UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Electrónica

Clases en C++

Agustín J. González
ELO329

Clases y objetos en C++

- El concepto de clase y objeto ya visto para Java no cambia en C++. Estos conceptos son independientes del lenguaje.
- Lenguaje: en los textos de C++ es común referirse a los atributos como los miembros dato de la clase y a los métodos como miembros función.
- Mientras en Java todos los objetos son creados y almacenados en el Heap (o zona de memoria dinámica), en C++ los objetos se pueden ubicar en stack (determinada a tiempo de compilación) o en memoria dinámica (solicitada a tiempo de ejecución).
- Al definir un objeto estáticamente, éste es creado inmediatamente. En java definimos sólo el nombre y para crear el objeto usamos new.
- Para crear objetos en el heap en C++, usaremos punteros a objetos que, como en Java, crearemos con new.

Estructura Básica de programas C++

- En C++ es recomendado separar en distintos archivos la definición de una clase de su implementación.
- Se crear un archivo de encabezado “clase.h”, en él podemos la **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.

Estructura de archivos

Estilo Java

```
import java.util.*;

class Employee {
   public Employee(String n, double s){
      name = n;
      salary = s;
   }
   public String getName() {
       return name;
   }
   public double getSalary() {
       return salary;
   }
   public void raiseSalary(double byPercent) {
       double raise = salary * byPercent / 100;
       salary += raise;
   }
   private final String name;
   private double salary;
}
```

Estilo C++

Employee.h

```
class Employee {
 public:
    Employee (string n, double s);
    string getName();
    double getSalary();
    void raiseSalary(double byPercent);
 private:
    Const String name;
    Doubel salary;
};
```

Employee.cpp

```
Employee::Employee(string n double s) {
  name = n;
  Salary = s;
}
string Employee::getName(){
  Return name;
}
double Employee::getSalary(){
  Return salary;
}
 // el resto  .....
```

Ejemplo: Definición de Clase Point

- En: Point.h

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

Métodos

Atributos

Ojo ; delimitador de definición de tipo. Igual a definir en C:

```
struct amigo {
  string nombre;
  Int edad;
};
```

Calificadores de Acceso Público y Privado: es similar a Java

- Los 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 es el objeto, get es la función de acceso público.
  - Valor definido por omisión en estructuras.
- Los miembros precedidos por el calificador private quedan ocultos para funciones o métodos fuera de la clase. **Valor usado por omisión en clases.**
- Miembros precedidos por protected pueden ser accedidos por miembros de la misma clase y clases derivadas.
- Las clases y funciones amigas (más adelante) tienen acceso a todo.
- Cuadro Resumen (√ significa que sí tiene acceso)

Calificador	Miembros de su clase	Friend	Clases derivadas	Otros
Privado o ausente	√	√		
Protected	√	√	√	
Public	√	√	√	√

Ejemplo: Clase Automóvil

- Imaginemos que queremos modelar un automóvil:
  - Atributos: marca, número de puertas, número de cilindros, tamaño del motor
  - Operaciones: fijar y obtener número de puertas, entrada y despliegue de atributos, partir, parar, revisar_gas

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 un método que modifica su objeto
- Un constructor es un método con el mismo nombre de la clase que se ejecuta tan pronto como una instancia de la clase es creada.
- Un **destructor** es un método el mismo nombre de la clase y una virgulilla (~) antepuesta. Ej.: ~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
     ~Autiomobile();                              // Destructor

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

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 se ejecuta cuando el objeto es creado, es decir tan pronto es definido en el programa. Ej. Esto es antes de la función main() en el caso de objetos globales y cuando una función o método es llamado en el caso de datos locales.
- En ausencia de constructores, C++ define un construcción por omisión, el cual no tiene parámetros.
- Debemos crear nuestro constructor por defecto si tenemos otros constructores.
- 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;
  }                    // Ojo no va ; aquí, es el fin del método.
private:
  int m_X;
  int m_Y;
};
```

Funciones Out-of-Line

- Todos los métodos deben ser declarados (el 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, en lugar de ponerlas in-line en la definición de la clase.
- Por ejemplo para el constructor Point, la implementación “of-line”:

```
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

- Como en Java, múltiples constructores pueden existir con diferente lista de parámetros:

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

    Automobile( string make, int doors,
        int cylinders, int engineSize=2);   // esta notación
                    // señala que este argumento es opcional, ante
                    // equivale a tener 2 constructores en Java.
    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 ) // ojo no va =2
{
  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 obligado para miembros constantes:

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

  {

  // notar asignación previa al cuerpo.

  // Esta es la forma obliga de inicializar

  // atributos constantes **const**

  }

Destructores

- Una diferencia importante con Java es la presencia de destructores. Lo más cercano en java es el método finalize() de la clase Object.
- Java tiene un proceso de “recolección de basura” por lo que hace los destructores innecesarios.
- En C++ el destructor se invoca en forma automática justo antes que la variable sea inaccesible para el programa.
- El método destructor no tiene parámetros, se llama igual que la clase y lleva un signo ~ como prefijo.
- Ej: Automobile::~Automobile() {} // éste se podría omitir en clase Automobile.