CompSci 201, L6: Hashing, HashMap, HashSet

Logistics, Coming Up

- Today, Wednesday 9/14
  - APT 2 due
- Friday 9/16
  - Discussion 3
- Monday 9/19
  - Project 1: Nbody due

Efficiency claims about HashSet/HashMap

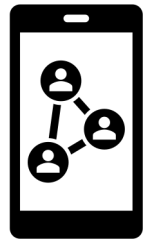
- HashSet: `.add()`, `.contains()`, both *constant time complexity*.
- HashMap: `.put()`, `.get()`, `.containsKey()`, all constant time complexity.

- Constant time? Methods take about the same time on a Set/Map with 1 hundred elements as on a Set/Map with 1 billion elements!

Seems like magic! How to search (`contains()`) without looping over everything?!?

Aside: Does constant time lookup (`contains()`, `get()`, etc.) matter?

- Social media: When you login, server needs to lookup to display the correct page for you.
  - Billions of accounts! Look it up in a List? NO! Constant time lookup with hashing.



- Routing/directions application: Need to lookup roads from a given intersection.
  - How many possible roads? Search through a list? NO! Constant time lookup with hashing.

- Could go on!

Big questions about hashing

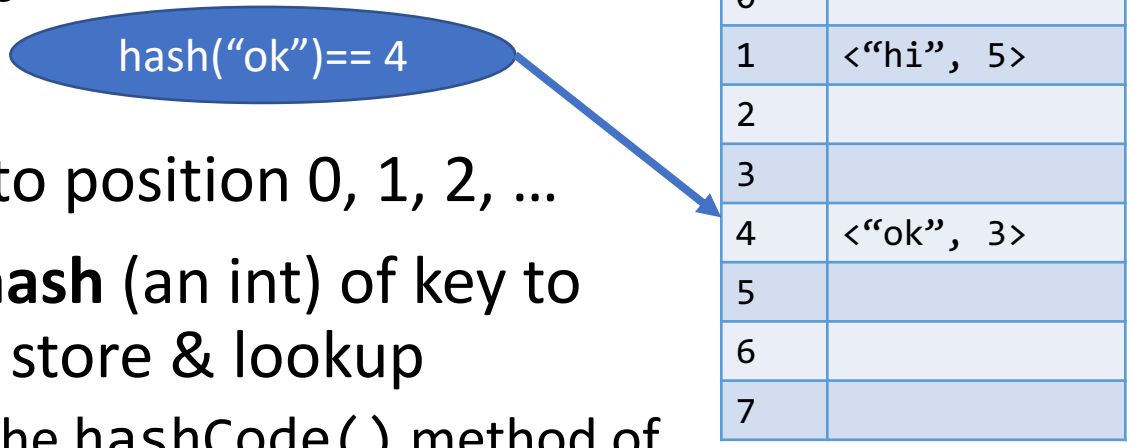
Last class: Usage of API HashSet/HashMap.

Today:

1. How does a hash table work to implement HashMap/HashSet?
2. Why do `.equals()` and `.hashCode()` matter?
3. Why are the `add()`, `contains()`, `put()`, `get()`, and `containsKey()`, etc., all constant time?

Hash Table Concept

- Implemented an ArrayList using an Array
- Implement HashMap with an ArrayList
  - Of <key, value> pairs



- Rather than adding to position 0, 1, 2, …
- **Big idea:** Calculate **hash** (an int) of key to determine where to store & lookup
  - Java OOP: Will use the `hashCode()` method of the key to get the hash
- Same hash to put and get, no looping over list

HashMap methods at a high level

Always start by getting the **hash** =
`Math.abs(key.hashCode()) % list.size()`

Absolute value and % (remainder when dividing by) list size ensures valid index

- put(key, value)
  - Add (<key, value>) to list at index hash
  - If key already there, update value
- get(key)
  - Return value paired with key at index hash position of list
- containsKey(key)
  - Check if key exists at index hash position of list

