Grado en Ingeniería Información

PROGRAMACIÓN II - Sesión 2

Tema 1.

Introducción a la programación orientada a objetos

Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

UNIVERSIDAD NEBRIJA

T1. Introducción a la Programación Orientada a Objetos

ÍNDICE

UNIVERSIDAD NEBRIJA

T1. Introducción a la Programación Orientada a Objetos

1.1. Construcción de una clase

1.2. Atributos

1.3. Métodos. Implementación de los métodos de una clase

1.4. Creación de objetos

1.5. Paso de mensajes

1.6. Constructores y destructores

1.7. Constructor copia

ÍNDICE

Ventajas de la Programación Orientada a Objetos

- ✓ **Reutilización del código:** Si tenemos programada una clase ***botón***, podremos **instanciar** los objetos que necesitemos para nuestra interfaz.
- ✓ **Abstracción:** un programa será más sencillo de leer y comprender ya que nos permitirá **ocultar los detalles de la implementación** dejando **visibles sólo las partes más relevantes**.
- ✓ **Modificabilidad:** con un programa estructurado en clases/objetos es **más fácil realizar cambios** (facilita el mantenimiento del programa).
- ✓ **Fiabilidad:** al dividir el programa en partes más pequeñas será **más fácil realizar pruebas y aislar errores**.

Una ***clase*** representa la **idea abstracta bajo la cual debe "construirse", instanciarse** un ***objeto***.

La ***clase*** es una especie de ***molde*** que permite ***crear*** diferentes ***objetos*** de forma ***sencilla*** y ***compartir*** la **implementación de sus funciones (métodos)**.

La ***clase*** es el ***prototipado*** del ***objeto***.

Objetos

COCHE
Tipo Matricula
Arrancar() Avanzar() Frenar()

miCoche
Seat Ibiza M-3425-XC
Arrancar() Avanzar() Frenar()

tuCoche
Seat Panda M-6825-PC
Arrancar() Avanzar() Frenar()

Clase

Las ***clases*** facilitan al programador una herramienta que le permita definir un ***nuevo tipo de datos*** para utilizarlo como un **tipo predefinido de C++**.

Las ***clases***:

- **Limitan los accesos directos a la representación interna de un tipo.**
- **Indican las operaciones básicas definidas para el tipo.**

Una ***clase*** tiene que diferenciar entre:

- **Interfaz:** muestra la forma para poder ***utilizar la clase***.
- **Implementación:** contiene y oculta los ***detalles del funcionamiento interno de la clase***.

Funciones de un *programador*:

Usuario: *utiliza la clase*.

- Sólo puede **utilizar** los **objetos de una clase** a través de la **interfaz** definida para los **métodos**.
- **No** puede **acceder** a los detalles internos de la **implementación**.

Implementador: *diseña y construye la clase*.

- Define su **interfaz** (cabecera de los métodos).
- **Desarrolla** los detalles internos de su **implementación** (atributos y cuerpo de los métodos).
- Tiene **acceso total a la clase**.

Declaración de una ***clase*** en C++ se hace con la palabra reservada ***class***:

```
class NombreClase  {
    [nivel de acceso 1:]
    // declaración de atributos
        tipo_dato dato1;
        tipo_dato dato2; […]
    [nivel de acceso 2:]
    // declaración de métodos
        tipo_retorno metodo1 (parametros);
        tipo_retorno metodo2 (parametros); […]
};
```

Clases - Ejemplo

Nombre de la clase

```
class Persona  {
   public:
   // declaración de atributos públicos
      std::string   nombre;
      int   edad;

   // declaración de métodos públicos
      void setNombre (std::string const &n);
      void setEdad (int a);

      std::string getNombre () const;
      int getEdad () const;
      void mostrar () const;
};
```

Indica el acceso público

Atributos de la clase (variables miembro)

Métodos de la clase (funciones)

UNIVERSIDAD NEBRIJA

T1. Introducción a la Programación Orientada a Objetos

ÍNDICE

Métodos

Permiten acceder al **estado** (***atributos***).

Provocan **acciones** que:

- Modifican el **estado** (***atributos***).
- Permiten la **interacción con otros objetos** (***mensajes***).

Funcionalidades:

- **Obtención** o **modificación** directa del **estado** del objeto.
- **Realización** de algún cálculo que **modifique el estado** del objeto.
- **Notificar** al objeto que tiene que hacer alguna **actividad** o **interactuar** con otros objetos.

Métodos

La **implementación de los métodos** de una clase en C++ se realiza, normalmente, **fuera del bloque de definición de la clase**:

class {

... }

Su sintaxis es similar a la de una función, excepto que el **nombre del método debe estar precedido por el nombre de la clase de la que forma parte y el operador de resolución de ámbito (::)**.

```
tipo_retorno NombreClase::nombreMetodo (parámetros)
{
    // Implementación del metodo.
}
```

Todos los métodos de una clase, con o sin parámetros, reciben como **parámetro de entrada/salida *implícito*** el **objeto al que se va a aplicar el método**.

La **referencia a los atributos del parámetro *implícito*** se hace escribiendo los **nombres de los atributos**.

Por defecto, **los métodos de una clase pueden modificar los atributos del objeto receptor** (**parámetro *implícito***).

```
void Persona::setEdad (int a) {
        edad = a;
}
```

Los **parámetros** de un método o de una función definidos como ***const*** **no se podrán modificar.**

```
void Persona::setNombre (std::string  const  & n) {
        nombre = n;
}
```

Cuando interesa indicar que un **método no va a modificar ningún atributo** del objeto receptor se añade la palabra clave ***const*** al final de la declaración del método (tanto en el **interfaz** como en la **implementación**).

```
std::string Persona::getNombre() const
{     return nombre;          }

int Persona::getEdad() const
{     return edad;            }

void Persona::mostrar () const {

      std::cout << "\n\n\tNombre: " << nombre;

      std::cout << "\n\tEdad: " << edad;

}
```

UNIVERSIDAD NEBRIJA

T1. Introducción a la Programación Orientada a Objetos

ÍNDICE

Objetos

Definición de objetos de una clase se hace como la declaración de una variable, donde ahora el tipo es la clase:

NombreClase nombreObjeto;

Ejemplos:

```
Persona persona1, persona2;
shared_ptr<Persona> persona3 = make_shared<Persona> ();
```

Las **operaciones** que se pueden realizar sobre un **objeto** de una **clase** son:

- Definidas en el **interfaz** de la clase (parte ***public***).
- **Asignación**.
- Paso de **parámetros**, por **valor** o por **referencia**. Si no queremos que el parámetro sea modificado, le añadiremos ***const***.

La aplicación de **métodos** sobre un objeto se realiza mediante el paso de ***mensajes***.

La **sintaxis de un mensaje** es similar a la de la **llamada a una función**, salvo que el **nombre del método** va **precedido** por el **nombre del objeto** (parámetro ***implícito***) al que **se aplica el método utilizando el operador "punto" (.) o el operador "flecha" (->), según se trate de objetos normales o punteros**.

nombreObjeto.nombreMetodo (parámetrosMetodo);

Ejemplo:

```
persona1.nombre = "Maria";
persona1.setEdad(35);
persona1.mostrar();
```

El ***usuario*** manipula los ***objetos*** a través del ***paso de mensajes*** a dichos objetos.

```
// Objeto persona1
persona1.nombre = “Maria”;
persona1.setEdad(35);
persona1.mostrar();

// Objeto persona2
persona2.setNombre (“Carlos”);
persona2.edad = 40;
std::cout << “\n\tHola ” << persona2.getNombre() << endl;

// Objeto persona3 de tipo puntero
persona3->setNombre(“Eva”);
std::cout << “\n\tEstoy con “ << persona3->getNombre;
```

Un método puede recibir parámetros ***explícitos*** de la clase a la que pertenece.

El acceso a los atributos de los parámetros ***explícitos*** se hace utilizando la **notación punto** (**.**) usada en las **estructuras** (struct).

```
bool Persona::esCoetanio (Persona const &a) {
    if ( edad == a.edad) return true;
    else return false;
}
```

Los **atributos** y los **métodos** de una clase son **siempre privados**, **salvo que se indique otra cosa**.

Los **modos de acceso** controlan **quién accede a los atributos o métodos de la clase**:

- ***Private*** Solo los objetos de la **propia clase o de una clase amiga** (***friend***) pueden acceder a los atributos y métodos. Esta parte de la clase corresponde, normalmente, a la **implementación** de la clase.
- ***Public*** **Cualquier objeto de la aplicación puede acceder** a los atributos y métodos así definidos. Esta parte define la **interfaz** de la clase.
- ***Protected*** Acceden a los atributos y métodos sólo los objetos de la **propia clase, las clases derivas de ella y clases amigas** (***friend***).

