Una metodología para la enseñanza en la programación orientada a objetos

Rubén Peredo Valderrama*
Francisco F. Córdova Quiroz**
Óscar Camacho Nieto***

Resumen

El trabajo resume los fundamentos de OPP (Object Oriented Programming) *como abstracción, herencia, polimorfismo, clase, objetos, organización, codificación, y así sucesivamente. Se busca establecer una metodología apropiada para desarrollar proyectos de software a nivel educativo que redunde en una mejor productividad en el proceso de enseñanza-aprendizaje y permita que el código se vuelva consistente y profesional en su organización y apariencia, además de crear lineamientos generales para establecer un lenguaje común cuando se involucran equipos de programadores.*

Abstract

This document makes a summary of the fundamentals of OOP (Object Oriented Programming), such as abstraction, inheritance, polymorphism, class, objects, organization, codification and so on. It tries to stablish an appropriate methodology for the development of software projects on an educational level which would redound in better results in the teaching-learning process and allow the code to become consistent and professional once more in its organization and appearance, as well as creating general guidelines for stablishing a common language when teams of programmers are involved.

1. Fundamentos

La OPP *(Object Oriented Programming)* se fundamenta en tres puntos: la abstracción, la herencia y el polimorfismo.

La abstracción es el agrupamiento de datos y funciones, conformando un nuevo tipo, con características (datos miembros) y funcionamientos (funciones miembro) particulares. En los lenguajes denominados de alto nivel la encapsulación cuenta con niveles que permiten y restringen el acceso a ciertas funciones y datos, por ejemplo en C++ este acceso está determinado por las palabras: *public*, *private* y *protected*. En el caso de *public* permite que cualquier otra clase accese a estos procedimientos, en el de *private* permite un acceso restringido solamente a funciones miembros de la clase y clases que han sido declaradas como amigas de ésta, por último tenemos *protected* que, además de contar con las características de acceso de las *private,* permite el acceso a las clases derivadas de la misma.

La herencia permite crear nuevos tipos de tipos existentes, que es de gran importancia en la actualidad, dada la enorme complejidad del software. La trascendencia principal de la herencia reside en el hecho de poder modificar estos tipos permitiendo agregarles mayor funcionalidad o quitándoles ciertas características, para un nuevo funcionamiento de los mismos. Existen lenguajes orientados a objetos que permiten herencia múltiple (C++) y otros que sólo permiten herencia simple (Smaltalk). En la actualidad el 90 % de los proyectos importantes se construyen en C++ por la popularidad de este lenguaje y en realidad es un lenguaje de nivel intermedio, permitiendo tener el poder de los lenguajes de alto nivel (Pascal, Visual Basic entre otros) y combinar los lenguajes denominados de bajo nivel (Ensamblador, MASM, TASM).

*Departamento de Programación del CINTEC-IPN.
**Profesor de la E.S.I.M.E. Culhuacán y estudiante de la maestría en el CINTEC.
***Coordinador de la maestría en el CINTEC-IPN.

El polimorfismo es otra razón para la herencia, dado que es posible instanciar n cantidad de clases de una misma clase, surge un problema importante: ¿cómo deberá responder una función que se encuentra en diferentes instancias y realiza un procedimiento diferente en cada una de ellas? En pocas palabras, ¿cómo el compilador hará el llamado correcto a la función adecuada de la clase adecuada? El polimorfismo es la solución a este problema, permitiendo una respuesta adecuada para cada función de cada clase, aunque estas funciones cuenten con el mismo nombre.

Entender todos estos fundamentos es vital para el aprendizaje del alumno, además la posibilidad de poder visualizar de una manera más clara estas palabras y representarlas sin escribir una sola línea de código; permitirá un diagrama de flujo adecuado con OOP facilitando el entendimiento y funcionamiento de cada clase. En el caso de la programación orientada a objetos en ambiente Windows es importante notar que dadas las características del sistema operativo, éste se encuentra orientado a mensajes y cada procedimiento está asociado a eventos del mismo sistema operativo, como es el caso de botones, menús, etcétera. En este caso la OOP es básicamente la misma, pero el modo de enfocar el programa difiere por lo expuesto.

A continuación se presenta la propuesta para el diseño de representaciones orientadas a objetos, la cual convertirá nuestro código en un lenguaje gráfico, fácil de entender y de analizar. Otra ventaja adicional es que a la hora de depurar estos programas permite una reducción del tiempo de depuración al comprender y visualizar el programa en su totalidad.

2. Diseño de representaciones orientada a objetos

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

- Comunicar el diseño externo de la clase (interface).
- Comunicar el diseño interno de la clase (datos en particular).
- Establecer la relación de herencia entre las clases (la clase A es hija de la clase B).
- Establecer la relación entre clases (la clase A es amiga de la clase B).

