Escuela de Ingeniería Informática

UNIVERSIDAD DE OVIEDO

TG2 - PRÁCTICAS

Metodología de la programación

Curso 2023-2024

Ejercicio: validación de transacciones

PayPal recibe fichero con transacciones bancarias de dos tipos (Tarjeta de crédito y cuenta corriente)

- Crea Lista de Transacciones

- Para transacciones con tarjeta de crédito:

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

- Para transacciones de cuenta corriente:

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

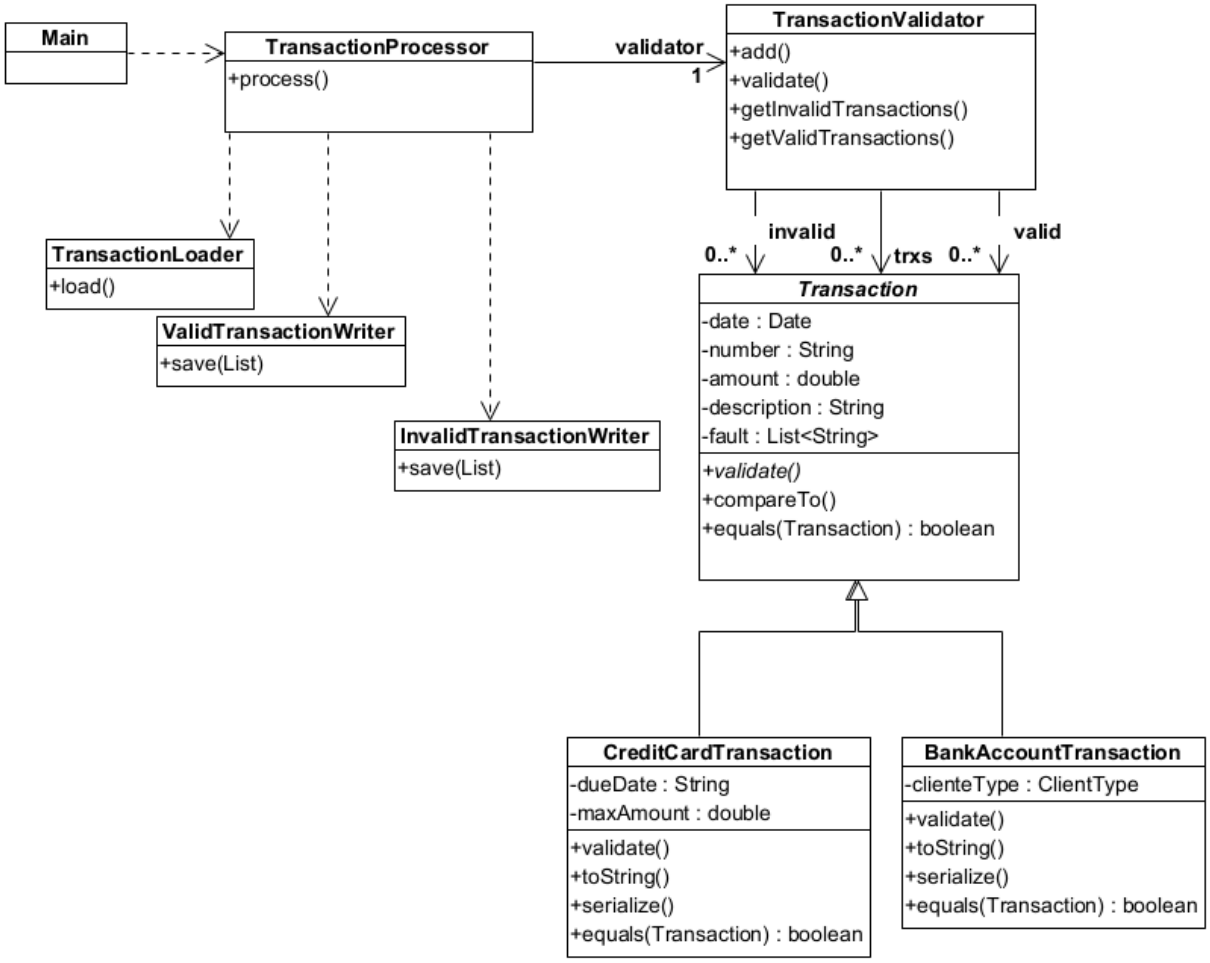
  El separador entre campos es el carácter de punto y coma (";"). Lo siguiente es una muestra de este tipo de fichero.

```
cc;2021/05/03;5552468190471987;2021/12/03;500;400;Carrefour Travel
cc;2021/04/05;6011898652586869;2021/12/05;1000;12;Valentin's cafeteria
acc;2021/04/08;ES0310250124910021214445;N;150;Car repairing at Caris
acc;2021/04/08;ES0712452342121241551584;N;1000;Shopping at CFRD
```

