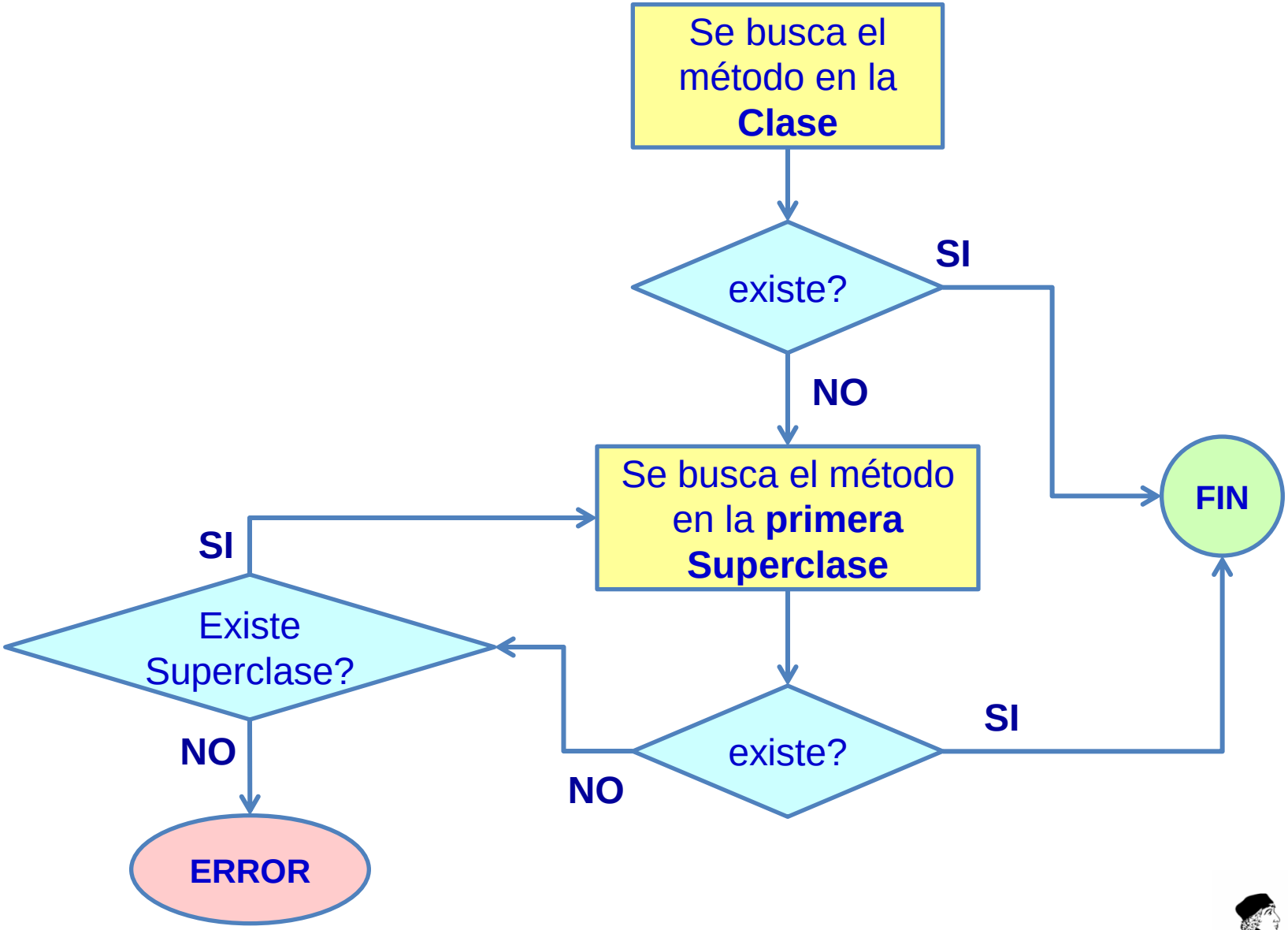
ÍNDICE

Ejecución en los pases de mensajes con herencia

Ejecución de un método invocado por una clase derivada:

1. Se busca el método en la propia clase derivada:
    - Si lo encuentra, **lo ejecuta**.
    - En otro caso, pasa al **punto 2**.
2. Busca el método en su clase base.
    - Si lo encuentra, **lo ejecuta**.
    - En otro caso, pasa al **punto 3**.
3. Si existe un nivel más en la jerarquía de clases, repite el **paso 2**. con la siguiente clase base y así **sucesivamente hasta que encuentre el método o se produzca un error**.

