Programming Languages and Techniques (CIS120)

Lecture 37

April 28, 2014

TreeSet and HashSet

Announcements

- Game project due Wednesday at midnight (HARD deadline)
- Check your grades online, should be up-to-date
- Final Exam
  - Wednesday, May 7th 9-11 AM
  - DRLB A1, Last name A-N
    DRLB A8, Last name P-Z
  - Old exams posted on the course website for review
  - OH will continue until the exam
  - Review session: Saturday, May 3rd 6-9PM in Levine 101
  - Mock exam: Sunday, May 4th 6-9PM in Levine 101

How is HW10 going so far?

1. not started
2. got an idea, but still working out the details
3. bugs, bugs, bugs everywhere
4. done and submitted!

```java
public class Game {
    private boolean x;

    public Game() {
       boolean x = true;
    }
    public boolean foo() {
       return x;
    }

    public static void main(String[] args) {
        System.out.println(new Game().foo());
    }

}
```

What gets printed to the console?

1. true
2. false
3. NullPointerException

Collections & User-defined classes

From Wednesday

```java
public class Point {
    private final int x;
    private final int y;
    public Point(int x, int y) { this.x = x; this.y = y; }
    public int getX() { return x; }
    public int getY() { return y; }
}

// somewhere in main…
List<Point> l = new LinkedList<Point>();
l.add(new Point(1,2));
System.out.println(l.contains(new Point(1,2)));
```

What gets printed to the console?

1. true
2. false

TreeSet

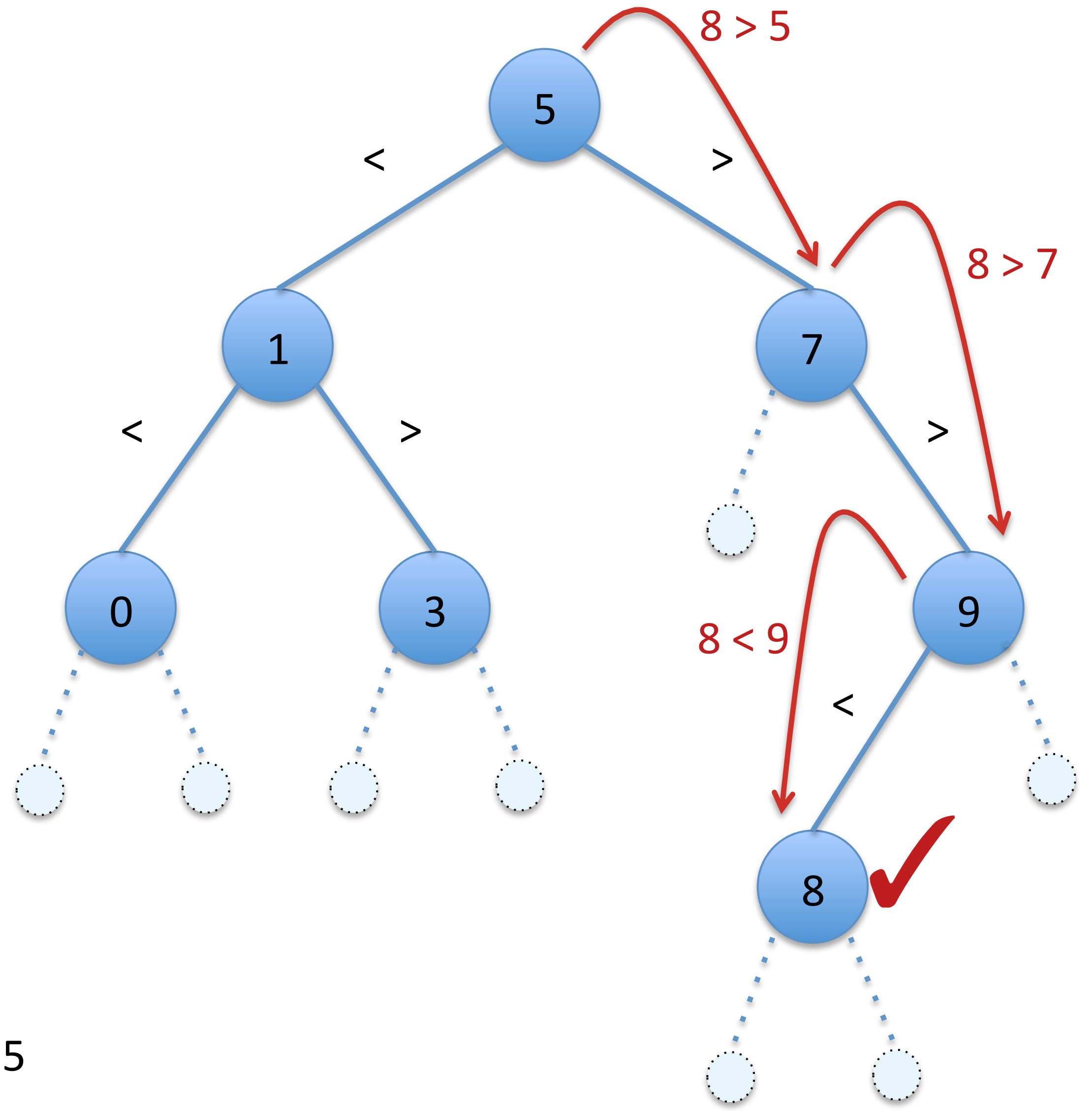
```
public class Point {
   private final int x;
   private final int y;
   public Point(int x, int y) { this.x = x; this.y = y; }
   public int getX() { return x; }
   public int getY() { return y; }
}

// somewhere in main…
Set<Point> l = new TreeSet<Point>();
l.add(new Point(1,2));
System.out.println(l.contains(new Point(1,2)));
```

