CS1112: Programación 2

Unidad 7: Polimorfismo

Sesión de Teoría - 12

Profesor:
José Antonio Fiestas Iquira jfiestas@utec.edu.pe

Material elaborado por:
Maria Hilda Bermejo, José Fiestas, Rubén Rivas

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Índice:

- Unidad 7: Polimorfismo
  - Polimorfismo paramétrico o genérico
  - Polimorfismo de inclusión herencia
  - Herencia de tipos con Override

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

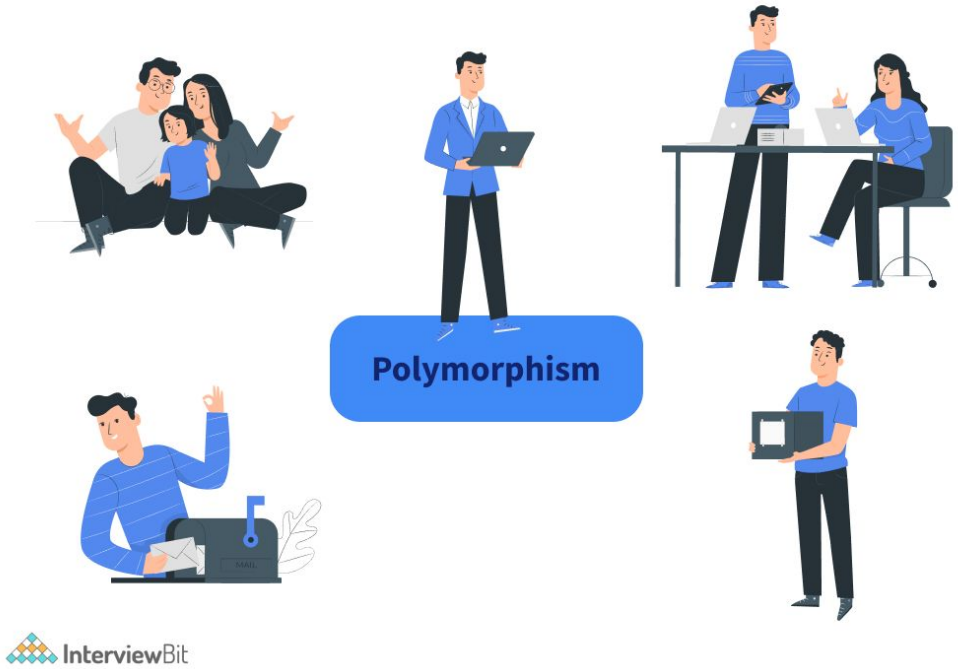
Logro de la sesión:

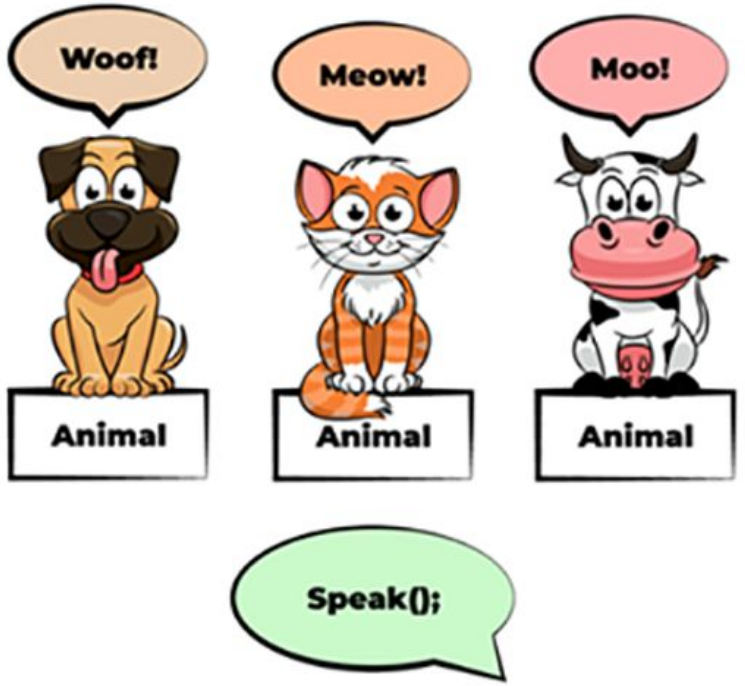
Al finalizar la sesión, los alumnos diseñan e implementan programas orientados a objetos utilizando Polimorfismo.

