



Algoritmos





Logro de sesión

- Al finalizar la sesión, el estudiante **utiliza relaciones de generalización entre clases** para la construcción de programas.

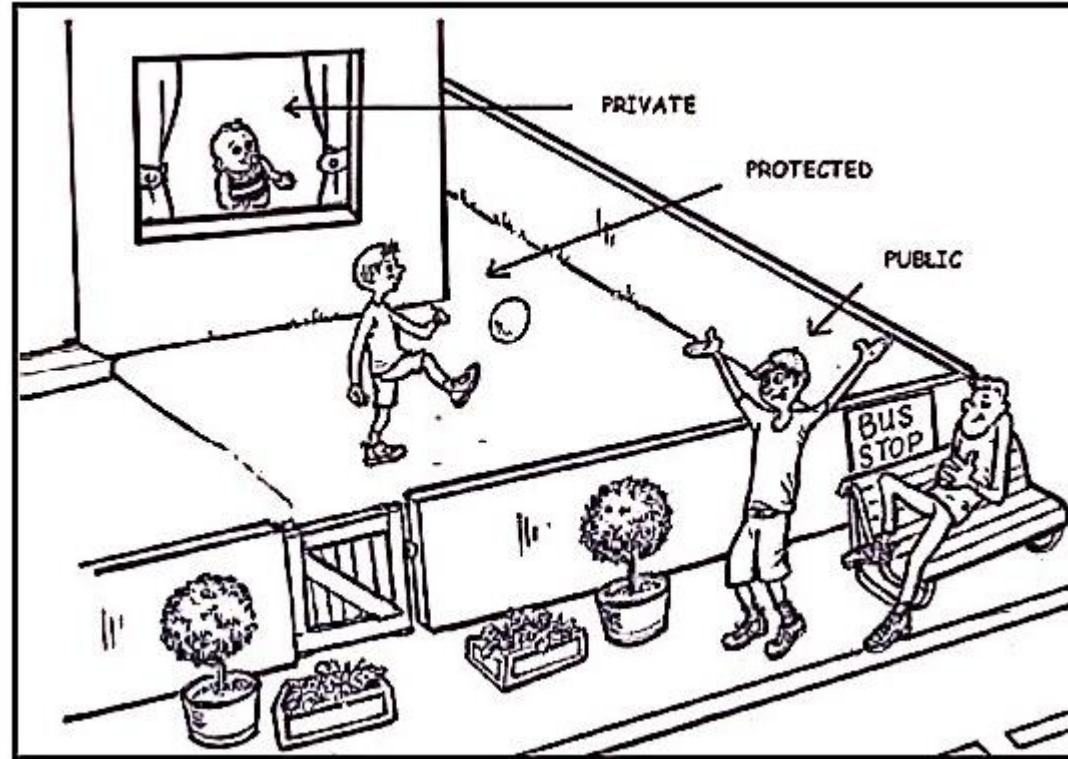


Relación de Generalización

Contenido:

- Generalización [Herencia]
 - Relaciones entre clases de generalización
 - Diagrama relaciones de generalización

¿Qué observamos?



<https://medium.com/@priyanka.nagaraj1494/access-specifiers-in-c-public-private-protected-369b578319e0>



