**Morse Code**

The HashMap class (in the java.util package) provides the implementation for another convenient "collection" of items, similar to (but significantly different than!) an ArrayList. A HashMap is essentially a **dictionary** – it contains *unique* **keys** that are "mapped" to some **value**. In a real dictionary, the key is a word and the value is the definition of that word. In a HashMap, the keys and values can be any type of object!

If you're unsure of what methods the HashMap class offers, use the class' API and our [website](http://bit.ly/mrbunnCS).

1. Create a new class called HashMapProbs with a main() method.
2. In main(), create a new HashMap object called animalSounds*.* Add four key / value pairs (as Strings) of your choosing. For example: "Dog" would be a key, which is mapped to the value "Woof". When done, print the HashMap (this can be done directly, as the HashMap class has an overridden toString() method).

/\* Like ArrayList objects, HashMap objects should be *parameterized.* Example: HashMap<String, Integer> map = new HashMap<>(); \*/

1. Get the value of a String called strfrom the keyboard, and return the value in animalSoundsmapped to str(should print null if it doesn't exist as a key in the map).
2. Print the number of key/value pairs in animalSounds.
3. Get the value of Strings newAnimal and newSound from the keyboard, and add them to the HashMap. Try this with values for newAnimal and newSound that already exist in the HashMap, and don't exist in the HashMap. What do you notice?

/\* If the map previously contained the key, the old value will be replaced. The put() method will also return the previous value associated with the key, or null if it didn't exist. This is a really important feature of HashMaps! \*/

1. (Riddle) 4 = F on M R (a "language equation" riddle, e.g. 24 = H in a D is "24 hours in a day")
2. Write a public HashMap<String, String> mapBully(HashMap<String, String> map) method that will modify and return the given map as follows: if the key "a" has a value, set the key "b" to have that value, and set the key "a" to have the value "". Basically, "b" is a bully, taking the value and replacing it with the empty string. The ordering of the key / value pairs in the output is irrelevant.

//Given a HashMap *map* containing the following: {"b": "dirt", "a": "candy"}

mapBully(map) >>> {"b": "candy", "a": ""}

1. Write a public HashMap<String, String> mapShare(HashMap<String, String> map) method that will modify and return the given map as follows: if the key "a" has a value, set the key "b" to have that same value. In all cases, remove the key "c", leaving the rest of the map unchanged.

//with a HashMap *map* containing the following: {"b": "bbb", "c": "ccc", "a": "aaa"}) mapShare(map) >>> {"b": "aaa", "a": "aaa"}

1. Using the text file "**dream.txt"**, write a method to print the word (and its value) with the highest frequency. For the sake of simplicity, convert all words to lower case and don't worry about punctuation (i.e. "hello" and "hello!" would be two different words). A word is any 'token' separated by whitespace. This must be done using HashMaps! Some methods that will help:

/\* Collections.max(map.values()) will return the largest **value,** assuming your HashMap is called *map* (the *values()* method returns a list of all the values in the map)

You can iterate through all the keys in the map, to find the highest frequency word, with the following:

for (String word : map.keySet())

where the *word* variable will be all the key Strings in the map \*/

**Smarter Morse code conversion**

If you took PAP / CS 1 with Mr. Bunn, you might have coded a Morse code translator before. However, encoding and decoding was tricky, and the algorithm didn't work the same both ways. A HashMap would work much better! Create a program that can perform Morse code encryption / decryption. Your class should have the following (copy/paste):

Instance variables

* private final String alphabet = "abcdefghijklmnopqrstuvwxyz0123456789 ";
* private final String[] morse = {".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--..", ".----", "..---", "...--", "....-", ".....", "-....", "--...", "---..", "----.", "-----", "|"};
* private HashMap<String, String> toText
* private HashMap<String, String> toCode

Methods

* public MorseCode()
* public String encode(String s)
* public String decode(String s)

Your class' constructor should initialize the HashMap objects, then populate them with values from alphabetand morse(provided for you, I hope you're using loops for this!)*.* In a main() method of a Runner class, you should make an object of your class and test its methods.

*Note that each Morse code "letter"* ***should be separated by a space****, and an actual space character is represented with a vertical line (this: | is the vertical line character, referred to as a "pipe"). Without the space, how you would determine if four dots (....) was four 'e' characters or a single 'h'?*

The two HashMap instance variables are included to make converting back and forth easier. You could complete this exercise with one HashMap – it is possible to 'reverse' the process by iterating over a Set containing all the key / value pairs, though this process involves a couple advanced concepts.

MorseCode m = new MorseCode();

m.encode("hello world") >>> .... . .-.. .-.. --- | .-- --- .-. .-.. -..

m.decode("--- -- --. | .. - | .-- --- .-. -.- . -.. ") >>> ???\*

*\* If you're copying the test string from Dropbox's web preview, it might be pretty close to a phrase you recognize, but not quite. Download the lab document and copy/paste the string from the downloaded Word doc and it should work.*

**(Advanced) Finding a majority element**

Given an integer array of size n, find all elements that appear more than n/3 times. The algorithm should run in O(n) time (should not use nested loops).

**(Advanced) Employee hierarchy**

You are given a dictionary (HashMap) that contains the mapping of an employee and the employee's manager as a number of {employee, manager} pairs like below:

{"A", "C"},

{"B", "C"},

{"C", "F"},

{"D", "E"},

{"E", "F"},

{"F", "F"}

In this example, C is manager of A, C is also manager of B, F is manager of C and so on.

Write a method to get the number of employees under each manager in the hierarchy (not just who they directly report to). It may be assumed that an employee directly reports to only one manager. In the above dictionary, the CEO (root node) is considered to report to herself.

Output should be a dictionary (HashMap) that contains following:

{{A, 0}, {B, 0}, {C, 2}, {D, 0}, {E, 1}, {F, 5}}

**(Advanced) Cube permutations**

The cube, 41063625 (3453), can be permuted to produce two other cubes: 56623104 (3843) and 66430125 (4053). In fact, 41063625 is the smallest cube that has exactly three permutations of its digits, which are also cubes. **Find the smallest cube for which exactly five permutations of its digits are cubes.**

//A helper class called Cube may be useful here