What gets printed to the console?

1. true
2. false
3. Exception in thread "main" java.lang.ClassCastException: Point cannot be cast to java.lang.Comparable

Search in a BST: `(lookup t 8)`



Comparison Operators

- OCaml's comparison operators (such as < and >) automatically work for user-defined types

```
type point = { x:int;  y:int }

;; print_string (if {x=1;y=2} < {x=2;y=1}
                    then "true" else "false")
```

What gets printed to the console?

1. true
2. false
3. I have no idea

- Java requires classes to explicitly implement the "Comparable<T>" interface

Comparable<T>

- Classes must implement Comparable<T> to be used with TreeSet

```
interface Comparable<T>  {
    public int compareTo(T o);
}
```

- Returns:
  - a negative number when o is less than this object
  - zero when o is equal to this object
  - a positive number when o is greater than this object

Implementing compareTo

```
public class Point implements Comparable<Point> {
    private final int x;
    private final int y;
    public Point(int x, int y) { this.x = x; this.y = y; }

    @Override public int compareTo(Point o) {
        final int BEFORE = -1;
        final int EQUAL  = 0;
        final int AFTER  = 1;

        if (this.x < o.x) return BEFORE;
        if (this.x > o.x) return AFTER;

        if (this.y < o.y) return BEFORE;
        if (this.y > o.y) return AFTER;

        return EQUAL;
    }
}
```

```
// optimization
if (this == o) return EQUAL;
```

What is the result of Point(1,2).compareTo(null)?