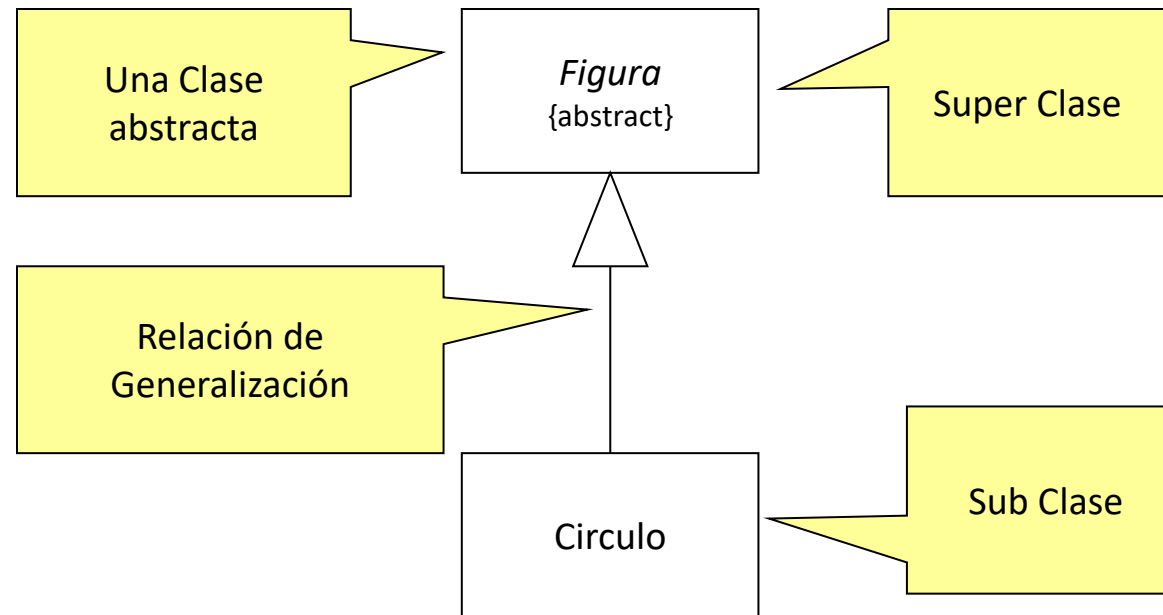
Relaciones entre clases



Generalización



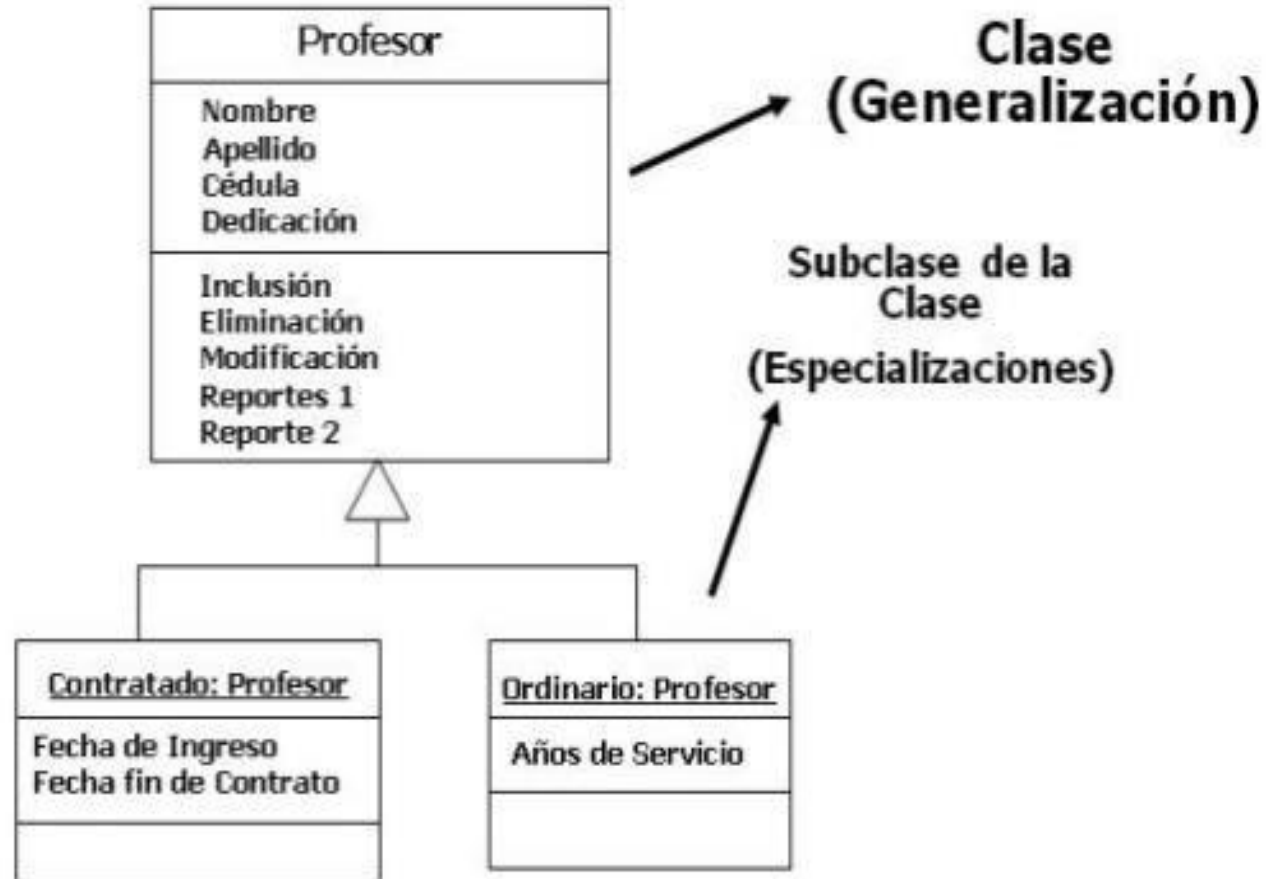
- Su relación "Es una especie de".



Generalización



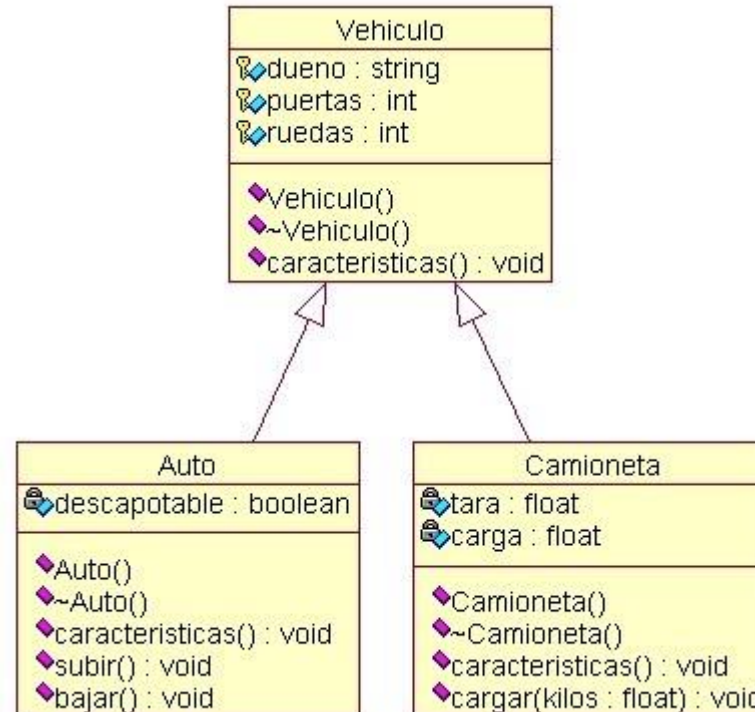
- Su relación "Es una especie de".



Generalización



- Indica que los objetos de la clase especializada (subclase) son sustituibles por objetos de la clase generalizada (super-clase).
- La generalización es una relación semántica entre clases, que determina que la interfaz de la subclase debe **incluir todas las propiedades públicas y privadas de la superclase.**





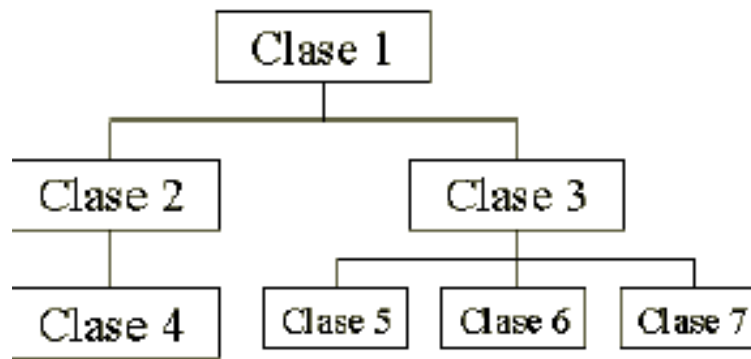
- Una subclase hereda de su super-clase:
 - Atributos
 - Operaciones
 - Relaciones
- Una subclase puede:
 - Agregar atributos y operaciones
 - Añadir relaciones
 - Refinar (sobrecargar) operaciones heredadas
- Una relación de generalización no se puede utilizar para modelar la implementación de la interfaz.

Generalización

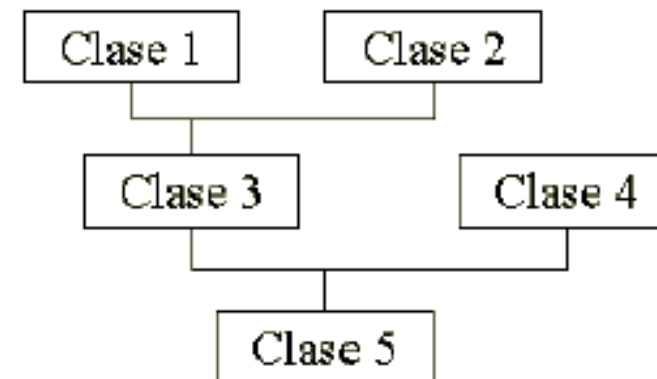


En orientación a objetos, el mecanismo que implementa la propiedad de **generalización** se denomina **herencia**.

