

CS1112: Programación 2

Unidad 6: Relaciones entre
clases-Herencia

Sesión de Teoría - 11

Profesor:

José Antonio Fiestas Iquira jfiestas@utec.edu.pe

Material elaborado por:

Maria Hilda Bermejo, José Fiestas, Rubén Rivas



Índice:

- Unidad 6: Relaciones entre clases
 - Herencia
 - Funciones virtuales
 - Clases abstractas
 - Herencia múltiple



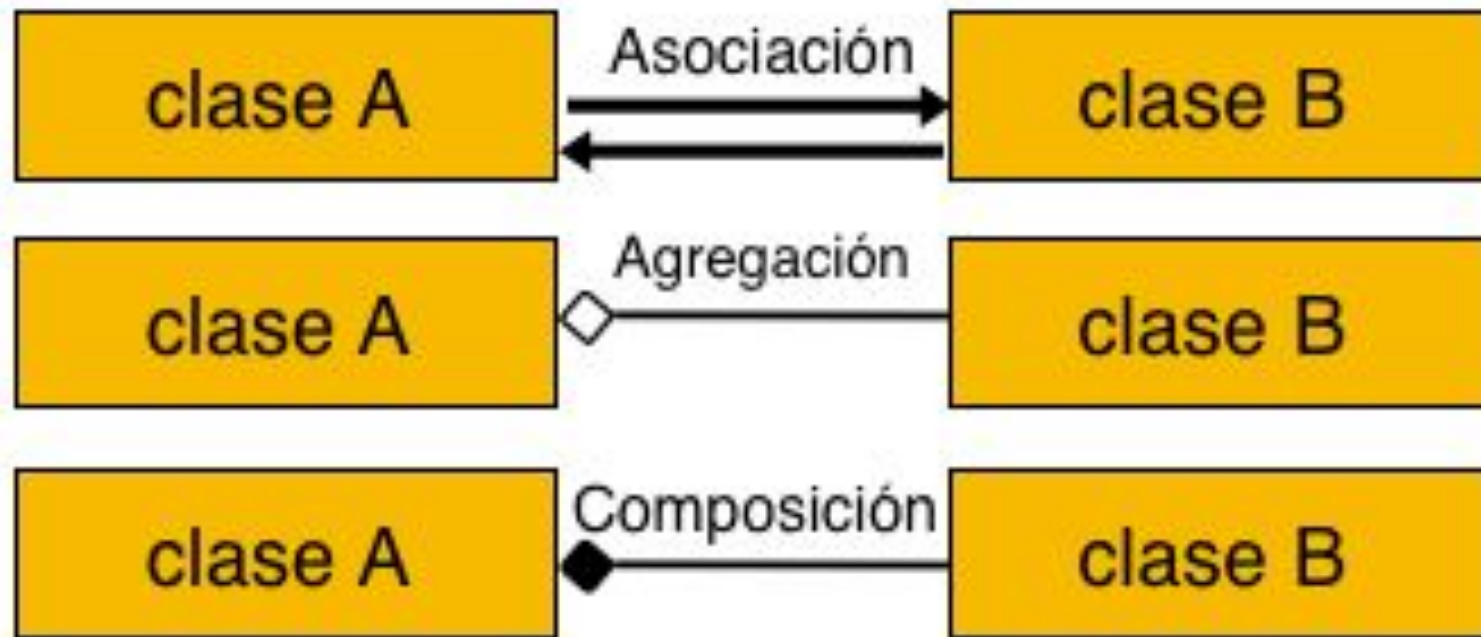
Logro de la sesión:

Al finalizar la sesión, los alumnos diseñan e implementan programas orientados a objetos utilizando relaciones de herencia y funciones virtuales

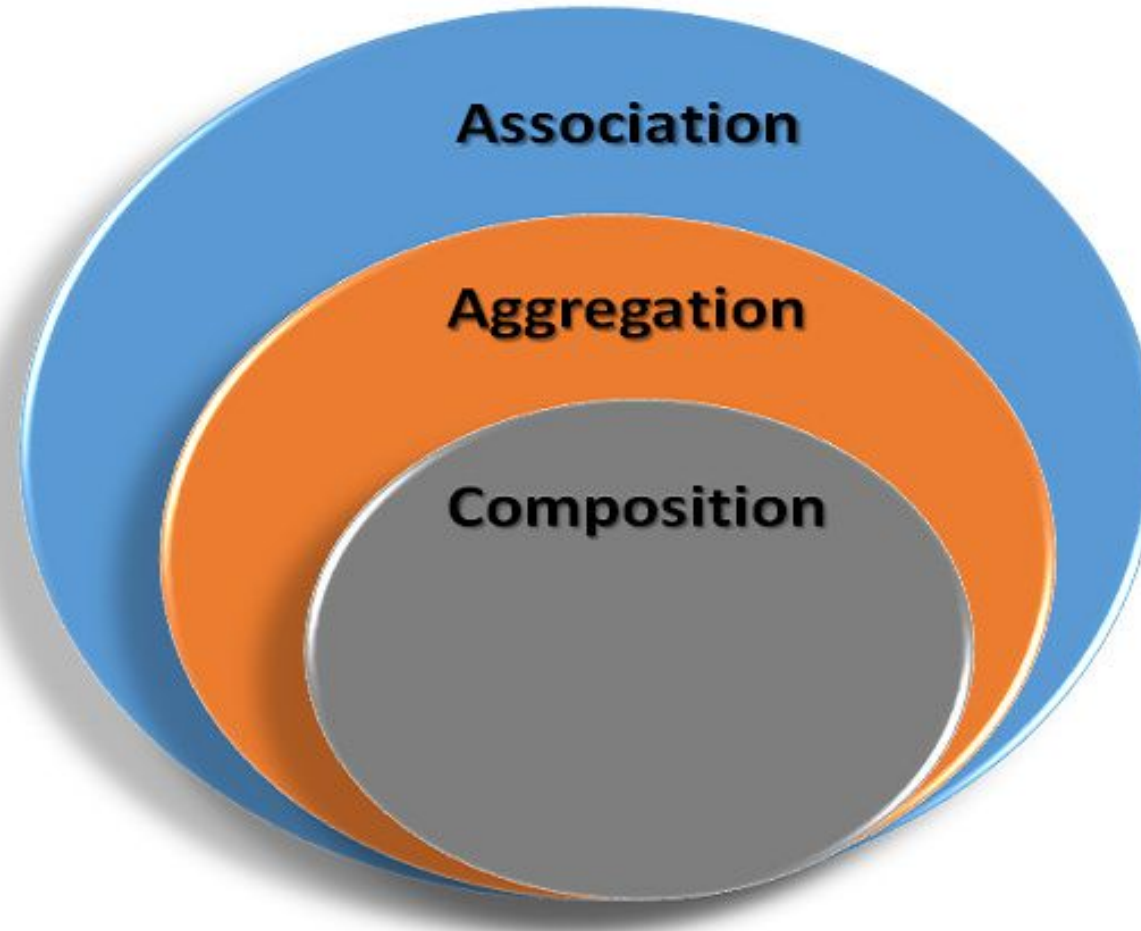
Resumen clase 10

Relaciones entre clases

Relaciones entre clases:



Relaciones entre clases:



6.1

Unidad 6: Relaciones entre clases

- Herencia
- Funciones virtuales
- Clases abstractas
- Herencia múltiple

Definición

La herencia de clases permite evitar la redundancia.

Una clase extendida también llamada: **subclase**, **clase derivada** o **clase hija** hereda de una superclase también llamada: **clase base** o **clase padre**.



Definición

- La subclase hereda todos los miembros: atributos y métodos de la superclase.
- La subclase definirá su (s) propio (s) constructor (es).
- La subclase puede definir miembros adicionales (atributos o métodos).

