

CS1112: Programación 2

Unidad 5: POO (parte 1)

Sesión de Teoría - 8

Profesor:

José Antonio Fiestas Iquira jfiestas@utec.edu.pe

Material elaborado por:

Maria Hilda Bermejo, José Fiestas, Rubén Rivas



Índice:

- Unidad 5: POO (Parte 1)
 - Proceso de abstracción (clase - objeto)
 - Setters y getters (acceso y modificación)
 - Constructores y destructores

5.1

Unidad 5: POO
Abstracción, setters y getters

UTEC

Logro de la sesión:

Al finalizar la sesión, los alumnos se familiarizan con el paradigma de la Programación Orientada a Objetos.

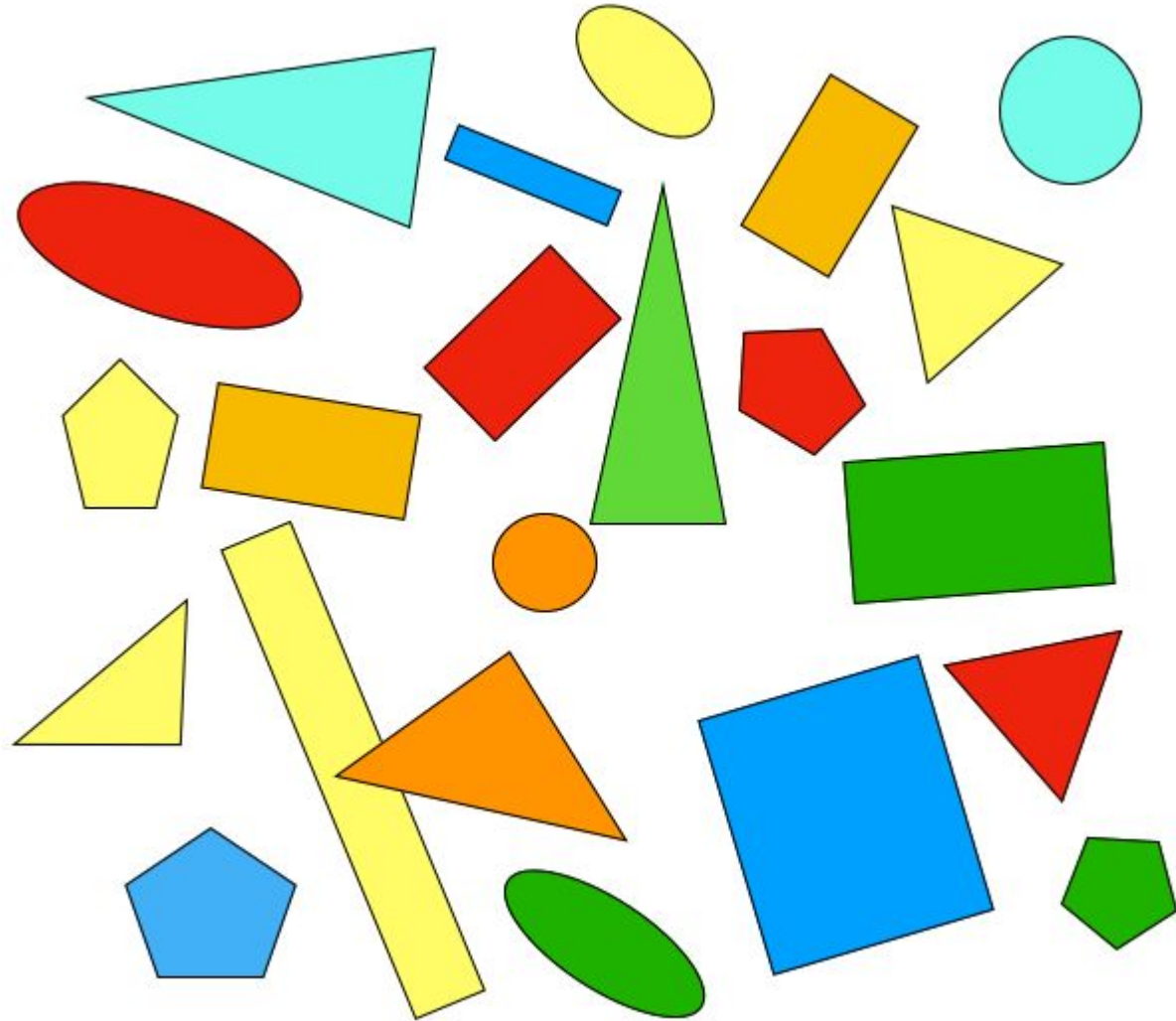
- Proceso de Abstracción
- Clase – Objeto
- Métodos de acceso y modificación (getters y setters)
- Constructores y destructores

Ejercicio Visual

¿Cuántos conjuntos podemos identificar en la siguiente imagen?



SLIDO
133308

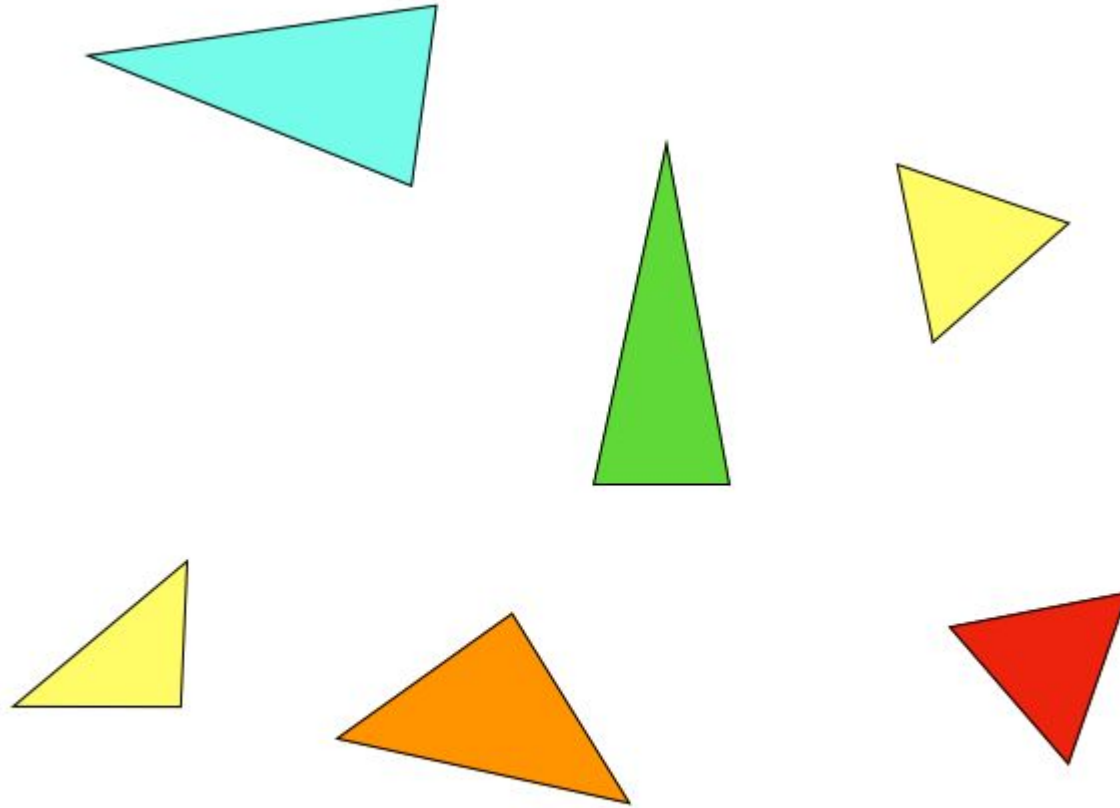


Ejercicio Visual

¿Qué criterios usó
para definir los
conjuntos?



