Programación Orientada a Objetos

Parcial 2

Notas de curso

Dr. Ezequiel Arceo May

25 de marzo de 2019

Índice

«Un lenguaje de programación es una forma de expresarnos»
— Ezequiel Arceo —

1. Concepto básicos

Los conceptos más básicos que debemos entender en la Programación Orientada a Objetos (POO) son:

- Clase
- Objeto
- Abstracción
- Encapsulación
- Herencia
- Polimorfismo

1.1. Clase

Una **clase** es una **abstracción** que hacemos de nuestra experiencia sensible. El ser humano tiende a agrupar seres o cosas (que son los *objetos*) con características similares en grupos o *clases*.

Clasificar es algo que usamos en nuestro día a día:

Figura 1: Existen muchas razas de perros; hay perros pequeños y grandes, perros de colores claros y de colores oscuros. Sin embargo, cuando vemos un perro lo distinguimos rápidamente, sabemos que "eso es un perro", ya lo hemos identificado dentro de un grupo de objetos al que llamamos "perro". Esto es, decimos que es un objeto *son de la clase* **Perro**.

1.2. Objeto

Un **objeto** es una **colección de Atributos y Métodos**, un objeto se deriva de una clase. Un objeto es un elemento de una clase. Un objeto pertenece a una clase.

Todo lo que nos rodea es un objeto, un perro, un ser humano, una mesa, una cámara, una computadora, un celular, etc.

Considere el siguiente ejemplo:

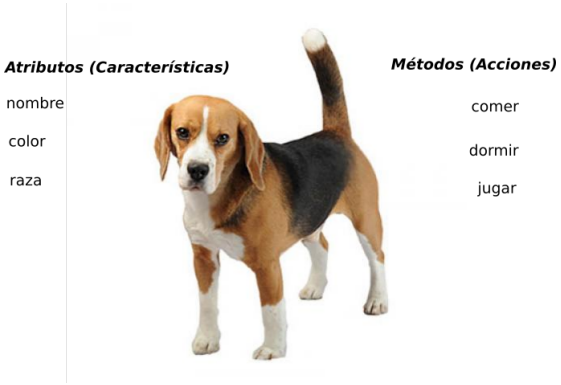


Figura 2: Un objeto perro de la clase **Perro** tiene Atributos como **nombre**, **raza**, **color**, etc, pero también tiene Métodos como **comer**, **dormir**, **jugar**, etc.

Todos los objetos tiene atributos (que son las características del objeto) y métodos (que son las acciones que el objeto realiza).

1.3. Abstracción

Una **abstracción** es el proceso mental de extraer los esencial de algo, ignorando los detalles irrelevantes/superfluos.

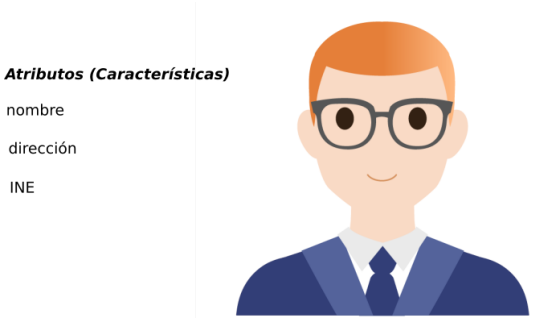


Figura 3: Si quisiéramos abstraer la clase **Persona**, tendríamos que considerar atributos relevantes como **nombre**, **dirección**, **INE**, etc. Ignoramos cosas tales como la cantidad de veces que ha visto su película favorita, el nombre de su perro, o la marca de sus calcetines.

1.4. Encapsulación

La **encapsulación** es un proceso mediante el cual ocultamos los detalles que dan soporte a las características de una abstracción.

Figura 4: Cuando vemos al médico y nos receta un medicamento en forma de píldora, esperamos que nos cure nuestra dolencia, no sabemos y no necesitamos saber qué es lo que contiene ni cómo funciona. La ciencia médica se ha encargado de poner lo esencial en la píldora, nosotros solamente la consumimos.

