

Unidad 6:

Programación Orientada a Objetos - Parte 1

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

Profesores:

Ernesto Cuadros- Vargas, PhD.

ecuadros@utec.edu.pe

María Hilda Bermejo, M. Sc.

mbermejo@utec.edu.pe

Telegram:

1. Configurar tu cuenta

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

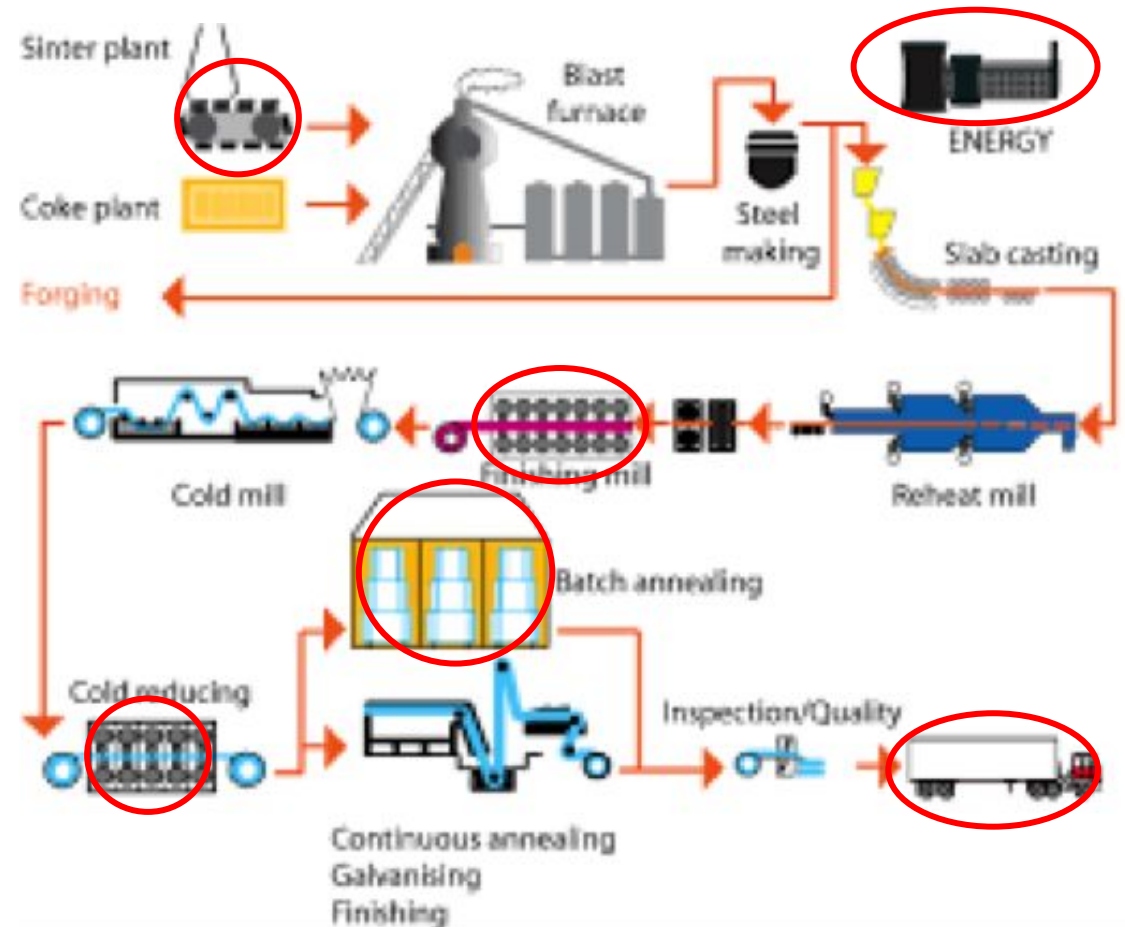
Logro de la sesión:

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

- **Clase – Objeto**
- **Métodos de acceso (setter y getters)**
- **Constructores**

Objeto:

- Un **objeto** es un componente del problema a resolver



Objetos:

Cada **objeto** pertenece a un tipo (**clase**), con propiedades (**atributos**) y funciones (**métodos**)

Tipo/Clase

MATERIA PRIMA

Propiedades

Peso
Volúmen
Precio
...

