CCH Tagetik Software Engineer Test

Congratulations on getting to this stage!

The following exercises are sent to you as an additional tool to be used in your technical interview. We ask you to solve it using `Java`.

All problems are containing all instructions to solve them but feel free to make your assumptions in case something is not clear enough. Please consider explaining your decisions by documenting them.

Goal

The goal is to provide us with a full understanding of your coding style and skills. We'll pay particular attention to:

- the code structure
- the design
- choice of data structures
- how do you approach the problem

In your repository you should also include:

- A `text` file with your **thought process** (for every question that requires a description)
- A `readme` file with all the instructions that you think will be useful for us to **test** your solution.

We kindly ask you to put your code on a private project in https://github.com and share your repo with the user **tgk-technical-interviews**. In alternative, you can send your code as an export of a git project that can be sent as an attachment to the address tgk-dl-technical-int@wolterskluwer.com .

Exercices

Entities model

Please consider the following entity model for the next exercises:

```
class Product {
    int productId;
    String description;
    Category category;
    int quantity;
    double price;
    {chooseYourPreferredDateClass} date;
}

enum Category {
    CAT1, CAT2, CAT3
}
```

Exercise 1

Consider a tree of *TreeNode*. Given a *Category* as input, returns the *List* of all descendants *Categories* (including *Category* passed as input). Note: assume that all *Categories* are different.

```
/**
* A node of the tree.
*/
class TreeNode {
    Category category;
    List<TreeNode> children;
}

/**
 * @param category the category of which find all descendants
 * @param root the root node of the tree
 *
 * @return the list of all descendants categories, including the category passed as input
 */
public List<Category> findAllDescendantsBy(Category category, TreeNode root) {

}
```

Exercise 2

Consider `total amount` of a *Product* as `price * quantity`. Determine the sum of all `total amount` of *Products* belonging to the input category using only `java.util.stream` methods.

```
/**
 * @param products the list of products
 * @param category the category on which to filter
 *
 * @return The sum of total amount for all products belonging to the input category
 */
public double amount(List<Product> products, Category category) {
    // to be implemented
}
```

Exercise 3

Implement a product service that exposes the following endpoints:

- POST
  - `addProduct(...)`: takes a product as parameter, stores it in a cache / in-memory database (up to you!). The *updatedAt* field must be filled automatically by the system with format *yyyy-mm-dd*. Price and quantity can't be < 0.
- GET
  - `getAll()`: get all products
  - `getProductById(...)`: takes a *productId* as parameter and returns the entire entity
  - `getAllProductByCategory(...)`: takes a *category* as parameter and returns all products belonging to that *category*
  - `getAllProductsGroupedByCategory()`: returns all products grouped by *category*
- PUT
  - `updateProduct(...)`: takes a *productId* and a *price* or/and a *quantity* and updates the product if exists, otherwise returns an error.

Endpoints can be either REST or gRPC, your choice.