Cuprins

Cerințe funcționale

Departamentul de marketing al unei companii care deține un magazin online dorește să obțină o serie de date statistice despre plățile efectuate prin intermediul site-ului.

Astfel, metricele solicitate de departamentul de marketing sunt:

- media valorilor plăților;
- numărul de plăți cu valoare până în 5000RON (inclusiv);
- numărul de plăți cu valoare mai mare de 5000RON;
- numărul de plăți efectuate de persoane ce nu au împlinit vârsta majoratului;
- suma totală a plăților efectuate de cetățeni români născuți în București;
- numărul de cetățeni străini ce au efectuat plăți.

Departamentul comercial acceptă să pună la dispoziția programatorului plățile încasate, dar fără a dezvălui identitatea completă a clienților ce au efectuat aceste plăți. Astfel datele ce pot fi puse la dispoziția programtorului sunt:

- CNP-ul plătitorului și
- valoarea plății.

Cerințe tehnice

Să se dezvolte o aplicație Java care să proceseze datele despre plățile puse la dispoziție de departamentul comercial și să extragă informațiile solicitate de departamentul de marketing.

Având în vedere că în trecut s-a constatat faptul că uneori CNP-urile clienților ce ajung la nivelul plăților nu sunt valide și, în plus, datele puse la dispoziție de departamentul comercial vor fi procesate manual,

se cere ca pe lângă informațiile solicitate de departamentul de marketing, în datele de ieșire să apară și erorile de procesare.

Date de intrare

- Datele de intrare se găsesc într-un fișier csv folosind ca separator de câmpuri `;`.
- Fiecare linie din acest fișier reprezintă o plată.
- În cadrul unei linii, pe prima poziție se află CNP-ul, iar pe a doua suma plății.
- Suma plăţii foloseşte caracterul "." (punct) ca separator de zecimale.
- Este posibil ca în fișier să apară linii goale. Acestea trebuie ignorate.

Exemplu:

```
1930107189436;15324.98
2961022513249;1860
1960608409971;18.1
```

Date de ieșire

Aplicația va furniza datele solicitate de departamentul de marketing și erorile găsite în datele de intrare într-un format structurat descris de următorul tip abstract de date:

```
PayMetrics {

    averagePaymentAmount: BigDecimal,

    smallPayments: Integer,

    bigPayments: Integer,

    paymentsByMinors: Integer,

    totalAmountCapitalCity: BigDecimal,

    foreigners: Integer,

    errors: List of {

        line: Integer,

        type: Integer

    }
}
```

unde:

- `averagePaymentAmount` reprezintă *media valorilor plăților*;
- `smallPayments` este *numărul de plăți cu valoare până în 5000RON (inclusiv)*;
- `bigPayments` este *numărul de plăți cu valoare mai mare de 5000RON*;
- `paymentsByMinors` este *numărul de plăți efectuate de persoane ce nu au împlinit vârsta majoratului*;
- `totalAmountCapitalCity` este *suma totală a plăților efectuate de cetățeni români născuți în București*;
- `foreigners` este *numărul de cetățeni străini ce au efectuat plăți*;
- `errors` este lista de erori, o eroare având proprietățile:
    - `line` - numărul liniei la care a apărut eroarea;
    - `type` - tipul erorii; una din valorile:
        - `0` - linia este invalidă, i.e. nu conține două valori separate prin `;`;
        - `1` - CNP-ul nu este valid;
        - `2` - suma plății nu este validă.

Datele din această structură vor fi serializate într-un fișier.

Cerințe suplimentare

Având în vedere că metricele ce trebuie extrase din aceste date de intrare se pot schimba la un moment dat și faptul că pot apărea alte situații în care e nevoie să se valideze CNP-uri și să se extragă informații din ele, se cere implementarea validării și interpretării unui CNP într-un API distinct, reutilizabil cu următoarea structură:

```
interface CnpValidator {

    /**
     * Valideza CNP-ul primit ca parametru si returneaza partile componente ale
     * acestuia.
     *
     * @param cnp
```

```
     *            CNP-ul de validat
     * @return partile componente ale CNP-ului
     * @throws CnpException
     *             daca CNP-ul nu este valid
     */
    CnpParts validateCnp(String cnp) throws CnpException;

}
interface CnpParts {

    /**
     * Sexul determinat pe baza primei cifrei din CNP.
     */
    Sex sex();

    /**
     * Posesorul CNP-ului este cetatean strain?
     *
     * @return {@code true} daca este cetatean strain, {@code false} in caz
     *         contrar
     */
    Boolean foreigner();

    /**
     * Judetul.
     */
    Judet judet();

    /**
     * Data nasterii.
     */
    CalDate birthDate();

    /**
     * Numarul de ordine.
     */
    Short orderNumber();

}
interface CalDate {

    Short year();

    Byte month();

    Byte day();
}
```

Cerințe detaliate

Toate clasele și interfețele trebuie să fie în sub-pachete ale package-ul `ro.axonsoft.internship21`

- clasele și interfețele referitoare la validarea CNP-ului vor fi în sub-package-ul `cnp`,
- iar cele referitoare la procesarea plăților în `pay`;

Implementarea interfeței `CnpValidator` se va face într-o clasă cu numele `CnpValidatorImpl` în același `package`.

