Telling a Random Story

Using and Improving GladLibs

Introduction 7 min
Brittle Code 3 min
Adding New Labels 4 min
Programming Exercise: Using GladLibs 10 min
HashMap 7 min
HashMap for Unique Words 4 min
HashMap for Flexible Design 7 min
Summary 3 min
Programming Exercise: Improving GladLibs 10 min
Practice Quiz: Using and Improving GladLibs 10 questions

Review

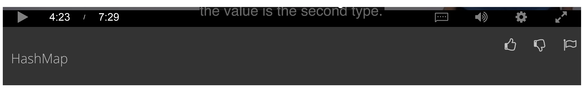
When defining a HashMap variable, you must specify the types of both the key and the value.

- True

Correct

- False

Continue



Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:03
Hi it's time to learn about a new way of structuring data. This also helps in structuring classes. You've seen some of the concerns with the design of the GladLib class. The design has flaws but you still were able to add a new label and create new stories, so the design works.

0:21
Larger programs with the same flaws might be harder to modify. You've also seen code account nucleotides in a strand of DNA and letter frequencies to break a Caesar Cipher. We extended this idea to counting word frequencies in any file with two parallel arrays. To count nucleotides we used four int variables to count each time a C, G, T, or A occurred. To count letter frequencies we used an array of 26 int values. One for each letter A through Z.

0:53
To count words you saw code that used two parallel ArrayLists. These contained as many values as there are different words being counted. Now you'll see how to use the concept of parallel ArrayList, to help understand the java.util.hashmap class. A new way to structure data.

1:11
Using HashMap we'll make it much easier to modify the GladLib class. The HashMap class is also very efficient, compared to using two parallel ArrayLists. We'll explain the HashMap class using the parallel ArrayList class to help. We've had a chance to see this code that uses two ArrayLists to count how many times each different word occurs in a file. The method .indexOf finds the location of a word in the ArrayLists of stings named myWords.

1:41
If the value returned by the .index sub is minus one then the string read from the file hasn't been seen before and it's stored in my words with a corresponding count of one in myFreqs. If the string has been seen before, the integer value in myFreqs at the same index is incremented by one. This code works, but, using a HashMap will make the code much faster. And the HashMap class will make it easier to modify the GladLib class as well. Let's work to understand the concepts in the HashMap class.

2:13
A HashMap object associates keys with values. In many languages this is called a map. You might think of the key or legend in a map as you can see here. The key helps in understanding the map. You can look up the color in the key or legend. And understand what the color means in the map. In programming the concept is more mathematical than geographical. The word map has meanings in both Math and Geography.

2:40
The key in an element in the domain that's being mapped to a value in the range. In math, a function is sometimes called a Mapping, and this expresses the ideas in the HashMap class as well.

2:52
In a HashMap, you look up a key, and you get the value associated with the key.

2:58
In an example illustrated here, we are counting word frequencies. The word rainbow occurs 41 times. So the integer value 41 is associated with the string key rainbow. in Java the value returned by the call map.get("rainbow") is 41. The value returned by map.get("truth") is 17, indicating the string truth occurs 17 times.

3:21
A football occurs 23 times then 23 is the value associated with the key football as we turn by the call map.get("football"). Wonderful occurs 23 times in our hypothetical example. The keys in the map are unique but it's possible to have the same value associated with different keys. To use a HashMap you'll need to understand the operations it supports. We'll use one HashMap to replace the two parallel ArrayLists. Here is the code that uses one ArrayLists of strings and one ArrayLists of integers with the .index of function used for associating integer values with strings.

3:56
The HashMap code also associates an integer value with the string key. The key in the HashMap will be a string. The value is an integer. That's the number of times the string occurs in a file being processed.

4:09
To define a HashMap variable, you'll need to specify the type of both the key and the value.

4:15
When calling new, you must specify the key and the value types as well. The key is the first type and the value is the second type. The code will determine if the key has never been seen before, whether it's stored in the HashMap or not. The method .containsKey on the map object returns a Boolean indicating whether that key is in the map. If it's not in the map, the value 1 is put in the map with the key, using the map method .put. If the string is a key in the map, meaning the string being processed has been seen before. The value it shows you if the key is found using the .get method and then a new value is stored for the key by calling map.put, with an updated value of one more.

5:00
In addition to accessing individual map keys and values, we also need to access all the keys and values in a map. Printing the strings and frequencies when parallel arrays are used requires a typical indexing for loop with the index used to access both strings and integers as shown here. Printing all the key value pairs in the map requires looping over all the keys, and getting the value associated with each key.

5:25
The method .keySet returns an interval you use to access each key in a loop. This is similar to using .words or dot .lines with the file resource or URL resource to access elements. Or the .data method to access each string in a storage resource.

5:42
Iterating over the key set and calling get for each key allows you to print the contents of a map.

5:48
Maps will allow you to modify the GladLib class more easily. But maps are incredibly fast too. When files are large, efficiency matters more than when they're small. This concept of using efficient structures or code is important. Computers are so fast that simple concepts lead to code that's fast enough given how fast computers are today.

6:09
As you can see in the first row of the table, counting how many times each word occurs in Shakespeare's Julius Caesar, it's fast enough on a laptop computer, even using the parallel ArrayList.

6:21
With a bigger file like the sayings of Confucius, the code for ArrayList is still under half a second. Whereas the HashMap code is incredibly fast.

6:30
For the novel the Scarlet Letter, there are many more unique words and total words. But the HashMap code is still roughly ten times faster than the ArrayList code.

6:41
For a large file, like the King James version of the Bible with over 800,000 words and 32,000 different words, the ArrayList code takes more than 20 seconds, while the HashMap code is under half a second. That's more than 40 times faster.

6:57
Looking up keys in a map takes time, that's independent of the number of keys. Roughly speaking this means getting the value for a key in a map with a million keys, takes the same time as looking up the value associated with the key in a map of just ten keys. With an ArrayList you might have to look at all million elements in the ArrayList, so HashMap is incredibly fast.

7:19
In later courses you'll study how the HashMap class can be so fast.

7:24
In this course you'll use HashMaps in many examples. Happy coding.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?