Diseño y Programación Orientados a Objetos

1er. Semestre 2025

Certamen II ELO-329 Diseño y Programación Orientada a Objetos (1s2025)

En este certamen usted no podrá hacer preguntas. Si algo no está claro, indíquelo en su respuesta, haga una suposición razonable y resuelva conforme a ella.

Segunda parte, **con apuntes**.

Segunda pregunta (35 pts.)

Sistema de pagos Móviles

Se desea implementar en C++ un sistema que simule una aplicación de pagos móviles. En este sistema, los usuarios pueden transferirse dinero entre sí, considerando distintos tipos de cuentas con políticas de comisión diferenciadas. Toda la lógica debe ser gestionada por una clase central llamada `Banco`.

A continuación se detallan las etapas que debe desarrollar.

2.1 Clase base `Usuario` (10pts)

Cree una clase abstracta `Usuario` con las siguientes características:

- Atributos protegidos:
  - `string nombre`
  - `double saldo`
- Atributo estático:
  - `static int totalTransaccionesExitosas`
    (lleva el conteo total de transacciones exitosas realizadas)
- Métodos públicos:
  - Constructor que reciba el nombre y el saldo inicial
  - Métodos `getNombre() const` y `getSaldo() const`
  - Método `static int getTotalTransaccionesExitosas()` (accesor para el atributo estático)
  - Método virtual puro `double calcularComision(double monto) const`
  - Método `bool enviarDinero(Usuario* receptor, double monto)`
    El método debe:
    - Verificar que emisor y receptor no sean el mismo objeto (**Hint** puede utilizar `this`)
    - Calcular la comisión usando `calcularComision`
    - Verificar que el emisor tenga saldo suficiente (monto + comisión)

- Descontar el saldo al emisor y sumarlo al saldo del receptor
- Descontar del saldo del emisor la comisión calculada
- Aumentar el contador de transacciones exitosas si fue válida
- Retornar `true` si fue exitosa, `false` en caso contrario

R:

```
/*
usuario.h
P2.1 Clase Usuario (10pts)

- Atributo protegido (incluyendo static) (3pts)
- Métodos constructor, accesores (2pts)
- Método virtual puro (2pts)
- Método enviarDinero (3pts)
*/

#include <string>
using namespace std;

class Usuario {
protected:
    string nombre;
    double saldo;
    static int totalTransaccionesExitosas;

public:
    Usuario(string nombre, double saldo);
    virtual ~Usuario() {}

    string getNombre() const;
    double getSaldo() const;
    static int getTotalTransaccionesExitosas();

    virtual double calcularComision(double monto) const = 0;

    bool enviarDinero(Usuario* receptor, double monto);
};
```

```
// usuario.cpp

#include "usuario.h"

int Usuario::totalTransaccionesExitosas = 0;

Usuario::Usuario(string nombre, double saldo) : nombre(nombre),
saldo(saldo) {}

string Usuario::getNombre() const {
    return nombre;
```

```
}

double Usuario::getSaldo() const {
    return saldo;
}

int Usuario::getTotalTransaccionesExitosas() {
    return totalTransaccionesExitosas;
}

bool Usuario::enviarDinero(Usuario* receptor, double monto) {
    if (this == receptor) return false;

    double comision = calcularComision(monto);
    if (saldo >= monto + comision) {
        saldo -= (monto + comision);
        receptor->saldo += monto;
        totalTransaccionesExitosas++;
        return true;
    }
    return false;
}
```

2.2 Clases derivadas `UsuarioNormal` y `UsuarioPremium` (5pts)

Cree dos clases que hereden de `Usuario`:

- `UsuarioNormal`: paga una comisión del 2% sobre el monto enviado. Dicha comisión se descuenta del saldo existente y debe ser considerado para determinar si hay saldo suficiente.
- `UsuarioPremium`: no paga comisión.

Ambas deben implementar el método `calcularComision()` de forma adecuada.

