Lecture 3: Working with Data Structures

Data Structures and Algorithms

Warm Up

From last lecture:

- What is an example of "constant time" complexity
- What is an example of "linear time" complexity
- What is the complexity class of the binary search algorithm?

From CSE 143:

- What is a "map" and what are some situations you would use it in?
- What are the main methods you use when traversing data with a Scanner?

Socrative:

www.socrative.com

Room Name: CSE373

Please enter your name as: Last, First

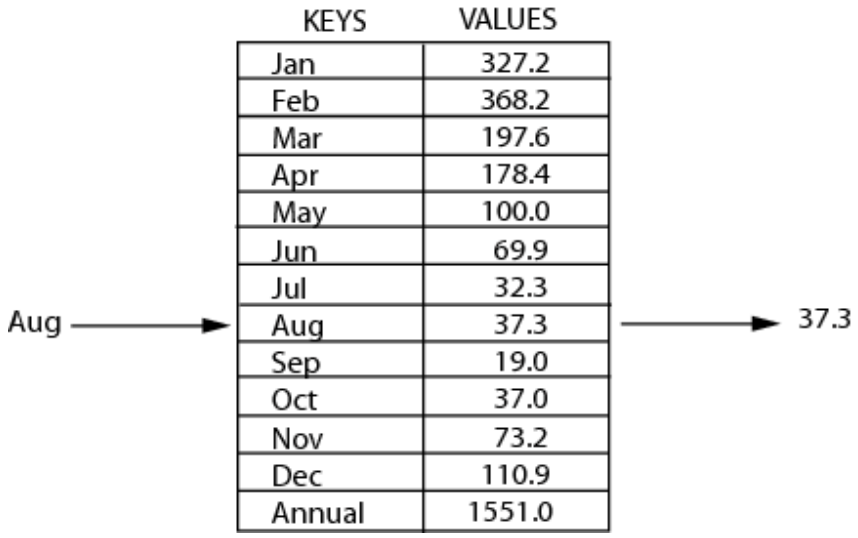
Review: Maps

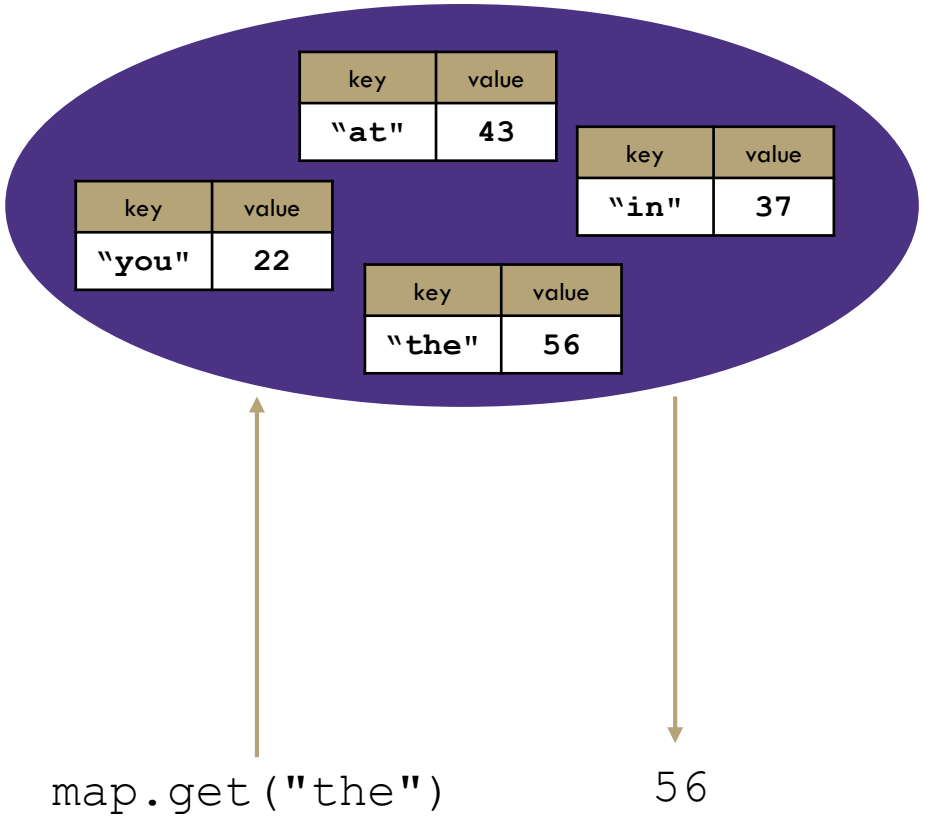
map: Holds a set of unique *keys* and a collection of *values*, where each key is associated with one value.