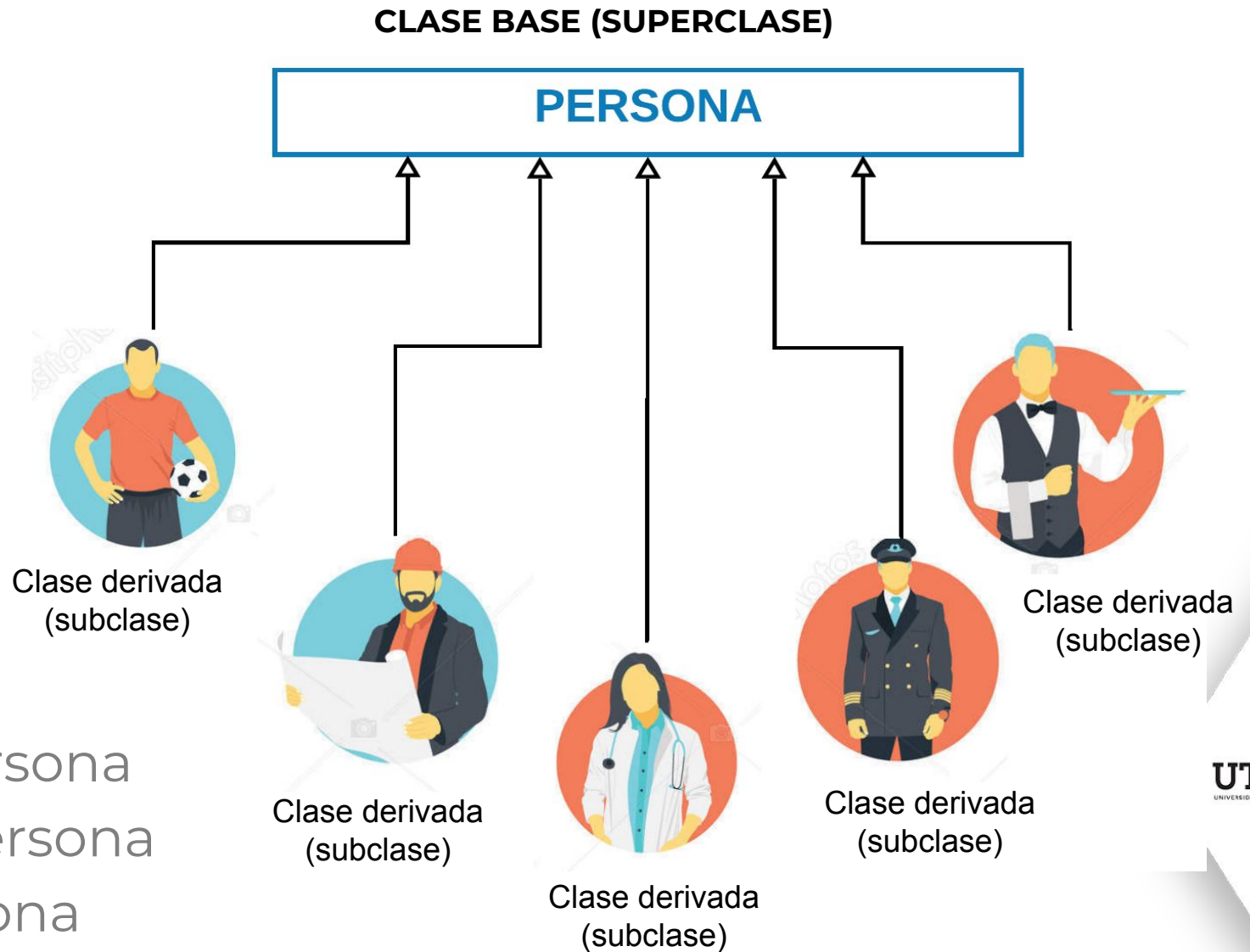
¿Se puede validar?

Se expresa como una relación de descendencia

“ ES UN ”

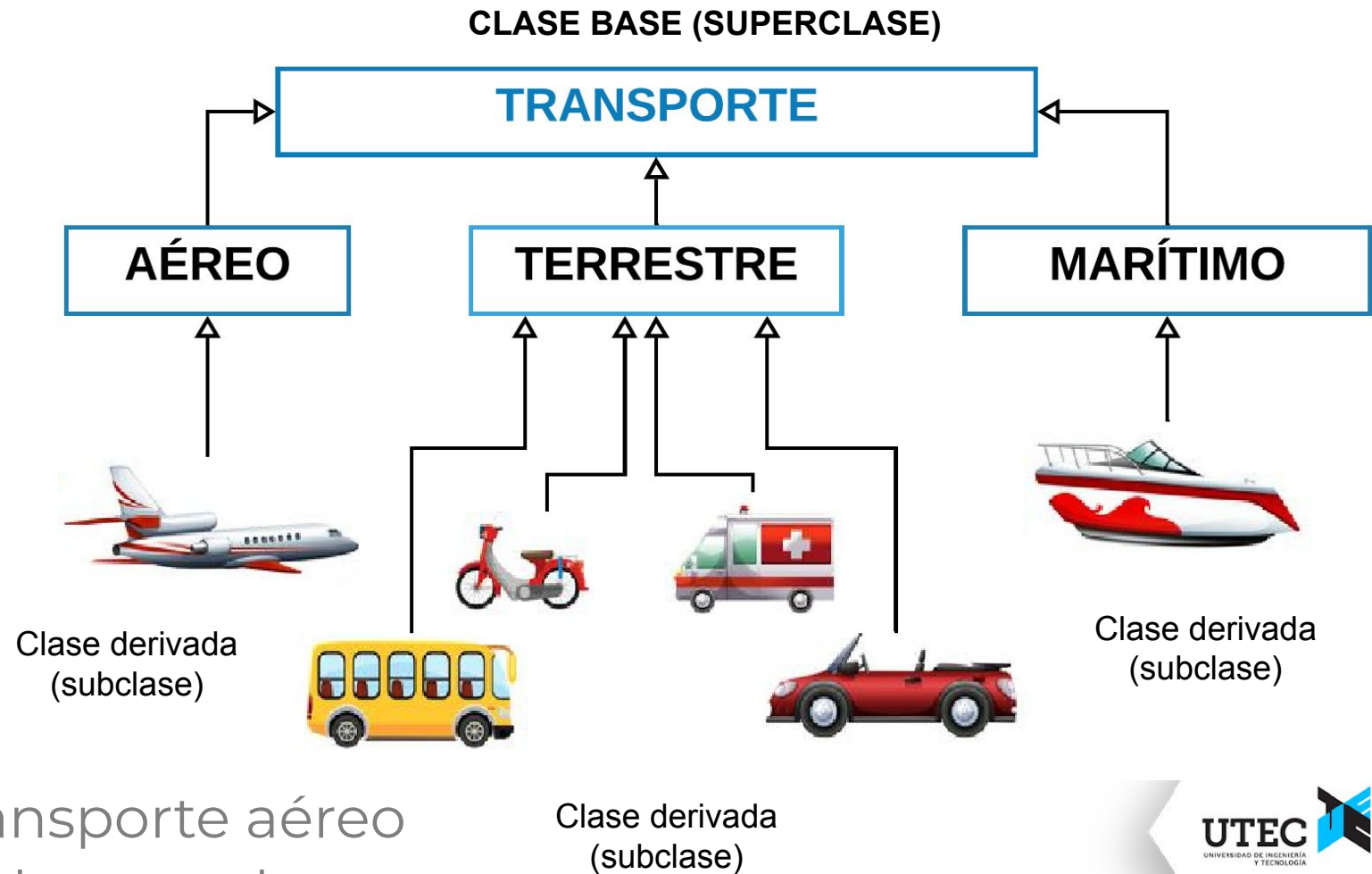
- Un pájaro es un animal
- Un gato es un mamífero
- Un pastel de manzana es un pastel
- Un coche es un vehículo

Ejemplo 1



Un futbolista **es una** persona
Un arquitecto **es una** persona
Un médico **es una** persona
....

Ejemplo 2



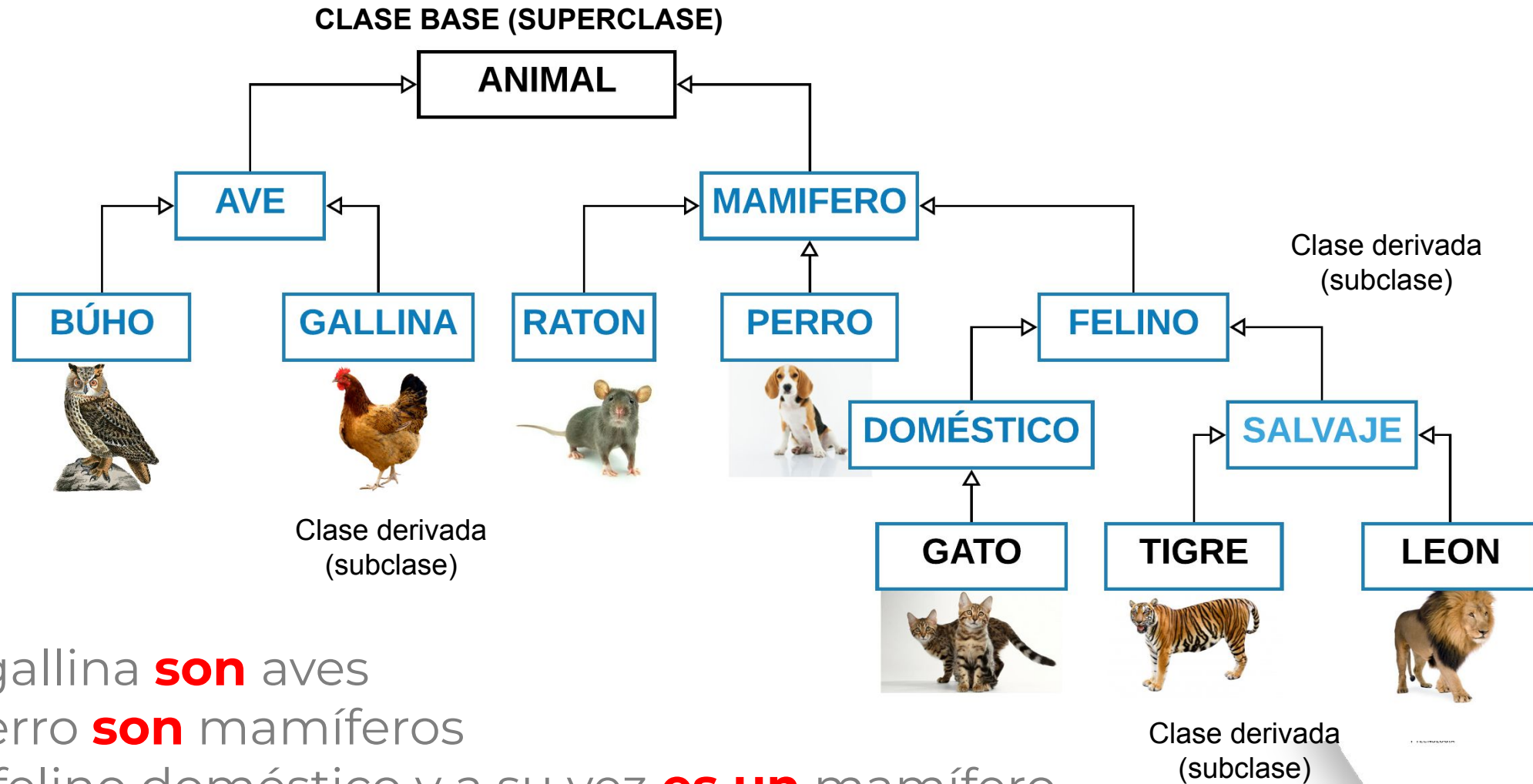
Un avión **es un** medio de transporte aéreo

