Sets and Maps

Outline

1. Sets
2. Foreach Loops
3. Maps

What is a Set?

Definition (Set)
A set is an unordered collection of unique values. You can do the following with a set:
- Add element to the set
- Remove element from the set
- Is element in the set?

How To Think About Sets
Think of a set as a bag with objects in it. You’re allowed to pull things out of the bag, but someone might shake the bag and re-order the items.

Example Set
“such strings”
“much wow”
“goodbye”
“very hello”

Is “goodbye” in the set? true
Is “doge” in the set? false

Set Implementations

Set is an interface in java.util; implementations of that interface are:

- TreeSet
  - Really fast
  - Does maintain the elements in sorted order

- HashSet
  - REALLY REALLY fast
  - Does not maintain a useful ordering

Alice in Wonderland

Count the Number of Distinct Words in a Text
Write a program that counts the number of unique words in a large text file (say, "Alice in Wonderland"). The program should:
- Store the words in a collection and report the number of unique words in the text file.
- Allow the user to search it to see whether various words appear in the text file.

What collection is appropriate for this problem?
We could use an ArrayList... We’d really like a data structure that takes care of duplicates for us.
### Set Reference

#### Constructors

- `new HashSet<E>()`: Creates a new `HashSet` of type `E` that initially has no elements.
- `new HashSet<E>(collection)`: Creates a new `HashSet` of type `E` that initially has all the elements in `collection`.
- `new TreeSet<E>()`: Creates a new `TreeSet` of type `E` that initially has no elements.
- `new TreeSet<E>(collection)`: Creates a new `TreeSet` of type `E` that initially has all the elements in `collection`.

#### Methods

- `add(val)`: Adds `val` to the set.
- `contains(val)`: Returns true if `val` is a member of the set.
- `remove(val)`: Removes `val` from the set.
- `clear()`: Removes all elements from the set.
- `size()`: Returns the number of elements in the set.
- `isEmpty()`: Returns true whenever the set contains no elements.
- `toString()`: Returns a string representation of the set such as `[3, 42, -7, 15]`.

### Looping Through Sets

**How can we list all the elements of a set?**

We can’t do a normal `for` loop, because there are no indexes.

**We also don’t know what is actually in the set...**

**Solution**

The solution is a new type of loop called the `foreach` loop.

```java
Set<Integer> set = new HashSet<Integer>();
set.add(5);
set.add(5);
set.add(5);
set.add(10);
set.add(12);
for (int i : set) {
    System.out.println(i);
}
```

**OUTPUT**

```
>> 10
>> 5
>> 12
```

### foreach Loops

In general, `foreach` loops look like the following:

```java
for (type var : collection) {
    // do something with var
}
```

You can use them for many other collections like Lists. You are not allowed to use them for Stacks or Queues.

**Another Example of foreach Loops**

```java
List<String> list = new ArrayList<String>();
list.add("a");
list.add("a");
list.add("b");
String everything = "";
for (String s : list) {
    everything += s;
}
System.out.println(everything);
```

**OUTPUT**

```
>> aabd
```

### Data Structure Performance

The following is the performance of various data structures at removing duplicates from a large dictionary of words.

- HashSet
- TreeSet
- ArrayList
- LinkedList

### Data Structure Performance, Part 2

Note that despite it looking like `HashSet` and `TreeSet` have the same runtime on the previous slide, they do not.

### Alice in Wonderland, Take 2

**Count the Number of Occurrences of Each Word in a Text**

Write a program that counts the number of unique words in a large text file (say, "Alice in Wonderland"). The program should:

- Allow the user to type a word and report how many times that word appeared in the book.
- Report all words that appeared in the book at least 500 times, in alphabetical order.

**What collection is appropriate for this problem?**

We could use something sort of like `LetterInventory`, but we don’t know what the words are in advance... We’d really like a data structure that relates tallies with words.
Map is an interface in java.util; implementations of that interface are:

**TreeMap**
- Really fast for all operations.
- Does maintain the keys in sorted order

**HashMap**
- REALLY REALLY fast for all operations.
- Does not maintain a useful ordering of anything

Each map can **answer one type of question**. For example:

If the keys are phone numbers and the values are people
Then, the map can answer questions of the form:

"Who does this phone number belong to?"

```
Map<String,String> phoneNumbers = new HashMap<String, String>();
phoneNumbers.put("Adam's Office", "(206) 616-0034");
phoneNumbers.put("Adam's Office", "(206) 616-0034"); // Returns "Adam's Office"
```

The people map can **only go in one direction**. If we want the other direction, we need a different map:

If the keys are people and the values are phone numbers
Then, the map can answer questions of the form:

"What is this person's phone number?"

```
Map<String,Integer> numChars = new HashMap<String,Integer>();
numChars.put("goodbye", 7);
numChars.put("such strings", 12);
numChars.put("much wow", 8);
```

Earlier, we had an example where
- keys were "phrases"
- values were "# of chars in the key"

That map can answer the question:

"How many characters is this string?"
There is no good way to go from a value to its key using a map. But we can go from each key to the values:

```java
Map<String, Double> ages = new TreeMap<String, Double>();
// These are all according to the internet... a very reliable source!
ages.put("Bigfoot", 100);
ages.put("Loch Ness Monster", 3.50);
ages.put("Chupacabra", 20); // ages.keySet() returns Set<String>
ages.put("Yeti", 40000);
for (String cryptid : ages.keySet()) {
    double age = ages.get(cryptid);
    System.out.println(cryptid + " -> " + age);
}
```

```
OUTPUT
>> Chupacabra -> 20
>> Loch Ness Monster -> 1500
>> Bigfoot -> 100
>> Yeti -> 40000
```

You can get a collection of all the values:

```java
Map<String, Double> ages = new TreeMap<String, Double>();
// These are all according to the internet... a very reliable source!
ages.put("Bigfoot", 100);
ages.put("Loch Ness Monster", 3.50);
ages.put("Chupacabra", 20); // ages.keySet() returns Set<String>
ages.put("Yeti", 40000);
for (int age : ages.values()) {
    System.out.println("One of the cryptids is aged " + age);
}
```

```
OUTPUT
>> One of the cryptids is aged 1500
>> One of the cryptids is aged 40000
>> One of the cryptids is aged 20
>> One of the cryptids is aged 100
```

Some Set/Map Tips!

- Sets and Maps are two more collections each with their own places
- Sets are for storing data **uniquely**
- Maps are for storing **relationships** between data; they only work in **one direction**
- **foreach** loops are a great tool for looping through collections
- You should know the syntax for **foreach** loops and that Hash and Tree are types of sets and maps