- Necesita Validarlas y crear dos listas
  - > Lista de Transacciones válidas
  - > Lista de Transacciones inválidas
- Finalmente deberá
  - Guardar a fichero comprimido las transacciones válidas ordenadas por fecha y número ascendente
  - Guardar a fichero de texto las transacciones inválidas ordenadas por fecha y número descendente

1 UML inicial

A partir del esqueleto y enunciado



2. Hacer que compile

- Revisar configure Build path y enlazar con paquete util
- Importar List de java.util en todas las clases
- Importar el resto que vaya surgiendo (empezar por clases má básicas)
- Crear clases y métodos en paquete adecuado
  - TransactionSerializer, TransactionParser..
- A veces puede haber que cambiar nombres de métodos para que encajen con paquetes de utilidad FileUtil (reaLines ..)
- Grabar de vez en cuando

3. Gestionar recogida básica de excepciones

- Recoger errores de programación y sistema
- Recoger errores de usuario que implican final de programa

¿Donde? En este caso la clase Main es la capa de interfaz

Método específico para su gestión

```
private void run() {
    try {
        process();
    } catch (RuntimeException e) {
        handleSystemError(e);
    } catch (TransactionException e) {
        handleUserError(e);
    }
}
```

O bien

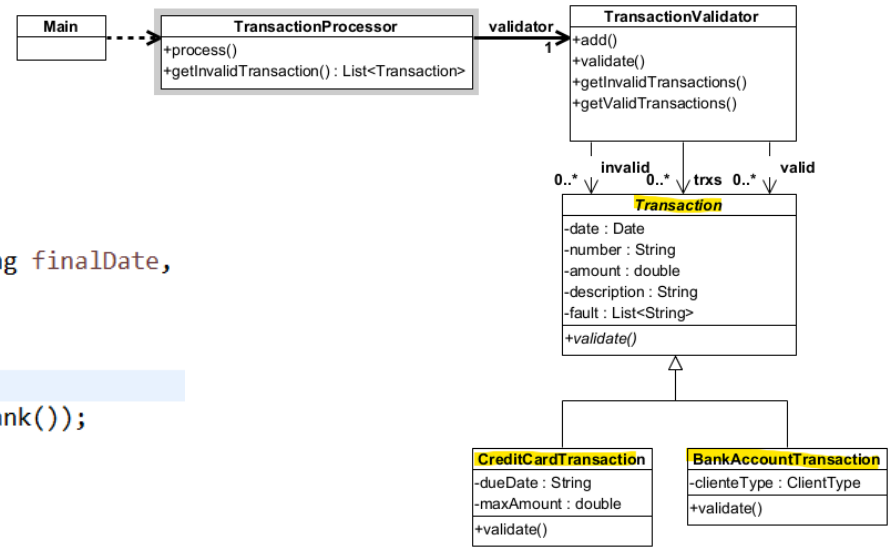
```
private void run() {
    try {
        process();
    } catch (RuntimeException e) {
        handleSystemError(e);
    } catch (FileNotFoundException | TransactionException e) {
        handleUserError(e);
    }
}
```

