CS2040S
Data Structures and Algorithms

Hashing!
(Part 2)

Today: More Hashing!

- Java hashing
- Collision resolution: open addressing
- Table (re)sizing

Review: Symbol Table Abstract Data Type

Which of the following is *not* typically a symbol table operation?

1. insert(key, data)
2. delete(key)
3. successor(key)
4. search(key)
5. None of the above.

Review: Symbol Table Abstract Data Type

Which of the following is *not* typically a symbol table operation?

1. insert(key, data)
2. delete(key)
3. successor(key)
4. search(key)
5. None of the above.

Abstract Data Types

Symbol Table

public interface	**SymbolTable**	
`void`	`insert(Key k, Value v)`	*insert (k,v) into table*
`Value`	`search(Key k)`	*get value paired with k*
`void`	`delete(Key k)`	*remove key k (and value)*
`boolean`	`contains(Key k)`	*is there a value for k?*
`int`	`size()`	*number of (k,v) pairs*

Note: no successor / predecessor queries.

Direct Access Tables

Attempt #1: Use a table, indexed by keys.

0	null
1	null
2	item1
3	null
4	null
5	item3
6	null
7	null
8	item2
9	null

$\Big\}$ Universe U={0..9} of size $m = 10$.

<u>(key, value)</u>

(2, item1)
(8, item2)
(5, item3)

Assume keys are distinct.

Direct Access Tables

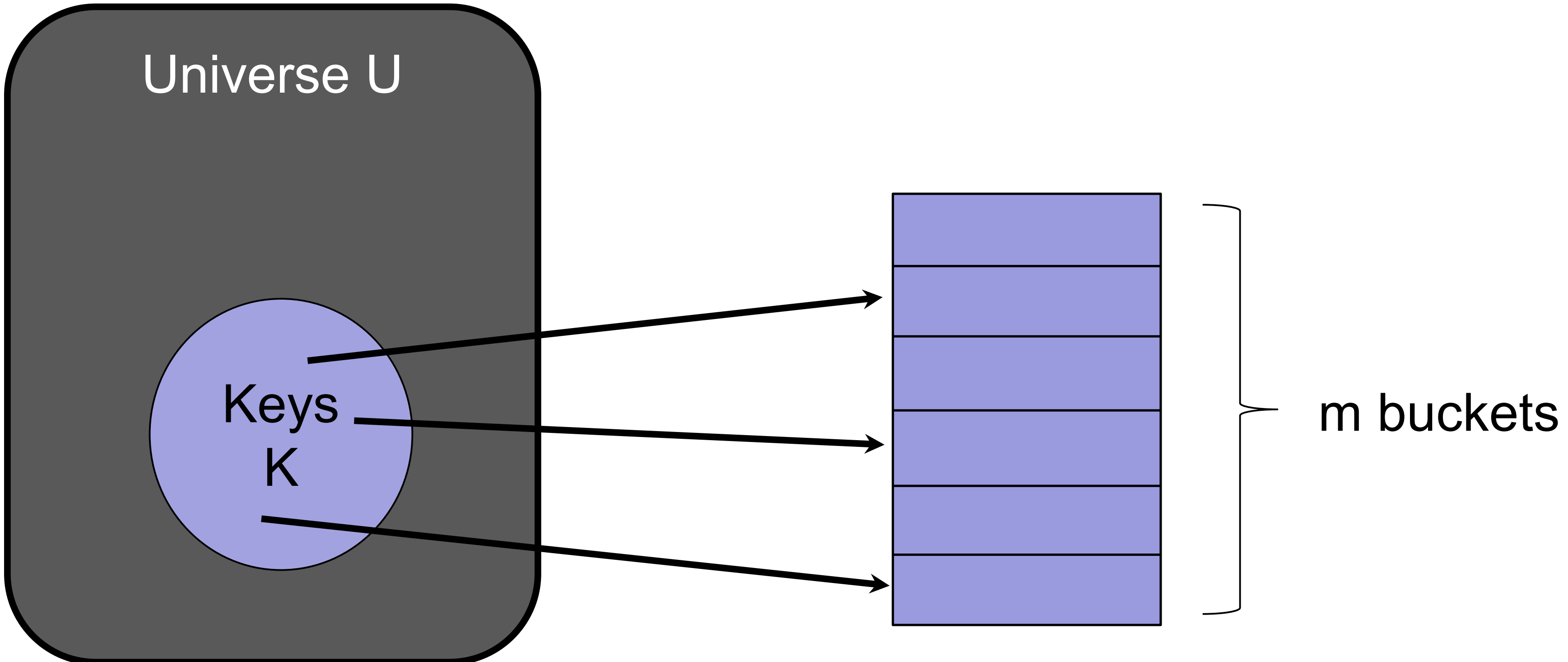
Problems:

- Too much space
  - If keys are integers, then table-size > 4 billion
- What if keys are not integers?
  - Where do you put the key/value "**(hippopotamus, bob)**"?
  - Where do you put 3.14159…?

Hash Functions

Problem:

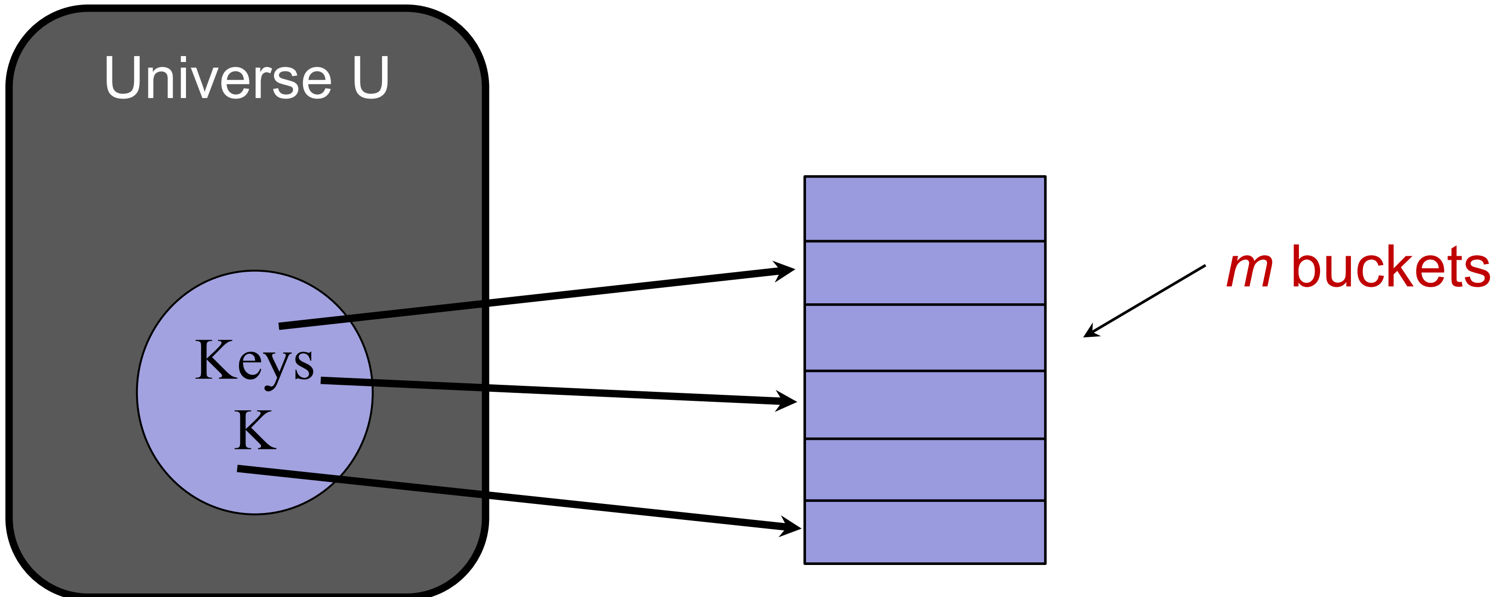
- Huge universe U of possible keys.
- Smaller number n of actual keys.
- How to map n keys to $m \approx n$ buckets?



Hash Functions

Define hash function h : U □ {1..*m*}

- Store key *k* in bucket h(*k*).