Funciones

Almacén
Procesamiento
Distribución de
productos
Venta de productos



Objetos:

Cada **objeto** pertenece a un tipo (**clase**), con propiedades (**atributos**) y funciones (**métodos**)

Tipo/Clase

INMUEBLE

Propiedades

Planos
Material 01
Material 02
...

Funciones

Cimientos
Sistema eléctrico
Construcción
Sistema agua/desagüe



Objetos:

Cada **objeto** pertenece a un tipo (**clase**), con propiedades (**atributos**) y funciones (**métodos**)

Tipo/Clase

PLAN DE ESTUDIO

Propiedades

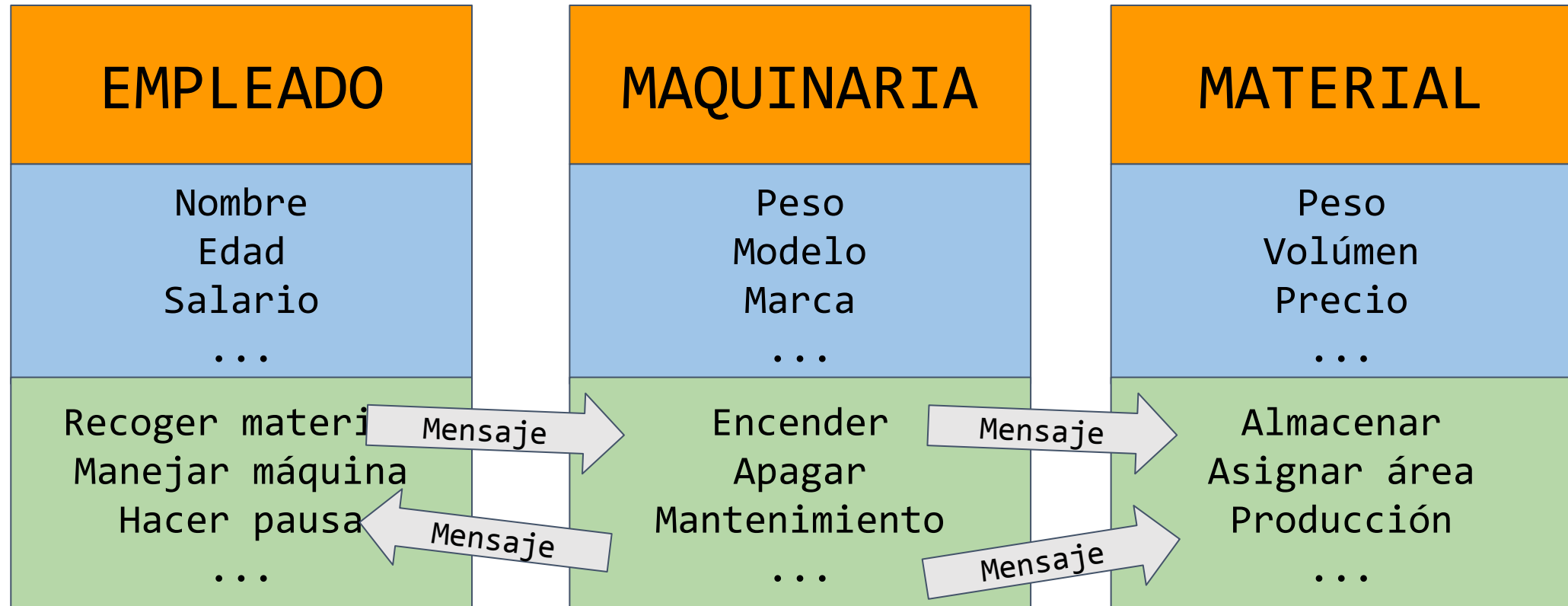
#Semestres
Lista de cursos
Plan
...