Ejecución en los pases de mensajes con herencia



Ejecución en los pases de mensajes con herencia

```
class Persona {
    private:
        string nif;
        int edad;
        string nombre, apellidos;
    public:
        Persona():nif(""), edad(0), nombre(""), apellidos(""){}
        Persona(string const &identif, int aa, string const &nom, string const &apel):
                nif(identif), edad(aa), nombre(nom), apellidos(apel) {}
        ...
        void mostrar() const;
        void okMatricula() const;
};


class Estudiante : public Persona
{
    private:
        string curso;
    public:
        Estudiante (string const &id, int a, string const &nom, string const &ape,
                    string const &cur):Persona(id, a, nom, ape), curso(cur) {}
        void mostrar() const;
};
```

Ejecución en los pases de mensajes con herencia

```
    cout << "\n\n\tEJEMPLO NUM. 3" << endl;
    cout << "\t==============" << endl << endl;

    Estudiante estud ("123456789S", 20, "Eva","Sanz", "Tercero");

    estud.okMatricula ();  // Se ejecuta el método de la clase Persona
//  estud.Beca ();  // Error, el método no existe en ninguna de las clases
```

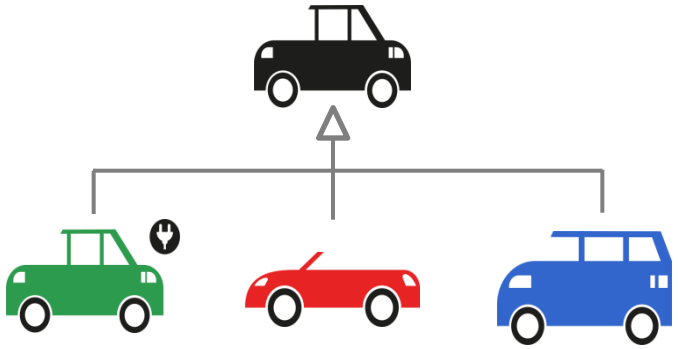
Ejecución en los pases de mensajes con herencia

```
    cout << "\n\n\tEJEMPLO NUM. 3" << endl;
    cout << "\t==============" << endl << endl;

    Estudiante estud ("123456789S", 20, "Eva","Sanz", "Tercero");

    estud.okMatricula ();  // Se ejecuta el método de la clase Persona
//  estud.Beca ();  // Error, el método no existe en ninguna de las clases
```

```
El nombre del alumno es: Eva Sanz

Su NIF es: 123456789S y su EDAD: 20
```

UNIVERSIDAD NEBRIJA

T5. Herencia



5.1. Herencia y Composición

5.2. Jerarquía de clases

5.3. Sintaxis en C++

5.4. Clase base y clase derivada

5.5. Control de acceso en la herencia

5.6. Miembros que no se hereden automáticamente

5.7. Redefinición de métodos

5.8. Ejecución en los pases de mensajes con herencia

5.9. Herencia múltiple

ÍNDICE

HERENCIA MÚLTIPLE:

Cuando una clase hereda los atributos y métodos de **varias clases simultáneamente**.