[SLIDO](#)
133308



Definición de clase

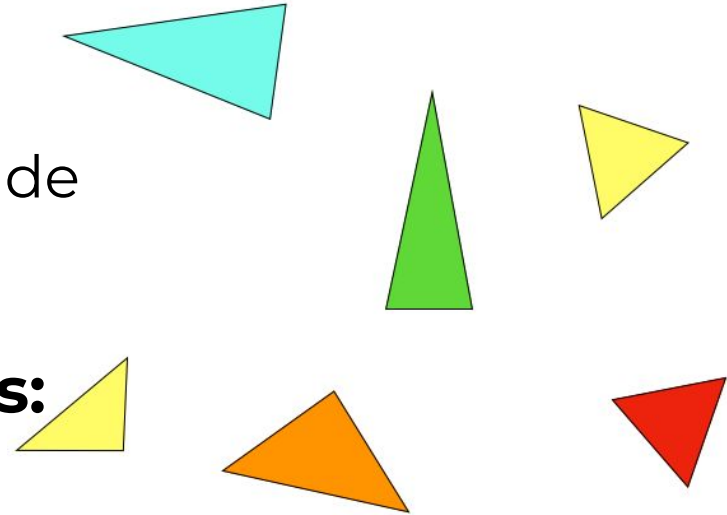
Conjunto → **Clase Triangulo**

propiedades o atributos:

número de lados, color, suma de ángulos internos, longitud de lados

procedimientos o métodos:

calculo de área, cálculo de perímetro

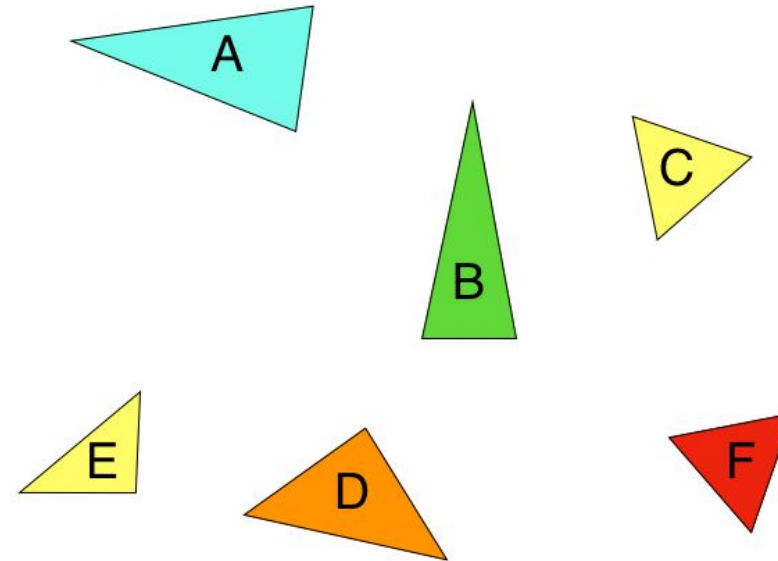
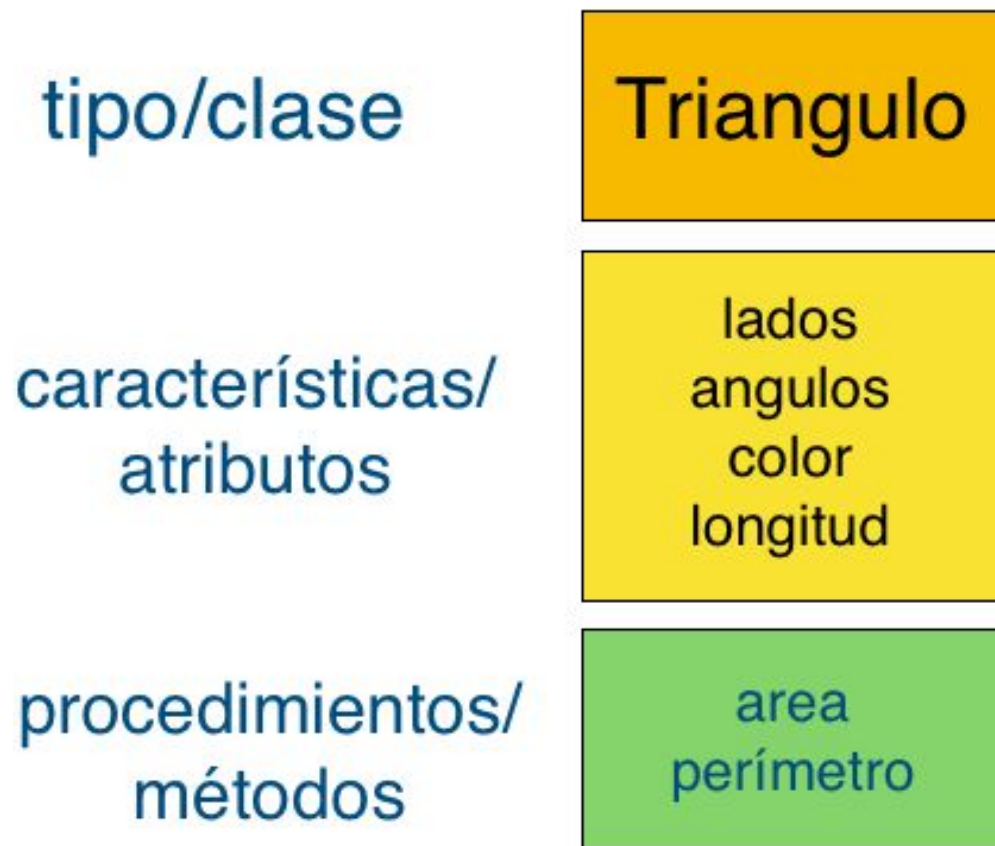


elemento → **instancia de clase: objeto A**

el objeto tendrá valores para cada uno de sus atributos

(Por ejemplo, lados: 3, color: rojo, suma de ángulos: 180, longitud de lados: 9,9,6)

Definición de clase (UML): encapsulando las propiedades y métodos en un modelo abstracto



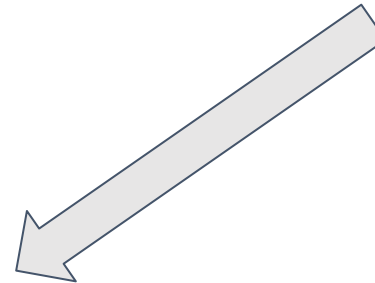
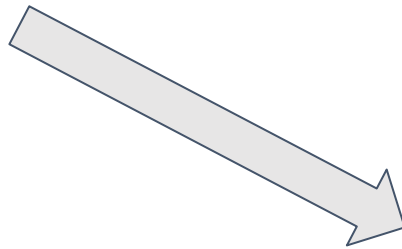
El **objeto A** de **clase Triángulo** tiene los **atributos**:
lados=3, ángulos=180, color=celeste, longitud=9,9,6
y define **métodos de clase** para:
 $\text{area} = \text{base} * \text{altura} / 2$
 $\text{perimetro} = \text{lado1} + \text{lado2} + \text{lado3}$

Encapsulamiento

CLASE

Atributos

Métodos



Ejemplo: BattleBots



¿Qué propiedades y comportamiento en común podemos abstraer?