Funciones

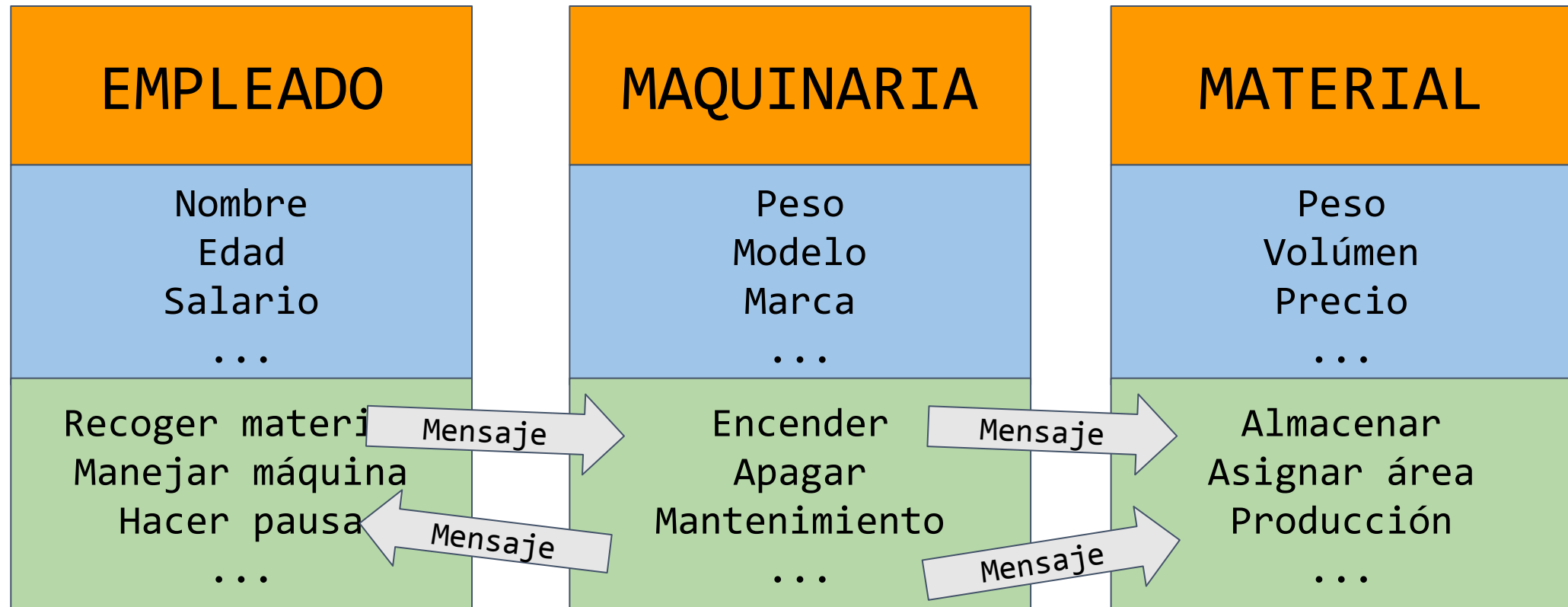
Planeamiento
Ejecución
Informes
Presentación



Un **programa** es un grupo de **objetos** interactuando (enviando mensajes)



Todos los **objetos** de una misma **clase** reciben los mismos mensajes/órdenes

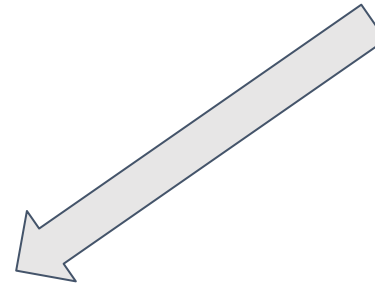
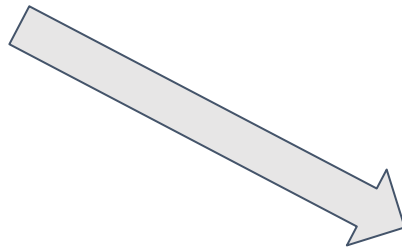


En C++ el código está **encapsulado** en **clases**

CLASE

Atributos

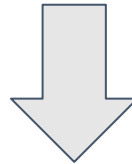
Métodos



En C++ el código está **encapsulado** en **clases**

CLASE

Encapsulamiento



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

Clase

luz

Interfaz

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



En C++:

```
luz foco;  
foco.encender();
```

Clase y objeto

Clase y Objeto

Una clase:

Es un conjunto de objetos que tienen las mismas características.

