Tecnología de la Programación

Tema 3. Clases y objetos en C++
Tercera parte

Clase en C++

- Una definición de clase comprende:
  - Cabecera
  - Campos o atributos:
    - Variables
    - Constantes
  - Métodos:
    - Funciones
    - Constructores
    - Destructores (finalizador)
- En C++ lo normal es separar entre archivos de cabecera .h y archivos de implementación .cpp
  - En los archivos de cabecera se escribe la definición de clase:
    - Atributos, normalmente privados
    - Cabecera de los métodos: funciones, constructores y destructores, normalmente públicos

Atributos

- Existen normas de codificación (especialmente en C++, pocas veces utilizado en Java) que sugieren comenzar los nombres de los atributos con el carácter de subrayado o guión bajo (_).
- De esta forma a la vista del código se sabe que si se hace referencia a un símbolo que comienza con él (por ejemplo _numCuenta) se está haciendo referencia a un atributo y no a una variable local. De esta forma, el uso del this para evitar posibles ambigüedades no es necesario.

Atributos

- En C++ la declaración es similar, los atributos deben ser privados, aunque *no* puede dárseles valor inicial en la propia declaración (hay que hacerlo en los constructores).
- En un fichero .h se escribiría:

```
class EjemploDeClase {
....
private :
TipoDelAtributo atributoDelObjeto ;
...
}
```

this

- El uso de *this* es ligeramente distinto en C++ y en Java. En C++, siempre aparece como this-> y en java como this

Constructoras

- En C++ también puede haber new. Este operador lleva a crear objetos independientes en memoria que sean accesibles fuera del ámbito de la variable asociada, un tema que se verá más adelante.
- También hay invocaciones sin este operador. En tales casos, basta con indicar el nombre de la clase, el nombre de la variable y colocar unos paréntesis a la derecha del nombre de la variable, agregando los parámetros del constructor que se quiera utilizar.
  - Si se va a llamar al constructor sin parámetros los paréntesis se pueden omitir.

Constructoras

```
class EjemploDeClase {
public :
   EjemploDeClase () {  this -> atributoEntero = 2;   }
   EjemploDeClase (int nuevoValor ) {  this -> atributoEntero = nuevoValor ;}
private :
   int atributoEntero ;
}
...
void main () {
   EjemploDeClase unaVariable ();
   EjemploDeClase unaVariableIgual ;
   EjemploDeClase otraVariable (4);
   EjemploDeClase * otraVariableMas = new EjemploDeClase ();
}
```


Destructoras

- En C++ :
  - No existe recolección de basura, por lo que la liberación de la memoria es tarea del programador.
  - Hay que implementar los destructores.
- En C++ el destructor se llama igual que la clase pero antecediéndolo con el carácter ~.

Destructoras

```
class EjemploDeClase {
public :
   EjemploDeClase (){
        this -> vectorEnteros = new int[ TAM_POR_DEFECTO ];
   }
   EjemploDeClase (int tam ){
              this -> vectorEnteros = new int[tam ];
   }
   ~ EjemploDeClase (){
              delete [] vectorEnteros ;  // Liberamos la memoria solicitada
   }
private :
   int * vectorEnteros ; // El array dinámico lo implementamos con punteros
   static const int TAM_POR_DEFECTO;  // El valor se da en el cpp
}
```


Clases y objetos

- Como con otros atributos, los atributos que contendrán objetos deben inicializarse igualmente, llamando a sus constructores:
  - En C++ si se utiliza un tipo valor el compilador llama automáticamente al constructor sin parámetros (si éste no existe, el programador deberá llamar a uno explícitamente en la lista de inicializadores del constructor).
  - Si el atributo es un puntero, deberá pedirse memoria explícitamente (igual que se vió en el ejemplo del vector de enteros anteriormente); de no hacerse el puntero nos llevaría a una dirección aleatoria.

Métodos

- En C++ la declaración de los métodos es similar a Java, con la particularidad de que pueden declararse métodos const para indicar que en la implementación *no se cambia* el this.
- Por su parte, la definición/implementación puede hacerse:
  - En un fichero distinto de implementación (normalmente extensión .cpp).
  - En la propia declaración de la clase, de forma similar a como se hace en Java. El compilador considerará el método *inline*.
    - Tanto en este caso como en el anterior el nombre del método habrá que antecederlo por el nombre de la clase seguida de ::.
  - En el propio fichero de cabecera fuera de la declaración de la clase; para eso habrá que indicar explícitamente en la declaración que el método es inline.

Métodos

En el ejemplo siguiente utilizamos las dos últimas propuestas:

```
class EjemploDeClase {
  public :
    TipoADevolver nombreDelMetodo1(TipoParametro1 nombreParametro1, TipoParametro1 nombreParametro2) {
      TipoADevolver resultado ;
      ...
      return resultado
    }
    inline void nombreDelMetodo2 ( TipoParametro1 nombreParametro1, TipoParametro1 nombreParametro2 );
};
inline void EjemploDeClase::nombreDelMetodo2 (TipoParametro1 nombreParametro1,
                                               TipoParametro1 nombreParametro2 ) {
 // Implementación que no devuelve nada
}
```

Métodos

- El operador de invocación (o envío del mensaje de invocación) es el . seguido del nombre del método que se quiere ejecutar.
- En C++ si la variable que manejamos es un *puntero a un objeto* en vez de . se utiliza ->.