- a.k.a. "dictionary", "associative array", "hash"

operations:

- **put**(*key*, *value*): Adds a mapping from a key to a value.
- **get**(*key*): Retrieves the value mapped to the key.
- **remove**(*key*): Removes the given key and its mapped value.





Implement a Map

Traversing Data

Array

```
for (int i = 0; i < arr.length; i++) {
    System.out.println(arr[i]);
}
```

List

```
for (int i = 0; i < myList.size(); i++) {
    System.out.println(myList.get(i));
}
```

```
for (T item : list) {
    System.out.println(item);
}
```

Iterator!

Iterators

iterator: a Java interface that dictates how a collection of data should be traversed.

Behaviors:

hasNext() – returns true if the iteration has more elements

next() – returns the next element in the iteration

```
while (iterator.hasNext()) {
    T item = iterator.next();
}
```

Implementing an Iterator

Testing

Computers don't make mistakes- people do!

"I'm almost done, I just need to make sure it works"
– Naive 14Xers

Software Test: a separate piece of code that exercises the code you are assessing by providing input to your code and finishes with an assertion of what the result should be.

1. Isolate

break your code into small modules

2. Build in increments

Make a plan from simplest to most complex cases

3. Test as you go

As your code grows, so should your tests

Types of Tests

Black Box

- Behavior only – ADT requirements
- From an outside point of view
- Does your code uphold its contracts with its users?
- Performance/efficiency

White Box

- Includes an understanding of the implementation
- Written by the author as they develop their code
- Break apart requirements into smaller steps
- "unit tests" break implementation into single assertions

What to test?

Expected behavior

- The main use case scenario
- Does your code do what it should given friendly conditions?

Forbidden Input

- What are all the ways the user can mess up?

Empty/Null

- Protect yourself!
- How do things get started?

Boundary/Edge Cases

- First
- last

Scale

- Is there a difference between 10, 100, 1000, 10000 items?

Thought Experiment

Discuss with your neighbors: Imagine you are writing an implementation of the List interface that stores integers in an Array. What are some ways you can assess your program's correctness in the following cases:

Expected Behavior

- Create a new list
- Add some amount of items to it
- Remove a couple of them

Forbidden Input

- Add a negative number
- Add duplicates
- Add extra large numbers

Empty/Null

- Call remove on an empty list
- Add to a null list
- Call size on an null list

Boundary/Edge Cases

- Add 1 item to an empty list
- Set an item at the front of the list
- Set an item at the back of the list

Scale

- Add 1000 items to the list
- Remove 100 items in a row
- Set the value of the same item 50 times

JUnit

JUnit: a testing framework that works with IDEs to give you a special GUI experience when testing your code

```
@Test
public void myTest() {
    Map<String, Integer> basicMap = new LinkedListDict<String, Integer>();
    basicMap.put("Kasey", 42);
    assertEquals(42, basicMap.get("Kasey"));
}
```

Assertions:

- assertEquals(item1, item2)
- assertTrue(Boolean expression)
- assertFalse(bollean expression)
- assertNotNull(item)

More: https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html

Write Tests for our Dictionary

Debugger

TODO list

Homework 1 is live!

Individual assignment

Due 4/6 at 11:59pm