Un Objeto:

Es la instancia de una clase, es decir es un ejemplar de una clase.



Clase: carta



Un objeto de la clase carta.



Otro objeto de la clase carta. **Un ejemplar de la clase carta.**

Clase: Perro Chow chow



Todos los objetos de una clase, tienen las mismas características, por Ej:

Tienen un **nombre, edad, estatura, peso**

Nacen, crecen, mueren

Son capaces de: **correr, ladrar, morder,**



Simba

Un objeto es un ejemplar de la clase.



Tifón

Un objeto es una instancia de la clase.

Atributos y métodos

Los atributos de una clase, normalmente guardan datos propiamente dichos, mientras que los métodos son acciones que son capaces de hacer los objetos de una clase.

Métodos de una clase suelen ser funciones y se suelen expresar como verbos.

En el ejemplo:

Son atributos: edad, talla, peso, nombre

Son métodos : correr, ladrar, morder.

Cómo se define una clase:

```
class CPerro
```

```
{
```

```
    private:
```

```
        float talla;
```

```
        float peso;
```

```
        int  edad;
```

```
        void Correr();
```

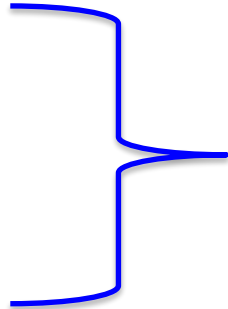
```
    public:
```

```
        string nombre;
```

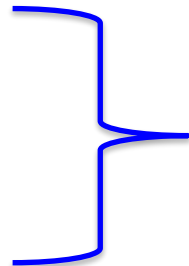
```
        void Morder();
```

```
        void Ladrar();
```

```
};
```



Con acceso restringido



Con acceso libre

Cómo se instancia un objeto?

```
CPerro  p1;
```

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

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

```
#include <string>
```

```
class CPerro
```

```
{
```

```
    private:
```

```
        float talla;
```

```
        float peso;
```

```
        int edad;
```

```
        void Correr();
```

```
    public:
```

```
        string nombre;
```

```
        void Morder();
```

```
        void Ladrar();
```

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

```
    void setTalla(float _talla) { talla = _talla; }
```

```
    void setPeso(float _peso) { peso = _peso; }
```

```
    void setEdad(int _edad) { edad = _edad; }
```

```
};
```

Cómo se asigna valores a los atributos?

```
CPerro    p1;
```

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

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

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

```
p1.nombre = "Simba";
```