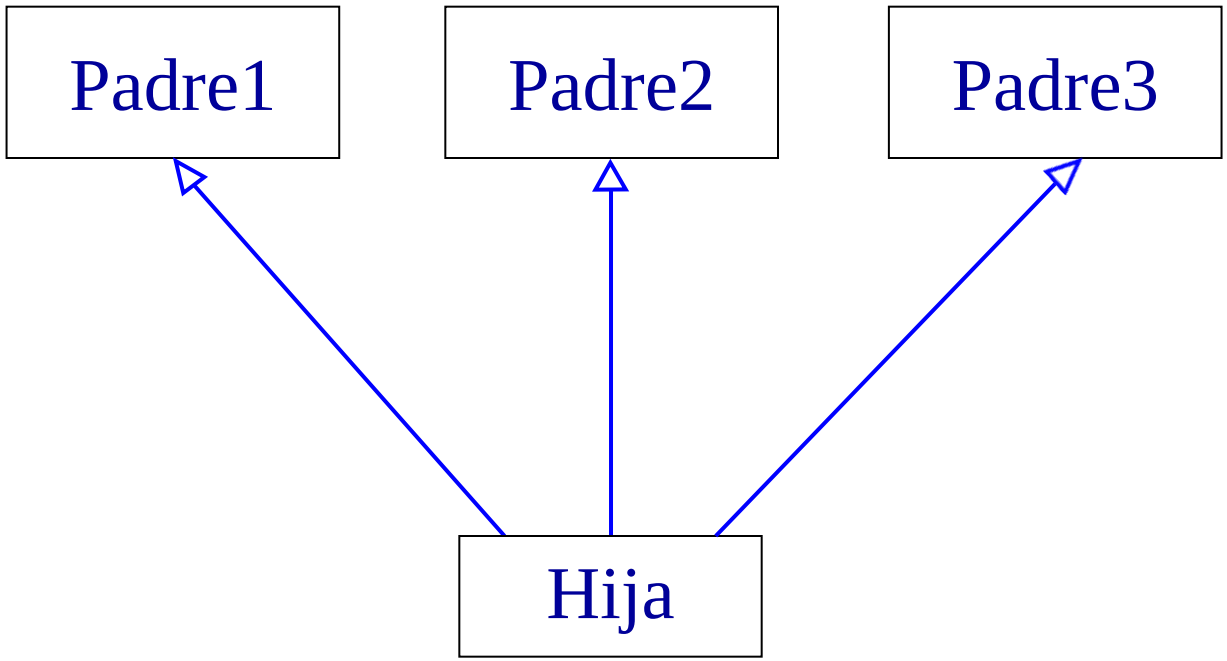
Se indica en la cabecera de la declaración de la clase derivada mediante una lista de las clases de las que hereda.

Sintaxis:

class <clase_derivada> : [public | private] <clase_base1>

[,[public|private] <clase_base2>] {};

Herencia múltiple



Herencia múltiple

```
// CLASES BASES
class padre1
{   private:
        int atrib1;
    public:
        padre1():atrib1(0){}
        padre1(int valor):atrib1(valor) {}
        void mostrar1() const;
};

class padre2
{   private:
        double atrib2;
    public:
        padre2():atrib2(0.0){}
        padre2(double valor):atrib2(valor) {}
        void mostrar2() const;
};

class padre3
{   private:
        char atrib3;
    public:
        padre3():atrib3(' '){}
        padre3(char valor):atrib3(valor) {}
        void mostrar3() const;
};
```

Herencia múltiple

```
// CLASE DERIVADA
class hija : public  padre1,  public  padre2,  public  padre3
{   private:
        int atrib1;
    public:
        hija():atrib1(0){}
        hija(int val1, double val2, char val3, int val4):
            padre1(val1), padre2(val2), padre3(val3), atrib1 (val4){}

        void mostrar4() const;
};
```

Herencia múltiple

```
#include "Clases.h"

using namespace std;

//---------------CLASES BASE---------------------

void padre1::mostrar1() const {
    cout << "\n\n\tATRIBUTO 1: "<< atrib1 << endl; }

void padre2::mostrar2() const {
    cout << "\n\n\tATRIBUTO 2: " << atrib2 << endl; }

void padre3::mostrar3() const {
    cout << "\n\n\tATRIBUTO 3: "<< atrib3 << endl; }

//---------------CLASE DERIVADA---------------------

void hija::mostrar4() const {

    mostrar1();
    mostrar2();
    mostrar3();
    cout << "\n\n\tATRIBUTO DE LA CLASE DERIVADA: "<< atrib1 << endl;
}
```