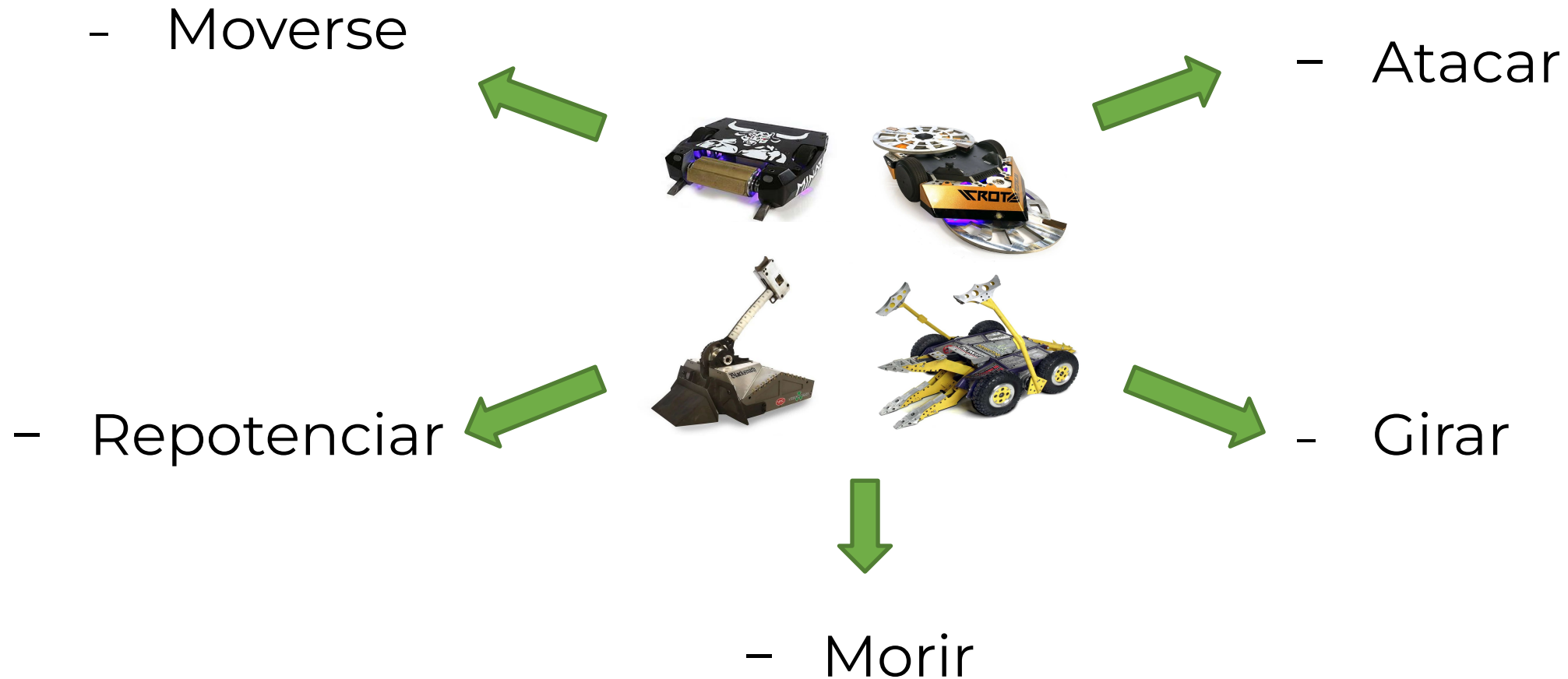


[Ver video](#)

Propiedades (atributos):



Comportamiento (métodos)



Modelo abstracto (UML)

Tipo/Clase

Robot

Características

☐ Atributos

Nombre
Color(es)
Tamaño
Vidas

Comportamiento

☐ Métodos

Moverse
Atacar
Girar
Morir

Pasos para realizar la Abstracción bajo el estilo de POO.

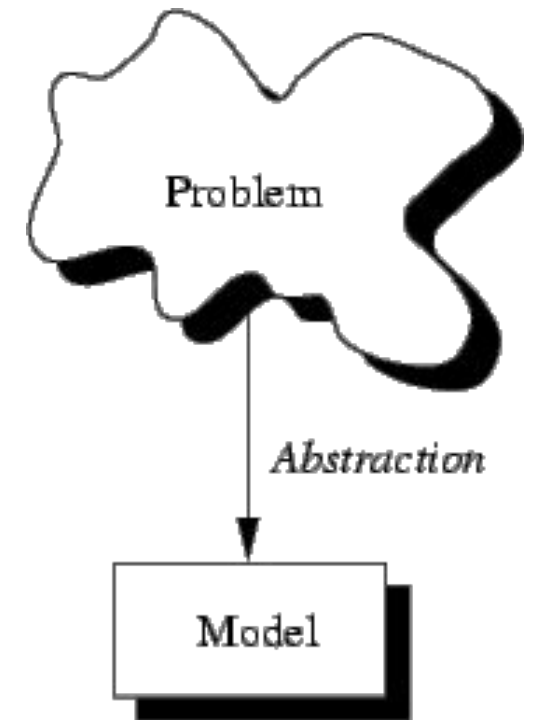


1. **Identificar** los objetos en el problema a resolver
2. **Identificar** propiedades y comportamientos esenciales de un objeto.
3. **Agrupar** objetos con las mismas características en clases.

Ventajas de la Abstracción bajo el estilo de POO.

Describe una **entidad del mundo real**, no importando lo compleja que pueda ser, para luego **utilizar esta descripción en un programa**.

Optimiza el trabajo colaborativo, al usar el **lenguaje de objetos**, de uso cotidiano en diferentes disciplinas.

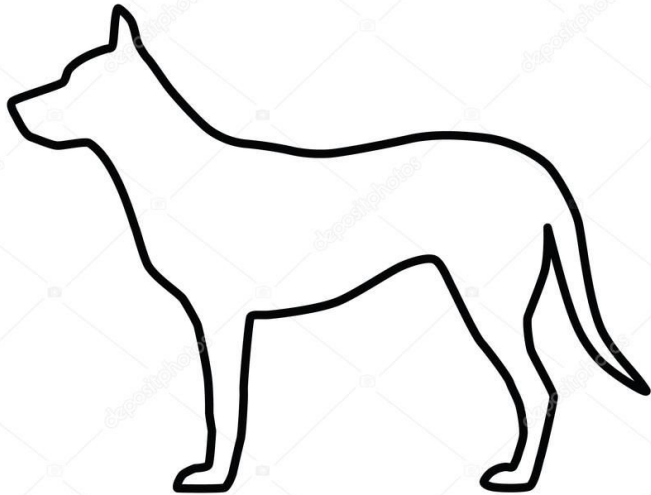


Clase y Objeto

Una Clase:

Es el concepto abstracto de lo que se quiere crear.

Clase Perro



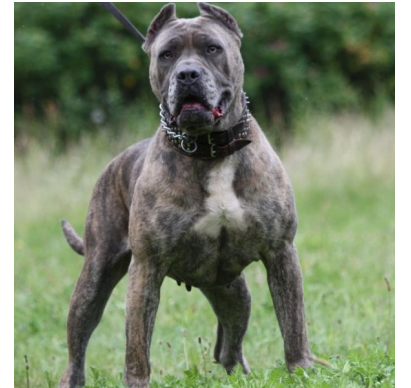
Un Objeto:

Es la instancia de una clase (un ejemplo concreto de una clase).



Simba

Un objeto es un ejemplar de la clase.



Tifón

Un objeto es una instancia de la clase.

Atributos y métodos

Los **atributos** son los datos que caracterizan a los objetos de una clase y determinan el estado de un objeto.

Los **métodos** (funciones) representan las acciones que son capaces de hacer los objetos de una clase.

En la **clase Perro**:

Atributos: edad, talla, peso, nombre.

