Metodología Para el Diseño Orientado a Objetos y Programación Orientada a Objetos

Ing. Rubén Peredo Valderrama
Profesor e Investigador del CINTEC-IPN.
Francisco F. Cordova Quiroz
Alumno de la Maestría del CINTEC-IPN.
Ing. Alberto Flores Rueda
Profesor e Investigador del CINTEC-IPN.

Este artículo describe una metodología para el diseño y programación orientada a objetos, la cuál es muy apropiada cuando se desarrollan proyectos de gran tamaño y aquellos que involucran equipos de programadores, además de proporcionar un lenguaje gráfico al código escrito, para su mejor comprensión. El siguiente trabajo trata el diseño orientado a objetos, clase, organización, codificación, y así sucesivamente. Siguiendo las sugerencias del presente artículo se asegura que el código C++ será consistente y profesional en su organización y apariencia.

Diseño de Representaciones Orientadas a Objetos

Cualquier método que se seleccione para diseñar definiciones de clase debe cubrir los siguientes aspectos:

o Comunicar el diseño externo de la clase (interface)

o Comunicar el diseño interno de la clase (datos en particular)

o Establecer la relación de herencia entre las clases (la clase A es hija de B)

o Establecer la relación entre clases (la clase A es amiga de B)

Los cuatro puntos arriba mencionados se cubrirán a continuación, y son ampliamente recomendados para el diseño en C++, aunque toda esta metodología es aplicable a otros lenguajes de programación como Pascal, Ensamblador, entre otros.

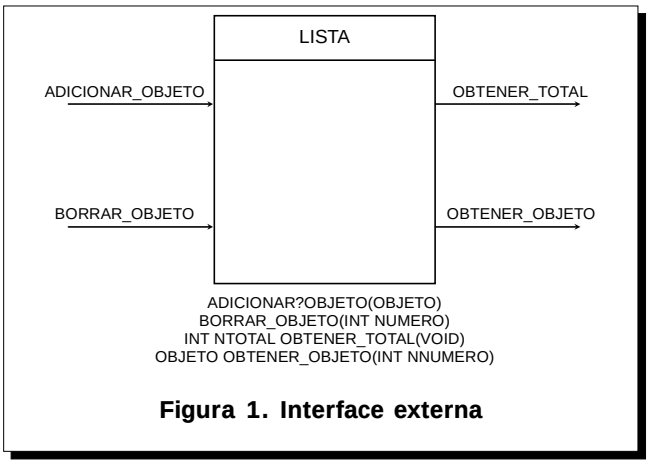

Figura 1. Interface externa

Diseño Externo

El primer paso en el diseño de una clase es determinar la interface hacia el exterior, y funciones miembros para su diseño. Como se muestra en la **figura 1**, la clase misma es representada como un rectángulo, en lo alto del rectángulo se pone el nombre de la clase. Si es necesario especificar él nombre de un objeto, este se coloca directamente bajo el nombre de la clase; esto es muy útil para el manejo de objetos en las aplicaciones, debido a que es posible determinar de manera rápida que tipo de objeto es, además de aumentar la legibilidad del código.

En el lado izquierdo se colocan flechas que entran al rectángulo y sus correspondientes etiquetas. Las etiquetas son los nombres de funciones miembros que ponen datos hacia el objeto o efectúan una acción sobre el objeto, e incluso en algunas ocasiones ambas tareas.

Debajo del rectángulo se ponen las definiciones de cada función miembro. Estas definiciones siguen los prototipos estandard con sus respectivos parámetros.

Diseño Interno

El diseño interno incluye datos incluidos por composición y funciones internas que no son accesibles por funciones externas. El diseño interno de la clase utiliza un formato similar al de la **figura 2**, la cuál muestra datos internos dentro del