Herencia múltiple

```cpp
#include <iostream>
#include <cstring>
#include <memory>
#include "Clases.h"

using namespace std;

int main ()  {
    hija h1(5, 3.8, 'm', 100);

    cout << "\n\n\tMOSTRAR DATOS DE CADA CLASE BASE: " << endl;
    h1.mostrar1();
    h1.mostrar2();
    h1.mostrar3();

    cout << "\n\n\tMOSTRAR DATOS DE LA CLASE DERIVADA: " << endl;
    h1.mostrar4();

    cout << "\n\n\tMOSTRAR DATOS DE CADA CLASE BASE DESDE PUNTEROS: " << endl;
    shared_ptr<padre1> p1{&h1};
    p1->mostrar1();
    shared_ptr<padre2> p2{&h1};
    p2->mostrar2();
    shared_ptr<padre3> p3{&h1};
    p3->mostrar3();

    cout << "\n\n\t";
    return 0;
}
```

Herencia múltiple

```cpp
#include <iostream>
#include <cstring>
#include <memory>
#include "Clases.h"

using namespace std;

int main ()  {
    hija h1(5, 3.8, 'm', 100);

    cout << "\n\n\tMOSTRAR DATOS DE CADA CLASE BASE: " << endl;
    h1.mostrar1();
    h1.mostrar2();
    h1.mostrar3();

    cout << "\n\n\tMOSTRAR DATOS DE LA CLASE DERIVADA: " << endl;
    h1.mostrar4();

    cout << "\n\n\tMOS                                          endl;
    shared_ptr<padre1>
    p1->mostrar1();
    shared_ptr<padre2>
    p2->mostrar2();
    shared_ptr<padre3>
    p3->mostrar3();

    cout << "\n\n\t";
    return 0;
}
```

```
MOSTRAR DATOS DE CADA CLASE BASE:


ATRIBUTO 1: 5


ATRIBUTO 2: 3.8


ATRIBUTO 3: m
```

UNIVERSIDAD NEBRIJA

Herencia múltiple

```
#include <iostream>
#include <cstring>
#include <memory>
#include "Clases.h"

using namespace std;

int main ()  {
    hija h1(5, 3.8, 'm', 100);

    cout << "\n\n\tMOSTRAR DATOS DE CADA CLASE BASE: " << endl;
    h1.mostrar1();
    h1.mostrar2();
    h1.mostrar3();

    cout << "\n\n\tMOSTRAR DATOS DE LA CLASE DERIVADA: " << endl;
    h1.mostrar4();

    cout << "\n\n\tMOS                                  PUNTEROS: " << endl;
    shared_ptr<padre1
    p1->mostrar1();
    shared_ptr<padre2
    p2->mostrar2();
    shared_ptr<padre3
    p3->mostrar3();

    cout << "\n\n\t";
    return 0;
}
```

```
MOSTRAR DATOS DE LA CLASE DERIVADA:

ATRIBUTO 1: 5

ATRIBUTO 2: 3.8

ATRIBUTO 3: m

ATRIBUTO DE LA CLASE DERIVADA: 100
```

