Universidad Simón Bolívar
Depto. de Computación y T.I.
Taller de Algoritmos y Estructuras III (CI-2693)
Septiembre Diciembre 2010

Proyecto #0
Programación Orientada a Objetos en Java

El objetivo de este proyecto es la implementación de un sistema de manejo de transacciones sobre cuentas bancarias con la finalidad de repasar los conceptos básicos de la programación orientada a objetos en Java. Nuestro banco, maneja un solo tipo de cuenta sobre las cuales se pueden hacer las siguientes transacciones: apertura de cuenta, depósito de un monto en una cuenta, retiro de un monto de una cuenta, transferencia de fondos de un monto determinado entre dos cuentas.

Para simplificar la interfaz con el usuario, toda las interacciones se harán a partir de comandos con operaciones. Los posibles comandos son:

1. Apertura de Cuenta:

   apt <numCuenta> <cedulaCliente> <montoApertura>
   - *<numCuenta>* debe ser un nuevo número de cuenta; es decir, que no exista en el banco una cuenta con ese número.
   - *<montoApertura>* debe ser un entero, no negativo

2. Depósito en Cuenta:

   dep <numCuenta> <monto>
   - *<numCuenta>* debe ser un número de cuenta existente
   - *<monto>* debe ser un entero mayor que 0

3. Retiro de Cuenta:

   ret <numCuenta> <monto>
   - *<numCuenta>* debe ser un número de cuenta existente
   - *<monto>* debe ser un entero mayor que 0 y menor o igual que el saldo de la cuenta *<numCuenta>*

4. Transferencia entre cuentas:

   trn <ctaOrigen> <ctaDestino> <monto>
   - *<ctaOrigen>* debe ser un número de cuenta existente
   - *<ctaDestino>* debe ser un número de cuenta existente
   - *<ctaOrigen> y <ctaDestino>* deben ser diferentes
   - *<monto>* debe ser un entero mayor que 0 y menor o igual que el saldo de la cuenta *<numCuenta>*

5. Estado de Cuenta:

   edc <numCuenta>
   - *<numCuenta>* debe ser un número de cuenta existente

Para cada una de estas operaciones deberá mostrarse por pantalla el string “Ok.” En caso de que la operación se haya podido realizar adecuadamente, o el mensaje error correspondiente en caso contrario. Los mensajes de error están definidos en la clase Status (suministrada por la cátedra)

Para verificar el funcionamiento de su implementación, la cátedra le proporcionará un programa de prueba (Main.java) y algunos archivos de prueba (transacciones de entrada, salida que debe reportar el programa de Prueba). Tenga en cuenta que la corrección de las corridas se realizará de forma automática, por lo que las salidas de su programa debe ser **exacta** a la que se pide.

A continuación la descripción (documentación y firmas) de las clases principales que deberá implementar:

Clase Banco

El banco es el encargado de almacenar y administrar las cuentas y las transacciones que se realizan sobre las mismas. El banco almacena un máximo de 1000 cuentas. Los métodos que debe ofrecer la clase Banco son:

```
// precondición: ninguna
// poscondición: se realiza (en caso de ser posible) la transacción t
// se agrega t al conjunto de transacciones del banco
// retorna: El status resultante de la ejecución de la transacion t
// Descripción: Función que permite despachar las transacciones
public Status ejecutarTransaccion(Transaccion t){. . .}

// precondición: ninguna
// poscondición: ninguna
// Realiza las verificaciones necesarias para que se realice la
// operación de apertura, descrita por la transacción t.
// En caso de falla construye y lanza una excepcion (Status).
void hacerApertura(Apertura t) throws Status {. . .}

// precondición: ninguna
// poscondición: ninguna
// Realiza las verificaciones necesarias para que se realice la
// operación de depósito, descrita por la transacción t.
// En caso de falla construye y lanza una excepción (Status).
void hacerDeposito(Deposito t) throws Status { . . .}

// precondición: ninguna
// poscondición: ninguna
// Realiza las verificaciones necesarias para que se realice la
// operación de retiro, descrita por la transacción t.
// En caso de falla construye y lanza una excepción (Status).
void hacerRetiro(Retiro t) throws Status {. . .}

// precondición: ninguna
// poscondición: ninguna
// Realiza las verificaciones necesarias para que se realice la
// operación de transferencia, descrita por la transacción t.
// En caso de falla construye y lanza una excepción (Status).
void hacerTransferencia(Transferencia t) throws Status {. . .}
```

```
// precondición: ninguna
// poscondición: ninguna
// Realiza las verificaciones necesarias para que se realice la
// operación de generación de estado de cuenta, descrita por la
// transacción t.
// En caso de falla construye y lanza una excepción (Status).
void hacerEstadoDeCuenta(EstadoDeCuenta t) throws Status {. . .}
```

