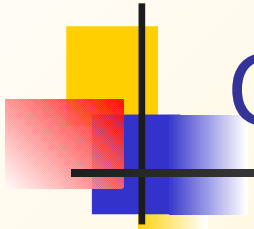
UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Clases en C++

Agustín J. González
ELO329



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 o categoría.
- Un objeto es una instancia de una clase.
  - Cada objeto tiene una localización única en memoria, y valores únicos para sus atributos
- Una clase contiene atributos (almacenan el estado del objeto) y operaciones (definen sus responsabilidades o capacidades).

Ejemplo: Clase Point en C++

- 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

Ojo ; delimitardor de definición de tipo

Calificadores de Acceso Público y Privado: es similar a Java

- Todos miembros precedidos por el calificador 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 calificador ***private*** quedan ocultos para funciones o métodos fuera de la clase.
  - Ellos pueden ser sólo accedidos por métodos 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):

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

Clase Point, revisado

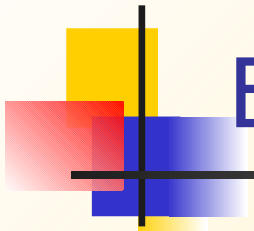
- Esta versión usa public y private:

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

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

Estructura Básica de programas C++

- En C++ es recomendado separar en distintos archivos la definición de una clase de su implementación.
- Crear un archivo de encabezado “clase.h”, en él podemos de **definición de la clase**, es decir los prototipos de métodos y los atributos.
- En otro archivo “clase.cpp” ponemos la **implementación** de cada método. En éste debemos incluir el archivo de encabezado “clase.h”
- Podemos implementar varias clases por archivo y un .h puede tener la definición de varias clases, pero se recomienda hacer un .h y .cpp por clase.



Ejemplo: Clase Automóvil

- Imaginemos que queremos modelar un 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 dependen 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 un método 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: Similar a Java

- 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.
- Siempre dispondremos de un construcción por omisión, el cual 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() {   // función inline
    m_X = 0;
    m_Y = 0;
  }
private:
  int m_X;
  int m_Y;
};
```

Funciones Out-of-Line

- Todas las funciones miembro deben ser declaradas (su 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;
}
```

  - El símbolo :: 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;
}
```

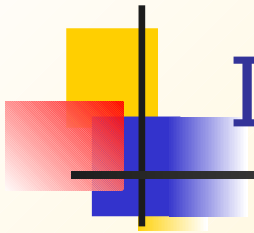
Sobrecarga del Constructor

- Múltiples constructores pueden existir con diferente lista de parámetros:

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

    Automobile( string make, int doors,
        int cylinders, int engineSize=2 );

    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 Parámetros (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 distinción 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 (en este caso obligatorio) :

```
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

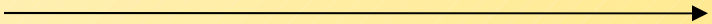
- Así es como se verían los miembros datos de la case Automobile usando prefijos:

```
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 molesto es 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?