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 3
Relaciones entre clases

2011

UNIDAD 3

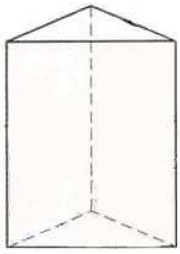
Relaciones entre clases

Ejercicio 3.1

Defina la clase *Circulo* con atributos y métodos que permitan obtener el área y el perímetro de esta figura. Proponga los constructores y métodos que considere necesarios. Luego, proponga una función amiga externa a la clase Circulo, llamada reducir(), que reciba un objeto de tipo Circulo como parámetro y devuelva otro Circulo con la mitad del diámetro del parámetro, accediendo a los atributos privados o protegidos del parámetro directamente. Utilice todo en un programa C++.

Ejercicio 3.2

Cree e inicialice dos objetos, uno de tipo Triángulo (equilátero) y otro de Tipo Rectángulo, cada clase consta de atributos y métodos para calcular su superficie. Diseñe un programa que calcule la superficie de un prisma triangular regular, cuyas caras son los objetos anteriores, a través de una función externa amiga acceda a los atributos privados y realice el cálculo, en el programa principal.



Ejercicio 3.3

La clase PrismaRectangular, consta de tres pares de rectángulos distintos que forman sus caras. Defina atributos y métodos que permitan obtener el área y el volumen de ese cuerpo. Luego otra clase Amiga Cubo que tiene los atributos y métodos para representar un cubo, calcular su área y volumen. Además tiene un método llamado ConvertirEnCubo que recibe un objeto tipo PrismaRectangular y devuelve un Cubo con las dimensiones del primer par de rectángulos de ese prisma, accediendo a los datos privados o protegidos directamente. Demuestre su implementación en un programa C++.

Ejercicio 3.4

Diseñe una clase *Persona* que contenga los siguientes atributos: apellido y nombre, DNI, año de nacimiento y estado civil. La clase debe poseer, además, un método *Edad()* que calcule la edad actual de la persona en base al año de nacimiento.

Implemente una clase *Alumno* para contener la siguiente información de un alumno: apellido y nombre, DNI, año de nacimiento, estado civil, promedio y cantidad de materias aprobadas. La clase debe poseer, además, un método *MeritoAcadémico()* que devuelve el mérito académico del alumno (este se calcula como el cociente entre el promedio y la cantidad de materias aprobadas.

Cree, también, una clase *Docente* para modelar un docente a partir de la siguiente información: apellido y nombre, DNI, año de nacimiento, estado civil y año de in-

greso. La clase debe poseer, además, un método *Antiguedad()* que calcule la antigüedad del docente en base a su año de ingreso.

Proponga una jerarquía de clases adecuada para evitar repetir atributos. Implemente constructores y métodos extra que considere adecuados. Codifique un programa cliente que cree instancias de *Alumno* y *Docente* y utilice sus funciones.

Responda:

- ¿Puede crearse un objeto de tipo persona?¿Para qué sirve esto?
- ¿Existe alguna clase abstracta en la jerarquía?

Ejercicio 3.5

Utilice las clases *Alumno* y *Docente* del ejercicio anterior para crear una clase *Curso* que modele el cursado de una materia. Cada curso tiene un nombre, un profesor a cargo y un número máximo de 10 alumnos. Implemente un método *AgregarAlumno* que permita agregar un alumno al curso y otro método *MejorPromedio()* que devuelva el alumno con mejor promedio. Proponga los constructores y métodos extra que considere necesarios.

Ejercicio 3.6

Proponga una clase *Punto* con atributos y métodos para definir un punto en el plano. Luego, proponga la clase *RectaExplicita* para definir la ecuación de la recta *y=mx+b* a partir de dos puntos. La declaración de dicha clase se muestra en el recuadro.

```
class RectaExplicita {
private:
   Punto P1, P2;
   float m, b;

public:
     RectaExplicita(Punto &p1, Punto &p2);
    void obtener_Ecuacion();
     float Ver_m();
     float Ver_b();
};
```

El método *MostrarEcuacion()* debe mostrar en pantalla la ecuación explícita de la recta.

Ejercicio 3.7

Proponga una clase *RectaGeneral* para representar una recta general, cuya ecuación es *Ax+By+C=0*, a partir de dos puntos. El prototipo de la clase se muestra en el siguiente recuadro.

```
class RectaGeneral {
private:
   Punto P1, P2;
   float a, b, c;

public:
     RectaGeneral(Punto &p1, Punto &p2);
      void obtener_Ecuacion();
     float Ver_a();
     float Ver_b();
     float Ver_c();
};
```

Diseñe un árbol de herencia que incluya una clase *Recta*, y dos herederas llamadas *RectaExplicita* y *RectaGeneral*. Utilizando los conceptos de polimorfismo, métodos virtuales y abstractos, desarrolle una estructura con métodos polimórficos.