Métodos : correr, ladrar, morder.

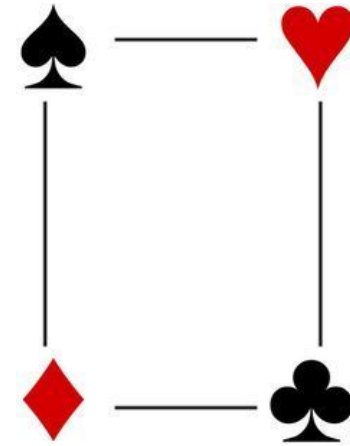
Clase y Objeto

Clase auto



Objetos: ejemplos concretos

Clase carta



Objetos: ejemplos concretos

CLASES Y OBJETOS EN C++

Cómo se define una clase en C++

CPerro

talla
peso
edad
nombre

correr
LadRAR
morder

```
class CPerro  
{
```

```
    private:
```

```
        float  talla;  
        float  peso;  
        int    edad;  
        void   morder();
```

```
    public:
```

```
        string nombre;  
        void   correr();  
        void   ladRAR();
```

```
};
```

Con acceso restringido

Con acceso libre

Cómo se define una clase en C++

CPerro

talla
peso
edad
nombre

correr
LadRAR
morder

```
using decimal = float; //typedef float decimal;  
using age = unsigned int;
```

```
class CPerro
```

```
{
```

```
    private:
```

```
        decimal  talla;  
        decimal  peso;  
        age      edad;  
        void     morder();
```

```
    public:
```

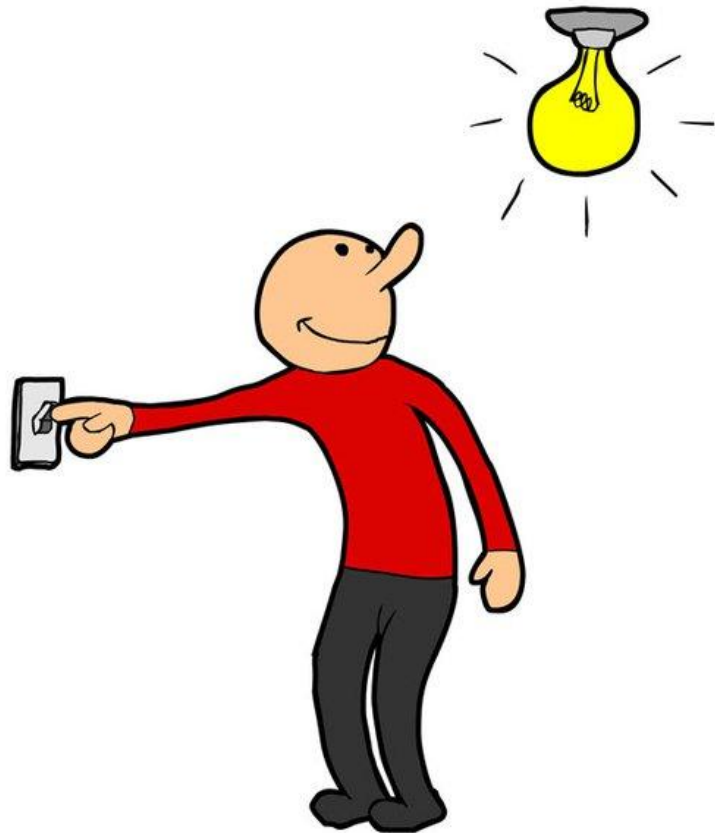
```
        string   nombre;  
        void     correr();  
        void     ladRAR();
```

```
};
```

Con acceso restringido

Con acceso libre

Interfaz: permite acceder a las propiedades de un objeto.



Foco

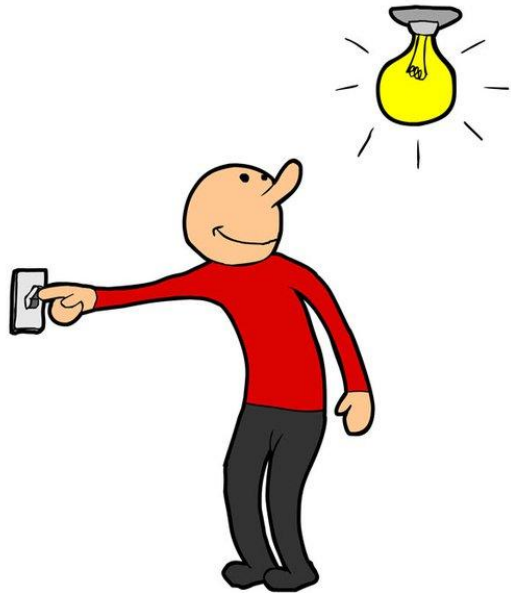
Clase

encender()
apagar()
atenuar()
...

Interfaz
public

Los mensajes/órdenes se hacen a través de la **interfaz**

Interfaz: permite acceder a las propiedades de un objeto.



Foco

Clase

encender()
apagar()
atenuar()
...

Interfaz
public

En C++:

```
Foco philips1;  
philips1.encender();
```

Interfaz: permite acceder a las propiedades de un objeto.



Monitor

Clase

encender()
apagar()
brillo(valor)
...

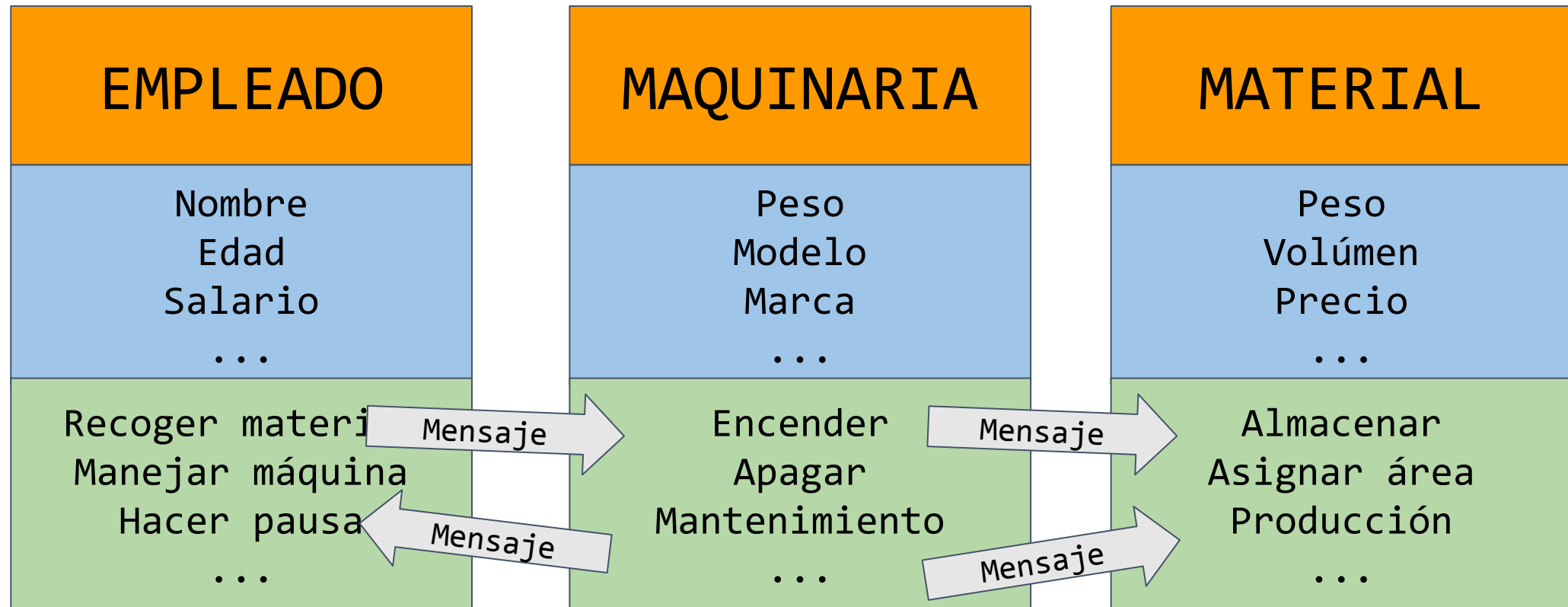
Interfaz
public

En C++:

```
Monitor asus1;  
asus1.brillo(90);
```

Interfaz: permite acceder a las propiedades de un objeto.