- Conocerán el Polimorfismo soportado en el lenguaje C++, utilizándolo en casos de herencia para mejorar organización y legibilidad de código.

POLIMORFISMO

Polimorfismo en programación:





UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Definición

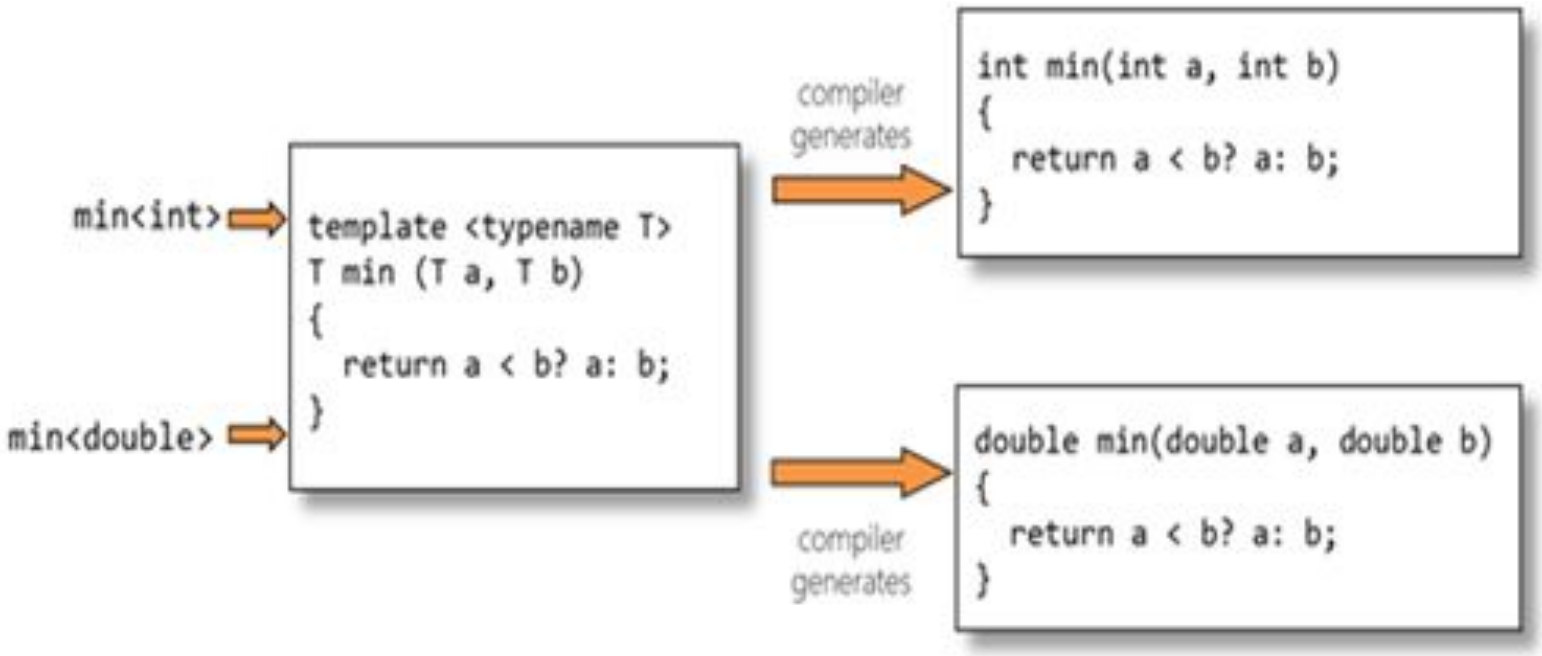
- Un **objeto polimórfico** es una entidad, como una variable o argumento de función, que puede tener valores de tipos diferentes en el curso de ejecución.
- Las **funciones polimórficas** son funciones que tienen argumentos polimórficos.
- Los **tipos polimórficos** son tipos cuyas operaciones son aplicables a valores de más de un tipo.