O bien

```
private void run() {
    try {
        process();
    } catch (RuntimeException e) {
        handleUserError(e);
    } catch (Exception e) {
        handleUserError(e);
    }
}
```

```
private void handleUserError(TransactionException e) {
    Console.println("APLICATION ERROR revise este error: " + e.getMessage());
}

private void handleSystemError(RuntimeException e) {
    Console.println("PROGRAMMING ERROR la aplicación finaliza debido a un error interno");
    Console.println(e.getMessage());
    Logger.log(e.getMessage());
    Logger.log(e);
}
```

4. Implementar clases del modelo



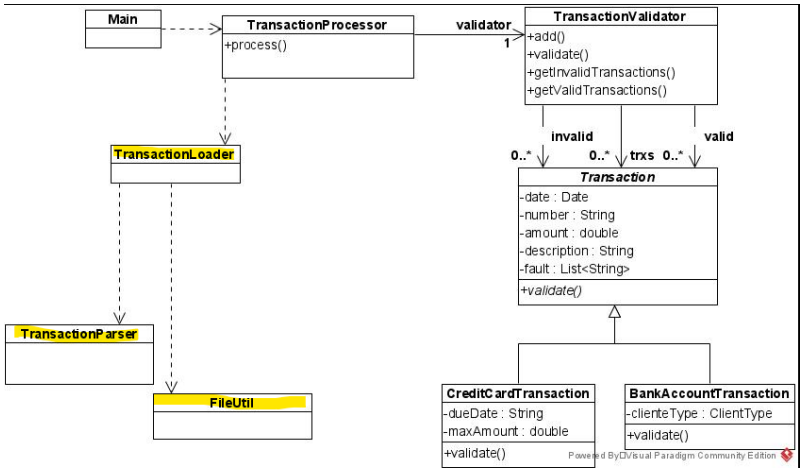

```java
public class CreditCardTransaction extends Transaction {
    private String finalDate;
    private double maxAmount;

    public CreditCardTransaction(String date, String number, String finalDate,
                                 double maxAmount, double amount,
                                 String description ) {
        super(date, number, amount, description);

        ArgumentsCheck.isTrue(finalDate!= null && !finalDate.isBlank());
        ArgumentsCheck.isTrue(maxAmount > 0);


        this.finalDate = finalDate;
        this.maxAmount = maxAmount;
    }
```

```java
public class CurrentAccountTransaction extends Transaction{
    private ClientType type;

    public CurrentAccountTransaction(String date, String number,
                                     ClientType type, double amount,
                                     String description) {
        super(date, number, amount, description);

        this.type = type;
    }
```

```java
public abstract class Transaction implements Comparable<Transaction> {

    private String date;
    private String number;
    private double amount;
    private String description;

    private List<String> validationFaults = new ArrayList<>();

    public Transaction(String date, String number, double amount,
            String description) {
        ArgumentsCheck.isNotEmpty(date);
        ArgumentsCheck.isNotEmpty(number);
        ArgumentsCheck.isTrue(amount > 0);
        ArgumentsCheck.isNotEmpty(description);
        this.date = date;
        this.number = number;
        this.amount = amount;
        this.description = description;
    }
```

5. Carga de fichero: TransactionLoader

- Se usa desde TransactionProcesor

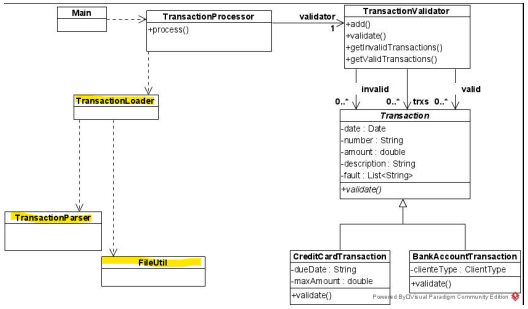



```
public class TransactionLoader {

    private String fileName;

    public TransactionLoader(String fileName) {
        ArgumentChecks.isNotEmpty(fileName);

        this.fileName = fileName;
    }

    public List<Transaction> load() throws TransactionException  {
        try {
            List<String> lines = new FileUtil().readLines( fileName );
            return new TransactionParser().parse( lines );
        } catch (FileNotFoundException e) {
            throw new TransactionException("El fichero de entrada no existe");
        }
    }

}
```