Hash Functions

Collisions:

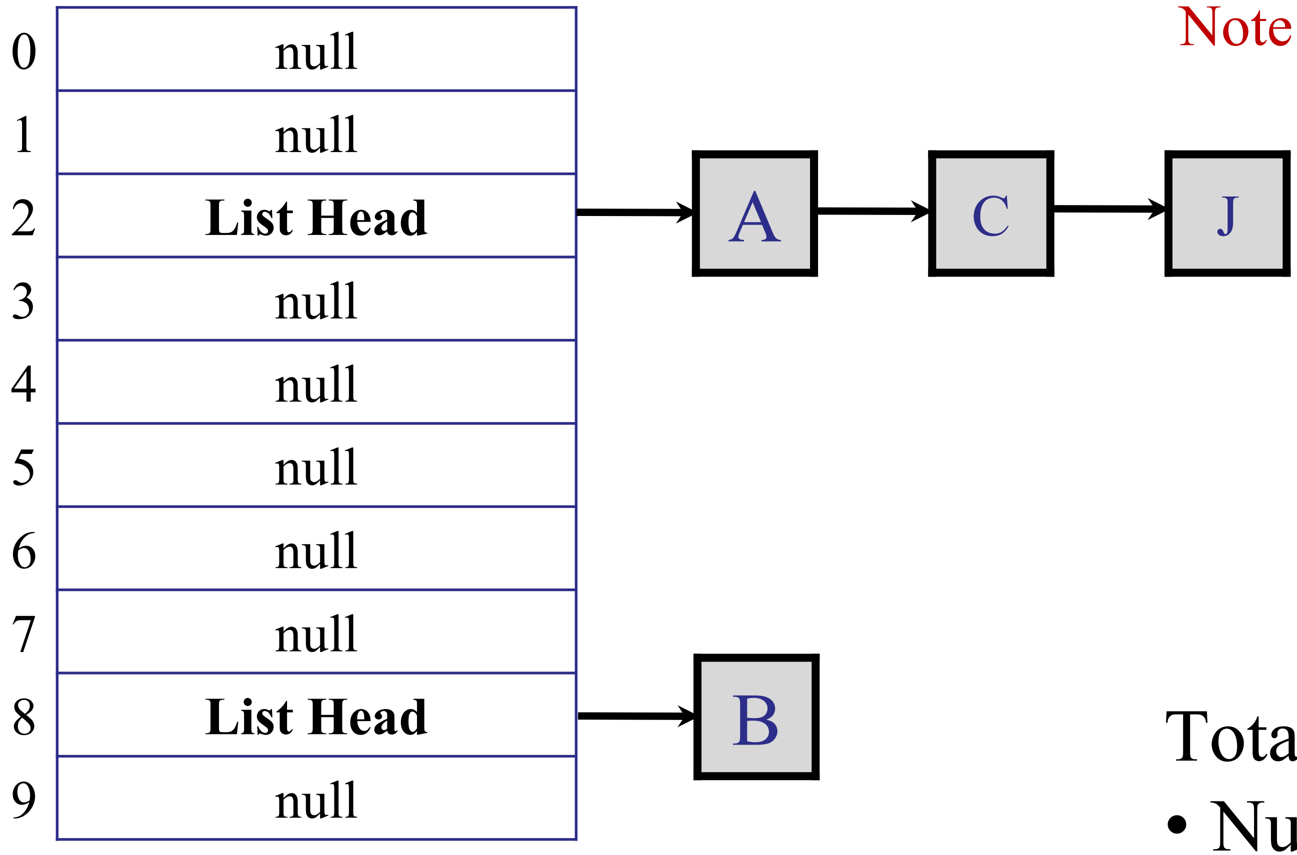
- We say that two distinct keys k_1 and k_2 collide if:

$$h(k_1) = h(k_2)$$

- Unavoidable!
  - The table size is smaller than the universe size.
  - The pigeonhole principle says:
    - There must exist two keys that map to the same bucket.
    - Some keys must collide!

Chaining

Each bucket contains a linked list of items.

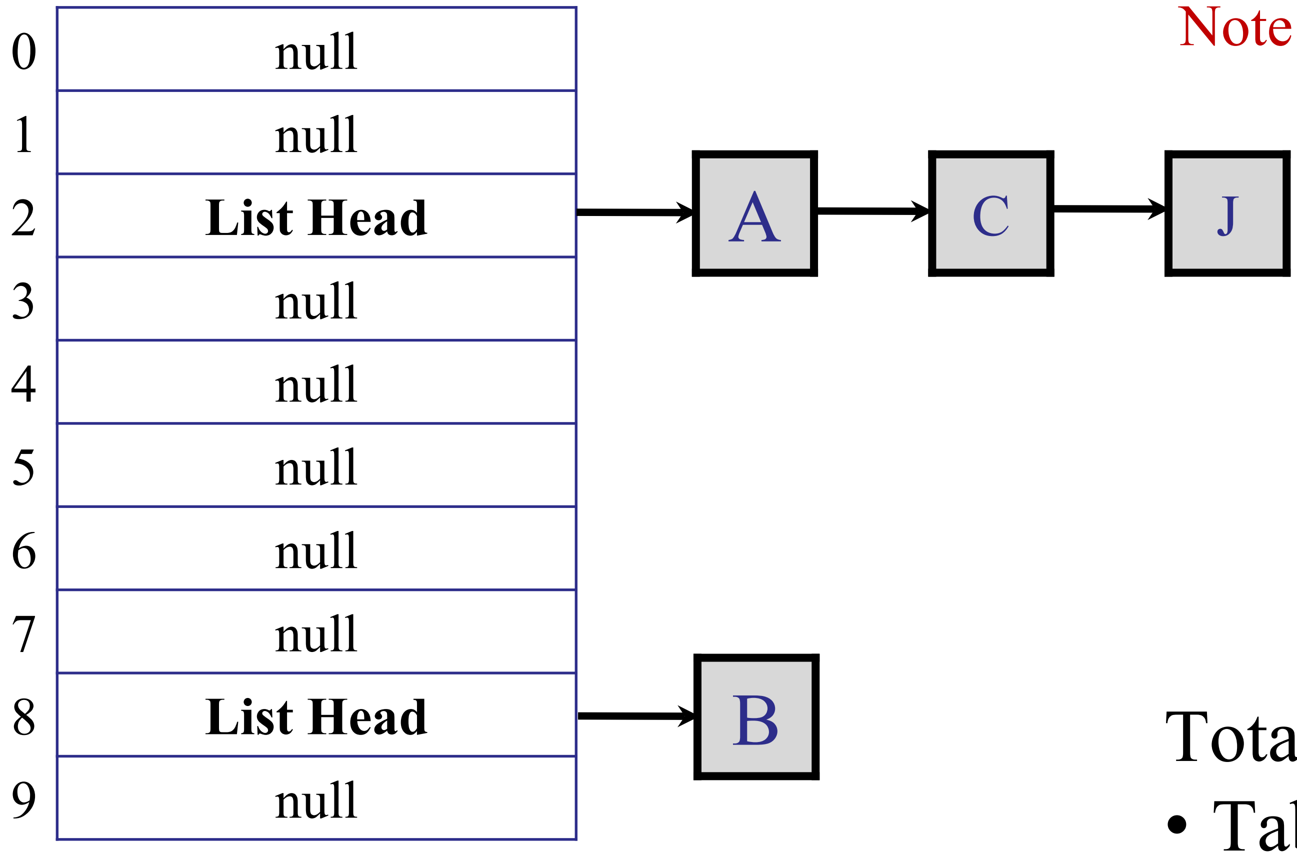


Note: h(A) == h(C) == h(J)

Total space:
- Number buckets: m
- Number entries: n

Chaining

Each bucket contains a linked list of items.



