Programación Orientada a Objetos - Clases

Pedro Fernando Flores Palmeros
ESIME - ZACATENCO

Abril 2020

1. Introducción a las clases y miembros

Una calse es sólo una colección de varables por lo general de tipos distintos, combinadas con un conjunto de funciones relacionadas.

Una forma de pensar en un auto es como si fuera una colección de ruedas, puerta, asientos, ventanas, etc. Otra forma es pensar en lo que un auto puede hacer: se puede mover, acelerar, desacelerar, detener, estacionar, etc. Una clase le permite conjuntar, o reunir en un solo paquete todas estas partes y funciones en una sola entidad, lo que se conoce como objeto.

Una ventaja de reunir un conjunto de características y funciones en una entidad, es la forma en que se puede definir su intergación . Una clase se puede componer de cualquier combinación de los tipos de variables y también de otros tipos de clases. Las variables que están en la clase se conocen como *variables miembro, datos miembro* o *atributos*.

Como se ha mencionado previamente una clase puede contener funciones que ayudan a manipuar los atributos de una clase, a estas funciones se les conoce como *métodos*.

2. Declaración de una clase

Para declarar una clase, se debe de utilizar la palabra reservada `class` seguida del nombre de la clase y una llave de apertura, después se debe de poner en forma de lista los atributos y métodos de esa clase. Termine la declaración con una llave de cierrre y un punto y coma. La declaración de una clase llamada `Gato` sería así:

```
class Felino{
   unsigned int Edad;
   unsigned int Peso;
   void Maullar();
}
```

Código 1: Declaración de una clase

A delcarar esta clase no se asigna memoria para un `Felino`. Sólo se le indica al compilador cómo es un `Felino`, qué datos tiene (`Edad, Peso`) y qué puede hacer(`Maullar`), observe que los datos que tienen son los atributos y lo que puede hacer está asociado con los métodos.

También se le indica al compilador qué tan grande es `Felino` (cuánto espacio debe reservar el compilador para cada `Felino` que vaya a crear)

3. Declaración de un objeto

Para poder declarar un objeto, se debe de hacer como se haría una variable de cualquier tipo de dato, sólo que en este caso el tipo de dato es la clase.

```
// Declarando una variable tipo entero
int a;

// Declarando un objeto tipo Gato
Felino miGato;
```

Código 2: Declaración de un objeto

El código anterior declara una variable llamada `a` que es tipo entero. También se declara a `miGato` que es un bojeto cuya clase (tipo) es `Felino` .

4. Comparación de clases y objetos

Para comprender la diferencia entre las clases y los objetos, consideremos la siguiente analogía. Una fotografía de una tigre, en realidad no es un tigre, sino un pedazo de papel fotográfico, que con ayuda de tinta logra capturar la imagen de un tigre sobre el papel, pero no es un tigre.

La fotografía sería la clase y el tigre (el animal que se mueve, que gruñe, bosteza, se alimenta, etc) corresponde al objeto.

5. Cómo acceder a los miembros de las clases

Después de definir un `Felino` (por ejemplo `Pantera`) se utiliza el operador de punto (`.`) para tener acceso a los miembros de ese objeto . Por lo tanto, para asingar `50` al atributo `Peso` se debería de hacer de la siugiente manera.

```
Pantera.suPeso = 50;
```

Código 3: Asignación directa a atributos de una clase

De la misma forma, para llamar a la función `Maullar()`, se escribiría lo siguiente:

```
Pantera.Maullar();
```

Código 4: Acceso a métodos de una clase

Cuando utliza el método de una clase, llama al método. En este ejemplo, usted llama a `Maullar()`, la cual se encuentra en Pelusa.

6. Definición del alcance público en comparación con la del privado

En la declaración de una clase se pueden utlizar otras palabras reservadas. Dos de las más importantes son: `public` y `private`.

Todos los miembros de una clase (datos y métodos) son privados de manera predeterminada. A éstos tipos de miembros sólo se puede acceder dentro de la misma clase, es decir, sólo los métodos de la clase tienen acceso a ellos. Consecuentemente todos los elementos del *Código 1* son privados y no se tendrá acceso a éstos fuera de la clase, por lo que se podría pensar que la clase declarada no es de gran utilidad.

Para que se pueda ingresar a los atributos y métodos de una clase desde fuera de ella, es necesario declararlos como `public` como se muestra en el siguiente bloque de código.

```
#include <iostream>

class Felino
{
   public:
   int Edad;
   int Peso;
};


int main(){
    Felino Pantera;
   Pantera.Edad = 5;
   std::cout << "Pantera es un gato que tiene: " << Pantera.Edad;
   std::cout << " anios de edad" << std::endl;
}
```

Código 5: Uso del especificadores de acceso public

7. ¿De qué sirve hacer los datos miembro privados?

Como regla de diseño general, debe mantener privados los datos miembro o atributos de una clase. Por lo tanto, debe crear funciones que sean públicas, conocidas como *métodos de acceso*, para modificar y obtener los valores de las variables miembro privadas.

Un *método de acceso público* es una función miembro de la clase que se utiliza para leer el valor de una variable miembro privada de la clase, o para darle un valor.

Estos niveles de "seguridad.ªyudan a que se lleve a cabo lo que se conoce como .enCapsulamiento"que ayuda a que el usuario sólo pueda utilizar la clase y que no vaya más allá sino es necesario.

Como ejemplo se puede proponer un auto, al conductor promedio no le interesa en realidad cómo funciona un auto, sólo se sube, sabe como arrancarlo y como manejar, no es de gran importancia para el conductor el funcionamiento del auto, entonces, se puede decir que los elementos como el volante y pedales son públicos ya que es a lo que el conductor tiene acceso, a toda la parte del motor, circuitería, computadora, sensores, etc, serían privados, están allí, ayudan al correcto funcionamiento pero es necesario que se queden ocultos, que el usuario no tenga acceso a ellos.

8. Implementación de los métodos de una clase

Como se ha visto, un método de acceso proporciona una interfaz pública para los datos miembros privados de la clase. Cada método de acceso, así como culquier otro método de la case que se declare, debe de tener una implmentación. A esto se le conoce como *definición de métodos*.

La definición de un método comienza con el tipo de retorno en caso de que exista, o `void` si el método no regresa nada, después debe de colocar el nombre de la clase, seguido por dos signos de dos puntos (`: :`), el nombre del método y sus parámetros. En la siguiente código se muestra la *declaración de la clase* y la *definición de los métodos*. Este código debe de tener una extensión `.h` ya que se manejará como librería definida por el usuario, esto facilita la modularidad del programa y brinda más facilidad de lectura.

```
// felino.h
// Muestra de la declaracion de una clase y la defincion de los metodos de la
   clase.

#include <iostream>

class Felino{
  public:
  int ObtenerEdad();
  void AsignarEdad(int Edad);
```

```
   void Maullar();

   private:
   int Edad;
}


// ObtenerEdad, metodo de acceso publico
// regresa el valor de la propiedad suEdad
int Felino::ObtenerEdad(){
   return Edad;
}

// AsingarEdad, metodo de acceso publico
// da un valor a la propiedad suEdad
void Felino::AsignarEdad(int edad){
   suEdad = edad;
}

// Definicion del metodo Maullar
// regresa: void
// parametros: Ninguno
// accion: Imprime "miau" en la pantalla
void Felino::Maullar()
{
   cout << "Miau.\n";
}
```

Código 6: Archivo donde se encuentra la definición y la implementación de la clase, felino.h

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

using namespace std;


int main(){

Felino Pantera;
Pantera.AsignarEdad(5);

cout << "Pantera es un felino que tiene ";
cout << Pantera.ObtenerEdad() << " anios de edad" << endl;

Pelus.Maullar();
}
```

Código 7: Función principal, para poder utilizar la clase Felino

9. Constructores y destructores

Existen dos maneras de declarar una variable tipo entero. Puede declarar la variable y asignarle un valor posteriormente en el programa, por ejemplo:

```
int Peso;
.
.
.
Peso = 7;
```

Código 8: Declaración de una variable y posteriormente se le asigna un valor

O se puede declarar el entero e inicializarlo inmendiatamente. Por ejemplo:

```
int Peso = 7;
```

Código 9: Declaración e inicialización de una variable

En el código anterior se lleva a cabo la declaración y la inicialización de la variable. Nada le impide cambiar ese valor más adelante. La inicialización asegura que la variable nunca tenga un valor sin significado.

Las clases tinen una función miembro llamada *constructor*, el cual puede ayudar a inicializar los datos miembro de esa clase. El constructor puede tomar los parámetros que necesite pero no puede tener un valor de retorno, ni siquiera `void`. El constructor es un método de clase y necesariamente debe de tener el mismo nombre de la clase.

Siempre que declare un constructor, también debe declarar un destructor. Así como los constructores se encargan de crear e inicializar objetos de la clase, los destructores se encargan de limpiar todo cuando ya no se va a utlizar el objeto y libera la memoria que se le haya asignado. Un destructor siempre tiene el nombre de la clase, antecedido por una `~`. Los destructores no deben de tomar argumentos y tampoco tienen valor de regreso .

9.1. Constructores y destructores predeterminados

Si el programador no declara un constructor o desctructor, el compilador crea uno por usted, los constructores y destructores predeterminados no llevan argumentos y no hacen nada a simple vista aunque ayudan a reservar y liberar la memoria.

9.2. Constructores definidos por el usuario

En el ejemplo que se ha venido trabajando que es el `Felino` , se puede desarrollar el siguiente código:

```
int main(){
  Felino Silvestre;
  Silvestre.AsignarEdad(5);
}
```

Código 10: Declaración e inicialización de un objeto

Que sería equivalente al ejemplo donde se declara la variable y después en alguna otra parte del programa se le asgina el valor.

La manera en que en una clase se puede declarar un objeto e inicializar al mismo tiempo es através del constructor que el usuario defina. Observe la defínción de la siguiente clase.

```
// Muestra de la declaracin de una clase y la defincion de los metodos de la
    clase.

#include <iostream>

using namespace std;

class Felino{
   public:
   Felino(int edad);
   int ObtenerEdad();
   void AsignarEdad(int Edad);
   void Maullar();
   ~Felino();
   private:
   int Edad;
};

Felino::Felino(int edad){
   cout << "Construyendo un felino de: " << edad << " anios de edad" << endl;
   Edad = edad;
}

// ObtenerEdad, metodo de acceso publico
// regresa el valor de la propiedad Edad
int Felino::ObtenerEdad(){
   return Edad;
}

// AsingarEdad, metodo de acceso publico
// da un valor a la propiedad Edad
void Felino::AsignarEdad(int edad){
   Edad = edad;
}
```

```
// Definicion del metodo Maullar
// regresa: void
// parametros: Ninguno
// accion: Imprime "miau" en la pantalla
void Felino::Maullar()
{
   cout << "Miau.\n";
}

Felino::~Felino(){
   cout << "Felino ya se va a destruir" << endl;
}
```

Código 11: Uso de constructor y destructor definido por el usuario

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

int main(){
   Felino Pantera(8);
   std::cout << "Pantera tiene ";
   std::cout << Pantera.ObtenerEdad() << " anios de edad" << std::endl;

   Pantera.AsignarEdad(5);
   std::cout << "Pantera tiene ";
   std::cout << Pantera.ObtenerEdad() << " anios de edad" << std::endl;

}
```

Código 12: Declaración e inicialización de un objeto utilizando el constructor definido por el usuario

10. Uso de const en los métodos de las clases

Es necesario garantizar que los elementos privados de la clase no se modificarán de manera accidental, ya que de lo contrario se puden tener errores al momento de la ejecución, o simplemente la lógica con la que se ha diseñado el programa no funcionará correctamente, observe el siguiente código, en donde se ha modificado el método `ObtenerEdad` ya que regresa la edad multiplicada 10 veces.

```
// Muestra de la declaracion de una clase y la defincion de los metodos de la
    clase.

#include <iostream>
```

```
using namespace std;

class Felino{
   public:
   Felino(int edad);
   int ObtenerEdad();
   void AsignarEdad(int Edad);
   void Maullar();
   ~Felino();
   private:
   int Edad;
};

Felino::Felino(int edad){
   cout << "Construyendo un felino de: " << edad << " anios de edad" << endl;
   Edad = edad;
}

// ObtenerEdad, metodo de acceso publico
// regresa el valor de la propiedad Edad
int Felino::ObtenerEdad(){
   Edad *= 10;
   return Edad;
}

// AsingarEdad, metodo de acceso publico
// da un valor a la propiedad Edad
void Felino::AsignarEdad(int edad){
   Edad = edad;
}

// Definicion del metodo Maullar
// regresa: void
// parametros: Ninguno
// accion: Imprime "miau" en la pantalla
void Felino::Maullar()
{
   cout << "Miau.\n";
}

Felino::~Felino(){
   cout << "Felino ya se va a destruir" << endl;
}
```

Código 13: Se ha modificado el método ObtenerEdad y habrá un error lógico

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

```
int main(){
   Felino Pantera(8);
   std::cout << "-------------" << endl;
   std::cout << "Pantera tiene ";
   std::cout << Pantera.ObtenerEdad() << " anios de edad" << std::endl;
   std::cout << "---------" << endl;
   std::cout << "Pantera tiene ";
   std::cout << Pantera.ObtenerEdad() << " anios de edad" << std::endl;
   std::cout << "---------" << endl;
   std::cout << "Pantera tiene ";
   std::cout << Pantera.ObtenerEdad() << " anios de edad" << std::endl;
   std::cout << "------------"<< endl;
}
```

Código 14: Programa principal, se utiliza varias veces el método ObtenerEdad y consecuentemente existe un error lógico

Observe que se ha invocado el método `ObtenerEdad`, sin embargo, en el archivo cabecera se multiplica por 10 veces cada vez que se llama la función por lo que al ejecutar la función principal la edad de `Pantera` es ilógica y se muestra la salida del programa a continuación"

```
Construyendo un felino de: 8 anios de edad
-------------
Pantera tiene 80 anios de edad
---------
Pantera tiene 800 anios de edad
---------
Pantera tiene 8000 anios de edad
------------
Felino ya se va a destruir
```

Código 15: Ejecución del programa, observe que ha dado valores fuera de la lógica

Para evitar este tipo de situaciones es recomendable poner la palabra reservada `const` en cada función que regresa algún valor al usuario, de tal forma que si usuario quiere modificar el valor, el compilador no lo permitirá y mostrará un error. Si se le hacen las siguientes modificaciones al programa se protege ante un mal uso de los métodos de acceso de la clase.

```
class Felino{
.
.
.
   int ObtenerEdad()const;
   void AsignarEdad(int Edad);
.
.
.
};
```

```
Felino::Felino(int edad){
   cout << "Construyendo un felino de: " << edad << " anios de edad" << endl;
   Edad = edad;
}

// ObtenerEdad, metodo de acceso publico
// regresa el valor de la propiedad Edad
int Felino::ObtenerEdad()const{
   Edad *= 10;
   return Edad;
```

Código 16: Se ha agregado la palabra const para proteger los elementos privados de la clase

Al momento decompilar el código anterior mostrará error debido a que se ha puesto `const` lo cual implica que no puede modificar el valor, entonces, para que pueda funcionar el código, simplemento no hay que modificar `Edad` dentro de la función `ObtenerEdad`. A propósito en se dejo también la función asignar edad, observe que ésta función no tiene la palabra rerservada `const` debido a que el objetivos principal de esta función **si** es modificar el valor.

11. Uso de clases con otras clases como atributos

Considere el ejemplo de un triángulo, dicho triángulo está compuesto básicamente por tres puntos y las líneas que unen a éstos puntos, para efectos de programación se puede definir una clase de un triángulo y dentro de la clase definir los puntos, este enfoque no es muy práctico, en cambio, se utilizará el segundo enfoque en donde se genera un objeto tipo `punto` y se crearán sus métodos y después se creará un triángulo que contendrá tres objetos tipo `punto` y además se agregarán nuevos métodos.