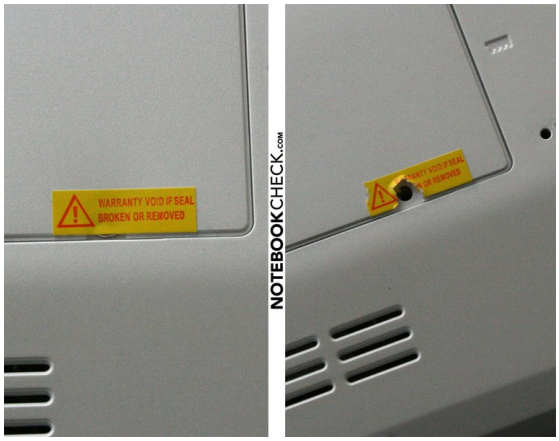


Figura 5: La encapsulación también es una medida de protección; si cualquiera tiene acceso al contenido interior puede ocasionar daños a propósito o accidentalmente. Recuerde los sellos de garantía en los productos que compra.

De forma práctica, la ecapsulación aísla los atributos para que no los accedamos directamente.

1.5. Herencia

La **herencia** sucede cuando una clase nueva de crea a partir de una clase existente, obteniendo (heredando) todos sus atributos y métodos.

A la clase de la cual se hereda se le llama *clase padre* o *súper clase*. A la clase que recibe la herencia se le llama *clase hija* o *subclase*.

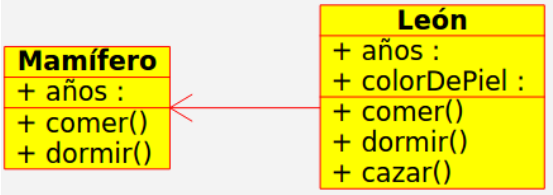


Figura 6: Ejemplo de herencia. **Mamífero** es la clase padre, y **León** es la clase hija. Note que todos los atributos y métodos de la clase **Mamífero** están en la clase **León**, junto con otros que son propios y especializan a la clase **León**.

El diagrama de la figura 6 se puede leer como:

- "la clase **León** hereda de la clase **Mamífero**",
- "la clase **León** es clase hija de la clase **Mamífero**",
- "la clase **León** es una subclase de la clase **Mamífero**",
- "la clase **Mamífero** es clase padre de la clase **León**",
- "la clase **Mamífero** es súper clase de la clase **León**".

Con la herencia nos ahorramos volver a definir los atributos y métodos de una clase. Basta con heredarlos de la clase padre y ya.

1.6. Polimorfismo

El **polimorfismo** es la cualidad que tiene los objetos de responder de distintas formas al mismo mensaje.

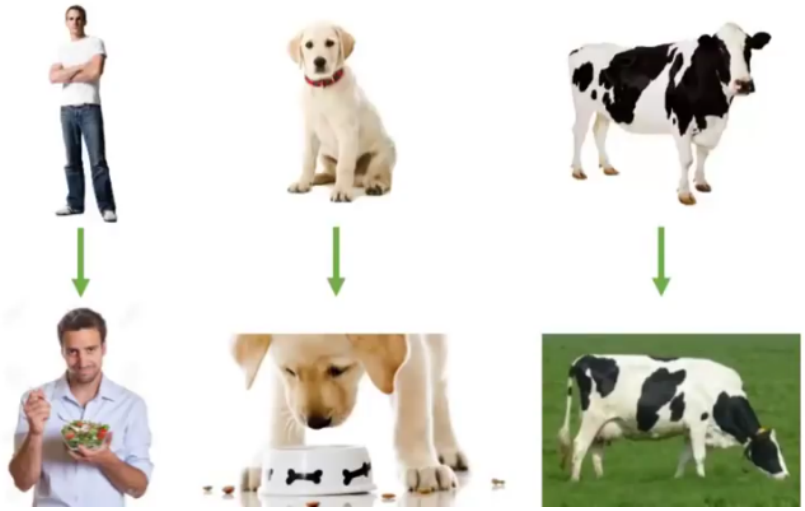


Figura 7: Considere objetos de las clases **Persona**, **Perro** y **Vaca**, cada uno de ellos puede **comer**, pero lo hacen de forma distinta.

2. Objetos en todas partes

Los objetos están en todas partes, es importante aprender a reconocerlos en situaciones de la vida real.

