CC/CS21120: Workshop 4
– Hashing and Phone Book Manager –

(Program Design, Data Structures and Algorithms)

Assessed	2.5% of module marks. Signed off during timetabled workshops; submission to Blackboard
Submission	Blackboard → Assignments → Worksheets → Worksheet 4
Deadline for sign-off	During one of the **timetabled** workshops in the **week starting 29th October 2018** (or by the end of the semester if your work was submitted to Blackboard by the below submission deadline).
Deadline for submission to Blackboard	**4pm on 2nd November 2018**

A note on gaining marks:

- To gain marks, the worksheet must be signed off during one of the timetabled workshops. It will only be signed off if you can explain what the code does.
- Submit your work to Blackboard as this will represent a proof that you have done the work.
- If you are unable to get the sign-off in the corresponding workshops, you **must** submit your solution to Blackboard by the above submission deadline and get it signed off by the teaching staff as soon as possible! You will not get any marks for late sign-offs if you do not submit your solution to Blackboard by this deadline.

1 Recommended Preparation

In this worksheet you will write a simple phone book manager using the Map abstract data type. You should particularly revise hashing. The relevant material can be found in the CC/CS21120 lecture notes of Lectures 5 and 6.

To solve the task, you do not have to implement your own Map, but should use classes from the Java collections framework, such as

- `java.util.HashMap<K,V>`:
  `https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html`

Please do not hesitate to ask any of the demonstrators or the teaching staff if you have any questions.

2 Task: Implementing a Phone Book

The task is to implement a simple, text-based phone book manager. Some framework code to get you started is provided on Blackboard (and described below). Note that only Task 1 is assessed. Once you have finished Task 1, Task 2 is an additional opportunity to further practice your coding skills in the context of the Map abstract data type. Please only upload your solution to Task 1 to Blackboard.

Task 1

You should implement a simple, text-based phone book manager. To solve the task you need to implement the `PhoneBook` interface provided on Blackboard. It contains the following four methods:

- `String search(int phoneNumber)`: The user looks for the name of a person with a given phone number. The method should return the name associated with the given number, or `null` if no such person exists.
- `boolean add(int phoneNumber, String name)`: The user tries to add a new contact to the phone book. If there is already a person with this number in the phone book, the query is ignored and `false` is returned. Otherwise the new contact is added and `true` is returned.
- `String update(int phoneNumber, String name)`: The user tries to update the name for an existing phone number in the phone book. If the number does not exist, the query is ignored and `null` returned. Otherwise, the name is updated and the old name is returned.
- `String remove(int phoneNumber)`: The user tries to remove a contact with a given phone number from the phone book. If there is no such number, the query is ignored and `null` returned. Otherwise, the contact is removed and the name associated with the number is returned.

Input: Phone numbers are stored as integers. We assume that they have at most 7 digits and no leading zeros. Phone numbers are unique, i. e., no two contacts are associated with the same phone number. Names are non-empty strings of latin letters.

1. Download the code from Blackboard and create a new project.
2. Run and understand how the existing program works:
   - The file `PhoneBookDemo.java` is a framework to run and test your implementation. It provides a basic text-based user interface and instantiates the phone book (i. e., `HashPhoneBook` and `ListPhoneBook`). It also lets you load a sample phone book or start with an empty phone book.
   - The file `PhoneBook.java` is the above mentioned interface for the phone book implementation.
   - The file `ListPhoneBook.java` is an implementation of the `PhoneBook` interface using a linked list.
   - The text files are sample phone books that you can use for initialisation. Copy them into your main project folder (on the same level as the `src` folder). Do not rename the files.
3. **Part 1 (2%):** Create a class called `HashPhoneBook` that implements the `PhoneBook` interface and uses `java.util.HashMap` to store the contacts in the phone book. You need to decide on a suitable key-value entry model for contacts in the phone book and should read the documentation of the `HashMap` to identify suitable methods.
4. Test your implementation.

5. **Part 2 (0.5%):** Run some experiments to compare the performance of `HashPhoneBook` and `ListPhoneBook`. You need to explain the results of your experiments during the sign off.
6. When finished, have the work signed off and upload to Blackboard. Please only upload your solution for the assessed part of this worksheet and start a new project if you attempt the optional task.

Task 2 (optional)

In this task we will expand the phone book manager by allowing to also retrieve contacts by name. Create a new project using your implementation of Task 1 as a starting point.

Important note: We now additional assume that names are unique, i. e., there are no two contacts with the same name and each contact has only one associated phone number.

1. Extend the `PhoneBook` interface by adding the following methods that allow searching, update and deletion based on the name as key:
   - `Integer search(String name)`: The user looks for the phone number of a person. The method should return the number associated with the given name, or `null` if no such person exists.
   - `String remove(int phoneNumber)`: The user tries to remove a contact with a given phone number from the phone book. If there is no such number, the query is ignored and `null` is returned. Otherwise, the contact is removed and the name associated with the number is returned.
   - `Integer update(String name, int phoneNumber)`: The user tries to update the phone number for an existing contact in the phone book. If the name does not exist, the query is ignored and `null` is returned. If the name exists, but the number is already taken by another contact, the new number is returned. Otherwise, the phone number is updated and the old number is returned.
2. Add all three methods to the `ListPhoneBook` and `HashPhoneBook` classes and implement the required functionality. For hashing, one option is to have two Hash tables for the two different keys, however, you need to ensure that the list of contacts in both maps remains consistent when removing, adding or modifying entries. You will also need to modify the `PhoneBookDemo.java` file to account for the new queries and return useful error messages where appropriate.
3. Finally, you need to update `boolean add(int phoneNumber, String name)` and `String update(int phoneNumber, String name)` as follows to match the above functionalities:
   - The user tries to add a new contact to the phone book. If either the person or the number already exists in the phone book, the query is ignored and `false` is returned. Otherwise the new contact is added and `true` is returned.
   - `String update(int phoneNumber, String name)`: The user tries to update the name for an existing phone number in the phone book. If the phone number does not exist, the query is ignored and `null` is returned. If the phone number exists, but the name is already associated with another number, the new name is returned. Otherwise, the name is updated and the old name is returned.