Un autobús **es un** medio de transporte terrestre

Un lancha **es un** medio de transporte marítimo

Todos a su vez **son un** medio de transporte

Ejemplo 3



Un búho y una gallina **son** aves

Un ratón y un perro **son** mamíferos

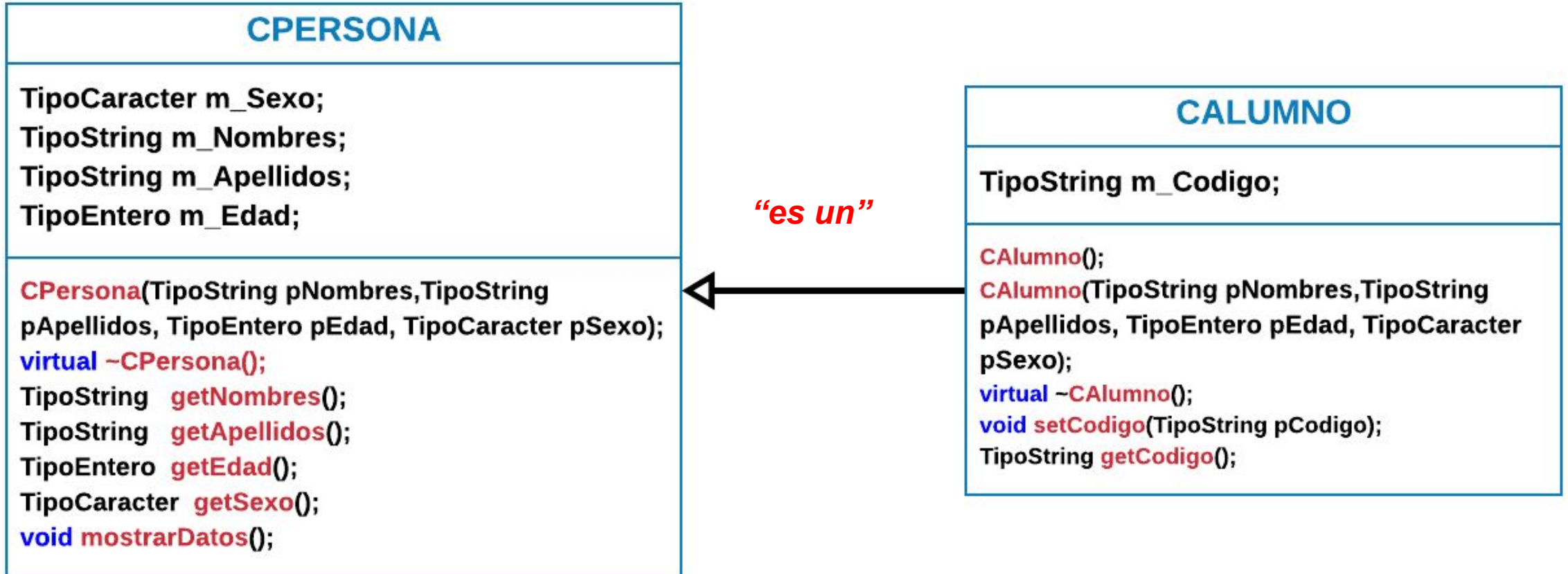
Un gato **es un** felino doméstico y a su vez **es un** mamífero

Un tigre y un león **son** felinos salvajes y a su vez **son** mamíferos

Todos a su vez **son** animales

Notación

Es una línea con un triángulo en el extremo apuntado a la clase base.



Sintaxis

```
class SubClassName : nivel-acceso-herencia BaseClass
{
    // .....
};
```

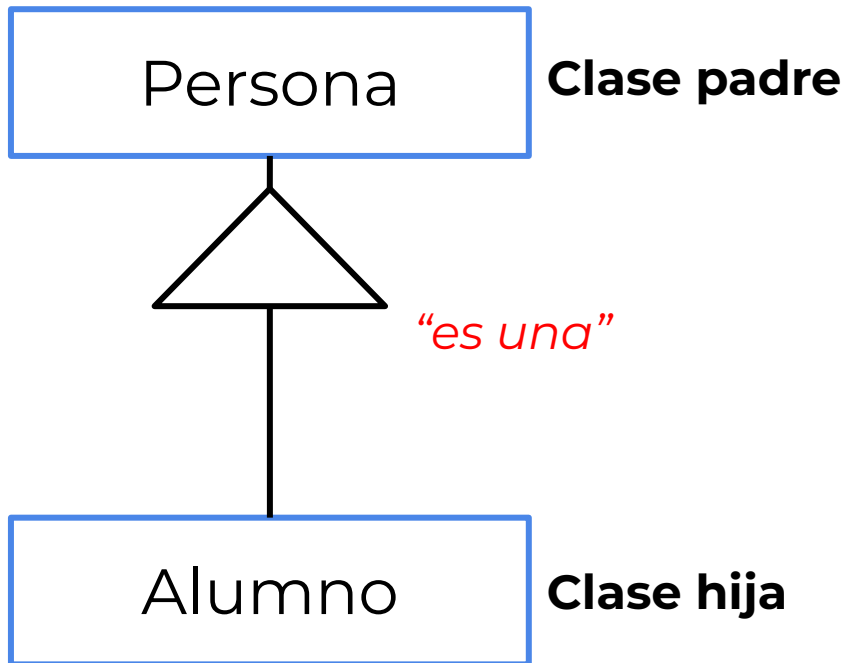
Donde: *nivel-acceso-herencia* especifica el tipo de derivación

- **private**
- **public**
- **protected**

Los miembros heredados en la subclase tienen la misma visibilidad que la superclase.

Ejemplo 1

Desarrollar un programa que defina una relación de herencia entre las clases Persona y Alumno



```
class CPersona {
    private:
        TipoCaracter m_Sexo;
    protected:
        TipoString m_Nombres;
        TipoString m_Apellidos;
        TipoNumerico m_Edad;
    public:
        CPersona(TipoString pNombres, TipoString pApellidos,
            TipoEntero pEdad, TipoCaracter pSexo);
};
```

CPersona.h

```
class CALumno : public CPersona {
    private:
        TipoString m_Codigo;
    public :
        CALumno(TipoString pNombres, TipoString pApellidos,
            TipoEntero pEdad, TipoCaracter pSexo,
            TipoString pCodigo);
};
```

CAlumno.h

Ejemplo de Herencia en C++

```
#ifndef TIPOS_H
#define TIPOS_H

#include <iostream>
using namespace std;

typedef int TipoEntero;
typedef char TipoCaracter ;
typedef string TipoString;

#endif
```

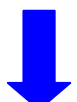
Tipos.h

Ejemplo de Herencia en C++

CPersona.h

```
#ifndef HERENCIA_PERSONA_H
#define HERENCIA_PERSONA_H
#include "Tipos.h"
class CPersona {
private :
    TipoCaracter m_Sexo;
protected:
    TipoString m_Nombres;
    TipoString m_Apellidos;
    TipoEntero m_Edad;
public:
    CPersona();
    CPersona(TipoString pNombres,
              TipoString pApellidos,
              TipoEntero pEdad,
              TipoCaracter pSexo);
```

Continúa ..



```
virtual ~CPersona();
TipoString  getNombres() const;
TipoString  getApellidos() const;
TipoEntero  getEdad() const;
TipoCaracter getSexo() const;
};
#endif
```

Ejemplo de Herencia en C++

```
#include "CPersona.h"
```

```
CPersona::CPersona(){}  
CPersona::CPersona(TipoString pNombres,  
    TipoString pApellidos, TipoEntero pEdad,  
    TipoCaracter pSexo) : m_Nombres(pNombres),  
        m_Apellidos(pApellidos),  
        m_Edad(pEdad),  
        m_Sexo(pSexo) {  
    }  
CPersona::~~CPersona(){}  
TipoString CPersona::getNombres() const  
    { return this->m_Nombres; }  
TipoString CPersona::getApellidos() const  
    { return this->m_Apellidos; }
```

CPersona.cpp



```
TipoEntero CPersona::getEdad() const  
    { return m_Edad; }  
TipoCaracter CPersona::getSexo() const  
    { return m_Sexo; }
```

¿Siempre es necesario el this?