La **herencia** permite **definir nuevas clases a partir de otras clases ya existentes**, de modo que **presentan las mismas características y comportamiento de éstas, así como otras adicionales.**



Herencia Simple: Todas las clases derivadas tienen una única clase base



Herencia Múltiple: Las clases derivadas tienen varias clases base

Realización

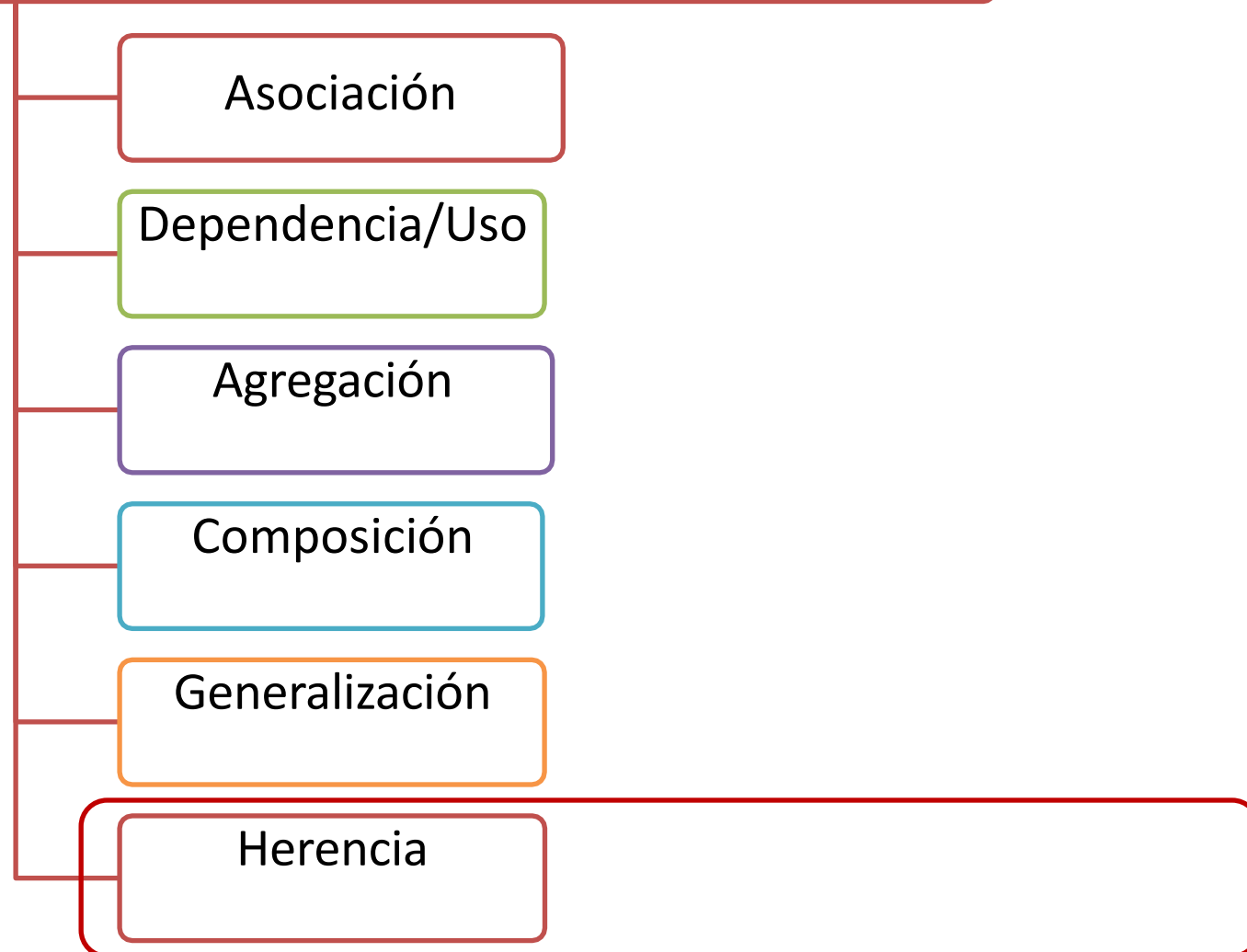


- Una relación de realización indica que una clase implementa un comportamiento especificado por otra clase (una interfaz o protocolo).
- Una interfaz puede ser realizada por muchas clases.
- Una clase puede realizar muchas interfaces.





Relaciones entre clases



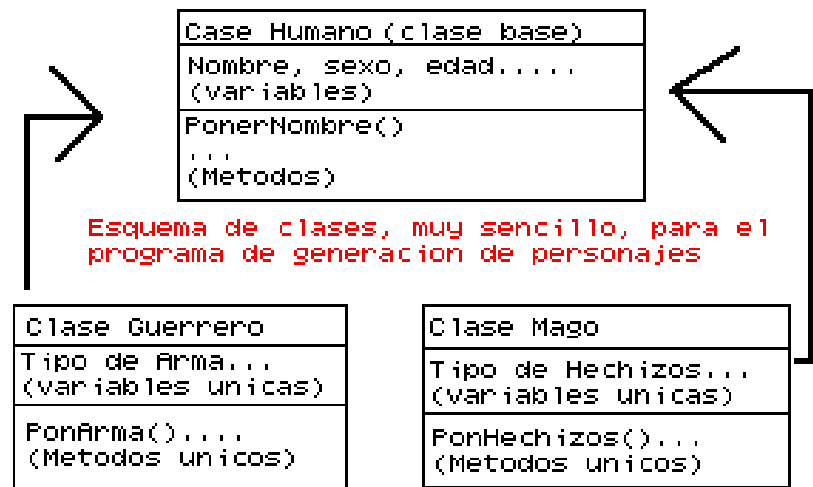
Herencia



La herencia básicamente consiste en que una clase puede heredar sus variables y métodos a varias **subclases** (la clase que hereda es llamada **clase base** o **clase padre**).

Esto significa que una **clase derivada**, aparte de los atributos y métodos propios, tiene incorporados atributos y métodos heredados de la **clase base**.

De esta manera se crea una jerarquía de herencia.



La clase Guerrero y Mago heredan de la Clase Humano

Herencia



Al crear una clase, en lugar de escribir datos miembro y funciones miembro completamente nuevos, podemos designar que la nueva clase herede los miembros de una ya existente.

La clase existente se llama **clase base**, y la clase nueva es la **clase derivada**.

Herencia



Una clase derivada representa especializado de objetos. Por lo general, una clase derivada contiene los comportamientos heredados de su clase base además de comportamientos adicionales.

Nombre de la clase derivada Especificador de acceso
Normalmente público
Herencia

Nombre de la clase base

```
class ClaseDerivada : public ClaseBase {  
    public:  
        // sección privada  
    ...  
    private:  
        // sección privada  
    ...  
};
```

Símbolo de derivación o herencia

Clases base y clases derivadas



A menudo, **un objeto de una clase es un objeto de otra clase** también. Por ejemplo, en geometría, un rectángulo *es un* cuadrilátero (así como los cuadrados, los paralelogramos y los trapezoides).

Por ende, en C++ se puede decir que la clase **Rectangulo** *hereda* de la clase **Cuadrilatero**. En este contexto, **Cuadrilatero** es una clase base y **Rectangulo** es una clase derivada.

Un rectángulo *es un* tipo específico de cuadrilátero, pero es **incorrecto decir que un cuadrilátero es un rectángulo**; el cuadrilátero podría ser un paralelogramo o cualquier otra figura.

