Objetos: lo real, lo abstracto y lo lógico. Breve introducción

Objects: The real, the abstract and the logical things. Brief introduction

[*]**Héctor Julio Fúquene Ardila**

[**]**Gerardo Alberto Castang Montiel**

Fecha de recepción: 2 de octubre de 2008
Fecha de aceptación: 9 de noviembre de 2008

Resumen

Se hace una breve introducción a lo que es la programación orientada a objetos, partiendo de un análisis del entorno y de la experiencia cotidiana, para llegar a abordar un problema y plantear una solución desde la perspectiva de la programación orientada a objetos utilizando como lenguaje de programación el C++.

Palabras clave: real, abstracto, objeto, relación, entorno.

Abstract

Presently article is made a brief introduction to what is the programming guided to objects; leaving of an analysis of the environment and of the daily experience, to end up approaching a problem and to outline a solution from the perspective of the programming guided to objects using as programming language the C++.

Key words: Real, abstract, object, relation, environment.

* Ingeniero de Sistemas. Magíster en Teleinformática de la Universidad Distrital Francisco José de Caldas. Miembro del grupo de investigación en Telemática. Director revista Vínculos. Profesor adscrito a la Facultad Tecnológica. hfuquene@udistrital.edu.co

* Ingeniero Electrónico. Magíster en Teleinformática de la Universidad Distrital Francisco José de Caldas. Miembro del grupo de investigación en Telemática Orión. Profesor adscrito a la Facultad Tecnológica. gacastangm@udistrital.edu.co

Introducción

El ser humano, desde que nace, comienza a tener una relación directa con su entorno; a través de sus sentidos comienza a identificar y a familiarizarse con el mundo que lo rodea. El interés por conocer y experimentar es innato al ser humano desde su llegada.

El tema que aquí se aborda tiene la ventaja de que la persona que inicia su estudio ya cuenta con una serie de conocimientos adquiridos en su diario vivir que le van a permitir asimilar los conceptos con gran facilidad. El estudio de los objetos, como base para la programación orientada a objetos, se considera vital a la hora de implementar programas bajo este paradigma. Así como los objetos son fundamentales en nuestras actividades cotidianas, también debemos contemplar la lógica como fundamental para ser más eficientes en lo que a diario hacemos; podemos decir que los objetos son universales, pero que la lógica también lo es; si no se tiene esta percepción, debemos trabajar de tal forma que la incorporemos a todas nuestras acciones y pensamientos, pues la lógica es la esencia de la programación, del ingenio y de la creación de software.

Desde la perspectiva del mundo académico, ya en los primeros años de educación de los niños se les está dando nociones formales de lo que son los objetos; más adelante, con el estudio de los conjuntos se formalizan estas nociones un poco más, se es capaz de identificarlos, de clasificarlos, de realizar operaciones entre ellos, de asumir similitudes y diferencias, entre otros.

¿Qué son los objetos para un ser humano cualquiera?

Podemos decir que un objeto es todo aquello que podemos percibir con los órganos de los sentidos, es decir que podemos ver, oler, saborear, palpar e inclusive oír. Esto implica que todo cuanto nos rodea puede ser considerado un 'objeto'.

Pero el ser humano tiene la posibilidad de concebir cosas mediante sus pensamientos, valiéndose de estos para representar situaciones que escapan del ámbito 'real', pero que de igual forma a cualquier ser humano le es imposible esquivar o sustraerse. Esto quiere decir que el término objeto involucra mucho más que lo 'real', incluyendo todo lo abstracto y conceptual, lo que implica una clasificación inicial de los objetos: hay objetos reales y hay objetos abstractos, un ejemplo de estos son los números, las letras, etc.