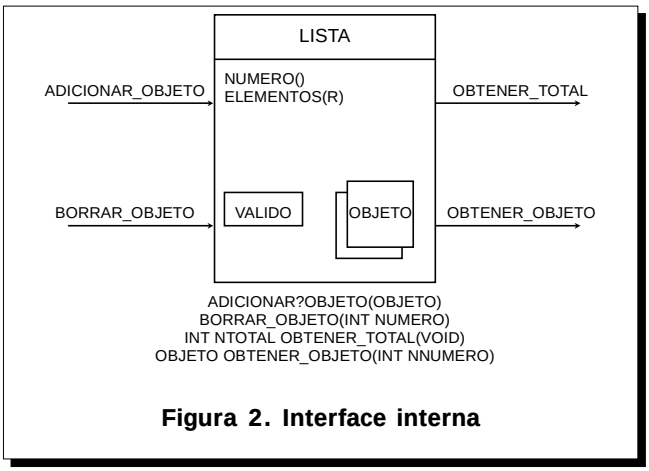

Figura 2. Interface interna

rectángulo. Los datos son listados por tipo y nombre en la parte alta del rectángulo. Es importante mostrar en seguida del dato un paréntesis, el cuál puede contener una L o una E, o en algunos casos nada. La L significa lectura y la E escritura. Dado que L y E aplican únicamente a datos, esta notación es encontrada únicamente dentro del rectángulo. Específicamente:

(L/E) las variables pueden ser leídas y escritas fuera de la clase. Las operaciones de lectura y escritura pueden ser efectuadas por funciones miembros (una que lea y otra que realice la escritura) para mantener la encapsulación interna de la clase. Sin embargo, es importante señalar que las funciones miembro que manipulan las variables no necesitan ser colocadas fuera del rectángulo con las otras funciones miembros. Las funciones miembro de lectura y escritura son implicadas por las letras L / E a continuación del nombre de la variable.

(L) las variables pueden ser leídas pero no escritas. Estas variables deben de ser inicializadas y mantenidas por la clase misma.

(E) las variables pueden ser escritas pero no leídas. Estas pueden ser banderas, o objetos tales como passwords que pueden ser establecidos pero no leídos.

Las variables sin L o E no pueden ser leídas o escritas fuera de la clase. Estas son variables son utilizadas internamente por la clase misma.

Otros objetos dentro de la clase son incluidos dentro del rectángulo debajo de los datos. Para cada uno de estos objetos, el nombre de la clase es mostrado en lo alto y un nombre descriptivo asignado a su instancia especifica de la clase (objeto), dentro del rectángulo. El nombre del objeto es frecuentemente escrito en itálicas. Si existen múltiples instancias de un objeto, tal como un arreglo de objetos, se deberá mostrar esto dibujando pilas de objetos como se muestran en la **figura 2**.

Las funciones internas que no son accesibles fuera de la clase deben mostrarse dentro del cuerpo del rectángulo del objeto. Los nombres de las funciones internas son incluidas en el cuerpo del rectángulo. Los parámetros específicos usados por funciones internas no son mostrados por lo regular en este nivel.

Relaciones de Herencia

Las relaciones de herencia entre los objetos, o alguna clase de jerarquía, son mostradas utilizando una organización jerárquica similar a la mostrada en la **figura 3**. Por ejemplo en dicha figura se muestra la relación entre la clase base (lista) y las clases derivadas (las demás) de esta.

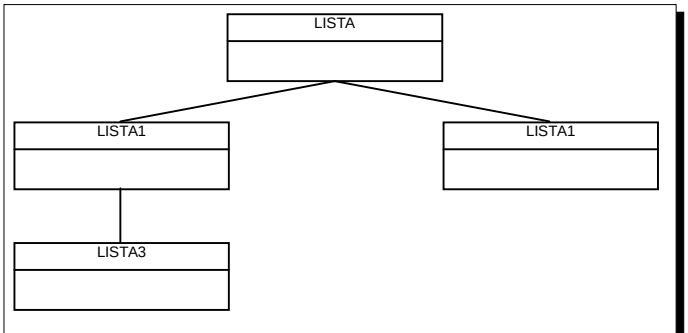

Figura 3. Relación de herencia entre clases.

Relaciones de Composición

Las relaciones de composición, forman parte de una jerarquía que se representa de manera similar a la **figura 4** (en forma de árbol). En este ejemplo, el titulo de la clase es LIBRO y esta compuesta por tres capítulos y dos secciones (dentro del capitulo 3).

Para mostrar más detalladamente la clase y sus funciones miembro, se realiza una subdivisión de clase y funciones miembro, es importante notar que no existen estandares para la representación de diseños en programación orientada a objetos, pero esta notación trabaja bastante bien para fines prácticos.

