

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



Ingeniería Informática

PROGRAMACIÓN ORIENTADA A OBJETOS

UNIDAD 03 **Relaciones entre clases**

Guía de trabajos prácticos
2023

UNIDAD 03

Relaciones entre clases

Ejercicio 1

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

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 *MeritoAcademico()* que devuelva el mérito académico del alumno (éste se calcula como el producto 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 fecha de ingreso. La clase debe poseer, además, un método *Antigüedad()* que calcule la antigüedad del docente en base a su fecha de ingreso y la fecha actual.

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 2

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 50 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

Implemente una clase **Monomio** para representar un monomio de la forma $a \cdot x^n$ a partir de un coeficiente y un exponente, con un método *Evaluar(...)* que reciba un real y retorne el valor del monomio evaluado con ese real, y los demás métodos que considere necesarios. Implemente, luego, una clase **Polinomio** que reutilice la clase **Monomio** para modelar un polinomio, y añada un método *Evaluar(...)* para evaluar un polinomio en un valor x real dado. ¿Qué relación debe haber entre las clases **Monomio** y **Polinomio**?

Ejercicio 4

Implemente una clase **Fraccion** para representar una fracción a partir de un numerador y un denominador, con un método *ConvertirADouble()* para obtener el real que representa, y los demás métodos que considere necesarios. Implemente una clase **NumeroMixto** para representar un número formado por una parte entera y una fracción impropia (fracción menor a 1). Reutilice la clase **Fraccion** al implementar la clase **NumeroMixto**. La clase **NumeroMixto** debe también poseer un método *ConvertirADouble()*. ¿Qué relación entre clases puede utilizar en este caso?

Ejercicio 5

Proponga un struct **Punto** con atributos para definir un punto en el plano (coordenadas x e y). 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 siguiente.

```
class RectaExplicita {
private:
    float m, b;
public:
    RectaExplicita(Punto p, Punto q);
    float verM();
    float verB();
    string obtenerEcuacion();
    bool pertenece(Punto p);
};
```

El método *obtenerEcuacion()* debe retornar una cadena de texto con la ecuación explícita de la recta, mientras que el método *pertenece(p)* debe determinar si el punto p está o no sobre la recta (si satisface la ecuación).

Ayuda 1: las ecuaciones para obtener m y b a partir de dos puntos P y Q son:

$$m = (Q_y - P_y) / (Q_x - P_x) \quad b = P_y - m P_x$$

Ayuda 2: la función *string to_string(float)* le permitirá convertir los coeficientes a strings; y además es posible concatenar strings con el operador $+$ ($s=s1+s2$).

Ejercicio 6

a) 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:  
    float a, b, c;  
public:  
    RectaGeneral(Punto p, Punto q);  
    float verA();  
    float verB();  
    float verC();  
    string obtenerEcuacion();  
    bool pertenece(Punto p);  
};
```

b) Diseñe un árbol de herencia que incluya una clase **Recta**, y dos clases herederas llamadas **RectaExplicita** y **RectaGeneral**.

c) Utilizando los conceptos de polimorfismo, métodos virtuales y abstractos, complemente el diseño con dos métodos virtuales: *obtenerEcuacion(...)*, para mostrar en pantalla la ecuación que corresponda para cada recta, y *pertenece(...)* para saber si un tercer punto dado está en la recta.

Ayuda: Las ecuaciones para obtener los coeficientes a , b y c son:

$$a = Q_y - P_y \quad b = P_x - Q_x \quad c = -a P_x - b P_y$$

Nota: al comparar flotantes en el segundo método, no debe utilizar `==`, sino preguntar de alguna otra forma si son *muy parecidos* en lugar de *exactamente* iguales.

Ejercicio 7

Una fábrica de Tanques los hace de forma de Cilindro o de Esfera, en ambos envases debemos rotular el volumen en m³ y el peso en kilogramos.

Modele una clase base Tanque con un constructor que reciba el peso, un método para consultar el peso, y un método virtual puro para obtener el volumen.

Modele la clase hija Cilindro que se construirá a partir de los argumentos adicionales radio y altura, cuya fórmula de volumen es: área de la base x altura, donde el área de la base se calcula como $\pi \cdot \text{radio}^2$; y otra clase hija Esfera con el radio como argumento adicional, cuya fórmula de volumen es: $\frac{4}{3} \pi \cdot \text{radio}^3$. En el programa principal debe usar un único puntero de tipo Tanque para crear primero un Cilindro y mostrar su volúmen, y luego una Esfera y también mostrar su volúmen.

Ejercicio 8

a. Defina una clase *Tecla* para representar una tecla de un piano. Cada tecla puede estar o no apretada, y tiene además una nota asociada (cuyo nombre se representará con un string). Su interfaz debe tener entonces:

- un constructor que reciba el nombre de la nota
- un método *VerNota* que retorne el nombre de la nota
- un método *Apretar* que cambie el estado de la tecla a apretada.
- un método *Soltar* que cambie el estado de la tecla a no apretada.
- un método *EstaApretada* que retorne true si la tecla está apretada, false en caso contrario

b. Defina una clase *Pedal* para representar el pedal de un piano. El pedal debe almacenar un valor (*float*) que indica la presión que el músico ejerce sobre el pedal. El constructor debe inicializar la presión en 0, y la clase debe tener métodos para modificarla y consultarla.

c. Reutilizando las clases *Tecla*, *Pedal* e *Instrumento*:

```
class Instrumento{  
    public:  
        virtual string VerTipo() { return "sin_nombre"; }  
};
```

defina una clase *Piano* que modele un instrumento de tipo "*piano*" con solo 7 teclas ("*do*", "*re*", "*mi*", "*fa*", "*sol*", "*la*" y "*si*") y 1 pedal. La clase piano debe tener métodos para:

- apretar una tecla, indicando el número de tecla, y que retorne la nota que debería sonar.
- soltar una tecla, indicando el número de tecla
- presionar el pedal, indicando la presión que se aplica

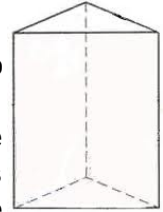
Cuestionario

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 qué 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. ¿Qué significa polimorfismo? ¿Y qué es invocación polimórfica en C++?
12. 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?

Ejercicios Adicionales

Ejercicio 2

Cree e inicialice dos objetos, uno de tipo Triángulo y otro de Tipo Rectángulo, cada clase consta de atributos y métodos para calcular su superficie y perímetro. Diseñe un programa que calcule la superficie de un prisma triangular, 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.



¿Es necesario utilizar amistad en este caso? ¿Es conveniente?

¿Si fuera a diseñar una clase Prisma para este fin, convendría utilizar herencia o composición? (justifique)

Ejercicio 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 tenga atributos y métodos para representar un cubo, calcular su área y volumen. Agregue además un método llamado *ConvertirEnCubo* que reciba un objeto tipo *PrismaRectangular* y devuelva un *Cubo* con las dimensiones del primer rectángulo de ese prisma, accediendo a los datos privados o protegidos directamente. Demuestre su implementación en un programa C++.

Ejercicio 4

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 7

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:

1. Aceptar un ingreso de un titular y actualizar el saldo.
2. Mostrar el saldo.
3. Calcular y abonar los intereses.
4. Permitir un reintegro y actualizar el saldo.
5. Comprobar que el saldo no esté por debajo del mínimo, imponer la sanción si es necesario, y actualizar el saldo.

Ejercicio 8

La clase *Widget* sirve para representar genéricamente a un control en una ventana (por ej: un cuadro de texto, un botón, una barra de scroll, etc):

```
class Widget {
    Ventana *m_v; // ventana que lo contiene
public:
    Widget() : m_v(nullptr) {}
    void SetearVentana(Ventana *v) { m_v = v; }
    Ventana* ObtenerVentana() { return m_v; }
    virtual void Dibujar()= 0;
    virtual ~Widget() {}
};
```

- a. Implemente una clase "Ventana" para representar la ventana gráfica de una aplicación. La clase debe tener un constructor que reciba su título; y métodos para: agregar un *Widget* a la misma (el *Widget* se recibe como argumento, será creado por el cliente y cedido a la ventana; la ventana deberá setearle su puntero *m_v*); dibujar la ventana (que debe invocar al método *Dibujar* de cada *Widget* que contenga).
- b. Implemente una clase para representar un control que solo muestra un mensaje en la ventana. La clase debe recibir como argumento en su constructor el mensaje a mostrar, y su método *Dibujar* solo deberá invocar a la función (que ya está hecha):

```
void escribirEnVentana(Ventana *v, string mensaje) { ... }
```

- c. Escriba un programa cliente que cree y dibuje una ventana con un único control que muestre el mensaje "Quiero aprobar Programación".

Ejercicio 9

Implemente una clase *Examen* para modelar el enunciado completo de un examen como, reutilizando la clase *Ejercicio*. La clase *Examen* debe tener: b) Un constructor que reciba el nombre de la materia y la fecha del examen, y métodos para consultar ambos datos. c) Un método para agregar un *Ejercicio* al *Examen*. d) Un método para consultar los datos de un *Ejercicio*. e) Un método *CalcularCalificacion* que reciba un *vector<int>* con las notas de un alumno en cada

ejercicio, y un *bool* indicando si el alumno era libre, y retorne su nota, calculada como porcentaje sobre la suma de los puntajes máximos de todos los ejercicios que le correspondan según su condición (esta suma no siempre es 100).

```
class Ejercicio {
    ...
public:
    Ejercicio(int puntaje, string enunciado, bool solo_libres);
    string VerEnunciado();
    int VerPuntajeMaximo();
    bool EsSoloParaLibres();
};

int main() { // posible programa cliente a modo de ejemplo
    Examen final_poo("POO", "09/08/2016");
    Ejercicio ej1(25, "Un archivo...", false), ej2(30, "Diseñe una clase...", false);
    Ejercicio ej3(25, "Escriba una func...", false), ej4(20, "Explique...", false),
    ej5(20, "Escriba...", true);
    final_poo.agregar(ej1); final_poo.agregar(ej2);
    final_poo.agregar(ej3); final_poo.agregar(ej4); final_poo.agregar(ej5);

    cout << final_poo.VerMateria() << " - " << final_poo.VerFecha() << endl;
    for(int i=0; i<final_poo.CantEjercicios(); i++)
        cout << "Ejercicio " << i+1 << ": " << final_poo.VerEjercicio(i).VerEnunciado() << endl;

    cout << "Ingrese los puntajes de los 5 ejercicios: ";
    vector<int> notas_de_un_alumno(5);
    for(int i=0; i<5; i++) cin >> notas_de_un_alumno[i];
    bool es_libre; cout << "Es libre? "; cin >> es_libre;
    cout << "Nota final: " << final_poo.CalcularNota(notas_de_un_alumno, es_libre);
}
```