Clases base y clases derivadas



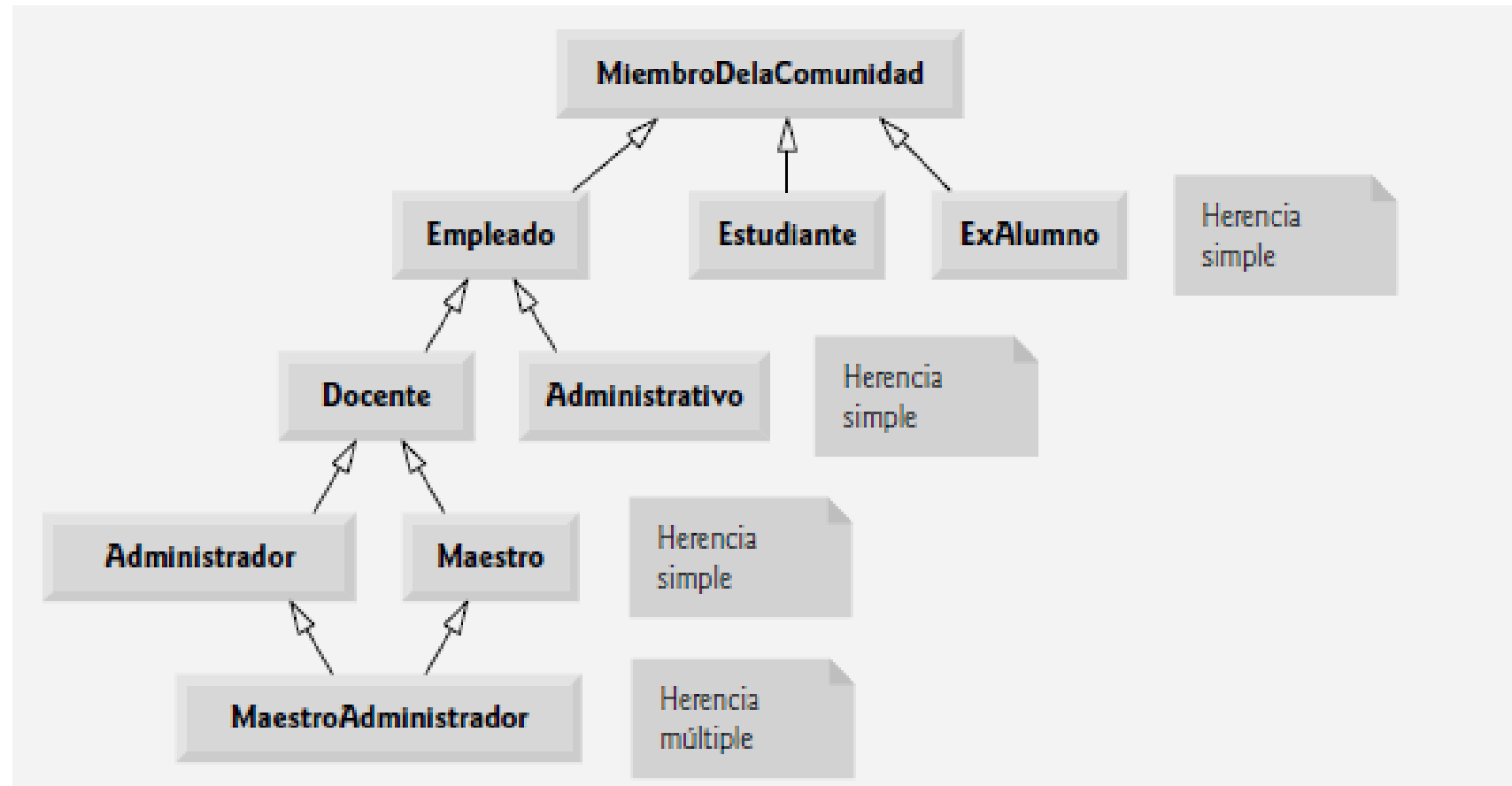
Una **clase base directa** es la **clase base** a partir de la cual una **clase derivada** hereda en forma explícita. Una clase base indirecta es la que se hereda de dos o más niveles hacia arriba en la jerarquía de clases. En el caso de la herencia simple, una clase se deriva de una sola clase base.

C++ también soporta la herencia múltiple, en la cual una clase derivada hereda de varias clases base (posiblemente no relacionadas).

Clases base y clases derivadas



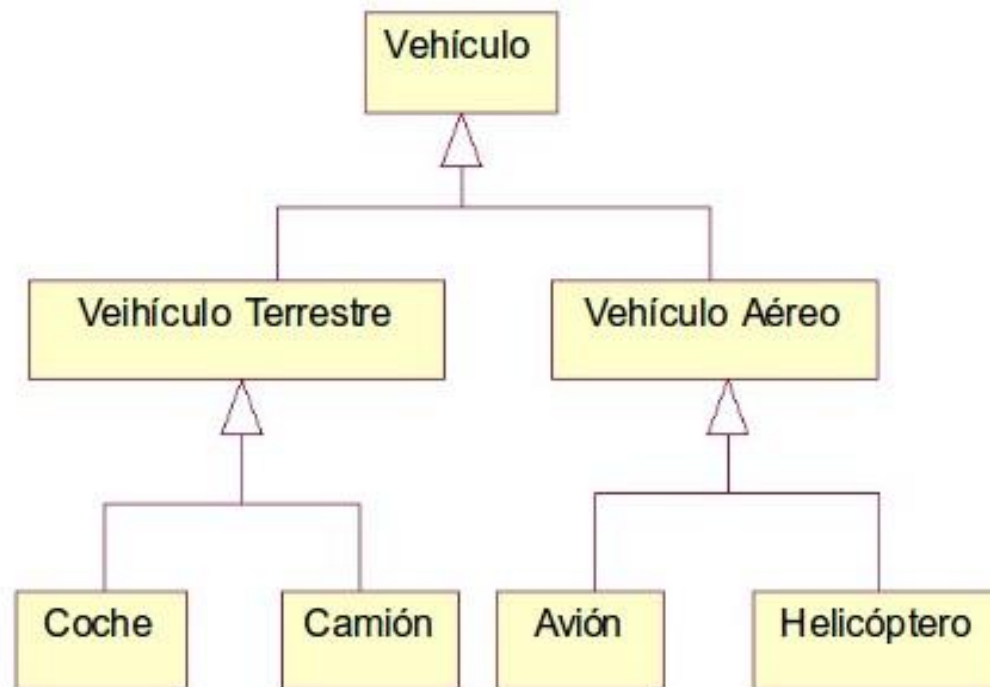
En este ejemplo se observa la herencia simple y la herencia múltiple:



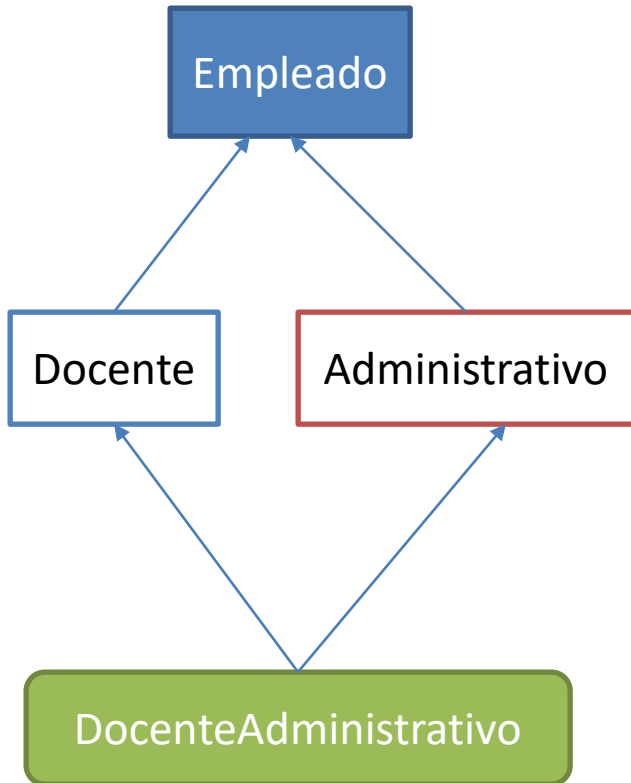
Clases base y clases derivadas



- Otro ejemplo, la clase base CVehiculo representa a todos los vehículos, incluyendo vehículos terrestres y aéreos.
- En contraste, la clase derivada Coche representa un subconjunto más pequeño y específico de todos los vehículos.