Un **programa** es un conjunto de **objetos** interactuando entre sí a través de mensajes.



¿Cómo se instancia un objeto?

CPerro.h

```
typedef float decimal;  
typedef unsigned int age;
```

```
class CPerro  
{  
private:  
    decimal talla;  
    decimal peso;  
    age edad;  
    void morder();  
public:  
    string nombre;  
    void setPeso(int p);  
    void correr();  
    void ladrar();  
};
```

Instanciar un objeto

```
#include "CPerro.h"  
int main()  
{  
    CPerro firulais;  
    firulais.nombre = "Firulais";  
    firulais.setPeso(40);  
    return 0;  
}
```

Ahora ¿Cómo podemos asignar valores a los atributos: talla, peso, edad y nombre ?

- Debemos usar los métodos de acceso (getter & setters) para acceder a los atributos que están con el nivel más elevado de restricción.

¿Cómo se asigna valores a los atributos?

```
typedef float decimal;  
typedef unsigned int tipoEdad;  
class CPerro
```

```
{
```

```
private:
```

```
    decimal m_talla;  
    decimal m_peso;  
    tipoEdad m_edad;  
    void correr();
```

```
public:
```

```
    string m_nombre;  
    void morder();  
    void ladrar();
```

```
    //-----metodos de acceso: setters
```

```
    void setTalla(decimal _talla) {m_talla = _talla;}  
    void setPeso(decimal _peso) {m_peso = _peso;}  
    void setEdad(tipoEdad _edad) {m_edad = _edad;}
```

```
};
```

CPerro.h

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

```
int main()
```

```
{
```

```
    CPerro p1;
```

```
    p1.setTalla(40.75);
```

```
    p1.setPeso(3500);
```

```
    p1.setEdad(3);
```

```
    p1.m_nombre = "Simba";
```

```
    return 0;
```

```
}
```

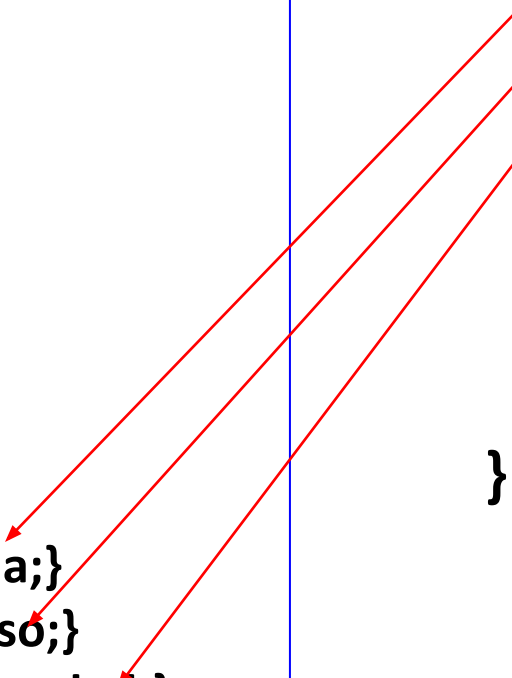
P.D. En el ejemplo falta implementar los métodos correr(), morder(), ladrar();

¿Cómo se asigna valores a los atributos?

```
typedef float decimal;
typedef unsigned int tipoEdad;
class CPerro
{
private:
    decimal m_talla;
    decimal m_peso;
    tipoEdad m_edad;
    void correr();
public:
    string m_nombre;
    void morder();
    void ladrar();
    //-----metodos de acceso: setters
    void setTalla(decimal _talla) {m_talla = _talla;}
    void setPeso(decimal _peso) {m_peso = _peso;}
    void setEdad(tipoEdad _edad) {m_edad = _edad;}
};
```

CPerro.h

```
#include "CPerro.h"
int main()
{
    CPerro p1;
    p1.setTalla(40.75);
    p1.setPeso(3500);
    p1.m_edad = 3; // Error. Private
                  attribute.
    p1.m_nombre = "Simba";
    return 0;
}
```



¿Cómo se asigna valores a los atributos?

```
typedef float decimal;
typedef unsigned int tipoEdad;
class CPerro
{
private:
    decimal m_talla;
    decimal m_peso;
    tipoEdad m_edad;
    void correr();
public:
    string m_nombre;
    void morder();
    void ladrar();
    //-----metodos de acceso: setters
    void setTalla(decimal _talla) {m_talla = _talla;}
    void setPeso(decimal _peso) {m_peso = _peso;}
    void setEdad(tipoEdad _edad) {m_edad = _edad;}
};
```

CPerro.h

```
#include "CPerro.h"
int main()
{
    CPerro p1;
    p1.setTalla(10000.50);
    p1.setPeso(0.0);
    p1.setEdad(3);
    p1.m_nombre = "Simba";
    return 0;
}
```

Serán admitidos



¿Cómo se asigna valores a los atributos?

CPerro.h

```
⋮  
  
void setTalla(decimal _talla) {  
    if(_talla > 0.0 && _talla < 120.0)  
        talla = _talla;  
}  
  
void setPeso(decimal _peso) {  
    if(_peso > 0.0 && _peso <= 80.0)  
        peso = _peso;  
}  
  
⋮
```

#include "CPerro.h"

```
int main()  
{  
    CPerro p1;  
    p1.setTalla(10000.50);  
    p1.setPeso(120);  
    p1.setEdad(3);  
    p1.m_nombre = "Simba";  
    return 0;  
}
```

Serán ignorados

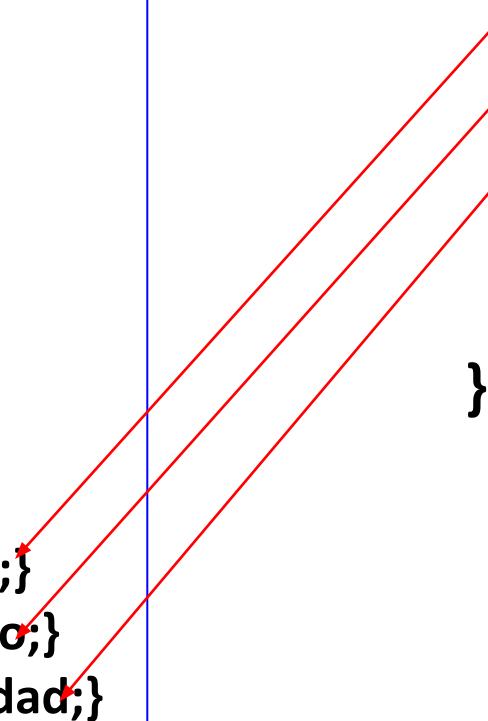


¿Cómo se asigna valores a los atributos?

```
#include <string>
typedef float decimal;
typedef unsigned int tipoEdad;
class CPerro
{
private:
    decimal m_talla;
    decimal m_peso;
    tipoEdad m_edad;
    void correr();
public:
    std::string m_nombre;
    void morder();
    void ladrar();
    void setTalla(decimal talla) {CPerro::m_talla = talla;}
    void setPeso(decimal peso) {CPerro::m_peso = peso;}
    void setEdad(tipoEdad edad) {CPerro::m_edad = edad;}
};
```