Organización de Filas Para el Desarrollo en C++

La **figura 5** muestra un ejemplo de organización para programas simples en C++ para Windows. Se contará con un conjunto de filas (encabezado y fuente) para cada clase que se

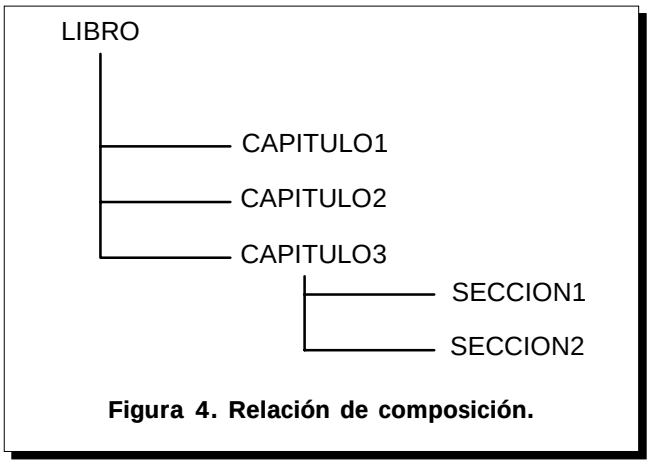

Figura 4. Relación de composición.

define. Es práctico y muy común entre los programadores poner en la fila de encabezado la definición de la clase, y definición de tipos que pertenecen a la clase; así como cualquier función relacionada para funciones no miembros. Para la clase, las definiciónes de función sirven como prototipos de las mismas. Para trabajar en Windows, es necesario incluir funciones miembro para exportarlas.

Es importante notar que la mayoría de los programadores no incluyen código, y aún código inline. La mayor parte de la literatura especializada en C++ recomienda de manera importante no poner código en las filas de encabezado para facilitar el mantenimiento y actualización del programa. El código es colocado en las filas fuente *.CPP. El único punto en contra de este manejo de definiciones y código, es que al definir la clase y colocar el código a continuación se obtiene una mayor eficiencia del mismo, dado que el código realmente se encuentra en inline, pero implica una mayor cantidad de trabajo al actualizar el programa. Sin embargo, es altamente recomendable el caso anterior cuando la eficiencia del código es el factor de mayor consideración a tomar en cuenta. Otra alternativa es colocar el código en la fila .CPP y utilizar la función inline para llegar a un resultado más o menos equivalente.

El código fuente es puesto en las filas *.CPP; por esto debe haber una correlación uno a uno entre las filas de código y las filas de encabezado. Dentro de la fila de código fuente, se definirá cada una de las funciones puestas en la fila de encabezado perteneciente a la clase.

A diferencia de los programas para DOS, los programas para Windows cuentan en la mayoría de los casos con filas de recursos .RC, las cuáles también cuentan con una fila de encabezado *.RH; en las filas RC se cuenta con los recursos propiamente dichos (esto es, equivalente al código en una fila *.CPP), y en las filas RH se encuentran con los identificadores de los recursos (similar a las filas de encabezado *.H).

Estructura de las Filas de Encabezado

Una fila de encabezado consiste principalmente de los siguientes elementos:

Un comentario: esta parte provee un nombre a la fila y una pequeña descripción de la misma.

Directivas para compilación: Esto previene al compilador de compilar dos veces la misma fila de encabezado, si es incluido más de una vez, dado que las filas de encabezado son incluidas múltiples veces en aplicaciones reales.