2.1. Reconoce objetos a partir de nombres

Imagine que tenemos que desarrollar una aplicación simple y recibimos una descripción con los requerimientos. La aplicación debe permitir al usuario calcular áreas y perímetros de cuadrados, rectángulos, círculos y elipses.

Entonces necesitamos calcular el área y perímetro de cuatro objetos. Los nombres que se refieren a estos objetos son:

- Cuadrado
- Rectangulo
- Circulo
- Elipse

Si tiene que dibujar estas figuras en papel para calcular manualmente su área y perímetro ¿qué información necesita para cada figura?

Figura	**Información requerida**
Cuadrado	lado
Rectangulo	base, altura
Circulo	radio
Elipse	semieje mayor, semieje menor

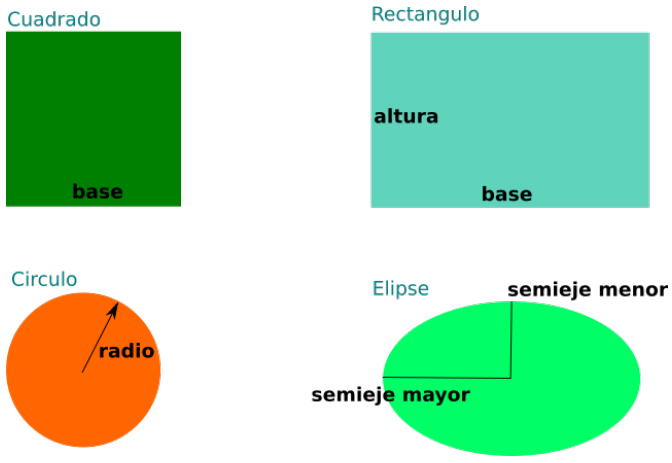


Figura 8: Figuras geométricas y su información requerida.

2.2. Haz plantillas para los objetos

Imagine ahora que tiene que calcular el área de cuatro rectángulos, cada uno con base y altura diferente. Terminaríamos con cuatro dibujos distintos, a menos que hagamos una plantilla en la cual solamente tengamos que especificar los valores de base y altura. A esta plantilla se le conoce como una **clase**. Una clase abstrae las características relevantes de los objetos.

Una vez que tenemos una clase, la podemos usar para crear objetos de dicha clase. A estos objetos se les llama **instancias** de clase.

2.3. Reconoce los atributos

Debemos estar seguros de que una clase tenga las variables necesarias que encapsulen toda la información requerida para realizar sus tareas.

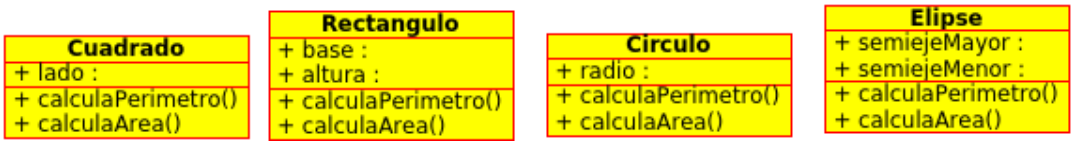
Nombre de Clase	Atributos
Cuadrado	lado
Rectangulo	base, altura
Circulo	radio
Elipse	semiejeMayor, semiejeMenor

La siguiente imagen muestra el diagram UML (Unified Modelling Language) muestra las cuatro clases con sus atributos

2.4. Reconoce acciones de los verbos

Hasta ahora hemos diseñado cuatro clases e identificado los atributos necesarios para cada una. Ahora, es tiempo de añadir las piezas necesarias de código que funciones con los atributos para desempeñar las tareas necesarias. En otras palabras, debemos encapsular dentro de las clases las funciones que procesen los atributos especificados en los objetos para realizar sus tareas.

En nuestro caso, para todas las figuras necesitamos calcular el perímetro y el área. Llamaremos `calculaPerimetro` a la función que calcula el perímetro de la figura *en función de sus atributos*. Llamaremos `calculaArea` a la función que calcula el área de la figura *en función de sus atributos*.



