**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 HashMaps the keys and values can be any type of object!

If you're unsure of what methods the HashMap class offers, use the powerpoints or the Java [documentation](https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html).

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.

/\* 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 method public static HashMap<String, String> takeBefore(String a, String b) that will return a HashMap containing all the characters in a and b in the following fashion: the character at index i that comes earlier in the alphabet should be the key, and the element that occurs later in the alphabet should be the value. Call from main to test.

You can assume the strings will be the same length and contain unique elements. Also, recall that you can compare primitive chars with relational (e.g. <) operators (will use their ASCII value).

takeBefore("str", "bye") >>> {b=s, t=y, e=r}

1. Write a method public static HashMap<String, Boolean> multiple(String s) where each different character in s is a key and its value is true if that characters appears two or more times in the supplied string. Recall that Java auto-boxes primitive types as their corresponding wrapper class.

multiple("hello") >>> {e=false, h=false, l=true, o=false}

1. Write a method public static HashMap<String, String> charWord(String[] words) that returns a HashMap with a key for every different first character seen; the value will be all strings starting with that character, appended together in the order they appear in the array.

charWord(new String[] {"tea", "salt", "soda", "taco"}) >>> {s=saltsoda, t=teataco}

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!

Now that the HashMap is populated, you need to locate the word (key) with the highest frequency (value). Replacing map with the name of your HashMap, use the following code to help you locate the highest frequency word (iterating "backwards" through a HashMap requires a couple concepts you haven't learned yet):

int max = Collections.max(map.values()); //the largest *value* in the map

//we will assume there is only one maximum

for (String key : map.keySet()) //iterate through all the keys in the map

if (map.get(key) == max) //if this key's value matches max...

System.out.println("Highest frequency word: " + key + ", " + max);

**Smarter Morse code conversion**

If you took PAP/CS 1 with Mr. Bunn, you might have coded a Morse code translator before. However, the process was tricky, and the algorithm didn't work the same both ways.

A HashMap is a natural choice for this process; each character is mapped to its Morse equivalent and vice-versa. The order of the <key, value> pairs in the map is irrelevant, all that matters is the corresponding characters are linked together.

Write a program that can convert text to/from Morse code. Your class should have the following (copy/paste):

|  |  |
| --- | --- |
| **Variables** | private final String **alphabet** = "abcdefghijklmnopqrstuvwxyz1234567890 ";  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 and then populate them with values from alphabetand morse(provided, I hope you're using loops for this!)*.*

***Note:*** *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.*

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 know if four dots (....) is four 'e' characters or a single 'h' character?

In the main method of a runner class, make an object of your class and test its methods:

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* 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) Employee hierarchy**

You are given a dictionary (HashMap) that contains the mapping of an employee and the employee's manager as a series 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 HashMap that contains following (the ordering of elements in the map is irrelevant):

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