COP 3330 – Object Oriented Programming in Java
Java Class: HashMap

HashMap

The difference between a map and a set is that instead of just storing a set of objects, each object in the set is stored with an associated value. Thus, a map is a set where each object in the set is paired with an associated value. We call the distinct objects for the map the keys and the values that each key is associated with values. In Python, the name given to a map is a dictionary because in a dictionary you have a set of words and each word is paired with its definition. (Now the analogy isn't perfect because actual words often have multiple definitions whereas in a map in Java each key can only be paired with precisely one value. One way around this would be to make the value for an associated key a list of some sort.)

Just like a set, we should be able to add, remove and search for keys. When we perform a search we should retrieve the associated value with the key we searched for. At any point in time, we should be able to change the value that a key is associated with (and this is very common).

Before we look at the Java, let's look at Python's syntax for a map. It's syntax is so intuitive that it's probably the best way to explain what a map is.

Here is a code segment in Python that uses a map:

```
age = {}
age["denise"] = 30
age["paola"] = 15
print(age["denise"])
print(age["paola"])

age["paola"] = age["paola"] + 1
print(age["paola"])

age["daniel"] = 23
for x in age:
    print(x, age[x])

del age["paola"]
if "paola" in age:
    print("paola is ",age["paola"],"years old.")
else:
    print("paola's age is unknown")
```

In short, we can think of a map as an array that can be indexed by any object. We can add an entry to the map (assignment), remove an entry (del statement above), search for an entry (just using the bracket operator or an in operator) or change an entry (assignment).

Now, let's look at the most important methods in the HashMap class:

```
// Clears the contents of this HashMap
void clear()

// Returns true if and only if this HashMap contains the key
// key.
boolean containsKey(Object key)

// Returns the associated value to key in this HashMap. If no
// such key is in the map, null is returned.
V get(Object key)

// Returns the set of keys in this HashMap.
Set<K> keySet()

// Maps key to value in this HashMap.
V put(K key, V value)

// If key is a key in this HashMap, the matching value
// is returned and the key-value pair is removed from the
// map. If no such key exists in the map, null is returned and
// no further action is taken.
V remove(Object key)

// Returns the number of mapped pairs in this HashMap.
int size()
```

TestHashMap Example

In this example, we'll just test out several of the HashMap methods to see how they work instead of solving some fixed problem. Instead of showing the whole code which is posted separately, we'll just look at each short segment of code and explain its output and what it does.

Our first section of the example creates a map of keys that are of type String to values that are of type Integer. Here is the call to the constructor:

```
HashMap<String,Integer> numSibs = new HashMap<String,Integer>();
```

The sytnax here is similar to the HashSet syntax, except the generic parameters in between the brackets have two types separated by a comma.

Now, let's add an entry (note, I actually have 2 siblings...):

```
numSibs.put("Arup", 3);
```

Here is the way to look up the number of siblings I have:

```
Integer ans = numSibs.get("Arup");
System.out.println(ans);
```

If the key is in the map, an Integer objects is returned, otherwise null is returned. I can reduce this to one line as follows:

```
System.out.println(numSibs.get("Arup"););
```

Either of these two code segments will print out 3.

A good question to ask is: what would happen if the key wasn't in the map, say we did something like this:

```
int sibs = numSibs.get("Bob");
```

This will crash the program. A safer way to do this is:

```
Integer ans2 = numSibs.get("Bob");
if (ans2 == null)
    System.out.println("There's no entry for Bob.");
```

As expected, running this will print out, "There's no entry for Bob."

Now, let's overwrite my entry with the correct number of siblings and prove that the map has changed.

```
numSibs.put("Arup", 2);
System.out.println(numSibs.get("Arup"));
```

As expected, this prints out 2.