Transformamos FileNotFoundException que se puede producir al crear el flujo en TranssactionException
Añadimos el mensaje personalizado

5. Carga de fichero: FileUtil.readLines

- Se usa desde TransactionLoader



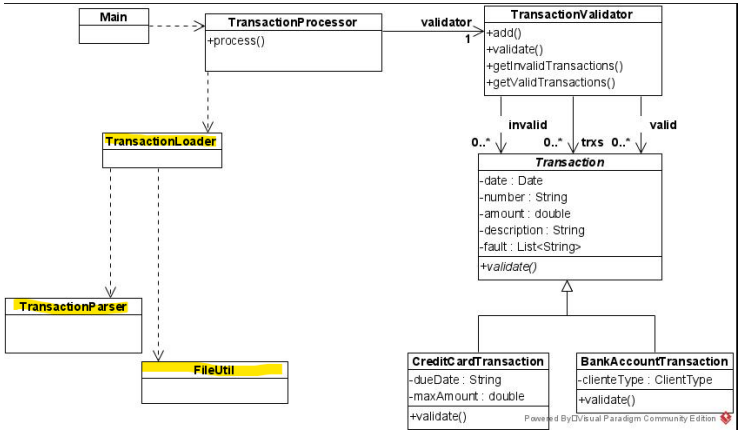

```
public class FileUtil {

    public List<String> readLines(String inFileName) throws FileNotFoundException  {
        ArgumentsCheck.isTrue(inFileName!= null && ! inFileName.isBlank() );
        List<String> res = new LinkedList<>();
        BufferedReader in = new BufferedReader(new FileReader(inFileName));
        try {
            try {
                String line;
                while ((line = in.readLine()) != null) {
                    res.add(line);
                }
            } finally {
                in.close();
            }
        } catch (IOException e) {
            throw new RuntimeException("Error de lectura en  " + inFileName);
        }
        return res;
    }
```

Recibe fichero con línea de texto.
Devuelve lista de líneas

Transformamos IOException que se puede producir en readLine en RuntimeException (considemamos error del sistema).
Añadimos el mensaje personalizado

5. Carga de fichero: TransactionParser.parse

Se realiza desde TransactionLoader




```java
public class TransactionParser {
    private int lineNumber = 1;
    public List<Transaction> parse(List<String> lines) {
        ArgumentChecks.isNotNull(lines);
        List<Transaction> transactions = new ArrayList<>();
        for (String line: lines) {
            try {
                checkIsBlank(line);
                transactions.add( parseLine(line) );
            } catch (InvalidLineFormatException e) {
                Logger.log( e.getMessage());
            }
            lineNumber ++;
        }
        return transactions;
    }
```

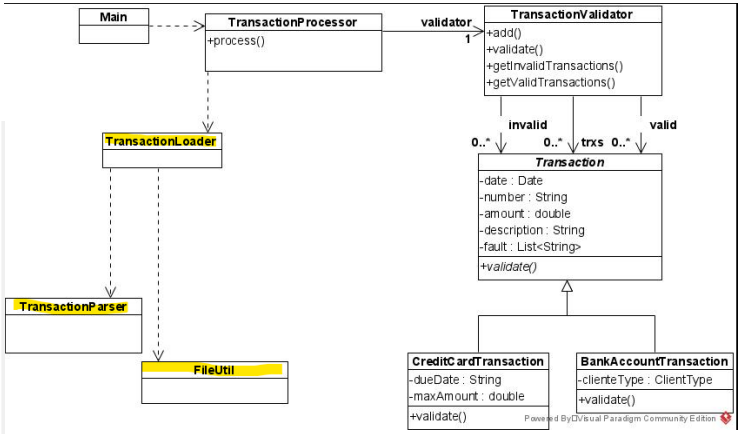
Recoger excepciones

```java
private Transaction parseLine(String line) throws InvalidLineFormatException {
    String [] parts = line.split(";");
    String type = parts[0];
    if (parts[0].equals("cc")) {
        return parseCreditCard(parts);
    } else if (type.equals("acc")) {
        return parseCurrentAccount(parts);
    } else {
        throw new InvalidLineFormatException(lineNumber,"PALABRA CLAVE NO VÁLIDA");
    }
}
```