Otro elemento a considerar es la forma como encontramos los objetos en la naturaleza, o como los concibe el hombre en su mente, en el caso de los objetos intangibles o abstractos, pues todo lo que vemos está siempre en relación constante con su entorno, lo que significa que no se puede analizar un objeto como algo independiente, pero esta relación no solo es exógena, pues también en el estudio de cualquier objeto identificamos partes, y todas estas están relacionadas de tal forma que le dan vida al objeto. Lo anterior implica que todo objeto, independiente de su naturaleza, está siempre constituido por más objetos, y que estos están dispuestos de tal manera que se relacionan en una forma lógica.

¿Qué es lo que nos debe interesar de los objetos?

Al estudiar un objeto debemos conocer todo acerca de él; su nombre, su uso, su relación con el medio, y para ello es fundamental identificar todas sus características o atributos, pues son ellas las que lo describen. A medida que logremos conocer plenamente los objetos, podremos realizar procesos importantes como el de clasificación, podremos enmarcarlos dentro de una 'familia' de objetos y podremos asociarlos o relacionarlos con objetos afines con los que compartan algunas características; este proceso (el de clasificación) es de vital importancia en la Programación Orientada a Objetos (POO), que es el tema que vamos a abordar de aquí en adelante.

Debemos tener en cuenta que la informática siempre se ha basado en las actividades cotidianas desarrolladas por el hombre para crear los modelos que a futuro representen esa realidad y mediante los cuales desarrolla las aplicaciones que solucionan los problemas desde diversas perspectivas, haciendo uso de herramientas tanto del área de la informática como de las comunicaciones. La POO no es la excepción, lo que pretende es abstraer la realidad para crear modelos computacionales con los que se solucionen problemas del entorno, enmarcados en ámbitos de la industria, de la educación, de la ciencia y en general de todos los aspectos en los que se involucran las personas.

Comencemos a trabajar con objetos. Ya dijimos que los objetos pueden ser tangibles o reales, y abstractos o conceptuales; para aterrizar la idea tratemos de identificar objetos del entorno y aquellos conceptos que a diario utilizamos.

Del listado anterior de objetos, se sugiere al lector realizar una clasificación en dos clases: objetos reales y objetos intangibles.

A continuación se procederá a identificar los atributos o características más representativos del objeto seleccionado (persona): nombre, identificación, teléfono, dirección, edad, fecha de nacimiento, peso, estatura, etc.

Es bueno aclarar que a los objetos se les puede identificar muchas características, pero que, de acuerdo con lo que se pretenda, solo se deberán tomar algunas de ellas y que, dependiendo del ambiente en el cual se analiza el objeto, su prioridad variará.

Representación gráfica de un objeto

Existen varias propuestas de la forma como se puede representar gráficamente un objeto, incluyendo la relación de este con su entorno; sin embargo, en este nivel no se abordará de manera formal este aspecto y solo se utilizará un modelo que inicialmente no se asociará a ningún autor reconocido en el análisis orientado a objetos.

Nombre_clase	Persona
Atributos	Nombre, edad, ID, teléfono,
Métodos	Leer(), calcular(), mostrar()

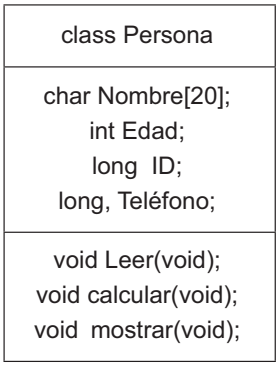
Figura 1. Representación gráfica de un objeto real.

Fuente: los autores.

La figura 1 representa el objeto persona; en la primera sección aparece el nombre, en la sección del medio aparecen los atributos y en la parte inferior los métodos o funciones.

Como el objetivo es crear un modelo computacional, vamos a perfeccionar el modelo un poco más. Como el lenguaje en el que se va a codificar es el C++, se colocará en unos términos más afines a este.

Figura 1.1 Representación gráfica de un objeto real, con atributos y métodos



Fuente: los autores.

Lo anterior nos permite tener un modelo afín a lo que nos interesa producir, que en este caso es un programa; entonces lo que se hizo fue determinar los tipos de datos de cada uno de los atributos, que en esta instancia ya figuran como variables con sus respectivas longitudes; adicionalmente, se dio forma a los métodos, asociándola a un prototipo de función que no recibe parámetros y no retorna valores.