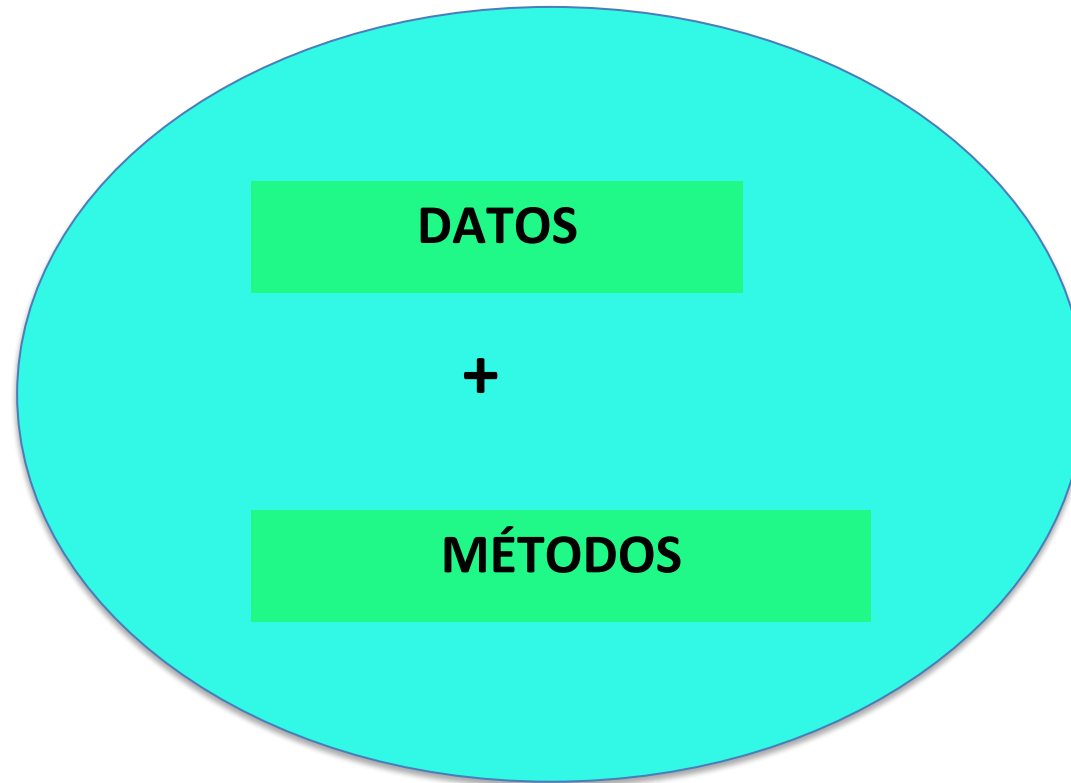
P.D. En el ejemplo falta implementar los métodos Correr(), Morder(), Ladrar();

```
#include <string>
Class CPerro
{
    private:
        float talla;
        float peso;
        int edad;
        void Correr();
    public:
        string nombre;
        void Morder();
        void Ladrar();
        //-----metodos de acceso: setters
        void setTalla(float _talla) { talla = _talla; }
        void setPeso(float _peso) { peso = _peso; }
        void setEdad(int _edad) { edad = _edad; }
        void setNombre(string _nombre) { nombre = _nombre; }
        //-----metodos de acceso: getters
        float getTalla() { return talla; }
        float getPeso() { return peso; }
        int getTdad() { return edad; }
};
```

```
CPerro p1;

p1.setTalla(40.75);
p1.setPeso(3500);
p1.setEdad(3);
p1.nombre = "Simba";
cout << "El perro tiene " << p1.getEdad() << "años";
```

OBJETO:



Entonces podemos decir: Que un objeto está conformado por DATOS Y MÉTODOS.

Una vez que se ha definido e implementado los métodos de la clase, se puede instanciar tantos objetos como se desee.

Cada objeto que se cree, se creará con las características que se han definido en la clase.

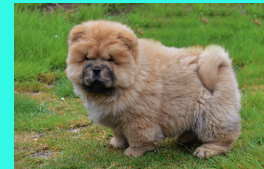
CPerro p1;

```
private:  
float talla;  // --- 110  
float peso;  // -- 7.500  
int edad;    // -- 3  
void Correr();  
public:  
string nombre; //--Simba  
void Morder();  
void Ladrar();
```



CPerro p2;

```
private:  
float talla;  //-- 50  
float peso;   //-- 4.700  
int edad;     /--- 12  
void Correr();  
public:  
string nombre; /--- Tifon  
void Morder();  
void Ladrar();
```



Ejemplo 1:

Escriba un programa Orientado a Objetos - POO, que permita hallar 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

class CRectangulo
{
private:
    float largo;
    float ancho;
public:
    float Area();
    float Perimetro();
    //----metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setlargo(float _largo) { largo = _largo; }
    void setancho(float _ancho) { ancho = _ancho; }
    //--- getters
    float getlargo() { return largo; }
    float getancho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

float CRectangulo::Area()
{
    return (largo*ancho);
}

float 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 << "El area es " << R1.Area() << "\n";
  cout << "El perimetro es " << R1.Perimetro();