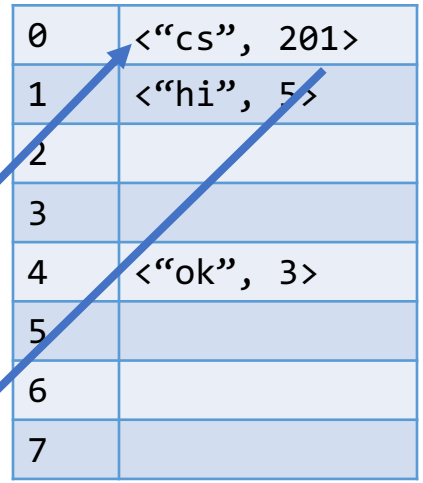
0	
1	<“hi”, 5>
2	
3	
4	<“ok”, 3>
5	
6	
7	

HashMap put/get example

- Suppose we have the <key, value> pair <“cs”, 201>.
- hash is `Math.abs(“cs”.hashCode()) % 8` which is `0`.

```
jshell> Math.abs("cs".hashCode()) % 8
$7 ==> 0
```

- put(“cs”, 201) in position 0
- get(“cs”) by looking up position 0, returning the value



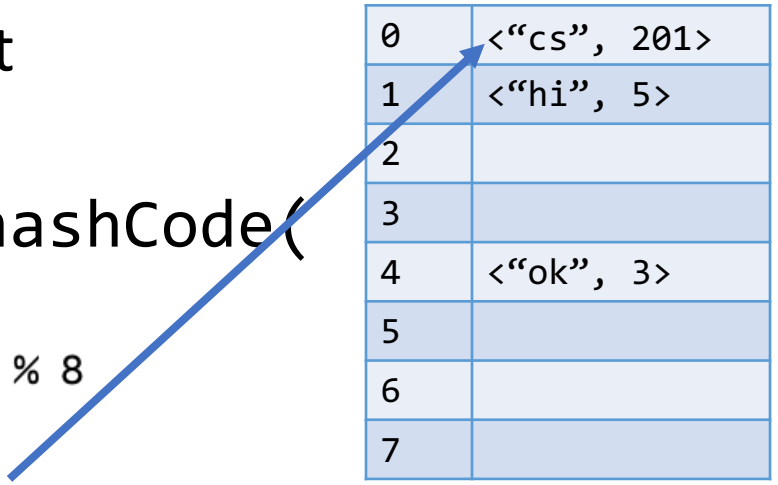
return 201

Collisions

- Suppose now we want to put <“fain”, 104>.
- hash=Math.abs(“fain”.hashCode()) % 8 which is 0.

```
jshell> Math.abs("fain".hashCode()) % 8
$11 ==> 0
```

- put(“fain”, 104) in position 0
- But <“cs”, 201> is already stored at position 0! Call this a **collision**.



Dealing with collisions: concepts

- Think of the hash table as an ArrayList of “buckets”.
- Each bucket can store multiple <key, value> pairs.
- `put(key, value)`
  - Add to hash index bucket
  - Update value if key already in bucket
- `get(key)`
  - Loop over keys in hash index bucket
  - Return value of one that equals() key

0	<“cs”, 201> <“fain”, 104>
1	<“hi”, 5>
2	
3	
4	<“ok”, 3>
5	
6	
7	

Dealing with collisions: details

- Bucket is really another list.
- Hash table is really a list of lists of <key, value> pairs.
- We call this technique for dealing with collisions **chaining**.