Ejercicio 3.8

Explique el siguiente código

```
// constructores y clases derivadas
#include <iostream.h>

class madre {
  public:
    madre ()
      { cout << "madre: sin parámetros\n"; }
    madre (int a)
      { cout << "madre: parámetro int\n"; }
};

class hija : public madre {
  public:
    hija (int a)
      { cout << "hija: parámetro int\n\n"; }
};

class hijo : public madre {
  public:
    hijo (int a) : madre (a)
      { cout << "hijo: parámetro int\n\n"; }
};

int main () {
  hija cynthia (1);
  hijo daniel(1);

  return 0;
}
```

Ejercicio 3.9

En un videojuego existen tres tipos de personajes: caballero, arquero y mago. Cada uno debe guardar sus cantidades de vida y maná (magia). Todos los personajes tienen un método Atacar() que recibe por referencia otro personaje al que le quitará puntos de vida y mana. Sin embargo cada personaje produce distintas cantidades de daño:

- el caballero quita 6 unidades de vida y ninguna de maná al personaje que ataca.
- el arquero resta 2 puntos de vida y 4 de maná al atacar.
- el mago quita 3 puntos de maná y 3 de vida al adversario al que ataca.

Cada personaje posee además un método *EstaVivo()* que permite saber si el personaje aún tiene algo de energía, y una función *Tipo()* que devuelve una cadena de texto indicando la clase del personaje (caballero, arquero o mago).
Con estas especificaciones **modele las clases** que considere necesarias y cree además una clase *Juego* para representar un videojuego simple que contenga 30 personajes de distinto tipo en único arreglo. Cada personaje deberá elegir a otro al azar y atacarlo (recuerde que ningún personaje puede atacar si no está vivo). Esto se debe repetir hasta que solo quede un personaje vivo. Mostrar de que tipo es el mismo y la posición que ocupa en el vector.

Antes de comenzar a codificar, responda ¿Existen clases abstractas en la jerarquía? En caso de responder afirmativamente, indique cuales, de lo contrario, indique por qué.

(OPCIONAL) Proponga reglas más complejas para el comportamiento de los personajes, por ejemplo, que la acción de atacar consuma unidades de maná al personaje.
(OPCIONAL) Codifique en C++.

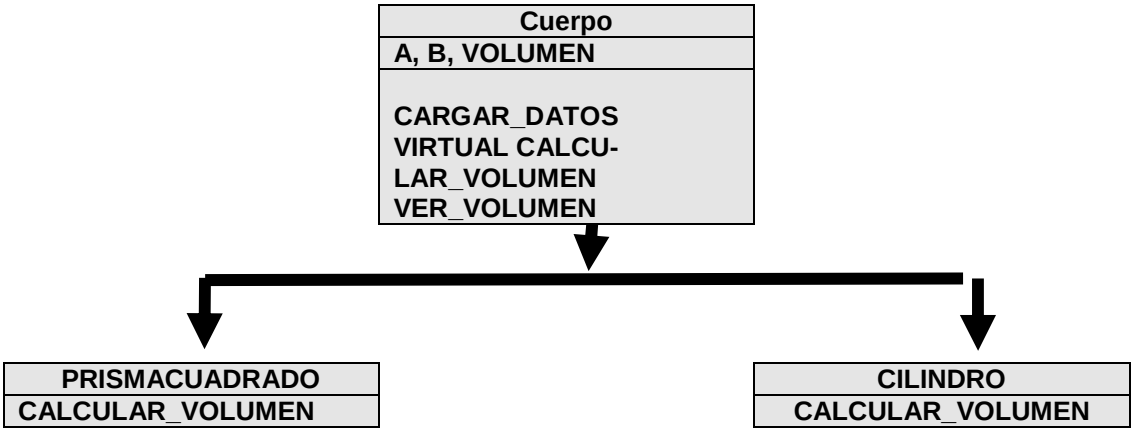
Ejercicio 3.10

Defina las clases *Circulo* y *Rectangulo* con atributos y métodos que permitan obtener el área de cada una de estas figuras. Los atributos deben inicializarse con el constructor. Proponga los constructores y métodos que considere necesarios. Luego, proponga la clase *Cilindro* reutilizando una o ambas clases anteriores. Los atributos deben inicializarse con el constructor. La clase *Cilindro* debe poseer un método *ObtenerVolumen()* que permita obtener el volumen del cuerpo. Escriba un programa cliente en C++ que permita ingresar el radio y la altura del de un cilindro y permita obtener su volumen.

Ejercicio 3.11

Construya una clase base abstracta Cuerpo que tenga los atributos a, b y volumen, además un método virtual puro calcular_volumen. En las clases hijas Prismacuadrado y Cilindro implemente el método virtual calcular_volumen. En el programa principal cree instancias de Prisma y Cilindro, cree un puntero de

tipo Cuerpo, con este puntero inicialice y calcule el volumen de un prisma rectangular. Con el mismo puntero inicialice y calcule el volumen de un cilindro.