R:

```
/*
usuario.h (continuación)
P2.2 Clases UsuarioNormal y UsuarioPremium (5pts)

Pueden ir en archivos separados o en el mismo archivo que Usuario (por su
dependencia).

- Heredan de Usuario (1pt)
- Implementan el método calcularComision (2pts, 1pt cada una)
- Constructor que recibe nombre y saldo (2pts, 1pt cada una)
*/

class UsuarioNormal : public Usuario {
public:
    UsuarioNormal(string nombre, double saldo);
```

```
    double calcularComision(double monto) const;
};

class UsuarioPremium : public Usuario {
public:
    UsuarioPremium(string nombre, double saldo);
    double calcularComision(double monto) const;
};
```

```
// usuario.cpp

// UsuarioNormal
UsuarioNormal::UsuarioNormal(string nombre, double saldo)
    : Usuario(nombre, saldo) {}

double UsuarioNormal::calcularComision(double monto) const {
    return monto * 0.02;
}

// UsuarioPremium
UsuarioPremium::UsuarioPremium(string nombre, double saldo)
    : Usuario(nombre, saldo) {}

double UsuarioPremium::calcularComision(double) const {
    return 0.0;
}
```

2.3 Clase `Banco` (10pts)

Implemente una clase `Banco` que gestione todos los usuarios y las transferencias.

- Atributos:
    - Un vector de punteros a `Usuario` para registrar los usuarios
    - Un vector de `string` para guardar el historial de transacciones exitosas
- Métodos:
    - `void agregarUsuario(Usuario* usuario)`.
    - `Usuario* buscarUsuario(string nombre) const` método privado que busca un usuario registrado en el banco por su nombre. Recorre el vector de usuarios y compara el nombre de cada uno usando el método getNombre(). Si encuentra coincidencia, retorna un puntero al objeto correspondiente. Si no lo encuentra, retorna `nullptr`.
    - `bool transferir(string nombreEmisor, string nombreReceptor, double monto)`
        - Busca ambos usuarios por nombre
        - Intenta realizar la transferencia usando el método `enviarDinero` desde el emisor. Una transferencia se realizará de manera exitosa **solo** si existe saldo suficiente para el monto

a transferir + la comisión en caso qu aplique.
- En caso exitoso, agrega un mensaje al historial: `"Ana envió $100 a Beto"`
- Imprime en consola si fue exitosa o no

- `void mostrarSaldos() const` Recorre el vector de usuarios y muestra su nombre y saldo. Ej.

```
--- Saldos ---
Ana: $1098
Beto: $900
```

- `void mostrarHistorial() const` Recorre el historial de transacciones y muestra cada entrada. Ej.

```
--- Historial ---
Ana envió $100 a Beto
Beto envió $200 a Ana
```

R:

```
/*
banco.h
P2.3 Clase Banco (10pts)

No es obligatorio constructor. Con el constructor por omisión es
suficiente. Usuarios son entregados luego.
Tampoco es obligatorio destructor, siempre y cuando luego se liberen los
usuarios MANUALMENTE.

- Atributos privados (2pts)
- Metodo privado buscarUsuario (2pts)
- Método agregar usuario (1pt)
- Método transferir (3pts)
- Métodos mostrarSaldos y mostrarHistorial (2pts)
*/

#include <vector>
#include <string>
#include "usuario.h"
using namespace std;

class Banco {
private:
    vector<Usuario*> usuarios;
    vector<string> historialTransacciones;

    Usuario* buscarUsuario(string nombre) const;
```

```
public:
    ~Banco();
    void agregarUsuario(Usuario* usuario);
    bool transferir(string emisor, string receptor, double monto);
    void mostrarSaldos() const;
    void mostrarHistorial() const;
};
```

```
// banco.cpp

#include "banco.h"
#include <iostream>

Banco::~Banco() {
    for (int i = 0; i < usuarios.size(); i++) {
        delete usuarios[i];
    }
}

void Banco::agregarUsuario(Usuario* usuario) {
    usuarios.push_back(usuario);
}

Usuario* Banco::buscarUsuario(string nombre) const {
    for(int i = 0; i < usuarios.size(); i++){
        if (usuarios[i]->getNombre() == nombre)
            return usuarios[i];
    }
    return nullptr;
}

bool Banco::transferir(string emisorNombre, string receptorNombre, double
monto) {
    Usuario* emisor = buscarUsuario(emisorNombre);
    Usuario* receptor = buscarUsuario(receptorNombre);

    if (!emisor || !receptor) return false;

    cout << "Intentando enviar $" << monto << " de " << emisorNombre << " a
" << receptorNombre << "..." << endl;

    if (emisor->enviarDinero(receptor, monto)) {
        historialTransacciones.push_back(emisorNombre + " envió $" +
to_string(int(monto)) + " a " + receptorNombre);
        cout << "Transacción exitosa" << endl;
        return true;
    } else {
        cout << "Saldo insuficiente o transacción inválida" << endl;
        return false;
```

```
    }
}

void Banco::mostrarSaldos() const {
    cout << "--- Saldos ---" << endl;
    for(int i = 0; i < usuarios.size(); i++){
        cout << usuarios[i]->getNombre() << ": $" << usuarios[i]-
>getSaldo() << endl;
    }
}

void Banco::mostrarHistorial() const {
    cout << "--- Historial ---" << endl;
    for(int i = 0; i < historialTransacciones.size(); i++){
        cout << historialTransacciones[i] << endl;
    }
}
```

