DATA STRUCTURES : Hashmap

* hashmap
* objects
* serializable

We discussed hashmaps in class today.

- Hashmaps have a key – value pair.
- Both the key and value are objects (in Java).
- Hashmaps are not sorted or ordered in anyway, but they allow for rapid retrieval of data as long as you know the key.
- The key should be immutable (eg. a String or Integer) otherwise you won't be able to retrieve the value if the contents of the key change.

Hashmap methods:

Assume that you have a hashmap called "map"

```
map.put(key, value);
```
//This adds a key and value. It replaces existing value if any

```
map.putIfAbsent(key, value);
```
//This prevents overwriting an existing key

```
String value = map.get(key);
```
//This gets the value, given a key

```
map.remove(key);
```
//this removes the key and value

```
map.size();
```
//how many elements are in the hashmap

```
map.clear();
```
//erases contents of hashmap

```
map.containsKey(key)
```
//True or False

```
map.containsValue(value)
```
//True or False. This operation is slow as it has to read every entry.

`.keySet()` //retrieve all of the keys into a set

```
for (String k : map.keySet()) {
  //print all keys (here we assume they are strings)
  System.out.println(k);
  //you could print all values too.
  System.out.println("key: " + k + " value: " + map.get(k))
}
```

`.values()` //retrieve all of the values into a set

```
// Print values
for (String v : map.values()) {
  System.out.println(v);
}
```

`.entrySet()` //seems to be another way of getting all keys and values

```
for (Map.Entry<String, Integer> e : map.entrySet()) {
     System.out.println(
          "Key: " + e.getKey() + " Value: " + e.getValue());
}
```

`.forEach()` -- lambda functions. *I haven't used this.*

Immutable Objects

All primitive wrapper classes (Integer, Byte, Long, Float, Double, Character, Boolean, and Short) are immutable in Java, so operations like addition and subtraction create a new object and not modify the old.
The below line of code in the modify method is operating on wrapper class Integer, not an int, and does the following as described below as follows:

```
Integer ii = new Integer (12);
ii = ii + 2;
```

When you print ii, it will print 13. How can this be?

Here's what happens:

- Unbox 'ii' to an int value
- Add 2 to that value
- Box the result into another Integer object
- Assign the resulting Integer to 'ii'
- This changes what object 'ii' references – it's a different object, but it's quite hard to prove this.

Any object you use as a **key** must have a .equals() that says whether the objects are equal, and a .hashcode() function too:

```
@Override
public int hashCode() {
    return Objects.hash(key.var1, key.var2, ...);
}
```