The Map interface

One commonly uses a data structure that implements the `Map` interface in Java to store items of information that are associated with and indexed by a range of keys. Sometimes the key is a sort of identification number; for example, a library’s on-line catalogue might be represented as a `Map` in which each key is a Library of Congress number (`QA76.73.J38 A76 1996`, say) and the associated item is a record giving full information about the book. In other cases, the key is an object, and the associated item is simply a value that expresses a property of the key that, for whatever reason, isn’t regarded as sufficiently essential to justify giving it a field within the object. For instance, a social linking site might use a `Map` in which each key is a URL and the associated value is the number of site members that have designated that URL as worthy of attention.

1. In the examples described above, what classes might the keys and associated values belong to?

2. Review the API documentation for the `TreeMap` and `HashMap` classes, both of which implement the `Map` interface. How do they differ? Which one should you use for the above examples?

Anagrams

Two character strings are anagrams if they contain the same characters (and the same number of occurrences of each character), possibly in a different order. For instance, `placer` and `parcel` are anagrams.

In this lab, we’ll do a few simple exercises involving anagrams. In Eclipse, create a project and a new class `Anagrams` for the code you write today.

To find anagrams efficiently, it’s often helpful to associate a signature, with each string, that inventories the string’s characters in a systematic way. One common way of preparing a signature is to sort the characters that make up the string.

3. (2 points) Write a (static) method that takes a `String` as argument and builds and returns a `String` containing the same characters, including repetitions, but arranged in ascending order. So, for example, given the `String` "creative", the method should return "aceeirtv".

For arranging characters in the ascending order, you may first convert the `String` to a character array and then sort that array.

Test this method to ensure that it returns the correct signature for the given string.

4. Two strings are anagrams if they have the same signature. Write a (static) method that takes two `Strings` as arguments, computes and compares their signatures, and returns a boolean indicating whether the given strings are anagrams.

Finding anagrams in a word list

If we have a collection of strings (such as the word list in the file `/usr/share/dict/words` on MathLAN workstations), we can identify any anagrams that occur among them by computing the signature of each string and using that signature as the key in a `Map`. The value associated with such a key would be a list—a `LinkedList`, perhaps—containing the strings that have that signature. (For instance, the value associated with the key "acelpr" would be a list containing both "placer" and "parcel").
5. What mapping structure should you use to associate values to keys for the anagram application? What are the types of keys and values in this structure?

6. Suppose that a class called `AnagramFinder` has such a map as one of its fields. Write the declaration for this field and initialize it to an empty map. Write a method, called `addToMap`, for the `AnagramFinder` that takes a string as argument, computes the signature of that string, and adds that string to the list associated with that signature in the map. Note that you’ll have to distinguish two cases: If the signature is already a key of the map, you should simply add the new string to the list associated with key (signature) and then update that key-value combination. But if the signature is not yet a key, you’ll need to create a new list, with the string as its only element, and extend the map by adding that key-value combination to it. Finally, you return from the method.

7. (2 points) Write the definition for an `AnagramFinder` class. Its `main` method should open a text file named on the command line using `args`, read in the text one line at a time, and call the method you defined in the previous exercise to add it to the map. When the end of the text file is reached, the `main` method should report all the anagrams it has collected by iterating through the map, checking each value to see whether the list of strings with the same signature has two or more elements, and printing out the contents of any list that meets this condition.

To iterate through the map, you may first obtain the `Set` of keys of the map. Then, iterate through the key set to get values for each of the keys.