Sesión 13. Resolver examen

Antes de comenzar a codificar, se debe crear un diagrama de clase UML, en papel, que muestre el diseño.

Planteamiento del problema

Nuestra empresa es un proveedor de servicios de pago. Ofrece servicios de procesamiento de pagos para bancos y otras empresas relacionadas con los pagos. Necesitamos una aplicación simple para validar las transacciones de pago en línea recibidas por nuestros clientes. **Recibimos las transacciones del cliente en un fichero y, después de analizarlo, creamos otros dos archivos: uno con las transacciones válidas y otro con las inválidas.**

Todas las *transacciones válidas* se guardan en un fichero comprimido y las *inválidas* se almacenan en un fichero de texto sin comprimir diferente. Estos ficheros serán enviados de vuelta a nuestro cliente para su procesamiento posterior.

Para el propósito de nuestra aplicación, consideramos una transacción como una operación comercial que se paga con un medio de pago. En aras de la simplicidad, solo consideraremos las tarjetas de crédito y las cuentas corrientes como medios de pago. En consecuencia, tenemos dos tipos de transacciones: *transacciones con tarjeta de crédito* y *transacciones con cuenta corriente*.

Cada **Transacción** tiene esta información:

- la *fecha* de la operación (como un string, con el formato yyyy/mm/dd, ejemplo "2020/05/23").
- el *número* del medio de pago (número de tarjeta de crédito o número de cuenta) (un string).
- el importe de la operación (un double).
- una descripción de la operación (un string).

Una *transacción con tarjeta de crédito* (**CreditCardTransaction)** tiene esta información extra:

- la *fecha de vencimiento* de la tarjeta (no se puede usar para pagos una vez pasada esta fecha) (un string aaaa / mm / dd).
- la cantidad máxima permitida por esta tarjeta para una operación (un valor double)

Una *transacción con cuenta corriente* (**CurrentAccountTransaction)** tiene, como información particular, el *tipo* del cliente client: *Normal* (N) o *Premium* (P).

Formato de ficheros

Los datos de entrada vienen en un fichero de texto plano con extensión ".trx". Tiene dos tipos de líneas.

- Para transacciones con tarjeta de crédito:

```
cc <date> <card number> <due date> <max amount per trx> <amount> <description>
```

- Para transacciones de cuenta corriente:

```
acc <date> <account number> <client type> <amount> <description>
```

El separador entre campos es el carácter de punto y coma (";"). Lo siguiente es una muestra de este tipo de fichero.

```
cc;2018/05/03;5552468190471987;2018/12/03;500;400;Carrefour Travel
cc;2018/05/05;6011898652586869;2018/12/05;1000;12;Valentin's cafeteria
acc;2018/05/08;ES0310250124910021214445;N;150;Car repairing at Caris
acc;2018/05/08;ES0712452342121241551584;N;1000;Shopping at CFRD
```

Una vez que el proceso de validación ha finalizado, se deben crear dos ficheros nuevos

- El *fichero de transacciones válido* es un fichero comprimido con la extensión ".trx.gz" y el mismo formato que el fichero de entrada.
- El *fichero de transacciones no válidas* es un fichero de texto plano, con el mismo formato que el de entrada, pero con la extensión ".invalid.trx"

Tenga en cuenta que las transacciones en el fichero de entrada pueden no estar ordenadas, pero la aplicación tiene que producir ficheros con las transacciones ordenadas por *fecha y número* de forma ascendente.

Procesamiento de transacciones

El propósito del proceso de validación es discriminar las transacciones válidas de las inválidas. Para eso, la clase *TransactionValidator* mantiene dos listas diferentes: las válidas (irán al fichero ".trx.gz") y las inválidas (irán al fichero ".invalid.trx").

Una transacción se considera inválida si no pasa **todas las reglas de validación** de la aplicación para el tipo de transacción. Ten en cuenta que una transacción puede tener más de un error y el programa debe indicar todos los errores de cada transacción

Para cada fallo detectado, se debe generar un mensaje explicativo y adjuntarlo a la transacción. Por ejemplo:

"(cc) La cantidad 850.0 es mayor que el máximo: 500.0 para la tarjeta de crédito", o
"(acc) El número IBAN[1] es inválido: ES6524155245212844135465".

Operaciones de validación

Para *transacciones de cuenta corriente*, estas reglas de validación son de aplicación:

- Si el tipo de cliente es *Normal*, la cantidad debe ser <= 1.000 €. Si es *Premium*, no hay límite para la cantidad cargada contra la cuenta.
- El número de cuenta debe ser un IBAN válido. Eso evita errores tipográficos en caso de introducir de forma manual del número.

