CASE STUDY FOR *Junior Java Backend Services*

Hi and welcome to the trivago *Junior Java Backend Services* challenge.

Your application made us curious and therefore we would like to take you to the next step.

Here is the chance for you to convince us that you are the right person for the job!

We wish you good luck!

PREPARATION TIME: *None*

SUBMISSION DEADLINE: *7 days*

HOW TO SUBMIT: *Please note that you are not allowed to put your solution on github, Bitbucket, or other publicly accessible repositories.*

THE CHALLENGE

Task: *eCandy Product Catalog*

eCandy is an online shop for candy (surprise!) their product range contains the world's best and sweetest candy. However, their IT-architecture is really messed up: A few weeks ago in order to extend their product range, eCandy acquired two competitor shops. Until now, they were not capable of aligning the diferent computer systems with each other. Nevertheless they want to present their customers a unified product catalog, containing all items for all integrated shops.

As eCandy (understandably enough) does not trust their own developers anymore, they hired you to build this integrated product catalog for them. Luckily, they also provided some information on the exact requirements. You can find the interface to all four systems attached.´Start with extracting the required libraries and make yourself familiar with th*em! As you will see, three of the systems provide their product information in XML format, one of them provides JSON data. Try to retrieve all four product ranges from the server. Please be aware that some of the data may be corrupted. Once you have access to all product ranges, you can build the overall product catalog.*

The required output format is:

```
<catalog>
    <product>
        <id></id>
        <name></name>
        <price></price>
        <currency></currency>
        <bestBefore></bestBefore>
    </product>
    <product>
        <id></id>
        <name></name>
        <price></price>
        <currency></currency>
        <bestBefore></bestBefore>
    </product>
</catalog>
```

Please note that the catalog should only contain unique items. If several product ranges contain the same product (i.e. product has the same name), the catalog should only contain it once (the one with the smallest bestBefore date). All product prices in the catalog should be in EUR and sorted alphabetically by name. The bestBefore date format should be DD/MM/YYYY.

eCandy wants to be able to provide the output path and file name of the resulting product catalog to your application. To ensure correct usage of the program, please provide a short documentation on how to build your source code and run your application.

Bonus question

Last but not least, there is an optional bonus question. You do not have to answer this, a well working product catalog is fair enough for eCandy. However, you would surely help them a lot by answering this:

To ensure a continuous and smooth operation of your program, eCandy would be really interested in how to test your product catalog. Which parts of the application can be tested? And how would you test them? You are free to decide how to answer this question: Not at all, pure text or maybesome code?

Hints / How To's:

1. Access to the servers:

In the attached tarball you can find a .jar file named case-study-ps.jar. Once you have this file on your system, run it using the command "java -jar case-study-ps.jar server". This will start an HTTP server with which you can communicate to get the JSON and the three XML products lists.

2. Currency conversion:

When you have a price in a currency that is not EUR, you can use our client to convert it. Grab the file "case-study-currency-client.jar" from the attached tarball and add it to your project. Now you can call the method getConvertedPrice(), which converts a price from a currency to another one. It's important that you have "case-study-ps.jar" running.

3. Paths to web resources:

- First XML products list: <server path>/products/xml?source=1

- Second XML products list: <server path>/products/xml?source=2

- Third XML products list: <server path>/products/xml?source=3

- JSON products list: <server path>/products/json

4. JSON and XMLs structures

4.1 XML format

```
<product bestBefore="">
        <name></name>
        <price></price>
```

```
    <currency></currency>
</product>
```

2.4.2 Second XML response format

```
<product name="">
    <price currency=""></price>
    <bestBefore date=""/>
</product>
```

2.4.3 Third XML response format

```
<item>
    <id></id>
    <product bestBefore="">
        <name></name>
        <price></price>
        <currency></currency>
    </product>
</item>
```

2.4.4 JSON response format

```
{
    "id": "...",
    "name": "...",
    "price": "...",
    "currency": "...",
    "bestBefore": "..."
}
```