Adam Blank | Lecture 22 | Winter 2015

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
```
public class MyArraySet {
   private int[] elements;
   ...
   public boolean remove(int o) {
      ...
      if (i == o) {
      ...
      if (i < o) {
      ...
}
```

Our goals are:

- To create a full generic version of `ArrayIntList`
- To make `MyArraySet` fully compatible with Java's `Set`s.
- 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>>();`

Generic-ifying

Adding generics, we make the following changes:

MyArraySet
```
public class MyArraySet<E> {
   private E[] elements;
   ...
   public boolean remove(Object o) {
      ...
      if (i.equals(o)) {
      ...
      if (i.compareTo(o) < 0) {
      ...
}
```

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`.

Generic-ifying

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

MyArraySet
```
public class MyArraySet<E extends Comparable<<E>> {
   private E[] elements;
   ...
   public boolean remove(Object o) {
      ...
      if (i.equals(o)) {
      ...
      if (i.compareTo(o) < 0) {
      ...
}
```

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
```
public class MyArraySet<E extends Comparable<<E>>
                                           implements Collection<E>, Set<E> {
   private E[] elements;
   ...
   public boolean remove(Object o) {
      ...
      if (i.equals(o)) {
      ...
      if (i.compareTo(o) < 0) {
      ...
}
```

Why Bother Doing This?

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

Using `foreach` Loops with OUR classes

Next, we tell Java that `MyArraySet` implements `Iterable`:

MyArraySet

```java
public class MyArraySet<E extends Comparable<<E>>
                       implements Collection<E>, Set<E>, Iterable<E> {
   private E[] elements;
   ...
   public boolean remove(Object o) {
      ...
      if (i.equals(o)) {
      ...
      if (i.compareTo(o) < 0) {
      ...
}
```

Why Bother Doing This?

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

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

`foreach` Loop Warning!

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

```java
Set<String> set = new TreeSet<String>();
set.add("hello");
set.add("world");
for (String s : set) {
   if (s.startsWith("h")) {
      set.remove(s);
   }
}
```

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`?

Iterators

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

```java
public interface Iterator<E> {
   public boolean hasNext();
   public E next();
   public void remove();
}
```

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

```java
public Iterator<E> iterator() {
   return new OurIterator();
}
```

And we must implement `OurIterator`:

```java
public class OurIterator {
   public boolean hasNext() { ... }
   public E next() { ... }
   public void remove() { ... }
}
```

Inner Classes

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

MyArraySet

```java
public class MyArraySet<E extends Comparable<<E>>
                       implements Collection<E>, Set<E>, Iterable<E> {
   private E[] elements;
   ...
   public boolean remove(Object o) {
      ...
      if (i.equals(o)) {
      if (i.compareTo(o) < 0) {
      ...
   }
   private class MyArraySetIterator {
      public boolean hasNext() {...}
      public E next() {...}
      public void remove() {...}
   }

   public Iterator<E> iterator() {
      return new MyArraySetIterator();
   }
}
```

Using An Iterator

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

```java
Set<String> set = new TreeSet<String>();
set.add("hello");
set.add("world");
for (String s : set) {
   if (s.startsWith("h")) {
      set.remove(s);
   }
}
```

The `iterator` Fix

```java
Set<String> set = new TreeSet<String>();
set.add("hello");
set.add("world");
Iterator<String> it = set.iterator();
while (it.hasNext()) {
   if (it.next().startsWith("h")) {
      it.remove(s);
   }
}
```

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