Actividad Evaluable 3

Descripción

PROGRAMA	**Ingeniería de sistema**
ASIGNATURA	**IS0287 - PROGRAMACIÓN INTERMEDIA**
Carácter	Individual
Nombre del Estudiante	
Código estudiante	

Enunciado:

Taller de Programación Intermedia: Arquitectura de Objetos Seguros

Tema: Encapsulamiento, Sobrecarga y Gestión de Memoria

1. Contexto del Problema

Una entidad financiera corporativa requiere actualizar su núcleo bancario. El sistema anterior (estructurado) ha presentado fallos de seguridad donde el saldo de los créditos era modificado arbitrariamente por módulos externos.

Se le ha contratado para diseñar el módulo `CreditoCorporativo`. El objetivo no es solo almacenar datos, sino garantizar la **integridad transaccional** del objeto mediante un encapsulamiento estricto y demostrar el comportamiento de la memoria al manipular estos objetos entre diferentes gestores.

2. Requerimientos Técnicos

Deberá implementar una solución utilizando el paradigma de Orientación a Objetos (Java o C#) que cumpla con las siguientes especificaciones:

A. La Clase `CreditoCorporativo` (El Objeto Blindado)

Esta clase debe representar la entidad financiera y proteger sus datos a toda costa.

1. **Atributos:**
    - `id` (String): Identificador inmutable.
    - `montoPrestado` (double): El capital original.
    - `saldoPendiente` (double): Cuánto falta por pagar. **Debe ser privado.**
    - `tasaInteres` (double): Porcentaje de interés.
    - `esRiesgoso` (boolean): Bandera de estado.
2. **Encapsulamiento y Lógica de Negocio (Setters/Getters Avanzados):**
    - No se permite acceso directo a ningún atributo.
    - **Regla de Integridad:** El `saldoPendiente` nunca puede ser negativo. Si una operación intenta reducirlo por debajo de cero, la operación debe ser rechazada y el saldo no debe cambiar.
    - **Regla de Riesgo:** El atributo `esRiesgoso` se calcula automáticamente (es de solo lectura). Un crédito es "riesgoso" si el `saldoPendiente` supera el 120% del `montoPrestado` (debido a intereses acumulados).
3. **Constructores (Sobrecarga):**
    - *Constructor 1 (Estándar):* Recibe ID, monto y tasa. El saldo inicial es igual al monto.
    - *Constructor 2 (Refinanciación):* Recibe ID, monto, tasa y un `saldoPendiente` inicial (para casos de compra de cartera).
    - *Constructor 3 (VIP):* Recibe solo ID y monto. La tasa se establece automáticamente al 1.5%.

B. Comportamiento y Métodos

1. **`abonarCapital(double cantidad):`** * Método público para reducir el saldo.
    - Debe validar que la cantidad sea positiva.
2. **`capitalizarIntereses():`**
    - Calcula el interés basado en la tasa y lo suma al `saldoPendiente`.
    - **Desafío:** Si al sumar el interés el crédito se vuelve "riesgoso" (ver Regla de Riesgo), el sistema debe imprimir una alerta de seguridad en consola inmediatamente.

C. Clase `GestorDeCobranza` (Prueba de Referencia)

Esta clase representa un agente externo que intenta manipular el crédito.

1. Método `intentarLiquidacionMaliciosa(CreditoCorporativo c)`:
    - Este método debe intentar acceder directamente a `c.saldoPendiente` para ponerlo en 0. **El estudiante debe demostrar que esto genera un error de compilación.**
    - Luego, debe usar los métodos públicos para intentar abonar una cantidad negativa.
2. Método `aplicarAlivioFinanciero(CreditoCorporativo c)`:
    - Recibe un objeto crédito.
    - Reduce su saldo en un 50% legítimamente usando los métodos públicos.
    - **Objetivo:** Demostrar que al modificar `c` dentro de este método, el objeto original en el `main` se ve afectado (Paso por Referencia).

3. Escenario de Prueba (El `main`)

Para aprobar la actividad, debe escribir una clase principal `SistemaBancario` que ejecute la siguiente secuencia exacta:

1. **Instanciación:** Crear un crédito (ID "CORP-99") de $10,000 al 5% de interés usando el Constructor 1.
2. **Simulación de Tiempo:** Ejecutar un ciclo que llame a `capitalizarIntereses()` 5 veces.
3. **Auditoría de Estado:** Imprimir si el crédito es riesgoso o no.
4. **Prueba de Referencia:**
    - Imprimir saldo actual.
    - Instanciar un `GestorDeCobranza`.
    - Llamar a `gestor.aplicarAlivioFinanciero(miCredito)`.
    - Imprimir nuevamente el saldo en el `main`.
    - *Pregunta reflexiva:* ¿Por qué cambió el saldo si el método `aplicarAlivioFinanciero` no retornó nada?

4. Criterios de Evaluación

Concepto	Criterio de Éxito	Puntos
Encapsulamiento	Los atributos son `private`. No es posible asignar saldos negativos ni acceder a variables internas directamente.	30%
Sobrecarga	Los 3 constructores funcionan correctamente y inicializan el objeto en estados válidos.	20%
Lógica de Negocio	El cálculo de `esRiesgoso` es dinámico y correcto. La validación de abonos funciona.	25%
Manejo de Memoria	El estudiante explica correctamente en comentarios por qué el objeto cambió de estado tras pasar por el `GestorDeCobranza`.	25%