1. -1
2. 0
3. 1
4. NullPointerException

Contract for compareTo

- *Anticommutivity*: `x.compareTo(y)` is the opposite of `y.compareTo(x)` (or they are both zero)
- *Exception symmetry*: `x.compareTo(y)` throws the same exceptions as `y.compareTo(x)`
- *Transitivity:* if `x.compareTo(y) > 0` and `y.compareTo(z) > 0` then `x.compareTo(z) > 0` (and same for <)
- *Consistency*: `if x.compareTo(y) == 0` then `x.compareTo(z)` returns the same result as `y.compareTo(z)`
- *Consistency with equals (strongly recommended)*: `x.compareTo(y) == 0` if and only if `x.equals(y)`

HashSet & HashMap

Clicker quiz

```
public class Point {
    private final int x;
    private final int y;
    public Point(int x, int y) { this.x = x; this.y = y; }
    public int getX() { return x; }
    public int getY() { return y; }
}

// somewhere in main…
Set<Point> l = new HashSet<Point>();
l.add(new Point(1,2));
System.out.println(l.contains(new Point(1,2)));
```

What gets printed to the console?

1. true
2. false
3. I have no idea

Hash Maps: The Big Idea

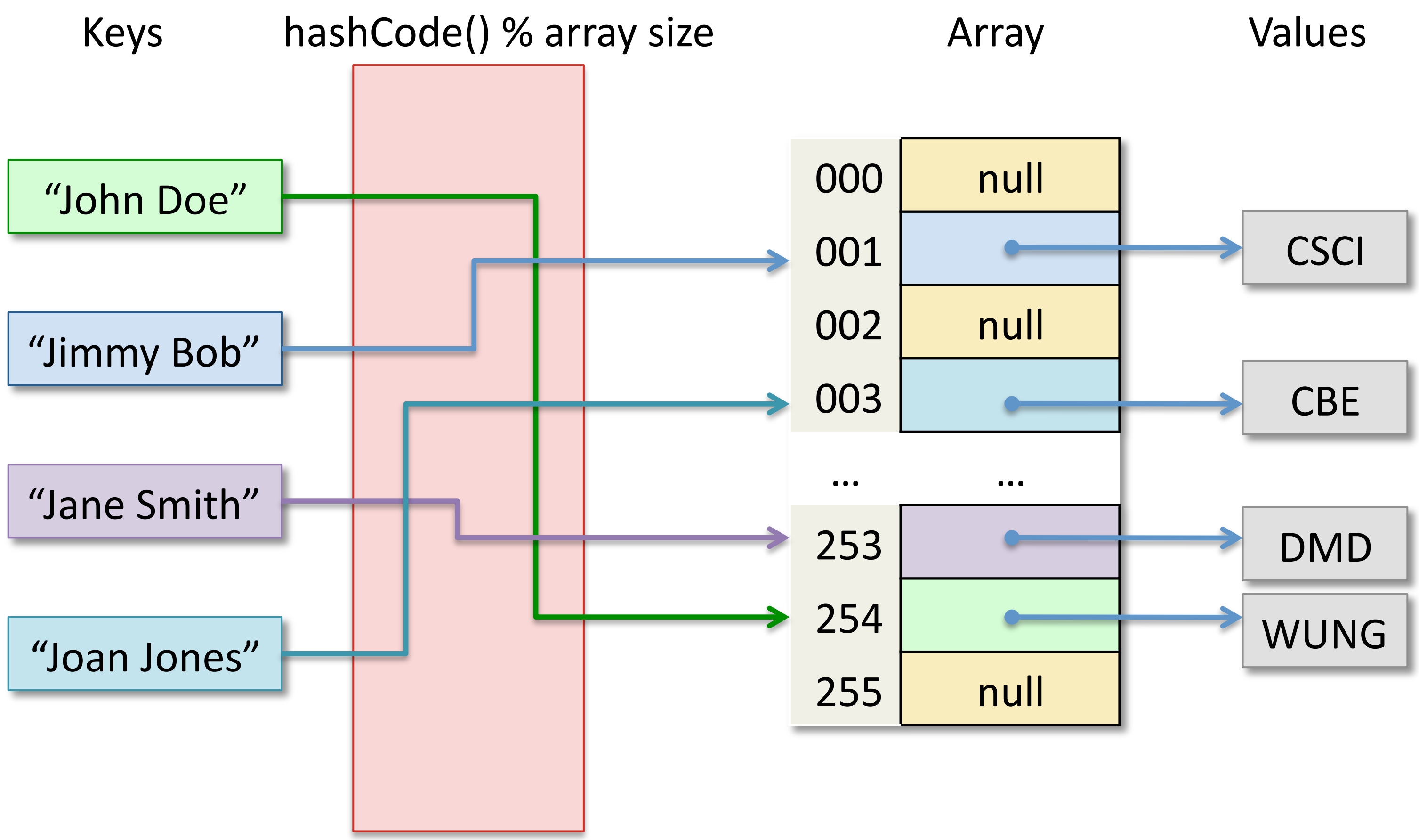
Combine:

- the advantage of arrays:
  - *efficient* random access to its elements
- with the advantage of a map datastructure
  - arbitrary keys (not just integers!)

How?

- Create an index into an array by *hashing* the data in the key to turn it into an int
  - The hashCode method of the Object class does this
- Resize the array as the number of stored elements grows

Hash Maps, Pictorially



A schematic HashMap taking Strings (student names) to Undergraduate Majors. Here, `"John Doe".hashCode()` returns an integer n, its *hash*, such that n mod 256 is 254.

Hash Collisions

- Uh Oh: Indices derived via hashing may not be unique!

  ```
  "Jane Smith".hashCode() % 256 → 253
  "Joe Schmoe".hashCode() % 256 → 253
  ```

- Good hashCode functions make it *unlikely* that two keys will produce the same hash

- But, it can happen that two keys do produce the same index – that is, their hashes *collide*

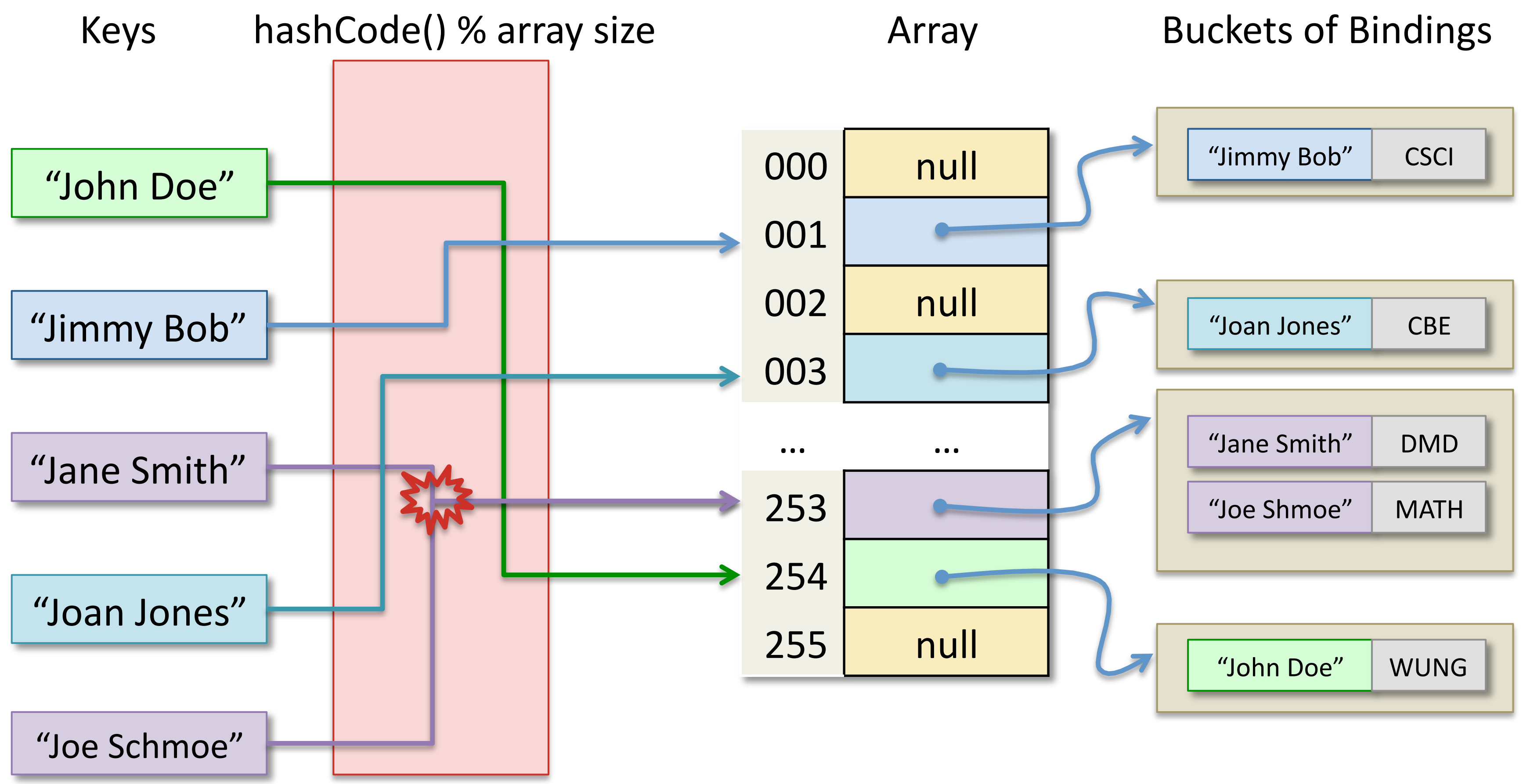
Bucketing and Collisions

- Using an array of *buckets*
  - Each bucket stores the mappings for keys that have the same hash.
  - Each bucket is itself a map from keys to values (implemented by a linked list).
  - The buckets can't use hashing to index the values – instead they use key equality (via the key's equals method)

- To lookup a key in the Hash Map:
  - First, find the right bucket by indexing the array through the key's hash
  - Second, search through the bucket to find the value associated with the key

- Not the only solution to the collision problem

Bucketing



Here, “Jane Smith”.hashCode() and “Joe Schmoe”.hashCode() happen to collide. The bucket at the corresponding index of the Hash Map array stores the map data.

Hash Map Performance

- Hashtables are efficient implementations of Maps and Sets