În ceea ce privește procesarea plăților, aceasta se va face într-o clasă `PayMetricsProcessorImpl` implementând următoarea interfață:

```
interface PayMetricsProcessor {

    /**
     * Proceseaza platile din {@code paymentsInputStream} si scrie metricele in
     * {@code metricsOutputStream}
     *
     * @param paymentsInputStream
     *             input stream al fisierului csv conținând plățile
     * @param metricsOutputStream
     *             output stream al fișierului in care se serializeaza
```

```
     *             obiectul conținând metricile și erorile
     * @throws IOException
     *             daca apare o eroare de I/O.
     */
    void process(InputStream paymentsInputStream, OutputStream metricsOutputStream) throws IOExce

}
```

Obiectul serializat în `OutputStream` trebuie să implementeze următoarea interfață:

```
interface PayMetrics {

    /**
     * Numarul de cetateni straini care au efectuat plati.
     */
    Integer foreigners();

    /**
     * Numarul de plati efectuate de catre minori.
     */
    Integer paymentsByMinors();

    /**
     * Numarul de plati peste pragul de 5000RON.
     */
    Integer bigPayments();

    /**
     * Numarul de plati pana in 5000RON inclusiv.
     */
    Integer smallPayments();

    /**
     * Media sumelor tuturor platilor. Valoarea are maxim doua zecimale.
     */
    BigDecimal averagePaymentAmount();

    /**
     * Totalul platilor efectuate de cetatenii romani nascuti in Bucuresti.
     */
    BigDecimal totalAmountCapitalCity();

    /**
     * Erorile de procesare.
     */
    Set<PayError> errors();
}
public interface PayError {

    /**
     * Numarul liniei la care s-a obtinut eroarea.
     */
    Integer line();

    /**
     * Tipul erorii:
     * <ul>
     * <li>0 linie invalida</li>
     * <li>1 pentru CNP invalid</li>
     * <li>2 pentru suma plata invalida</li>
     * </ul>
     */
    Integer type();
}
```

Descrierea formatului CNP este disponibilă aici.

În implementare se pot folosii oricare din bibliotecile open-source, disponibile pe mavenCentral.