Recibe lista de líneas
Devuelve lista de Transacciones

5. Carga de fichero: TransactionParser

Se realiza desde TransactionLoader




```
private Transaction parseCreditCard(String[] parts) throws InvalidLineFormatException {
    checkFields(parts, 7);
    checkDate(parts[1]);
    String date = parts[1];
    String cardNumber = parts[2];
    String dueDate = parts[3];
    double maxAmount = toDouble(parts[4]);
    double amount = toDouble(parts[5]);
    String description = parts[6];

    return new CreditCardTransaction(date, cardNumber, dueDate, maxAmount, amount, description)
}
```

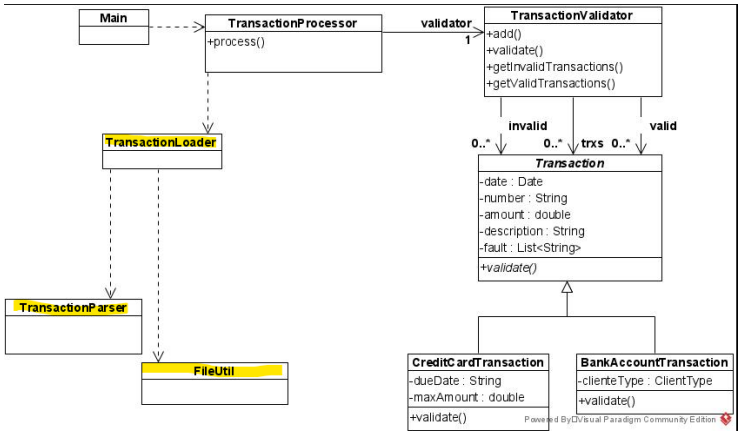
```
    private void checkDate(String string) throws InvalidLineFormatException {
        String[] partDate = string.split("/");

        if (partDate.length != 3)
            throw new InvalidLineFormatException(lineNumber, "FORMATO DE FECHA INVÁLIDO");
        checkYear(partDate[0]);
        checkMonth(partDate[1]);
        checkDay(partDate[2]);
    }
```

```
private void checkMonth(String string) throws InvalidLineFormatException {
    if (string.length()!= 2) {
        throw new InvalidLineFormatException(lineNumber, "FORMATO DE FECHA INVÁLIDO");
    }
    try {
        int month = Integer.parseInt(string);
        if (month <=0 || month > 12) {
            throw new InvalidLineFormatException(lineNumber, "FORMATO DE FECHA INVÁLIDO");
        }

    } catch (NumberFormatException e) {
        throw new InvalidLineFormatException(lineNumber,"FORMATO DE FECHA INVÁLIDO");
    }
}
```

Se recoge excepción del sistema y se transforma en excepción propia

5. Carga de fichero. Carga de lista en Validator

El Transaction Processor crea un validador y le pasa la lista de transacciones




```java
public class TransactionValidator {
    List<Transaction> allTransactions = new ArrayList<>();
    List<Transaction> validTransactions = new ArrayList<>();
    List<Transaction> invalidTransactions = new ArrayList<>();


    public void add(List<Transaction> trxs) {
        ArgumentChecks.isNotNull(trxs);
        for (Transaction t : trxs) {
            try {
                checkRepeated(t);
                allTransactions.add(t);
            } catch (TransactionException e) {
                Logger.log(e.getMessage());
            }

        }
    }
```