De igual manera, podemos hacer la representación de un objeto abstracto, en este caso se tomó el objeto *cuenta*, muy utilizado en las entidades financieras. Veamos:

Figura 2. Representación gráfica de un objeto intangible

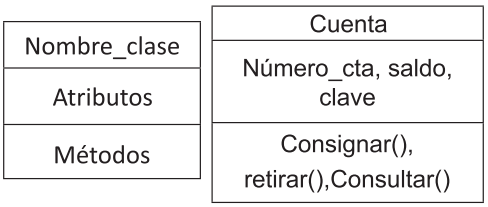
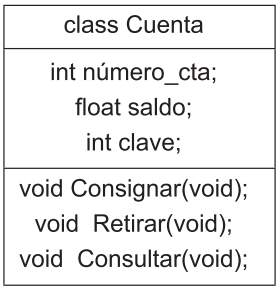


Figura 2.1 Representación gráfica de un objeto abstracto, con atributos y métodos



Fuente: los autores.

Recordemos que la cantidad de atributos y el tipo de datos de los mismos están determinados por el problema que se quiera solucionar; de esto dependerá en gran medida la 'calidad' de la solución que obtengamos.

Características de la POO

Uso de funciones

Bajo este paradigma, el uso de funciones diseñadas por el programador es indispensable; esto nos representa que ya no asumimos el problema de manera global, sino que podemos fragmentarlo en tantos subproblemas como deseemos; cada fragmento es solucionado mediante una función, y la suma de todas las soluciones me debe solucionar el problema global.

Mayor orden

Estructurar el código en pequeños bloques llamados funciones nos permite organizar nuestro código de una forma más clara y fácil de entender; esto en aplicaciones grandes es bastante útil, tanto para los usuarios programadores como para los usuarios analistas y para la misma administración de las compañías; si el código se entiende, cualquier otra persona podrá actualizarlo o reutilizarlo.

Reutilización de código

Al estar el código dispuesto en funciones, y si estas responden a una organización jerárquica, previo un proceso de clasificación, la POO nos brinda la posibilidad de reutilizar este código, es decir, si tengo una función que lee datos, y a posteriori necesito leer datos, me es suficiente con invocar la función sin necesidad de volverla a escribir en su totalidad; es un gran potencial de este paradigma.

Encapsulamiento

Al no estar todo el código dispuesto dentro de una sola función (por ejemplo, dentro de la función principal 'main'), sino dentro de una clase, y dentro de un número determinado de funciones, hace que el acceso a los datos sea un poco más complejo; pero esta complejidad redunda en algo que para los que manejan información es lo más importante: la seguridad; el encapsulamiento involucra esconder, reservar, y esto se logra con la característica aquí planteada. Podríamos decir que una clase es una cápsula, que tiene en su interior otras tantas que son las funciones.

Polimorfismo

En POO se denomina polimorfismo la capacidad que tienen los objetos de una clase de responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación. Un objeto polimórfico es una entidad que puede contener valores de diferentes tipos durante la ejecución del programa. Esto aplica principalmente a las funciones y a los operadores.

Herencia

Se refiere a la posibilidad que tienen los objetos de tomar objetos de otros de mayor jerarquía. Es una propiedad que permite que los objetos sean creados a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a las de los ya existentes. Es la relación entre una clase general y otra clase más específica. Es un mecanismo que nos permite crear clases derivadas a partir de la clase base, nos permite compartir automáticamente métodos y datos entre clases, subclases y objetos.

¿Qué es la Programación Orientada a Objetos?

Es un enfoque bastante utilizado para el desarrollo de aplicaciones software. La POO permite descomponer fácilmente un problema en subgrupos (objetos) de partes relacionadas, facilitando así la solución de los mismos.