Clase Cuenta

La cuentas poseen como atributos un identificador, un cliente y un saldo. También deberán almacenar las transacciones que se realizan sobre ellas. El número máximo de transacciones de una cuenta es de 100.000. Los métodos que debe ofrecer la clase cuenta son:

```
//Precondicion: c.saldo >= monto && Existe(c)
//Postcondicion: c.saldo = c.saldo – monto
//Retorna: void
public void retirar(int monto) {. . . }

//Precondicion: monto>0 && Existe(c)
//Postcondicion: c.saldo = c.saldo + monto
//Retorna: void
public void depositar(int monto) {. . .}

//Precondicion: Existe(c)
//Postcondicion:
//Retorna: un String con las operaciones de la cuenta c y el saldo actual
//en el formato requerido. Las operaciones se deben mostrarse en el orden
//que se realizaron, y estar precedidas por un entero que indica su
//índice seguido de un punto: Ejemplo "0. apt 1 17 200"
public String estadoDeCuenta() {. . .}

//Precondicion:  Existe(t)
//Postcondicion: c.transacciones = c.transacciones ∪ {t}
//Retorna: void
public void agregarTransaccion(Transaccion t) {. . . }

//Precondicion:  ninguna
//Postcondicion:
//Retorna: un entero con en identificador del número de cuenta
public int obtCuenta(){. . . }

//Precondicion:  ninguna
//Postcondicion:
//Retorna: un entero con el identificador (cédula) del cliente al que
pertenece la cuenta
public int obtCedula(){. . .}

//Precondicion:  ninguna
//Postcondicion:
//Retorna: un entero con el saldo de la cuenta
public int obtSaldo(){ . . .}
```

Clase Transacción:

Las Clase Transacción es una clase abstracta, de la que heredan cada unas de las transacciones particulares (Apertura, Deposito, Transferencia, EstadoDeCuenta). Debe proveer los siguientes métodos:

```
// Debe ser reescrita por cada Transacción en particular, quién le
// solicitará al Banco pBanco que realice la transacción particular
abstract public void ejecutar(Banco banco) throws Status;

// Debe ser reescrita por cada Transacción en particular, quién le
// creará un String con la información de la transacción (con el mismo
// formato que tiene un comando de transacción
abstract public String toString();
```

NOTAS:

- Hubo cambios entre el código que se trabajo en clase, y el nuevo que está en P0_base.zip. Deberán usar el nuevo código como base para su proyecto. En particular:
  - En la Clase Banco:
    - hacerApertura(Apertura t) realiza las verificaciones adecuadas para la apertura de una cuenta.
    - Incluimos la implementación de hacerEstadoDeCuenta(EstadoDeCuenta t)
  - En el Main:
    - Para el manejo de los estados de cuenta. Observe que el constructor de EstadoDeCuenta deberá recibir un segundo parámetro de tipo PrintStream, que indica el "stream" (flujo), que establece por donde será "impreso" el estado de cuenta

Sobre la entrega:

La entrega será a mas tardar el martes 11 de Mayo al inicio de la clase (1:30 pm)

Deberán entregar en un sobre manila sellado e identificado con el número del grupo y sus integrantes todo el código del proyecto impreso. Es necesario que el código esté debidamente documentado.

Deberán subir al aula virtual un archivo de nombre P0G<#grupo>.zip que contenga un directorio P0G<#grupo> que a su vez contenga todos los archivos .java de su proyecto