Los cuatro puntos mencionados los trataremos a continuación y son ampliamente recomendados para el diseño en C++; claro que toda esta metodología es aplicable a otros lenguajes de programación como Pascal, Ensamblador, entre otros.

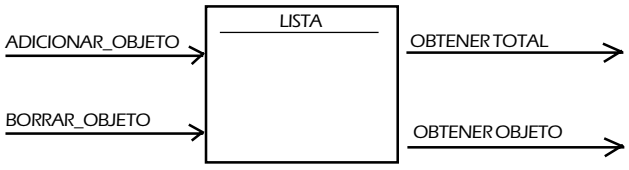
2.1 Diseño externo

El primer paso en el diseño de una clase es determinar la interface hacia el exterior y las funciones miembro para su diseño. Como se muestra en la figura 1, la clase misma es representada como un rectángulo y en lo alto del rectángulo se anota el nombre de la clase.

Si es necesario especificar el nombre de un objeto 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 qué 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.

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



ADICIONAR_OBJETO(OBJETO).
BORRAR_OBJETO(INT NÚMERO).
INT NTOTAL OBTENER_TOTAL(VOID).
OBJETO OBTENER_OBJETO(INT NNÚMERO).

Figura 1. Interface externa

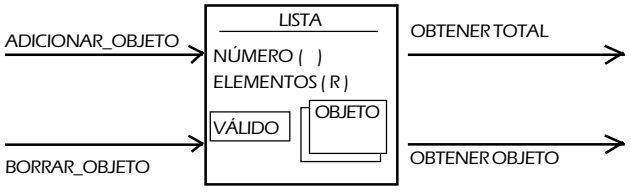
2.2 Diseño interno

El diseño interno comprende 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, que muestra datos internos dentro del rectángulo. Los datos son listados por tipo y nombre en la parte alta del rectángulo. Es importante mostrar enseguida del dato un paréntesis que puede contener una L, una *E* y en algunos casos nada; la *L* significa lectura y la *E* escritura, dado que *L* y *E* se aplican únicamente a datos, esta notación se encontrará 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 incluidas fuera del rectángulo con las otras funciones miembro. 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 ser inicializadas y mantenidas por la clase misma.

/E/ las variables pueden ser escritas pero no leídas. Éstas pueden ser banderas u objetos tales como passwords, que pueden ser establecidos pero no leídos.



ADICIONAR_OBJETO(OBJETO).
BORRAR_OBJETO(INT NÚMERO).
INT NTOTAL OBTENER_TOTAL(VOID).
OBJETO OBTENER_OBJETO(INT NNÚMERO).

Figura 2. Interface interna

Las variables sin ninguna *L* o *E* no pueden ser leídas o escritas fuera de la clase. Son variables 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 la instancia específica de la clase (objeto) mostrado 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, deberá mostrarlas dibujando pilas de objetos, como se muestran en la figura 2.

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 por lo regular no son mostrados en ese nivel.

2.3 Relaciones de herencia

Las relaciones de herencia entre los objetos o alguna clase de jerarquía, son mostradas utilizando una organización jerárquica similar a la representada en la figura 3. En el ejemplo se muestra la relación entre la clase (lista) y las clases derivadas (las demás) de ésta.

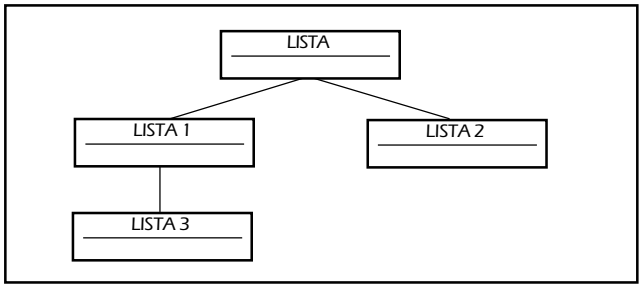


Figura 3. Relación de herencia entre clases

2.4 Relaciones de composición

Las relaciones de composición forman parte de una jerarquía que son representadas de manera similar a la figura 4 (en forma de árbol). En este ejemplo el título de

la clase es LIBRO y está compuesto por tres capítulos y dos secciones en el tercer capítulo.

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 estándares para la representación de diseños en la programación orientada a objetos, pero esta notación trabaja bastante bien para fines prácticos.

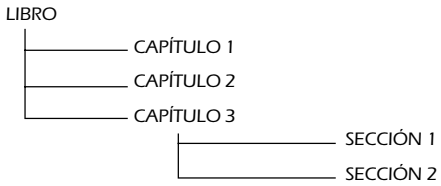


Figura 4. Relación de composición

3. 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 (encabezados y fuente) para cada clase que se define. Es práctico y muy común entre los programadores poner en fila de encabezado la definición de la clase, la definición de tipos que pertenecen a la clase y cualquier función relacionada para funciones no miembros.

