UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Electrónica

Clases en C++

Agustín J. González

ELO329

Departamento de Electrónica, UTFSM, Valparaíso, Chile

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 (zona de memoria dedicada al proceso y determinada durante compilación) o en memoria dinámica (solicitada durante ejecución).
- Al definir un objeto estáticamente en C++, su creación queda determinada. En cambio, en java se define el nombre y el objeto se crea usando new.
- Para crear objetos en el heap, en C++, se usa punteros a objetos y, como en Java, se crean con new. En este caso cambia la notación pero na semántica es similar a la de Java.

Ubicación de Objetos en C++ (para ir notando diferencia)

- Persona juan; /* *aquí de define un nombre y se crea el objeto en memoria asociada al programa* */
- Persona juan("Juan"); /* *ídem al anterior invocando otro constructor* */
- Persona *pJuan; /* *puntero a una instancia de Persona, es lo más parecido a Java* */
- Persona *pJuan; /**creamos puntero a Persona**/
  pJuan = new Persona("Juan"); /**Creamos instancia en heap, como en Java* */
- Hay más formas que veremos en detalle más adelante:
- Persona &rJP= Juan; /**un objeto Juan, con un alias rJP referencia a Juan**/
- Mientras en Java teníamos una forma y notación acá tenemos varias.

Estructura Básica de programas C++

- En C++ es recomendado separar en distintos archivos la definición de una clase por un lado y la implementación de ésta en otro archivo.
- Se crea un archivo de encabezado “clase.h”, en él podemos la **definición de la clase**, es decir los atributos y prototipos de métodos.
- En otro archivo “clase.cpp” ponemos la **implementación** de cada método. En este archivo 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 un .cpp por cada 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++

Archivo Employee.h

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

Archivo Employee.cpp

```
#include "Employee.h"
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;
};
```

Notar calificador de acceso (visibilidad): Define inicio de miembros con esa visibilidad. Omisión equivale a privado.

Métodos

Atributos

Ojo ; delimitador de definición de tipo. Igual al 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.
  - Es el calificador por omisión en estructuras (uso de struct en lugar de class).
- Los miembros precedidos por el calificador private quedan ocultos para funciones o métodos fuera de la clase. **Calificador 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.

Calificador	Miembros de su clase	Friend	Clases derivadas	Otros
Privado o ausente	√	√	No	No
Protected	√	√	√	No
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

```
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.: ~Automobile()

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
     ~Automobile();                     // Destructor

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

Creando y accediendo un Objeto

```
void main()   {
    Automobile myCar;      // objeto es creado

    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";
}
```

- Suponemos que la clase ya fue implementada en Automobile.cpp
- Función main no está asociada a una clase!. C++ no es 100% orientado a objetos.
- cout es un objeto predefinido para la salida estándar.
- cout << “saludo”; equivale a
- cout.operator<<(“saludo”);
- Es una forma “elegante” para expresar la escritura de datos en out.

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
                      // a diferencia de Java aquí ya tenemos 50 Puntos.
```

Implementación de Constructores

- Podemos implementar un método en la definición de la clase. Se conoce como código “inline”
- Un constructor por defecto para la clase Point podría inicializar X e Y:

```
class Point {
public:
  Point() {   // función inline , usarla solo si
    m_X = 0;    // implementación es simple.
    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” sería:

```
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;     // usamos const cuando el método
      int get_NumDoors() const;  // es un "accesor", no cambia atributos.
                                 // No es obligación, pero evita cambios no deseados.
    private:
        string Make;
        int    NumDoors;
        int    NumCylinders;
        int    EngineSize;
```

Implementaciones de los métodos de Automobile

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

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

Notar: no repetimos calificador de acceso.

Implementación del método 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;
}
```

cin es un objeto predefinido, corresponde a la salida estándar.

Constructores 1/5: Sobrecarga

- Como en Java, múltiples constructores pueden existir con diferentes listas de parámetros:

```
class Automobile {
public:
    Automobile();
    Automobile( string make, int doors, int cylinders,
                int engineSize=2);   // esta notación
                 // señala que este argumento es opcional,
                 // equivale a tener 2 constructores en Java.
    Automobile( const Automobile & A ); // constructor copia
    :
    :
```

Significa: el valor de A no puede ser modificado en implementación.

Constructores (2/5): Invocación

```
// muestra de llamadas a constructores:

    Automobile myCar;                  // Automobile()

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

    Automobile hisCar( yourCar ); // Constructor copia
```

Constructores (3/5): Implementación

```
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 (4/5): visibilidad de Parámetros

- 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;      // en Java era this.xxx

this->NumCylinders = NumCylinders;
```

Constructores (5/5): 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() (hoy obsoleto) de la clase Object.
- Java tiene un proceso de “recolección de basura” por lo que hace los destructores no críticos.
- En C++ el destructor se invoca en forma automática justo antes que el objeto sea inaccesible para el programa.
- El método destructor no tiene parámetros, se llama igual a la clase y lleva un signo ~ como prefijo.
- Ej: Automobile::~Automobile() {}