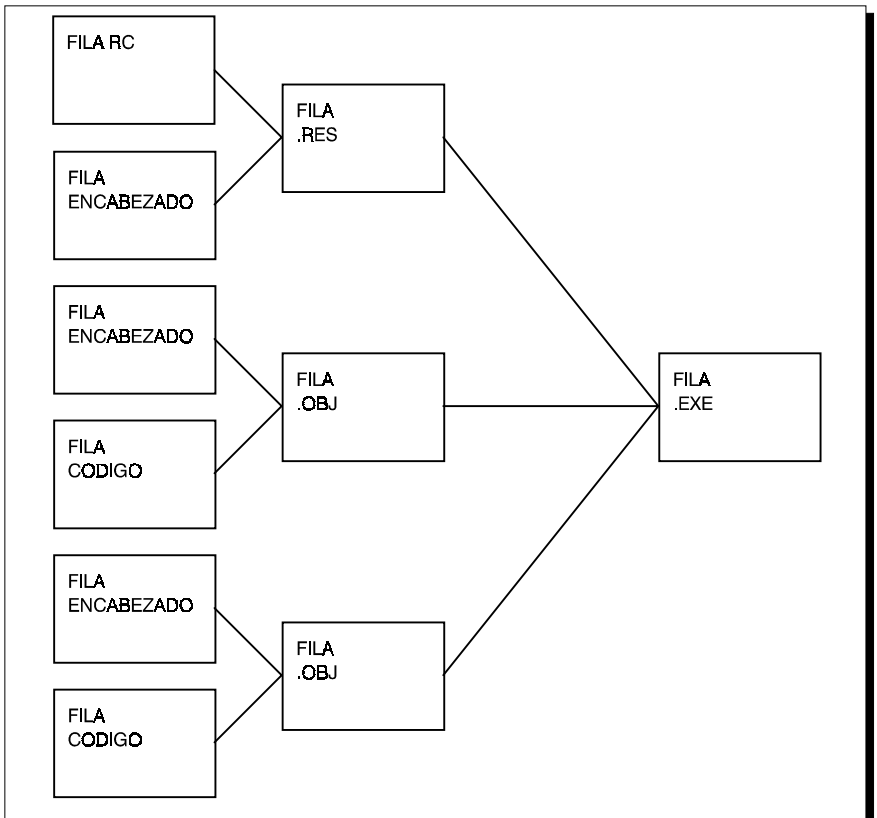

Figura 5. Organización de un proyecto en C++.

Definición de prototipos de funciones: esta parte incluye el establecimiento de funciones, para su posterior definición en las filas .CPP.

Directivas al compilador: Esta parte consiste de llamadas al compilador con palabras reservadas como **typedef** y inclusión de bibliotecas **include<>**.

Declaración y descripción de una clase: Esta parte incluye la definición de la clase y definición de los niveles de acceso de las funciones (**publicas, protegidas y privadas**), y declaración de datos miembros, además de establecer las relaciones entre otras clases (**friend**).

La directiva #endif: En caso de incluirla esta directiva al compilador establece una condicional, a la manera de **if** en C, pero al compilador.