2.4 Simulación en `main()` (10pts)

En la función principal:

1. Cree un objeto `Banco`
2. Cree dos usuarios mediante punteros a `Usuario`:
   - `"Ana"` de tipo `UsuarioNormal` con saldo `$1000`
   - `"Beto"` de tipo `UsuarioPremium` con saldo `$1000`
3. Agréguelos al banco con `agregarUsuario()`
4. Realice las siguientes transferencias:
   - Ana → Beto por $100
   - Beto → Ana por $5000 (debe fallar)
   - Beto → Ana por $200
5. Finalmente, imprima:
   - Saldos finales de los usuarios
   - Historial de transacciones exitosas
   - Número total de transacciones exitosas usando el método estático

---

Ejemplo de salida esperada:

```
Intentando enviar $100 de Ana a Beto...
Transacción exitosa
Intentando enviar $5000 de Beto a Ana...
Saldo insuficiente o transacción inválida
Intentando enviar $200 de Beto a Ana...
Transacción exitosa
--- Saldos ---
Ana: $1098
Beto: $900
--- Historial ---
```

```
Ana envió $100 a Beto
Beto envió $200 a Ana
Total de transacciones exitosas: 2
```

R:

```
/*
main.cpp
P2.4 Simulación main.cpp (10pts)


- Crear un objeto Banco (1pt)
- Crear dos usuarios, uno normal y otro premium (1pt)
- Agregar los usuarios al banco (1pt)
- Realizar transferencias entre los usuarios (3pts)
- Mostrar los saldos de los usuarios (1pts)
- Mostrar el historial de transacciones (1pt)
- Mostrar el total de transacciones exitosas (1pt)
- Liberar memoria, puede ser en el destructor de Banco o manualmente al
final del main dependiendo de la implementación (1pt)
*/


#include <iostream>
#include "banco.h"
using namespace std;

int main() {
    Banco banco;

    Usuario* ana = new UsuarioNormal("Ana", 1000);
    Usuario* beto = new UsuarioPremium("Beto", 1000);

    banco.agregarUsuario(ana);
    banco.agregarUsuario(beto);

    banco.transferir("Ana", "Beto", 100);
    banco.transferir("Beto", "Ana", 5000);
    banco.transferir("Beto", "Ana", 200);

    banco.mostrarSaldos();
    banco.mostrarHistorial();

    cout << "Total de transacciones exitosas: " <<
Usuario::getTotalTransaccionesExitosas() << endl;

    // Liberar memoria o haber implementado un destructor en Banco

    return 0;
}
```

- **¿Qué subir a AULA?** Cree un archivo `P2.tar` (`.zip` o `.rar`) con todas sus clases (archivos `.h` y `.cpp`) y súbalo.