CS1102 – PROGRAMACIÓN ORIENTADA A OBJETOS 1
CICLO 2019-1

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CIENCIA DE LA COMPUTACIÓN

Unidad 8:
Programación Orientada a Objetos - Parte 4
Herencia y polimorfismo

http://bit.ly/2HRBWgq

Profesores:

Ernesto Cuadros- Vargas, PhD. *ecuadros@utec.edu.pe*

María Hilda Bermejo, M. Sc. *mbermejo@utec.edu.pe*

http://bit.ly/325B0Nc

Logro de la sesión:

Al finalizar la sesión, los alumnos diseñan POO, utilizando herencia y polimorfismo

- **Herencia**
- **Polimorfismo**

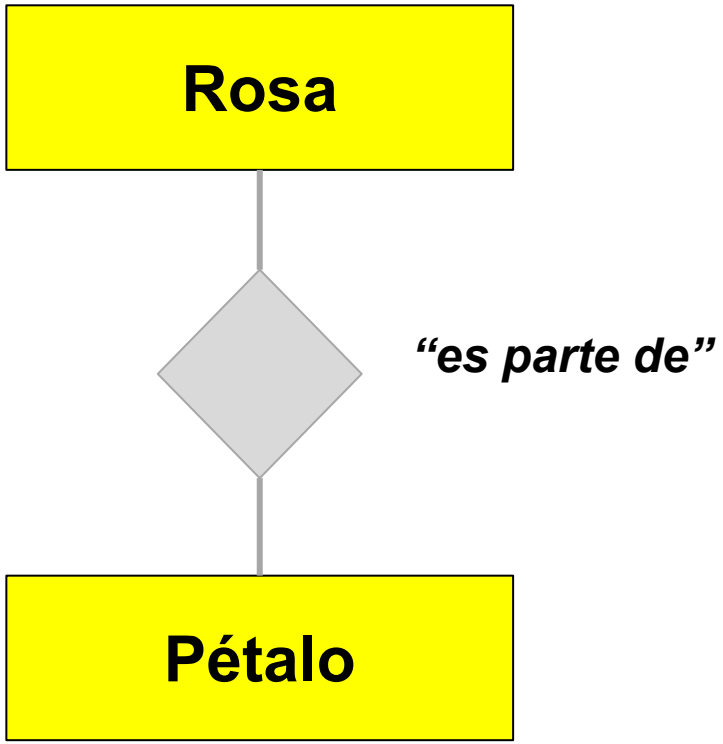
https://github.github.com/training-kit/downloads/github-git-cheat-sheet.pdf

Video:

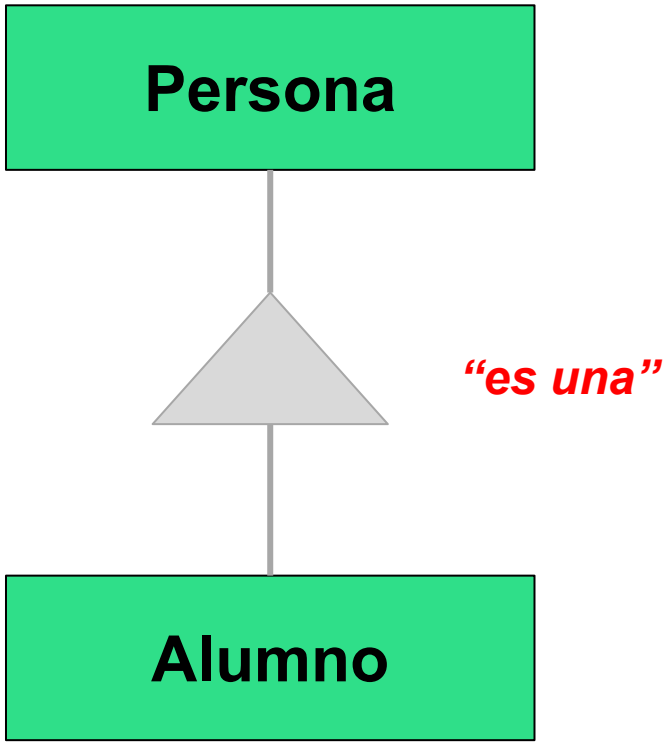
https://www.youtube.com/watch?v=HVsySz-h9r4

Relaciones entre clases:

Relación de Agregación



Relación de Herencia

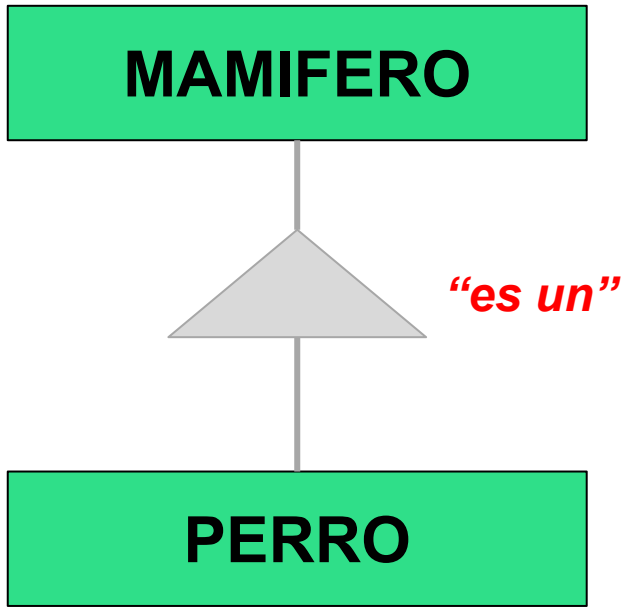


UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Herencia

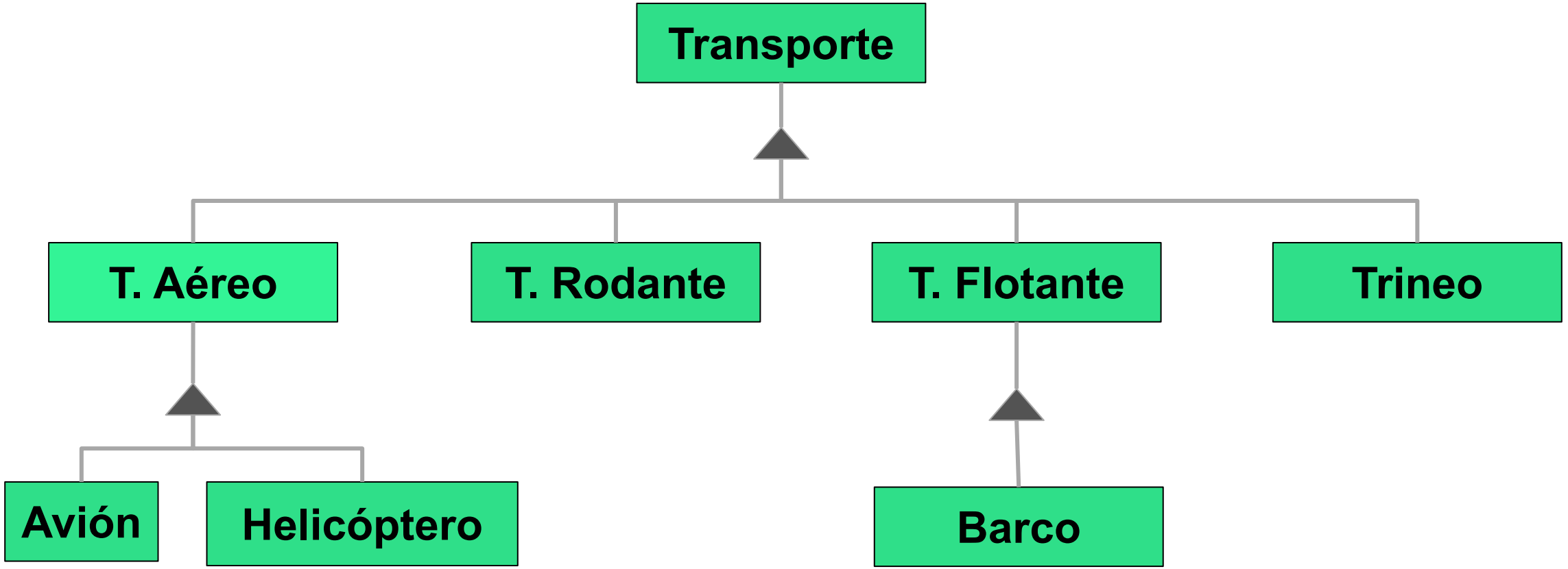
UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Herencia



Clase padre

Clase hija



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.
- La subclase hereda todos los miembros: atributos y métodos de la superclase.

Definición

- La subclase definirá su (s) propio (s) constructor (es).
- La subclase puede definir miembros adicionales (atributos o métodos).
- La sintaxis para la herencia de clases es la siguiente:

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

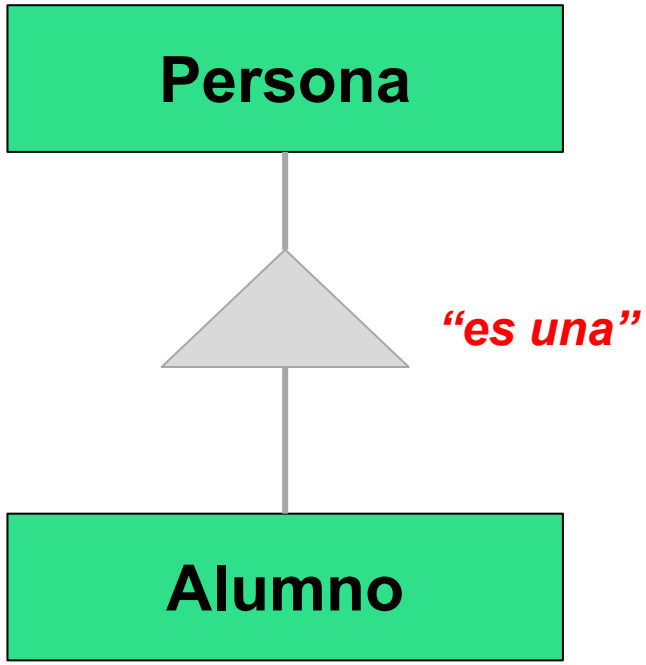
UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Jerarquía de Clases

- Las clases se pueden organizar como una jerarquía de clases.
- Las clases más generales se definirán en las jerarquías superiores.
- Las clases más específicas se definirán en las jerarquías inferiores.

Ejemplo 1:

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



Clase padre

Clase hija

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

main.cpp

```cpp
#include <iostream>
#include <memory>  //-- para usar punteros inteligentes
#include "CPersona.h"
#include "CAlumno.h"

using namespace std;

int main()
{ //---usando punteros inteligentes
 shared_ptr<CPersona> pPersona = make_shared<CPersona>("Ricardo", 34);
 unique_ptr<CAlumno> pAlumno = make_unique<CAlumno>("201812345", "Pedro", 18);

 pPersona->Imprimir();
 cout <<"\n";
 pAlumno->Imprimir();

 return 0;
}
```

```
Nombre : Ricardo
Edad   : 34

Codigo : 201812345
Nombre : Pedro
Edad   : 18
```

Definiciones.h

```cpp
#ifndef HERENCIA01_DEFINICIONES_H
#define HERENCIA01_DEFINICIONES_H

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

using TipoNumerico = float;
using TipoCadena = string;

#endif //HERENCIA01_DEFINICIONES_H
```