¿Es la misma o distinta especie?

7.1 Unidad 7: Polimorfismo paramétrico

Sintaxis: A.1 Polimorfismo paramétrico o de tiempo de compilación o generic.

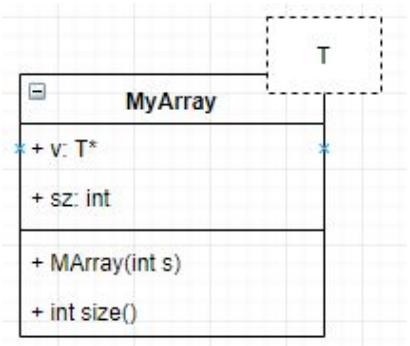
```
template <typename T>
T someFunction(T arg)
{
  ... .. ...
}
```



UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Sintaxis: A.1 Polimorfismo paramétrico o generic.

```
template <typename T>
class className
{
public:
    T var;
    T someOperation(T arg);
};
```




```
template <typename T>
class MyArray {
private:
    T* ptr;
    entero size;
public:
    MyArray(T *arr, entero s);
    T max();
    ~MyArray();
};
```

Para plantillas es recomendable definir la clase y su implementación en un solo archivo ClassName.h sin necesidad de un ClassName.cpp dado que no se compila la plantilla.

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Ejemplo 1:

Construya una función paramétrica para intercambiar números de cualquier tipo.

Funciones.h

```
template <typename T>
void swap(T &a, T &b){
  T temp = a;
  a = b;
  b = temp;
}
```

main.cpp

```
#include <iostream>
#include “Funciones.h”
int main() {
   int x = 20, y = 1;
   double d1 = 3.14, D2=2.79;
   swap(x, y);
   swap(d1, d2);
}
```

Ejemplo 2:

Construya una clase genérica para representar un arreglo de 1 dimensión de cualquier tipo de dato y que tenga una función mensaje para encontrar el máximo valor.

array.h

```
template <typename T>
class MyArray{
private:
    T *ptr;
    int size;
public:
    MyArray(int s);
    MyArray(T arr[], int s);
    T max();
    ~MyArray();
};
```

main.cpp

```
#include <iostream>
#include "array.h"


int main() {
    int arr[5] = {1, 2, 3, 4, 5};
    MyArray<int> a(arr, 5);
    auto x = a.max();
    std::cout << x;
    return 0;
}
```

Ejemplo 2: ...continuación

Construya una clase genérica para representar un arreglo de 1 dimensión de cualquier tipo de dato y que tenga una función mensaje para encontrar el máximo valor.

array.h

```
template <typename T>
MyArray<T>::MyArray(T arr[], entero s) {
    ptr = new T[s];
    size = s;
    for(entero i = 0; i < size; i++)
        ptr[i] = arr[i];
}
template <typename T>
MyArray<T>::~MyArray() {
    delete[] ptr;
    ptr = nullptr;
}
```