Ejemplo: Clases base y clases derivadas



Clase base	Clases derivadas
Estudiante	EstudianteGraduado, EstudianteNoGraduado
Figura	Circulo, Triangulo, Rectangulo, Esfera, Cubo
Prestamo	PrestamoAuto, PrestamoMejoraHogar, PrestamoHipotecario
Empleado	Docente, Administrativo
CuentaBanco	CuentaCheques, CuentaAhorros

Acceso a los miembros de una clase



When the component is declared as:	When the class is inherited as:	The resulting access inside the subclass is:
public	public	Public
protected		protected
private		none
public	protected	protected
protected		protected
private		none
public	private	private
protected		private
private		none

Accesos a los miembros de una clase



```
class A {  
    public: int x;  
    protected: int y;  
    private: int z;  
};  
class B : public A {  
    // x  
    // y  
    // z  
};  
class C : protected A {  
    // x  
    // y  
    // z  
};
```

Accesos a los miembros de una clase



- Configuración de las secciones de private, protected y public:

```
class CFigura { // clase padre
    protected:
        int x;
        int y;
};

class CCuadrado : public CFigura {
    // x e y son protegidos
};

class CCirculo : public CFigura {
    private:
        int radio;

    // x e y son protegidos
};
```



```
#pragma once
class CClaseA
{
public:
    int a;
    int b;
    int c;
public:
    CClaseA();
    ~CClaseA();
    int getA();
    void setA(int a);
    int getB();
private:
    void setB(int b);
    int getC();
protected:
    void setC(int c);
    int suma();
};
```

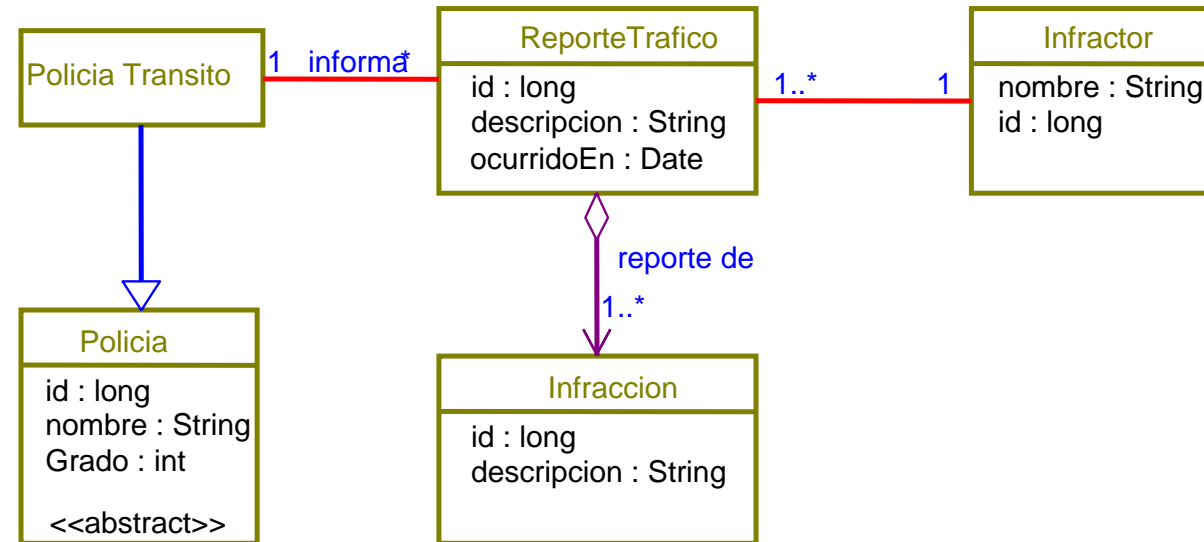
Sin acceso privado del método

Sin acceso protegido del método

```
#include "ClaseA.h"
CClaseA::CClaseA()
{
}
CClaseA::~~CClaseA()
{
}
int CClaseA::getA(){ return a; }
void CClaseA::setA(int a){ this->a = a; }
int CClaseA::getB(){ return b; }
void CClaseA::setB(int b){ this->b = b; }
int CClaseA::getC(){ return c; }
void CClaseA::setC(int c){ this->c = c; }
int CClaseA::suma(){ return a + b + c; }
```

```
#include <iostream>
#include <conio.h>
#include "ClaseA.h"
using namespace std;
using namespace System;
int main(){
    CClaseA *obj = new CClaseA();
    int a, b, c;
    cout << "ingrese a b c: ";
    cin >> a>>b>>c;
    obj->setA(a);
    cout << "Valor ingresado para A: " << endl;
    obj->getA();
    cout << "Valor ingresado para B: " << endl;
    obj->setB(a);
    obj->getB();
    cout << "Valor ingresado para C: " << endl;
    obj->setC(c);
    obj->getC();
    cout << "La suma es: " << endl;
    obj->suma();
}
```

Ejemplo de sistema de informes de infracciones de tráfico



Ejemplo 2



Escribir un POO de entorno de consola que permita hallar:

- 1.La longitud y área de un círculo
- 2.El área y volumen de un cilindro
- 3.El área y el volumen de un cilindro hueco.

Del Círculo:

$$\text{Longitud} = 2\pi r$$

$$\text{Area} = \pi r^2$$

Del Cilindro:

$$\text{Area} = 2\pi r h + 2\pi r^2$$

$$\text{Volumen} = \pi r^2 h$$

Cilindro Hueco:

$$\text{Area} = 2\pi r h + 2\pi h i + 2\pi r^2 - 2\pi i^2$$

$$\text{Volumen} = \pi r^2 h - \pi i^2 h$$

Donde:

r es el radio mayor
 i es el radio menor
 h es la altura

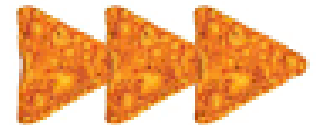
Ejemplo 2



1er paso: Identificamos clases:

- Circulo
- Cilindro
- Cilindro Hueco

2do paso: Definimos atributos y métodos para cada clase



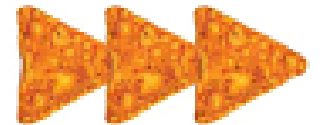
Ejemplo 2



```
class CCirculo
{private:
    double r;
public:
    CCirculo(double r);
    ~ CCirculo();
    double AreadelCirculo();
    double Longitud();
};
```

```
class CCilindro
{ private:
    double r;
    double h;
public:
    CCilindro(double r, double h);
    ~CCilindro();
    double AreadelCilindro();
    double VolumendelCilindro();
};
```

```
class CCilindroHueco
{private:
    double r; //--- radio mayor
    double i; //--- radio menor
    double h;
public:
    CCilindroHueco(double r, double i, double h);
    ~ CCilindroHueco();
    double AreadelCilindroHueco();
    double VolumendelCilindroHueco();
};
```