Todos los lenguajes de programación orientados a objetos tienen tres cosas en común: objetos, polimorfismo y herencia.

La característica más importante de la programación orientada a objetos es el objeto. Un

objeto es simplemente una entidad lógica que contiene datos y un código que manipula estos datos. Este código y estos datos pueden ser públicos, privados o protegidos, lo que permite el encapsulamiento de los datos.

Un objeto lo podemos concebir en el campo de la programación como una variable de tipo definida por el usuario.

Introducción a la Programación Orientada a Objetos con C++

La clase es una herramienta poderosa de C++ para la programación orientada a objetos. Una clase es un objeto y para definir un objeto en C++ utilizamos la palabra reservada "class", su sintaxis es la siguiente:

```
class nombre_clase
  {
  private:
    datos y funciones privadas o protegidas;
  public:
    datos y funciones públicas;
  }lista de objetos;
```

Retomando nuestro objeto de estudio 'persona', miremos cómo llevamos el contenido de ese modelo gráfico a un programa utilizando clases y objetos; revisemos detenidamente la estructura como la sintaxis, pues es el modelo a seguir de aquí en adelante.

```
#include<iostream.h>        // línea 1
class persona               // línea 2

{                           // línea 3
private:                    // línea 4
char Nombre[20];            // línea 5
int Edad;                   // línea 6
long  ID, Teléfono;         // línea 7
public:                     // línea 8
void Leer(void);            // línea 9, 10 y 11
void  calcular(void);
void  mostrar(void);
};                          // línea 12
```

Miremos ahora qué es lo que se pretende hacer en cada una de las líneas, de forma independiente; si observamos con detenimiento, lo único que se le agregó al modelo gráfico fue la inclusión de la librería, esto aparece en la primera línea.

Línea 1: inclusión de la librería, toda vez que vamos a utilizar cout y cin, inicialmente con esta es suficiente, se recomienda revisar cuántas y cuáles librerías posee C++.

Línea 2: inicio de clase; utilizamos la palabra reservada class y el nombre del objeto que estamos estudiando, en este caso es persona.

Línea 3: con este corchete se simboliza el inicio de cuerpo de la clase.

Línea 4: inicio de un bloque privado; la inclusión de esta palabra es opcional; si no se coloca, la máquina asume que todo lo que aparece antes del bloque público es privado.

Línea 5: declaramos la variable nombre con una capacidad de 20 caracteres.

Línea 6: declaramos la variable edad de tipo entero.

Línea 7: declaramos las variables identificación (ID) y teléfono de tipo long.

Línea 8: inicio del bloque público, tanto aquí como en el bloque privado las palabras reservadas public y private van seguidas de dos puntos (:)

Línea 9, 10 y 11: declaración de las funciones leer, calcular y mostrar que no reciben ni retornan valores; este tipo de funciones casi siempre implica que los datos del programa deben ser globales.

Línea 12: el corchete, el punto y coma siempre simbolizan el fin del cuerpo de la clase; con mucha frecuencia, cuando iniciamos a programar con objetos se suele olvidar este detalle.

Hasta esta instancia podemos notar la coincidencia de nuestro modelo gráfico con el encabezado de nuestro programa; inicialmente, dicha coincidencia se da entre el bloque de atributos con el bloque privado y el bloque de métodos con el bloque público. Como se expuso en el modelo genérico de las clases, tanto en el bloque privado como en el público podemos encontrar funciones y variables.

¿Para qué objetos privados y públicos?

Al estar todos los datos encapsulados o ser miembros de una clase, se deben seguir unas reglas para su acceso y manipulación; se asume que en la POO la regla de oro nos dice: "*que todos los datos que sean privados la única forma de accederlos es a través del bloque público*".

Al comienzo dijimos que todos los objetos estaban constituidos por más objetos y que estos estaban interrelacionados entre sí; entonces, para completar nuestro programa es importante recordar este aspecto, pues en la definición de las funciones y en su invocación debemos hacer ver que estos no son libres, sino que son miembros de una clase. Veamos.