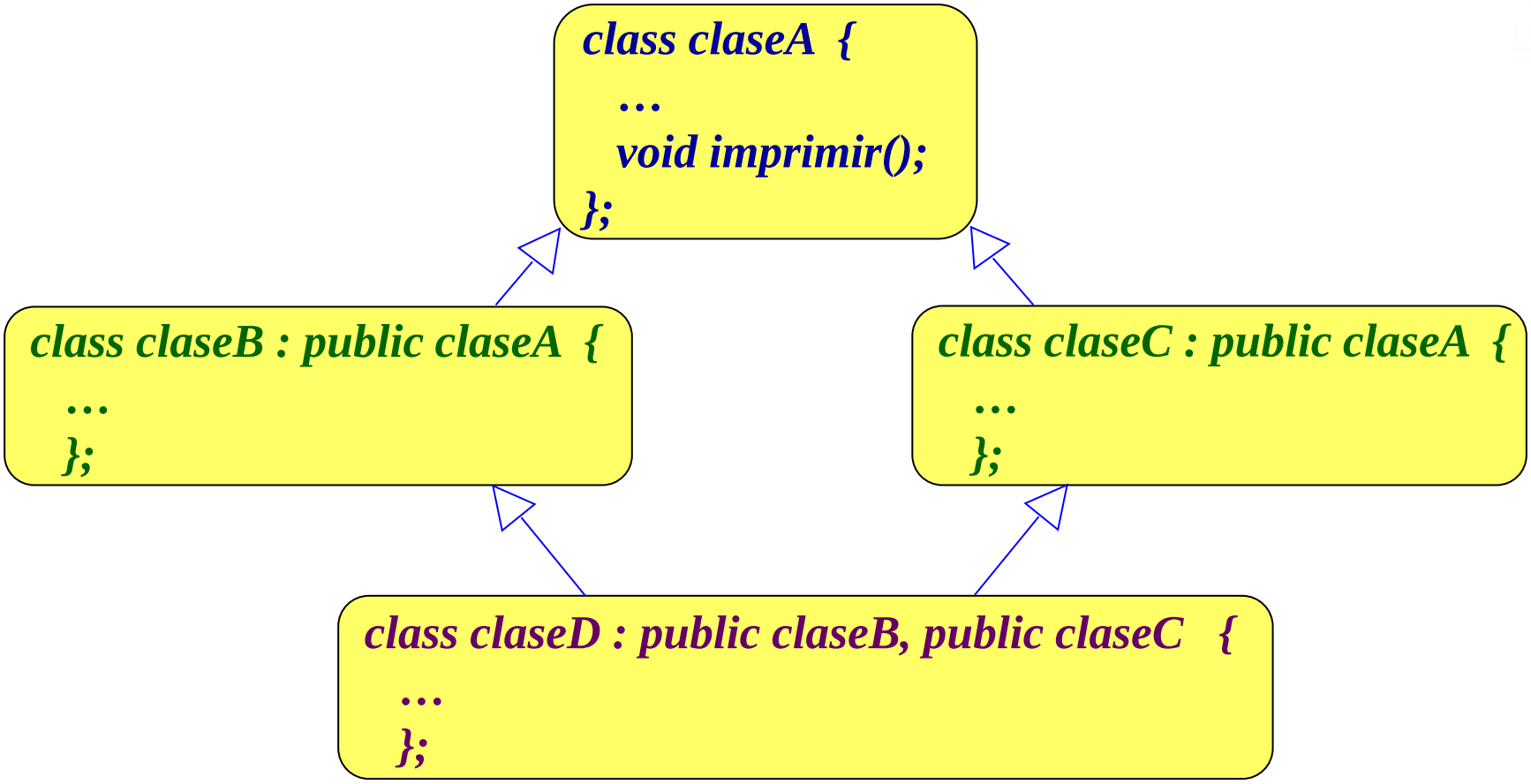
UNIVERSIDAD NEBRIJA

Herencia múltiple

```
#include <iostream>
#include <cstring>
#include <memory>
#include "Clases.h"

using namespace std;

int main ()  {
    hija h1(5, 3.8, 'm', 100);

    cout << "\n\n\tMOSTRAR DAT
    h1.mostrar1();
    h1.mostrar2();
    h1.mostrar3();

    cout << "\n\n\tMOSTRAR DAT
    h1.mostrar4();

    cout << "\n\n\tMOSTRAR DATOS DE CADA CLASE BASE DESDE PUNTEROS: " << endl;
    shared_ptr<padre1> p1{&h1};
    p1->mostrar1();
    shared_ptr<padre2> p2{&h1};
    p2->mostrar2();
    shared_ptr<padre3> p3{&h1};
    p3->mostrar3();

    cout << "\n\n\t";
    return 0;
}
```

```
MOSTRAR DATOS DE CADA CLASE BASE DESDE PUNTEROS:

ATRIBUTO 1: 5

ATRIBUTO 2: 3.8

ATRIBUTO 3: m
```

Herencia múltiple - Clase Base Virtual




```
int main ()
{   claseD  obj1;
    obj1.imprimir();
    return 0;
}
```

Ambigüedad

El **objeto de la clase D** contiene un **objeto de la clase B** y un **objeto de la clase C**. Estos dos objetos, cada uno de ellos, contiene un objeto de la clase A.