Diagrama API-ului validatorului de CNP-uri

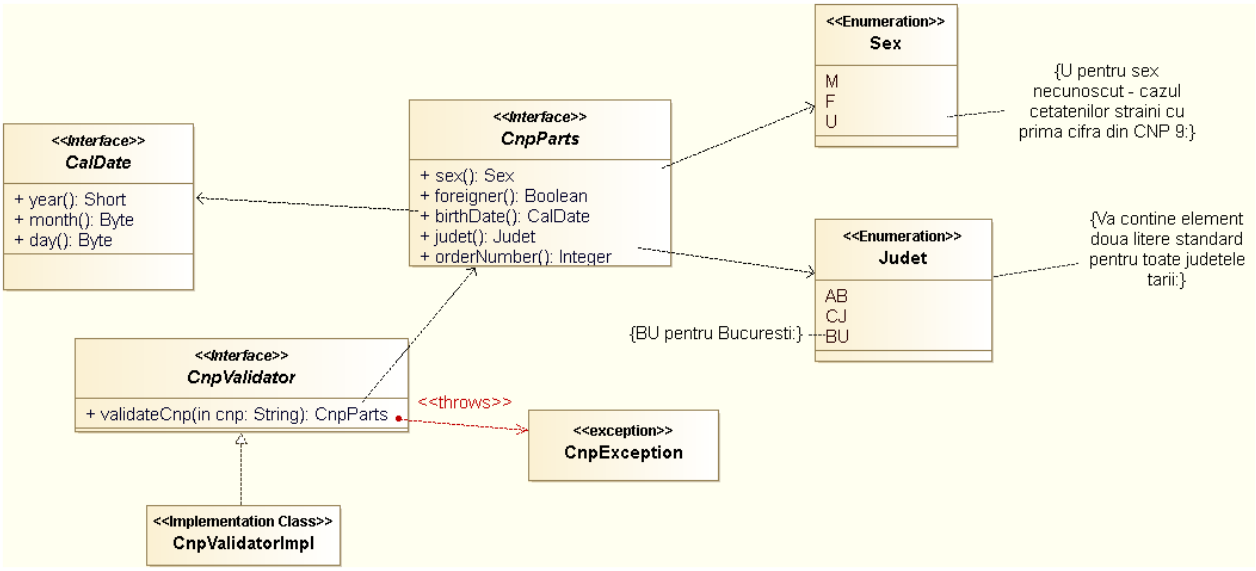
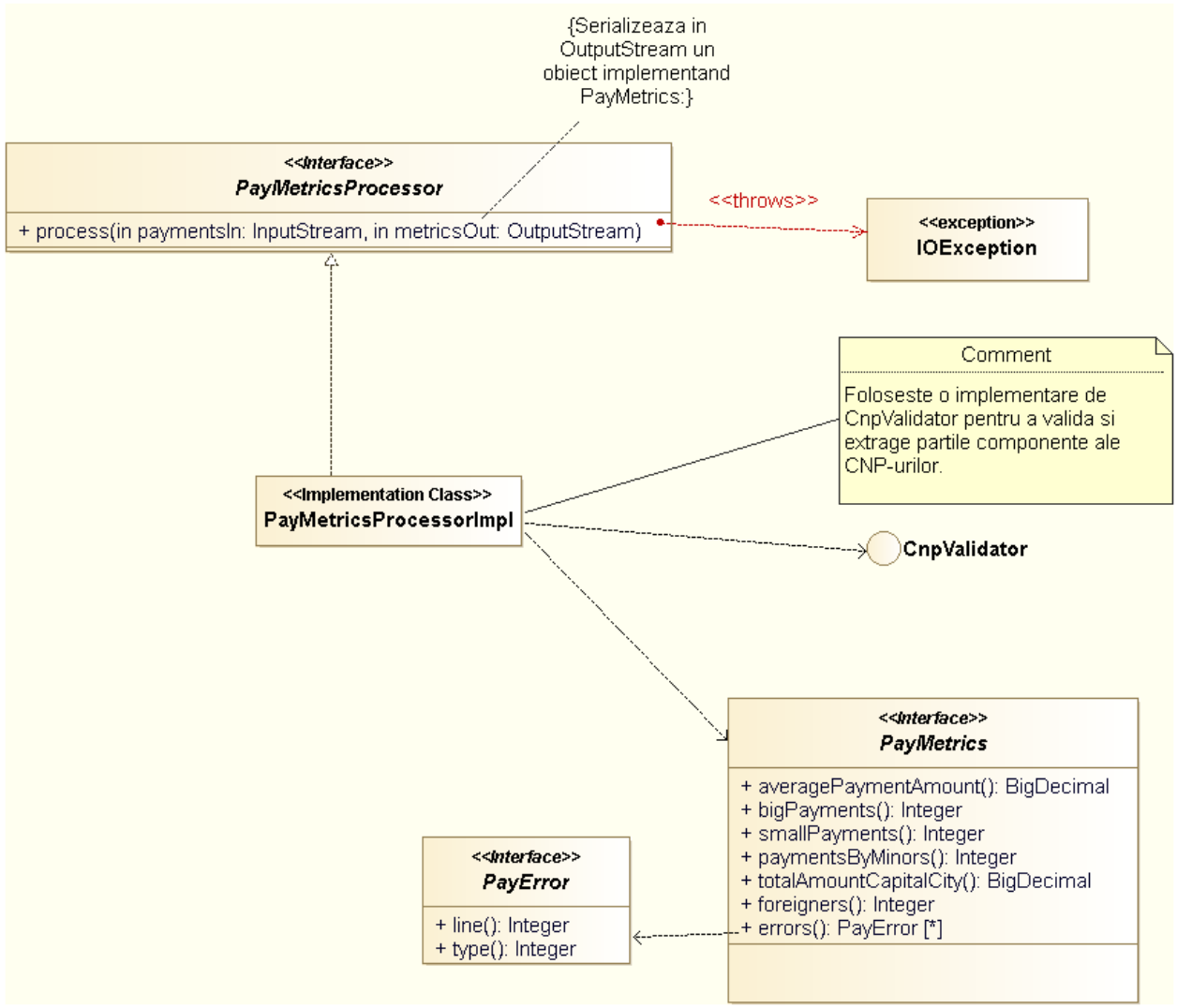


Diagrama API-ului procesatorului de plăţi



Despre livrare și evaluare

Livrare

- Livrarea se va face sub forma unei arhive zip cu numele `[nume]-[prenume].zip`

    - conținând un director `src` cu `package`-urile de surse
    - dacă folosiți biblioteci externe de pe `mavenCentral` treceți numele lor într-un fișier cu numele `dependencies` inclus în arhivă.
    - dacă aveți precizări cu privire la codul vostru sau ați avut nelămuriri cu privire la cerințe, treceți-le într-un fișier `readme` inclus în arhivă.
- Încărcaţi arhiva prin intermediul acestei aplicaţii accesând secţiunea Trimite.

Evaluare

- Se vor evalua respectarea cerințelor cu privire la:
    - validatorul de CNP
    - procesarea tranzacțiilor.
- În `mavenCentral` există cel puțin o bibliotecă implementând validarea de CNP-uri. Dacă nu implementați propriul vostru validator de CNP-uri puteți să folosiți una din acestea sau să copiați codul open-source, caz în care va trebui să menționați acestea în `dependencies` sau `readme`.
- Puncte bonus vor fi acordate pentru:
    - cod ușor de citit, respectând paradigmele OOP şi bunele practici de programare
    - folosirea cu rost a cât mai multor API-uri din JDK
    - folosirea cu rost a unor biblioteci utilitare externe (exceptând implementarea validatorului de CNP)
    - folosirea sintaxei Java 8
    - comentarea codului, în special în format `javadoc`
    - scrierea de teste unitare

Mult succes!