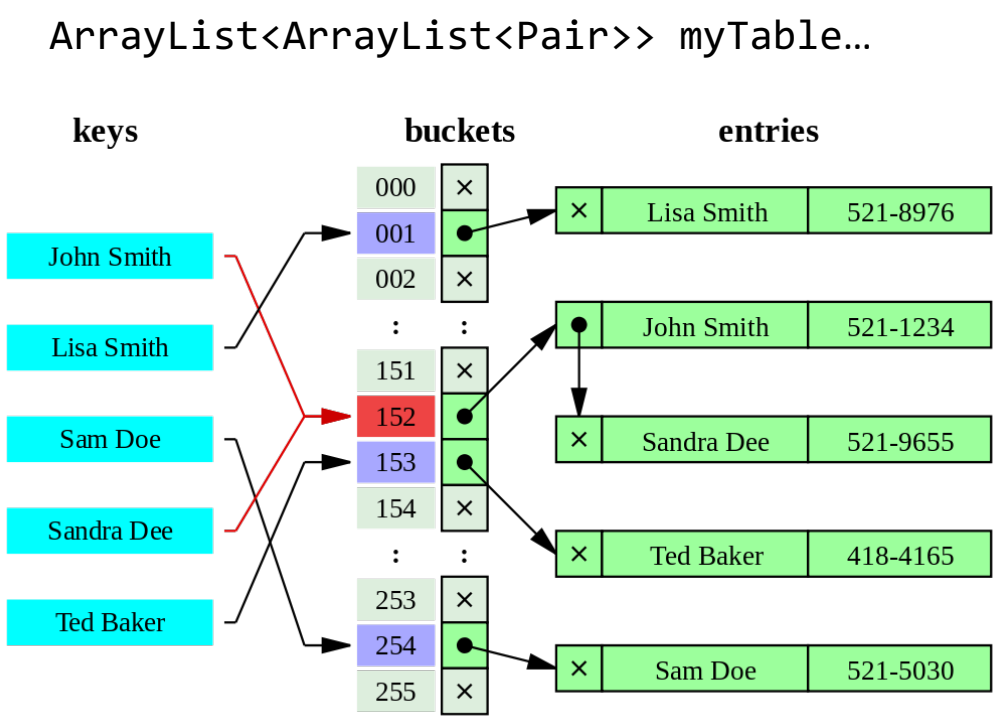


Illustration credit: By Jorge Stolfi - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=6471915

WOTO

Go to duke.is/mt2ms

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

Where does `equals()` come in?

- If multiple <key, value> pairs in same bucket, need to know which to get() or update on a put() call.
- Always the pair where the key in the bucket `equals()` the key we `put()` or `get()`.
- Need `equals()` to work correctly for the key type
  - String keys? Integer? Already implemented for you.
  - Storing objects of a class *you write*? Need to override and implement `equals()`.

What happens without equals()? Hashing cats

```
4   public class Cat {
5       String name;
6       int age;
7
8       @Override
9       public int hashCode() {
10          return 0;
11      }
12
        Run | Debug
13      public static void main(String[] args) {
14          Set<Cat> myCats = new HashSet<>();
15          myCats.add(new Cat("kirk", 2));
16          myCats.add(new Cat("kirk", 2));
17          System.out.println(myCats.size());
18      }
```

Even though all cat objects have the same `hashCode()` of 0 and so go to the same bucket…

And these 2 Cat objects have the same values

Prints 2, cannot detect duplicates without `equals()`

hashCode Correctness

- Need `hashCode()` to work correctly for the key type.
  - String keys? Already implemented for you.
  - Storing objects of classes *you write*? Need to override and implement `hashCode()`.

- What makes a `hashCode()` "correct" (not necessarily efficient)?
  - **Any two objects that are `equals()` should have the same `hashCode()`.**

What happens without hashCode()? Hashing more cats

```
4   public class Cat {
5       String name;
6       int age;
7
8       @Override
9       public boolean equals(Object o) {
10          Cat other = (Cat) o;
11          if ((other.name.equals(this.name)) && (other.age == this.age)) {
12              return true;
13          }
14          return false;
15      }
16
        Run | Debug
17      public static void main(String[] args) {
18          Set<Cat> myCats = new HashSet<>();
19          myCats.add(new Cat("kirk", 2));
20          myCats.add(new Cat("kirk", 2));
21          System.out.println(myCats.size());
22      }
```

Fixed `equals()` but removed `hashCode()`, using default

Still prints 2! `equals()` works, but they get hashed to different buckets.

