CSE 143

Computer Programming II

Iterators, Generics, Inner Classes, Oh My!

Today's Goals

We begin with a class like the `ArrayIntList` we wrote the first week:

MyArraySet

```
1  public class MyArraySet {
2     private int[] elements;
3     ...
4     public boolean remove(int o) {
5        ...
6        if (i == o) {
7        ...
8        if (i < o) {
9        ...
10 }
```

Our goals are:

- To create a full generic version of `ArrayIntList`
- To make `MyArraySet` fully compatible with Java's `Sets`.
- To make `MyArraySet` use the power of Java's `Collections` library.

We begin by making `MyArraySet` generic; so, the following are valid:

- `MyArraySet<Integer> set = new MyArraySet<Integer>();`
- `MyArraySet<String> set = new MyArraySet<String>();`
- `MyArraySet<MyArraySet<String>> set = new MyArraySet<MyArraySet<String>>();`

Adding generics, we make the following changes:

MyArraySet

```
1  public class MyArraySet<E> {
2     private E[] elements;
3     ...
4     public boolean remove(Object o) {
5        ...
6        if (i.equals(o)) {
7        ...
8        if (i.compareTo(o) < 0) {
9        ...
10 }
```

Summary of Changes

- We add a type parameter, `E`, to the class.
- We replace `int` with `E` everywhere.
- Instead of `i == o`, we use `i.equals(o)`.
- Instead of `i < o`, we use `i.compareTo(o) < 0`.

One gotcha, is that we need the type parameter, `E`, to be `Comparable`:

MyArraySet

```
1  public class MyArraySet<E extends Comparable<<E>> {
2     private E[] elements;
3     ...
4     public boolean remove(Object o) {
5        ...
6        if (i.equals(o)) {
7        ...
8        if (i.compareTo(o) < 0) {
9        ...
10 }
```

If we don't make this change, Java won't compile our class.

Implementing Collection & Set

Next, we tell Java that MyArraySet implements Set and Collection:

MyArraySet

```
1  public class MyArraySet<E extends Comparable<<E>>
2                                  implements Collection<E>, Set<E> {
3     private E[] elements;
4     ...
5     public boolean remove(Object o) {
6        ...
7        if (i.equals(o)) {
8        ...
9        if (i.compareTo(o) < 0) {
10       ...
11 }
```

Why Bother Doing This?

- Now, we can say `Set<Integer> s = new MyArraySet<E>();`
- Now, we can say `Collections.sort(s)`.

Next, we tell Java that MyArraySet implements Iterable:

MyArraySet

```
1  public class MyArraySet<E extends Comparable<<E>>
2                       implements Collection<E>, Set<E>, Iterable<E> {
3     private E[] elements;
4     ...
5     public boolean remove(Object o) {
6        ...
7        if (i.equals(o)) {
8        ...
9        if (i.compareTo(o) < 0) {
10       ...
11 }
```

Why Bother Doing This?

- Now, we can use foreach loops with our class!!

```
1 Set<Integer> s = new MyArraySet<Integer>();
2 s.add(10);
3 s.add(5);
4 for (int i : s) {
5    System.out.println(i);
6 }
```

You Can't Remove In A foreach Loop!

```
1 Set<String> set = new TreeSet<String>();
2 set.add("hello");
3 set.add("world");
4 for (String s : set) {
5    if (s.startsWith("h")) {
6       set.remove(s);
7    }
8 }
```

OUTPUT

```
>> Exception in thread "main" java.util.ConcurrentModificationException
>>     at java.util.TreeMap$PrivateEntryIterator.nextEntry(TreeMap.java:1115)
>>     at java.util.TreeMap$KeyIterator.next(TreeMap.java:1169)
>>     at Client.main(Client.java:12)
```

ConcurrentModificationException

A ConcurrentModificationException happens when you try to edit a structure that you are looping through in a foreach loop. **You should not try to remove inside a foreach loop! It will fail!**

So, how do we remove from a Set?

The solution to this problem is called an `Iterator`. The interface is:

```
1 public interface Iterator<E> {
2    public boolean hasNext();
3    public E next();
4    public void remove();
5 }
```

Implementing `Iterable` requires adding a method called `iterator`:

```
1 public Iterator<E> iterator() {
2    return new OurIterator();
3 }
```

And we must implement `OurIterator`:

```
1 public class OurIterator {
2    public boolean hasNext() { ... }
3    public E next() { ... }
4    public void remove() { ... }
5 }
```

To **hide** a class from the client, we can use an idea called an **inner class**:

MyArraySet

```
1  public class MyArraySet<E extends Comparable<<E>>
2                       implements Collection<E>, Set<E>, Iterable<E> {
3     private E[] elements;
4     ...
5     public boolean remove(Object o) {
6        ...
7        if (i.equals(o)) {
8        if (i.compareTo(o) < 0) {
9        ...
10    }
11    private class MyArraySetIterator {
12       public boolean hasNext() {...}
13       public E next() {...}
14       public void remove() {...}
15    }
16
17    public Iterator<E> iterator() {
18       return new MyArraySetIterator();
19    }
20 }
```

You Can't Remove In A `foreach` Loop!

```
1 Set<String> set = new TreeSet<String>();
2 set.add("hello");
3 set.add("world");
4 for (String s : set) {
5    if (s.startsWith("h")) {
6       set.remove(s);
7    }
8 }
```

The `iterator` Fix

```
1 Set<String> set = new TreeSet<String>();
2 set.add("hello");
3 set.add("world");
4 Iterator<String> it = set.iterator();
5 while (it.hasNext()) {
6    if (it.next().startsWith("h")) {
7       it.remove(s);
8    }
9 }
```

Note that we call `it.remove`, not `set.remove`!