CPersona.h

```
#ifndef HERENCIA01_CPERSONA_H
#define HERENCIA01_CPERSONA_H

#include "Definiciones.h"

class CPersona
{
 protected:
   TipoCadena    name;
   TipoNumerico  age;
 public:
   CPersona(){};
   CPersona(TipoCadena _name, TipoNumerico _age);
   virtual ~CPersona(){}
   void Imprimir();
   //---metodos de acceso
   void         setName(TipoCadena _name){name=_name;}
   void         setAge(TipoNumerico _age){ age = _age;}
   TipoCadena   getName()  { return name; };
   TipoNumerico getAge() { return age; }
};

#endif //HERENCIA01_CPERSONA_H
```

CPersona.cpp

```
#include "CPersona.h"

CPersona::CPersona(TipoCadena _name, TipoNumerico _age)
{
 name  = _name;
 age   = _age;
}

void CPersona::Imprimir()
{
 cout << "Nombre : " << name << "\n";
 cout << "Edad   : " << age << "\n";
}
```

CAlumno.h

```
#ifndef HERENCIA01_CALUMNO_H
#define HERENCIA01_CALUMNO_H

#include "CPersona.h"

class CAlumno:public CPersona
{
 private:
   TipoCadena code;
 public:
   CAlumno(){};
   CAlumno(TipoCadena _code, TipoCadena _name,
           TipoNumerico _age);
   virtual ~CAlumno(){};
   void Imprimir();
   //--- metodos de acceso
   void setCode(TipoCadena _code) { code = _code; }
   TipoCadena getCode() { return code; };
};

#endif //HERENCIA01_CALUMNO_H
```

CAlumno.cpp

```
#include "CAlumno.h"

CAlumno::CAlumno(TipoCadena _code, TipoCadena _name,
                 TipoNumerico _age):CPersona(_name, _age)
{
 code = _code;
}

void CAlumno::Imprimir()
{
 cout << "Codigo : " << code << "\n";
 CPersona::Imprimir();
}
```

Especificador de Acceso de Herencia

- Especifica el tipo de herencia: ***public,*** que es el más usado, pero existen otros.

- En este caso, los miembros heredados en la subclase tienen la misma visibilidad que la superclase.

Herencia: Constructores

- 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 ...continua

- 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:

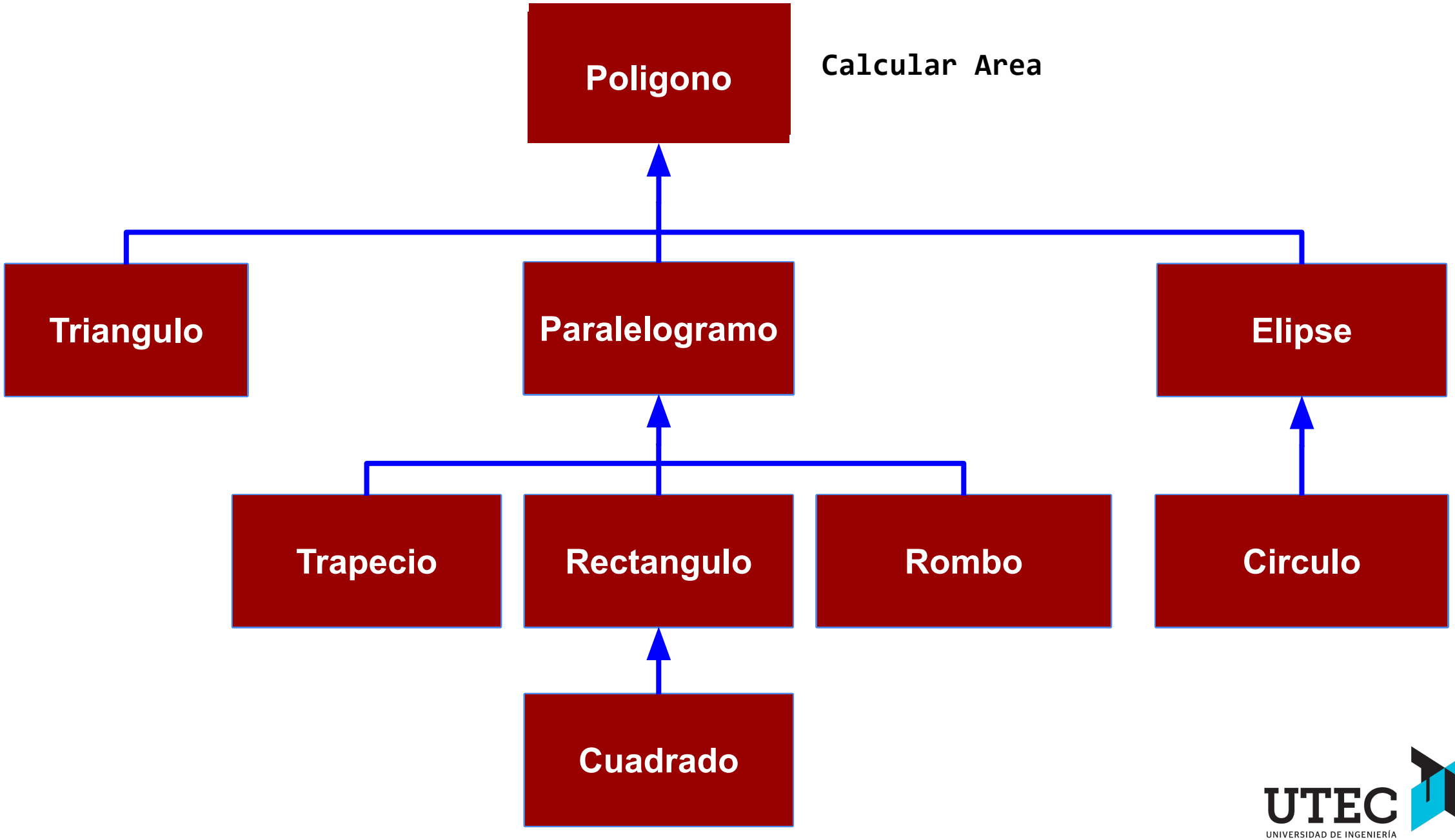
```
SuperclassName::memberName
```

Por ejemplo:

```
Persona::imprime();

Persona::getEdad();
```

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Polimorfismo

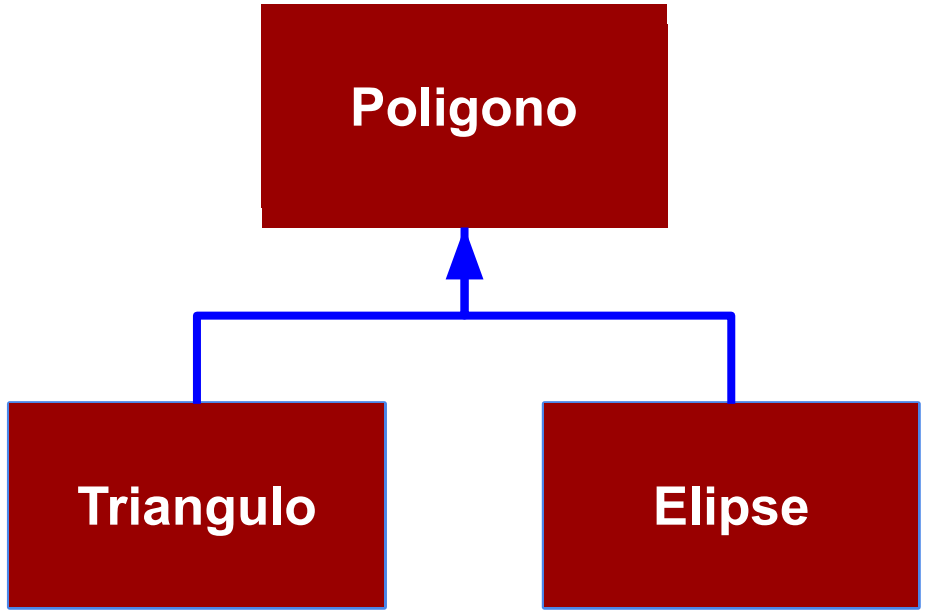


UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

```cpp
int main() {

 vector<Poligono*> poligonos = {new Triangulo(10, 10), new Paralelograma(20, 30)};

 poligonos.push_back(new Elipse(10, 12));

 for (auto i = 0; i < poligonos.size(); i++)
   cout << "Area es: " << poligonos[i]->calcularArea() << endl;

 for (auto i = 0; i < poligonos.size(); i++)
   delete poligonos[i];

 poligonos.clear();

}
```

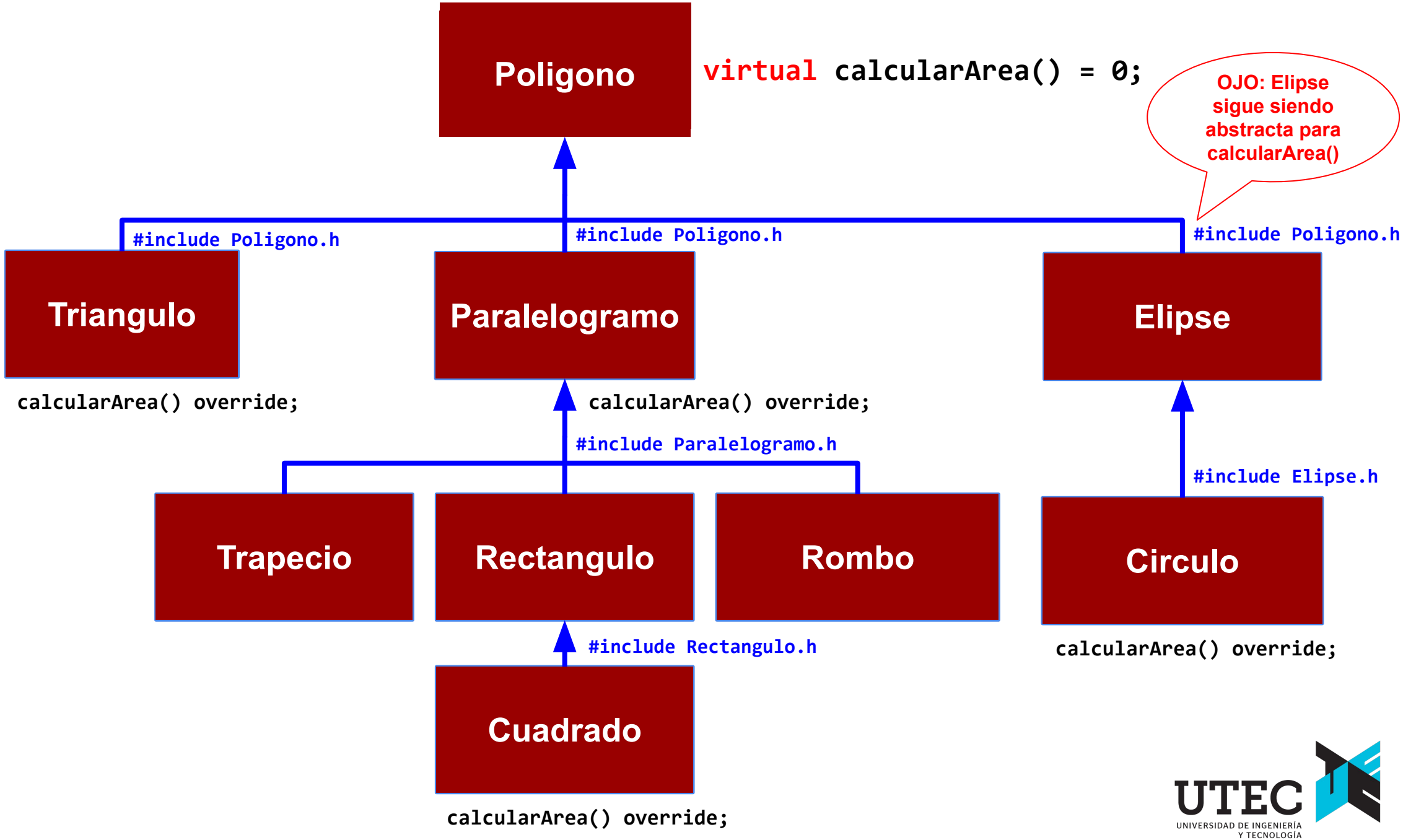
https://repl.it/@RubenDemetrioDemetrio/POO-Polimorfismo?language=cpp11&folderId=

```
class Poligono {
 public:
   virtual ~Poligono() {};
   virtual Number calcularArea() = 0;
};
```




```
class Triangulo: public Poligono {
   Number base;
   Number altura;
 public:
   Triangulo(Number base, Number altura);
   Number calcularArea() override;
};
```

```
class Elipse: public Poligono {
   Number ejeMayor;
   Number ejeMenor;
 public:
   Elipse(Number ejeMayor, Number ejeMenor);
   Number calcularArea() override;
};
```



UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

POLIMOFISMO

El polimorfismo es la propiedad que tienen los objetos de responder de diferente manera frente a un mismo mensaje.

Tipos de polimorfismo:

1. Por herencia
2. Por sobrecarga de operadores
3. Por sobrecarga de funciones
4. Utilizando templates o por programación genérica.

En esta clase, solo revisaremos el polimorfismo, utilizando herencia.

Ejemplo:

Un polígono regular es aquel que tiene sus lados iguales. Son polígonos regulares por ejemplo: un cuadrado, un triángulo equilátero, un hexágono.
En el ejemplo se quiere calcular el área, el semiperímetro y el apotema de cualquiera de estas figuras.
Se sabe que para calcular el área de cualquiera de estos polígonos regulares se puede utilizar:

Area del Poligono = SemiPerimetro * Apotema

El apotema es la distancia que hay entre el centro de la figura y el punto medio de cualquier lado.
Para realizar el cálculo del semiperímetro y el apotema se pueden aplicar las siguientes fórmulas:

Polígono	Apotema	Semiperímetro
Triángulo	L*raiz(3)/6	3 L/2
Cuadrado	L/2	2 L
Hexágono	L*raiz(3)/2	3 L

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Ejecución 1

```
Seleccione el tipo de poligono que quiere crear
1. Triangulo
2. Cuadrado
3. Hexagono
Ingresa el tipo : 1
Lado : 10
Apotema : 2.88675
Semiperimetro :15
El area es 43.3013
```

Ejecución 2

```
Seleccione el tipo de poligono que quiere crear
1. Triangulo
2. Cuadrado
3. Hexagono
Ingresa el tipo : 3
Lado : 10
Apotema : 8.66025
Semiperimetro :30
El area es 259.808
```

Ejecución 3

```
Seleccione el tipo de poligono que quiere crear
1. Triangulo
2. Cuadrado
3. Hexagono
Ingresa el tipo : 2
Lado : 10
Apotema : 5
Semiperimetro :20
El area es 100
```

Ctriangulo.h

```
#ifndef PROG_01_CTRIANGULO_H
#define PROG_01_CTRIANGULO_H

#include "Definiciones.h"

class CTriangulo
{
 private:
   TipoNumerico m_lado;
 public:
   CTriangulo() {};
   CTriangulo(TipoNumerico lado);
   virtual ~CTriangulo(){};
   //---metodo de acceso
   void set_m_Lado(TipoNumerico lado) {m_lado=lado;}
   TipoNumerico getLado(){ return m_lado;}

   TipoNumerico Apotema();
   TipoNumerico SemiPerimetro();
   TipoNumerico Area();
};

#endif //PROG_01_CTRIANGULO_H
```

Ctriangulo.cpp

```
#include "CTriangulo.h"

CTriangulo::CTriangulo(TipoNumerico lado)
{
 m_lado = lado;
}

TipoNumerico CTriangulo::Apotema()
{
 return (m_lado * sqrt(3)/6.0);
}

TipoNumerico CTriangulo::SemiPerimetro()
{
 return (3.0 * m_lado/2.0);
}

TipoNumerico CTriangulo::Area()
{
 return(SemiPerimetro()*Apotema());
}
```

CCuadrado.h

```
#ifndef PROG_01_CCUADRADO_H
#define PROG_01_CCUADRADO_H

#include "Definiciones.h"

class CCuadrado
{
 private:
   TipoNumerico m_lado;
 public:
   CCuadrado() {};
   CCuadrado(TipoNumerico lado);
   virtual ~CCuadrado(){};
   //---metodo de acceso
   void set_m_Lado(TipoNumerico lado){m_lado=lado;}
   TipoNumerico getLado(){ return m_lado;}

   TipoNumerico Apotema();
   TipoNumerico SemiPerimetro();
   TipoNumerico Area();
};

#endif //PROG_01_CCUADRADO_H
```

CCuadrado.cpp

```
#include "CCuadrado.h"

CCuadrado::CCuadrado(TipoNumerico lado)
{
 m_lado = lado;
}

TipoNumerico CCuadrado::Apotema()
{
 return (m_lado/2.0);
}

TipoNumerico CCuadrado::SemiPerimetro()
{
 return( 2.0*m_lado);
}

TipoNumerico CCuadrado::Area()
{
 return(SemiPerimetro()*Apotema());
}
```

CHexagono.h

```
#ifndef PROG_01_CHEXAGONO_H
#define PROG_01_CHEXAGONO_H

#include "Definiciones.h"

class CHexagono
{
 private:
   TipoNumerico m_lado;
 public:
   CHexagono() {};
   CHexagono(TipoNumerico lado);
   virtual ~CHexagono(){};
   //---metodo de acceso
   void set_m_Lado(TipoNumerico lado){m_lado=lado;}
   TipoNumerico getLado(){ return m_lado;}
   TipoNumerico Apotema();
   TipoNumerico SemiPerimetro();
   TipoNumerico Area();
};

#endif //PROG_01_CHEXAGONO_H
```