Ejemplo 2



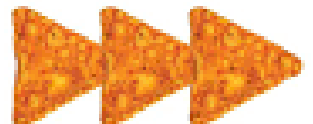
```
class CCirculo
{private:
    double r;
public:
    CCirculo(double r);
    ~CCirculo();
    double AreadelCirculo();
    double Longitud();
};
```

```
class CCilindro
{ private:
    double r;
    double h;
public:
    CCilindro(double r, double h);
    ~CCilindro();
    double AreadelCilindro();
    double VolumendelCilindro();
};
```

```
class CCilindroHueco
{private:
    double r; //--- radio mayor
    double i; //--- radio menor
    double h;
public:
    CCilindroHueco(double r, double i, double h);
    ~CCilindroHueco();
    double AreadelCilindroHueco();
    double VolumendelCilindroHueco();
};
```

3er paso: Se observan si hay atributos en común, si los hay se define la clase base.

Si hay métodos comunes se analiza y si los hay se los define en la clase base.



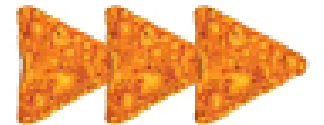
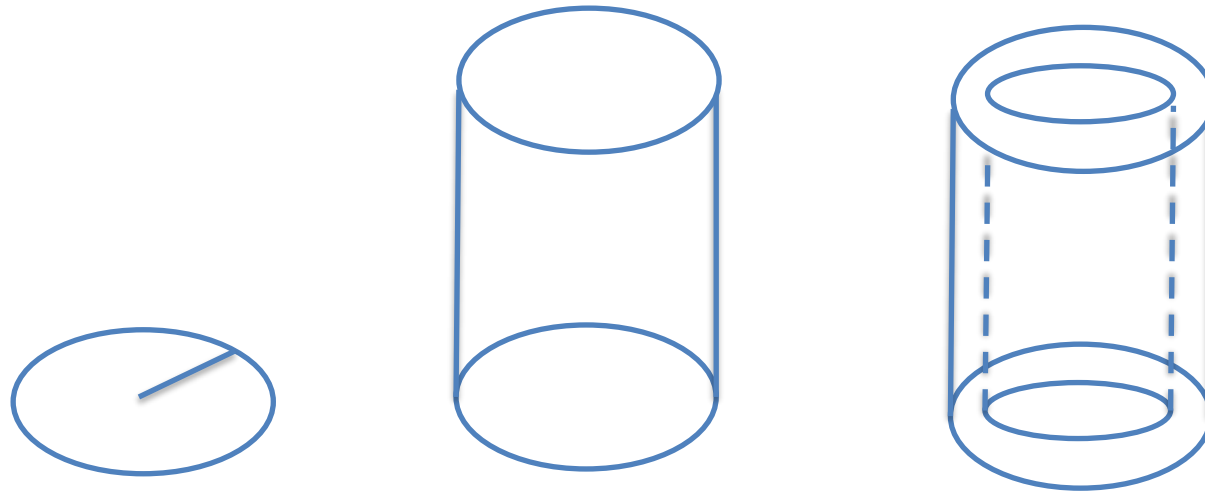
Ejemplo 2



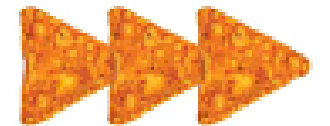
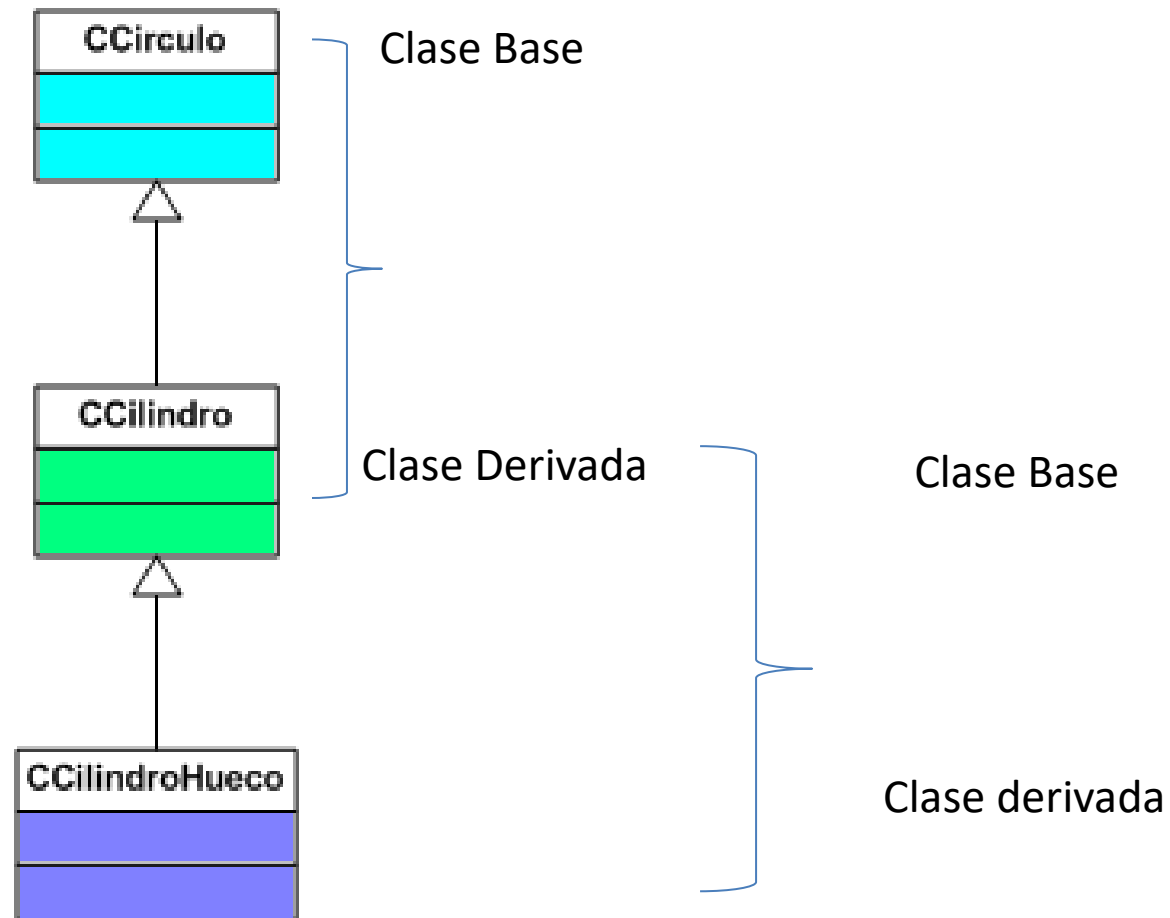
En esencia podríamos decir que:

- Un objeto cilindro es un objeto círculo con una altura
- Un objeto cilindro hueco es un cilindro con un espacio hueco.