  - There are many different strategies for dealing with hash collisions with various time/space tradeoffs
  - Real implementations also dynamically resize of the array (which might require re-computing the bucket contents)

- If the hashCode function gives a good (close to uniform) distribution of hashes the buckets are expected to be small (only one or two elements)

Whenever you override `equals` you must also override `hashCode` in a consistent way:

- whenever `o1.equals(o2)== true` you must ensure that `o1.hashCode() == o2.hashCode()`
- note: the converse does not have to hold:

Why? Because comparing hashes is supposed to be a quick approximation for equality.

Example for Point

```
public class Point {
   @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + x;
        result = prime * result + y;
        return result;
    }
}
```

- Examples:
  - (new Point(1,2)).hashCode()   yields 994
  - (new Point(2,1)).hashCode()   yields 1024
- Note that *equal* points have the same hashCode
- Why 31? Prime chosen to create more uniform distribution
- Note: eclipse can generate this code

Computing Hashes

- What is a good recipe for computing hash values for your own classes?
  - intuition: “smear” the data throughout all the bits of the resulting integer

1. Start with some constant, arbitrary, non-zero int in `result.`
2. For each significant field f of the class (i.e. each field taken into account when computing equals), compute a “sub” hash code `c` for the field:
   - For boolean fields: `(f ? 1 : 0)`
   - For byte, char, int, short: `(int) f`
   - For long: `(int) (f ^ (f >>> 32))`
   - For references: 0 if the reference is null, otherwise use the `hashCode()` of the field.
3. Accumulate those subhashes into the result by doing (for each field’s `c`):
   `result = prime * result + c;`
4. return `result`