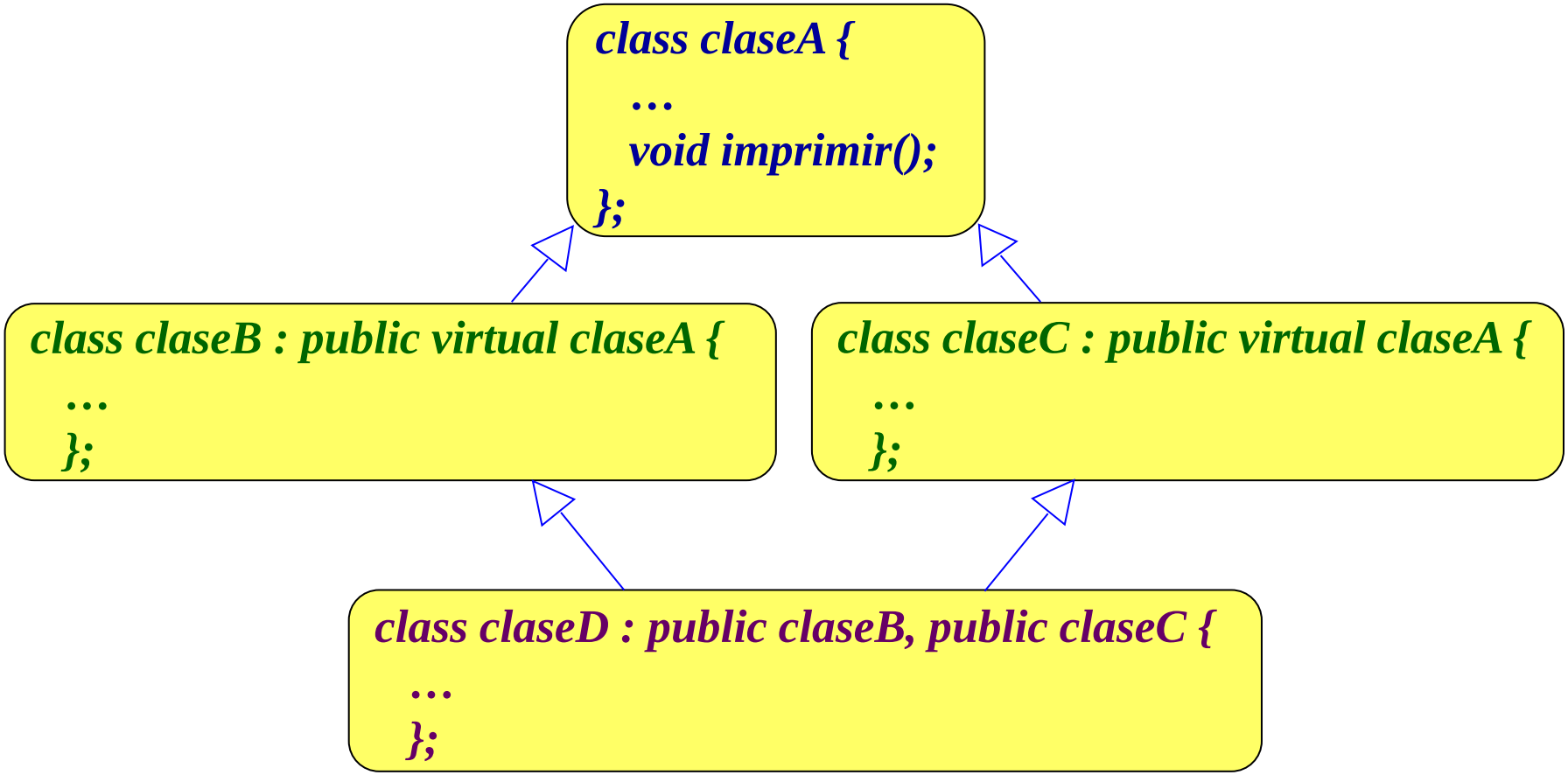
Sin embargo, lo que se buscaba era tener un objeto que tuviera el comportamiento de las clases A, B y C.