Ejercicio 3.12 **(OPCIONAL)**

Un banco tiene dos tipos de cuentas para los clientes; unas son cuentas de ahorro y las otras cuentas corrientes. Las cuentas de ahorro tienen un interés compuesto y la posibilidad de reintegro, pero no admiten talonarios de cheques. Las cuentas corrientes ofrecen talonarios de cheques, pero no tienen interés. Los titulares de una cuenta corriente también deben mantener un saldo mínimo; si el saldo desciende por debajo de este nivel, se les cobra una comisión por el servicio.
Cree una clase *Cuenta* que almacene: el nombre del titular, el número de cuenta y el tipo de cuenta. A partir de ésta, derive las clases *CuentaCorriente* y *CuentaAhorro* para adaptarlas a los requerimientos específicos. Incluya las funciones miembro necesarias para realizar las siguientes tareas:

a) Aceptar un ingreso de un titular y actualizar el saldo.
b) Mostrar el saldo.
c) Calcular y abonar los intereses.
d) Permitir un reintegro y actualizar el saldo.
e) Comprobar que el saldo no esté por debajo del mínimo, imponer la sanción si es necesario, y actualizar el saldo.

Ejercicio 3.13 **(OPCIONAL)**

a) Crear una clase llamada **Calcu**, esta debe tener un constructor que reciba dos valores enteros (a y b) como argumentos para realizar operaciones y debe tener funciones públicas para sumar, restar, dividir, multiplicar, elevar a una potencia (de cualquier grado) y para modificar dichos números. La función pública que calcula potencias debe llamar a una función auxiliar privada que le permita multiplicar "n" veces un número pasado como argumento.

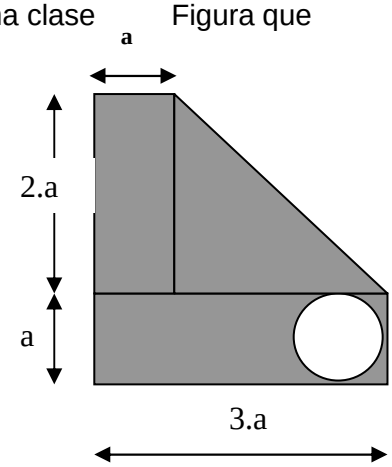
b) Derivar de **Calcu** una clase **Calcu_cientifica** que contenga métodos públicos para calcular las funciones trascendentes: seno, coseno y tangente.
NOTA: considere que el lenguaje de programación a emplear carece de estas funciones trigonométricas por lo que el seno debe calcularse aproximadamente con las expresiones:

seno(x) = $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots\ldots\ldots\ldots$; (*tomar 5 términos*)

cos(x)= 1- seno(x);
tan(x)= seno(x)/cos(x);

Ejercicio 3.14 (OPCIONAL)

Reutilice las clases Rectangulo y Circulo para componer una clase Figura que represente a la imagen de la derecha.
La clase debe tener –entre otros- un método que obtenga el área sombreada de la figura.
Pruebe la clase en un programa C++ cliente.
La solución se puede pensar con 2 o 3 rectángulos y 1 circulo.



Ejercicio 3.15 (OPCIONAL)

a) ¿Cuál es la salida del siguiente programa?
b) ¿Qué sucede si quita la palabra virtual?
c) Analice el resultado obtenido en cada caso y comente.

```
class B{//clase padre
public:
        virtual void m() {cout<<"B::m()"<<endl;};
};

class D : public B {//clase hija
public:
        void m() {cout<<"D::m()"<<endl;};
};

int main() {
        D d1;
        B * p=&d1;
        p->m();
}
```

Ejercicio 3.16 **(OPCIONAL)**

Aplicando el concepto de herencia y polimorfismo: a) Proponga la clase CPoligono y como clases derivadas CRectangulo y CTriangulo. Estas clases son empleadas en el programa del recuadro.

a) Proponga una función virtual pura para el método area() de la clase base CPoligono.

b) Proponga un método virtual area() de las clases derivadas de CPoligono.

c) Cuál es la salida del programa si no se emplea la palabra virtual en el método area()?.

```
int main () {
  CRectangulo rect;
  CTriangulo trgl;
  CPoligono * p1 = &rect;
  CPoligono * p2 = &trgl;
  p1->fijar_valores (3,5);
  p2->fijar_valores (4,6);
  cout << p1->area() << endl;
  cout << p2->area() << endl;
  return 0;
}
```

Cuestionario

1. ¿Qué significa herencia?
2. ¿A qué se denominan clase base y clase heredada?
3. ¿Cuándo se utiliza la etiqueta *protected* en un miembro de una clase?
4. ¿Qué es herencia múltiple?
5. ¿Pueden crearse instancias de una clase base?
6. ¿Para que sirve la palabra reservada *virtual*?
7. ¿Qué es una clase abstracta?
8. ¿Qué es un método virtual?¿Y un método virtual puro?
9. ¿Qué significa agregación o inclusión?
10. ¿En qué se diferencian agregación y herencia?
11. Un método abstracto, ¿es siempre virtual?. ¿Y uno virtual siempre abstracto?
12. ¿Qué significa polimorfismo?¿Y qué es invocación polimórfica en C++?
13. Antes de comenzar a codificar, ¿cómo reconoce que dos clases conforman una relación de herencia?¿Cómo reconoce que dos clases pueden componerse mediante una relación de agregación?