Grammars, Sets, and Maps

Languages and Grammars

**Definition (Formal Language)**

A **Formal Language** is a set of words or symbols.

For example:

\{1, 2, 3, 4, 5\} is a language, and \{hello, goodbye\} is a language.

**Definition (Grammar)**

A **Grammar** is a set of rules that generates a particular language.

Grammars are used to:

- **generate** strings, and to
- **check** if strings are in the language

**An Example Grammar**

BNF is a syntax for describing language grammars in terms of transformation rules, of the form:

\[(symbol) ::= \langle expression \rangle \mid \langle expression \rangle \mid ... \mid \langle expression \rangle\]

BNF is made up of two types of symbols:

- **Terminals**: Literals (symbols that are interpreted literally)
- **Non-terminals**: A symbol describing how to generate other symbols based on the rules of the grammar

To generate \langle object\rangle\rangles from this grammar, we do the following steps:

1. Start at \langle object\rangle and look at what to transform to:
   \langle article\rangle \langle thing\rangle
2. For each non-terminal, look at its rule and choose an option.

Some \langle object\rangle\rangles in this grammar:

- The ball
- That index card
- The balloon
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"
"very hello"
"goodbye"
"such strings"
```

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:

- HashSet
  - \(O(1)\) for all operations.
  - Does not maintain a useful ordering

- TreeSet
  - \(O\log(n)\) for all operations
  - Does maintain the elements in sorted order

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(12);
for (int i : set) {
    System.out.println(i);
} // The set remains unchanged.
```

Output:
```
>> 5
>> 12
```

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.

foreach Loops

In general, foreach loops look like the following:
```
1 for (type var : collection) {
2    // do something with var
3 }
```

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**
```
List<String> list = new ArrayList<String>();
list.add("a");
list.add("b");
list.add("c");
list.add("d");
String everything = "";
for (String s : list) {
    everything += s;
}
System.out.println(everything); // output: "abcd"
```
**Data Structure Performance**

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

![Graph showing performance comparison between LinkedList, ArrayList, TreeSet, and HashSet.](image)

**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.

**What is a Map?**

**Definition (Map)**

A map is a data structure that relates keys and values. You can do the following with a map:

- Ask what value a particular key maps to.
- Change what value a particular key maps to.
- Remove whatever the relation is for a given key.

**How To Think About Maps**

Maps are a lot like functions you’ve seen in math: \( f(x) = x^2 \) maps 0 to 0, 2 to 4, ... Your keys are identifiers for values. Ex: social security numbers (maps SSN \( \rightarrow \) person).

Safe-deposit boxes are another useful analogy. You get a literal key to access your belongings. If you know what the key is, you can always get whatever you’re keeping safe.

**Example Map**

<table>
<thead>
<tr>
<th>Keys</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;much wow&quot;</td>
<td>8</td>
</tr>
<tr>
<td>&quot;goodbye&quot;</td>
<td>7</td>
</tr>
<tr>
<td>&quot;such strings&quot;</td>
<td>12</td>
</tr>
</tbody>
</table>

How many characters is "much wow"? 8
What does "goodbye" map to? 7
What is the value for "such strings"? 12

**Map Implementations**

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

- **HashMap**
  - \( O(1) \) for all operations.
  - **Does not** maintain a useful ordering of anything

- **TreeMap**
  - \( O(\log(n)) \) for all operations
  - **Does maintain** the keys in **sorted order**

**Map Constructors & Type Parameters**

Creating A Map

To create a map, you must specify two types:

- What type are the keys?
- What type are the values?

They can be the same, but they aren’t always.

**Constructors**

- `new HashMap<K, V>()` creates a new HashMap with keys of type `K` and values of type `V` that initially has no elements.
- `new TreeMap<K, V>()` creates a new TreeMap with keys of type `K` and values of type `V` that initially has no elements.
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?”

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:

```
1 Map<String, Integer> numChars = new HashMap<String, Integer>();
2 numChars.put("goodbye", 7);
3 numChars.put("such strings", 12);
4 numChars.put("much wow", 8);
5 numChars.get("much wow"); // Returns 8
```

Using A Map

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 are in this string?”

```
1 Map<String, Integer> numChars = new HashMap<String, Integer>();
2 numChars.put("very hello", 10);
3 numChars.put("such strings", 12);
4 numChars.put("much wow", 8);
5 numChars.get("much wow"); // Returns 8
```

Some Grammar/Set/Map Tips!

- BNF is another recursive structure!
- 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