CPerro.h

```
#include "CPerro.h"
int main() {
    CPerro P1;
    P1.setTalla(40.75);
    P1.setPeso(3500);
    P1.setEdad(3);
    P1.m_nombre = "Lucas";
    return 0;
}
```



¿Cómo se obtiene los valores de los atributos?

```
class CPerro
```

```
CPerro.h
```

```
{  
    private:  
        decimal m_talla;  
        decimal m_peso;  
        tipoEdad m_edad;  
        void correr();  
    public:  
        string m_nombre;  
        void morder();  
        void ladrar();  
        //-----metodos de acceso: setters  
        void setTalla(decimal _talla) {m_talla = _talla;}  
        void setPeso(decimal _peso) {m_peso = _peso;}  
        void setEdad(tipoEdad _edad) {m_edad = _edad;}  
        //-----metodos de acceso: getters  
        decimal getTalla() {return m_talla;}  
        decimal getPeso() {return m_peso;}  
        tipoEdad getEdad() {return m_edad;}  
};
```

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

```
int main() {  
    CPerro p1;  
    p1.setTalla(40.75);  
    p1.setPeso(3500);  
    p1.setEdad(3);  
    p1.m_nombre = "Simba";  
    cout << "El perro tiene"  
          << p1.getEdad() << "años";  
  
    CPerro *p2 = new CPerro();  
    p2->setTalla(25.05);  
    (*p2).setTalla(25.05);  
    p2->setPeso(1500);  
    p2->setEdad(1);  
    p2->m_nombre = "Lucky";  
    cout << "El perro tiene"  
          << p2->getEdad() << "años";  
    delete p2;  
    p2=nullptr;  
    return 0;  
}
```

¿Cómo se obtiene los valores de los atributos?

```
class CPerro
```

```
{  
    private:  
        decimal m_talla;  
        decimal m_peso;  
        tipoEdad m_edad;  
        void correr();  
    public:  
        string m_nombre;  
        void morder();  
        void ladrar();  
        //-----metodos de acceso: setters  
        void setTalla(decimal _talla) {m_talla = _talla;}  
        void setPeso(decimal _peso) {m_peso = _peso;}  
        void setEdad(tipoEdad _edad) {m_edad = _edad;}  
        //-----metodos de acceso: getters  
        decimal getTalla() { return m_talla; }  
        decimal getPeso() {return m_peso; }  
        tipoEdad getEdad() { return m_edad; }  
};
```

CPerro.h

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

```
int main() {  
    CPerro      p3;  
    CPerro*     p = nullptr;  
    CPerro &rp1 = p3;  
  
    p = &p3;  
    CPerro &rp2 = *p;  
    p3.setTalla(40.75);  
    p->setTalla(40.75);  
    rp1.setPeso(3500); //p3.setPeso(3500);  
    rp2.setEdad(3); //p3.setPeso(3500);  
    p->m_nombre = "Simba";  
    cout << "El perro tiene"  
          << rp1.getEdad() << "años";  
    return 0;  
}
```

¿Cómo se obtiene los valores de los atributos?

```
class CPerro
```

```
{
```

```
private:
```

```
    decimal m_talla;
```

```
    decimal m_peso;
```

```
    tipoEdad m_edad;
```

```
    void correr();
```

```
    std::string m_nombre;
```

```
public:
```

```
    void morder();
```

```
    void ladrar();
```

```
    void setTalla(decimal Talla) {CPerro::m_talla = Talla;}
```

```
    void setPeso(decimal Peso) {CPerro::m_peso = Peso;}
```

```
    void setEdad(tipoEdad Edad) {CPerro::m_edad = Edad;}
```

```
    void setNombre(std::string _nombre) {CPerro::m_nombre = _nombre;}
```

```
    decimal getTalla() const {return m_talla;}
```

```
    decimal getPeso() const {return m_peso;}
```

```
    tipoEdad getEdad() const {return m_edad;}
```

```
    const std::string &getNombre() const {return m_nombre;}
```

```
};
```

```
CPerro.h
```

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

```
int main() {
```

```
    CPerro P1;
```

```
    P1.setTalla(40.75);
```

```
    P1.setPeso(3500);
```

```
    P1.setEdad(3);
```

```
    P1.setNombre("Lucas");
```

```
    std::cout << "El perro " << P1.getNombre()
    << " tiene " << P1.getEdad() << " años.";
```

```
    std::cout << P1.talla; // Error. Private
                           attribute.
```

```
    std::cout << P1.getNombre();
```

```
    return 0;
```

```
}
```

Ejemplo:

Escriba un programa Orientado a Objetos - POO, que permita hallar el el área y el perímetro de un rectángulo.

Se diseñará la clase CRectangulo.

El proyecto tiene código en 3 archivos:

main.cpp

CRectangulo.h

CRentangulo.cpp

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H
typedef float decimal;

class CRectangulo
{
private:
    decimal largo;
    decimal ancho;
public:
    decimal area();
    decimal perimetro();
    //---metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setLargo(decimal _largo)
    { largo = (_largo > 0)? _largo : largo; }
    void setAncho(decimal _ancho)
    { ancho = (_ancho > 0)? _ancho : ancho; }
    //--- getters
    decimal getLargo() { return largo; }
    decimal getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

decimal CRectangulo::area() {
    return (largo*ancho);
}

decimal CRectangulo::perimetro()
{
    return( 2*largo + 2*ancho);
}
```

main.cpp

```
#include <iostream>
#include "CRectangulo.h"
using namespace std;

int main()
{ CRectangulo R1, Rx;

    R1.setLargo(110.49);
    R1.setAncho(55.25);
    cout << "El area es " << R1.area() << "\n";
    cout << "El perimetro es " << R1.perimetro();
    return 0;
}
```

Ahora, vamos a crear un segundo objeto de la clase CRectangulo, y los datos para los atributos los leemos desde el teclado.

```

#include <iostream>
#include "CRectangulo.h"
using namespace std;

int main()
{ CRectangulo R1;
  R1.setLargo(110.49);
  R1.setAncho(55.25);
  cout << "\nArea y perimetro de R1 \n";
  cout << "El area    : " << R1.area() << "\n";
  cout << "El perimetro : " << R1.perimetro();

  CRectangulo R2;
  decimal l,a;
  cout << "\n\nDatos para el segundo rectangulo \n";
  cout << "Largo : ";
  cin >> l;
  cout << "Ancho : ";
  cin >> a;
  R2.setLargo(l);
  R2.setAncho(a);
  cout << "\nArea y perimetro de R2 \n";
  cout << "El area    : " << R2.area() << "\n";
  cout << "El perimetro : " << R2.perimetro();
  return 0;
}

```

Pantalla de salida del programa:

Area y perimetro de R1

El area : 6104.57

El perimetro : 331.48

Datos para el segundo rectángulo

Largo : 19

Ancho : 8

Area y perimetro de R2

El area : 152

El perimetro : 54

Importante:

¿Por qué es necesario utilizar los métodos de acceso: setter y getters?

¿Qué es mejor, declarar los atributos en la zona private de la clase, o en la zona public de la clase?

5.2

Unidad 5: POO Constructores y destructores

UTEC

CONSTRUCTORES

El constructor:

- Un constructor es una función especial que tiene el “**mismo nombre**” que el de la clase, sin un tipo de retorno (no se debe especificar “void”).
- Como su nombre lo indica, el constructor se llama cada vez que se declara una instancia (o se construye).
- Si no se define ningún constructor en la clase, el compilador inserta un constructor predeterminado, que no toma ningún argumento y no hace nada, es decir:

```
ClassName :: ClassName () {}
```

El constructor:

- Sin embargo, si se define uno (o más) constructores, el compilador no insertará el constructor por defecto. Si es que se requiere, debe definir explícitamente el constructor por defecto.
- El constructor por defecto suele ser sin parámetros. O en el cuerpo del método inicializar los atributos con valores fijos.

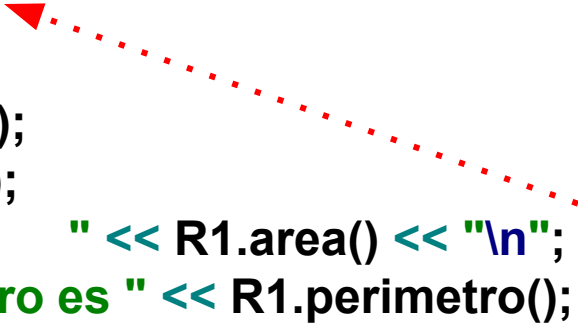
El constructor:

main.cpp

```
#include <iostream>
#include "CRectangulo.h"
using namespace std;

int main()
{ CRectangulo R1;

  R1.setLargo(110.49);
  R1.setAncho(55.25);
  cout << "El area es      " << R1.area() << "\n";
  cout << "El perimetro es " << R1.perimetro();
  return 0;
}
```



En el ejemplo anterior, no se ha declarado un constructor, entonces se está utilizando el constructor por defecto.

El constructor:

Lista de inicializadores de atributos:

- Cuando se declara el constructor, se utiliza para inicializar los miembros de datos de las instancias creadas. La sintaxis es:

```
//----- En el archivo *.h o header
// Declaración del constructor dentro de la declaración de la clase
class CRectangulo {
    CRectangulo(decimal _largo, decimal _ancho);    // solo prototipo
}
```

```
//----- En el archivo *.cpp
// Implementación del Constructor - identificado via operator alcance
ClassName::ClassName(decimal _largo, decimal _ancho) {
    // Cuerpo del constructor
}
```

Ahora declaramos el constructor:

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H
typedef float decimal;
typedef float tipoArea;

class CRectangulo
{
private:
    decimal largo;
    decimal ancho;
public:
    CRectangulo(decimal _largo, decimal _ancho);
    decimal area();
    decimal perimetro();
    //----metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setLargo(decimal _largo)
    { largo = (_largo > 0)? _largo : largo; }
    void setAncho(decimal _ancho)
    { ancho = (_ancho > 0)? _ancho : ancho; }
    //--- getters
    decimal getLargo() { return largo; }
    decimal getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

CRectangulo::CRectangulo(decimal _largo, decimal _ancho)
{
    largo = _largo;
    ancho = _ancho;
}

tipoArea CRectangulo::area()
{
    return (largo*ancho);
}

decimal CRectangulo::perimetro()
{
    return( 2*largo + 2*ancho);
}
```

main.cpp

```
#include <iostream>
#include "CRectangulo.h"
using namespace std;
int main()
{ CRectangulo R1(43.5,22.5);

    cout <<"El area del segundo rectangulo es "
          << R1.area() << "\n";
    cout <<"El perimetro del segundo rectangulo es "
          << R1.perimetro() << "\n";
    return 0;
}
```

Ahora usando "this":

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H
typedef float decimal;
typedef float tipoArea;

class CRectangulo
{
private:
    decimal largo;
    decimal ancho;
public:
    CRectangulo(decimal largo, decimal ancho);
    tipoArea area();
    decimal perimetro();
    //----metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setLargo(decimal _largo)
    { largo = (_largo > 0)? _largo : largo; }
    void setAncho(decimal _ancho)
    { ancho = (_ancho > 0)? _ancho : ancho; }
    //--- getters
    decimal getLargo() { return largo; }
    decimal getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

Note que los nombres de los parámetros del constructor son iguales a los nombres de los atributos.

CRectangulo.cpp

```
#include "CRectangulo.h"

CRectangulo::CRectangulo(decimal largo, decimal ancho)
{
    this->largo = largo;
    this->ancho = ancho;
}

tipoArea CRectangulo::area()
{
    return (largo*ancho);
}

decimal CRectangulo::perimetro()
{
    return ( 2*largo + 2*ancho);
}
```

main.cpp

```
#include <iostream>
#include "CRectangulo.h"
using namespace std;
int main()
{ CRectangulo R(43.5,22.5);
  //CRectangulo R3;

  cout <<"El area del segundo rectangulo es "
        << R.area() << "\n";
  cout <<"El perimetro del segundo rectangulo es "
        << R.perimetro() << "\n";
  return 0;
}
```

SOBRECARGA DE CONSTRUCTORES

Sobrecarga del constructor

- Cualquier función (incluyendo el constructor) puede tener muchas versiones, diferenciadas por su lista de parámetros (número, tipos y orden de los parámetros).
- La invocación o llamada a la función puede optar por invocar una versión determinada haciendo coincidir la lista de parámetros.

La clase puede tener más de un constructor, con una cantidad diferente de parámetros.

Sobrecarga del constructor

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H
#include "Tipos.h"

class CRectangulo
{
private:
    decimal largo;
    decimal ancho;
public:
    CRectangulo(): largo(0), ancho(0){ }
    CRectangulo(decimal _largo, decimal _ancho):
        largo(_largo), ancho(_ancho){ }

    tipoArea area();
    //----metodos de acceso
    //--- setter
    void setLargo(decimal _largo) { largo = _largo; }
    void setAncho(decimal _ancho) { ancho= _ancho; }
    //--- getters
    decimal getLargo() { return largo; }
    decimal getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

#include "CRectangulo.h"

CRectangulo.cpp

```
tipoArea CRectangulo::area()
{
    return (largo*ancho);
}
```

#include <iostream>

#include "CRectangulo.h"

main.cpp

using namespace std;

int main()

```
{
    CRectangulo R1;
    cout <<"Rectangulo 1 \n";
    cout <<"El area    : " << R1.area() << "\n";
```

```
    CRectangulo R2;
    R2.setLargo(23.45);
    R2.setAncho(12.09);
    cout <<"Rectangulo 2 \n";
    cout <<"El area    : " << R2.area() << "\n";
```

```
    CRectangulo R3(43.5,22.5);
    cout <<"Rectangulo 3 \n";
    cout <<"El area    : " << R3.area() << "\n";
    return 0;
}
```

Rectangulo 1
El area : 0

Rectangulo 2
El area : 283.51

Rectangulo 3
El area : 978.75

DESTRUCTOR

Destructor:

- Similar a un constructor, un **destructor** tiene el mismo nombre como el de la clase, pero precedido con una tilde (~).
- El **destructor** es llamado automáticamente cuando la instancia expira.
- No tiene argumentos y no retorna un tipo.
- Solo debe haber un destructor en una clase.
- Si no se ha definido un destructor, el compilador provee un destructor que no hace nada.
- El **destructor** realizará la limpieza de la memoria, en particular, si la memoria ha sido asignada dinámicamente.
- Si el constructor usa **new** para asignar dinámicamente almacenamiento, el destructor deberá usar **delete** para eliminarlo.