Métodos

En C++ existen dos formas de llamar al objeto, cuando manejamos objetos y cuando tenemos punteros a objetos:

```
int main () {
    TipoDeParametro1 nombreParametro1 ;
    TipoDeParametro2 nombreParametro2 ;

    EjemploDeClase ejemplo ();
    TipoADevolver resultado = ejemplo.nombreDelMetodo1 ( nombreParametro1 , nombreParametro2 );
    ejemplo.nombreDelMetodo2 ( nombreParametro1 , nombreParametro2 );

    EjemploDeClase * ejemplo2 =new EjemploDeClase ();
    TipoADevolver resultado = ejemplo2 -> nombreDelMetodo1 (nombreParametro1 , nombreParametro2 );
     ejemplo2 -> nombreDelMetodo2 ( nombreParametro1 , nombreParametro2 );
}
```

Atributos de clase

- En C++ también se pueden especificar atributos constantes, utilizando el modificador const que, al igual que en Java, suele implicar convertir el atributo en un atributo de clase con static.
- No obstante, en C++ para evitar consumir memoria las constantes de tipo entero suelen convertirse en enumerados.
- Otra alternativa es utilizar la opción de los #define heredada de C (aunque en ese caso la constante no pertenece a la clase, sino que es un “símbolo” global)

Atributos de clase

```
// Constantes en C
# define NOMBRE_DE_LA_CONSTANTE VALOR_DE_LA_CONSTANTE
class EjemploDeClase {
  public :
  // Atributo de clase constante; el valor se da en el .cpp
  static const TipoDeLaConstante NOMBRE_DE_LA_CONSTANTE2 ;
  // Definición de la constante entera sin consumir memoria
   enum { NOMBRE_DE_LA_CONSTANTE3 = VALOR };
}
El acceso a estas constantes variaría, según fuera su declaración:
int main () {
   cout << NOMBRE_DE_LA_CONSTANTE ;
   cout << EjemploDeClase :: NOMBRE_DE_LA_CONSTANTE2 ;
   cout << EjemploDeClase :: NOMBRE_DE_LA_CONSTANTE3 ;
}
```

Visibilidad de atributos y métodos

- En Java se indica la visiblidad de cada uno de los elementos antecediendo a su declaración las palabras reservadas public y private
- En C++ la declaración de la clase se divide en distintas secciones, marcando el comienzo de cada una con public: y private:

Visibilidad de atributos y métodos

```
class Fecha {
  public :
    void escribe () {
          std :: cout << "Día: " + this ->dia;
          ...
    }
    int resta (Fecha fecha ) {
          // se puede acceder a this->dia y a fecha.dia
    }
private :
  int dia ;
  int mes ;
  int anyo ;
};
```

Visibilidad de atributos y métodos

- Dado que en C++ no existen reglas de visibilidad entre namespaces, se puede conseguir algo parecido utilizando las clases “amigas” (friend), que permiten al programador de una clase indicar al compilador el nombre de otras clases que podrán acceder (desde “fuera”) a elementos declarados como no públicos.

Tipos en C++

- En C++ todos los tipos son tipos-valor. Todas las variables almacenan directamente los valores a los que representan. Entre las implicaciones de esto tenemos que:
  - Si declaramos una variable local de una clase, el espacio reservado en la pila será el tamaño completo de ese objeto.
  - Si una clase tiene un atributo que pertenece a otra clase, el tamaño de sus instancias se incrementa en base a los atributos de esa segunda clase.
  - Si declaramos una variable local de una clase, la instancia a la que representa esa variable será destruida cuando ésta salga de ámbito (al final de la función o de la sección de código). Eso provocará la llamada a su destructor.
  - Si una clase A tiene un atributo de otra clase B, cuando se construye el objeto de la clase A se crea automáticamente el objeto de la clase B (y llama a su constructor). Cuando el objeto de la clase A se destruye, se destruye el objeto de la clase B (y llama al destructor).
  - El operador de comparación, ==, entre dos instancias de una clase debe devolver true si los dos objetos son *iguales* (no si son *el mismo*).
  - El operador de asignación, =, hace una *copia* de un objeto a otro.

Tipos en C++

```
void func1 () {
   Fecha f; // Provoca la llamada al constructor de Fecha
            // En la pila se reservan 3 * sizeof(int) bytes
   ....
} // Al final se llama al destructor

void func2 () {
   Fecha f1 (12 , 10, 1492);
   Fecha f2 (1, 1, 2000);
   f1 == f2; // Esto evalúa a false, representan días distintos
   f2 = f1; // Copiamos el valor de f1 en f2, pero son instancias distintas
   f1 == f2; // Esto evalúa a true
   f1. avanzaUnDia (); // Cambiamos f1
   f1 == f2; // Ahora evalúa a false, f2 no ha cambiado
}
```


Tipos en C++

- Es el programador el que, si quiere manejar tipos referencia, debe hacerlo explícitamente manejando o bien punteros o bien referencias. En ese caso:
  - Igual que ocurre en Java tendrá que crear explícitamente las instancias con new.
  - Tendrá que encargarse de liberarlas con delete.
  - Si utiliza punteros, para invocar a los métodos tendrá que utilizar el operador ->.
  - El == con punteros tendrá el mismo significado que en Java (true si es el mismo objeto).
  - El equivalente al equals se consigue haciendo el == sobre *los contenidos* de los punteros (operador *)

Tipos en C++

```
void func1 () {
   Fecha *f;  // Variable sin inicializar
   f = NULL ; // puntero que no hace referencia a ninguna instancia
   f = new Fecha (12 , 10, 1492); // Creamos la instancia (se invoca al constructor)
   ...
   delete f; // Eliminamos la instancia (se invoca al destructor)
}
```