Java Iterators

Motivation

- We often want to access every item in a collection of items
  - We call this ***traversing*** or ***iterating over every item***
- Example: array

```
for (int i = 0; i < array.length(); i++)
    /*  do something to array[i]   */
```

  - This is straighforward because we know exactly how an array works!

9-2

Motivation

- What if we want to traverse a *collection* of objects?
  - Its underlying implementation may not be known to us
- Java provides an *interface* for stepping through all elements in *any* collection, called an *iterator*

9-3

Reminder: Iterating through ArrayList

- Iterating through an ArrayList of Strings:

```
for (int i = 0; i < list.size(); i++) {
   String s = list.get(i);
   //do something with s
}
```

- Alternative:

```
while (list.hasNext()) {
   String s = list.next();
}
```

This syntax of iteration is generic and applies to any Java iterable.

Iterators

- An ***iterator*** is a mechanism used to step through the elements of a collection one by one
  - Each element is "***delivered***" exactly once
- ***Example***
  - Iterate through an ordered list and print each element in turn

The Java **Iterator** Interface

- The Java API has a generic interface called Iterator<T> that specifies what methods are required of an iterator
  - `public boolean hasNext();`
    returns true if there are more elements to iterate over
  - `public T next();`
    returns the next element
  - `public void remove();`
    removes the last element returned by the iterator (*optional operation*)
- It is in the java.util package of the Java API

9-6

Using an iterator

```
ArrayIterator<Integer> itr = new
                ArrayIterator<Integer>(array);
while (itr.hasNext()){
    Integer element = itr.next();
}
```

Example: an array iterator

```
public class ArrayIterator<T> implements Iterator<T>{
   private int current;
   private T[] array;
   public ArrayIterator (T [] array){
      this.array = array;
      this.current = 0;
   }
   public boolean hasNext(){
      return (current < array.length);
   }
   public T next(){
      if (!hasNext())
         throw new NoSuchElementException();
      current++;
      return array[current - 1];
   }
}
```

The Iterable interface

Instead of:

```
while (list.hasNext()) {
   String s = list.next();
}
```

We can do:

```
for (String s : list) {
   //do something with s
}
```

That's because a list is iterable

The Iterable interface

- The Java API has a generic interface called Iterable<T> that allows an object to be the target of a "foreach" statement
  - `public Iterator<T> iterator();` returns an iterator
- Why do we need Iterable?
  - An Iterator can only be used once, Iterables can be the subject of "foreach" multiple times.

Why use Iterators?

- Traversing through the elements of a collection is very common in programming, and iterators provide a *uniform* way of doing so.
- Advantage? Using an iterator, we don't need to know how the data structure is implemented!

9-11