Facultad de Ingeniería y Ciencias Hídricas

FICH

UNL

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Ingeniería Informática

PROGRAMACIÓN ORIENTADA A OBJETOS

UNIDAD 2

Introducción a la Programación Orientada a Objetos

2011

UNIDAD 2

Introducción a la Programación Orientada a Objetos

Ejercicio 2.1

Diseñe una clase *Cilindro* que modele un cilindro con el objetivo de calcular el volumen del cuerpo conociendo el radio y la altura del mismo.

a) Cree los métodos *AsignarDatos()* y *CalcularVolumen()* para asignar los datos del problema y calcular el volumen del cuerpo.
b) Escriba un programa cliente que utilice dicha clase. Defina 2 instancias de *Cilindro* llamadas c1 y c2. El objeto c1 debe utilizar datos ingresados por el usuario, mientras que para c2 utilice 5.3 cm y 10.2 cm para el radio y la altura respectivamente.
c) Agregue un constructor a la clase *Cilindro* que reciba 2 parámetros para inicializar el radio y la altura. Luego intente compilar nuevamente el programa. ¿Puede explicar por qué se produce el error?
d) Elija y realice alguna de las 2 siguientes modificaciones:
   - Cree un constructor que no reciba parámetros e inicialize el radio y la altura en 0.
   - Modifique el constructor de *Cilindro* para emplear parámetros por defecto.

   Intente compilar el programa nuevamente. ¿Se producen errores de compilación esta vez?¿Por qué?

Ejercicio 2.2

Cree una clase *Rectangulo* que posea:

a) Un constructor que permita crear el objeto a partir de la base y altura del rectángulo.
b) Otro constructor que permita crear el objeto a partir de las coordenadas (x, y) de sus 4 vértices.
c) Métodos *Area*() y *Perimetro*() que calculen respectivamente el área y perímetro del rectángulo.
d) Un método *EsCuadrado*() que permita saber si el rectángulo es cuadrado.
e) Cree un programa cliente donde utilice varias instancias de la clase *Rectangulo*. Cree los objetos dinámicamente y no olvide borrarlos al terminar de utilizarlos.

Ejercicio 2.3

Proponga una clase *EcuaciónCuadratica* para modelar ecuaciones cuadráticas de la forma $ax^2+bx+c=0$. La clase debe incluir:

a) Un constructor que reciba los valores de los coeficientes *a*, *b* y *c*.
b) Una función **bool** *ObtenerRaices*(**float** &x1, **float** &x2) que devuelva verdadero o falso según las raíces de la ecuación sean reales o no y además devuelva por referencia en x1 y x2 los valores de las raices reales o los coeficientes de las raices complejas.
c) Una función con el prototipo **bool** *TieneRaicesReales*(**void**) que devolverá verdadero o falso según las raíces de la ecuación sean reales o no. Dicha función no estará disponible para ser utilizada directamente por el usuario (es decir, será privada) y será utilizada por la función *ObtenerRaices*() para saber el tipo de raíces.
d) Cree un programa cliente que utilice objetos de la clase *EcuaciónCuadratica* e intente llamar a la funcion *TieneRaicesReales*() a partir de uno de los objetos creados. ¿Qué es lo que sucede?

Ejercicio 2.4

Escriba una clase llamada *Fecha* con atributos para definir una fecha (día, mes, año). Implemente los siguientes métodos:

a) Un constructor que reciba el día, mes y año y los inicialice.
b) Métodos para devolver el día, mes y año de la fecha.
c) Una función *DiferenciaEnAnios()* que reciba otro objeto de tipo fecha y devuelva la diferencia en años entre ambas.
d) **(OPCIONAL)** Investigue como crear un constructor que no reciba parámetros e inicialize la fecha con los datos del reloj de la CPU.
e) **(OPCIONAL)** Cree una función llamada *SignoZodiacal()* que devuelva una cadena de texto con el nombre del signo del zodiaco al que pertenece la fecha.

Implemente un programa cliente que utilice dicha clase.

Ejercicio 2.5

Implemente una clase *VectorDinamico* que posea como atributo un puntero a entero que apunte a la memoria donde se almacenan los datos. Dicha clase debe poseer:

a) Un constructor que reciba el tamaño inicial del vector, reserve la memoria necesaria e inicialize los valores del vector de manera aleatoria.
b) Un destructor que se encargue de liberar la memoria reservada.
c) Un método *Duplicar()* que duplique la cantidad de memoria reservada manteniendo los datos que ya estaban en el vector e inicializando al azar los nuevos valores.
d) Un método *VerValor()* que reciba el número de elemento y devuelva su valor.

Cree un programa cliente que muestre la utilización de todas las funciones implementadas.