Definición de la función leer: para definir una función miembro, siempre debemos escribir el prototipo de la función (lo que precede al nombre), el nombre de la clase, el operador de ámbito de resolución (::) y, por último, el nombre de la función; luego entre corchetes se le da el cuerpo a la función, es decir, escribimos sus instrucciones.

La función leer capturará del teclado los datos (atributos) que están incluidos en el objeto.

```
void persona::leer(void)                    //línea 1
{                                           //línea 2
  cout<<"Digite nombre   "<<endl;           //línea 3
  gets(nombre) ;                            //línea  4
  cout<<"Digite edad   "<<endl;             //línea 5
  cin>>edad ;                               //línea 6
cout<<"Digiteidentificación"<<endl;//línea7
  cin>>ID ;                                 //línea 8
  cout<<"Digite teléfono   "<<endl;         //línea 9
  cin>>teléfono;                            //línea 10
}                                           //línea 11
```

Ahora se explicará lo que hace cada línea.

Línea 1: encabezado de la definición de la función, como se explicó en el párrafo anterior.

Línea 2: inicio del cuerpo de la función.

Líneas 3, 5, 7 y 9: se imprime en pantalla la petición de un dato y se pasa a la línea siguiente (endl).

Líneas 4, 6, 8 y 10: se recibe el dato digitado por el usuario; dependiendo del tipo de dato, se utiliza o cin o gets.

Línea 11: este corchete simboliza el fin de la función, debe siempre verificarse que el número de corchetes que se abren sea igual al número de corchetes que se cierran.

Ahora definamos la función mostrar:

```
void persona::mostrar(void)              //línea 1
{                                        //línea 2
 cout<<"Nombre "<<nombre<<endl;          //línea 3
cout<<"Edad "<<edad<<endl;               //línea 4
cout<<"Identificación "<<ID<<endl;       //línea 5
cout<<"Teléfono "<<teléfono<<endl;       //línea 6
}                                        //línea 7
```

Línea 1: encabezado de la definición de la función.

Línea 2: inicio del cuerpo de la función.

Líneas 3, 4, 5 y 6: se imprime en pantalla el nombre del dato solicitado, se imprime el contenido de la variable que lo almacenó y se pasa a la línea siguiente (endl).

Línea 7: corchete que significa que el cuerpo de la función terminó.

En el método calcular se identificará si los datos corresponden a una persona mayor de edad o no. Veamos la definición del método calcular:

```
void persona::calcular(void)             //línea 1
{                                        //línea 2
 if(edad>=18)                            //línea 3
 cout<<nombre<<"Es mayor de edad
 "<<endl;                                //línea 4
 else                                    //línea 5
 cout<<nombre<<"Es menor de edad
 "<<edad<<endl;                          //línea 6
}                                        //línea 7
```

Ahora se explicará lo que hace cada línea.

Línea 1: encabezado de la definición de la función, como se explicó anteriormente.

Línea 2: corchete que simboliza el inicio del cuerpo de la función.

Línea 3: se evalúa mediante un condicional compuesto si la edad corresponde a una persona mayor de 18 años.

Línea 4: si la condición se cumple (es decir que es verdadera) muestra en pantalla el nombre de la persona y la leyenda 'Es mayor de edad'.

Línea 5: esta opción se ejecuta si la condición de la línea 3 no se cumple.

Línea 6: muestra en pantalla el nombre de la persona y la leyenda 'Es menor de edad' y pasa a la siguiente línea.

Línea 7: fin de la función.

El orden de la definición de las funciones no es relevante, por eso aparece primero la función mostrar que la función calcular; en la instancia de la invocación de las funciones este orden sí es relevante.

En este momento lo único que nos hace falta es la función principal (main); en el esquema gráfico inicial esta no aparece referenciada, pues como todo programa debe tenerla, y por tanto no se incluye.