Si se repite alguna, no se añade y se graba mensaje en el log
Se recoge aquí para que continúe con la siguiente transacción

```java
private void checkRepeated(Transaction t) throws TransactionException {
    for (Transaction transaction:allTransactions) {
        if (isSameNumber(transaction, t) && isSameDate(transaction, t)) {
            throw new TransactionException("operación repetida");
        }
    }
}

private boolean isSameDate(Transaction t1, Transaction t2) {
    return t1.getDate().equals(t2.getDate());
}

private boolean isSameNumber(Transaction t1, Transaction t2) {
    return t1.getNumber().equals(t2.getNumber());
}
```

6. Validación

Se realiza desde TransactionValidator

```java
public void validate() {
    for (Transaction transaction: allTransactions) {
        transaction.validate();
        if (transaction.getFaults().size() > 0 ) {
            invalidTransactions.add(transaction);
        } else {
            validTransactions.add(transaction);
        }
    }
}
```

Se delega la validación en las transacciones

Se añade a lista de inválidas o válidas según haya tenido o no fallos

```java
public abstract class Transaction implements Comparable<Transaction> {

    private String date;
    private String number;
    private double amount;
    private String description;

    private List<String> validationFaults = new ArrayList<>();
```

```java
public class CreditCardTransaction extends Transaction {

@Override
public void validate() {
    checkDate();
    checkAmount();
    checkNumber();
}

@SuppressWarnings("static-access")
private void checkNumber() {
    if (! new Lhun().isValid(this.getNumber())) {
        this.getFaults().add("El Lhun es inválido");
    }
}
```

```java
public class CurrentAccountTransaction extends Transaction{
@Override
public void validate() {
    checkAmount();
    checkIban();
}

@SuppressWarnings("static-access")
private void checkIban() {
    if (! new IBAN().isValid(this.getNumber())) {
        this.getFaults().add("IBAN inválido");
    }
}

private void checkAmount() {
    if (type.equals(ClientType.NORMAL) && this.getAmount() > 1000) {
        this.getFaults().add("Cantidad mayor que máxima permitida");
    }
}
```

7. Ordenación de transacciones

Desde la clase TransactionValidator se devuelve lista ordenada de transacciones válidas

```
public List<Transaction> getValidTransactions() {
    //Collections.sort(validTransactions);
    Collections.sort(validTransactions, new byDateAndAmountComparator());
    return new ArrayList<>(validTransactions);
}
```

Opción 1. Crear compareTo dentro de Transaction. Orden natural

SIEMPRE DEVOLVER UNA COPIA

Opción 2. Crear comparador

```
public abstract class Transaction implements Comparable<Transaction> {

public int compareTo(Transaction t) {
    int result = this.getDate().compareTo(t.getDate());
    if (result == 0) {
        return this.getNumber().compareTo(t.getNumber());
    }
    else {
        return result;
    }
}
```

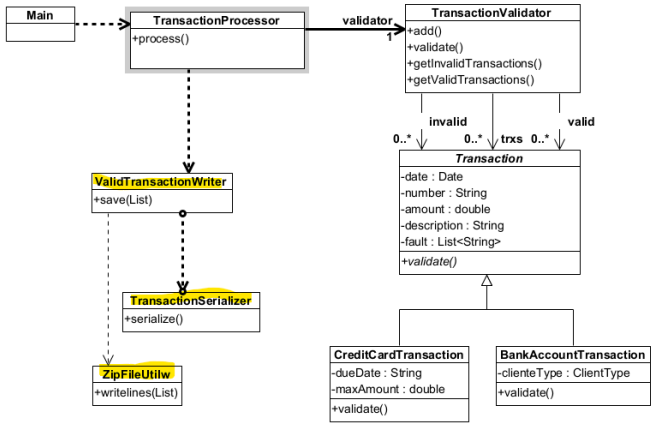
Sólo puede haber un orden natural.
Resto: comparadores

```
public class byDateAndAmountComparator implements Comparator<Transaction>{

    @Override
    public int compare(Transaction o1, Transaction o2) {
        ArgumentChecks.isTrue(o1 != null);
        ArgumentChecks.isTrue(o2 != null);
        int result = o1.getDate().compareTo(o2.getDate());
        if (result == 0) {
            return ((Double) o1.getAmount()).compareTo(o2.getAmount());
        }
        return result;
    }
```