CHexagono.cpp

```
#include "CHexagono.h"

CHexagono::CHexagono(TipoNumerico lado)
{
 m_lado = lado;
}

TipoNumerico CHexagono::Apotema()
{
 return( m_lado*sqrt(3.0)/2.0);
}

TipoNumerico CHexagono::SemiPerimetro()
{
 return(3.0* m_lado);
}

TipoNumerico CHexagono::Area()
{
 return(SemiPerimetro()*Apotema());
}
```

Main.cpp

```
#include <iostream>
#include "Definiciones.h"
#include "CCuadrado.h"
#include "CTriangulo.h"
#include "CHexagono.h"

using namespace std;

int main()
{
 EnteroSinSigno Opcion;
 TipoNumerico lado;
 CTriangulo *pTriangulo= nullptr;
 CCuadrado *pCuadrado= nullptr;
 CHexagono *pHexagono = nullptr;

 do
 {
   cout << "Seleccione el tipo de poligono que quiere crear
\n";
   cout << "1. Triangulo \n";
   cout << "2. Cuadrado \n";
   cout << "3. Hexagono \n";
     cout << "Ingresa el tipo : ";
     cin >> Opcion;
 }while( Opcion<1 || Opcion>3);

cout <<"Lado : ";
   cin >> lado;
 switch(Opcion)
 {
   case 1:
     pTriangulo = new CTriangulo(lado);
     cout << "El apotema del triangulo " << pTriangulo->Apotema()<<"\n";
     cout << "El semiperimetro del triangulo " <<
pTriangulo->SemiPerimetro()<< "\n";
     cout << "El area es " << pTriangulo->Area()<< "\n";
     delete pTriangulo;
     break;
   case 2:
     pCuadrado = new CCuadrado(lado);
     cout << "Apotema : "<< pCuadrado->Apotema() <<"\n";
     cout << "Semiperimetro :" << pCuadrado->SemiPerimetro() << "\n";
     cout << "El area es " << pCuadrado->Area()<< "\n";
     delete pCuadrado;
     break;
   case 3:
     pHexagono = new CHexagono(lado);
     cout << "Apotema : "<< pHexagono->Apotema() <<"\n";
     cout << "Semiperimetro :" << pHexagono->SemiPerimetro() << "\n";
     cout << "El area es " << pHexagono->Area()<< "\n";
     delete pHexagono;
 }
 return 0;
}
```

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Aplicamos “Generalización y Especialización”

Ahora analizamos los métodos:

Qué	Cómo	Acción
Son iguales	Son iguales	Se declara el método en la clase ancestra, y las clases hijas lo heredarán. Se elimina la definición de las clases hijas.
Son iguales	Son diferentes	Se declara el método en la clase ancestra como "virtual" y se redefine en las clases hijas se utiliza "override"

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Analizamos métodos

```
 9  class CTriangulo
10  {
11    private:
12      TipoNumerico m_lado;
13    public:
14      CTriangulo() {};
15      CTriangulo(TipoNumerico lado);
16      virtual ~CTriangulo(){};
17      //---metodo de acceso
18      void set_m_Lado(TipoNumerico lado) { m_lado = lado;}
19      TipoNumerico getLado(){ return m_lado;}
20
21      TipoNumerico Apotema();
22      TipoNumerico SemiPerimetro();
23      TipoNumerico Area();
24  };
```

```
10  class CCuadrado
11  {
12  private:
13      TipoNumerico m_lado;
14  public:
15      CCuadrado() {};
16      CCuadrado(TipoNumerico lado);
17      virtual ~CCuadrado(){};
18      //---metodo de acceso
19      void set_m_Lado(TipoNumerico lado) { m_lado = lado;}
20      TipoNumerico getLado(){ return m_lado;}
21
22      TipoNumerico Apotema();
23      TipoNumerico SemiPerimetro();
24      TipoNumerico Area();
25  };
```

```
10  class CHexagono
11  {
12  private:
13      TipoNumerico m_lado;
14  public:
15      CHexagono() {};
16      CHexagono(TipoNumerico lado);
17      virtual ~CHexagono(){};
18      //---metodo de acceso
19      void set_m_Lado(TipoNumerico lado) { m_lado = lado;}
20      TipoNumerico getLado(){ return m_lado;}
21
22      TipoNumerico Apotema();
23      TipoNumerico SemiPerimetro();
24      TipoNumerico Area();
25  };
```

CTriangulo.cpp

```
#include "CTriangulo.h"

CTriangulo::CTriangulo(TipoNumerico lado)
{
 m_lado = lado;
}

TipoNumerico CTriangulo::Apotema()
{
 return (m_lado * sqrt(3)/6.0);
}

TipoNumerico CTriangulo::SemiPerimetro()
{
 return (3.0 * m_lado/2.0);
}

TipoNumerico CTriangulo::Area()
{
 return(SemiPerimetro()*Apotema());
}
```

CCuadrado.cpp

```
#include "CCuadrado.h"

CCuadrado::CCuadrado(TipoNumerico lado)
{
 m_lado = lado;
}

TipoNumerico CCuadrado::Apotema()
{
 return (m_lado/2.0);
}

TipoNumerico CCuadrado::SemiPerimetro()
{
 return( 2.0*m_lado);
}

TipoNumerico CCuadrado::Area()
{
 return(SemiPerimetro()*Apotema());
}
```

CHexagono.cpp

```
#include "CHexagono.h"

CHexagono::CHexagono(TipoNumerico lado)
{
 m_lado = lado;
}

TipoNumerico CHexagono::Apotema()
{
 return( m_lado*sqrt(3.0)/2.0);
}

TipoNumerico CHexagono::SemiPerimetro()
{
 return(3.0* m_lado);
}

TipoNumerico CHexagono::Area()
{
 return(SemiPerimetro()*Apotema());
}
```

Los métodos Semiperímetro y apotema, coinciden en el Qué, pero no Coinciden en el Cómo.
Se define el método en la clase ancestra como “virtual”
Y en las clases hijas se define como “override”

El método Area() tiene el mismo codigo en los 3 casos, por ello se generaliza y se incluye en la clase ancestra, y las clases hijas lo heredarán.