A la hora de definir la función principal, también debemos recordar que las funciones son miembro de la clase persona, por lo tanto debemos hacer ver esta relación. Para poder obtener esto, debemos declarar un objeto y relacionarlo con la función a través del operador punto.

El objeto lo declaramos usando el nombre de la clase y luego asignando el nombre del objeto; este es aquel que deseemos como programadores y que cumpla con las normas

para la declaración de variables; adicionalmente, recordemos que toda sentencia en C++ remata con punto y coma (;).

```
int main()                          //línea 1
{                                   //línea 2
persona estudiante;                 //línea 3
estudiante.leer();                  //línea 4
estudiante.calcular();              //línea 5
estudiante.mostrar();               //línea 6
system("pause");                    //línea 7
}                                   //línea 8
```

Ahora se explicará lo que hace cada línea.

Línea 1: inicio de la función principal.

Línea 2: corchete que indica el inicio del cuerpo de la función principal.

Línea 3: declaración del objeto estudiante de tipo persona.

Líneas 4, 5 y 6: invocación de la función leer, la función calcular y la función mostrar (respectivamente).

Línea 7: esta instrucción nos mantiene la última pantalla de la ejecución del programa.

Línea 8: corchete de fin de función principal.

Ahora veamos el programa en su conjunto.

```
#include<iostream.h>
class persona
{
private:
char nombre[20];
int edad;
long  ID, teléfono;
public:
void leer(void);
void  calcular(void);
void  mostrar(void);
};
void persona::leer(void)
{
 cout<<"\n\n\t****  Programa Datos Perso-
 nales ****  "<<endl;

 cout<<"\n\tDigite nombre         ==>   ";
 gets(nombre) ;

 cout<<"\tDigite edad             ==>   ";
 cin>>edad ;

 cout<<»\tDigite identificación   ==>   «;
 cin>>ID ;

 cout<<»\tDigite teléfono         ==>   «;
 cin>>teléfono;
}

void persona::mostrar(void)

{
  cout<<»\n\n\t****    Datos  Recibidos  ****
 «<<endl;

 cout<<»\n\tNombre    «<<nombre<<endl;

 cout<<»\tEdad              «<<edad<<endl;

 cout<<»\tIdentificación       «<<ID<<endl;

cout<<»\tTeléfono       «<<teléfono<<endl;
}

void persona::calcular(void)

{
 if(edad>=18)
   cout<<»\n\n\t»<<nombre<<»   de
   «<<edad<<»  agnos, es mayor de edad
   «<<endl<<endl;
 else
 cout<<»\n\n\t»<<nombre<<»   de
```

```
 «<<edad<<»  agnos, es menor de edad
 «<<endl;
}
int main()
{
persona estudiante;
estudiante.leer();
system(«cls»);
estudiante.mostrar();
estudiante.calcular();
system(«pause»);
}
```

El éxito en la ejecución del anterior programa depende del compilador con el cual se esté trabajando; las pantallas que se presentan a continuación corresponden al compilador DEV C, que es un software libre que se puede conseguir en internet.

En compiladores como los del sistema Linux también se puede ejecutar, tan solo que el procedimiento para hacerlo es un poco diferente al utilizado en un ambiente gráfico. En Linux se compila con la siguiente instrucción: g++ persona.cpp –o ejecutable. Lo que aparece después de g++ corresponde al nombre que le haya dado al programa fuente (con extensión CPP), y lo último que aparece es el nombre que va a tomar el archivo ejecutable. Para la ejecución del programa se utiliza la siguiente instrucción: ./ejecutable. Un punto y una barra inclinada preceden al nombre del archivo ejecutable. Es bueno aclarar que los resultados que se obtienen en sistemas Windows como en Linux son iguales.