CLASE BASE VITUAL

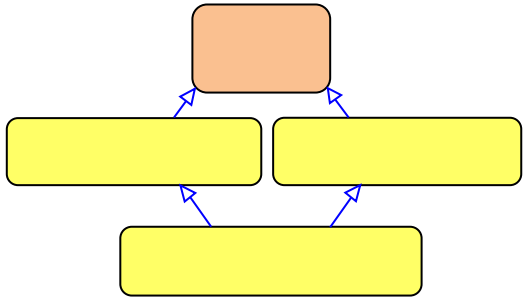
Mecanismo que permite especificar que al heredar de una clase A otras dos clases B y C, si hay otra clase D que hereda de estas dos últimas clases (B y C), no se duplicará el objeto de la clase A.

Se utiliza la palabra reservada *virtual* al realizar la herencia.

Herencia múltiple - Clase Base Virtual

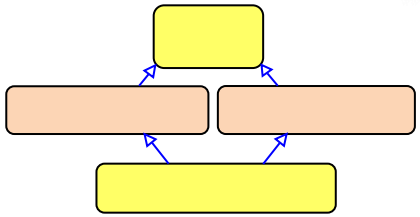


Herencia múltiple - Clase Base Virtual



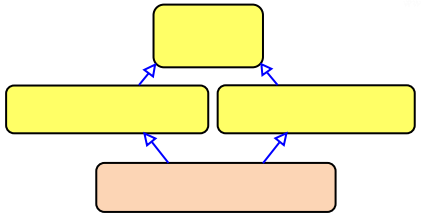
```
// CLASES BASES
class claseA
{    private:
         int atribA;
     public:
         claseA():atribA(10) {}
         void imprimir() const {
             cout << "\n\n\tClase Original" << endl; }
         void mostrarA() const {
             cout << "\n\n\tATRIBUTO A: " << atribA << endl; }
};
```