Continúa ..



Ejemplo de Herencia en C++

CAumno.h

```
#ifndef HERENCIA_CALUMNO_H
#define HERENCIA_CALUMNO_H
#include "Tipos.h"
#include "CPersona.h"
```

public
private
protected

```
class CAumno : public CPersona {
private:
    TipoString m_Codigo;
public:
    CAumno();
    CAumno(TipoString pCodigo,
            TipoString pNombres,
            TipoString pApellidos,
            TipoEntero pEdad,
            TipoCaracter pSexo);
```

Continua ..



```
virtual ~CAumno();
TipoString getCodigo() const;
};

#endif
```


Ejemplo de Herencia en C++

CAumno.cpp

```
#include "CAumno.h"
CAumno::CAumno(){
}

CAumno::CAumno(TipoString pCodigo,
TipoString pNombres, TipoString pApellidos, TipoEntero
pEdad, TipoCaracter pSexo)
    : CPersona(pNombres, pApellidos, pEdad, pSexo) {
    m_Codigo=pCodigo;
}
CAumno::~~CAumno(){
}

TipoString CAumno::getCodigo() const {
    return m_Codigo;
}
```

Setteando valores
desde el constructor



Ejemplo de Herencia en C++

CAumno.cpp

```
#include "CAumno.h"
ostream& operator<<(ostream& os, CPersona &rPersona);
ostream& operator<<(ostream& os, CAumno &rAlumno);

int main() {
    auto *pPersona = new CPersona("Ricardo", "Paredes",
                                    34, 'M');
    auto *pAlumno= new CAumno("201812345", "Pedro",
                              "Flores", 18, 'M');

    cout << *pPersona << endl;
    cout << *pAlumno << endl;

    auto pErnesto = CPersona("Ernesto", "Cuadros", 45, 'M');
    auto pAlessia = CAumno("201612345", "Alessia", "Perez",
                           16, 'F');

    cout << pErnesto << endl;
    cout << pAlessia << endl;
    return 0;
}
```

```
ostream& operator<<(ostream& os, CPersona &rPersona) {
    os << "\n----- Datos de la Persona -----\n";
    os << "Nombre : " << rPersona.getNombres() << "\n";
    os << "Apellidos : " << rPersona.getApellidos() << "\n";
    os << "Edad : " << rPersona.getEdad() << "\n";
    os << "Sexo : " << rPersona.getSexo() << "\n";
    return os;
}

ostream& operator<<(ostream& os, CAumno &rAlumno) {
    CPersona &rPersona = rAlumno;
    os << rPersona;
    os << "Código : " << rAlumno.getCodigo() << "\n";
    return os;
}
```

Sobrecargando la salida de datos

Ejemplo de Herencia en C++

El programa lo pueden encontrar en repl.it

<http://bit.ly/322qdSC>

Herencia: Constructores ..1

- Cuando la **subclase** construye su instancia, primero debe **construir** un **objeto** de la **superclase**, del cual heredó.
- Para inicializar los miembros heredados, el **constructor** de la **subclase** invoca al **constructor** de la **superclase**, que es público, en la lista de inicializadores de miembros.
- Es necesario utilizar la lista de inicializadores (: Persona (edad)...) para invocar al **constructor** de la **superclase** Persona para inicializar la **superclase**, antes de inicializar la **subclase**. Los atributos del objeto sólo se pueden inicializar mediante la lista de inicializadores de los atributos.

Herencia: Constructores ..2

- Si no se invocó explícitamente al **constructor** de la **superclase**, el compilador invocará implícitamente al **constructor** por defecto de la **superclase** para construir un **objeto** de **superclase**.
- Para utilizar los miembros de la **superclase**, utilice el operador de resolución de ámbito en la forma de:

SuperclassName::memberName

Por ejemplo:

CPersona :: getNombres();

CPersona :: getApellidos();

CPersona :: getEdad();

6.2

Unidad 6: Relaciones entre clases

- Herencia
- Funciones virtuales
- Clases abstractas
- Herencia múltiple

UTEC

Funciones Virtuales ..1

- El uso de la palabra clave **virtual** en una función de **superclase** hace que la función sea virtual para la **superclase**, así como a TODAS sus **subclases**.
- Si se invoca una función **virtual** con un puntero (o referencia), el programa utiliza el método definido para el tipo de objeto en lugar del tipo de puntero. Esto se denomina vinculación dinámica o vinculación tardía, en contraste con la vinculación estática durante el tiempo de compilación

Funciones Virtuales ..2

- El **Constructor** no puede ser **virtual**, porque no se hereda. La **subclase** define su propio **constructor**, que invoca al **constructor** de **superclase** para inicializar los miembros de datos heredados.
- El **Destructor** debe ser declarado **virtual**, si se va a utilizar una clase como una **superclase**, de modo que se invoque el **destructor** de objeto apropiado para liberar la memoria asignada dinámicamente en la **subclase**, si la hay.

Funciones Virtuales ..3

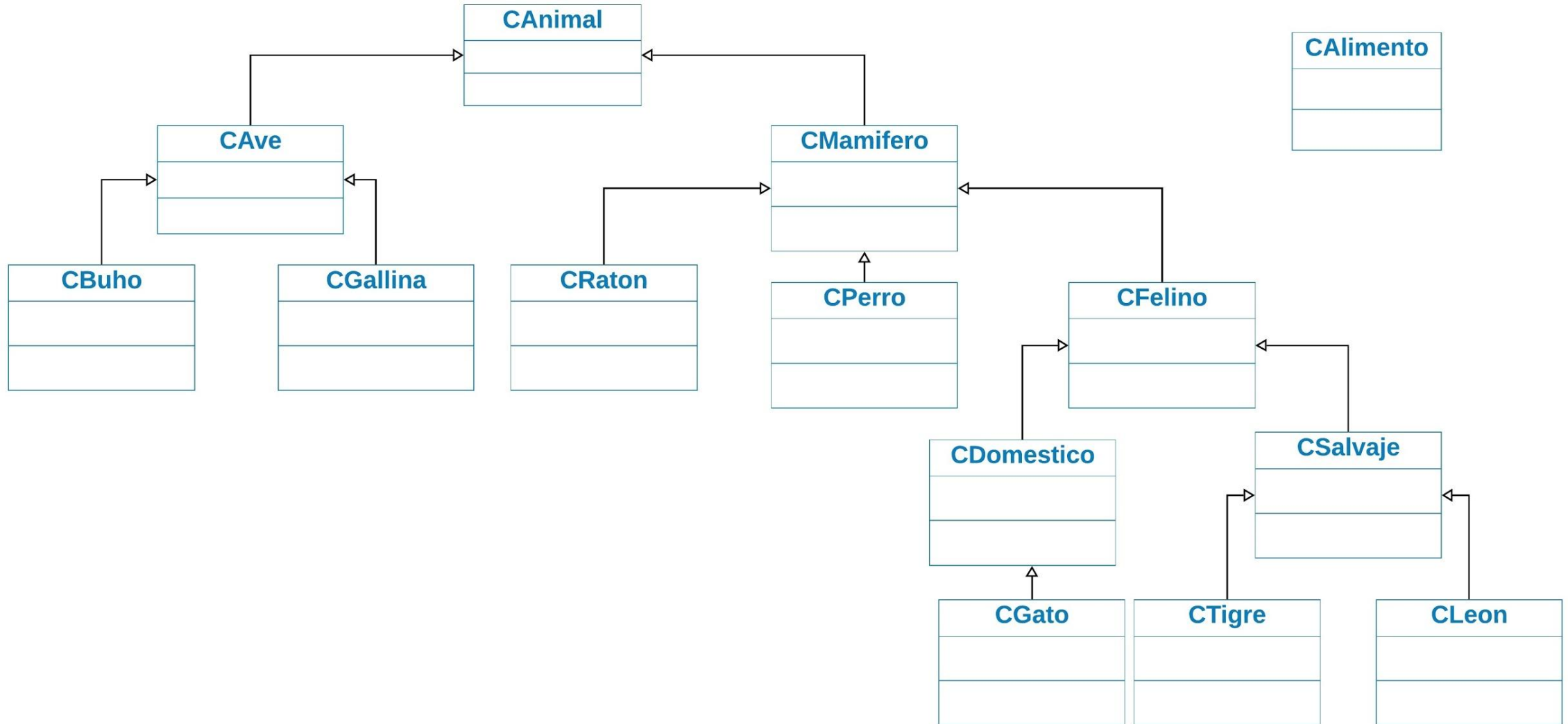
- Las funciones amigas (**friend**) no pueden ser **virtuales**, ya que las funciones amigas no son miembros de la clase y no son heredados.
- Si se **sobrescribe** (**override**) la función en la **subclase**, la función **sobrescrita** tendrá la misma lista de parámetros que la versión de la **superclase**.
- Se recomienda que las funciones a **sobrescribir** en la **subclase** sean declaradas como **virtual** en la **superclase**.