Esto puede observarse en las siguientes secciones de código.

```
Parte 2
_____________________________________________________________________

#if !defined(__redesapp_h)
#define __redesapp_h
_____________________________________________________________________
Fin Parte 2

Parte 1
_____________________________________________________________________

/*  Proyecto Redes Neuronales
   INSTITUTO POLITECNICO NACIONAL


   SUBSISTEMA:   redes.exe Aplicación
   FILA:        redesapp.h
   AUTOR:       CINTEC-IPN

   DESCRIPCION
   ===========
   Definición de la clase para redesApp (TApplication).
*/
_____________________________________________________________________
Fin Parte 1

Parte 4
_____________________________________________________________________
#include <owl\owlpch.h>
#pragma hdrstop
#include <classlib\bags.h>
#include «rdsmdicl.h»
#include «redesapp.rh»   // Identificadores de los recursos.
_____________________________________________________________________
Fin Parte 4

Parte 5
_____________________________________________________________________

/* TFileDrop class Mantiene la información de la fila, su nombre,  donde fué
arrastrada, y si se encuentra en el área del cliente */

class TFileDrop {
public:
  operator == (const TFileDrop& other) const {return this == &other;}

   char*   FileName;
   TPoint  Point;
   bool    InClientArea;

Parte 3
_____________________________________________________________________

   TFileDrop (char*, TPoint&, bool, TModule*);
   ~TFileDrop ();
_____________________________________________________________________
Fin Parte 3

private:

Parte 3
_____________________________________________________________________

   TFileDrop (const TFileDrop&);
   TFileDrop & operator = (const TFileDrop&);
_____________________________________________________________________
Fin Parte 3
};
_____________________________________________________________________
Fin de la parte 5

Parte 4
_____________________________________________________________________
```

```
typedef TIBagAsVector<TFileDrop> TFileList;
typedef TIBagAsVectorIterator<TFileDrop> TFileListIter;
_____________________________________________________________________
Fin Parte 4

Parte 5
_____________________________________________________________________
class redesApp : public TApplication {
private:

   bool          HelpState;
   bool         ContextHelp;
   HCURSOR        HelpCursor;

Parte 3
_____________________________________________________________________

   void SetupSpeedBar (TDecoratedMDIFrame *frame);
    void AddFiles (TFileList* files);
_____________________________________________________________________
Fin Parte 3

public:

Parte 3
_____________________________________________________________________

   redesApp ();
   virtual ~redesApp ();
   void CreateGadgets (TControlBar *cb, bool server = false);
   redesMDIClient  *mdiClient;
_____________________________________________________________________
Fin Parte 3
   TPrinter       *Printer;
   int            Printing;
public:

Parte 3
_____________________________________________________________________

   virtual void InitMainWindow ();
   virtual void InitInstance ();
   virtual bool CanClose ();
   virtual bool ProcessAppMsg (MSG& msg);
_____________________________________________________________________
Fin Parte 3

protected:

Parte 3
_____________________________________________________________________

   void EvNewView (TView& view);
   void EvCloseView (TView& view);
   void CmHelpAbout ();
   void CmHelpContents ();
   void CmHelpUsing ();
   void EvDropFiles (TDropInfo drop);
   void EvWinIniChange (char far* section);
   void CmBAM1 ();
_____________________________________________________________________
Fin Parte 3

DECLARE_RESPONSE_TABLE(redesApp);
};
_____________________________________________________________________
Fin Parte 5

Parte 6
_____________________________________________________________________

#endif
_____________________________________________________________________
Fin Parte 6
```

Estructura de las Filas de Código Fuente

Una fila fuente consiste de los siguientes elementos:

Comentarios: sirven para indicar el nombre de la fila y una breve descripción de la misma.

```
Parte 1
_____________________________________________________________________

/*  Proyecto  Redes Neuronales
   INSTITUTO POLITECNICO NACIONAL


   SUBSISTEMA:    redes.exe Application
   FILA:         redesapp.cpp
   AUTOR:       CINTEC-IPN



   DESCRIPCION
   =============
   Fila fuente para la implementación de redesApp (TApplication).
*/
_____________________________________________________________________
Fin Parte 1

Parte 2
_____________________________________________________________________

#include <owl\owlpch.h>
#pragma hdrstop
#include <dir.h>
#include «redesapp.h»
#include «rdsmdicl.h»
#include «rdsmdich.h»
#include «rdsedtvw.h»
#include «rdsabtdl.h»                    // Definición acerca del dialogo.
_____________________________________________________________________
Fin Parte 2

Parte 3
_____________________________________________________________________

const char HelpFileName[] = «redes.hlp»;

TFileDrop::TFileDrop (char* fileName, TPoint& p, bool inClient, TModule*)
{
Código Fuente
}

TFileDrop::~TFileDrop ()
{
Código Fuente
}

const char *TFileDrop::WhoAmI ()
{
Código Fuente
}

// Etc....etc..
_____________________________________________________________________
Fin Parte 3
```

Incluir bibliotecas: consiste en incluir las bibliotecas necesarias para el programa.

Descripción de las funciones miembro: Declarar las funciones miembros incluidas en la fila de encabezado.

Bibliografía

[1] García-Pelayo y Gross, Ramón; *"Diccionario Pequeño Larousse en Color"*, 1981.

[2] *"Standard Dictionary of the English Language"*; Funk & Wagnalls, 1967.

[3] Wienderhold, Gio; *"File Organization for Data Base Design"*; McGraw-Hill, 1967.

[4] Shaft, Adam; *"Historia y Verdad. Teoría y Praxis"*.

[5] William Roetzheim; *"Programming Windows with Borland C++4.5"*; ZD PRESS 1994.

[6] Namir Shammas, Craig Arnush & Edward Mulroy; *"Teach Yourself Borland C++ 4.5 in 21 Days"*; SAMS Publishing 1995.

[7] Paul Perry; *"Using Borland C++ 4"*; QUE 1994.