Note: h(A) == h(C) == h(J)

Total space: $O(m + n)$
- Table size: m
- Linked list size: n

Hashing with Chaining

Operations:

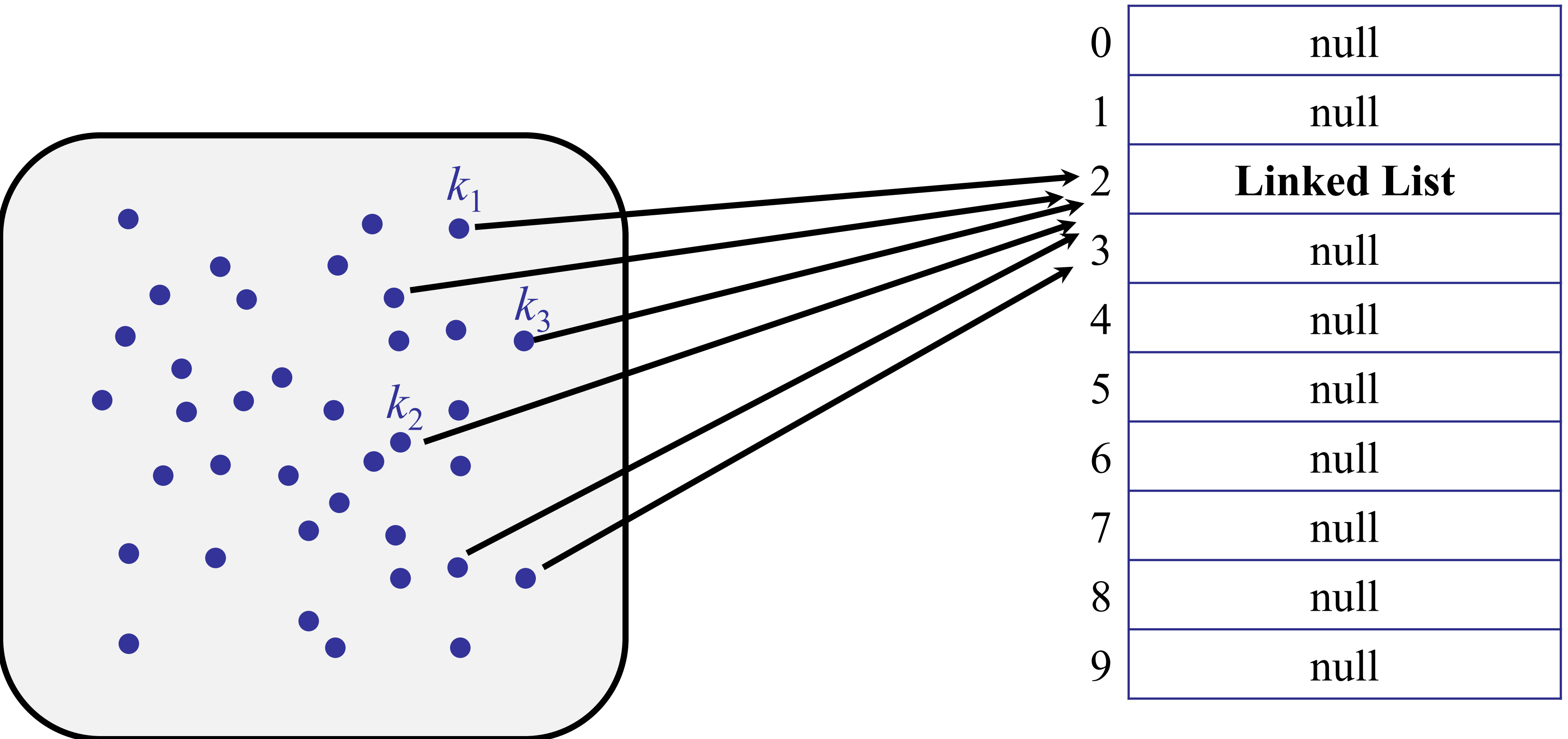
- insert(key, value)
  - Calculate h(key)
  - Lookup h(key) and add (key,value) to the linked list.

- search(key)
  - Calculate h(key)
  - Search for (key,value) in the linked list.

Hashing with Chaining

What if all keys hash to the same bucket!

- Worst-case search costs O(*n*)
- Oh no!



Let's be optimistic today.

The Simple Uniform Hashing Assumption

- Every key is equally likely to map to every bucket.
- Keys are mapped independently.

Assume hash function has this property, even if it may not!

Intuition:

- Each key is put in a random bucket.
- Then, as long as there are enough buckets, we won't get too many keys in any one bucket.

Let's be optimistic today.

The Simple Uniform Hashing Assumption

- Assume:
  - n items
  - m buckets
- Define: load(hash table) = n/m

  = average # items / bucket.

- Expected search time = 1 + *expected # items per bucket*

hash function + array access

linked list traversal

Let's be optimistic today.

The Simple Uniform Hashing Assumption

- Assume:
  - n items
  - m buckets
- Define: load(hash table) = n/m

  = average # items / buckets.

- Expected search time = $\underbrace{1}_{\text{hash function + array access}} + \underbrace{n/m}_{\text{linked list traversal}}$

Let's be optimistic today.

The Simple Uniform Hashing Assumption

- Assume:
  - n items
  - $m = \Omega(n)$ buckets, e.g., $m = 2n$

- Expected search time $= 1 + n/m$

  $= O(1)$

Hashing with Chaining

Searching:

- Expected search time = 1 + n/m = O(1)
- Worst-case search time = O(n)

Inserting:

- Worst-case insertion time = O(1)

** In this case, inserting allows duplicates…
Preventing duplicates requires searching.

Hashing with Chaining

What if you insert n elements in your hash table?

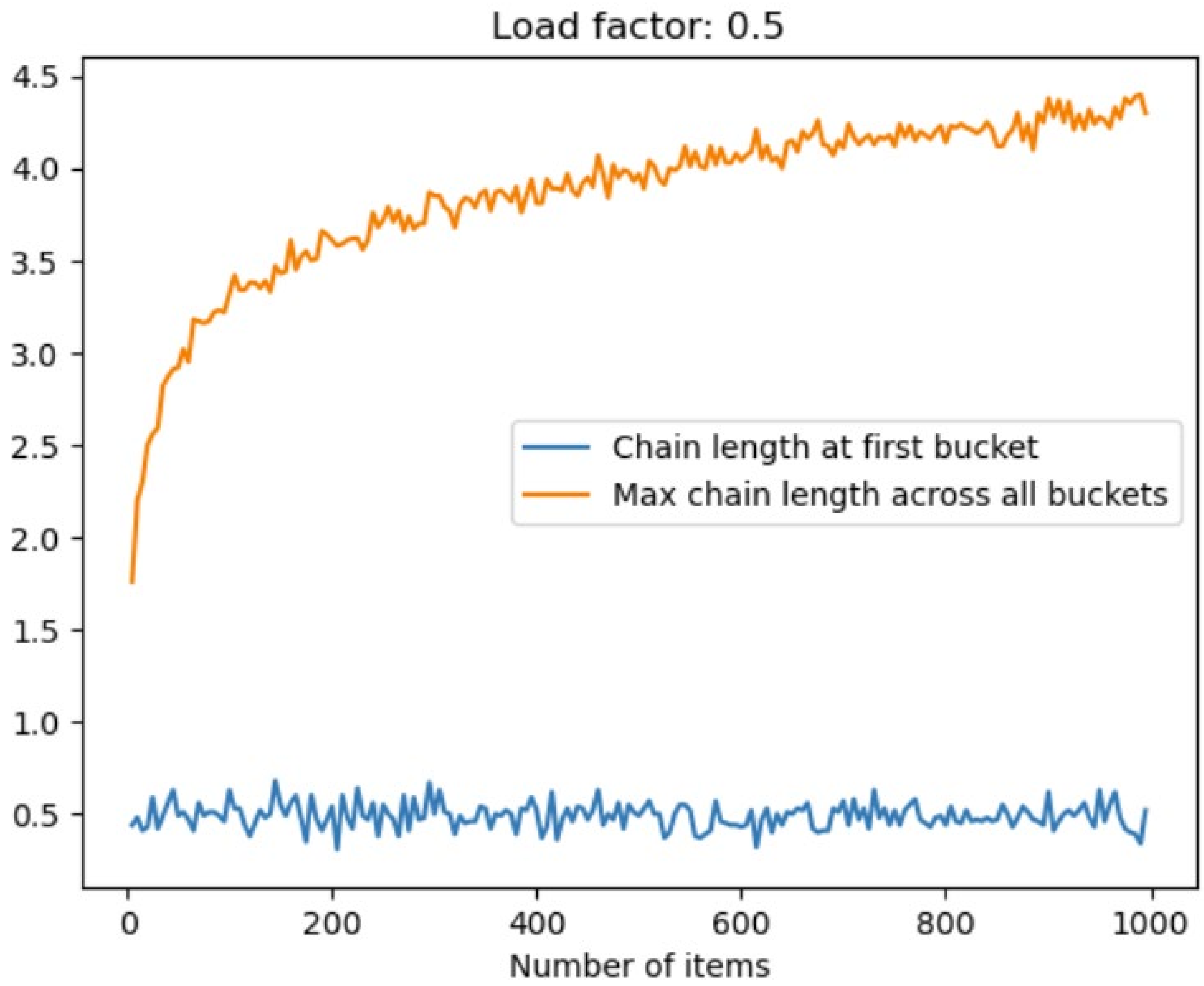
What is the expected *maximum* cost?

