Java Lab 2022 - TP N°1

Nos pidieron modelar un sistema bancario donde vamos a tener cuentas, que podrán ser de tipo **Caja de Ahorro** o **Cuenta Corriente**.
El sistema debe permitir realizar operaciones como depositar y retirar dinero (a una cuenta), así como también realizar transferencias entre cuentas.
Los requerimientos son los siguientes:

1. Una cuenta (sin importar su tipo) debe contar con las siguientes propiedades: saldo, nroCuenta, titular y si está habilitada. Además, las cuentas corrientes poseen una propiedad adicional denominada saldoDescubierto, que representa un saldo adicional al principal.

2. Las cuentas deben permitir realizar las operaciones que se detallan a continuación:
    a. Al retirar dinero de una caja de ahorro se deberá validar si su saldo es suficiente, caso contrario no se podrá realizar la operación. Si se intenta retirar saldo de una cuenta corriente se deberá tener en cuenta el propio saldo de la cuenta más el saldo descubierto (sólo si dicha suma resulta mayor o igual que la cantidad solicitada se podrá hacer el retiro). Si se utiliza parte del descubierto se deberá mostrar un mensaje para notificarlo.
    b. Las transferencias entre cuentas de distinto tipo y con distintos titulares tendrán un cargo adicional que se descontará de la cuenta de origen. Dicho cargo adicional será del 1,5% si la cuenta origen es de tipo Caja de Ahorro y 3% si es de tipo Cuenta Corriente (en estos casos se deberá mostrar un mensaje con el detalle del cargo aplicado y el tipo y número de cuenta sobre el que se aplicó).
    c. En todas las operaciones se deberá validar que la cuenta y/o cuentas involucradas estén habilitadas, caso contrario no se realizará la operación. Nota: Una cuenta está habilitada si su atributo habilitada es verdadero.
    d. En todas las operaciones se deberá actualizar el estado de el/los atributos de la/s cuentas que se hayan modificado, es decir, las cuentas deben quedar siempre con un estado correcto luego de una operación (si se hace una transferencia de una cuenta a otra debe debitarse el saldo de la cuenta origen y sumarse al saldo de la cuenta destino).
    e. En todos los casos en que las operaciones se realicen en forma correcta, mostrar un mensaje indicándolo.
    f. Implementar el manejo de errores que crean conveniente: por ejemplo, ¿qué sucede si intento debitar saldo en una cuenta y no posee el suficiente?

3. Además, el banco desea ofrecer un préstamo bancario por un monto de 10.000 pesos a los clientes que cumplan ciertos requisitos:
    a. Para ello, desea conocer la lista de titulares (nombres) de aquellos clientes que posean una cuenta con un saldo igual o superior a 10.000 pesos (sin importar si la cuenta es caja de ahorro o cuenta corriente, en cuyo caso el saldo debe considerar también el saldo descubierto). Obviamente, también es necesario que la cuenta esté habilitada para que el cliente aplique al préstamo.

       La lista con los nombres de los clientes debe generarse con todas sus letras en mayúsculas y no debe contener nombres repetidos (ya que tenemos la certeza que todos los clientes del banco tienen diferente nombre).

       Un resultado válido de ejemplo sería:

       "MARIA PEREZ"
       "RODRIGO ESTEVEZ"
       "CLARA ALCARAZ"
       "PEDRO ARIELI"

       Nota: Modelar el préstamo bancario está fuera del alcance del TP. Nuestro objetivo es crear un método ***obtenerTitularesAptosParaPrestamo(Lista cuentas)*** que permita, dada la lista de cuentas que recibe por parámetro, obtener los titulares que cumplan los requisitos mencionados.
    b. Por un problema de seguridad, el banco descubrió que las cuentas cuyo nroCuenta es par (sean del tipo que sean) pueden ser propensas a un hackeo si su saldo es mayor a 50.000 pesos y el titular de la cuenta tiene más de 15 caracteres (letras).

       Queremos saber si, actualmente, existe alguna cuenta que pueda ser hackeada, para lo que debemos implementar el método ***algunaCuentaPuedeSerHackeada(Lista cuentas)***, que recibe una lista de cuentas por parámetro y retorna un booleano (true o false) si alguna de las cuentas puede ser hackeada.
    c. **BONUS:** No utilizar bucles (for, while ni do-while) ni tampoco condicionales (if-else) para resolver los dos puntos anteriores, sino apoyarse en la API de Streams de Java.

4. Crear un programa **TestCuenta** que cree varias cuentas y realice operaciones con ellas (generar combinaciones de prueba que permitan probar las distintas casuísticas planteadas en los puntos anteriores). Cada vez que se realice una operación (depósito, retiro o transferencia) se deberán imprimir los datos de la/s cuenta/s involucrada/s en la misma para poder visualizar si ha habido cambios. Se valorará que se ilustren todos los casos posibles mediante ejemplos y que cada ejemplo se pueda ejecutar en forma independiente.