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Analizamos métodos

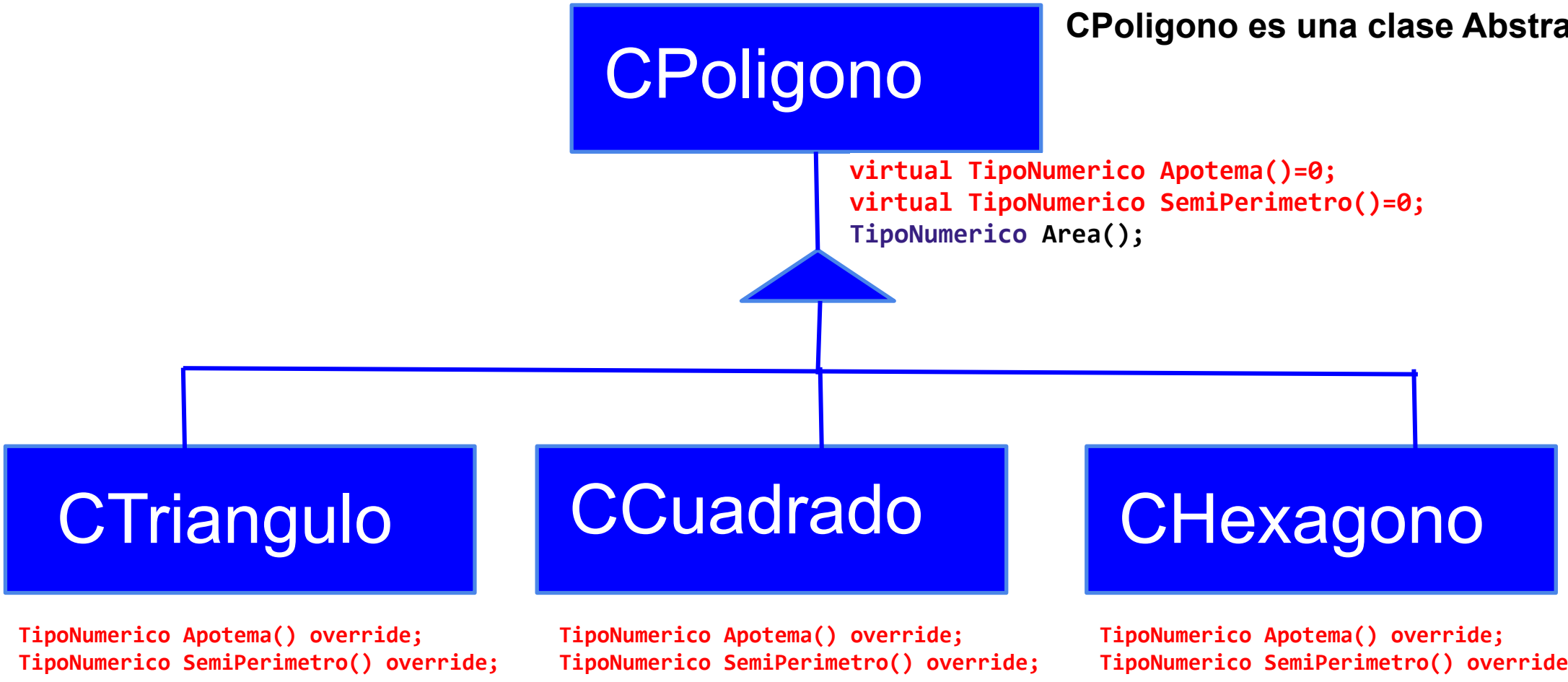
```
 9  class CTriangulo
10  {
11    private:
12      TipoNumerico m_lado;
13    public:
14      CTriangulo() {};
15      CTriangulo(TipoNumerico lado);
16      virtual ~CTriangulo(){};
17      //---metodo de acceso
18      void set_m_Lado(TipoNumerico lado) { m_lado = lado;}
19      TipoNumerico getLado(){ return m_lado;}
20
21      TipoNumerico Apotema();
22      TipoNumerico SemiPerimetro();
23      TipoNumerico Area();
24  };
```

```
10  class CCuadrado
11  {
12  private:
13      TipoNumerico m_lado;
14  public:
15      CCuadrado() {};
16      CCuadrado(TipoNumerico lado);
17      virtual ~CCuadrado(){};
18      //---metodo de acceso
19      void set_m_Lado(TipoNumerico lado) { m_lado = lado;}
20      TipoNumerico getLado(){ return m_lado;}
21
22      TipoNumerico Apotema();
23      TipoNumerico SemiPerimetro();
24      TipoNumerico Area();
25  };
```

```
10  class CHexagono
11  {
12  private:
13      TipoNumerico m_lado;
14  public:
15      CHexagono() {};
16      CHexagono(TipoNumerico lado);
17      virtual ~CHexagono(){};
18      //---metodo de acceso
19      void set_m_Lado(TipoNumerico lado) { m_lado = lado;}
20      TipoNumerico getLado(){ return m_lado;}
21
22      TipoNumerico Apotema();
23      TipoNumerico SemiPerimetro();
24      TipoNumerico Area();
25  };
```

El diseño final quedaría asi…

UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA



CPoligono.h

Clase Ancestra

```
#ifndef PROG_01_CPOLIGONO_H
#define PROG_01_CPOLIGONO_H
#include "Definiciones.h"
class CPoligono
{protected:
   TipoNumerico m_lado;
 public:
   CPoligono(){}
   CPoligono(TipoNumerico lado);
   virtual ~CPoligono(){}
   //---metodo de acceso
   void set_m_Lado(TipoNumerico lado) { m_lado = lado;}
   TipoNumerico getLado(){ return m_lado;}
   virtual TipoNumerico Apotema()=0;
   virtual TipoNumerico SemiPerimetro()=0;
   TipoNumerico Area();
};
#endif //PROG_01_CPOLIGONO_H
```