array.h

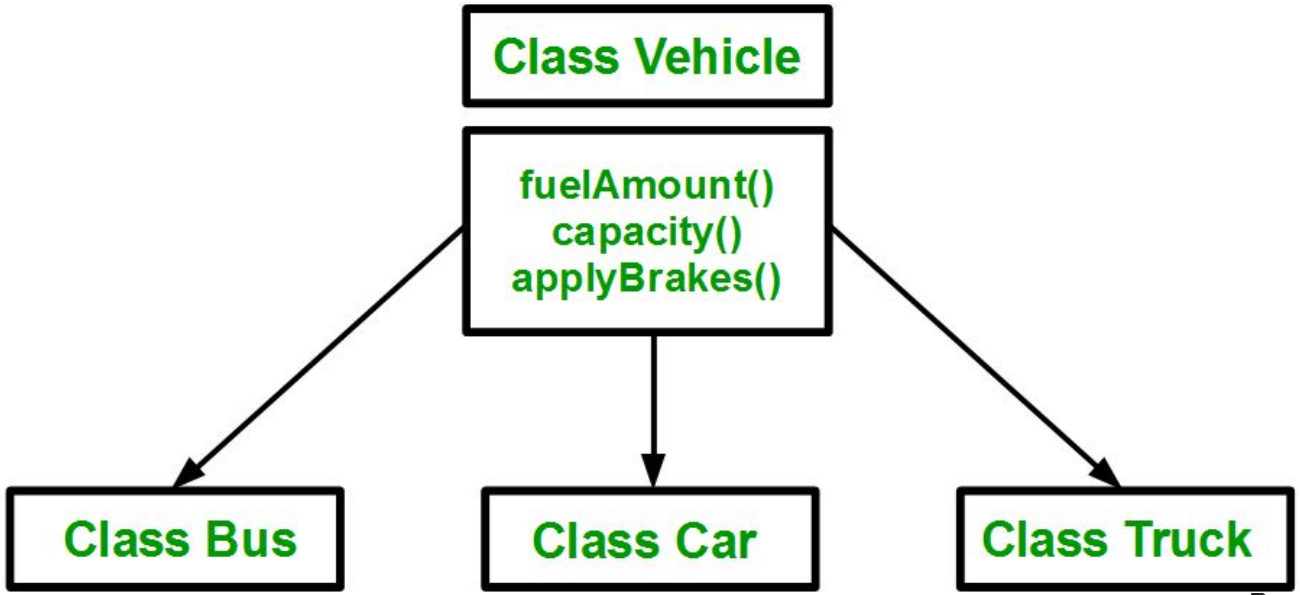
```
template <typename T>
T MyArray<T>::max() {
    T vmax = ptr[0];
    for (entero i = 1; i < size; i++){
        if (ptr[i] > vmax) vmax= ptr[i];
    }
    return vmax;
}
```

7.2

Unidad 7: Polimorfismo de inclusión o herencia

Sintaxis: A.2 Polimorfismo de inclusión o herencia.

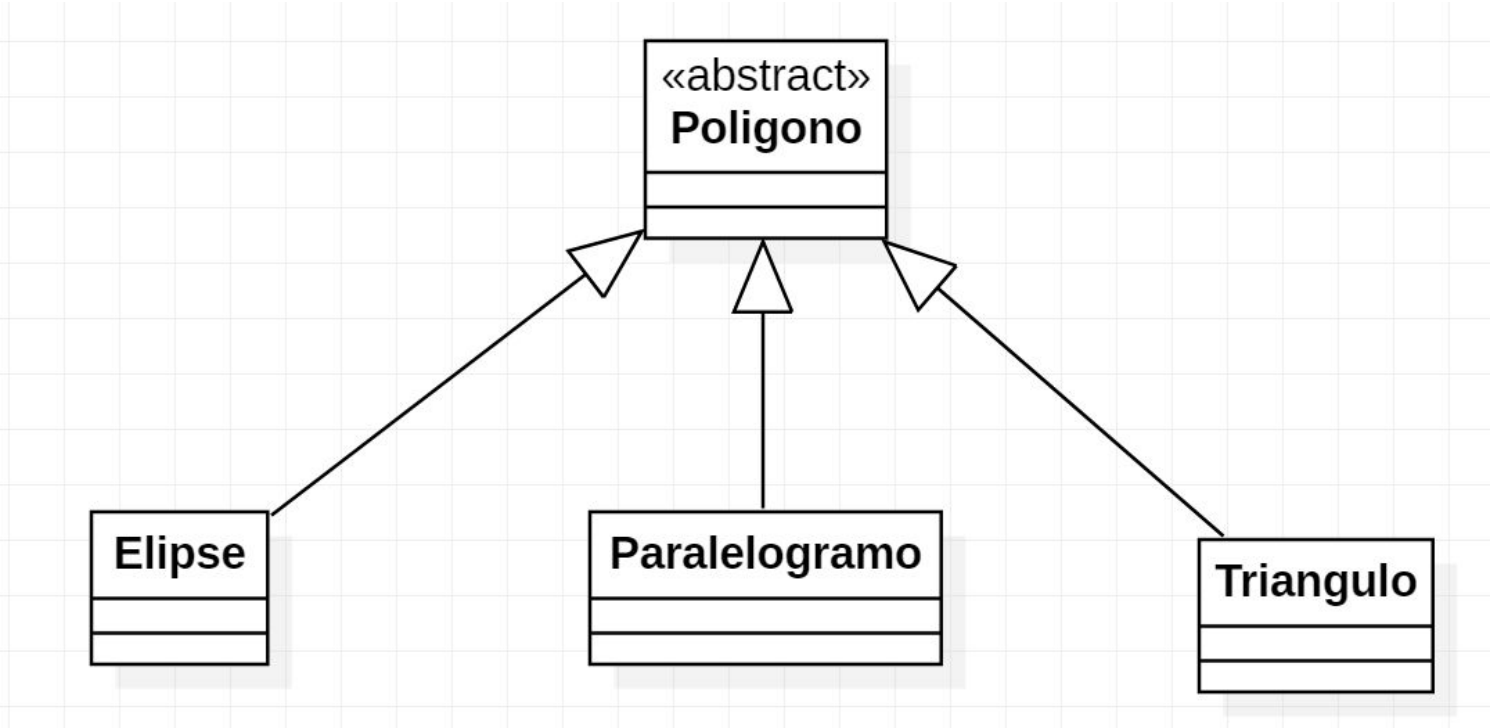
```
class subclass_name : access_mode base_class_name

{

  //body of subclass

};
```



UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Ejemplo 2: Herencia de tipos con Override

Desarrolle un programa que dibuje una serie de figuras geométricas respetando la jerarquía que aparece a continuación.



UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

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"

Ejemplo 2: Herencia de tipos con Override

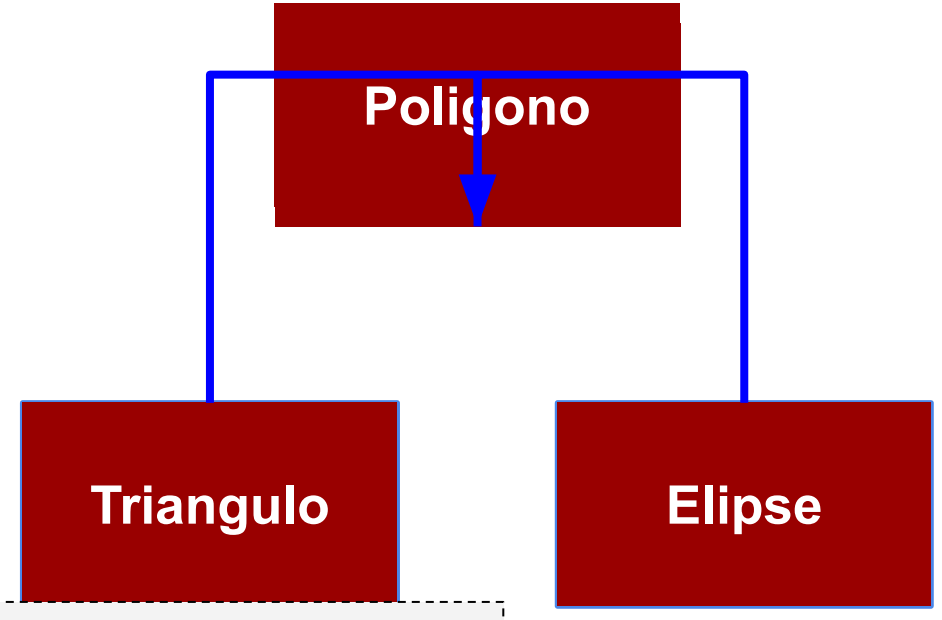
```
#include <iostream>
#include <vector>
using namespace std;

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();
}
```

