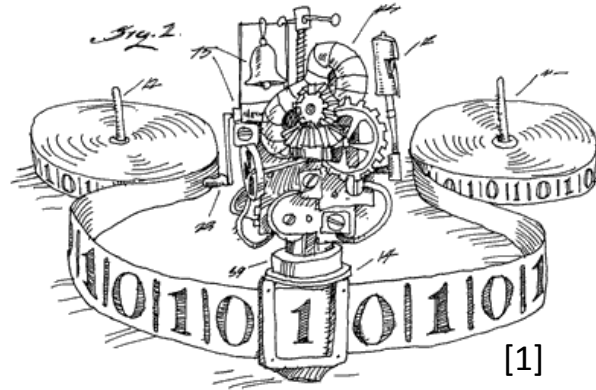


Function objects and comparators



Charles McAnany



Matthew Mercer



Justin Stone



Functions as objects

```
// Let f represent an arbitrary function of one  
// argument.
```

```
A = [x, y, z]
```

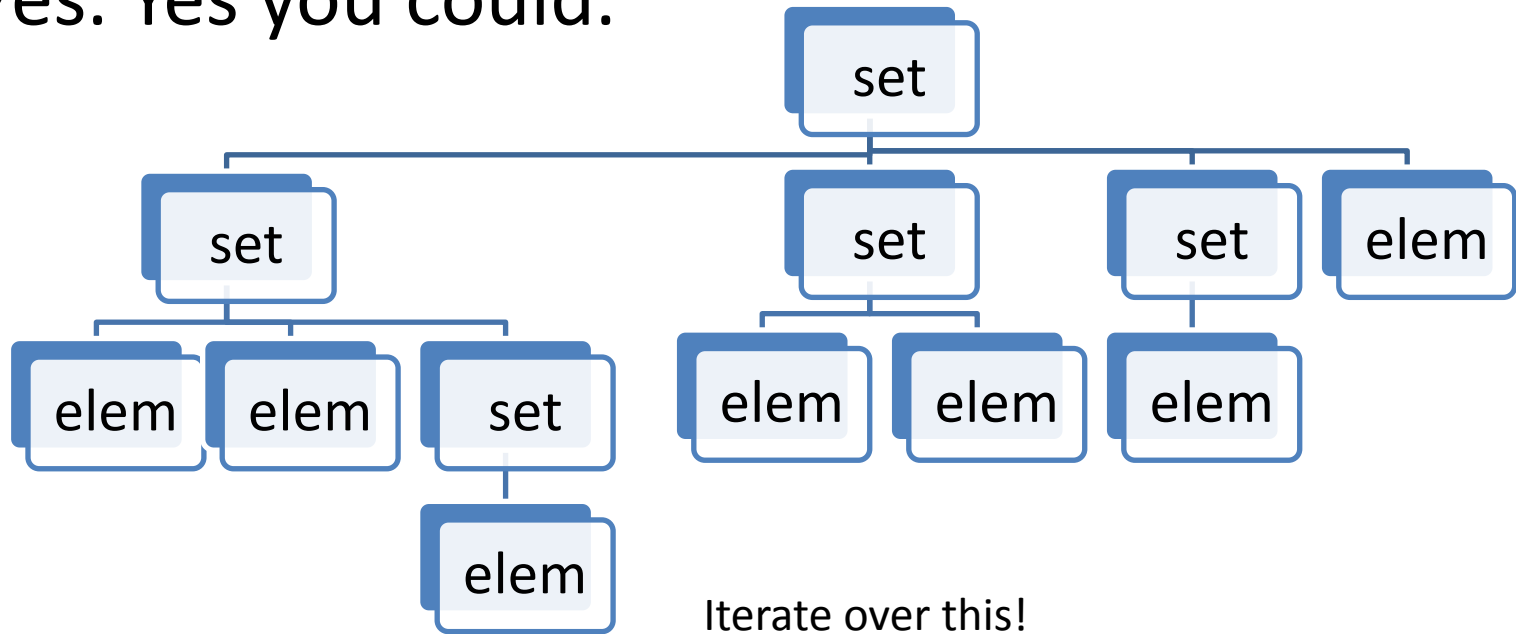
```
    [x, y, z]
```

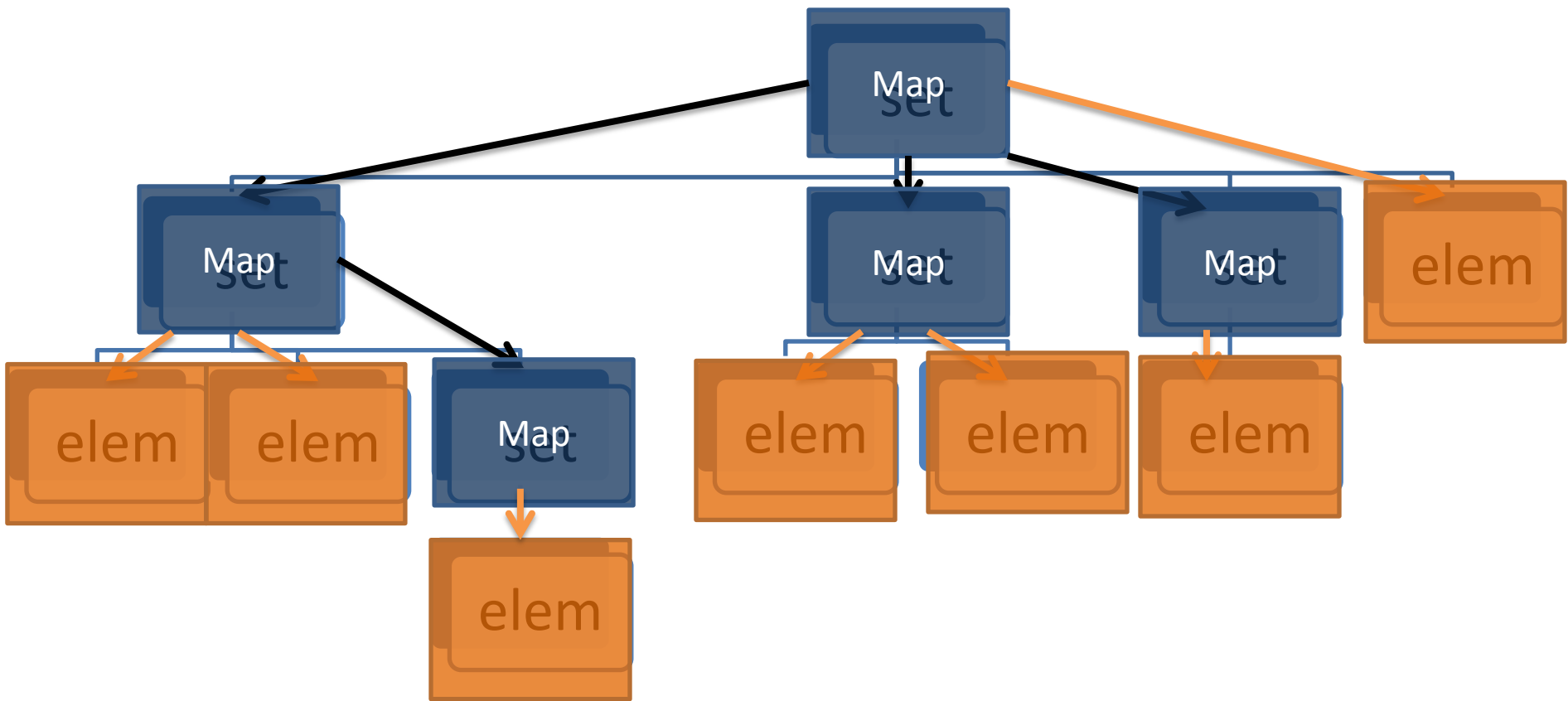
```
map(f, A)
```

```
[f(x), f(y), f(z)] ← This is awesome.
```


Hey, I could just use an iterator!

- Yes. Yes you could.





```
int [] A = {1,2,3};  
map(Math.sin,A);
```

 sin cannot be resolved or is not a field
Press 'F2' for focus

// :: (



Demo


Password (required)

Birthday (required)

March 1981

Human test (required)

Type in the text you see in the box below.



Sorry, your text and the image didn't match. Please try again.

Read (really!) I have read and agree to the [Terms of Use](#) and [Privacy Policy](#).

[2]

Comparators

```
public class Student {  
    public double  
        height,  
        GPA,  
        weight,  
        classRank;  
  
    public Student(double h, double g, double w, double c) {  
        height = h; GPA = g; weight = w; classRank = c;  
    }  
}
```

} Can we sort Student[]?

compareTo(Student other)
would be meaningless.

sortWithComparators(
Student[], heightComparator);

We could write a sortInTermsOfHeight(Student[])
But that's a lot of coding, especially if we're to use
an efficient algorithm.

Enter comparators! A comparator is a function* that takes
two arguments, and compares them! For example, a comp-
arator for height would be

```
double heightComparator(Student s1, Student s2){  
    return s1.height-s2.height;  