A continuación se presentan los resultados del programa, tomando datos para que ejecute las dos opciones del *if*; inicialmente cuando la condición es verdadera, ver figuras 3 y 4; luego cuando la condición no se cumple y ejecuta la opción *else,* ver figuras 5 y 6. Veamos:

Figura 3. Ejecución del programa, condición verdadera

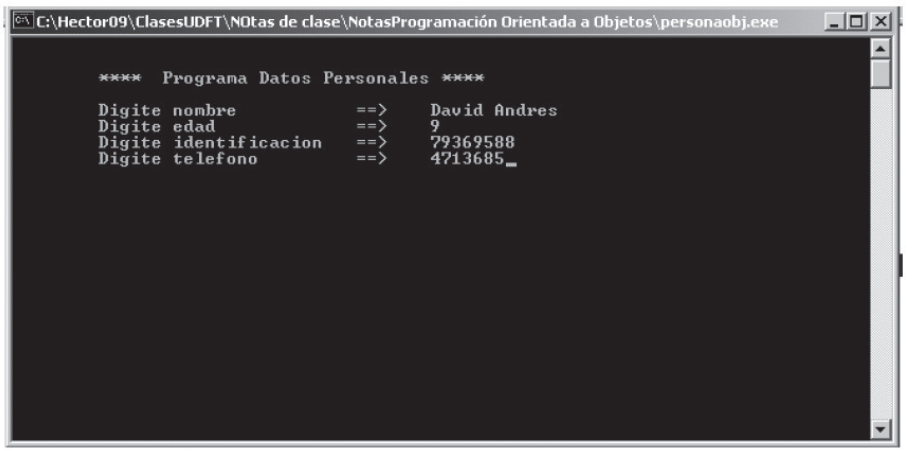


Figura 4. Ejecución del programa, condición verdadera, datos de salida

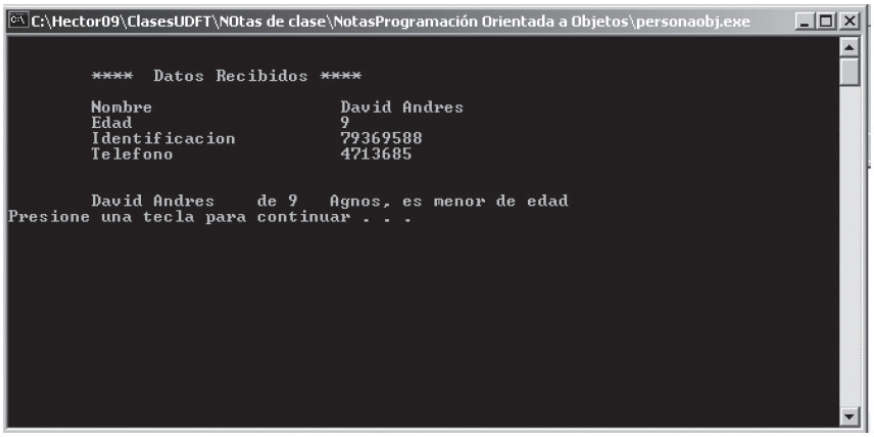


Figura 5. Ejecución del programa, condición falsa

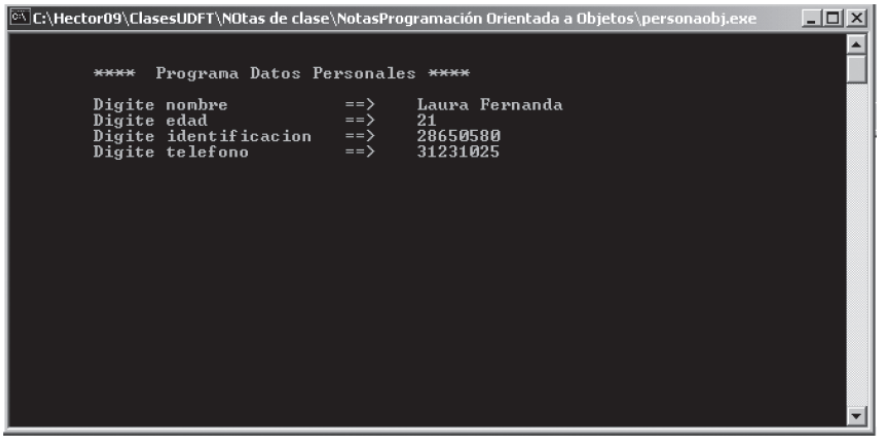
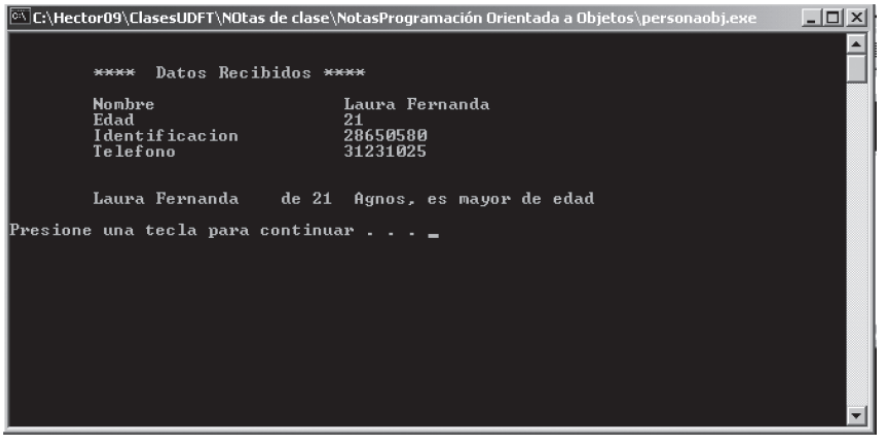


Figura 6. Ejecución del programa, condición falsa, datos de salida



Conclusiones

El conocer los objetos, su naturaleza, sus relaciones con el entorno y con el hombre hace que este modelo de programación sea de fácil acceso para los estudiantes que ya cuentan con bases en lógica y en programación.

A pesar de que aquí se abordó solo un pequeño ejemplo de lo que es la POO, podemos identificar varias fortalezas de este paradigma de programación de computadores; si esto lo sumamos a todo el potencial que tiene el C al darle al programador la libertad de manipular todos los recursos de la máquina inclusive a nivel de bits, podremos obtener grandes resultados.