Ejemplo 2: Herencia de tipos con Override

```
#include <iostream>
#include <vector>
using namespace std;
void funcion1(Poligono *pPol)
{    cout << "Area1 es: " << pPol->calcularArea() << endl; }

void funcion2(Poligono &rPol)
{    cout << "Area2 es: " << rPol.calcularArea() << endl; }

int main()
{vector<Poligono*> poligonos = {new Triangulo(10, 10), new Paralelogramo(20, 30)};
 poligonos.push_back(new Elipse(10, 12));
 size_t i;
 for (i = 0; i < poligonos.size(); i++)
   cout << "Area es: " << poligonos[i]->calcularArea() << endl;
 for (i = 0; i < poligonos.size(); i++)
    funcion1(poligonos[i]);
 for (i = 0; i < poligonos.size(); i++)
    funcion2(*poligonos[i]);
 for (i = 0; i < poligonos.size(); i++)
   delete poligonos[i];
 poligonos.clear();
}
```

Ejemplo 3: El proceso de una clase abstracta

```
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;
};
```

Ejemplo 3: Herencia de tipos con Override

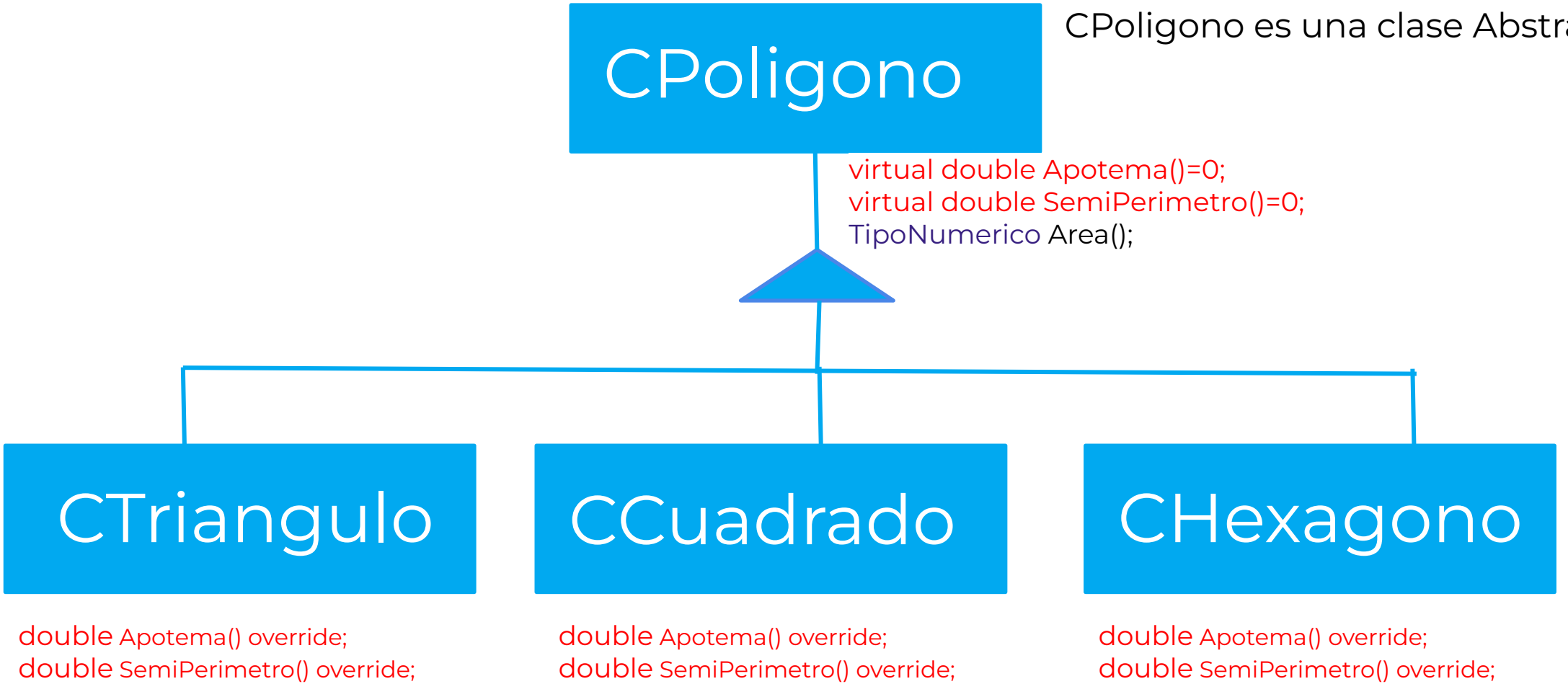
Un polígono regular es aquel que tiene sus lados iguales. Son polígonos regulares por ejemplo: un cuadrado, un triángulo equilátero, un hexágono.
En el ejemplo se quiere calcular el área, el semiperímetro y el apotema de cualquiera de estas figuras.
Se sabe que para calcular el área de cualquiera de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