- Analogy:
  - Throw n balls in $m = n$ bins.
  - What is the maximum number of balls in a bin?

Cost: $\Theta(\log n / \log\log n)$

(See CS5330 for a proof.)

Hashing with Chaining



Some Remark

- How to reduce maximum chain length?

- **Power of two choices!**: Use two hash functions h_1 and h_2. For a key k, store in bucket $h_1(k)$ if it has the shorter chain, otherwise store in bucket $h_2(k)$. The maximum chain length becomes $O(\log \log n)$ instead of $O(\log n)$ in expectation!

- IMO, one of the most stunning facts in algorithm design.

Hashing: Recap

Problem: coping with large universe of keys

- Number of possible keys is very, very large.
- Direct Access Table takes too much space

Hash functions

- Use hash function to map keys to buckets.
- Sometimes, keys collide (inevitably!)
- Use linked list to store multiple keys in one bucket.

Analyze performance with simple uniform hashing.

- Expected number of keys / bucket is $O(n/m) = O(1)$.

Today

- Java hashing

Symbol Tables in Java

Symbol Tables in Java

java.util.Map

```
public interface  java.util.Map<Key, Value>

         void   clear()                   removes all entries
      boolean   containsKey(Object k)     is k in the map?
      boolean   containsValue(Object v)   is v in the map?
        Value   get(Object k)             get value for k
        Value   put(Key k, Value v)       adds (k,v) to table
        Value   remove(Object k)          remove mapping for k
          int   size()                    number of entries
```

Note: no successor / predecessor queries.

Symbol Tables in Java

Parameterized by key and value.
Not necessarily comparable

java.util.Map

```
public interface  java.util.Map<Key, Value>

         void  clear()                      removes all entries
      boolean  containsKey(Object k)        is k in the map?
      boolean  containsValue(Object v)      is v in the map?
        Value  get(Object k)                get value for k
        Value  put(Key k, Value v)          adds (k,v) to table
        Value  remove(Object k)             remove mapping for k
          int  size()                       number of entries
```

Note: no successor / predecessor queries.

Symbol Tables in Java

Search by key.

java.util.Map

public interface	java.util.Map<Key, Value>	
void	clear()	*removes all entries*
boolean	containsKey(Object k)	*is k in the map?*
boolean	containsValue(Object v)	*is v in the map?*
Value	get(Object k)	*get value for k*
Value	put(Key k, Value v)	*adds (k,v) to table*
Value	remove(Object k)	*remove mapping for k*
int	size()	*number of entries*

Note: no successor / predecessor queries.

Symbol Tables in Java

java.util.Map

Search by value.
(May not be efficient.)

```
public interface  java.util.Map<Key, Value>

         void   clear()                   removes all entries
      boolean   containsKey(Object k)     is k in the map?
      boolean   containsValue(Object v)   is v in the map?
        Value   get(Object k)             get value for k
        Value   put(Key k, Value v)       adds (k,v) to table
        Value   remove(Object k)          remove mapping for k
          int   size()                    number of entries
```

Note: no successor / predecessor queries.

Symbol Tables in Java

java.util.Map

Can use any Object as key?

```
public interface  java.util.Map<Key, Value>

         void   clear()                       removes all entries
      boolean   containsKey(Object k)         is k in the map?
      boolean   containsValue(Object v)       is v in the map?
        Value   get(Object k)                 get value for k
        Value   put(Key k, Value v)           adds (k,v) to table
        Value   remove(Object k)              remove mapping for k
          int   size()                        number of entries
```

Note: no successor / predecessor queries.

Symbol Tables in Java

java.util.Map

Put new (key, value) in table.

public interface java.util.Map<Key, Value>

void	clear()	*removes all entries*
boolean	containsKey(Object k)	*is k in the map?*
boolean	containsValue(Object v)	*is v in the map?*
Value	get(Object k)	*get value for k*
Value	put(Key k, Value v)	*adds (k,v) to table*
Value	remove(Object k)	*remove mapping for k*
int	size()	*number of entries*

Note: no successor / predecessor queries.

Map Interface in Java

java.util.Map<Key, Value>

- No duplicate keys allowed.
- No *mutable* keys

  If you use an *object* as a key, then you can’t modify that object later.

Symbol Table

Key Mutability

```
SymbolTable<Time, Plane> t =
           new SymbolTable<Time, Plane>();

Time  t1 = new Time(9:00);
Time  t2 = new Time(9:15);

t.insert(t1, "SQ0001");
t.insert(t2, "SQ0002");

t1.setTime(10:00);

x = new Time(9:00);
t.search(x);
```

What time does this plane depart at?

Symbol Table

Key Mutability

Moral: Keys should be immutable.

Examples: Integer, String

```
SymbolTable<Time, Plane> t =
           new SymbolTable<Time, Plane>();

Time  t1 = new Time(9:00);
Time  t2 = new Time(9:15);

t.insert(t1, "SQ0001");
t.insert(t2, "SQ0002");

t1.setTime(10:00);

x = new Time(9:00);
t.search(x);
```

Design Decisions

Allow duplicate keys?

- No: need to search on insertion
- Yes: faster insertion

What to do if user inserts duplicate key?

- Replace existing key.
- Add new value (i.e., key has two values).
- Error.

Insert empty/null value?

- Deletes existing (key, value) pair.
- Creates a null value.
- Error.

Symbol Tables in Java

java.util.Map

```
           public interface  java.util.Map<Key, Value>
-------------------------------------------------------------------------
Set<Map.Entry<Key, Value>  entrySet()      set of all mappings
                 Set<Key>  keySet()        set of all keys
        Collection<Value>  values()        collection of all values
```

Note: not sorted

not necessarily efficient to work with these sets/collections.

What is wrong here?

Example:

There is a bug here!

```
Map<String, Integer> ageMap = new Map<String, Integer>();

ageMap.put(“Alice”, 32);
ageMap.put(“Bernice”, 84);
ageMap.put(“Charlie”, 7);

Integer age = ageMap.get(“Alice”)
```

- Key-type: String
- Value-type: Integer

What is wrong here?

