CSC8012 Assessed Coursework: Interactive System

Dr Konrad Dabrowski
konrad.dabrowski@newcastle.ac.uk

2023/2024

Deadline

The deadline for this coursework is **14:30 on Friday, 17th November 2023** and must be submitted on NESS.

Submission Notes

You should submit a .zip file containing four Java files and three text files through NESS as follows:

- `SortedLinkedList.java`, which contains your `SortedLinkedList<E>` class for Task 1.
- `Meal.java`, which contains your `Meal` class for Task 2.
- `Subscriber.java`, which contains your `Subscriber` class for Task 2.
- `MainProgram.java`, which contains your `MainProgram` driver class for Task 2; your `main()` method should be in this class.
- `input_data.txt`, which is the input file for your program (this can be the same as the provided `input_data.txt` file; your program should only read this file in, not change it).
- `clerk.txt` should contain a record of the clerk's interactions with the program displayed in the Terminal Window (both what the clerk inputs and what your program prints out; copy and paste this from the Terminal Window in IntelliJ; your program should not write to this file).
- `letters.txt` should be the output file created by your program, which contains the letters to subscribers from when they try to add meals to their subscription, but there are not enough meals available.

Your text files should show that you have tested your program and the `letters.txt` file should correspond to the input in `clerk.txt`.

Aims

- To gain experience in designing an interactive system of practical importance.
- To reinforce your knowledge about list classes in java.
- To gain experience at using the `java.lang.Comparable<E>` interface, inheritance and generic classes in Java.

Background

Sometimes we want to keep the items in a list in sorted order, which we can achieve with a sorted list. The fundamental difference between a sorted list and an unsorted one is the "adding an item"/insertion method. Having a definition of a class for lists, we can obtain a sorted list class by altering the existing one or adding a new insertion method.

Specification

This document should be read together with the coursework guidance notes document and the coursework FAQ at `https://ncl.instructure.com/courses/50031/pages/summative-coursework`.

Task 1

`LinkedList<E>` is a java class that implements the `java.util.List<E>` interface, just like the `ArrayList<E>` you are familiar with from CSC8011; both these classes have similar methods available to use. Derive a `SortedLinkedList<E>` class as a subclass of the `java.util.LinkedList<E>` class in such a way that the items of a sorted LinkedList are sorted in ascending order. This subclass of the `LinkedList<E>` class will be needed to complete Task 2 (see Additional Assumptions (4) below). You only need to provide your new insertion method in the `SortedLinkedList<E>` class. You do **not** need to consider the other methods from the `LinkedList<E>` class that can modify the list and you must use the inherited `LinkedList<E>` class, **not** implement a `SortedLinkedList<E>` class from scratch.

Task 2

The FrozenFood company is setting up a weekly subscription service that delivers frozen meals to customers. You are asked to write a program to help an office clerk manage subscribers' subscriptions. The company offers various types of meals to its subscribers. Subscribers can add/remove meals to/from their subscription, but the number of each type of meal available is limited. Your program should read a list of registered subscribers and a list of available meals from a file. The content of the input file should have the following form: The first line contains an integer representing the number of registered subscribers, followed by the information about the subscribers (one line for every subscriber with their first name and surname). The next line contains an integer representing the number of different types of meal available, and is followed by the information about the meals (two lines for every meal: one line containing the name of the meal and the second one containing the number of these meals available each week). Example file content is given on the next page (same as the `input_data.txt` file available from Canvas), but your program should run correctly on any file in the correct format:

```
4
Ted Smith
Carl Smith
Anna Jones
Anna Smith
7
Curry
2
Chilli
6
Lasagne
8
Steak
11
Fish
8
Mushroom Stew
4
Spaghetti
5
```

The program should be able to store information about meals and subscribers:

1. For each meal, the information required is: the name of the meal and the number meals still available to be ordered.

2. For each subscriber, the office should know their first name, surname and their chosen types of meals together with the number of each type of meal that they have subscribed to. To simplify shipping, a subscriber can be subscribed to at most three different types of meal at a time (but can subscribe to more than three meals in total). We also assume that no two subscribers share both their first name and their surname.

After the initial information has been read from the file, the clerk will work with the program interactively. The program should display a menu on the screen offering a choice of possible operations, each represented by a lower case letter:

- `f` - to finish running the program.
- `m` - to display on the screen information about all the meals.
- `s` - to display on the screen information about all the subscribers.
- `a` - to update the stored data when a registered subscriber adds meals to their subscription.
- `r` - to update the stored data when a registered subscriber removes meals from their subscription.

You should implement all the above operations.

Additional Assumptions

1. When `f` is selected, the program stops running and all the data is lost. The program could be extended to save all the data to a file, but this is **not** part of the project and you must not do this!

2. When `m` is selected, the names of all meals should be displayed, along with the number of each type of meal still available.

3. When `s` is selected, the names of all the registered subscribers should be displayed, along with the types of meals they are subscribed to (if any) and the number of each of these in their subscription.

4. To store meals and subscribers you should use your `SortedLinkedList<E>` class. Meals should be sorted in the ascending (lexicographic) order of names of meals. Subscribers should be sorted in the ascending (lexicographic) order of their surnames. If two subscribers have the same surname, then their first names should decide their order. You may assume that each subscriber has exactly one first name and exactly one surname.

5. When a subscriber adds meals to their subscription, your program must check whether the subscriber is a valid subscriber, and that the meal is on the list of the available meal types. If not, an appropriate message should be displayed on the screen. If the request is a valid one, the program should check whether there are enough meals of this type available. You should assume that in one transaction a subscriber can add one type of meal to their subscription, but they should be able to add multiple meals of this type at once. If the meals are not available (or there are not enough of this type of meal), a note (in the form of a letter) should be printed to a file informing the subscriber that the meals are not available; no other errors should be logged to `letters.txt`. You may assume that the order is either fully satisfied or not carried out at all. If the ordered number of meals is available, the transaction should be processed and the stored information should be updated accordingly.

6. When a subscriber removes meals (a meal) from their subscription, your program must check whether the subscriber is a valid subscriber, whether the meal specified is on the list of meals types and whether the subscriber has subscribed to at least this many meals of this type. If not, an appropriate message should be displayed on the screen. If the request is a valid one, the stored information should be updated accordingly. You should assume that a subscriber can remove meals of only one type from their subscription in a single transaction. If a subscriber is subscribed to a number of meals of some type, they they should be able to remove one, some or all such meals from their subscription in a single transaction (your program should ask them how many of this type of meal they want to remove).

7. We assume that the company only sells the meals initially allocated, serves only its registered subscribers and processes transactions sequentially, one after the other.

8. When the program first starts, you should assume that no subscriber is subscribed to any meals.

Marking Scheme

The total number of marks which can be awarded for this coursework is 50 and accounts for 60% of the module mark for CSC8012. Marks will be awarded as follows:

- `Meal`, `Subscriber`, `SortedLinkedList<E>` classes: 15 marks
- Input and Output: 6 marks
- Implementation of operations: 15 marks
- Readability of code: 8 marks
- Evidence of testing: 6 marks

Your submission should be your own work. Submissions will be checked for plagiarism.