Herencia múltiple - Clase Base Virtual



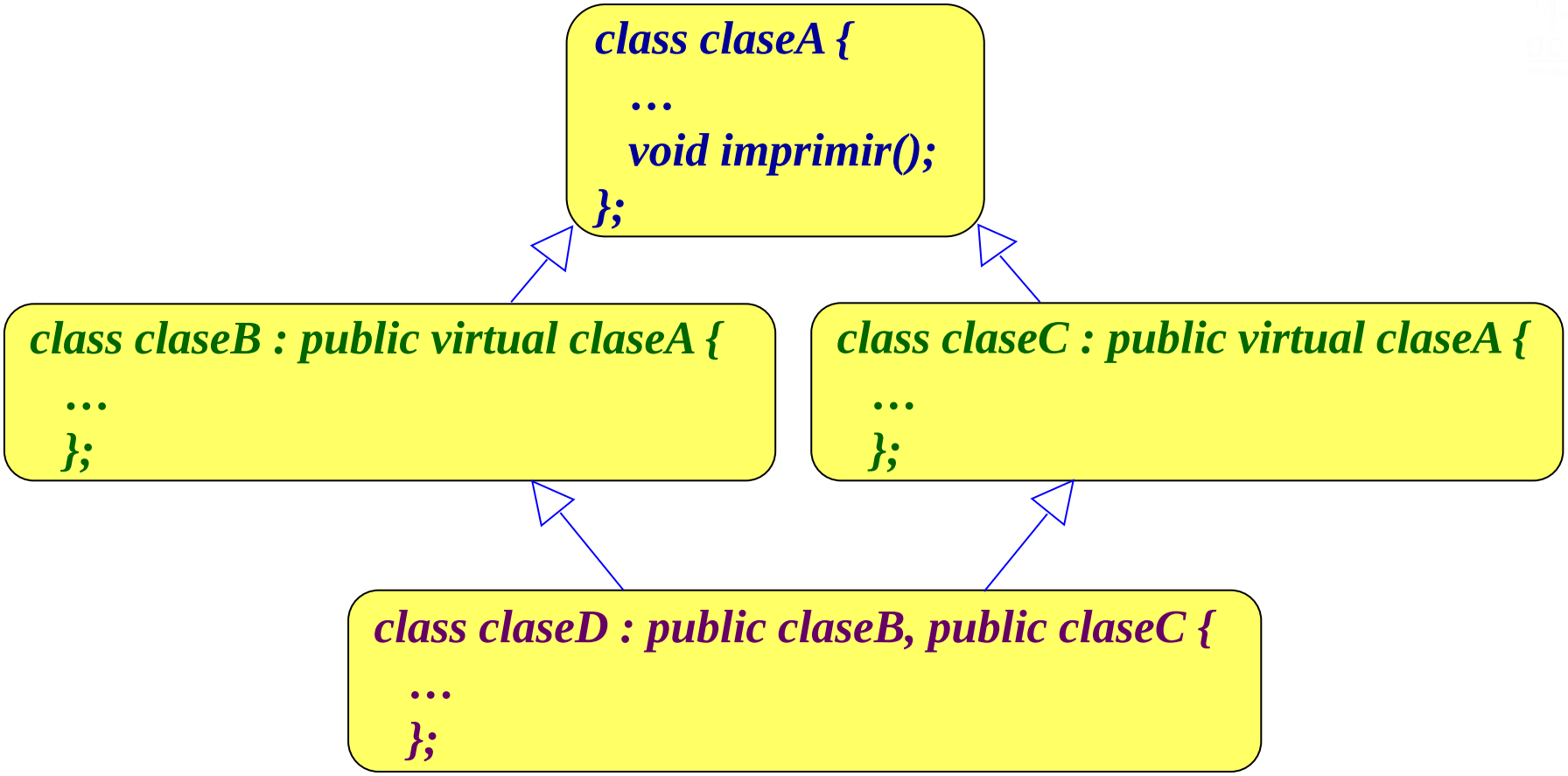
```
class claseB: public virtual claseA
{   private:
        double atribB;
    public:
        claseB():atribB(110.0) {}
        void mostrarB() const {
            cout << "\n\n\tATRIBUTO B: " << atribB << endl; }
};

class claseC: public virtual claseA
{   private:
        char atribC;
    public:
        claseC():atribC('A') {}
        void mostrarC() const {
            cout << "\n\n\tATRIBUTO C: "<< atribC << endl; }
};
```

Herencia múltiple - Clase Base Virtual



```
class claseD : public  claseB,  public  claseC
{   private:
        int atribD;
    public:
        claseD():atribD(20) {}
        void mostrarD() const {
            cout << "\n\n\tATRIBUTO D: "<< atribD << endl; }
};
```

Herencia múltiple - Clase Base Virtual




```
int main ()
{   claseD   obj1;
    obj1.imprimir();
    return 0;
}
```

El objeto interno de la clase A está **compartido** entre el objeto interno de B y de C.

Herencia múltiple - Clase Base Virtual

```cpp
#include <iostream>
#include <cstring>

#include "Clases.h"

using namespace std;

int main ()
{
    claseD   obj1;

    obj1.imprimir();

    obj1.mostrarA();
    obj1.mostrarB();
    obj1.mostrarC();
    obj1.mostrarD();

    cout << "\n\n\t";
    return 0;
}
```

```
Clase Original


ATRIBUTO A: 10


ATRIBUTO B: 110


ATRIBUTO C: A


ATRIBUTO D: 20
```