Example:

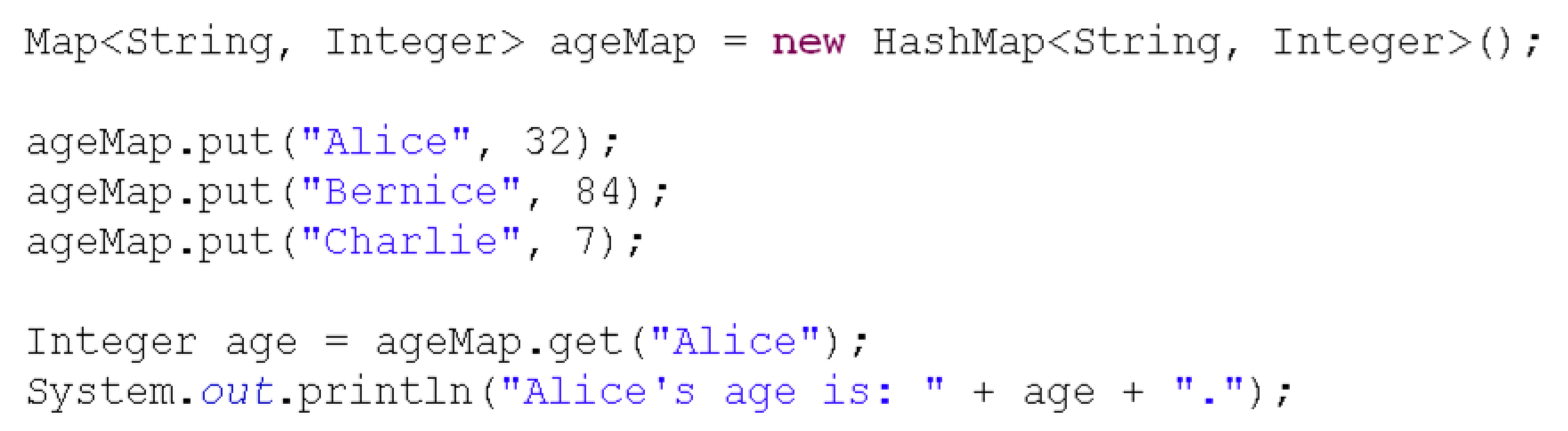
Map is an interface!
Cannot instantiate an interface.

```
Map<String, Integer> ageMap = new Map<String, Integer>();

ageMap.put("Alice", 32);
ageMap.put("Bernice", 84);
ageMap.put("Charlie", 7);

Integer age = ageMap.get("Alice")
```

- Key-type: String
- Value-type: Integer

Map Class in Java

Example: HashMap

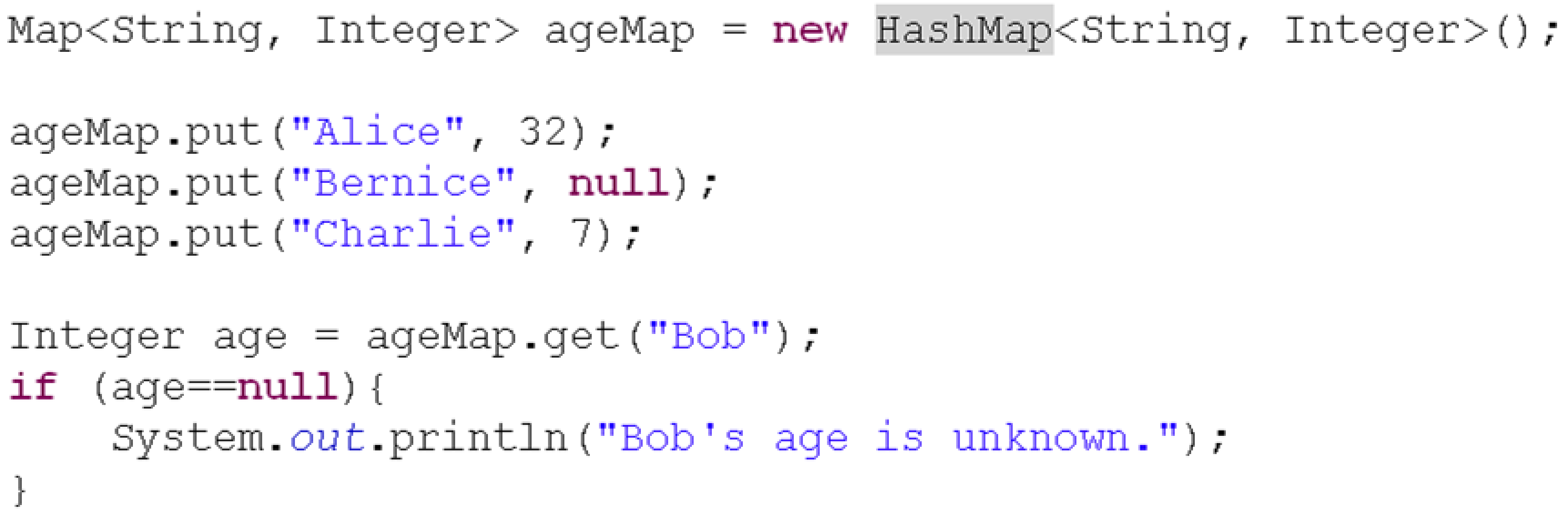


```
Map<String, Integer> ageMap = new HashMap<String, Integer>();

ageMap.put("Alice", 32);
ageMap.put("Bernice", 84);
ageMap.put("Charlie", 7);

Integer age = ageMap.get("Alice");
System.out.println("Alice's age is: " + age + ".");
```

- Key-type: String
- Value-type: Integer

Map Class in Java

Example: HashMap



```
Map<String, Integer> ageMap = new HashMap<String, Integer>();

ageMap.put("Alice", 32);
ageMap.put("Bernice", null);
ageMap.put("Charlie", 7);

Integer age = ageMap.get("Bob");
if (age==null){
    System.out.println("Bob's age is unknown.");
}
```

- Returns “null” when key is not in map.
- Returns “null” when value is null.

Map Classes in Java

HashMap

Symbol Table

- containsKey
- containsValue
- entrySet
- get
- isEmpty
- keySet
- put
- putAll
- remove
- values

TreeMap

Dictionary

- containsKey
- containsValue
- entrySet
- get
- isEmpty
- keySet
- put
- putAll
- remove
- values

Map Classes in Java

HashMap

Symbol Table

TreeMap

Dictionary

- ceilingEntry
- ceilingKey
- descendingKeySet
- firstEntry
- firstKey
- floorEntry
- floorKey
- headMap
- higherEntry
- higherKey
- … (and more)

Hashing in Java

How does your program know which hash function to use?

```
HashMap<MyFoo, Integer> hmap = new ...
MyFoo foo = new MyFoo();


hmap.put(foo, 8);
```

Java Hash Functions

Every object supports the method:

```
int hashCode()
```

Java Object

Every class implicitly extends Object

public class	**Object**	
Object	clone()	*creates a copy*
boolean	**equals(Object obj)**	***is obj equal to this?***
void	finalize()	*used by garbage collector*
Class	getClass()	*returns class*
int	**hashCode()**	***calculates hash code***
void	notify()	*wakes up a waiting thread*
void	notifyAll()	*wakes up all waiting threads*
String	**toString()**	***returns string representation***
void	wait(…)	*wait until notified*

Hashing in Java

How does your program know which hash function to use?

```
HashMap<MyFoo, Integer> hmap = new ...
MyFoo foo = new MyFoo();


int hash = foo.hashCode();


hmap.put(foo, 8);
```

Java Hash Functions

Every object supports the method:

```
int hashCode()
```

Rules:

- Always returns the same value, if the object hasn't changed.
- If two objects are equal, then they return the same hashCode.

No random hashcodes!

Is it legal for every object to return 32?

Java Hash Functions

Every object supports the method:

```
    int hashCode()
```

Rules:

- Always returns the same value, if the object hasn't changed.
- If two objects are equal, then they return the same hashCode.

Is it *legal* for every object to return 32? (YES)

Java Hash Functions

Every object supports the method:

```
int hashCode()
```

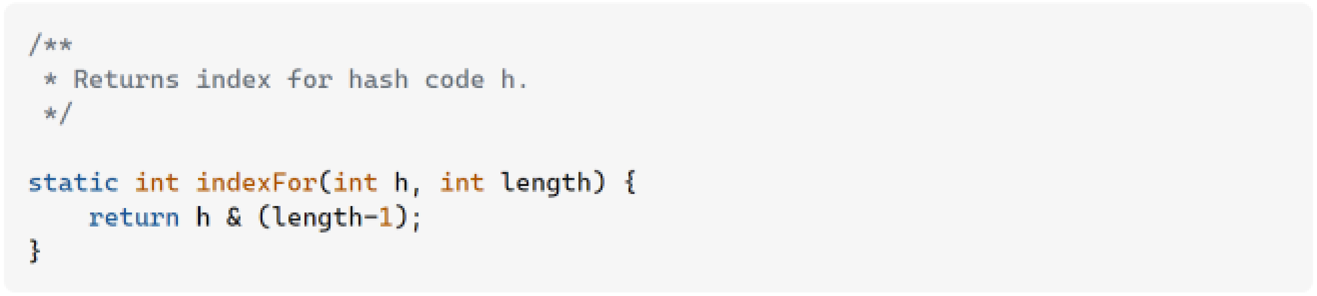
Default Java implementation:

- hashCode returns the memory location of the object
- Every object hashes to a different location

Must implement/override `hashCode()` for your class.