Ejemplo 2

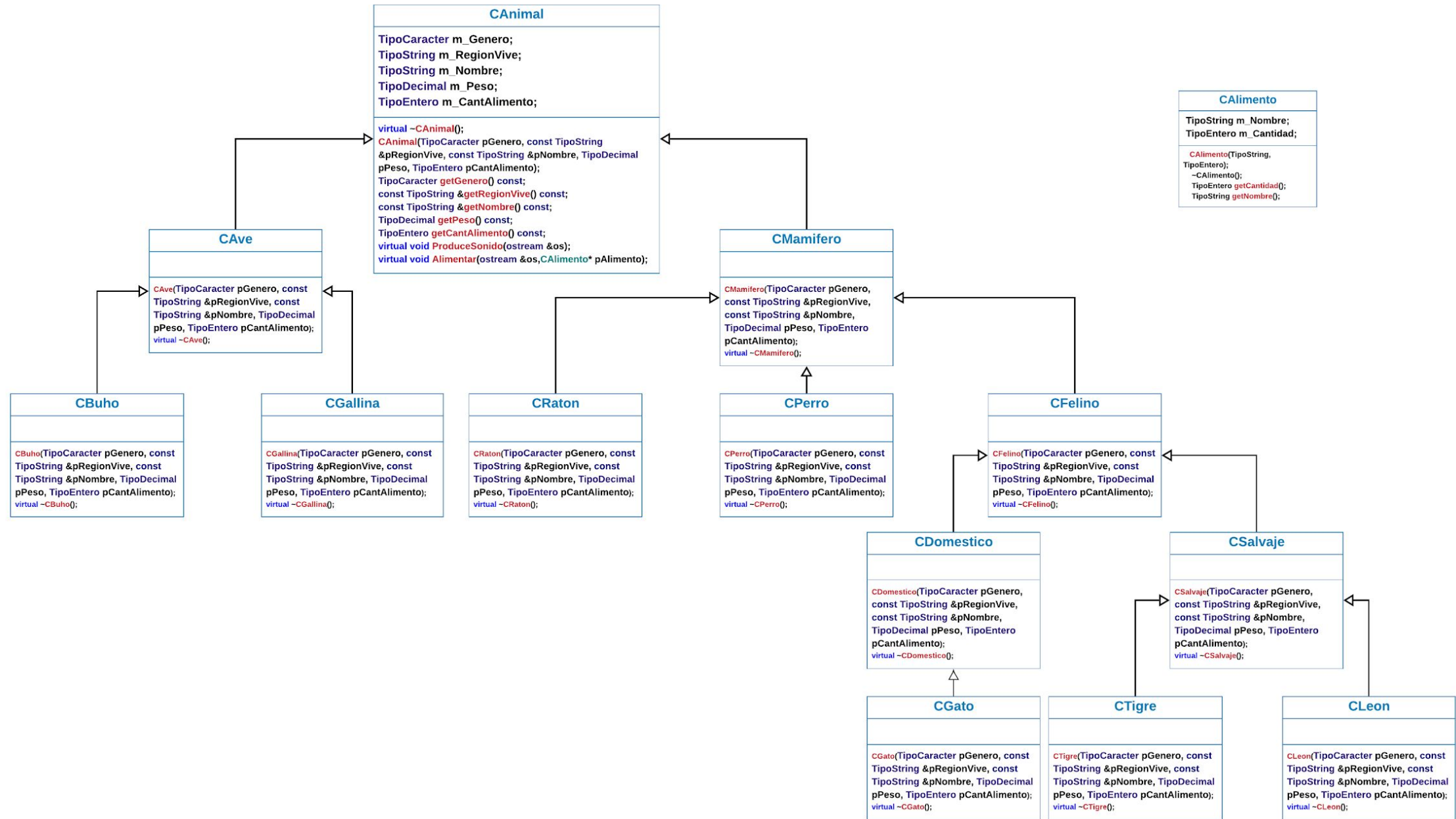
Desarrollar un programa que defina una relación de herencia entre las clases Animal, Gallina, Gato y Perro, teniendo en cuenta que todos los animales tienen la capacidad de pedir comida y producir un sonido y el peso aumentará con cada alimento que ingiera:

Animal	Sonido	Comida	Aumenta Peso
Búho	"Hoot Hoot"	Carne	0.25
Gallina	"Cluck"	De todo	0.35
Ratón	"Squeak"	Verdura y frutas	0.10
Perro	"¡Guau!"	Carne y verdura	0.40
Gato	"Miau"	Carne	0.30
Tigre	"iiiROAR !!!"	Carne	1.00

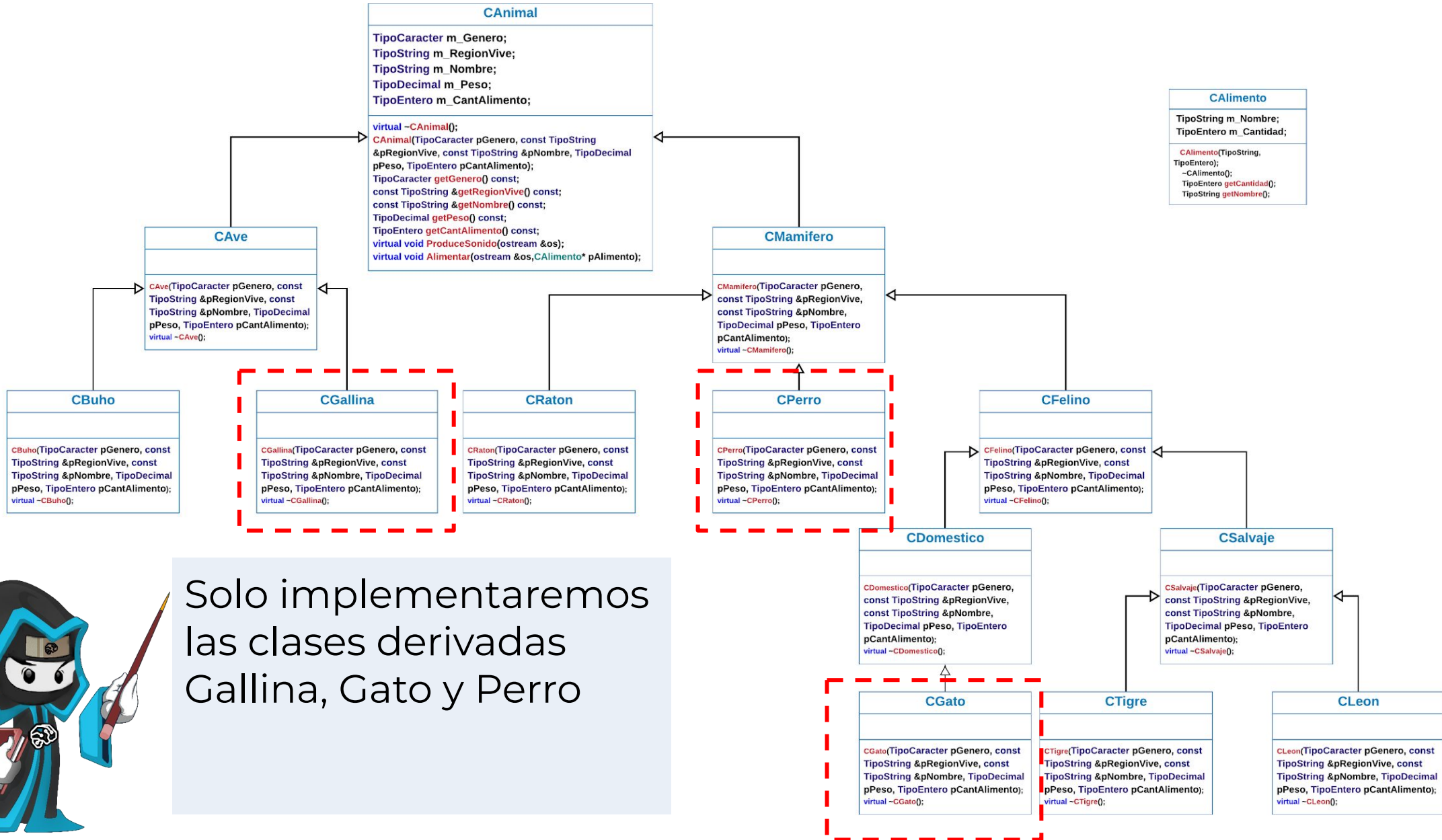
Ejemplo 2: Formalizando con la Notación de Herencia



Ejemplo 2: Formalizando con la Notación de Herencia



Ejemplo 2: Formalizando con la Notación de Herencia



Ejemplo 2

```
Felix está comiendo Carne y su peso actual es: 5.2
Boby está comiendo Verdura y su peso actual es: 5.6
Boby está comiendo Verdura y su peso actual es: 7.2
Boby está comiendo Verdura y su peso actual es: 8.8
```

```
----- Datos del Animal -----
```

```
Nombre      : Felix
Cantidad Comida: 4
Peso        : 5.2
Género      : M
Región      : Perú
Felix el gato dijo: Miauuuu
Felix ronronea
```

```
----- Datos del Animal -----
```

```
Nombre      : Boby
Cantidad Comida: 12
Peso        : 8.8
Género      : M
Región      : Perú
Boby el perro dijo: BauBau
Boby menea su cola
```

```
----- Datos del Animal -----
```

```
Nombre      : Bataraza
Cantidad Comida: 0
Peso        : 5
Género      : M
Región      : Perú
Clucclk
Bataraza la gallina dijo: Clucclk
```