Destructor:

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H
#include "Tipos.h"

class CRectangulo
{
private:
    decimal largo;
    decimal ancho;
public:
    CRectangulo(){};
    CRectangulo(decimal _largo, decimal _ancho):
        largo(_largo), ancho(_ancho){};

    virtual ~CRectangulo();
    tipoArea area();
    //----metodos de acceso
    //--- setter
    void setLargo(decimal _largo) { largo = _largo; }
    void setAncho(decimal _ancho) { ancho = _ancho; }
    //--- getters
    decimal getLargo() { return largo; }
    decimal getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

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

CRectangulo.cpp

```
tipoArea CRectangulo::area()
{
    return (largo*ancho);
}

CRectangulo::~~CRectangulo()
{
    std::cout << "\nDestruyendo objeto. \n";
}
```

```
#include "CRectangulo.h"
using namespace std;
```

main.cpp

```
int main()
{
    CRectangulo R1;
    R1.setLargo(23.45);
    R1.setAncho(12.09);
    cout << "Rectangulo 1 \n";
    cout << "El area    : "
        << R1.area() << "\n";
    return 0;
}
```

Rectangulo 1
El area : 283.51

Destruyendo objeto.

Tema para ir estudiando:
USO DE PLANTILLAS

Definamos primero nuestro archivo de tipos de

Tipos.h

```
typedef float decimal;  
typedef float tipoArea;
```

Clase CRectangulo

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H
#include "Tipos.h"

class CRectangulo
{
private:
    decimal largo;
    decimal ancho;
public:
    CRectangulo(decimal largo, decimal ancho);
    tipoArea area();
    decimal perimetro();
    //----metodos de acceso
    //--- setter
    void setLargo(decimal _largo) { largo = _largo; }
    void setAncho(decimal _ancho) { ancho= _ancho; }
    //--- getters
    decimal getLargo() { return largo; }
    decimal getAncho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

CRectangulo::CRectangulo(decimal largo, decimal ancho)
{
    this->largo = largo;
    this->ancho = ancho;
}

tipoArea CRectangulo::area()
{
    return (largo*ancho);
}

decimal CRectangulo::perimetro()
{
    return ( 2*largo + 2*ancho);
}
```

Segunda clase CCirculo

CCirculo.h

```
#ifndef CIRCULO_00_CCIRCULO_H
#define CIRCULO_00_CCIRCULO_H
#include "Tipos.h"

class CCirculo
{
private:
    decimal radio;
    const decimal PI = 3.1416;
public:
    CCirculo(decimal radio);
    tipoArea area();
    decimal perimetro();
    //----metodos de acceso
    //--- setter
    void setRadio(decimal _radio) { radio = _radio; }
    //--- getter
    decimal getRadio() { return radio; }
};

#endif //CIRCULO_00_CCIRCULO_H
```

CCirculo.cpp

```
#include "CCirculo.h"

CCirculo::CCirculo(decimal radio)
{
    this->radio = radio;
}

tipoArea CCirculo::area()
{
    return this->PI * this->radio * this->radio;
}

decimal CCirculo::perimetro()
{
    return this->PI * this->radio * 2;
}
```

Ahora implementemos funciones para imprimir datos de cada figura

Funciones.h

```
#include <iostream>
#include "CRectangulo.h"
#include "CCirculo.h"
using namespace std;

void imprimirDatos(CRectangulo &R)
{
    cout << "El area del rectangulo es "
          << R.area() << "\n";
    cout << "El perimetro del rectangulo es "
          << R.perimetro();
}

void imprimirDatos(CCirculo &C)
{
    cout << "El area del circulo es "
          << C.area() << "\n";
    cout << "El perimetro circulo es "
          << C.perimetro();
}
```

main.cpp

```
#include "Funciones.h"

int main()
{
    CRectangulo R(43.5, 22.5);
    CCirculo C(10.5);

    imprimirDatos(R);
    imprimirDatos(C);


    return 0;
}
```

Las dos funciones hacen lo mismo,
solo varía el tipo de dato.

Entonces reemplacemos ambas funciones por una sola

```
void imprimirDatos(CRectangulo &R)
{
    cout <<"El area del rectangulo es "
          << R.area() << "\n";
    cout <<"El perimetro del rectangulo es "
          << R.perimetro();
}
```

```
void imprimirDatos(CCirculo &C)
{
    cout <<"El area del circulo es "
          << C.area() << "\n";
    cout <<"El perimetro circulo es "
          << C.perimetro();
}
```



```
template <typename T>
void imprimirDatos(T &R)
{
    cout <<"El area de la figura es "
          << R.area() << "\n";
    cout <<"El perimetro de la figura es "
          << R.perimetro();
}
```

Las plantillas nos permiten generalizar funciones para cualquier tipo de dato

Funciones.h

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

template <typename T>
void imprimirDatos(T &R)
{
    cout << "El area de la figura es "
        << R.area() << "\n";
    cout << "El perimetro de la figura es "
        << R.perimetro();
}
```

main.cpp

```
#include "CRectangulo.h"
#include "CCirculo.h"
#include "Funciones.h"

int main()
{
    CRectangulo R(43.5, 22.5);
    CCirculo C(10.5);

    imprimirDatos< CRectangulo >(R);
    imprimirDatos< CCirculo >(C);

    return 0;
}
```

Tema para ir estudiando:
TIPOS DE DATOS
ABSTRACTOS

Ahora generalicemos el tipo de dato de largo y ancho del rectángulo y además generalizamos del area

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H

template <typename T, typename A>
class CRectangulo
{
private:
    T largo;
    T ancho;
public:
    CRectangulo(T largo, T ancho);
    A area();
    T perimetro();
    //---metodos de acceso
    //--- setter
    void setLargo(T _largo) { largo = _largo; }
    void setAncho(T _ancho) { ancho = _ancho; }
    //--- getter
    T getLargo() { return largo; }
    T getAncho() { return ancho; }
};
```

CRectangulo.cpp

```
#include "CRectangulo.h"

template <typename T, typename A>
CRectangulo<T,A>::CRectangulo(T largo, T ancho)
{
    this->largo = largo;
    this->ancho = ancho;
}

template <typename T, typename A>
A CRectangulo<T,A>::area()
{
    return (largo*ancho);
}

template <typename T, typename A>
T CRectangulo<T,A>::perimetro()
{
    return( 2*largo + 2*ancho);
}
```

Ahora generalicemos el tipo de dato de largo y ancho del rectángulo

Funciones.h

```
#ifndef UNTITLED_FUNCIONES_H
#define UNTITLED_FUNCIONES_H

#include <iostream>
#include "CRectangulo.h"
using namespace std;

template <typename T, typename A>
void imprimirDatos<T,A>(CRectangulo<T,A> &R);
{
    cout <<"El area del rectangulo es " << R.area() << "\n";
    cout <<"El perimetro del rectangulo es " << R.perimetro();
}

template <typename T,typename A>
CRectangulo<T,A> crearRectangulo(T largo, T ancho)
{
    CRectangulo<T,A> R(largo, ancho);
    return R;
}

#endif //UNTITLED_FUNCIONES_H
```

main.cpp

```
#include "Funciones.h"
#include "Tipos.h"

int main()
{
    decimal largo;
    decimal ancho;
    cin >> largo >> ancho;
    auto R = crearRectangulo<decimal, tipoArea>(largo, ancho);
    imprimirDatos(R);

    return 0;
}
```

Resumen: Preguntas

- ✓ ¿En qué consiste el proceso de abstracción?
- ✓ ¿Qué es un objeto?
- ✓ ¿Qué es una clase?
- ✓ ¿En qué zona de la clase es recomendable que se declaren los atributos?
- ✓ ¿Cuándo se usa this?
- ✓ ¿Por qué es necesario utilizar métodos de acceso?
- ✓ ¿Una clase puede tener varios constructores?
- ✓ ¿Una clase puede tener más de un destructor?

¡Nos vemos en la
siguiente clase!

