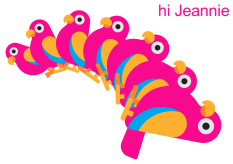
✨ No Laptop Monday! ✨

hi Jeannie

Week07

- map (dictionary, table) interface



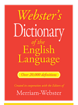
WARMUP

long, long, ago, many people owned a large, physical book called a ✨dictionary✨
- what was it used for?
- did you know there were different kinds of dictionaries?

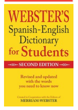
[record lecture]

real-world dictionaries

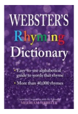
note: these are NOT hash maps
(hash maps are NOT ordered)



```
Map<String, String> dictionary;
// {Dog=A cute, four-legged mammal, ...}
```



```
Map<String, String> spanishToEnglishDictionary;
// {Gato=Cat, Perro=Dog, ...}
```



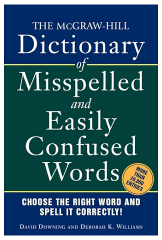
```
Map<String, List<String>> rhymingDictionary;
// {Dog=[Bog, Cog, Frog], ...}
```

common occurrence in ye olden times

- **child:** "how do you spell trebuchet?"
- **parent:** "look it up in the dictionary."

note: this advice made no sense

unless...


THE McGRAW-HILL
Dictionary of Misspelled and Easily Confused Words
CHOOSE THE RIGHT WORD AND SPELL IT CORRECTLY!
David Downing and Deborah K. Williams

modern alternative

Google

maps

map interface

map

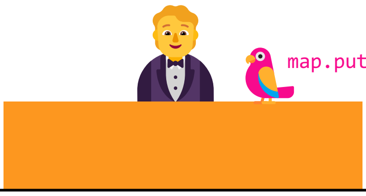
- a **pair** is two things
- a **map** (**dictionary**, **table**) stores **key-value pairs**
  - in a map, you can **lookup** a key's value
  - a map in Java's standard library is `HashMap<KeyType, ValueType>`
    - `HashMap<String, ArrayList<String>> rhymingDictionary;`

analogy

let us imagine...

you have a three week long road trip planned to Bennington, but your prized parrot Hans is deathly afraid of driving

it's time to **put** Hans in the bird kennel



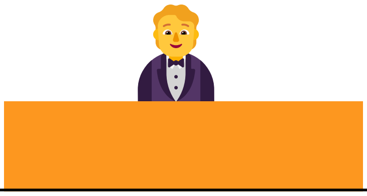
```
map.put("Hans", 🦜 )
```

you return invigorated from a lovely trek to Bennington

it's time to **get** Hans from the bird kennel

```
map.get("Hans")
```

```
HashMap<String, Bird>
```



map interface

```
– // Put (add, insert) a new key-value pair into the map.
  // NOTE: Can also be used to update a key's value.
  // NOTE: All keys are unique.
  void put(KeyType key, ValueType value);

– // Get (lookup) a key's value in the map.
  ValueType get(KeyType key);

– // Get the (unordered) set of all keys.
  // NOTE: Use a for-each loop to iterate through this.
  //       for (KeyType key : map.keySet()) { ... }
  Set<KeyType> keySet();
```

simple example

```
HashMap<Character, Integer> map = new HashMap<>();
PRINT(map); // {}

map.put('a', 1);
PRINT(map); // {a=1}

map.put('b', 1);
PRINT(map); // {a=1, b=1}

map.put('b', 2);
PRINT(map); // {a=1, b=2}
```

motivating example

Pokémon

- there are 151 **Pokémon**
- each has its own **number**
  1. Bulbasaur
  2. Ivysaur
  3. Venusaur
  4. Charmander
  5. ...

number → name

```
1 → "Bulbasaur"
2 → "Ivysaur"
3 → "Venusaur"
4 → "Charmander"
...
```

what data structure should we use to implement this?

an array

the answer is always array

even during the map lecture

number → name (Option A)

```
String[] names = { "Bulbasaur", "Ivysaur", ... };
```

```
String name = names[number - 1]; // NOTE: Bulbasaur is #1
```

number → name (Option B)

```
String[] _names = { "Bulbasaur", "Ivysaur", ... };
String getName(int number) {
    return _names[number - 1]; // NOTE: Bulbasaur is #1
}
```

```
String name = getName(number);
```

number → name (Option C)

```
// NOTE: Bulbasaur is #1
String[] names = { "", "Bulbasaur", "Ivysaur", ... };
```

```
String name = names[number];
```

lesson: if you're mapping from A → B, and A is something like 0, 1, 2, 3, ...