Ejemplo de Herencia en C++ (Animal)

```
#ifndef HERENCIAANIMAL_TIPOS_H
#define HERENCIAANIMAL_TIPOS_H
#include <iostream>
using namespace std;
    typedef int TipoEntero;
    typedef double TipoDecimal;
    typedef char TipoCaracter;
    typedef string TipoString;
#endif //HERENCIAANIMAL_TIPOS_H
```

Tipos.h

Ejemplo de Herencia en C++ (Animal)

```
#ifndef HERENCIAANIMAL_UFUNCIONES_H
#define HERENCIAANIMAL_UFUNCIONES_H
#include "CGato.h"
#include "CPerro.h"
#include "CGallina.h"
ostream& operator<<(ostream& os, CAnimal &rAnimal) {
    os << "\n----- Datos del Animal ----- \n";
    os << "Nombre          : " << rAnimal.getNombre() << "\n";
    os << "Cantidad Comida: " << rAnimal.getCantAlimento() << "\n";
    os << "Peso           : " << rAnimal.getPeso() << "\n";
    os << "Género        : " << rAnimal.getGenero() << "\n";
    os << "Región       : " << rAnimal.getRegionVive() << "\n";
    return os;
}
ostream& operator<<(ostream& os, CGato &rGato) {
    CAnimal &rAnimal = rGato;
    os << rAnimal;
    rGato.ProduceSonido(os);
    rGato.ExpresionFelicidad(os);
    return os;
}
```

UFunciones.h

```
ostream& operator<<(ostream& os,
                    CPerro &rPerro) {
    CAnimal &rAnimal = rPerro;
    os << rAnimal;
    rPerro.ProduceSonido(os);
    rPerro.ExpresionFelicidad(os);
    return os;
}
ostream& operator<<(ostream& os,
                    CGallina &rGallina) {
    CAnimal &rAnimal = rGallina;
    os << rAnimal;
    rGallina.ProduceSonido(os);
    return os;
}

#endif //HERENCIAANIMAL_UFUNCIONES_H
```

Ejemplo de Herencia en C++ (Animal)

CAimento.h

```
#ifndef HERENCIAANIMAL_CALIMENTO_H
#define HERENCIAANIMAL_CALIMENTO_H
#include "Tipos.h"
class CAimento {
private:
    TipoString m_Nombre;
    TipoEntero m_Cantidad;
public:
    CAimento(TipoString, TipoEntero);
    virtual ~CAimento();
    TipoEntero getCantidad();
    TipoString getNombre();
};
#endif //HERENCIAANIMAL_CALIMENTO_H
```

CAimento.cpp

```
#include "CAimento.h"
CAimento::CAimento(TipoString pNombre,
TipoEntero pCantidad) {
    m_Nombre = pNombre;
    m_Cantidad = pCantidad;
}
CAimento::~CAimento() {
}
TipoEntero CAimento::getCantidad() {
    return m_Cantidad;
}
TipoString CAimento::getNombre() {
    return m_Nombre;
}
```

Ejemplo de Herencia en C++ (Animal)

```
#ifndef HERENCIAANIMAL_CANIMAL_H
#define HERENCIAANIMAL_CANIMAL_H
#include "Tipos.h"
#include "CAimento.h"
class CAnimal {
private:
    TipoCaracter m_Genero;
    TipoString m_RegionVive;
protected:
    TipoString m_Nombre;
    TipoDecimal m_Peso;
    TipoEntero m_CantAlimento;
public:
    virtual ~CAnimal();
    CAnimal(TipoCaracter pGenero, const TipoString
&pRegionVive, const TipoString &pNombre,
TipoDecimal pPeso, TipoEntero pCantAlimento);
```

Continúa ..

CAnimal.h

```
TipoCaracter getGenero() const ;
const TipoString &getRegionVive() const ;
const TipoString &getNombre() const ;
TipoDecimal getPeso() const ;
TipoEntero getCantAlimento() const ;
virtual void ProduceSonido(ostream &os);
virtual void Alimentar(ostream &os, CAimento* pAlimento);
};
#endif //HERENCIAANIMAL_CANIMAL_H
```

CAimento.h

Ejemplo de Herencia en C++ (Animal)

```
#ifndef HERENCIAANIMAL_CAVE_H
#define HERENCIAANIMAL_CAVE_H
#include "CAve.h"
class CAve: public CAnimal {
private:
public:
    virtual ~CAve();
    CAve(TipoCaracter mGenero, const TipoString
&mRegionVive, const TipoString &mNombre,
TipoDecimal mPeso,
    TipoEntero mCantAlimento);
};
#endif //HERENCIAANIMAL_CAVE_H
```

CAve.h


```
#ifndef HERENCIAANIMAL_CGALLINA_H
#define HERENCIAANIMAL_CGALLINA_H
#include "CAve.h"
class CGallina: public CAve {
private:
public:
    virtual ~CGallina();
    CGallina(TipoCaracter mGenero, const TipoString
&mRegionVive, const TipoString &mNombre,
TipoDecimal mPeso, TipoEntero mCantAlimento);
    virtual void ProduceSonido(ostream &os);
    virtual void Alimentar(ostream &os, CAimento*
pAlimento);
};
#endif //HERENCIAANIMAL_CGALLINA_H
```

CGallina.h


La clase CGallina hereda de CAve y CAve esta hereda de CAnimal.

Ejemplo de Herencia en C++ (Animal)

```
#ifndef HERENCIAANIMAL_CDOMESTICO_H
#define HERENCIAANIMAL_CDOMESTICO_H
#include "CFelino.h"
class CDomestico: public CFelino {
private:
public:
    CDomestico(TipoCaracter mGenero, const TipoString
&mRegionVive, const TipoString &mNombre,
TipoDecimal mPeso, TipoEntero mCantAlimento);
    virtual ~CDomestico();
};
#endif //HERENCIAANIMAL_CDOMESTICO_H
```



```
#ifndef HERENCIAANIMAL_CGATO_H
#define HERENCIAANIMAL_CGATO_H
#include "CDomestico.h"
class CGato: public CDomestico {
private:
    TipoEntero m_Vidas;
public:
    CGato(TipoCaracter mGenero, const TipoString
&mRegionVive, const TipoString &mNombre, TipoDecimal
mPeso,
    TipoEntero mCantAlimento);
    virtual ~CGato();
    TipoEntero getVidas();
    void ProduceSonido(ostream &os);
    void Alimentar(ostream &os, CAimento* pAlimento);
    void ExpresionFelicidad(ostream &os);
};
#endif //HERENCIAANIMAL_CGATO_H
```



La clase CGato hereda de CDomestico y CDomestico hereda de CFelino y CFelino hereda de CAnimal.

Ejemplo de Herencia en C++ (Animal)

CGato.cpp


```
#include "CGato.h"

CGato::CGato(TipoCaracter mGenero, const TipoString
&mRegionVive, const TipoString &mNombre,
TipoDecimal mPeso, TipoEntero mCantAlimento) :
CDomestico(mGenero, mRegionVive, mNombre, mPeso,
mCantAlimento) {}

CGato::~CGato() {}
TipoEntero CGato::getVidas() {
    return m_Vidas;
}

void CGato::ProduceSonido(ostream &os) {
    os<<m_Nombre<<" el gato dijo: Miauuuu"<<endl;
}

void CGato::ExpresionFelicidad(ostream &os){
    os<<m_Nombre<<" ronronea"<<endl;
}
```



```
void CGato::Alimentar(ostream &os,CAimento*
pAlimento) {
    if (pAlimento->getNombre() == "Carne") {
        float cantAlimento = 0.3 * pAlimento->getCantidad();
        m_Peso += cantAlimento;
        m_CantAlimento += pAlimento->getCantidad();
        os << m_Nombre << " está comiendo"
        << pAlimento->getNombre()
        << " y su peso actual es: " << m_Peso << endl;
    }
    else {
        os << m_Nombre << " el gato no quiere comer "
        << pAlimento->getNombre() << endl;
    }
}
```

Ejemplo de Herencia en C++ (Animal)

```
#include "CPerro.h"
```

```
CPerro::CPerro(TipoCaracter mGenero, const TipoString  
&mRegionVive, const TipoString &mNombre,  
TipoDecimal mPeso, TipoEntero mCantAlimento)  
:CMamifero(mGenero, mRegionVive, mNombre, mPeso,  
mCantAlimento) {}
```

```
CPerro::~~CPerro() {}
```

```
void CPerro::ProduceSonido(ostream &os) {  
    os<<m_Nombre<<" el perro dijo: BauBau"<<endl;  
}
```

```
void CPerro::ExpresionFelicidad(ostream &os){  
    os<<m_Nombre<<" meneaa su cola"<<endl;  
}
```