CPoligono.cpp

```
#include "CPoligono.h"
CPoligono::CPoligono(TipoNumerico lado)
{
 m_lado = lado;
}

TipoNumerico CPoligono::Area()
{
 return ( SemiPerimetro() * Apotema());
}
```

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CTriangulo.h

```
#ifndef PROG_01_CTRIANGULO_H
#define PROG_01_CTRIANGULO_H
#include "Definiciones.h"
#include "CPoligono.h"

class CTriangulo:public CPoligono
{public:
   CTriangulo() {};
   CTriangulo(TipoNumerico lado);
   virtual ~CTriangulo(){};
   TipoNumerico Apotema() override;
   TipoNumerico SemiPerimetro() override;
};
#endif //PROG_01_CTRIANGULO_H
```

CCuadrado.h

```
#ifndef PROG_01_CCUADRADO_H
#define PROG_01_CCUADRADO_H
#include "Definiciones.h"
#include "CPoligono.h"

class CCuadrado:public CPoligono
{public:
   CCuadrado() {};
   CCuadrado(TipoNumerico lado);
   virtual ~CCuadrado(){};
   TipoNumerico Apotema() override;
   TipoNumerico SemiPerimetro() override;
};
#endif //PROG_01_CCUADRADO_H
```

CHexagono.h

```
#ifndef PROG_01_CHEXAGONO_H
#define PROG_01_CHEXAGONO_H
#include "Definiciones.h"
#include "CPoligono.h"

class CHexagono:public CPoligono
{public:
   CHexagono() {};
   CHexagono(TipoNumerico lado);
   virtual ~CHexagono(){};
   TipoNumerico Apotema() override ;
   TipoNumerico SemiPerimetro() override;
};
#endif //PROG_01_CHEXAGONO_H
```

CTriangulo.cpp

```
#include "CTriangulo.h"
CTriangulo::CTriangulo(TipoNumerico lado)
                       :CPoligono(lado)
{
}

TipoNumerico CTriangulo::Apotema()
{
 return (m_lado * sqrt(3)/6.0);
}

TipoNumerico CTriangulo::SemiPerimetro()
{
 return (3.0 * m_lado/2.0);
}
```

CCuadrado.cpp

```
#include "CCuadrado.h"
CCuadrado::CCuadrado(TipoNumerico lado)
                     :CPoligono(lado)
{
}

TipoNumerico CCuadrado::Apotema()
{
 return (m_lado/2.0);
}

TipoNumerico CCuadrado::SemiPerimetro()
{
 return( 2.0*m_lado);
}
```

CHexagono.cpp

```
#include "CHexagono.h"
CHexagono::CHexagono(TipoNumerico lado)
                     :CPoligono(lado)
{
}

TipoNumerico CHexagono::Apotema()
{
 return( m_lado*sqrt(3.0)/2.0);
}

TipoNumerico CHexagono::SemiPerimetro()
{
 return(3.0* m_lado);
}
```

Definiciones.h

```
#ifndef PROG_01_DEFINICIONES_H
#define PROG_01_DEFINICIONES_H

#include <cmath>
typedef double TipoNumerico;
typedef unsigned int EnteroSinSigno;
enum class Opciones{ Triangulo=1, Cuadrado,  Hexagono };

void Menu();
#endif //PROG_01_DEFINICIONES_H
```

main.cpp

```
#include <iostream>
#include "Definiciones.h"
#include "CCuadrado.h"
#include "CTriangulo.h"
#include "CHexagono.h"
using namespace std;

int main()
{EnteroSinSigno Opcion;
 TipoNumerico lado;
 CPoligono *pUnPoligono = nullptr;

 do
 { Menu();
   cout << "Selecciona poligono: ";
   cin >> Opcion;
 }while( Opcion<1 || Opcion>3);

 cout <<"Lado : "; cin >> lado;
 switch(Opciones(Opcion))
 {case Opciones::Triangulo:
     pUnPoligono = new CTriangulo(lado);
     break;
   case Opciones::Cuadrado:
     pUnPoligono = new CCuadrado(lado);
     break;
   case Opciones::Hexagono:
     pUnPoligono = new CHexagono(lado);
 }
 cout << "Apotema       : " << pUnPoligono->Apotema() <<"\n";
 cout << "Semiperimetro : " << pUnPoligono->SemiPerimetro() << "\n";
 cout << "El area es    : " << pUnPoligono->Area()<< "\n";
 delete pUnPoligono;
 pUnPoligono= nullptr;
 return 0;
}
```

```
void Menu()
{ cout << "Seleccione el tipo de poligono que quiere
           crear \n";
 cout << "1. Triangulo \n";
 cout << "2. Cuadrado \n";
 cout << "3. Hexagono \n";
}
```

El programa terminado descargarlo del Github:
https://github.com/Hildiu/PoligonosRegulares.git

Clases Abstractas

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 double getArea() = 0;
```

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

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***, sobreescribir 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.

Explorando lo aprendido

- ¿Qué es el polimorfismo ?
- ¿Cuántos tipos hay de polimorfismo?
- ¿Cuándo una clase es abstracta?

CS1102 – PROGRAMACIÓN ORIENTADA A OBJETOS 1
CICLO 2019-1

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CIENCIA DE LA COMPUTACIÓN

Unidad 8:
Programación Orientada a Objetos - Parte 4
Herencia y polimorfismo

http://bit.ly/2HRBWgq

Profesores:

Ernesto Cuadros- Vargas, PhD. *ecuadros@utec.edu.pe*

María Hilda Bermejo, M. Sc. *mbermejo@utec.edu.pe*

http://bit.ly/325B0Nc