La clase punto debe de tener como atributos las coordenadas x y y, como métodos debe de tener funciones que sean capaces de modificar los valores de las coordenadas, debe de tener el constructor predeterminado y otro constructor que inicializa las coordenadas. El archivo de cabezera que describe a la clase punto está dado por el siguiente código

```
#include <iostream>

using namespace std;

class Punto{
   private:
   float x;
   float y;
   public:
   Punto();
   Punto(float val_x, float val_y);
   void asignarX(float val_x);
```

```
   float obtenerX();
   void asignarY(float val_y);
   float obtenerY();
   void asignarXY(float val_x, float val_y);
   void mostrarCoordenadas();
};

//Constructor predeterminado
Punto::Punto(){}

// Constructor definido por usuario
// Construye e inicializa al punto
Punto::Punto(float val_x, float val_y){
   x = val_x;
   y = val_y;
}

void Punto::asignarX(float val_x){
   x = val_x;
}

float Punto::obtenerX(){
   return x;
}


void Punto::asignarY(float val_y){
   y = val_y;
}

float Punto::obtenerY(){
   return y;
}

void Punto::asignarXY(float val_x, float val_y){
   asignarX(val_x);
   asignarY(val_y);
}

void Punto::mostrarCoordenadas(){
   cout << "Coordenada en x: " << x << endl;
   cout << "Coordeanda en y: " << y << endl;
}
```

Código 17: Definición e implementación de la clase Punto, el archivo se debe de llamar punto.h

A continuación se muestra un programa sencillo que ayudará sólo a la verificación del

buen funcionamiento de la clase `Punto`

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

using namespace std;

int main(){
   Punto A;
   Punto B(1,0);
   cout << "Las coordenadas del punto A son: " << endl;
   A.mostrarCoordenadas();
   cout << endl << "Las coordenadas del punto B son: " << endl;
   B.mostrarCoordenadas();
}
```

Código 18: Archivo que ayuda a verificar el correcto funcionamiento de la librería punto.h

La clase `Triánuglo` está dada por el siguiente código:

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

using namespace std;

class Triangulo{
   private:
   Punto A;
   Punto B;
   Punto C;
   public:
   void asignarA(float val_x,float val_y);
   void asignarB(float val_x,float val_y);
   void asignarC(float val_x,float val_y);
   void mostrarCoordenadas();
};

void Triangulo::asignarA(float val_x,float val_y){
   A.asignarXY(val_x,val_y);
}

void Triangulo::asignarB(float val_x,float val_y){
   B.asignarXY(val_x,val_y);
}

void Triangulo::asignarC(float val_x,float val_y){
   C.asignarXY(val_x,val_y);
}
```

```
void Triangulo::mostrarCoordenadas(){
   cout << "Las coordenadas del punto A son: " << endl;
   A.mostrarCoordenadas();
   cout << endl << "Las coordenadas del punto B son: " << endl;
   B.mostrarCoordenadas();
   cout << endl << "Las coordenadas del punto C son: " << endl;
   C.mostrarCoordenadas();
}
```

Código 19: Definición e implementación de la clase triángulo

El programa donde estaría la función principal se muestra en la siguiente celda

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


int main(){
   Triangulo miTriangulo;
   miTriangulo.asignarA(0,0);
   miTriangulo.asignarB(5,0);
   miTriangulo.asignarC(0,9);
   miTriangulo.mostrarCoordenadas();

}
```

Código 20: Definición e implementación de la clase triángulo

12. Ejercicios

12.1.

Genere una clase se llame Perro que tenga como atributos privados, el nombre (string), edad (entero), peso en kilogramos (float) y raza (string), genere los métodos necesarios para poder modificar cada uno de los atributos, genere un constructor definido por usted y que inicialice al objeto, haga un método que se llame `imprimirDatos()` que muestre los datos del perro. Genere un programa principal en donde cree un perro con el constructor predeterminado y vaya asignando uno a uno los atributos utilizando los métodos de acceso que ha diseñado. Genere un objeto de la clase `perro` utilizando el constructor definido por el usuario e imprima los datos de los dos objetos tipo `perro`.

12.2.

Genere una clase que se llame `Jauria` que tenga 5 objetos tipo perro. Haga un programa en el que genere un objeto tipo `Jauria` e ingrese los datos de cada uno de los 5 perros, después muestre la información de cada perro.

13.

A la clase triángulo agrege un método que calcule la distancia entre los puntos y si las distancias son iguales que imprima que es un triángulo equilátero.

14.

Genere una clase que se llame `rectangulo` en donde si los cuatro lados son iguales imprima que es un cuadrado. Genere un programa principal para verificar el correcto funcionamiento de la clase y la libería.