Inh.04 — Supermarket

Table of Contents

We are processing products sold in a supermarket for its new online-shop offering Unlike a traditional grocer, a supermarket offers not only food, but also non-food articles.

1. Products

Both food and non-food products share some common properties:

- A barcode
  - Has to be a valid barcode, see validation
- A product name
  - Cannot be empty
- A stock quantity
  - Must not be negative

Invalid values for barcode & product name are replaced by `"Invalid!"`, a negative quantity is set to `0`.

1.1. Barcode Validation

- All products have an EAN-8 barcode
    - 7351 3537
- It consists of 8 digits
    - So no letters are allowed
    - Shorter or longer EANs are not supported
- The last (eight) digit is a *check digit* which is calculated from the first 7 digits
    - Each digit is multiplied by a certain weight
        - A digit at an *even* index has a weight of 3
        - A digit at an *uneven* index has a weight of 1
    - All *weighted* digits are summed up
    - Then the *difference* of the *unit place* to *the next multiple of ten* is calculated
    - This difference has to *match* the check digit ⇒ then the barcode is valid

1.1.1. Example

- For the EAN 73513537
- ⇒ Check digit is 7

Position	0	1	2	3	4	5	6
Weight	3	1	3	1	3	1	3
EAN	7	3	5	1	3	5	3
Multiplication-Product	21	3	15	1	9	5	9

- Sum of all multiplication products is 63
- The next (bigger) multiple of ten is 70
- The difference between 70 and 63 is 7 ⇒ the check digit should be 7
    - Actually, it is sufficient to subtract the unit place (*remainder*) from 10
- ⇒ This is a valid EAN-8 barcode

1.2. CSV Export

- All products are ready to be exported as CSV

- To support this task they have to provide two functions:
    a. Provide a header with all individual column names
    b. Turn the instance values into a CSV string

> The actual export to a file does *not* have to be implemented this time — but you should be able to do that, including proper path handling, without any problems by now!

1.2.1. `AppendToArray`

- This method is useful for tasks related to the CSV export capabilities
- It is a *generic* method
    - Despite only being called for `string` in this assignment □
- Its job is to create a *new* array which contains the content of the original array and (at the end) additional elements

1.3. Food

- Food products can contain various allergens
- The supermarket is legally required to list those to avoid life-threatening reactions for affected customers
- Those allergens are identified by a standardized code
    - Already available in the `AllergenType` `enum`
    - As defined by the Wirtschaftskammer Österreich (WKO)
- The list must not contain duplicates
- We need to offer a way of checking if any (one or more) specific allergen(s) is contained within the product
- No matter the order allergens are added or removed, they are always stored *in order*
    - You are already *so very good* at implementing sorting algorithms (□) that we will use a proper one this time by utilizing a `SortedSet`
        - Try to remember what we recently learnt about collections: why aren't we using a `SortedList` □
- The list of allergens will be *encoded* in the CSV string as *another* CSV string using `'|'` as separator

1.4. Non-Food

- Non-food products can be reviewed by customers
- Each `Review` has the following properties:
    - Date & Time it was posted

    - A star-rating
        - As defined in the `Rating` `enum`
    - A comment
- Based on all posted reviews an *average* rating can be calculated

2. Tasks

1. Create a UML class diagram using PlantUML
2. Complete the program by implementing all missing code pieces
    - Look for `TODO`s
    - Extensive unit tests have been provided
3. Write all necessary XMLDoc comments
    - Usually everything `public` & `protected` if *not* inherited in a meaningful way

This assignment makes liberal use of the `params` feature — if you can't remember what that is, check it out again