
Recitation 11



Lambdas added to Java 8

Customizing Comparison

`new TreeSet<E>()`

- uses `compareTo` built into the elements

But what if you want to use a different order?

- reverse order
- case insensitive

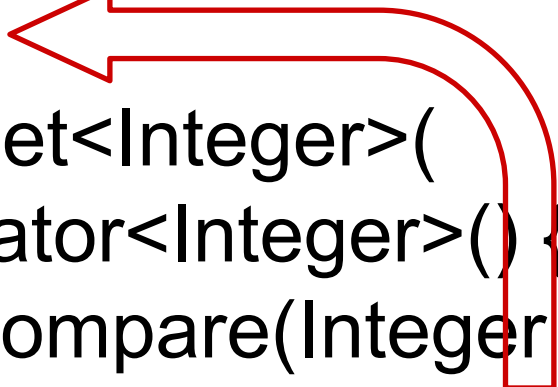
TreeSet's constructor can take a *Comparator*:

- `new TreeSet<K>(Collections.reverseOrder())`
-

Anonymous Inner Class

Goal: sort non-negative integers modulo n

```
int n = ...; // > 0
... = new TreeSet<Integer>(
    new Comparator<Integer>() {
        public int compare(Integer x, Integer y) {
            return x % n - y % n;
        }
    }
);
```



Can access variables that are assigned to exactly once

This is clunky!
Old Java

Lambdas

Only one abstract method to implement,
so we can use a lambda!

Can still access
variables that
are assigned to
exactly once

```
int n = ...; // > 0
```

```
... = new TreeSet<Integer>(
    (Integer x, Integer y) -> x % n - y % n
);
```

parameters

arrow expression

Java takes care of turning this *lambda* into a *Comparator*

Try it Out

```
/** Print out the lower-cased versions of the
 * non-empty strings in strs in order of length. */
public void practice(List<String> strs) {
    // no loops!
    strs.removeIf(                                     );
    strs.replaceAll(                                   );
    strs.sort(                                         );
    strs.forEach(                                       );
}
```

Answer

```
/** Print out the lower-cased versions of the
 * non-empty strings in strs in order of length. */
public void practice(List<String> strs) {
    // no loops!
    strs.removeIf(s -> s.isEmpty());
    strs.replaceAll(s -> s.toLowerCase());
    strs.sort((s1, s2) -> s1.length() - s2.length());
    strs.forEach(s -> System.out.println(s));
}
```

More Complex Lambdas

```
/** Maps [a, b, c] to [a, a, b, b, c, c] */  
public <T> List<T> doubleList(List<T> list) {  
    List<T> d = new ArrayList<T>();  
    list.forEach(t -> {  
        d.add(t);  
        d.add(t);  
    });  
    return d;  
}
```

Diagram illustrating the lambda expression and its block:

- The lambda expression is `t -> {`.
- The block contains two statements: `d.add(t);` and `d.add(t);`.
- The closing brace of the lambda expression is `}`.
- The closing brace of the method is `}`.
- Red annotations highlight the lambda expression and the block:

 - A vertical red line and a bracket labeled "braces" point to the lambda expression `t -> {` and its closing brace `}`.
 - A red bracket labeled "block" encompasses the two `d.add(t);` statements.

More Complex Lambdas

```
List<List<Integer>> lists = ...; // no nulls
// sort so that [1, 3] is before [2, 4, 5]
lists.sort((List<Integer> left, List<Integer> right) -> {
    if (left.size() > right.size())
        return left.size() - right.size();
    for (int i = 0; i < left.size(); i++)
        if (left.get(i) > right.get(i))
            return left.get(i) - right.get(i);
    return left.size() - right.size();
});
```

The lambda's
block can
return values!

Try it Out

```
/** Remove any non-increasing elements. */  
public void filter(List<List<Integer>> lists) {
```

```
}
```

Answer

```
/** Remove any non-increasing elements. */
public void filter(List<List<Integer>> lists) {
    lists.removeIf((List<Integer> list) -> {
        int prev = Integer.MIN_VALUE;
        for (int i : list) // Ignoring nulls ☹️
            if (prev > i) return true;
            else prev = i;
        return false;
    });
}
```

Difference: Fields


Anonymous classes can have fields

```
/** Maps [3, 4, 5] to [3, 7, 12] */
```

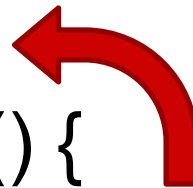
```
public void sumPrev(List<Integer> ints) {  
    ints.replaceAll(new UnaryOperator<Integer>() {  
        int sum = 0;  
        public Integer apply(Integer i) {  
            sum += i;  
            return sum;  
        }  
    });  
}
```

Difference: this

```
class Foo {  
    void bar() {  
        baz(new Anon() {  
            Object func() {  
                return this;  
            }  
        });  
    }  
}
```



```
class Foo {  
    void bar() {  
        baz(() -> this);  
    }  
}
```



Watch out!!!