Java Hash Functions

- **CAVEAT**: `hashCode()` returns an int, so using it directly to specify the bucket would require a table of size 2^{32}…not good!



```java
/**
 * Returns index for hash code h.
 */

static int indexFor(int h, int length) {
    return h & (length-1);
}
```

- In HashMap, the `hashCode` is truncated to fit the table size. Usually, table size is a power of 2, so above code extracts suffix of appropriate length from `hashCode`.

Java Hash Functions

e.g. table size of 16

16 = 10000 (in binary/base 2)

15 - 1 = 01111 (in binary/base 2)

Java Hash Functions

e.g. table size of 16

16 = 10000 (in binary/base 2)

15 - 1 = 01111 (in binary/base 2)

e.g. table size of 16

example hash code = 45

45 = 101101 (in binary)

Java Hash Functions

e.g. table size of 16

16 = 10000 (in binary/base 2)

15 - 1 = 01111 (in binary/base 2)

e.g. table size of 16

example hash code = 45

45 = 101101 (in binary)   use index = 1101 (in binary)
i.e. index = 13

Java Library Classes

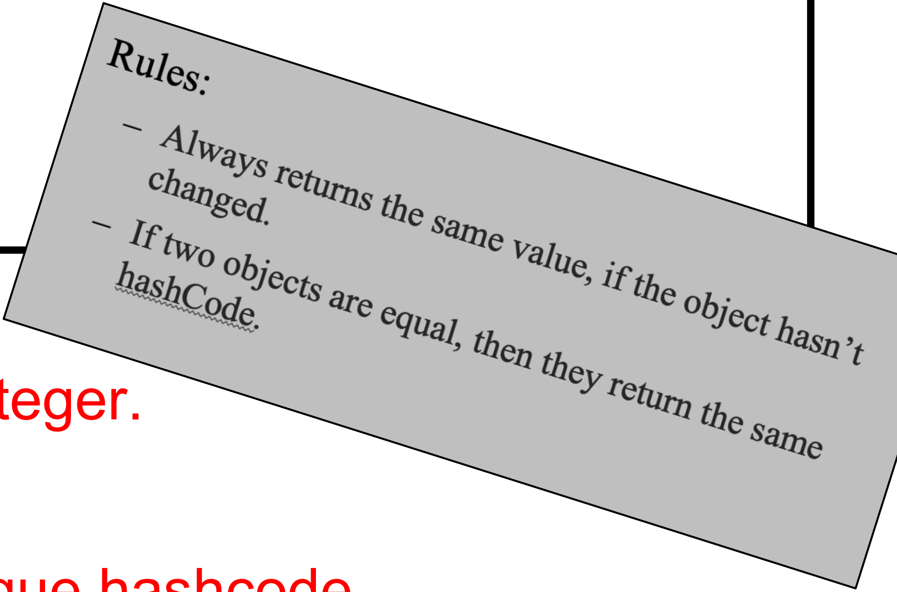
`Integer`

`Long`

`String`

Integer

```
public int hashCode() {
   return value;
}
```


Rules:

- Always returns the same value, if the object hasn't changed.
- If two objects are equal, then they return the same hashCode.

Note: hashcode is always a 32-bit integer.

Note: every 32-bit integer gets a unique hashcode.

What do you do for smaller hash tables?
Can there be collisions?

Long

Collision can happen!

```
public int hashCode() {
   return (int)(value ^ (value >>> 32));
}
```

$$\text{hash}(\overbrace{0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0}^{\text{32 bits}}\ \overbrace{0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0}^{\text{32 bits}})$$

$$\begin{array}{rl} & 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \\ \text{XOR} & 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \\ \hline & 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0 \end{array}$$

String

```
public int hashCode() {
  int h = hash; // only calculate hash once
  if (h == 0 && count > 0) {  // empty = 0
      int off = offset;
      char val[] = value;
      int len = count;
      for (int i = 0; i < len; i++) {
          h = 31*h + val[off++];
      }
      hash = h;
  }
  return h;
}
```

String

HashCode calculation:

```
hash = s[0]*31^(n-1) +
       s[1]*31^(n-2) +
       s[2]*31^(n-3) +
       ... +
       s[n-2]*31 +
       s[n-1]
```

Why did they choose 31?

String

HashCode calculation:

```
hash = s[0]*31^(n-1) +
       s[1]*31^(n-2) +
       s[2]*31^(n-3) +
       ... +
       s[n-2]*31 +
       s[n-1]
```

Why did they choose 31? `Prime,` 2^5`-1`

Creating a new class

```
public class Pair {
    private int first;
    private int second;

    Pair(int a, int b){
        first = a;
        second = b;
    }
}
```

Creating a new class

```
public void testPair() {

    HashMap<Pair, Integer> htable =
                new HashMap<Pair,
Integer>();

    Pair one = new Pair(20, 40);
    htable.put(one, 7);

    Pair two = new Pair(20, 40);
    int question = htable.get(two);
}
```

htable.get(new Pair(20, 40)) == ?

1. 1
2. 7
3. 11
4. [✓ null]

Creating a new class

```
Pair one = new Pair(20, 40);
Pair two = new Pair(20, 40);

one.hashCode() != two.hashCode()
```

Creating a new class

```
Pair one = new Pair(20, 40);
Pair two = new Pair(20, 40);
htable.put(one, "first item");
```

```
htable.get(one) → "first item"

htable.get(two) → null
```

Creating a new class

```
public class Pair {
    private int first;
    private int second;

    Pair(int a, int b){
        first = a;
        second = b;
    }

    int hashCode(){
        return (first ^ second);
    }
}
```

Creating a new class

```
Pair one = new Pair(20, 40);
Pair two = new Pair(20, 40);
htable.put(one, "first item");
```

```
htable.get(one) → "first item"
htable.get(two) → null

one.equals(two) → false
```

Java Hash Functions

Every object supports the method:

```
int hashCode()
```

Rules:

- Always returns the same value, if the object hasn't changed.
- If two objects are equal, then they return the same hashCode.
- **Must redefine .equals to be consistent with hashCode.**

Creating a new class

```
Pair one = new Pair(20, 20);
Pair two = new Pair(20, 20);
htable.put(one, “first item”);

htable.get(one)  → “first item”

htable.get(two)  → null
```

Java Hash Functions

Every object supports the method:

```
boolean equals(Object o)
```

Rules:

- **Reflexive**: x.equals(x) == true
- **Symmetric**: x.equals(y) == y.equals(x)
- **Transitive**: x.equals(y), y.equals(z) $\rightarrow$ x.equals(z)
- **Consistent**: always returns the same answer
- **Null is null**: x.equals(null) $\rightarrow$ false

Java Hash Functions

Every object supports the method:

```
boolean equals(Object o)
```

```
boolean equals(Object p){
      if (p == null) return false;
      if (p == this) return true;

      if (!(p instanceOf Pair)) return false;
      Pair pair = (Pair)p;

      if (pair.first != first) return false;
      if (pair.second != second) return
false;
      return true;
```

Java HashMap

```
public V get(Object key) {
    if (key == null) return getForNullKey();
    int hash = hash(key.hashCode());
    for (Entry<K,V> e = table[indexFor(hash,table.length)];
         e != null;
         e = e.next)
    {
          Object k;
          if (e.hash==hash
    &&((k=e.key)==key)||key.equals(k)))
                return e.value;
    }
    return null;
}
```

