Assignment 5

CSE 214

This document has 4 pages. Read everything before starting to code!
In this assignment, you will be implementing a very simple version of a **dictionary** data structure in Java, with the three main operations *insert*, *delete*, and *search*. For this, you will need the concepts we studied in class about direct address tables and how to interpret keys as integer values. For a data type to be interpretable as integer, it needs to be *hashable*. In Java terminology, we can ensure that a data type is hashable by making it implement the following interface:

```
/**
 * The <code>Hashable</code> interface provides the basic structure for any
 * data type that can be hashed. This means that any instance of the data type
 * should be interpretable as an integer. The <code>hash()</code> method of
 * this interface achieves this. For example, for a <code>String</code> data
 * type, the key of a string instance <code>s</code> should be an integer value
 * achieved by calling <code>s.hash()</code>.
 * @author Ritwik Banerjee
 */
public interface Hashable {

    /**
     * Converts an instance of a hashable data type to a non-negative integer.
     * @return the integer value.
     */
    int hash();
}
```

The next step is to define a dictionary structure in Java. This, too, is given to you as an interface:

```
/**
 * The <code>Dictionary</code> is an interface that provides the overall
 * structure for dynamic sets that support insert, delete and search.
 * @author Ritwik Banerjee
 */
public interface Dictionary<V extends Hashable> {

  /**
   * Checks whether or not the dictionary is empty.
   * @return <code>true</code> if the dictionary has no items,
   *         <code>false</code> otherwise.
   */
  boolean isEmpty();

  /**
   * The dictionary <b>insert</b> operation.
   * @param value the value to be inserted into the dictionary.
   * @throws java.lang.NullPointerException if the value is <code>null</code>.
   */
  void insert(V value);
  /**
   * The dictionary <b>delete</b> operation.
   * @param value the value to be removed from the dictionary.
   * @return the given value, if it was actually deleted from the dictionary,
   *         and <code>null</code> otherwise.
   * @throws java.lang.NullPointerException if the value to be deleted is
   *         <code>null</code>.
   */
  V delete(V value);

  /**
   * The dictionary <b>search</b> operation.
   * @param key the key so search for.
   * @return the value associated with the given key, or <code>null</code> if
   *         the key is not present in the dictionary.
   */
  V find(int key);
}
```

The code you will write builds on these two interfaces. The first task is to write a generic class with the following *signature*:
`public class DirectAddressTable<V extends Hashable> implements Dictionary<V>`
The methods in this class that are implemented based on the two given interfaces **must follow the rules specified by the interface documentations**. You can have extra features, of course, but you cannot violate the conditions already given (*e.g.*, if the interface documentation specified throwing an exception under some condition, your method must actually do so).
The aim of this assignment is to focus on direct address tables, so the actual data type will be very simple. This is the second task: to write a class `Alphabet` that is hashable. The instances of this class can only be lowercase letters from the English alphabet, *i.e.* $a, b, \ldots, z$. The documentation of your `hash()` method **must explain what it does and how it converts letters to integers**. Don't just repeat the interface's documentation!
Finally, use this `main` method to create a direct-address table of alphabets:

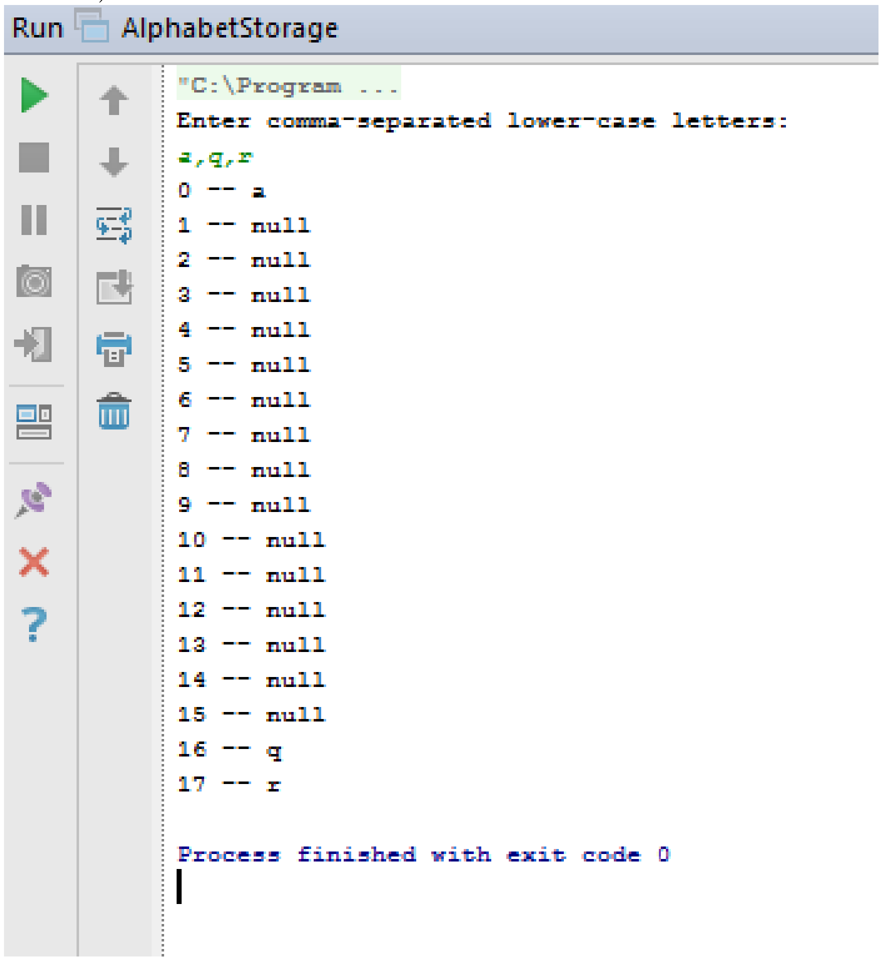
```
import java.io.*;

public class AlphabetStorage {
    public static void main(String[] args) throws IOException {
        DirectAddressTable<Alphabet> alphabetTable = new DirectAddressTable<>();
        System.out.println("Enter comma-separated lower-case letters:");
        InputStreamReader isr = new InputStreamReader(System.in);
        try (BufferedReader reader = new BufferedReader(isr)) {
            String input = reader.readLine();
            for (String s : input.trim().split(","))
                alphabetTable.insert(new Alphabet(s.charAt(0)));
        }
        System.out.println(alphabetTable.toString());
    }
}
```

NOTES

1. You are required to understand the workings of `InputStreamReader` and `BufferedReader`. There are many details that are beyond CSE 214, but you will be expected to be thorough with the basics. For the purpose of our syllabus, the following reading material is enough:
   (a) `http://tutorials.jenkov.com/java-io/inputstreamreader.html`
   (b) `http://tutorials.jenkov.com/java-io/bufferedreader.html`
2. You are required to write your own `toString()` method for the alphabet table. Just like the previous homework, an example input/output scenario is given below in Figure 1.
3. Remember, the three dictionary operations **must run in $\theta(1)$ time**.
4. Your homework will be tested with deletion and search operations, even though the given main method doesn't show that part.
5. When you submit the homework, **any code that is already provided in *this* document must remain unchanged**.

Figure 1: Sample input and output (the actual keys may differ according to your implementation of the `hash()` method):



```
"C:\Program ...
Enter comma-separated lower-case letters:
a,q,r
0 -- a
1 -- null
2 -- null
3 -- null
4 -- null
5 -- null
6 -- null
7 -- null
8 -- null
9 -- null
10 -- null
11 -- null
12 -- null
13 -- null
14 -- null
15 -- null
16 -- q
17 -- r

Process finished with exit code 0
```

Points Distribution	**Total: 50 points**
1. Implementation of the direct-address table class	35 points
2. Implementation of the alphabet class	10 points
3. Javadoc documentation of the `hash()` method	5 points

Submission Guidelines

1. Submit a single `.zip` file consisting of your `.java` files.
2. Remember to include the proper documentation in your comments in all files!
3. The `.zip` file that you submit should be named as `<SBUID>_<NETID>.cse214.hw5.zip`. For example, Mr. John Doe with student ID number 123456789 will submit the file: `123456789_jdoe.cse214.hw5.zip`.
4. Please make sure that your code compiles and can be run from the command line. **Code that does not compile will not be graded**. In this homework, the main method and sample input/output is already given. So no exceptions will be for uncompilable code.
5. Assignments will **not be graded if you submit the wrong files. No exceptions.** So, **double/triple-check what you are submitting!**

Submission Deadline

The due date for this assignment is midnight of **Sunday, Nov 29, 2015**.