Maps Introduction

The map data structure stores pairs of keys and values. Keys can never be repeated, but values can. Think of a function in math. *f(x)* is a function that **maps** inputs (keys) to outputs (values).In a function, an input can never map to multiple outputs (it needs to pass the vertical line test). Meanwhile, the same output can be generated from different inputs.

A good use of a map is a dictionary, which maps a word in one language (key) to another word from a different language (value).

Map: 5 → true 6 → false -9 → true	Not a Map: "Green" → 12.5 "Blue" → -4.0 "Green" → 6.5

Basically, each key should only appear once!

HashMap and TreeMap

In Java, maps are made as the interface `Map<K,V>`, where K and V are two type parameters (generics). Maps are implemented by **HashMap** and **TreeMap**.

Notice K and V must be classes. If you need to use primitive data types, instantiate the map with the wrapper class.
Ex:

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

HashMap	**TreeMap**
Elements may be arranged randomly Null is a valid value for a key	Elements are in sorted order by Comparable / Comparator Null is not a valid value for a key

Methods for Map

`get(key)` -- returns the value for a key, or `null` if that key is not in the map.

`put(key, value)` -- stores a new key/value pair in the map, overwriting any existing value for that key.

`containsKey(key)` -- returns `true` if the key is in the map, otherwise `false`.

`remove(key)` -- removes the key-value pair if the key is in the map. Returns the value in the key-value pair, and returns `null` if key was not in the map

`keySet()` -- returns a `Set<K>` of the keys in a `Map<K,V>`. Using a set makes sense because a key can appear at most once.

`values()` -- returns a `Collection<V>` of the values in a `Map<K,V>`. Using a set does not work because it is possible that a certain value shows up multiple times - this makes using a collection better.

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Map<Integer, Exam> finals = new
TreeMap<Integer, Exam>();
        finals.put(5, new Exam(3, false));
        finals.put(5, new Exam(2, true));
        System.out.println(finals.get(2));

        for(int i = 0; i < 4; i++) {
            finals.put(i, new Exam((i % 3) +
1, false));
        }
        System.out.println(finals.values());
        System.out.println(finals.remove(5));
        for(int a: finals.keySet()) {
            System.out.print(a + " ");
        }
    }
    private static class Exam {
        int hours;
        boolean curve;
        public Exam(int h, boolean c) {
            hours = h;
            curve = c;
        }
        public String toString() {
            return hours + " " + (curve ?
"t":"f");
        }
    }
}
```

```
null
[1 f, 2 f, 3 f, 1 f, 2 t]
2 t
0 1 2 3
```

Iteration through a Map

Use `keySet()` and then a for-each loop to iterate through.

Example: add 1 to each value in all key-value pairs of a Map<Integer, Integer>.

```
for(int a: m.keySet()) {
            m.put(a, m.get(a) + 1);
        }
```

Frequency Map

A common application of maps is to count the frequency of elements.
Ex: Print out a frequency map of certain words in a string. Print the words in alphabetical order.

```
public void frequencyMap(String text) {
      String[] words = text.split(" ");
      Map<String, Integer> map = new TreeMap<String, Integer>();
      for(String word: words) {
            if(map.containsKey(word)) {
                  map.put(word, map.get(word) + 1);
            }
            else {
                  map.put(word, 1);
                  }
      }
      System.out.println(map);
}
```

`frequencyMap("hi hi bye hi bye hi ");`	{bye=2, hi=4}
`frequencyMap("like like dislike like like love");`	{dislike=1, like=4, love=1}
`frequencyMap("if it's on the test, it's in this book");`	{book=1, if=1, in=1, it's=2, on=1, test,=1, the=1, this=1}