Java HashMap

```
public V get(Object key) {
   if (key == null) return getForNullKey();
   int hash = hash(key.hashCode());
   for (Entry<K,V> e = table[indexFor(hash,table.length)];
        e != null;
        e = e.next)
   {
         Object k;
         if (e.hash==hash
   &&((k=e.key)==key)||key.equals(k)))
               return e.value;
   }
   return null;
}
```

Java HashMap

```
public V get(Object key) {
    if (key == null) return getForNullKey();
    int hash = hash(key.hashCode());
    for (Entry<K,V> e = table[indexFor(hash,table.length)];
           e != null;
           e = e.next)
    {
            Object k;
            if (e.hash==hash
    &&((k=e.key)==key)||key.equals(k)))
                     return e.value;
    }
    return null;
}
```

Java HashMap

```
public V get(Object key) {
    if (key == null) return getForNullKey();
    int hash = hash(key.hashCode());
    for (Entry<K,V> e = table[indexFor(hash,table.length)];
            e != null;
            e = e.next)
    {
             Object k;
             if (e.hash==hash
    &&((k=e.key)==key)||key.equals(k)))
                    return e.value;
    }
    return null;
}
```

Java HashMap

```
public V get(Object key) {
    if (key == null) return getForNullKey();
    int hash = hash(key.hashCode());
    for (Entry<K,V> e = table[indexFor(hash,table.length)];
         e != null;
         e = e.next)
    {
          Object k;
          if (e.hash==hash
    &&((k=e.key)==key)||key.equals(k)))
                return e.value;
    }
    return null;
}
```

Java HashMap

```
public V get(Object key) {
    if (key == null) return getForNullKey();
    int hash = hash(key.hashCode()); ????
    for (Entry<K,V> e = table[indexFor(hash,table.length)];
            e != null;
            e = e.next)
    {
            Object k;
            if (e.hash==hash
    &&((k=e.key)==key)||key.equals(k)))
                    return e.value;
    }
    return null;
}
```

Java HashMap

```
// This function ensures that hashCodes that differ only
// by constant multiples at each bit position have a
// bounded number of collisions (approximately 8 at
// default load factor).
```

```
static int hash(int h) {
   h ^= (h >>> 20) ^ (h >>> 12);
   return h ^ (h >>> 7) ^ (h >>> 4);
}
```

Java HashMap

```
// This function ensures that hashCodes that differ only
// by constant multiples at each bit position have a
// bounded number of collisions (approximately 8 at
// default load factor).
```

```
static int hash(int h) {
   h ^= (h >>> 20) ^ (h >>> 12);
   return h ^ (h >>> 7) ^ (h >>> 4);
}
```

Eldon! Show them the documentation!

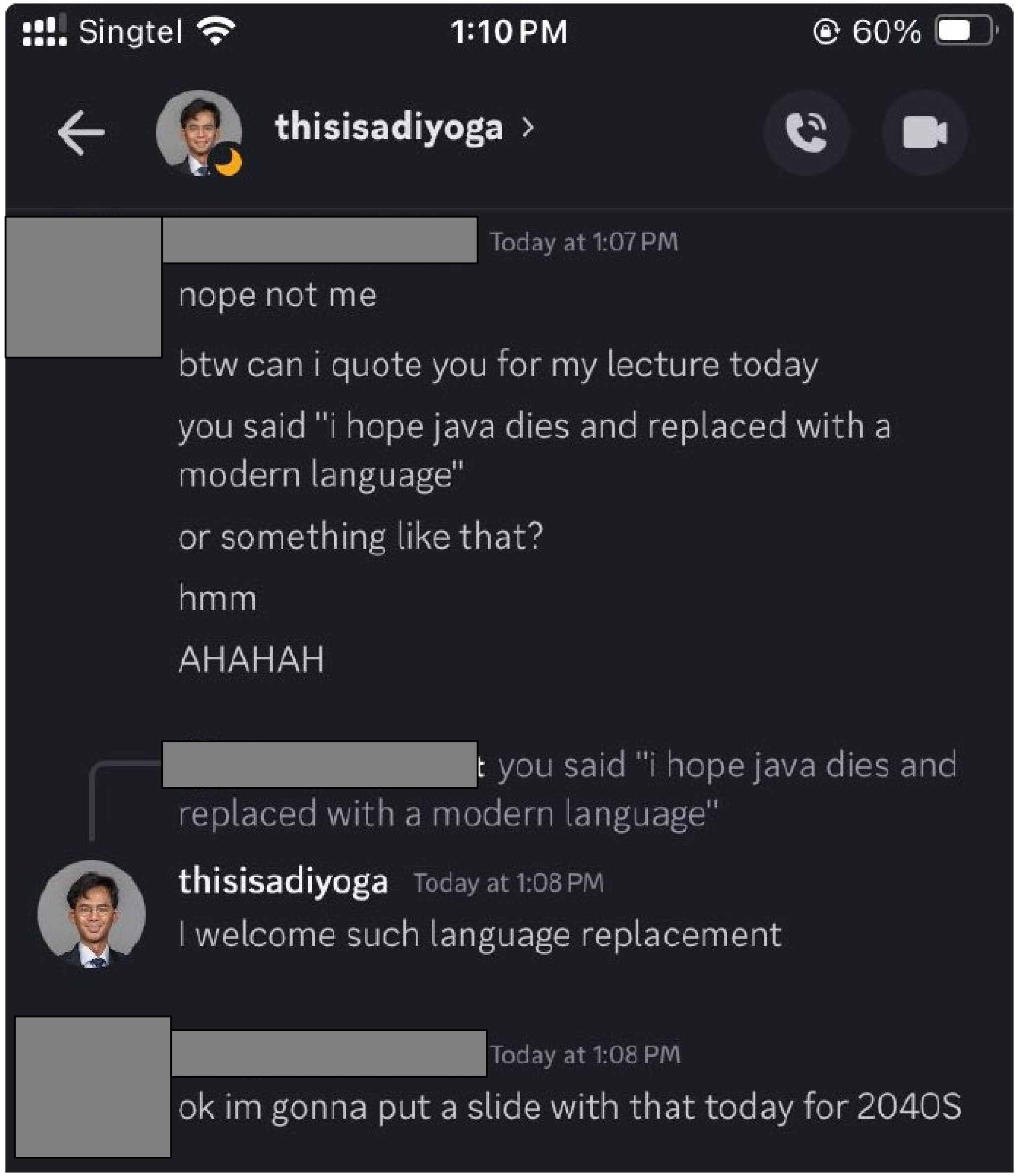
- CS2040S Slides Reminder

Java HashMap

```
public V get(Object key) {
   if (key == null) return getForNullKey();
   int hash = hash(key.hashCode());
   for (Entry<K,V> e = table[indexFor(hash,table.length)];
        e != null;
        e = e.next)
   {
         Object k;
         if (e.hash==hash
   &&((k=e.key)==key)||key.equals(k)))
               return e.value;
   }
   return null;
}
```

Java checks if the key is equal to the item in the hash table before returning it!

Java HashMap



[redacted] [redacted] Today at 1:07 PM

nope not me

btw can i quote you for my lecture today

you said "i hope java dies and replaced with a modern language"

or something like that?

hmm

AHAHAH

[redacted]t you said "i hope java dies and replaced with a modern language"