2.5. Organicemos las clases

Hasta ahora, nuestra solución orientada a objetos tiene cuatro clases, con sus atributos y métodos. Sin embargo, si nos damos cuenta, notaremos que las cuatro clases tienen los mismos métodos: `calculaPerimetro` y `calculaArea`. El código en los métodos de cada clase es diferente, porque cada figura usa datos y una fórmula diferente. Sin embargo, las declaraciones de estos métodos son idénticas: ambos tienen el mismo nombre, carecen de parámetros, y retornan un valor flotante.

Todas la formas tratadas son figuras geométricas. Podemos abstraer las características comunes, en este caso la capacidad de calcular el perímetro y área. Definimos así la clase **FiguraGeometrica** con los métodos **calculaPerimetro** y **calculaArea**, y hacemos que las otras cuatro clases hereden de esta:

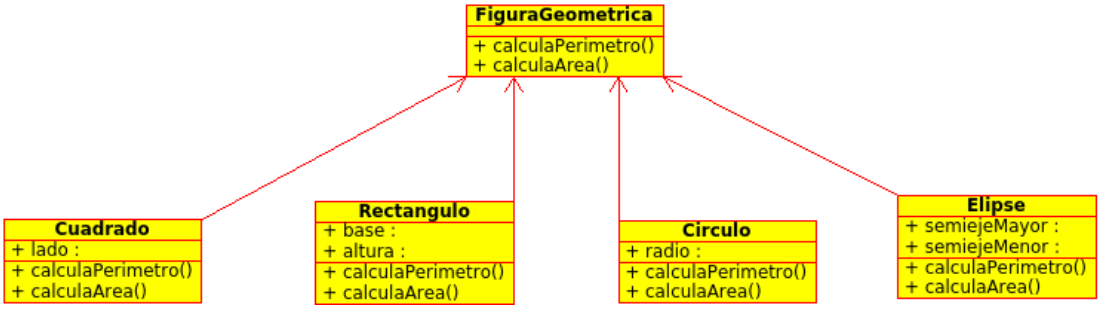


Figura 9: Jerarquía de figuras geométricas

Trabajo B1. Entrega Antes de: Lunes 25 de Marzo de 2019 a las 23:59.
Para cada clase en la figura 9 llene una tabla como la siguiente

Nombre	Atributos	Métodos	Clase Padre	Clase(s) Hija(s)

- Enviar por correo a `ezequiel_arceo@my.uvm.edu.mx`
- El asunto del correo será `POO_TrabajoB1_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `POO_TrabajoB1_NOMBRE_DEL_AUTOR.*`

Ejemplo:
ASUNTO: `POO_TrabajoB1_JUAN_PEREZ_FERNANDEZ`
ADJUNTO: `POO_TrabajoB1_JUAN_PEREZ_FERNANDEZ.cpp`

3. Clases y Objetos en C++

Para hacer una clase en C++ usamos la palabra reservada `class`, seguida del nombre de la clase **con la primera letra en mayúscula**, un par de llaves {}, y finalizamos con un `;`.

3.1. Estructura de una clase

Dentro de las llaves, se escribe palabra reservada `private` seguida de `:` y se declaran los atributos de la clase. Se escribe la palabra reservada `public` seguida de `:` y declaran los métodos de la clase.

Siempre usaremos algo de la forma:

```
class NombreDeLaClase
{
  private: // Atributos
    // declaración del atributo 1
    // declaración del atributo 2
    // ...
  public: // Métodos
    // declaración del método 1
    // declaración del método 2
    // ...
}; // Es importante colocar ;
```

Todo lo declarado como `private` es accesible solamente a los elementos de la clase. Los elementos bajo `private` son una encapsulación de los datos de una clase.

Todo lo declarado como `public` es accesible desde el exterior de la clase. Esta es la interfaz de la clase.

Juntas, las declaraciones de atributos y métodos forman la abstracción de una clase. Podemos considerar que esta es la declaración de una clase, pero dicha clase todavía no puede ser usada porque no ha sido definida, es decir, no ha sido implementada.

3.2. Método Constructor

Toda clase debe definir al menos un método especial denominado **constructor**, con el mismo nombre de la clase, y que tome como argumentos a todos los atributos de la clase.

Como su nombre lo indica, el constructor tiene la tarea de inicializar los atributos de un objeto de la clase con los valores de los parámetros que recibe.