En los últimos años ha surgido un sinnúmero de lenguajes de programación, pero podemos afirmar que el C++ sigue siendo el mejor leguaje para quienes realmente quieren aprender a programar computadores. El C++ nos ofrece la posibilidad de manipular los datos a nivel de bit, interactuar con *assembler* y otras herramientas de una forma fácil y rápida. Sistemas operativos como Unix y Windows fueron desarrollados en C++.

En el entorno todos los objetos están constituidos por más objetos y estos a través de su interrelación conforman un todo; en programación en C++ esto sigue siendo válido y a través de operadores se pueden establecer estas relaciones; de forma tal que se asimila a lo que sucede en la realidad con los objetos.

Bibliografía

[1] Joyanes, Luis. *Programación en C++. Algoritmos, estructuras de datos y objetos*. McGraw-Hill. 2000.

[2] Schild, Herbert. *Aplique Turbo C++ para Windows*. Osborne - McGraw-Hill. 1994.

[3] Joyanes, Luis, Borland. *C++. Iniciación y referencia*. McGraw-Hill. 1996.

[4] Joyanes, Luis. *C++ a su alcance*. McGraw-Hill. 1994.

[5] Perry, Greg. *Aprendiendo programación orientada a objetos con Turbo C++ en 21 días*. Prentice Hall. 1995.

[6] Martin, James y Odel. *Análisis y diseño orientado a objetos*. Prentice Hall. 1996.

[7] Becerra, César. *C++: una herramienta para la programación orientada a objetos.*

[8] *Jamas, C.* Aprenda C++ paso a paso.

[9] Deitel, D. *Cómo programar C++*. Prentice Hall. 2006.

[10] Kernighan, Brian W., Ritchie, Dennis. *El lenguaje de programación C++*. Prentice Hall.