  CRectangulo R2;
  float 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 << "El area del segundo rectangulo es " << R2.Area() << "\n";
  cout << "El perimetro del segundo rectangulo es " << R2.Perimetro();
  return 0;
}

```

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?

Constructores y destructores

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:

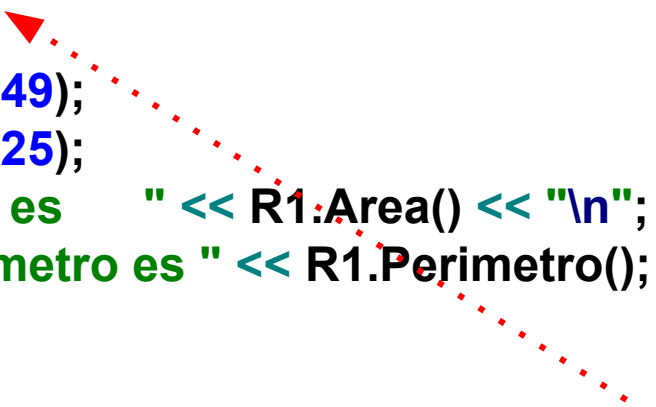
ClassName :: ClassName () {}

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

```
#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 ClassName {  
    ClassName(parameter-list);    // solo prototipo  
}
```

```
//----- En el archivo *.cpp  
// Implementación del Constructor - identificado via operator alcance  
ClassName::ClassName(parameter-list) {  
    function-body;  
}
```

Ahora declaramos el constructor:

CRectangulo.h

```
#ifndef
RECTANGULO_00_CRECTANGULO_H
#define
RECTANGULO_00_CRECTANGULO_H

class CRectangulo
{
private:
    float largo;
    float ancho;
public:
    CRectangulo(float _largo, float _ancho);
    float Area();
    float Perimetro();
    //----metodos de acceso
    //--- setter que permiten asignar un valor al
atributo
    void setlargo(float _largo) { largo = _largo; }
    void setancho(float _ancho) { ancho =
_ancho; }
    //--- getters
    float getlargo() { return largo; }
    float getancho() { return ancho; }
};

#endif
//RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

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

float CRectangulo::Area()
{
    return (largo*ancho);
}

float 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);

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

Ahora, usando this

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H
```

```
class CRectangulo
```

```
{
private:
    float largo;
    float ancho;
public:
    CRectangulo(float largo, float ancho);
    float Area();
    float Perimetro();
    //---metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setlargo(float _largo) { largo = _largo; }
    void setancho(float _ancho) { ancho = _ancho; }
    //--- getters
    float getlargo() { return largo; }
    float getancho() { return ancho; }
};
```

```
#endif //RECTANGULO_00_CRECTANGULO_H
```

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

CRectangulo.cpp

```
#include "CRectangulo.h"
```

```
CRectangulo::CRectangulo(float largo, float ancho)
```

```
{
    this->largo = largo;
    this->ancho = ancho;
}
```

```
float CRectangulo::Area()
```

```
{
    return (largo*ancho);
}
```

```
float 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);
```

```
    cout <<"El area del segundo rectangulo es " << R.Area() <<
    "\n";
```

```
    cout <<"El perimetro del segundo rectangulo es " <<
    R.Perimetro();
```

```
    return 0;
```

```
}
```

Según la sintaxis de las últimas versiones de C++, se puede declarar el constructor así:

CRectangulo.h

```
#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H

class CRectangulo
{
private:
    float largo;
    float ancho;
public:
    CRectangulo(float _largo, float _ancho): largo(_largo), ancho(_ancho){};
    float Area();
    float Perimetro();
    //---metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setlargo(float _largo) { largo = _largo; }
    void setancho(float _ancho) { ancho = _ancho; }
    //--- getters
    float getlargo() { return largo; }
    float getancho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H
```

CRectangulo.cpp

```
#include "CRectangulo.h"

float CRectangulo::Area()
{
    return (largo*ancho);
}

float 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);

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

Sobrecarga de funciones : Constructor

- Una 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 (o al constructor) puede optar por invocar una versión determinada haciendo coincidir la lista de parámetros.