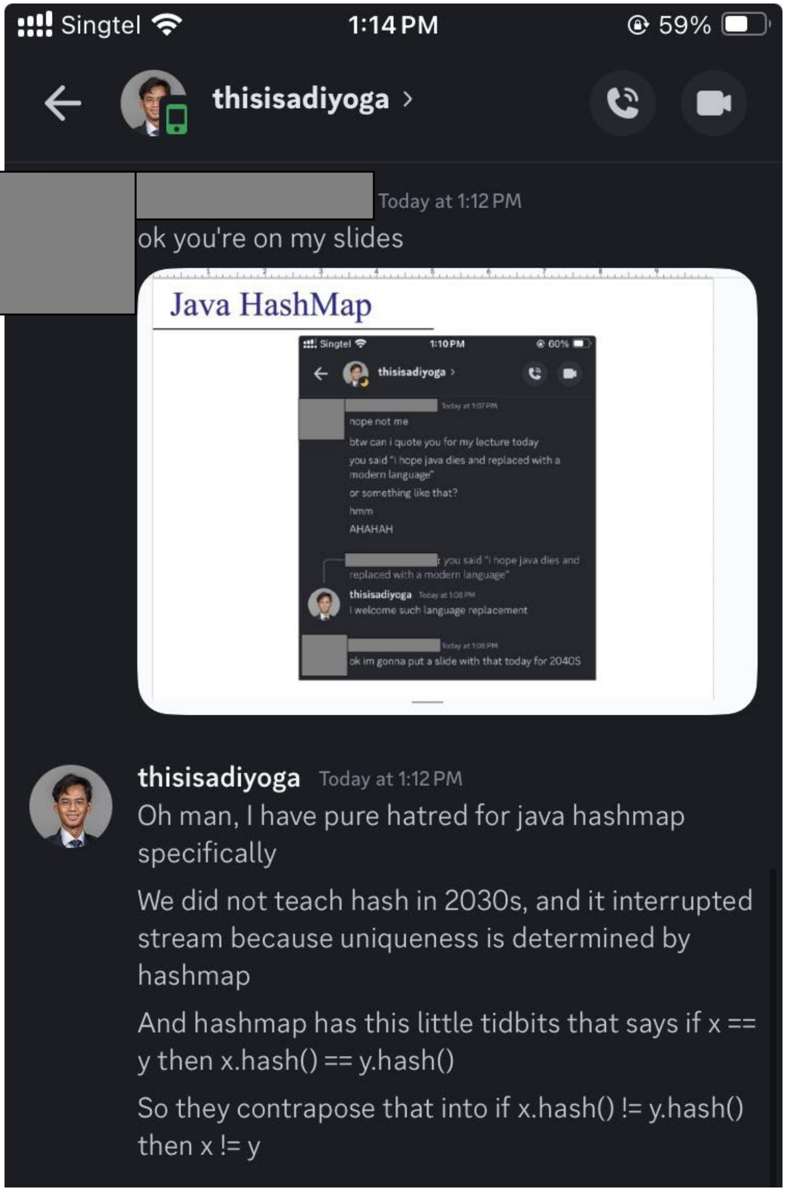
thisisadiyoga Today at 1:08 PM

I welcome such language replacement

[redacted] [redacted] Today at 1:08 PM

ok im gonna put a slide with that today for 2040S

Java HashMap



Singtel 1:14 PM 59%

thisisadiyoga

[redacted] Today at 1:12 PM

ok you're on my slides

Java HashMap

Singtel 1:10 PM 60%

thisisadiyoga

[redacted] Today at 1:07 PM

nope not me

btw can i quote you for my lecture today

you said "i hope java dies and replaced with a modern language"

or something like that?

hmm

AHAHAH

[redacted]: you said "i hope java dies and replaced with a modern language"

thisisadiyoga Today at 1:08 PM

I welcome such language replacement

[redacted] Today at 1:08 PM

ok im gonna put a slide with that today for 2040S

thisisadiyoga Today at 1:12 PM

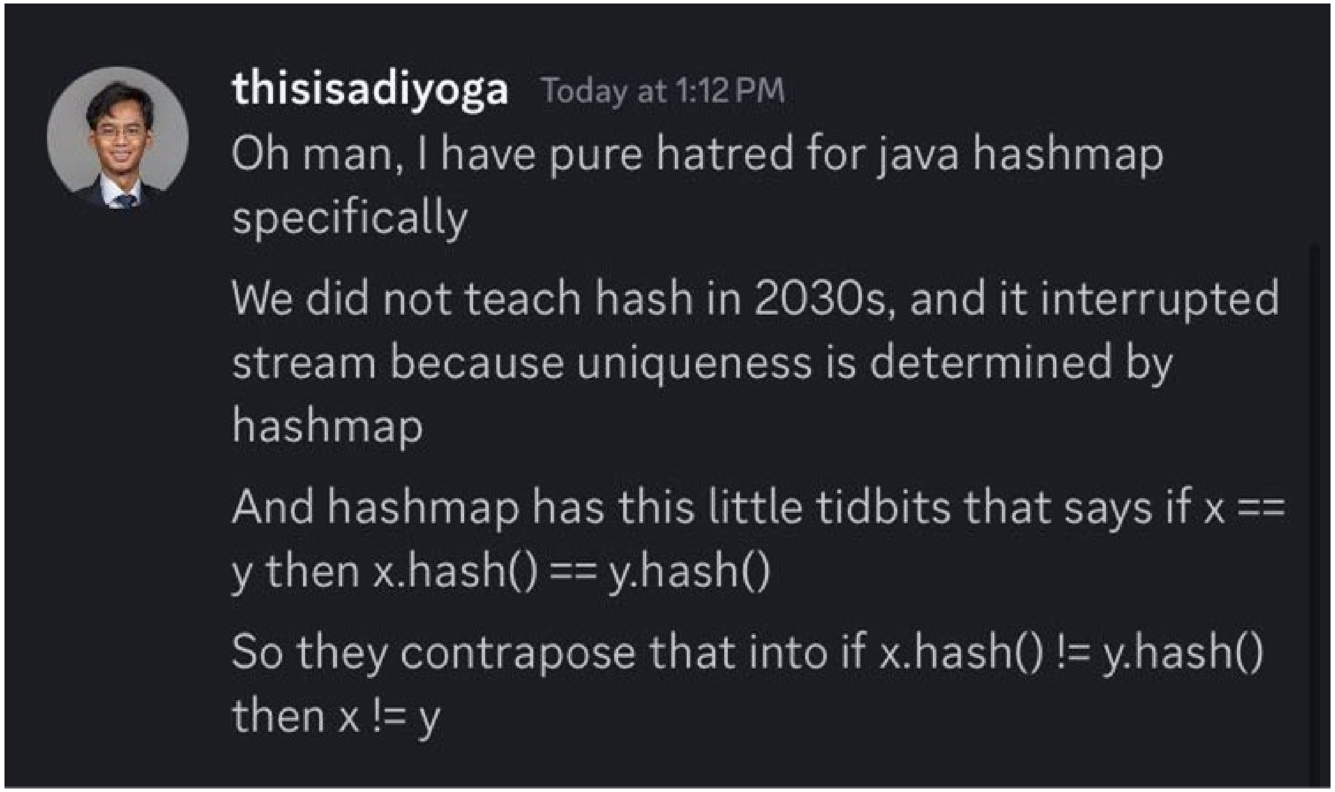
Oh man, I have pure hatred for java hashmap specifically

We did not teach hash in 2030s, and it interrupted stream because uniqueness is determined by hashmap

And hashmap has this little tidbits that says if x == y then x.hash() == y.hash()

So they contrapose that into if x.hash() != y.hash() then x != y

Java HashMap



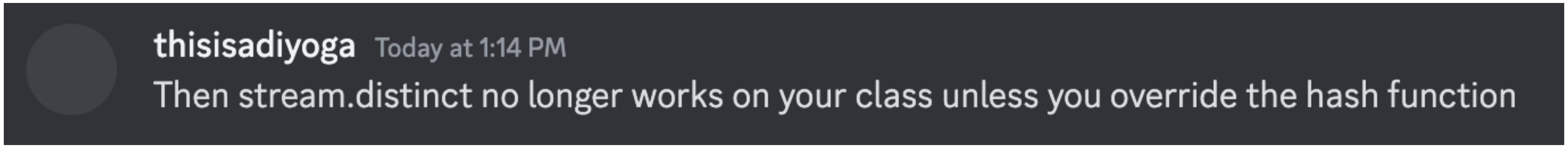
thisisadiyoga Today at 1:12 PM

Oh man, I have pure hatred for java hashmap specifically

We did not teach hash in 2030s, and it interrupted stream because uniqueness is determined by hashmap

And hashmap has this little tidbits that says if x == y then x.hash() == y.hash()

So they contrapose that into if x.hash() != y.hash() then x != y



thisisadiyoga Today at 1:14 PM

Then stream.distinct no longer works on your class unless you override the hash function

Today & Next Week

- **Java hashing**
- Collision resolution: open addressing
- Table (re)sizing