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  };
```

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);
}
```

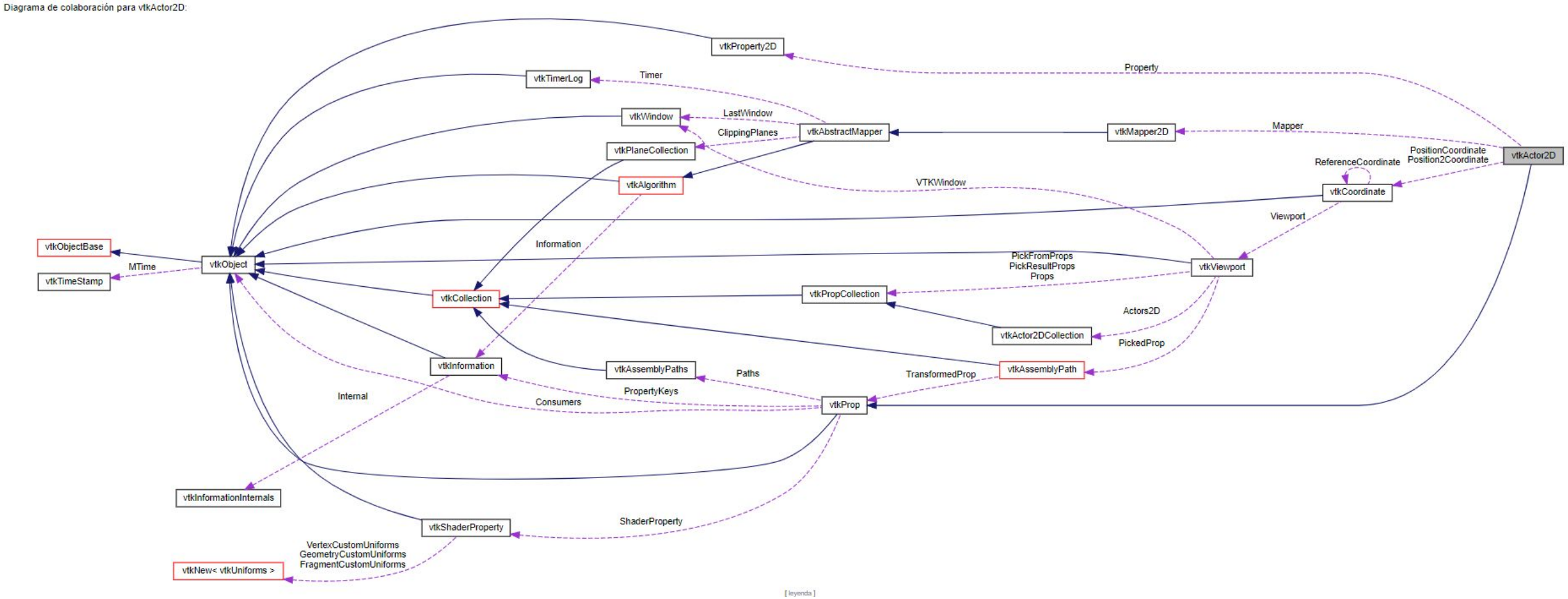
CHexagono.cpp

```
#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
```

Un Ejemplo más real sobre herencia



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

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Resumen

- Aprendimos a usar Polimorfismo soportado en el lenguaje C++.

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Bibliografía:

Deitel. P.J. and Deitel. H. M. (2016) C++ How to Program, Prentice Hall.

Stroustrup, Bjarne (2013). The C++ Programming Language, 4th Addison-Wesley.

Eckel, Bruce, 2000. Thinking in C++, Vol 1: Introduction to Standard C++, 2nd Edition, Prentice Hall

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

¡Nos vemos en la siguiente clase!

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA