Introducción a Clases

Agustín J. González
Versión original de Kip Irvine

ELO326: Seminario II

2do. Sem. 2001

Programación Orientada a Objetos (POO)

- Literalmente, ”Programación con objetos"
- Modelamiento del mundo real
  - Los objetos en el programa surgen de los objetos de la especificación del proyecto
- Originalmente, C++ fue llamado "C with classes” (C con clases) (*The Design and Evolution of C++* by Stroustrup.)

Clases y Objetos

- Una clase es un tipo de datos definido por el usuario.
  - Provee un “molde” o ”diseño" para múltiples objetos del mismo tipo.
- Un objeto es una instancia de una clase.
  - Cada objeto tiene una localización única en memoria, y valores únicos para sus atributos
- A clase contiene atributos (almacenan el estado del objeto) y operaciones (definen sus responsabilidades o capacidades):

Atributos y Operaciones

- Atributos son las características comunes entre todas las instancias de un clase
  - En C++ y Java son implementadas como variables (también llamadas miembros dato)
- Operaciones son implementadas en C++ y Java como funciones (también llamados miembros función o métodos)

(Objetos del mundo real poseen atributos y operaciones)

Ejemplo: Clase Point

Una clase contiene atributos (variables) y operaciones (funciones):

```
class Point {
  void Draw();
  void MoveTo( int x, int y );
  void LineTo( int x, int y );

  int m_X;
  int m_Y;
};
```

Operaciones

Atributos

Encapsulación

- Encapsulación es el acto de ocultar los detalles de implementación de una clase de los usuarios de la clase.
- Los usuarios de una clase sólo deberían interactuar con la clase a través de su interface.
- La implementación podría cambiar, pero esto debería tener efecto mínimo en los usuarios de la clase.
- Los especificadores private y protected fuerzan encapsulación en clases C++.

Especificadores de Acceso Público y Privado

- Todos miembros precedidos por el especificador public son visibles fuera de la clase
  - por ejemplo, un miembro público es visible desde el main(), como es el caso de cin.get():

```
cin.get();  // cin es el objeto, get es la
  función de acceso público.
```

- Todos los miembros precedidos por el especificador private quedan ocultos para funciones fuera de la clase.
  - Ellos pueden ser sólo referenciados por funciones dentro de la misma clase
- Miembros precedidos por protected pueden ser accedidos por miembros de la misma clase, clases derivadas y clases amigas (friend)

Cuadro Resumen (X representa que tiene acceso):

Espicificador	Miembros de clase	Friend	Clases derivadas	Otros
Privado	X	X		
Protected	X	X	X	
Public	X	X	X	X

Clase Point, *revisado*

Esta versión usa public y private:

```
class Point {
public:
  void Draw();
  void MoveTo( int x, int y );
  void LineTo( int x, int y );

private:
  int m_X;
  int m_Y;
};
```

Ejemplo: Clase Automóvil

Imaginemos que queremos programar la simulación de una automóvil:

- Atributos: marca, número e puertas, número de cilindros, tamaño del motor
- Operaciones:entrada, despliegue, partir, parar, chequear_gas

La elección de qué atributo y operaciones incluir depende de las necesidades de la aplicación.

Clase Automóvil

```
class Automobile {
 public:
   Automobile();
   void Input();
   void set_NumDoors( int doors );

   void Display();
   int get_NumDoors();

 private:
   string Make;
   int    NumDoors;
   int    NumCylinders;
   int    EngineSize;
 };
```

Clasificación de Funciones Miembros en una Clase

- Un accesor es una función que retorna un valor desde su objeto, pero no cambia el objeto (sus atributos). Permite acceder a los atributos del objeto.
- Un mutador es una función que modifica su objeto
- Un constructor es una función con el mismo nombre de la clase que se ejecuta tan pronto como una instancia de la clase es creada.
- Un destructor es una función con el mismo nombre de la clase y un ~ antepuesto ~Automobil()

Ejemplo...

Clase Automóvil

```
class Automobile {
public:                                 // public functions

  Automobile();                         // constructor

  void Input();                         // mutador
  void set_NumDoors( int doors );     // mutador

  void Display();                       // accesor
  int get_NumDoors();                   // accesor
  ~Autiomobil();                        // Destructor
private:                                // private data
  string Make;
  int    NumDoors;
  int    NumCylinders;
  int    EngineSize;
};
```

Creación de Objetos

- Cuando declaramos una variable usando una clase como su tipo de dato, estamos creando una instancia de la clase.
- Una instancia de una clase es también llamada un objeto u objeto clase (ej. objeto auto)
- En el próximo ejemplo creamos un objeto Automobile y llamamos a sus funciones miembros.

Ejemplo...

Creando y accediendo un Objeto

```
void main()
{
   Automobile myCar;

   myCar.set_NumDoors( 4 );

   cout << "Enter all data for an automobile: ";
   myCar.Input();

   cout << "This is what you entered: ";
   myCar.Display();

   cout << "This car has "
        << myCar.get_NumDoors()
        << " doors.\n";

}
```

Constructores

- Un constructor es una función que tiene el mismo nombre de la clase. Puede aparecer varias veces con distinta lista de parámetros.
- Un constructor se ejecuta cuando el objeto es creado, es decir tan pronto es definido en el programa. Ej. Cuando la función es llamada en el caso de datos locales, antes de la función main() en el caso de objetos globales.
- Un default constructor no tiene parámetros.
- Los constructores usualmente inicializan los miembros datos de la clase.
- Si definimos un arreglo de objetos, el constructor por defecto es llamado para cada objeto:

```
Point drawing[50];
// calls default constructor 50 times
```

Implementación de Constructores

Un constructor por defecto para la clase Point podría inicializar X e Y:

```
class Point {
public:
  Point() {
    m_X = 0;
    m_Y = 0;
  }
private:
  int m_X;
  int m_Y;
};
```

Funciones Out-of-Line

- Todas las funciones miembro deben ser declaradas (prototipo) dentro de la definición de una clase.
- La implementación de funciones no triviales son usualmente definidas fuera de la clase y en un archivo separado.
- Por ejemplo para el constructor Point:

```
Point::Point()
{
  m_X = 0;
  m_Y = 0;
}
```

Los :: le permite al compilador saber que estamos definiendo la función Point de la clase Point. Este también es conocido como operador de resolución de alcance.

Clase Automobile (revisión)

```
class Automobile {
public:
  Automobile();
  void Input();
  void set_NumDoors( int doors );

  void Display() const;
  int get_NumDoors() const;

private:
  string Make;
  int    NumDoors;
  int    NumCylinders;
  int    EngineSize;
};
```

Implementaciones de las funciones de Automobile

```
Automobile::Automobile()
{
  NumDoors = 0;
  NumCylinders = 0;
  EngineSize = 0;
}

void Automobile::Display() const
{
  cout << "Make: "  << Make
    << ", Doors: "  << NumDoors
    << ", Cyl: "    << NumCylinders
    << ", Engine: " << EngineSize
    << endl;
}
```

Implementación de la Función de entrada

```
void Automobile::Input()
{
   cout << "Enter the make: ";
   cin >> Make;
   cout << "How many doors? ";
   cin >> NumDoors;
   cout << "How many cylinders? ";
   cin >> NumCylinders;
   cout << "What size engine? ";
   cin >> EngineSize;
}
```

Sobrecarga del Constructor

Múltiples constructores pueden existeir con diferente lista de parámetros:

```
class Automobile {
public:
   Automobile();

   Automobile( string make, int doors,
       int cylinders, int engineSize );

   Automobile( const Automobile & A );
   // copy constructor
```

Invocando a un Constructor

// muestra de llamada a constructor:

```
Automobile myCar;

Automobile yourCar("Yugo",4,2,1000);

Automobile hisCar( yourCar );
```

Implementación de un Constructor

```
Automobile::Automobile( string p_make, int doors,
                        int cylinders, int engineSize )
{
  Make = p_make;
  NumDoors = doors;
  NumCylinders = cylinders;
  EngineSize = engineSize;
}
```

Constructor con Parametros (2)

Algunas veces puede ocurrir que los nombres de los parámetros sean los mismos que los datos miembros:

```
NumDoors = NumDoors;              // ??
NumCylinders = NumCylinders;      // ??
```

Para hacer la distición se puede usar el calificador this (palabra reservada), el cual es un puntero definido por el sistema al objeto actual:

```
this->NumDoors = NumDoors;
this->NumCylinders = NumCylinders;
```

Lista de Inicialización

Usamos una lista de inicialización para definir los valores de las miembros datos en un constructor.. Esto es particularmente útil para miembros objetos y miembros constantes :

```
Automobile::Automobile( string make, int doors,
   int cylinders, int engineSize ) :
   Make(make),
   NumDoors(doors),
   NumCylinders(cylinders),
   EngineSize(engineSize)
{

}
```

Esquema para identificar Nombres

- Microsoft normalmente usa prefijos estándares para los miembros datos: m_
- Microsoft también usa letras como prefijos para tipos de datos:
  - s = string
  - n = numeric
  - b = boolean
  - p = pointer
- En ocasiones usan prefijos más largos:
  - str = string, bln = boolean, ptr = pointer, int = integer, lng = long, dbl = double, sng = float (*single precision*)

Ejemplo: En el caso de Miembros datos para clase Automobile

Así es como se verían los miembros datos de la case Automobile usando prefijos:

```
class Automobile {

private:
  string m_strMake;
  int    m_nNumDoors;
  int    m_nNumCylinders;
  int    m_nEngineSize;
};
```

Una ventaja es que miembros datos nunca son confundidos por variables locales.

Uso de Const

- Siempre usamos el modificador const cuando declaramos miembros funciones si la función no modifica los datos del objeto:

```
void Display() const;
```

- Esto disciplina a los usuarios de la clase a usar constantes correctamente en sus programas
- Un detalle que puede ser molestoes que un objeto no puede ser pasado como referencia constante a una función si dentro de la función se invoca un método no definido constante para ese objeto..

Ejemplo ...

Ejemplo: Uso de Const

La función garantiza que no modificará el parámetro

```
void ShowAuto( const Automobile & aCar )
{
  cout << "Example of an automobile: ";
  aCar.Display();
  cout << "---------------------------\n";
}
```

¿Qué tal si la función Display() no está definida como función constante?

Ideas para el Diseño de Clases

- *Descriptiva:* Una clase debería corresponder a una entidad o proceso en el dominio del problema
- *Simple:* Una clase describe sólo los atributos y operaciones necesarios para el problema en cuestión.
- *Operaciones* deberían describir acciones o modificar datos de la calse
- *Cohesiva:* Una clase es autocontenida y no depende de los detalles de implemantación de otras clases.

Fin