Acceso desde **diferentes partes de la aplicación** a los **atributos** y **métodos** declarados en cada uno de los tres posibles bloques de una **clase**.

Acceso desde / Bloque	**funciones miembro**	clases ***amigas***	clases ***derivadas***	**otras clases**
public	**Si**	**Si**	**Si**	**Si**
private	**Si**	**Si**	**No**	**No**
protected	**Si**	**Si**	**Si**	**No**

```
class nombreClase
{
    private:
    // Implementación: declaración de atributos y métodos privados

    public:
    // Interfaz: declaración de atributos y métodos públicos

    protected:
    // Declaración de atributos y métodos protegidos

};
```

Parte privada: sólo los **atributos de la clase** y **algunos tipos intermedios** que puedan ser necesarios.

Parta pública: suelen aparecer solamente **las declaraciones** (**cabeceras** o **prototipos**) **de los métodos de la clase**.

```
class nombreClase
{
   private:
   // implementación de la clase y los atributos
   public:
   // interfaz de la clase
};
```

En C++, la implementación de los métodos de la clase, normalmente, se realiza en otra parte.

Clases - Ejemplo

```
class Persona  {
    private:
    // declaración de atributos privados
       std::string   nombre;
       int   edad;
    public:
    // declaración de métodos públicos
       void setNombre (std::string const &n);
       void setEdad (int a);

       std::string getNombre () const;
       int getEdad () const;
       void mostrar () const;
       bool esCoetanio (Persona const &a);
};
```

Implementación

Estado

Interfaz

Comportamiento

Este cambio tiene ciertas implicaciones:

```
Persona persona1, persona2;

shared_ptr<Persona> persona3 = make_shared<Persona>();

//Objeto persona1
persona1.nombre = "Maria";

persona1.setEdad(35);

persona1.mostrar();

//Objeto persona2

persona2.setNombre ("Carlos");

persona2.edad = 40;

std::cout << "\n\tHola " << persona2.getNombre() << endl;

//Objeto persona3

persona3->setNombre("Eva");

std::cout << "\n\tEstoy con " << persona3->getNombre;
```

Este cambio tiene ciertas implicaciones:

```
Persona persona1, persona2;
shared_ptr<Persona> persona3 = make_shared<Persona>();
//Objeto persona1
persona1.nombre = "Maria";    // ERROR porque nombre es privado
persona1.setEdad(35);
persona1.mostrar();
//Objeto persona2
persona2.setNombre ("Carlos");
persona2.edad = 40;           // ERROR porque edad es privado
std::cout << "\n\tHola " << persona2.getNombre() << endl;
//Objeto persona3
persona3->setNombre("Eva");
std::cout << "\n\tEstoy con " << persona3->getNombre;
```

Hay que utilizar los métodos *get* y *set*:

```
Persona persona1, persona2;

shared_ptr<Persona> persona3 = make_shared<Persona>();

string nom;

//Objeto persona1
cout << "\n\n\tTeclee su nombre: ";

getline(cin, nom);


persona1.setNombre(nom);

persona1.setEdad(35);


cout << "\n\n\tSu nombre es: " << persona1.getNombre() << " y su edad "

        << persona1.getEdad();
```

Hacer los **atributos** o **variables miembro privadas** permite tener más **control** sobre ellas. Por ejemplo, la edad no puede tener un **valor negativo**, por tanto el método ser debería ser:

```
void Persona::setEdad (int a) {
        if (a < 0)
        {
           edad = 0;
           cout << "\n\n\tError, la edad toma el valor 0" << endl;
        }
        else
        {
           edad = a;
        }
}
```

El siguiente programa sería:

```
Persona persona1;

string nom;

//Objeto persona1
cout << "\n\n\tTeclee su nombre: ";

getline(cin, nom);


persona1.setNombre(nom);

persona1.setEdad(-35);   // Error, la edad no puede ser negativa


cout << "\n\n\tSu nombre es: " << persona1.getNombre() << " y su edad "

        << persona1.getEdad();
```