Por ejemplo, el constructor de la clase **Persona** con atributos **nombre** y **edad** se llama **Persona** y tiene dos argumentos, uno de tipo **string** y otro de tipo **int**.

Considere la siguiente abstracción simple de la clase **Persona**

```cpp
//EJEMPLO: Declaración, implementación y uso de la clase Persona
#include <iostream>
#include <string>
using namespace std;

class Persona
{
  private: // Atributos
    string nombre;
    int edad;
  public: // Métodos
    Persona(string,int); // Constructor (sin tipo de retorno!!)
    void leer();
    void caminar();
};

// Definición del método Constructor de la clase Persona
Persona::Persona(string _nombre, int _edad)
{
  nombre = _nombre;
  edad = _edad;
}
// Definición del método leer de la clase Persona
Persona::leer()
{
  cout<<"Soy "<<nombre<<" y estoy leyendo."<<endl;
}
// Definición del método caminar de la clase Persona
Persona::caminar()
{
  cout<<"Tengo "<<edad<<" años y estoy caminando."<<endl;
}
// continuará ...
```

IMPORTANTE: Dentro de la definición del método constructor **Persona** no hemos declarado las variables `nombre` y `edad`, estas variables son los atributos **nombre** y **edad** de la clase **Persona**. ¿Cómo es esto posible? . . . bueno, al usar `Persona::` especificamos que estamos trabajando en el ámbito de la clase **Persona** y por lo tanto, todos los miembros (atributos y métodos) de la clase **Persona** son accesibles.

Los métodos **leer** y **caminar** también tienen acceso a los atributos **nombre** y **edad** porque están en el ámbito de la clase **Persona**.

3.3. Creación de Objetos

Ahora que ya tenemos una clase podemos crear objetos, y usar sus métodos.

A la creación de objetos también se le denomina **instanciación de clase**, y se dice que el nuevo objeto es una instancia de la clase instanciada.

Para usar el método **miMetodo** de un objeto `miObjeto` escribimos `miObjeto.miMetodo(...)` donde `...` se refiere a la lista de argumentos que **miMetodo** requiere.

```
// ... continuación
//EJEMPLO: Declaración, implementación y uso de la clase Persona
void()
{
  // Definición de objetos de la clase Persona
  // Forma 1 usando el constructor
  Persona p1 = Persona("Ezequiel",33);
  // Forma 2 usando el constructor
  Persona p2("Itzel",6);

  // Llamamos a los métodos de objetos Persona
  p1.leer();
  p2.caminar();
}
```

Un método de clase no es más que una función dentro del ámbito de la clase en donde se declara, por lo tanto puede requerir argumentos y puede tener valor de retorno, como cualquier función.

EJEMPLO: Construcción de una clase llamada **Rectangulo** con los atributos base y altura y los métodos **perimetro** y **area**.

```
//EJEMPLO: clase Rectangulo
#include <iostream>
#include <string>
using namespace std;

class Rectangulo
{
  private: // Atributos
    float base,altura;
  public: // Métodos
    Persona(string,int); // Constructor (sin tipo de retorno!!)
    void perimetro();
    void area();
};
// Definición del método Constructor
Rectangulo::Rectangulo(float _base, float _altura)
{
  base = _base;
  altura = _altura;
}
Rectangulo::perimetro()
{
  float _perimetro;
  _perimeto = 2*base + 2*altura;
  cout<<"El perimetro es "<<_perimeto<<" unidades."<<endl;
}
Rectangulo::area()
{
  float _area;
  _area = base*altura;
  cout<<"El area es "<<_area<<" unidades cuadradas."<<endl;
}

void main()
{
  Rectangulo r1 = Rectabgulo(5,10);
  r1.perimetro();
  r1.area();
  Rectangulo r2(2,3);
  r2.perimetro();
  r2.area();
}
```

Actividad 1.
Construya una clase llamada **Triangulo** con los atributos **base** y **altura** y los métodos **perimetro** y **area**.
Trabajo B2. Entrega Antes de: Lunes 25 de Marzo de 2019 a las 23:59.
Construya una clase llamada **Circulo** con el atributo **radio** y los métodos **perimetro** y **area**.