Para las *transacciones con tarjeta de crédito*, se deben cumplir estas otras reglas de validación:

- La fecha de la operación no puede ser posterior a la fecha de vencimiento de la tarjeta.
- El importe cobrado no puede ser mayor que el máximo por transacción para la tarjeta.
- El número de tarjeta debe ser válido para Lhun[2]

Notas:

- Para verificar si un número de cuenta es un IBAN válido, usa la clase IBAN provista con el esqueleto inicial (package *uo.mp.transaction.model.util*) de esta manera:

```
if ( IBAN.isValid( number ) ) …
```

- La fecha de la operación no puede ser posterior a la fecha de vencimiento de la tarjeta.

- Del mismo modo, para verificar si un número de tarjeta de crédito es válido, usa la clase Lhun proporcionada en (package *uo.mp.transaction.model.util*):

```
if ( Lhun.isValid( number ) ) …
```

- Para comparar dos fechas usa String.compareTo(…). Esto funciona por la forma en que se representan las fechas (aaaa / mm / dd).

```
2019/11/04 > 2019/11/03
```

[1] IBAN: International Bank Account Number
[2] https://en.wikipedia.org/wiki/Luhn_algorithm

Para cada una de las validaciones anteriores que no se cumplen, se debe adjuntar un mensaje de error a la transacción.

Manejo de errores

Durante el análisis del fichero de entrada, se debe tener en cuenta la posibilidad de errores en el fichero de entrada: *líneas en blanco, número incorrecto de campos o números no válidos*. Si ocurre algo de esto, la línea debe ignorarse y el error debe registrarse en el log.

No debería suceder que dos transacciones con el mismo sistema de pago puedan ocurrir en la misma fecha. En ese caso, todo el archivo de entrada se considera corrupto: se debe mostrar un mensaje de error al usuario (recuerde que solo la clase Main puede imprimir en la consola) y el programa debe detenerse.

En caso de errores del sistema o de programación, la ejecución debe detenerse, el usuario debe ser informado ("ha ocurrido un error interno") y los detalles del error enviados al log.

Tareas:

UML

- Dibuje un diagrama UML para mostrar el diseño de la solución. Esto debe entregarse al final del examen, pero la primera versión debe elaborarse al principio.
- Te dan un esqueleto con algunas clases que dan algunas pistas sobre el diseño. Por favor, asegúrate de comprender el diseño previsto y verifica tu UML

Desarrollo de la aplicación

Diseño

- Desarrolla la aplicación. Crea una jerarquía de clases adecuada para representar las transacciones. La transacción de clase base ya está dada y complete.
- Completa la clase *TransactionValidator* para que pueda contener transacciones válidas e inválidas (por separado).
- Complete el método *validate()* de la clase *TransactionValidator*. Tiene que llenar las listas de transacciones válidas e inválidas (haciendo uso de los métodos *validate*() y *hasFaults*() de cada transacción).
- Haz que la clase *TransactionValidator* pueda ordenar todas las transacciones en orden ascendente por fecha y número antes de la validación cuando se agregan las transacciones.
- Haz que la clase *TransactionProcessor* pueda cargar toda la información de un fichero especificado. Si alguna línea es incorrecta, debe ignorarse. Si una transacción es un número de fecha repetido, todo el fichero no es válido y el *usuario* debe ser informado ...
- Haz que la clase *TransactionProcessor* pueda guardar ambos tipos de transacciones en el fichero adecuado (ten en cuenta el nombre del fichero y la naturaleza del mismo).

Excepciones

- Cree una clase de excepción específica para indicar errores de aplicación / lógica.
- Usa/maneja las excepciones correctamente para tratar posibles problemas con el manejo de ficheros y otros tipos de errors.
- Completa la clase Main con las cláusulas catch de captura adecuadas. La clase Main es la única que puede imprimir información en la salida estándar.

Tests

- Añade Junit tests to comprobar el método *TransationValidator.validate()*.
- Añade Junit tests to comprobar el método *validate() de la clase* Transaction (casos positivos y negativos).
- Añade Junit tests para verificar el método TransactionLoader (...) .load (). Se proporciona un fichero con errores con el esqueleto.

Evaluación

Un programa que funcione NO significa un aprobado. Hay cinco aspectos principales que deben revisarse y **debe tener un mínimo** en cada uno:

- Diseño expresado como un diagrama de clase UML.
- Polimorfismo
- Manejo de excepciones
- Streams
- Calidad de las pruebas unitarias.

Si utilizas tus propias colecciones y clases de ordenación, se suma un punto extra a la calificación final.