Before we show how to go through each map entry, let's add a few more entries to the map:

```
numSibs.put("OnlyChild", 0);
numSibs.put("Jace", 1);
numSibs.put("BigFamily", 13);
```

Now, let's search for several keys in this map. Here is a declaration of what we'll search through:

```
String[] keys = {"Janice", "Arup", "Bob", "OnlyChild", "onlyChild",
"BigFamily", "Bruno"};
```

Now, let's search for each of these keys in the map:

```
for (String k: keys) {

    // Safe to look up k in map.
    if (numSibs.containsKey(k))
        System.out.println(k+"has "+numSibs.get(k)+" brothers and sisters.");

    // k's not there so I print an error message.
    else
        System.out.println("I have no information about "+k+".");
}
```

The output of this code segment is:

```
I have no information about Janice.
Arup has 2 brothers and sisters.
I have no information about Bob.
OnlyChild has 0 brothers and sisters.
I have no information about onlyChild.
BigFamily has 13 brothers and sisters.
I have no information about Bruno.
```

Thus, the standard way to guard against a run time error is to use the containsKey method before looking up a particular key in a map.

Now here's one way to loop through each pair in the map once using the keySet() method. In this example we add up the total number of siblings and the total number of people involved in in the sibling pairs, assuming that each person in the map is unrelated:

```
int total = 0;
System.out.println("\nFull listing of the map.");

for (String k : numSibs.keySet()) {
    System.out.println(k+" has "+numSibs.get(k)+" brothers and sisters.");
    total += numSibs.get(k);
}

System.out.println("The total # of sibs is "+total);
System.out.println("The total # of people is "+(total+numSibs.size()));
```

We can use an iterator loop through the set of keys in the map. Inside the loop, k will be the reference to each k in the map once. In this manner we can add up everyone's siblings. At the end, we need to add each person (the keys) to the total person count. The size() method returns this value.

Finally, let's remove the big family and reprint the map contents:

```
numSibs.remove("BigFamily");

// Proving removal.
System.out.println("\nReprint after removal.");
for (String k : numSibs.keySet())
    System.out.println(k+" has "+numSibs.get(k)+" brothers and sisters.");
```

Here is the output generated by the final for loop:

```
Reprint after removal.
Arup has 2 brothers and sisters.
Jace has 1 brothers and sisters.
OnlyChild has 0 brothers and sisters.
```

The entry for BigFamily is not part of the map any more.

Kattis Problem: Unbearable Zoo

The problem statement is here:

`https://open.kattis.com/problems/zoo`

A summary of the problem is that there is a list of several animals in the zoo and we want to produce a list in alphabetical order of each animal and how many of them there are. One slightly annoying portion of the problem is that each animal name can be several tokens long, but we just require the last token. Thus, we must read the input line by line and use a StringTokenizer to parse out the last token.

Finally, when we print, since HashMaps are unordered, we have to add each key to a list, sort that list and then go through the keys in the order of the sorted list.

Here is a portion of the code that creates the map, reads in the input and stores the final information in the map. Note that this code is inside of a loop which process multiple cases.

```
HashMap<String,Integer> map = new HashMap<String,Integer>();
for (int i=0; i<n; i++) {
    StringTokenizer tok = new StringTokenizer(stdin.nextLine());

    while (tok.countTokens() > 1) tok.nextToken();

    String animal = tok.nextToken().toLowerCase();

    if (map.containsKey(animal))
        map.put(animal, map.get(animal)+1);
    else
        map.put(animal, 1);
}
```

The while loop inside of the for skips over the tokens in the animal name we don't need. Then, we get the animal. Note that we have to deal with case issues and since our output is lowercase, we just change the animal name for the map to all lower case.

After that, the classic way to keep a count of items with a map is to either add a new entry mapping to 1 if no such entry exists, or add 1 to the entry. The syntax here isn't nearly as clean as Python, but it adheres to the strict rules for calling instance methods in Java.

Now, we must retrieve all the keys and add them to a list, then sort the list. Here is that code:

```
ArrayList<String> keys = new ArrayList<String>();
for (String s: map.keySet())
    keys.add(s);
Collections.sort(keys);
```

Here we use the keySet() method to obtain the set of keys, then add them to our list. Finally we sort the collection via Collections.sort.

To obtain our output, we can now just loop through the keys as they are ordered in the list keys:

```
System.out.println("List "+loop+":");
for (String s: keys)
    System.out.println(s+" | "+map.get(s));
```

In terms of run time, the slowest part of the code is the sort. which takes O(mlgm) time, where m is the number of unique entries in the map. The rest of the code, except for the sort, runs in O(n) time, where n represents the number of lines of input. In the worst case, m = n. In cases where there are only a few unique entries in the map, the run time is dominated by the O(n) factor.