Cat with `equals()` and `hashCode()`

```
4   public class Cat {
5       String name;
6       int age;
7
8       @Override
9       public boolean equals(Object o) {
10          Cat other = (Cat) o;
11          if ((other.name.equals(this.name)) && (other.age == this.age)) {
12              return true;
13          }
14          return false;
15      }
16
17      @Override
18      public int hashCode() {
19          return (name + Integer.toString(age)).hashCode();
20      }
```

`equals()` if have same name and age

Uses String `hashCode()` of name `concat` with age, if `equals()` will have same `hashCode()`

Aside: `toString()`

Don't need for hashing, but `toString()` method allows "nice" printing.

```
4   public class Cat {
5       String name;
6       int age;
7
8       @Override
9       public String toString() {
10          return name;
11      }
12
        Run | Debug
13      public static void main(String[] args) {
14          Set<Cat> myCats = new HashSet<>();
15          myCats.add(new Cat("kirk", 2));
16          System.out.println(myCats);
17      }
```

`toString()` method used for printing, including inside a Collection

Prints `[kirk]` instead of `[Cat@...]`

What is the String hashCode()?

Remember how hashCode() is used to get the bucket index.

```
42      private int getBucket(String s) {
43          int val = Math.abs(s.hashCode()) % myTable.size();
44          return val;
45      }
```

hashCode

```
public int hashCode()
```

Returns a hash code for this string. The hash code for a String object is c

```
s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]
```

using int arithmetic, where s[i] is the *i*th character of the string, n is th the string, and ^ indicates exponentiation. (The hash value o empty s

Overrides:

hashCode in class Object

Returns:

a hash code value for this object.

```
jshell> "hello".hashCode();
$4 ==> 99162322

jshell> "hellp".hashCode();
$5 ==> 99162323

jshell> "what".hashCode();
$6 ==> 3648196
```

Java API String documentation

Interprets each character as an int, does arithmetic.

Revisiting Hashing Efficiency

- Real runtime of `get()`, `put()`, and `containsKey()` =
  - Time to get the hash
  - + Time to search over the hash index "bucket", calling .equals() on everything in the bucket

Constant, does not depend on number of pairs in Map

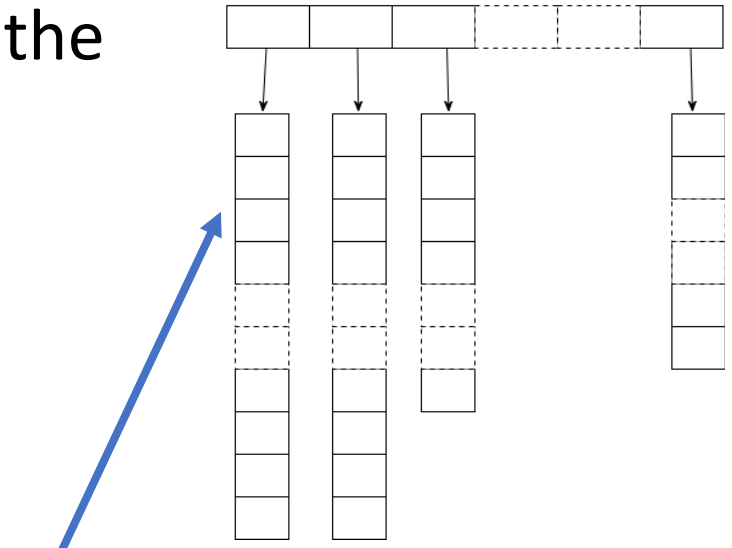
Depends on number of pairs per bucket

→HashMaps faster with more buckets

"correct" but inefficient `hashCode()`

Correctness requirement: Any `.equals()` keys should have the same `hashCode()`.

```
28      @Override
29      public int hashCode() {
30          return 0;
31      }
```



Still satisfies, but not good…

Stores everything in the first bucket!
No more efficient than ArrayList!

