UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS
DEPARTAMENTO DE INFORMÁTICA

Ingeniería en Informática

PROGRAMACIÓN ORIENTADA A OBJETOS

Ingeniería en Inteligencia Artificial

PROGRAMACIÓN II

UNIDAD 3
Relaciones Entre Clases

Guía de Trabajos Prácticos

EJERCICIOS BÁSICOS

Ejercicio 1

a) Diseñe una clase `Persona` que contenga los siguientes atributos: apellido y nombre, y fecha de nacimiento. La clase debe poseer, además, un método `CalcularEdad(...)` que permita obtener la edad actual de la persona en base a la fecha de nacimiento y la fecha actual.

b) Implemente una clase Alumno para contener la siguiente información de un alumno: apellido y nombre, año de nacimiento, promedio y cantidad de materias aprobadas. La clase debe poseer, además dos métodos adicionales: `AgregarMateria(...)` para actualizar el promedio y la cantidad de materias cada vez que el alumno rinde una nueva materia; y `CalcularMeritoAcademico(...)` que retorne el mérito académico del alumno, el cual se calcula como el producto entre el promedio y la cantidad de materias aprobadas.

c) Cree, también, una clase `Docente` para modelar un docente a partir de la siguiente información: apellido y nombre, año de nacimiento y fecha de ingreso. La clase debe poseer, además, un método `CalcularAntiguedad(...)` que calcule la antigüedad del docente en base a su fecha de ingreso y la fecha actual.

Notas: Para b) y c) proponga una jerarquía de clases adecuada para evitar repetir atributos. Implemente todos los constructores y los métodos adicionales que considere adecuados.

d) Codifique un programa cliente que cree instancias de `Alumno` y `Docente` y utilice sus métodos para verificar su correcto funcionamiento.

e) Finalmente responda:

- Puede crearse un objeto de tipo persona? ¿Para qué podría servir?
- ¿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 alumnos. Implemente un método `AgregarAlumno(...)` que permita agregar un alumno al curso (si es que no se llegó al cupo), 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 ax^n a partir de un coeficiente (*a*) y un exponente (*n*), 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, complete la clase `RectaExplicita` para definir la ecuación *explícita* de la recta a partir de dos puntos: $y = mx + b$. La declaración de dicha clase se muestra en el recuadro siguiente:

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

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).

Ayudas:

- Las ecuaciones para obtener m y b a partir de dos puntos P y Q son:

$$m = (Qy - Py)/(Qx - Px) \qquad b = Py - mPx$$

- La función `string to_string(float)` le permitirá convertir los coeficientes a `string`s; y además es posible concatenar `string`s con el operador + (`s=s1+s2`).
- Al comparar flotantes en `pertenece`, no debe utilizar `==`, sino preguntar de alguna otra forma si son muy parecidos en lugar de exactamente iguales (los cálculos con `floats` tienen errores *de truncamiento/redondeo* menores a `1e-6`).

Ejercicio 6

Una fábrica de tanques los hace de forma de cilindro y de esfera. En ambos envases debemos rotular el volumen en m^3 y el peso en kg. Modele una clase base `Tanque` con un constructor que reciba el peso, un método para consultarlo, y un método virtual para calcular su 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* * *altura*, donde el área de la base se calcula como π * radio^2; y otra clase hija `Esfera` con el radio como argumento adicional, cuya fórmula de volumen es: $\frac{4}{3}$ * π * *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 7

a) Complete la clase `RectaGeneral` para representar una recta mediante su ecuación *general*: $Ax + By + C = 0$, a partir de dos puntos. La definición de la clase se muestra en el siguiente recuadro:

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

b) Diseñe un árbol de herencia que incluya una clase `Recta`, y dos clases herederas llamadas `RectaExplicita` y `RectaGeneral` (modifique las versiones ya implementadas en este ejercicio y el anterior). **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 de la ecuación general son:

$$a = Qy - Py \qquad b = Px - Qx \qquad c = -aPx - bPy$$

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 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 indique 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:
    Instrumento(string nombre) : m_nombre(nombre) { }
    string VerNombre() const { return m_nombre; }
    virtual string Sonido() const;
    virtual ~Instrumento() { }
private:
    string m_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

Nota: el método `Sonido` de `Instrumento` debe retornar el sonido que haría el instrumento en su estado actual. En el piano, será la suma de las teclas que estén apretadas.

d) Utilizando la siguiente función:

```
void mostrarEnPantalla(const Instrumento *inst) {
    cout << "El " << inst->VerNombre() << " suena: " << inst->Sonido() << endl;
}
```

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 1

Se desea gestionar la venta de entradas de un teatro. Una sala de teatro está compuesta por butacas y cada sala puede tener diferente número de butacas. **a)** Cada butaca tiene un tipo (cadena), un precio y un estado (si está libre u ocupada). Codifique una clase `Butaca` que tenga un constructor para cargar tipo y precio; y métodos para modificar o consultar sus datos, entre otras funcionalidades que considere necesarias. **b)** Defina una clase `Sala` que, reutilizando la clase `Butaca`, guarde el nombre de la sala y la lista de butacas. La misma deberá tener:

- Un constructor que reciba el nombre de la sala.
- Un método que permita agregar varias butacas de un mismo tipo y precio a la sala.
- Un método para registrar la compra de una entrada recibiendo el *número* de butaca. El método debe verificar que la butaca esté libre y retornar `true` si la compra se puede hacer, o `false` si ya estaba ocupada.
  - Nota: asuma que el *número* se determina por el orden en que se agregó esa butaca a la sala
- Un método para obtener la recaudación total de una función
- Un método para obtener el porcentaje de ocupación de la sala

Ejercicio 2

a) Implemente una clase `Rueda` que en su constructor reciba el rodado de la misma (*Rod*, tamaño en pulgadas), y tenga un método para obtener su perímetro en *cm* ($Per = Rod * \pi * 2.54$). **b)** Diseñe una clase `Transmision` para modelar la transmisión de una bicicleta con cambios. La clase debe recibir en su constructor un vector con los tamaños de los piñones (los tamaños son enteros, se miden por cantidad de dientes) y un entero con el tamaño de la única corona. La clase debe además guardar en su estado el cambio (el piñón) seleccionado (que al iniciar será el 1ro), y ofrecer métodos para subir y bajar un cambio, para consultar el cambio actual, y para calcular la "relación" (Rel = tamaño de la corona divido tamaño del piñón seleccionado). **c)** Modele una clase `Bicicleta` reutilizando las clases `Transmision` y `Rueda` mediante la relación que considere más adecuada. La clase debe tener un método que reciba la velocidad de pedaleo (VP, vueltas de pedal por minuto) y calcule a qué velocidad avanza la bicicleta (la fórmula es $VP * Rel * Per * 0.0006$, y da en *km/h*).

Ejercicio 3

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

Implemente una clase `Examen` para modelar el enunciado completo de un exámen, reutilizando la clase `Ejercicio`. La clase `Examen` debe tener:

- Un constructor que reciba el nombre de la materia y la fecha del examen, y métodos para consultar ambos datos.
- Un método para agregar un ejercicio al examen.
- un método para consultar los datos de un ejercicio.
- 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. El método debe retornar 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).

Posible programa cliente a modo de ejemplo:

```
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),
              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);
}
```

Ejercicio 4

Una empresa de correos ofrece los siguientes tipos de envío: carta berreta y carta supermegaplus. El precio de la carta berreta es de $55 (hasta 20 gramos), $90 (hasta 150 gramos) o $125 (más de 150 gramos). El precio de la carta supermegaplus es de $650 (hasta 150 gramos) más un recargo de $165 si se excede de este peso. La carta supermegaplus permite contratar el servicio opcional ”aviso de recibo” que tiene un costo adicional de $195. **a)** Programe un `struct Dirección` que tenga como datos el nombre, dirección y código postal, y una clase `Carta` que tenga como datos el peso de la pieza, la dirección del remitente y la del destinatario. Agregue constructores y métodos que considere necesarios. **b)** Utilizando polimorfismo, programe las clases `CartaBerreta` y `CartaSuperMegaPlus`, con un método `calcularPrecio()` que devuelva el precio del envío. **c)** Programe una función libre `calcularPrecioTotal(...)` que reciba como parámetro un vector de cartas y devuelva el importe total correspodiente a las mismas. Si el precio total es mayor a $5000 se aplicará una bonificación del 10%.

Ejercicio 5

Una empresa de turismo ofrece paquetes de vacaciones utilizando una combinación de distintos medios de transporte por viaje: barco, tren y colectivo. Escriba un programa C++ en donde se declare una clase `Transporte` para modelar un tramo del recorrido que tendrá como atributos la distancia, costo y duración. Además declare otras tres clases: `TransporteBarco`, `TransporteTren` y `TransporteColectivo`, que deben servir para representar un viaje en cada medio de transporte. Puede agregar otros atributos si lo considera necesario. Las clases deben tener métodos para consultar la distancia, la duración y el costo de ese trayecto. Los constructores reciben como datos cual es la distancia a recorrer en kilómetros. La duración en horas y el costo se calcula a partir de la distancia de la siguiente manera: • Barco: duración: 4+distancia/30 costo: 20+distancia*10 • Tren: duración: 1+distancia/120 costo: distancia*6 • Colectivo: duración: distancia/80 costo: 2+distancia*3 Para calcular el costo total y la duración de un paquete turístico, deberá crear una función independiente llamada `viaje(...)`, que reciba como parámetro un vector con todos recorridos del viaje. Cree un breve programa cliente para probar la función.

Ejercicio 6

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:

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

Ejercicio 7

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

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 suponemos ya 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”.