- Enviar por correo a `ezequiel_arceo@my.uvm.edu.mx`
- El asunto del correo será `POO_TrabajoB2_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `POO_TrabajoB2_NOMBRE_DEL_AUTOR.*`

Ejemplo:
ASUNTO: `POO_TrabajoB2_JUAN_PEREZ_FERNANDEZ`
ADJUNTO: `POO_TrabajoB2_JUAN_PEREZ_FERNANDEZ.cpp`

4. Múltiples constructores

En ocasiones necesitamos instanciar una clase, pero sin especificar los parámetros en el constructor, como si estuviéramos declarando una variable del tipo de la clase. En este caso, se usa el denominado **constructor por defecto**.

También, llegan a ser necesarios constructores especializados que tengan un formato de entrada de datos distinto al constructor principal.

El constructor por defecto y cualquier constructor especializado tienen el mismo nombre del contructor, es decir, se llaman igual que la clase. ¿Cómo es esto posible?

4.1. Sobrecarga de Funciones

C++ permite definir varias funciones con el mismo nombre dentro del mismo ámbito, siempre y cuando todas tengan una cantidad diferente de parámetros, o estos sean de distintos tipos.

```
1  // EJEMPLO: Sobrecarga de funciones
2  #include <iostream>
3  using namespace std;
4  // Función cuadrado para valores int
5  int cuadrado(int x)
6  {
7    int _cuadrado = x*x
8    cout<<"El cuadrado del int "<<x<<" es "<<_cuadrado<<endl;
9    return cuadrado;
10 }
11 // Función cuadrado para valores float
12 float cuadrado(float x)
13 {
14   float _cuadrado = x*x;
15   cout<<"El cuadrado del float "<<x<<" es "<<_cuadrado<<endl;
16   return cuadrado;
17 }
18
19 void main()
20 {
21   cuadrado(3);
22   cuadrado(3.0);
23 }
```

Actividad 2.
Sobrecargue una función llamada `suma` que sume dos números. Considere que los argumentos pueden ser de tipo **int** o **float**. El tipo de retorno de todas será **float**. HINT: Primero determine cuantas funciones necesita escribir.

Trabajo B3. Entrega Antes de: Jueves 28 de Marzo de 2019 a las 23:59.
Sobrecargue una función llamada `producto` que multiplique dos números. Considere que los argumentos pueden ser de tipo **int** o **float**. El tipo de retorno de todas será **float**.
HINT: Primero determine cuantas funciones necesita escribir.

- Enviar por correo a `ezequiel_arceo@my.uvm.edu.mx`
- El asunto del correo será `POO_TrabajoB3_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `POO_TrabajoB3_NOMBRE_DEL_AUTOR.*`

Ejemplo:
ASUNTO: `POO_TrabajoB3_JUAN_PEREZ_FERNANDEZ`
ADJUNTO: `POO_TrabajoB3_JUAN_PEREZ_FERNANDEZ.cpp`

4.2. Constructor por defecto

El constructor por defecto permite dar valores por defecto a los objetos.

```
// EJEMPLO: Constructor por defecto (Sobrecarga de funciones)
#include <iostream>
using namespace std;
class Punto()
{
  private: // Atributos
    float x,y;
  public:  // Métodos
    Punto(float,float); // Constructor
    Punto();            // Constructor por defecto
    muestraPunto();
};
Punto::Punto(float _x, float _y)
{// Colocamos el punto en donde indica el usuario
  x = _x;
  y = _y;
}
Punto::Punto()
{// Colocamos al punto en el origen
  x = 0.0;
  y = 0.0;
}
Punto::muestraPunto()
{
  cout<<"("<<x<<", "<<y<<")"<<endl;
}

void main()
{
   Punto p1(2.0,3.0);
   Punto p2;

   p1.muestraPunto();
   p2.muestraPunto();
}
```

Cuando declaramos variables de tipo **int**, **float** o **string** obtenemos valores por defecto:

```cpp
// EJEMPLOS: valores por defecto en tipos estándar
#include <iostream>
#include <string>
using namespace std;

int main()
{
  int entero;
  float flotante;
  string cadena;
  cout << "Valor int por defecto '" << entero <<"'"<< endl;
  cout << "Valor float por defecto '" << flotante <<"'"<< endl;
  cout << "Valor strng por defecto '" << cadena <<"'"<< endl;
}
```