Graph Storage Example

A very important structure in algorithms classes is a graph. A graph is a collection of objects of which, some pairs are connected. (We can think of people as the objects and the pairs who are connected are people who are friends. Facebook induces a graph structure between all accounts on the site, for example.)

In this example, we'll read in a list of connections between people and create a graph structure from it. We'll assign each unique person an identifier number and then to store the graph, we'll use a list of lists. graph[i] will store the list of integers that are the id numbers of all of the people that person i is friends with.

Here is the sample input for our test:

```
8
Anna Bob
Anna Carmen
Dave Eddy
Eddy Carmen
Joanna Khoa
Martin Sam
Carmen Dave
Khoa Sam
```

As we read in the pairs, we'll do something similar to our previous example, we'll check if the entry is in the map and if not, we'll add the mapping between the person and their new id number. To maintain the id number, we'll start a variable at 0 and increment it each time we add a new person to our map.

Here is the section of the code before we start reading in the pairs that sets up all of our data structures:

```
Scanner stdin = new Scanner(System.in);
int n = stdin.nextInt();

ArrayList<ArrayList<Integer>> graph = new ArrayList<ArrayList<Integer>>();

int id = 0;
HashMap<String,Integer> idLookUp = new HashMap<String,Integer>();

ArrayList<String> nameList = new ArrayList<String>();
```

We can create a list of something complicated, such as list. This is how to call the constructor appropriately to cerate a list of lists.

Then we set up our HashMap that will map each person to their assigned id number. Finally, for each unique name, we'll store the name in a list at the index that corresponds to their assigned id number.

Here's the code that reads in the pairs and maintains these structures:

```
for (int i=0; i<n; i++) {

    // Get the two names.
    String name1 = stdin.next();
    String name2 = stdin.next();

    // Add to map if necessary.
    if (!idLookUp.containsKey(name1)) {
        nameList.add(name1);
        idLookUp.put(name1, id++);
        graph.add(new ArrayList<Integer>());
    }

    // Do same for name two.
    if (!idLookUp.containsKey(name2)) {
        nameList.add(name2);
        idLookUp.put(name2, id++);
        graph.add(new ArrayList<Integer>());
    }

    // Get the ids for both people.
    int u = idLookUp.get(name1);
    int v = idLookUp.get(name2);

    // Add v to u's list and u to v's list.
    graph.get(u).add(v);
    graph.get(v).add(u);
}
```

Once a name is read in, we need to see if we've seen the name before. If we haven't we need to add the name to our name list and add a map from the name back to its ide and also create a new list in our graph for the friends of the new person added to the graph. Once we do this for both names, then we just need to retrieve the id's of both people and add two connections to our graph, one from person with id u to the person with id v, and then another connection from person with id v to the person with id u. (There are graphs where we don't automatically store both links, but in this example, "being friends" is assumed to be a symmetric property.)

To show how everything is stored take a look at the output produced by this code:

```
// Print names in order.
System.out.println("Name list in order.");
for (int i=0; i<nameList.size(); i++)
    System.out.print(nameList.get(i)+" ");
System.out.println("\n");

// Print map.
System.out.println("Map contents");
for (String person: idLookUp.keySet())
    System.out.println(person+" "+idLookUp.get(person));
System.out.println();
```

```
// Print graph.
for (int i=0; i<graph.size(); i++) {
    System.out.print("vertex "+i+": ");
    for (Integer x: graph.get(i))
        System.out.print(x+", ");
    System.out.println();
}
```

The first portion of the code prints the unique names in the order they appeared in the input. The second portion of the code prints out each name and the corresponding mapped id. Finally, the last portion prints out each person's id number, listed with the id's of every person they're friends with.

Here is all of the output produced by the program:

```
Name list in order.
Anna Bob Carmen Dave Eddy Joanna Khoa Martin Sam

Map contents
Carmen 2
Bob 1
Eddy 4
Joanna 5
Martin 7
Dave 3
Khoa 6
Sam 8
Anna 0

vertex 0: 1, 2,
vertex 1: 0,
vertex 2: 0, 4, 3,
vertex 3: 4, 2,
vertex 4: 3, 2,
vertex 5: 6,
vertex 6: 5, 8,
vertex 7: 8,
vertex 8: 7, 6,
```