}
```

But that requires passing a function as an argument.

[1]<http://www.ecs.syr.edu/faculty/fawcett/handouts/webpages/pictures/turingMachine.gif>

[2] <http://www.seosmarty.com/impossible-captcha-it-doesnt-really-matter-if-you-are-human-or-not/>

Basic Comparator Code

- Arrays class provides static sort method for integers, floating-numbers, and objects

```
int[] a = "...";  
Arrays.sort(a);
```

- Objects need to belong to Comparable

```
public class Coin implements Comparable {  
    public int compareTo(Object otherObject) {  
        Coin other = (Coin) otherObject;  
        if (value < other.value) return -1;  
        if (value == other.value) return 0;  
        return 1;  
    }  
}
```

Basic Comparator Code

- Once your class implements Comparable, you can pass an array of the objects to the `Arrays.sort()` method
 - `Coin[] coins = new Coin[n];`
 - `Arrays.sort(coins);`
- Use sort from Collections class for array lists
 - `ArrayList<Coin> coins = new ArrayList<Coin>;`
 - `Collections.sort(coins);`

Rules of Comparators

- Total ordering relationship
 - Antisymmetric:
 - If $a.compareTo(b) \leq 0$, then $b.compareTo(a) \geq 0$
 - Reflexive:
 - $a.compareTo(a) = 0$
 - Transitive:
 - If $a.compareTo(b) \leq 0$ and $b.compareTo(c) \leq 0$, then $a.compareTo(c) \leq 0$

Common Errors

- If `(a.compareTo(b) == -1)` // Wrong
- If `(a.compareTo(b) < 0)` // Right

Parameterized Comparable Interface

- Comparable interface is now a parameterized type

```
public class Coin implements Comparable<Coin> {  
    ... /* earlier compareTo() code */  
}
```

- No casting object parameters

Demo