Para la clase la definición de función sirve como prototipo de la función. Para trabajar en Windows es necesario incluir funciones miembro para exportarlas.

Es importante notar que la mayoría de los programadores no incluyen código y aun 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 contra 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. 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.

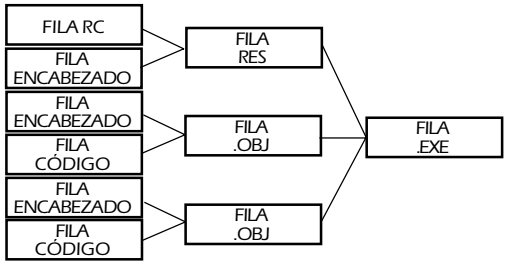


Figura 5. Organización de un proyecto en C++

El código fuente se pone en las filas .CPP, por esto debe haber una correlación uno a uno entre las filas de código y las de encabezado. Dentro de la fila de código fuente se definirán cada una de las funciones colocadas 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 cuales también tienen una fila de encabezado .RH; en las filas .RC se cuenta con los recursos propiamente dichos (esto equivale al código en una fila .CPP), y en las filas .RH se encuentran los identificadores de los recursos (similar a las filas de encabezado .H).

3.1 Estructura de las filas de encabezado

Una fila de encabezado consiste principalmente en los siguientes elementos:

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

·Directivas para compilación. 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.

·Definición de prototipos de funciones. Incluye el establecimiento de funciones para su posterior definición en las filas .CPP.

·Directivas al compilador. Este elemento consiste en llamadas al compilador con palabras reservadas como *typedef* e inclusión de bibliotecas *include* <>.

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

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

Examine:

Parte 2

```
#If !defined *(_redesapp_h)
#define_redesapp_h
```

Fin Parte 2

Parte 1

```
/* Proyecto Redes Neuronales
  Instituto Politécnico Nacional
  Copyright © 1996. Todos los derechos reservados.

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

    DESCRIPCIÓN
  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 y donde fue 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

Parte 3

```
private:

   TFileDrop (const TFileDrop&);
   TFileDrop & operator = (const TFileDrop&);
```

Fin Parte 3

Parte 5

```
} ;
```

Fin Parte 5

Parte 4

```
Typedef TIBagAsVector<TFileDrop> TFileList;
Typedef TIBagAsVectorIteractor<TFileDrop> TFileListIter;
```

Fin Parte 4

Parte 5

```
class redessApp : 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

3.2 Estructura de las filas de código fuente

Una fila fuente consta de los siguientes elementos:

·Comentarios. Este elemento sirve para indicar el nombre de la fila y una breve descripción de la misma.
·Incluir bibliotecas. Esta parte consiste en incluir bibliotecas necesarias para el programa.
·Descripción de las funciones miembro. Se trata de declarar las funciones miembro declaradas en la fila de encabezado.

Examine:

Parte 1

```
/* Proyecto Redes Neuronales
   Instituto Politécnico Nacional
   Copyright © 1996. Todos los derechos reservados.
```

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

    DESCRIPCIÓN
    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 diálogo.
```

Fin Parte 2

Parte 3

```
const char HelpFileName [ ] = "redes.hlp";
TFileDrop::TFileDrop (char* fileName, TPoint& p. Bool
inClient, TModule*)
{
Código Fuente
}
TFileDrop:: ~TFileDrop 0
{
Código Fuente
}
const char *TFileDrop::WhoAml 0
{
Código Fuente
}
// Etc...etc.
```

Fin Parte 3

4. Manteniendo claro el código

Todos los programadores comienzan escribiendo código simple, pero en la medida en que incrementan sus conocimientos y su experiencia, la complejidad del código que van generando también se incrementa; esto lo logran a través de poderosas características del lenguaje, lo cual se convierte en un código por lo general más eficiente, además de mantener un entendimiento del código más claro tanto para su lectura como para su comprensión.

Referencias

García-Pelayo y Gross, Ramón. *Diccionario pequeño Larousse en color,* 1981.

Funk & Wagnalls. *Standard Dictionary of the English Language*, 1967.

Wienderhold, Gio (1967). *File Organization for Data Base Design.* Mc Graw-Hill, 1967.

Shaft, Adam. *Historia y verdad. Teoría y praxis.*

Roetzheim, William (1994). *Programming Windows with Borland C++ 4.5.* Zd Press, 1994.

Namir, Shammas; Arnush, Craig & Mulroy, Edward (1995). *Teach Yourself Borland C++ 4.5 in 21 Days.* SAMS Publishing, 1995.

Perry, Paul (1994). *Using Borland C++ 4*. QUE, 1994.