Documentation of workflow

Author: Daniel Hunacek
Version: 2.0

Library: ofen-workflow-v2 .0.jar
Date: 8/5/2016

General

The Java library contains 2 different workflows that take a CSV as input and return some JSON formatted string.

The first workflow deals with energy accounting data. It extracts the data from the accounts of the organization, transforms it and prepares it to be reported.

The second workflow deals with energy consumption. It also gets data from a CSV file and prepares the data for report.

To run the workflow, follow these few steps:

- Add the library .jar file to your project (file: **ofen-workflow-v2.0.jar**).
- Import the needed classes in your Class:

```
# import ch.hevs.iig.cockpit.Workflow ;                    for accounting
# import java.util.ArrayList ;                             for accounting
# import ch.hevs.iig.cockpit.Account ;                     for accounting
# import ch.hevs.iig.cockpit.WorkflowCons ;                for energy consumption
# import ch.hevs.iig.cockpit.Consumption ;                 for energy consumption
# import java.util.Locale ;                                needed in both cases
```

- To run a workflow instantiate Workflow (or WorkflowCons) in your code:

```
# Workflow wf = new Workflow(nameOfCity, populationSize, locale, [accounts filter]) ;
```

  - *nameOfCity*: String (ex: “Sierre”)
  - *populationSize*: int (ex: 19543)
  - *locale*: java.util.Locale used to format the localized data like currency
    - this can be changed to something different according to needs (ex: new Locale(“fr”, “CH”))
  - accountsFilter String list semi-colon separated of accounts number to be filtered
    - (ex: “70.210.314.37;70.260.314.04;70.310.314.02;70.330.314.09”)

or

```
# WorkflowCons wf = new WorkflowCons(nameOfCity, year, populationSize, locale) ;
```

  - year: int (ex: 2015)

```
# wf.parseCSV(inputStream, delimiter) ;  this will read the input file
```

  - *inputStream*: java.io.InputStream content of the input file (“input.csv”)

- *delimiter*: char representing the delimiter used (ex: ',' for input.csv and ';' for inputCons.csv)

`ArrayList<Account> accounts = wf.run() ;` this will execute the workflow and return results
`Consumption c = wf.run() ;` this will execute the workflow and return results
Return of `wf.run()` is in ArrayList of type **Account** for accounting workflow or in **Consumption** type for energy consumption workflow. Result format is explained below.

Input format for Workflow (accounting):

- CSV file
- Encoding: UTF-8
- Delimiter: "**;**" / "**,**" (others might work too)
- Text surrounded by "…" is supported
- Columns:
  - "Numero" (account number)
  - "Libelle" (account name)
  - "Debit year"
  - "Credit year"
  - "Debit year-1"
  - "Credit year-1"
  - "Debit year-2"
  - "Credit year-2"
- Order and number of columns matters, column names for Debit + Credit needs full year at the end (prefix is not important)
- You can use this example as column names (just change the years):
  - **"Numero","Libelle","D2013","C2013","D2012","C2012","D2011","C2011"**

Input format for WorkflowCons (energy consumption):

- CSV file
- Encoding: UTF-8
- Delimiter: "**;**" / "**,**" (others might work too)
- Text surrounded by "…" is supported
- Columns:
  - No installation (installation number, format: xxxxxxxxx.xxx)
  - Bâtiment (building name, format: text)
  - Désignation (type of energy consumption, format: text, ex: "Ecl. Public")
  - Adresse (address of the installation, format: text)
  - Localité (city, format: text)
  - N° facture (-)
  - Compteur (meter device, format: number)
  - kWh (consumption in kWh, format : decimal)
  - Acheminent (-)
  - Fourniture(-)
  - PCP (-)
  - RPC (-)
  - Frais mutation (-)

    - Total (-)
    - TVA (-)
    - Montant TTC (total amount, format: decimal)
- Order and number of columns matters
- current year is past as argument in constructor
- Columns with "(-)" are not specified as they are not used but don't remove them, the workflow handles the filtering (removing them might break the extraction)
- You can use this example as column names:
    - **No installation;Bâtiment;Désignation;Adresse;Localité;n° facture;Compteur;kWh;Acheminent;Fourniture;PCP;RPC;Frais mutation;Total net;TVA;Montant TTC**

Output

Workflow (accounting):

Accounting workflow returns an **ArrayList** of **Account** objects. Each account refers to a high level account (ex: 30.40) containing a group of sub accounts (ex: 30.400.312.00). Each sub account is a **SubAccount** object containing a list of years (**AccountYear** object). Each year contains a name value (ex: 2015) and an amount value (total debit of the sub account for the specific year). The list of attributes for each class is described below.

Class Account
accountNumber (String)
label (String)
subAccounts (ArrayList<SubAccount>)

Class SubAccount
accountNumber (String)
label (String)
type (Energy) -> can get String value from object method: *mySubAccount*.**getTypeAsString()**
years (ArrayList<AccountYear>)

Class AccountYear
name (int)
amount (BigDecimal)
type (Energy) -> only used internally for CHF to kWh conversion
kwh (BigDecimal) -> auto computed kWh value from amount according to Energy attributes

*Energy static attributes are **meanCost** and **conversionFactor** specific to each type (currently set to: ELECTRICITE, MAZOUT and GAZ (all other Energy types return 0.0 as kWh value).*

WorkflowCons (consumption):

Consumption workflow returns a **Consumption** object which contains some general attributes (ex: total kWh, total amount, average cost per kWh) and 3 lists: consumption, small consumers and big consumers. Each list is an **ArrayList** containing **ConsumptionLocation** objects that hold values about a specific location (ex: “Feux Limineux – Carrefour Temple”). A location contains a name, a kWh value, an amount and a cost per kWh. Attributes of these classes are described below.

Class Consumption
city (String)
year (int) -> year to be reported
totalKwh (BigDecimal) -> global kWh for the current year
totalAmount (BigDecimal) -> global amount for the current year
averageCostPerKwh (BigDecimal) -> global average cost per kWh for current year
consumption (ArrayList<ConsumptionLocation>)
smallConsumers (ArrayList<ConsumptionLocation>)
bigConsumers (ArrayList< ConsumptionLocation >)

Class ConsumptionLocation
location (String) -> name of the location
kwh (BigDecimal)
amount (BigDecimal)
costPerKwh (BigDecimal)