Correct and efficient `hashCode()`

From the Java 17 API documentation:

- Correctness: "If two objects are equal…hashCode…must produce the same integer result."
- Efficiency: "…producing distinct integer results for unequal objects may improve the performance of hash tables."
- String hashCode() satisfies both

```
jshell> "hello".hashCode();
$4 ==> 99162322

jshell> "hellp".hashCode();
$5 ==> 99162323

jshell> "what".hashCode();
$6 ==> 3648196
```

Cat `hashCode()` revisited

```
4   public class Cat {
5       String name;
6       int age;
7
8       @Override
9       public boolean equals(Object o) {
10          Cat other = (Cat) o;
11          if ((other.name == this.name) && (other.age == this.age)) {
12              return true;
13          }
14          return false;
15      }
16
17      @Override
18      public int hashCode() {
19          return (name + Integer.toString(age)).hashCode();
20      }
```

`equals()` if have same name and age

If `equals()` will have same `hashCode()`

If unequal? Unlikely (but possible!) to have the same `hashCode()`.

Simple uniform hashing assumption (SUHA)

- Suppose we hash N pairs to M buckets.

- **Simple uniform hashing assumption:** Probability two random (unequal) keys hash to same bucket is just 1/M.
  - Spread of pairs to buckets **looks random** (but is not).
  - Ways to design such hash functions, not today
  - We will make the assumption to analyze efficiency in theory, can verify runtime performance in practice

Implications of SUHA

- Expected number of pairs per bucket under SUHA? N/M [N pairs, M buckets].
- Stronger statements are true: Very high probability that a bucket has approximately N/M pairs.
- Runtime implication?
  - Time to get the hash
  - Time to search over the hash index "bucket"
    - Calling .equals() on everything in the bucket

Constant, does not depend on N or M.

Roughly N/M pairs to search

Memory/Runtime Tradeoff

- N pairs, M buckets, assuming SUHA / good hashCode()

- **Case 1: N >> M** – too many pairs in too few buckets
  - Overall runtime is ~N/M is NOT constant
- **Case 2: M >> N** – too many buckets, not many pairs
  - Overall runtime constant, NOT memory efficient
- **Case 3: M slightly larger than N** – sweet spot
  - Overall runtime constant, memory usage reasonable
  - Still uses more than a simple ArrayList – “No free lunch”

Load Factor and HashMap Growth

- N pairs, M buckets
- Load factor = maximum N/M ratio allowed
  - Java default is 0.75
- Whenever N/M exceeds the load factor?
  - Create a new larger table, rehash/copy everything
  - Double the size, geometric growth pattern for amortized efficiency just like ArrayList!
  - Called resizing

Hash table resizing

```
jshell> Math.abs("cs".hashCode()) % 4
$15 ==> 0
jshell> Math.abs("hi".hashCode()) % 4
$16 ==> 1
jshell> Math.abs("ok".hashCode()) % 4
$17 ==> 0
```

```
jshell> Math.abs("cs".hashCode()) % 8
$19 ==> 0
jshell> Math.abs("hi".hashCode()) % 8
$20 ==> 1
jshell> Math.abs("ok".hashCode()) % 8
$21 ==> 4
```

0	<"cs", 201> <"ok", 3>
1	<"hi", 5>
2	
3	

0	<"cs", 201>
1	<"hi", 5>
2	
3	
4	<"ok", 3>
5	
6	
7	

WOTO

Go to duke.is/2caye

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

Grace Hopper

- PhD in math from Yale in 1930s
- Joined Navy Reserve during WW2
- 1940s, began working on developing early computers:
  - Mark 1
  - UNIVAC 1
- 1950s, began work on the earliest "high level" programming languages
  - FLOW-MATIC
  - COBOL – Still in use!
- Annual Grace Hopper Celebration of Women in Computing, usually in the Fall. Consider attending!

USS Hopper