Orden ascendente
O2.getDate().compareTo(o1) sería descendente

8. Grabación de transacciones válidas

Se realiza desde TransactionValidator




```java
public class TransactionProcessor {

    private TransactionValidator validator;

    public void process(String trxFileName) throws TransactionException {

        ArgumentsCheck.isNotEmpty(trxFileName);

        List<Transaction> trxs = new TransactionLoader( trxFileName ).load();

        validator = new TransactionValidator();
        validator.add( trxs );
        validator.validate();

        List<Transaction> validTransactions = validator.getValidTransactions();
        List<Transaction> invalidTransactions = validator.getInvalidTransactions();

        new ValidTransactionWriter( trxFileName ).save( validTransactions );
        new InvalidTransactionWriter( trxFileName ).save( invalidTransactions );
    }
```

```java
public class ValidTransactionWriter {

    private String fileName;

    public ValidTransactionWriter(String fileName) {
        ArgumentsCheck.isNotNull(fileName !=null);
        ArgumentsCheck.isNotNull(!fileName.isBlank());
        this.fileName = fileName;
    }

    public void save(List<Transaction> validTrx) {
        ArgumentsCheck.isTrue(validTrx!=null);
        List<String> lines = new TransactionSerializer().serialize( validTrx );
        new ZipFileUtil().writeLines(fileName + ".gz" ,lines);
    }
}
```

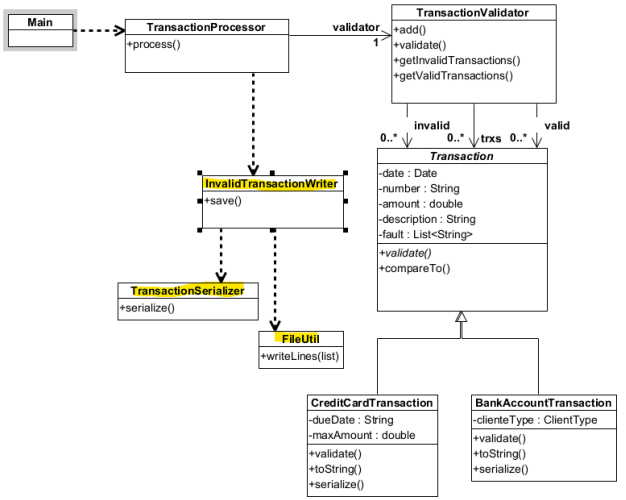
```java
public class TransactionSerializer {

    public List<String> serialize(List<Transaction> invalidTrx) {
        ArgumentsCheck.isTrue(invalidTrx != null);
        List<String> lines = new ArrayList<>();
        for (Transaction t: invalidTrx) {
            lines.add(t.serialize());
        }
        return lines;
    }
}
```

```java
public class CurrentAccountTransaction extends Transaction{

    @Override
    public String serialize() {
        String result = String.format("acc;%s;%s;%s;%f;%s",
                this.getDate(),
                this.getNumber(),
                this.type,
                this.getAmount(),
                this.getDescription()
               );
        return result;
    }
```

```java
public class CreditCardTransaction extends Transaction {
    @Override
    public String serialize() {
        String result = String.format("cc;%S;%S;%S;%f;%f;%s",
                this.getDate(),
                this.getNumber(),
                this.finalDate,
                this.maxAmount,
                this.getAmount(),
                this.getDescription()
               );
        return result;
    }
```