CPerro.cpp

```
void CPerro::Alimentar(ostream &os,CAimento*  
pAlimento) {  
    if (pAlimento->getNombre() == "Carne" ||  
pAlimento->getNombre() == "Verdura") {  
        float cantAlimento = 0.4 * pAlimento->getCantidad();  
        m_Peso += cantAlimento;  
        m_CantAlimento += pAlimento->getCantidad();  
        os << m_Nombre << " está comiendo "  
        << pAlimento->getNombre()  
        << " y su peso actual es: " << m_Peso << endl;  
    }  
    else {  
        os << m_Nombre << " el perro no quiere comer "  
        << pAlimento->getNombre() << endl;  
    }  
}
```

Ejemplo de Herencia en C++ (Animal)

```
#include "UFunciones.h"
```

main.cpp

```
int main() {  
    auto pCarne = new CAlimento("Carne",4);  
    auto pVerdura = new CAlimento("Verdura", 4);  
    auto pFelix = new CGato('M', "Perú", "Felix", 4, 0);  
    pFelix->Alimentar(cout, pCarne);  
    auto pBataraza = new CGallina('M',"Perú","Bataraza",5,0);  
    pBataraza->Alimentar(cout, pVerdura);  
    auto pBoby = new CPerro('M', "Perú", "Boby", 4,0);  
    pBoby->Alimentar(cout, pVerdura);  
    pBoby->Alimentar(cout, pVerdura);  
    pBoby->Alimentar(cout, pVerdura);  
    cout << *pFelix << endl;  
    cout << *pBoby << endl;  
    cout << *pBataraza << endl;  
    ...  
    return 0;  
}
```

```
Felix está comiendo Carne y su peso actual es: 5.2  
Boby está comiendo Verdura y su peso actual es: 5.6  
Boby está comiendo Verdura y su peso actual es: 7.2  
Boby está comiendo Verdura y su peso actual es: 8.8
```

```
----- Datos del Animal -----  
Nombre           : Felix  
Cantidad Comida: 4  
Peso             : 5.2  
Género           : M  
Región           : Perú  
Felix el gato dijo: Miauuuu  
Felix ronronea
```

```
----- Datos del Animal -----  
Nombre           : Boby  
Cantidad Comida: 12  
Peso             : 8.8  
Género           : M  
Región           : Perú  
Boby el perro dijo: BauBau  
Boby menea su cola
```

```
----- Datos del Animal -----  
Nombre           : Bataraza  
Cantidad Comida: 0  
Peso             : 5  
Género           : M  
Región           : Perú  
Clucclk  
Bataraza la gallina dijo: Clucclk
```

Ejemplo de Herencia en C++ (Animal)

main.cpp

```
#include <memory> //-- para usar punteros inteligentes
```

```
#include "UFunciones.h"
```

```
int main() {
```

```
    auto pCarne = new CAlimento("Carne",4);
```

```
    auto pVerdura = new CAlimento("Verdura",4);
```

```
    shared_ptr<CGallina> pBataraza = make_shared<CGallina>('M',"Perú","Bataraza",5,0);
```

```
    shared_ptr<CPerro> pBoby = make_shared<CPerro>('M',"Perú","Boby",4,0);
```

```
    shared_ptr<CGato> pFelix = make_shared<CGato>('M',"Perú","Felix",4,0);
```

```
    pFelix->Alimentar(cout,pCarne);
```

```
    pBoby->Alimentar(cout,pVerdura);
```

```
    pBoby->Alimentar(cout,pVerdura);
```

```
    pBoby->Alimentar(cout,pVerdura);
```

```
    pBataraza->Alimentar(cout,pVerdura);
```

```
    cout << *pFelix << endl;
```

```
    cout << *pBoby << endl;
```

```
    cout << *pBataraza << endl;
```

```
    return 0;
```

```
}
```



Ahora usemos punteros inteligentes.



```
Felix está comiendo Carne y su peso actual es: 5.2
Boby está comiendo Verdura y su peso actual es: 5.6
Boby está comiendo Verdura y su peso actual es: 7.2
Boby está comiendo Verdura y su peso actual es: 8.8
```

```
----- Datos del Animal -----
```

```
Nombre      : Felix
Cantidad Comida: 4
Peso       : 5.2
Género     : M
Región     : Perú
Felix el gato dijo: Miauuuu
Felix ronronea
```

```
----- Datos del Animal -----
```

```
Nombre      : Boby
Cantidad Comida: 12
Peso       : 8.8
Género     : M
Región     : Perú
Boby el perro dijo: BauBau
Boby menea su cola
```

```
----- Datos del Animal -----
```

```
Nombre      : Bataraza
Cantidad Comida: 0
Peso       : 5
Género     : M
Región     : Perú
Cluclk
Bataraza la gallina dijo: Cluclk
```

El programa lo pueden encontrar en repl.it

<http://bit.ly/2NwNyqa>

6.3

Unidad 6: Relaciones entre clases

- Herencia
- Funciones virtuales
- Clases abstractas
- Herencia múltiple

UTEC

Funciones Virtuales Puras

- Una función **virtual pura** normalmente no tiene cuerpo de implementación.
- La implementación se deja a la **subclase**.
- Una función **virtual pura** se especifica colocando:
" =0 " (denominado especificador puro) en su declaración.

Ejemplo:

//función virtual Pura, será implementada en la subclase

virtual void ProduceSonido(ostream &os)=0;

virtual void Alimentar(ostream &os,CAimento* pAlimento)=0;

Clase Abstracta

- Una clase que contiene una o más funciones **virtuales puras** se denomina clase **abstracta**. No se puede instanciar una clase **abstracta**, porque su definición puede ser incompleta.
- La clase **abstracta** debe ser una **superclase**. Para utilizar una clase **abstracta**, es necesario derivar una **subclase**, sobrescribir e implementar a todas sus funciones **virtuales puras**. Luego, se procede a crear una instancia de la **subclase** concreta.
- C++ permite la implementación de la función **virtual pura**. En este caso, **el = 0** simplemente hace la clase abstracta. Como resultado, no podrá instanciarla.

Ejemplo: Clase Abstracta

CAnimal

```
TipoString m_Nombre;  
TipoDecimal m_Peso;  
TipoEntero m_CantAlimento;
```

```
CAnimal(TipoString pNombre,TipoDecimal pPeso,TipoEntero pCantAlimento);
```

```
CAnimal();
```

```
virtual ~CAnimal();
```

```
TipoString getNombre();
```

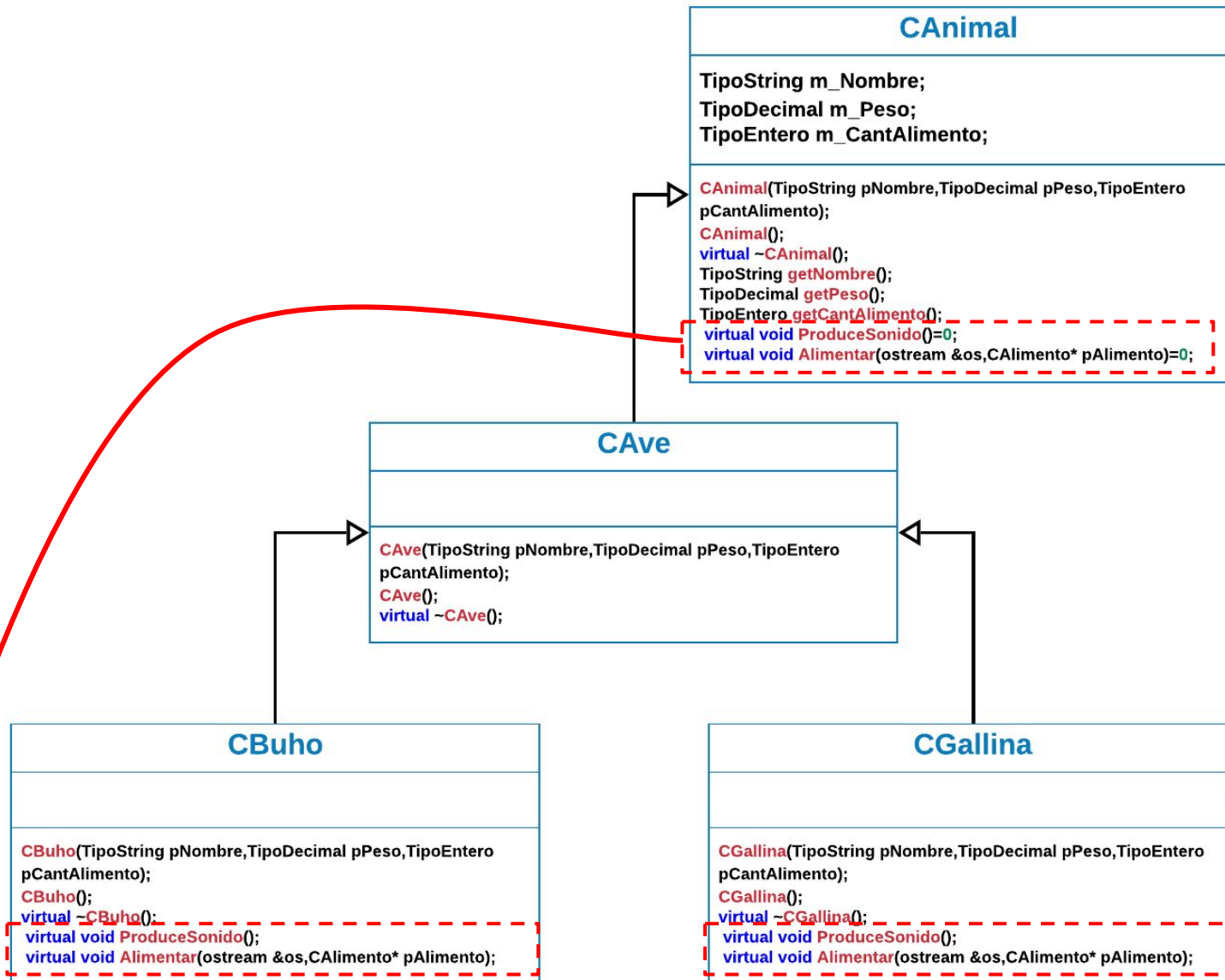
```
TipoDecimal getPeso();
```

```
TipoEntero getCantAlimento();
```

```
virtual void ProduceSonido()=0;
```

```
virtual void Alimentar(ostream &os,CAimento* pAlimento)=0;
```

