Programación Avanzada

UNIDAD 5: PROGRAMACIÓN ORIENTADA A OBJETOS – 2DA PARTE

Unidad 5

Programación Orientada a Objetos. Clases y Objetos. Instanciación de objetos. El objeto this. El constructor. Extensión y herencia. Visibilidad y encapsulación

Visibilidad de una clase

Los miembros (atributos y métodos) pueden ser visibles fuera de la clase si se definen como ***public***, o no visibles si se definen como ***private***. Por lo tanto, dentro de una clase se van a tener dos secciones diferenciadas:

- **La sección privada:** Es la sección de miembros de la clase a la que no tiene acceso el exterior de la clase. Está oculta para los accesos desde fuera de la clase. Es como si se construyera un muro alrededor de los miembros de la clase para protegerlos de errores accidentales del resto del programa. Esta sección se marca con la palabra reservada ***private***.

- **La sección pública:** Es la sección de miembros de la clase a la que se tiene acceso desde el exterior de la clase. Esta sección se marca con la palabra reservada ***public***.

Es correcto incluir datos y funciones en la sección privada, y más datos adicionales y funciones en la parte pública. Pero en la mayoría de los casos prácticos, los datos se incluyen sólo en la sección privada, y en la sección pública se declaren funciones para acceder a ellos.

Existe una tercera sección denominada ***sección protegida***. Los miembros (atributos y métodos) que aquí se declaran son privados para el exterior, excepto para las clases que se *derivan* de una clase

Visibilidad en C++

Con las palabras ***public***, ***private*** y ***protected*** pueden declararse respectivamente como *públicos, privados o protegidos* los miembros de una clase, por lo que la sintaxis general de la definición de una clase es:

class <identificador> {

 [***public***:

 [<miembros atributos>]

 [<métodos miembro>]]

 [***protected***:

 [<miembros atributos>]

 [<métodos miembro>]]

 [***private***:

 [<miembros atributos>]

 [<métodos miembro>]]

};

Visibilidad en C++

Las partes ***public, protected y private*** en la definición de una clase pueden aparecer o no, y no importa su orden.

Para una clase existen tres tipos de usuarios:

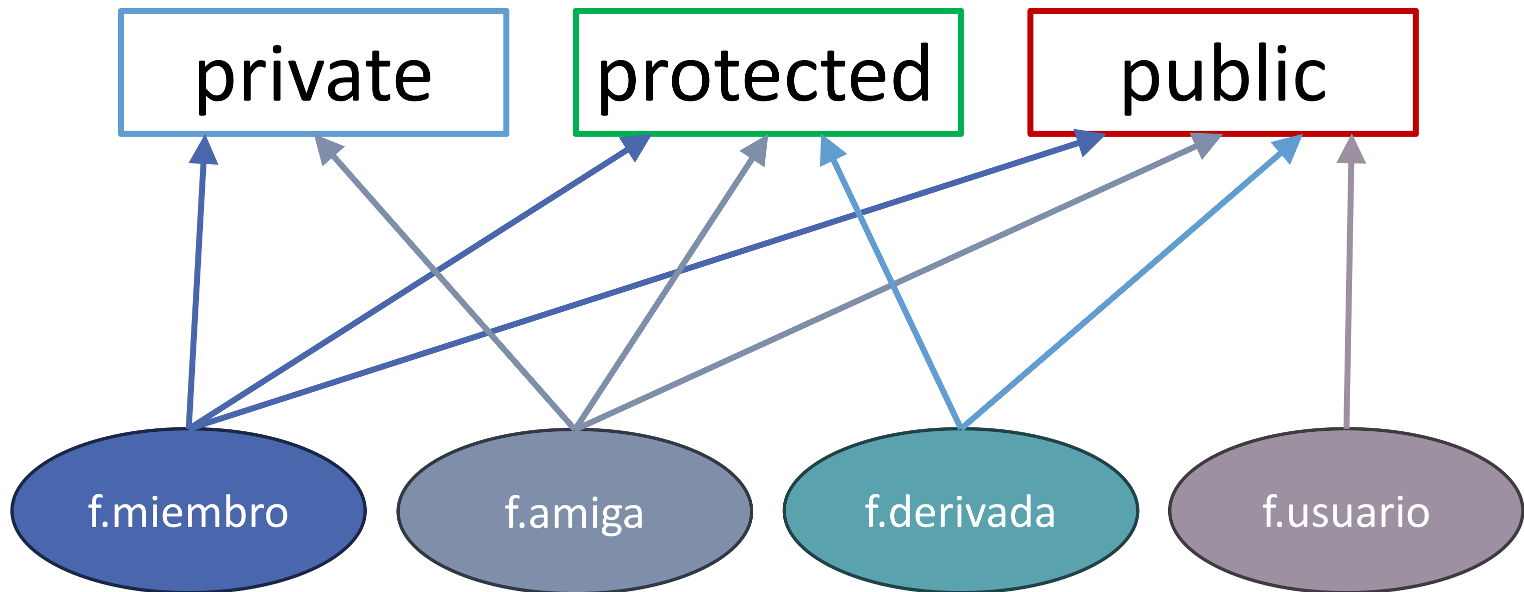
- la propia clase
- usuarios genéricos
- clases derivadas

Cada nivel de privilegio tiene asociada una palabra reservada:

- ***private:*** sólo la propia clase
- ***public:*** cualquier usuario
- ***protected:*** clases derivadas

Los métodos de la clase tienen acceso a todo lo declarado dentro de ésta. También tienen este privilegio un tipo de funciones (métodos) que no pertenecen a la clase pero que se declaran como amigas (friends).

Modos de acceso



Encapsulamiento

- Una clase debe ***ocultar*** su ***estructura*** y la ***implementación*** de sus servicios, haciendo ***públicas*** las ***interfaces*** de sus servicios.
- Sus **atributos** deberán ser también **privados**, y el acceso a éstos para el resto del programa debe producirse exclusivamente a través de los ***métodos públicos***.
- Los ***métodos*** que sean usados para la ***manipulación interna*** de datos o como auxiliares a otros métodos de la propia clase deben ser ***privados***.
- Los atributos de un objeto deben ser privados, y el acceso a éstos debe producirse exclusivamente a través de las funciones miembro públicas.

Constructores

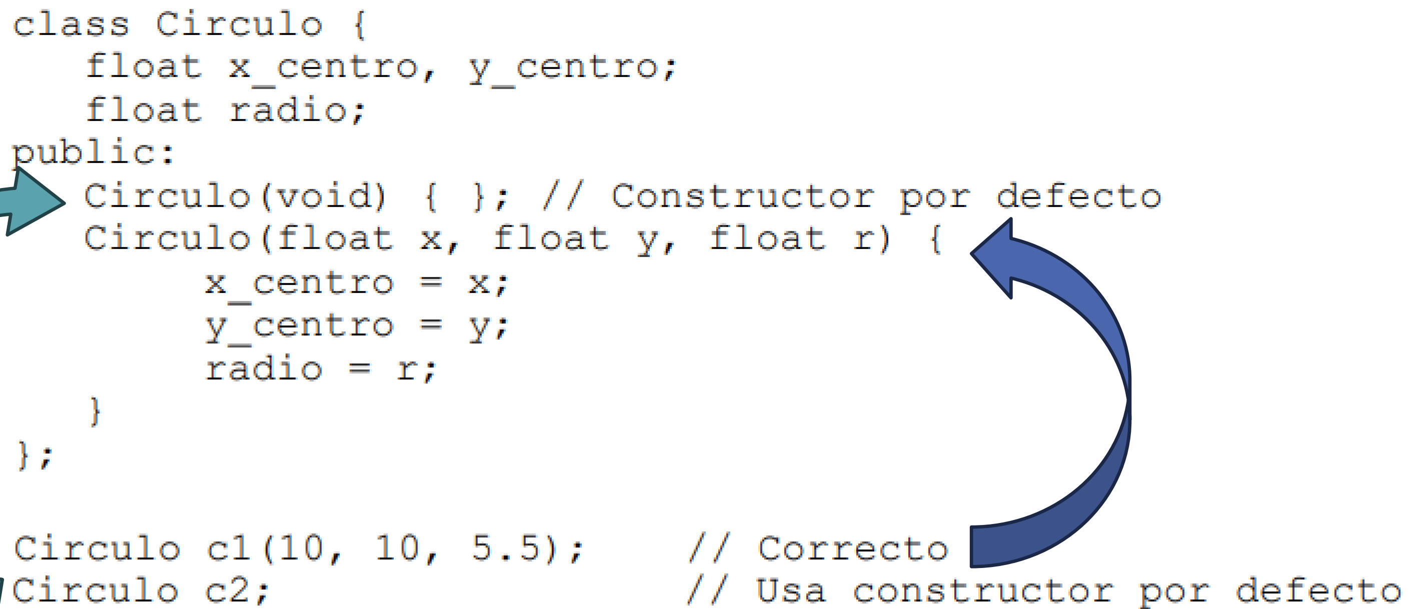
Normalmente se utilizan para iniciar el objeto cuando éste se crea.

Sus características son:

- Tiene en mismo nombre que la clase.
- Puede definirse ***inline*** o fuera de la declaración de la clase.
- Pueden aceptar parámetros, los cuales puede definirse con valores por defecto.
- Puede estar sobrecargado. Se puede declarar más de un constructor para una misma clase si todos ellos tienen diferente lista de argumentos. Esto es útil si se quiere iniciar los objetos de varias formas diferentes.
- No devuelven valores. No se puede especificar un valor de retorno cuando se declara un constructor, ni siquiera void. En consecuencia, un constructor no puede tener ninguna sentencia return.
- El constructor se ejecuta automáticamente cuando se crea un objeto de tipo clase. No se invoca nunca de forma explícita.
- El constructor se suele declarar en la sección pública. Si se hiciera en la sección privada sólo podría ser utilizado por los objetos que se creasen en las funciones miembro y en las funciones amigas.

Se denomina ***constructor por defecto de la clase*** al constructor que no tiene argumentos.

Constructores. Ejemplo



```
class Circulo {
   float x_centro, y_centro;
   float radio;
public:
   Circulo(void) { }; // Constructor por defecto
   Circulo(float x, float y, float r) {
        x_centro = x;
        y_centro = y;
        radio = r;
   }
};

Circulo c1(10, 10, 5.5);    // Correcto
Circulo c2;                 // Usa constructor por defecto
```

Constructores copia

Se denomina ***constructor copia*** a aquel constructor que crea un nuevo objeto, iniciándolo con los datos tomados de otro objeto ya existente. Este tipo de constructor tiene solamente un argumento: una **referencia constante** a un **objeto** de la **misma clase**.

Se invocan implícitamente cuando se hace una de estas cosas:

- Asignaciones entre objetos.
- Paso de parámetros-objeto por valor.

Los ***constructores copia*** tienen gran importancia debido a:

- Son el único medio de hacer una copia de un objeto.
- Se emplea cuando se pasan objetos por valor a una función.
- Se emplea cuando una función devuelve un objeto.

Constructores copia. Ejemplo

```
class MC {
      float m, n, o, p;
public:
      MC () { m=n=o=p=0.0;}          // Constructor 1
      MC (float i) {m=n=o=p=i;}      // Constructor 2
      MC (const MC &fuente) {        // Constructor copia
            m=fuente.m;
            n=fuente.n;
            o=fuente.o;
            p=fuente.p;
      }
      void Cambiar(float a=0.0, float b=0.0, float c=0.0,
                   float d=0.0);
      void Escribir(void);
};

// Funciones miembro
void MC::Cambiar(float a, float b, float c, float d) {
      m=a;  n=b;  o=c;  p=d;
}

void MC::Escribir(void) {
      cout << "\nValor de m:  " << m;
      cout << "\nValor de n:  " << n;
      cout << "\nValor de o:  " << o;
      cout << "\nValor de p:  " << p << "\n";
}
```

Constructores copia. Ejemplo

```
// Programa Principal
void main (void) {
    MC o1(3),   // La primera instancia utiliza el constructor 2
       o2,      // La segunda instancia utiliza el constructor 1
       o3(o1); // La tercera utilizará el constructor copia

    o1.Escribir();        // Saca m=n=o=p=3
    o2.Escribir();        // Saca m=n=o=p=0
    o3.Escribir();        // Saca m=n=o=p=3
    o3.Cambiar(5,6,7,8); // Cambiamos los valores del objeto 3
    MC o4(o3);            // Nueva instancia de MC
                          // utiliza el constructor copia

    o4.Escribir();        // Saca por pantalla m=5, n=6, o=7, p=8
    MC o5 = o4;           // Nueva instancia de MC
                          // utiliza también el constructor copia
    o5.Escribir();        // Saca por pantalla m=5, n=6, o=7, p=8
}
```

Funciones *friend*

Es posible que una función que no es miembro de una clase tenga acceso a la parte privada de esa clase, declarándola como ***friend*** (amiga) de la clase. El prototipo estará definido dentro de la clase, pero con la palabra reservada ***friend***. La definición de la función puede estar en cualquier sitio.

Las funciones ***friend*** no pueden ser ***inline***.

El formato de la declaración de funciones ***friend*** es el siguiente:

```
class ejemplo {
    <...>
    public:
        friend tipo funcion (parametros); // Prototipo
}
```

Funciones *friend*. Ejemplo

```
class Mesa;
class Silla {
   Color    color;
   Material material;
   int      patas;
public:
   Silla() {color=marron; material=madera; patas=4;}
   void PonColor(Color);
   void PonMaterial(Material);
   void PonPatas(int);
   friend int MismoColor(Silla, Mesa);
};

class Mesa {
   Color color;
   Material material;
   int patas;
public:
   Mesa() {color=transparente; material=cristal; patas=4;}
   void PonColor(Color);
   void PonMaterial(Material);
   void PonPatas(int);
   friend int MismoColor(Silla, Mesa);
};
```

Funciones *friend*. Ejemplo

```
// Funciones miembro de la clase Silla
void inline Silla::PonColor(Color c) { color=c; }
void inline Silla::PonMaterial(Material m) { material=m; }
void inline Silla::PonPatas(int num) { patas=num; }

// Funciones miembro de la clase Mesa
void inline Mesa::PonColor(Color c) { color=c; }
void inline Mesa::PonMaterial(Material m) { material=m; }
void inline Mesa::PonPatas(int num) { patas=num; }

// Función amiga a las dos clases
int MismoColor (Silla s, Mesa m) {
    if (s.color == m.color)
         return SI;
    else return NO;
}
```

Herencia

Una de las principales propiedades de las clases es la ***herencia***. Esta propiedad nos permite crear ***nuevas clases*** a partir de ***clases existentes***, conservando las propiedades de la clase original y añadiendo otras nuevas.

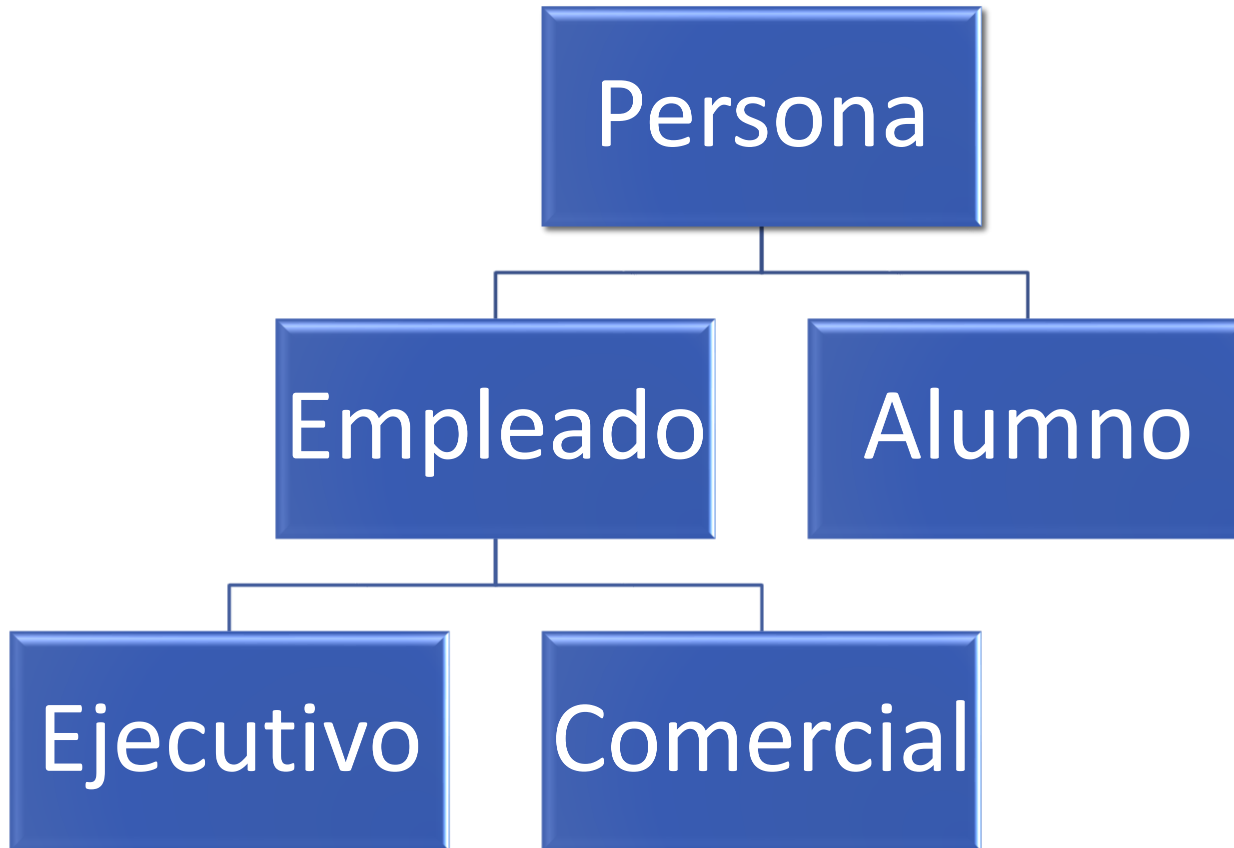
Jerarquía, clases base y clases derivadas

Cada nueva clase obtenida mediante ***herencia*** se conoce como ***clase derivada***, y las clases a partir de las cuales se deriva, ***clases base***. Además, cada clase derivada puede usarse como clase base para obtener una nueva clase derivada. Y cada clase derivada puede serlo de una o más clases base. En este último caso hablaremos de ***derivación múltiple.***

Esto nos permite crear una ***jerarquía de clases*** tan compleja como sea necesario.

Derivar clases nos permite encapsular diferentes partes de cualquier objeto real o imaginario, y vincularlo con objetos más elaborados del mismo tipo básico, que ***heredarán*** todas sus características.

Herencia. Ejemplo



Derivar clases, sintaxis

La forma general de declarar clases derivadas es la siguiente:

```
class <clase_derivada> :
   [public|private] <base1> [,[public|private] <base2>] {};
```

Derivar clases. Ejemplo

```
// Clase base Persona:
class Persona {
  protected:
   char nombre[40];
   int edad;

  public:
   Persona(char *n, int e);
   const char *LeerNombre(char *n) const;
   int LeerEdad() const;
   void CambiarNombre(const char *n);
   void CambiarEdad(int e);
};
```

```
// Clase derivada Empleado:
class Empleado : public Persona {
  protected:
   float salarioAnual;

  public:
   Empleado(char *n, int e, float s);
   float LeerSalario() const;
   void CambiarSalario(const float s);

};
```

Constructores de clases derivadas

Cuando se crea un objeto de una clase derivada, primero se invoca al constructor de la clase o clases base y a continuación al constructor de la clase derivada. Si la clase base es a su vez una clase derivada, el proceso se repite recursivamente.

Lógicamente, si no hemos definido los constructores de las clases, se usan los constructores por defecto que crea el compilador.

Constructores de clases derivadas

```
class ClaseA {
  public:
   ClaseA() : datoA(10) {
      cout << "Constructor de A" << endl;
   }
   int LeerA() const { return datoA; }

  protected:
   int datoA;
};

class ClaseB : public ClaseA {
  public:
   ClaseB() : datoB(20) {
      cout << "Constructor de B" << endl;
   }
   int LeerB() const { return datoB; }

  protected:
   int datoB;
};
```

```
int main() {
   ClaseB objeto;

   cout << "a = " << objeto.LeerA()
        << ", b = " << objeto.LeerB() << endl;

   return 0;
}
```

Ejercicio

El siguiente programa tiene fallos. Encontrarlos, explicar cuál se la causa, e indicar cómo se eliminaría el error.

```
1  #include <iostream>
2  using namespace std;
3
4  class Circulo {
5      int c_x, c_y;
6      float radio;
7      public:
8          void Circulo (int x, int y, float r) {
9              c_x=x;
10             c_y=y;
11             radio=r;
12         }
13 void Visualizar(void) {
14     cout << c_x << " " << c_y << " " << radio << "\n";
15 }
16
17 float Longitud(void) {
18     return 3.14159*2*radio;
19 }
20 };
21 int main () {
22     Circulo c1(5, 4, 4);
23     Circulo c2;
24     Circulo c3(5);
25     c1.Visualizar();
26     c2.Visualizar();
27     c3.Visualizar();
28     cout<<c1.Longitud << "\n";
29     return 0;
30 }
```