8. Grabación de transacciones inválidas

Se realiza desde TransactionValidator




```java
public class TransactionProcessor {

    private TransactionValidator validator;

    public void process(String trxFileName) throws TransactionException {

        ArgumentsCheck.isNotEmpty(trxFileName);

        List<Transaction> trxs = new TransactionLoader( trxFileName ).load();

        validator = new TransactionValidator();
        validator.add( trxs );
        validator.validate();

        List<Transaction> validTransactions = validator.getValidTransactions();
        List<Transaction> invalidTransactions = validator.getInvalidTransactions();

        new ValidTransactionWriter( trxFileName ).save( validTransactions );
        new InvalidTransactionWriter( trxFileName ).save( invalidTransactions );
    }
```

```java
public void save(List<Transaction> invalidTrx) {
    ArgumentsCheck.isTrue(invalidTrx != null);
    List<String> lines = new TransactionSerializer().serialize( invalidTrx );
    new FileUtil().writeLines(  fileName + ".invalid.trx" ,lines);
}
```

```java
public class TransactionSerializer {

    public List<String> serialize(List<Transaction> invalidTrx) {
        ArgumentsCheck.isTrue(invalidTrx != null);
        List<String> lines = new ArrayList<>();
        for (Transaction t: invalidTrx) {
            lines.add(t.serialize());
```

```java
public class CurrentAccountTransaction extends Transaction{

    @Override
    public String serialize() {
        String result = String.format("acc;%s;%s;%s;%f;%s",
                this.getDate(),
                this.getNumber(),
                this.type,
                this.getAmount(),
                this.getDescription()
               );
        return result;
    }
```

```java
public class CreditCardTransaction extends Transaction {
    @Override
    public String serialize() {
        String result = String.format("cc;%S;%S;%S;%f;%f;%s",
                this.getDate(),
                this.getNumber(),
                this.finalDate,
                this.maxAmount,
                this.getAmount(),
                this.getDescription()
               );
        return result;
    }
```

9. Pruebas. TransactionLoader::load

Casos
1- Fichero vacío
2- Fichero con varias líneas y crea lista de Transacciones correctas

Los casos con líneas incorrectas se contemplan en los tests del parser

```
@Before
public void setUp( ) {
    expectedList= new ArrayList<>();

    // se crea una transacción de tipo cuenta corriente como esta
//      cc;2021/05/01;3531331821537868;2021/12/01;1000;50;Alibaba Supermarket
    ccTransaction = new CreditCardTransaction ("2021/05/01",
                                               "3531331821537868",
                                               "2021/12/01" ,
                                               1000,
                                               50,
                                               "Alibaba Supermarket" );

    // se crea una transacción de tipo crédito como esta
//      acc;2021/06/09;ES0310250124910021214445;N;150;Car repairing at Caris
    accTransaction = new CurrentAccountTransaction("2021/06/09",
                                                   "ES0310250124910021214445",
                                                   ClientType.NORMAL,
                                                   150,
                                                   "Car repairing at Caris");

    // se añaden a la lista esperada puesto que son las mismas que se deben
    // cargar del fichero
    expectedList.add(ccTransaction);
    expectedList.add(accTransaction);
}
```

```
/**
 * GIVEN fichero con una transacción de cada
 * WHEN se carga
 * THEN devuelve lista con las dos transacciones
 *
 */
@Test
public void twoTransactionsFile() throws TransactionException{
    // tiene las dos transacciones creadas en el setup
    TransactionLoader loader
      = new TransactionLoader("input_transactions.trx");
    List<Transaction> result = loader.load();

    assertEquals(2, result.size());
    assertEquals(expectedList, result);
}
```

Asegúrate de que el número de líneas no sobrepase las 120
Window/Preferences/Editors/TextEditors/show print Margin 120

Para hacer esta comparación ES IMPRESCINDIBLE tener redefinido el equals en las clases Transactions y sus derivadas

10. UML completo