Sobrecarga de funciones : Constructor.

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

CRectangulo.h

```

#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H

class CRectangulo
{
private:
    float largo;
    float ancho;
public:
    CRectangulo(){};
    CRectangulo(float _largo, float _ancho): largo(_largo), ancho(_ancho){};
    float Area();
    float Perimetro();
    //---metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setlargo(float _largo) { largo = _largo; }
    void setancho(float _ancho) { ancho = _ancho; }
    //--- getters
    float getlargo() { return largo; }
    float getancho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H

```

CRectangulo.cpp

```

#include "CRectangulo.h"

float CRectangulo::Area()
{
    return (largo*ancho);
}

float CRectangulo::Perimetro()
{
    return( 2*largo + 2*ancho);
}

```

main.cpp

```

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

int main()
{
    CRectangulo R1;
    R1.setlargo(23.45);
    R1.setancho(12.09);
    cout <<"Rectangulo 1 \n";
    cout <<"El area    : " << R1.Area() << "\n";
    cout <<"El perimetro : " << R1.Perimetro();

    CRectangulo R2(43.5,22.5);
    cout <<"\n\nRectangulo 2 \n";
    cout <<"El area    : " << R2.Area() << "\n";
    cout <<"El perimetro : " << R2.Perimetro();
    return 0;
}

```

Rectangulo 1
El area : 283.51
El perimetro : 71.08

Rectangulo 2
El area : 978.75
El perimetro : 132

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.

CRectangulo.h

```

#ifndef RECTANGULO_00_CRECTANGULO_H
#define RECTANGULO_00_CRECTANGULO_H

class CRectangulo
{
private:
    float largo;
    float ancho;
public:
    CRectangulo(){};
    CRectangulo(float _largo, float _ancho): largo(_largo), ancho(_ancho){};
    virtual ~CRectangulo(){};
    float Area();
    float Perimetro();
    //----metodos de acceso
    //--- setter que permiten asignar un valor al atributo
    void setlargo(float _largo) { largo = _largo; }
    void setancho(float _ancho) { ancho = _ancho; }
    //--- getters
    float getlargo() { return largo; }
    float getancho() { return ancho; }
};

#endif //RECTANGULO_00_CRECTANGULO_H

```

CRectangulo.cpp

```

#include "CRectangulo.h"

float CRectangulo::Area()
{
    return (largo*ancho);
}

float CRectangulo::Perimetro()
{
    return( 2*largo + 2*ancho);
}

```

main.cpp

```

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

int main()
{
    CRectangulo R1;
    R1.setlargo(23.45);
    R1.setancho(12.09);
    cout <<"Rectangulo 1 \n";
    cout <<"El area    : " << R1.Area() << "\n";
    cout <<"El perimetro : " << R1.Perimetro();

    CRectangulo R2(43.5,22.5);
    cout <<"\n\nRectangulo 2 \n";
    cout <<"El area    : " << R2.Area() << "\n";
    cout <<"El perimetro : " << R2.Perimetro();
    return 0;
}

```

Rectangulo 1
El area : 283.51
El perimetro : 71.08

Rectangulo 2
El area : 978.75
El perimetro : 132



1. ¿Qué es un objeto?
2. ¿Qué es una clase?
3. ¿En qué zona de la clase es recomendable que se declaren los atributos?
4. ¿Cuándo se usa this?
5. ¿Por qué es necesario utilizar métodos de acceso?
6. ¿Una clase puede tener más de un destructor?
7. ¿Una clase puede tener varios constructores?

Unidad 6:

Programación Orientada a Objetos - Parte 1

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

Profesores:

Ernesto Cuadros- Vargas, PhD.

ecuadros@utec.edu.pe

María Hilda Bermejo, M. Sc.

mbermejo@utec.edu.pe