...then you can just use an array 🙄 👍
(or an array list if you don't know the length ahead of time)

buuut, if you're mapping from A → B, and A is *nothing* like 0, 1, 2, 3, ...

...then you're going to want a **map** 🗺️

name → number

1. what **data structure** will you need? (**hint:** `HashMap<???, ???>`)
2. how will you **set it up** (how will you populate it)?
3. how will you **use it**?

```
"Bulbasaur"   → 1
"Ivysaur"     → 2
"Venusaur"    → 3
"Charmander"  → 4
...
```

```
void put(KeyType key, ValueType value);
ValueType get(KeyType key);
```

name → number

```
// data structures
HashMap<String, Integer> numberFromName;
```

```
// setup
String[] names = { "Bulbasaur", "Ivysaur", ... };
numberFromName = new HashMap<>();
for (int i = 0; i < names.length; ++i) {
    // Bulbasaur = 1
    numberFromName.put(names[i], i + 1);
}
```

```
// usage
String name = ...;
int number = numberFromName.get(name);
```

[let's implement `numberFromName`]

TODO (Jim): use `keySet()` method in test code

PokemonExample

```
import java.util.*;
class Main extends Cow {
    public static void main(String[] args) {
        // setup
        HashMap<String, Integer> numberFromName = new HashMap<>();
        String[] names = { "Bulbasaur", "Ivysaur", "Venusaur", ... };
        for (int number = 0; number < names.length; ++number) {
            numberFromName.put(names[number], number + 1);
        }


        // usage
        for (String name : numberFromName.keySet()) {
            PRINT(name + " -> " + numberFromName.get(name));
        }
    }
}
```

```
Mewtwo -> 150
Raichu -> 26
Arbok -> 24
Kabuto -> 140
Charmander -> 4
Charmeleon -> 5
Shellder -> 90
Fearow -> 22
Kangaskhan -> 115
Mankey -> 56
Dodrio -> 85
Vaporeon -> 134
Golem -> 76
Marowak -> 105
Pikachu -> 25
Raticate -> 20
Kadabra -> 64
Primeape -> 57
Venomoth -> 49
Magikarp -> 129
Pidgeotto -> 17
Slowbro -> 80
Gastly -> 92
Magneton -> 82
```

reminder: hash maps are NOT ordered
(even though we put Pokemon into the hash map in order, the hash map did NOT store them in order)

maps in other languages

## Python has a built-in map (dictionary)

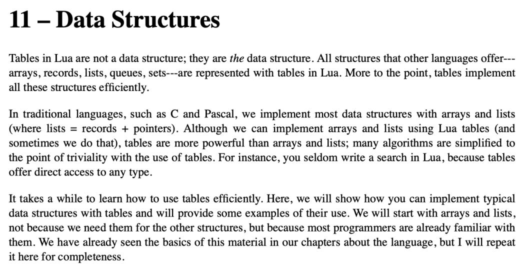
```python
numbers = {}
numbers["Bulbasaur"] = 1
numbers["Ivysaur"] = 2
print(numbers["Bulbasaur"]) # 1
print(numbers) # {"Bulbasaur": 1, "Ivysaur": 2}
```

```python
# hmm...
map = {}
map[True] = False
map['Jim'] = 3
map[-1] = map
print(map) # {True: False, 'Jim': 3, -1: {...}}
```

Lua's *only* data structure is a map (table)

- why?
- how?
  - what about like...arrays?

```lua
names = {}
names[1] = "Bulbasaur" -- okay fine, i guess Lua also has strings
names[2] = "Ivysaur"
print(names[1]) -- Bulbasaur
```



11 – Data Structures

Tables in Lua are not a data structure; they are *the* data structure. All structures that other languages offer---arrays, records, lists, queues, sets---are represented with tables in Lua. More to the point, tables implement all these structures efficiently.

In traditional languages, such as C and Pascal, we implement most data structures with arrays and lists (where lists = records + pointers). Although we can implement arrays and lists using Lua tables (and sometimes we do that), tables are more powerful than arrays and lists; many algorithms are simplified to the point of triviality with the use of tables. For instance, you seldom write a search in Lua, because tables offer direct access to any type.

It takes a while to learn how to use tables efficiently. Here, we will show how you can implement typical data structures with tables and will provide some examples of their use. We will start with arrays and lists, not because we need them for the other structures, but because most programmers are already familiar with them. We have already seen the basics of this material in our chapters about the language, but I will repeat it here for completeness.