Ejemplo: Clase Abstracta



La clase **CAnimal** es una clase abstracta y obliga a sus clases derivadas a implementar las **funciones virtuales**

6.4

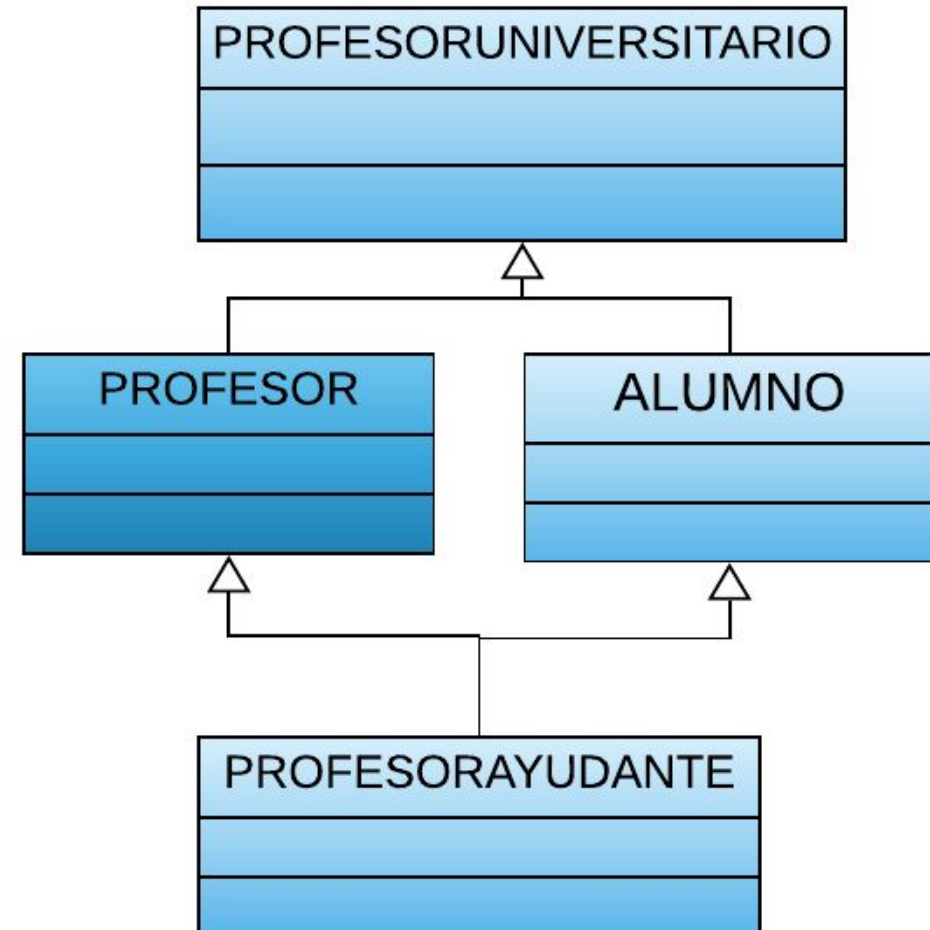
Unidad 6: Relaciones entre clases

- Herencia
- Funciones virtuales
- Clases abstractas
- Herencia múltiple

UTEC

Herencia Múltiple

- Una Clase puede derivar de **más de una clase Base**.
- Este proceso se conoce como **derivación múltiple**.
- Un objeto de una clase con herencia múltiple, heredará los datos y funciones de todas las clases base.



Herencia Múltiple: Sintaxis

```
class <clase_derivada> :  
    [public|private] <base1> , [public|private] <base2>  
{  
  
    // . . .  
};
```

Herencia Múltiple: Ambigüedad ..1

En caso exista una función con el mismo nombre en las clases base, la ambigüedad se puede solucionar de dos maneras:

- a. Referenciando a la clase Base para invocar al método.

Ejemplo:

```
// La función imprime esta implementado en ClaseA y ClaseB
class ClaseC : public ClaseA, public ClaseB {};
int main() {
    ClaseC heredero;
    // La sentencia: heredero.imprime();
    // Produce error de compilación por ambigüedad.
    heredero.ClaseA::imprime();
    imprime();
}
```

Herencia Múltiple: Ambigüedad ..2

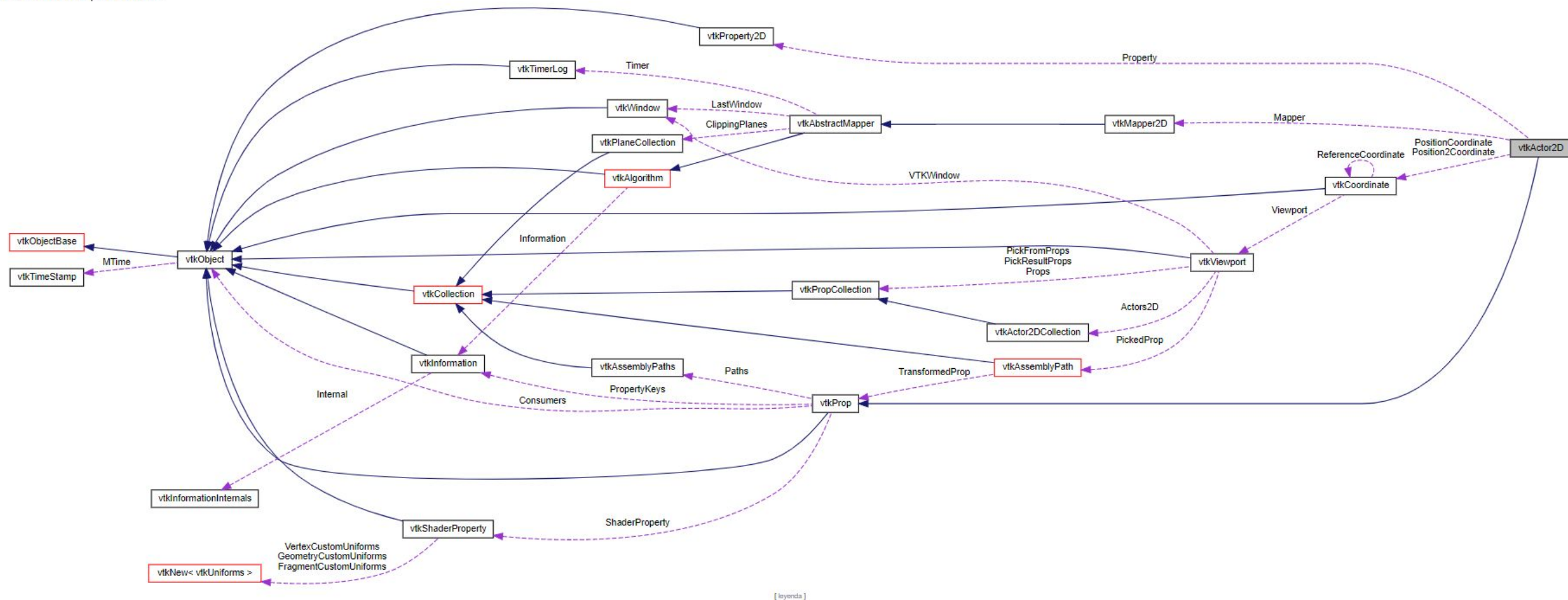
b. Sobre escribiendo la función ambigua en la clase derivada.

Ejemplo:

```
class ClaseC : public ClaseA, public ClaseB {  
    public:  
        void imprime() { cout<<"Sobreescribiendo la funcion" << endl; }  
};  
.....  
int main() {  
    ClaseC heredero;  
    heredero.imprime();  
}
```


Un Ejemplo más real sobre herencia

Diagrama de colaboración para vtkActor2D:



Fuente: <https://vtk.org/doc/nightly/html/classvtkActor2D.html>



- ✓ ¿En qué consiste la relación de Herencia?
- ✓ ¿En qué consiste una función virtual?
- ✓ ¿En qué consiste una clase abstracta ?
- ✓ ¿En qué consiste la herencia múltiple?

Resumen

En esta sesión aprendimos:

- Relaciones entre clases
 - Herencia
 - Funciones virtuales
 - Clases abstractas
 - Herencia múltiple

¡Nos vemos en la
siguiente clase!