Aunque semánticamente el ejemplo no es perfecto nos permitirá indicar la sintaxis del lenguaje para los casos de herencia.



Ejemplo 2



Ejemplo 2

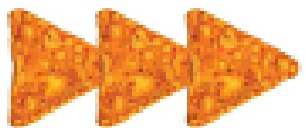


3er paso: Continuando...

```
class CCirculo
{ protected:
    double r;
public:
    CCirculo(double r);
    ~CCirculo();
    double AreadelCirculo();
    double Longitud();
};
```

```
class CCilindro : public CCirculo
{ protected:
    double h;
public:
    CCilindro(double r, double h);
    ~CCilindro();
    double AreadelCilindro();
    double VolumendelCilindro();
};
```

```
class CCilindroHueco : public CCilindro
{ protected:
    double i; //--- radio menor
public:
    CCilindroHueco(double r, double i, double h);
    ~CCilindroHueco();
    double AreadelCilindroHueco();
    double VolumendelCilindroHueco();
};
```




```
//-- Figuras.h
```

```
class CCirculo
```

```
{protected:
```

```
    double r;↵
```

```
public:
```

```
    CCirculo(double r);
```

```
    ~ CCirculo();
```

```
    double AreadelCirculo();
```

```
    double Longitud();
```

```
};
```

```
class CCilindro : public CCirculo
```

```
{ protected:
```

```
    double h;
```

```
public:
```

```
    CCilindro(double r, double h);
```

```
    ~CCilindro();
```

```
    double AreadelCilindro();
```

```
    double VolumendelCilindro();
```

```
};
```

```
class CCilindroHueco : public CCilindro
```

```
{protected:
```

```
    double i;  //-- radio menor
```

```
public:
```

```
    CCilindroHueco(double r, double i, double h);
```

```
    ~ CCilindroHueco();
```

```
    double AreadelCilindroHueco();
```

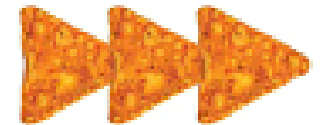
```
    double VolumendelCilindroHueco();
```

```
};
```

Las palabras reservadas: **private**, **protected** y **public** pueden aparecer en cualquier orden y cualquier número de veces en una clase, particionando éstas en múltiples partes, privadas, públicas o protegidas.

Protected indica que todo aquello que se encuentre en esta sección puede ser accedida por cualquier clase derivada de la clase base.

public en la cabecera de la definición de una clase, define la relación de herencia



Class Circulo



```
class CCirculo
{
protected:
    double r;
public:
    CCirculo(double r);
    ~CCirculo();
    double AreadelCirculo();
    double Longitud();
};
```

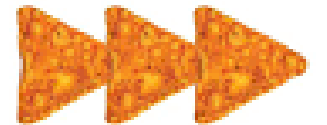
```
//----- CCirculo -----
```

```
CCirculo::CCirculo(double r)
{
    this->r = r;
}
```

```
CCirculo::~~CCirculo() {}
```

```
double CCirculo::AreadelCirculo()
{
    return(3.1416*r*r);
}
```

```
double CCirculo::Longitud()
{
    return(2*3.1416*r);
}
```



Clase Cilindro



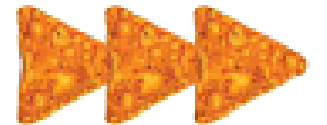
```
class CCilindro : public CCirculo
{
protected:
    double h;
public:
    CCilindro(double r, double h);
    ~CCilindro();
    double AreadelCilindro();    double VolumendelCilindro();
};

//----- CCilindro -----
CCilindro::CCilindro(double r, double h) :CCirculo(r)//Invoca al constructor del Padre
{
    this->h = h;
}

CCilindro::~~CCilindro() {}

double CCilindro::AreadelCilindro()
{
    return(Longitud() *h + 2 * AreadelCirculo());
}

double CCilindro::VolumendelCilindro()
{
    return(AreadelCirculo() *h);
}
```



Clase CilindroHueco



```
class CCilindroHueco : public CCilindro
{
protected:
    double i;  //-- radio menor
public:
    CCilindroHueco(double r, double i, double h);
    ~CCilindroHueco();
    double AreadelCilindroHueco();
    double VolumendelCilindroHueco();
};
```

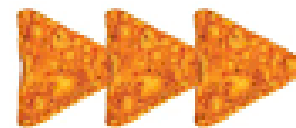
```
//----- CCilindroHueco -----
```

```
CCilindroHueco::CCilindroHueco(double r, double i, double h) :CCilindro(r, h)//invoca al constructor de la clase Padre
{
    this->i = i;
}
```

```
CCilindroHueco::~~CCilindroHueco() {}
```






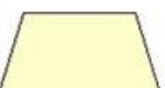
```
double CCilindroHueco::AreadelCilindroHueco()
{
    return(AreadelCilindro() + 2 * 3.1416*h*i - 2 * 3.1416*i*i);
}
```

```
double CCilindroHueco::VolumendelCilindroHueco()
{
    return(VolumendelCilindro() - 3.1416*i*i*h);
}
```



Ejercicio de Aplicación



FORMA	ELEMENTOS	FÓRMULA PERÍMETRO	FÓRMULA ÁREA
TRIÁNGULO 	b: Base h: Altura l: Lado1 m: Lado2 n: Lado3	$P = l + m + n$	$A = \frac{b \times h}{2}$
CUADRADO 	a: Lado	$P = 4a$	$A = a^2$
RECTÁNGULO 	b: Base h: Altura	$P = 2b + 2h$	$A = b \times h$
ROMBO 	a: Lado d: Diagonal menor D: Diagonal mayor	$P = 4a$	$A = \frac{D \times d}{2}$
ROMBOIDE 	b: Base h: Altura	$P = 2b + 2h$	$A = b \times h$
TRAPECIO 	l: Lado1 m: Lado2 n: Lado3 o: Lado4 b: Base menor B: Base mayor h: Altura	$P = l + m + n + o$	$A = \frac{h (B + b)}{2}$

Se le solicita que, haciendo uso de conceptos de POO y relaciones de herencia, elabore un programa para calcular el **perímetro y área** de las figuras mostradas.

Para la solución considere todas las relaciones de herencia que crea conveniente.



Ejemplo 3



La EISC requiere un programa en *Console Application* con C++ que le permita poder administrar los datos de sus docentes para el presente ciclo académico.