Ejercicio 2.6

Escriba una clase *POOString* que permita modelar una cadena de texto. La clase debe manejar dinámicamente la memoria para guardar la cadena de texto y para

ello poseerá como atributo un puntero de tipo **char** que apuntará a la memoria donde se almacena la cadena. La clase debe tener:

a) Un constructor que reciba como parámetro una cadena para inicializar el objeto. El constructor deberá reservar la cantidad de memoria necesaria para la cadena (tener en cuenta el carácter '\0' a la hora de reservar memoria) y guardar una copia de la misma.
b) Un destructor que libere la memoria reservada por el constructor.
c) Una función **char** **GetString*(**void**) que devuelva un puntero a la cadena para poder mostrarla.
d) **(OPCIONAL)** Una función **void** *Copy*(**char** *c) que copie la cadena c en la memoria del objeto. Para esto deberá descartar la memoria reservada anteriormente y reservar una nueva porción de memoria con el tamaño adecuado para la nueva cadena.
e) **(OPCIONAL)** Una función **void** *Cat*(**char** *c) que concatene la cadena c a la cadena existente en la memoria del objeto. Para esto deberá descartar la memoria reservada anteriormente y reservar una nueva porción de memoria con el tamaño adecuado para contener la concatenación de ambas cadenas.
f) Utilize la clase POOString desde un programa cliente.
g) Intente compilar el siguiente código y luego responda:

```
...
int main(int argc, char *argv[]) {
     POOString a("Esta es una POOString");
     POOString b(a);

     cout<<a.GetString()<<endl;
     cout<<b.GetString()<<endl;
     return 0;
}
...
```

- ¿A qué se denomina un *constructor de copia*?
- ¿Por qué la compilación es correcta a pesar de no haber definido ningún constructor de ese tipo?
- ¿El constructor utilizado para el objeto b hace lo que realmente se desea?

Ejercicio 2.7

Se dispone del siguiente tipo de dato:

```
struct Alumno {
     char nombre[30];
     float nota;
};
```

En base al mismo se desea crear una clase *Curso* para modelar el cursado de diversas materias. La clase deberá contener el nombre de la materia y la cantidad de

alumnos en el curso (dicha cantidad nunca superara los 40 alumnos) junto con una lista de los mismos. Proponga los siguientes métodos:

a) Constructores y destructores según lo considere conveniente.
b) Un método que permita agregar un alumno especificando su nombre y su calificación.
c) Un método que determine el promedio del curso.
d) Un método que devuelva la calificación mas alta y el nombre del alumno que la obtuvo.

Ejercicio 2.8 (OPCIONAL)

Consulte la bibliografía y averigüe el significado de la palabra reservada *static* y su utilización en las clases. Proponga una clase *CuentaObjetos* que permita llevar cuenta de los objetos que son creados. Para ello defina un constructor y un destructor que notifiquen por pantalla que el objeto fue creado/destruido y además muestre la cantidad de objetos que aun se encuentran en memoria. Utilice un programa cliente para probar la clase y realiza la creación/destrucción de los objetos de forma dinámica.

Ejercicio 2.9 (OPCIONAL)

Consulte la bibliografía y averigüe el significado de la palabra reservada *static* y su utilización en las clases. Proponga una clase llamada *MathUtils* que posea una función que calcule la distancia entre 2 puntos recibiendo sus coordenadas. Cree un programa cliente que llame a dicha función sin crear ningún objeto.

Ejercicio 2.10 (OPCIONAL)

a) Intente responder a la siguiente pregunta antes de continuar leyendo este enunciado: ¿Tiene sentido colocar un constructor en la parte privada de una clase?
b) Analice el siguiente código y determine qué particularidad tendrá esta clase:

```
class Singleton {
    Singleton() { /*unico constructor*/ };
    static Singleton *instancia;
public:
    static Singleton *ObtenerInstancia() {
            if(!instancia) instancia=new Singleton();
            return instancia;
    };
    // ...interfaz de la clase...
};
Singleton *Singleton::instancia=NULL;
```

Cuestionario

1. ¿Qué es un objeto? ¿Qué es una clase? ¿Qué es una instancia?
2. ¿Qué es un constructor?
3. ¿Cuándo se invoca al constructor de un objeto?¿Cuándo al destructor?
4. ¿Qué significa constructor por defecto?¿Y constructor de copia?
5. ¿Qué significa encapsulamiento?
6. ¿Cómo se consigue el ocultamiento de datos en C++?
7. ¿Para qué sirven las etiquetas *private* y *public*?
8. ¿Cuál es la diferencia, en C++, entre una clase y una estructura?
9. ¿Para que sirve el puntero *this*?
10. ¿Es posible usar el mismo nombre de función para una función miembro de una clase y una función externa? Si es posible, indique como distinguirlas. En caso contrario, indique las razones.
11. ¿Qué implica que un atributo se declare con el modificador *static*?¿Que pasa cuando un método es declarado con dicho modificador?