Implemente una clase padre llamada CDocente. La clase tiene los siguientes atributos:

- Código: Tipo string
- Nombres Completos: Tipo string
- Estudios de Postgrado: Tipo int
 - Ninguno (0), Maestría (1), Doctorado (2), Ambas (3)
- Años de Antigüedad: Tipo int
- Horas de Clase: Tipo int

Clase Padre



- La clase padre CDocente tiene la siguiente definición:

```
class CDocente { // clase padre
    protected:
        string Codigo;
        string NombresCompleto;
        int EstudiosPostgrado;
        int AniosAntigüedad;
        int Horas;

    public:
        CDocente();
        CDocente(string Codigo, string NombresCompleto, int
EstudiosPostgrado, int AniosAntigüedad, int Horas);
        ~CDocente();

        // Métodos de Acceso
        string Get_Codigo();
        void Set_Codigo(string Codigo);
};
```


Implementación de las Clases: Principal, Asociado y Auxiliar



Implemente las clases hijas heredadas de CDocente, donde las reglas para el cálculo del sueldo neto son las siguientes:

$$\text{Pago Parcial} = \text{Horas de Clase} \times \text{Pago por Hora}$$

Donde el Pago por Hora se clasifica por categoría:

Categoría	Pago por hora
Principal	25.00
Asociado	18.00
Auxiliar	15.00

Además, la bonificación que se calcula del porcentaje del Pago Parcial en base a los estudios de Postgrado como se muestra en la siguiente tabla:

Implementación de las Clases: Principal, Asociado y Auxiliar



Categoría	Estudios de Postgrado (% de Bonificación)		
	Con Doctorado	Con Maestría	Ambas
Principal	20%	17%	25%
Asociado	15%	10%	20%
Auxiliar	12%	8%	17%

Entonces, el Sueldo bruto es:

$$\text{Sueldo Bruto} = \text{Pago Parcial} + \text{Bonificación}$$

Y los descuentos se representan con las siguientes reglas:

Años de Antigüedad	% de Descuento sobre el sueldo bruto
< 7 años	5%
>=8 años	4%

Implementación de las Clases: Principal, Asociado y Auxiliar



Por lo tanto, el sueldo neto es:

$$\text{Sueldo Neto} = \text{Sueldo Bruto} - \text{Descuentos}$$

Registrar un Docente

Implemente la funcionalidad de registro para un nuevo docente y se debe actualizar el listado general de docentes.

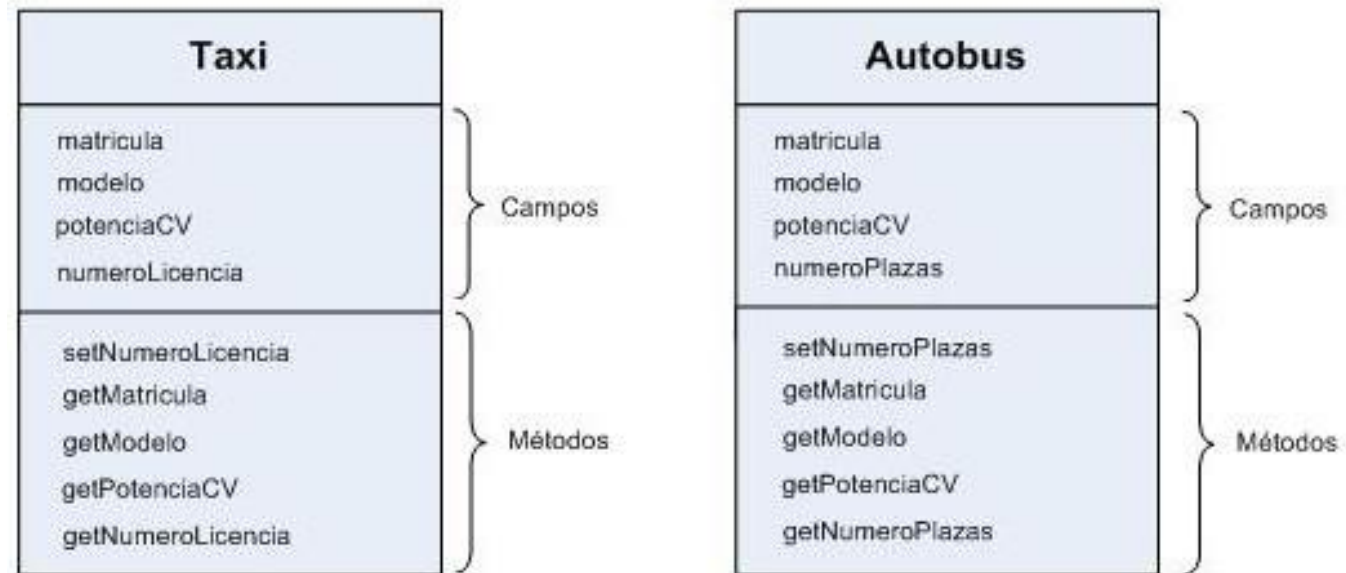
Reportes Gerenciales

- Todos los docentes entre 5 y 10 años de experiencia
- Todos los docentes que no tienen ningún estudio de Postgrado
- Todos los docentes que tienen un sueldo neto menor a S/. 1,500

Ejemplo 4:



Supongamos que tenemos que implementar un programa en C++ en donde se considera dos objetos: Taxi y Autobús podríamos encontrarnos algo así:

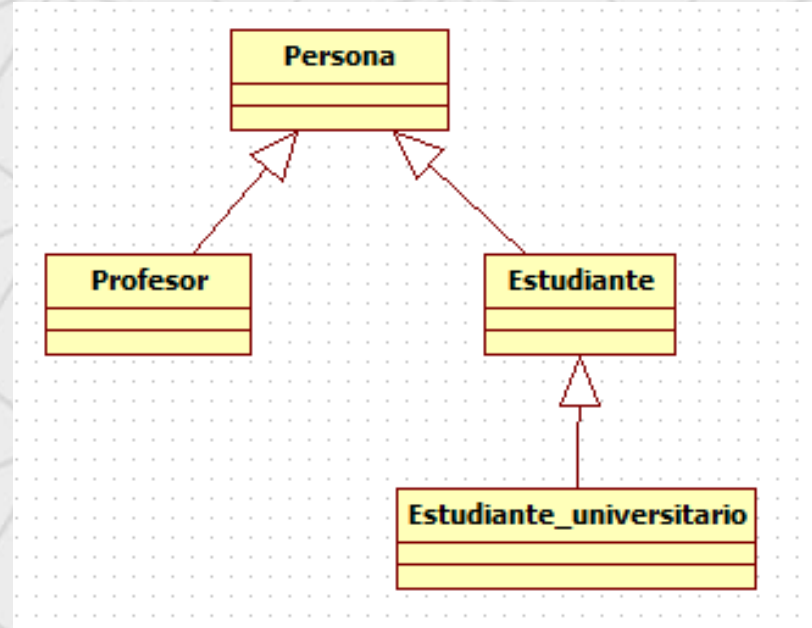


Aplicando el concepto de herencia, diseñar el diagrama de clases.

Ejemplo 5



Se tiene el siguiente diagrama de clases:



- Identifique los atributos y métodos
- Defina e implemente las clases
- Implementar en C++ un programa que construya una solución para la jerarquía de las clases mostradas.

Ejemplo 6



El Zoológico de Lima requiere administrar un aplicativo que permita manejar sus animales, inicialmente se considera que hay perros y gatos.

- De todos los animales se registran: Nombre y Edad
 - Dentro de los animales hay Felinos de cuales se anotan: Tipo de Cabeza y Tipo de Pelaje
 - Un perro no pertenece a los felinos, además se anotan: El tipo de Hocico y el nivel de peligro que son catalogados.
 - Los gatos son felinos, se registran la clase a la cual pertenece y su tipo alimentación que es especial.
- a) Diseñar el diagrama de clases
 - b) Implementar las clases de acuerdo al diseño
 - c) Integrar en la función main, ingresar un datos para cada tipo de animal