CSE 142

Iterators

Outline for Today

- Quick Review
  - ArrayList collections; add, size, get, etc. methods
  - Iteration and while loops
- Today
  - Iterating through collections
  - Iterator objects

Using Collections

- We can create ArrayLists, and put things into them.

```
ArrayList names = new ArrayList( );
names.add("Bob");
names.add("Sue");
names.add("Jeremiah");
```

- We can pick out elements at particular index positions.

```
String someNames = names.get(0) + " and " + names.get(1);
```

- But how can we do something for all names?
  - Print out all names in the list.
  - Find the first name, alphabetically.
  - Find what the longest name.
  - See if a given name is in the list.

Iterating Through Collections

- What we really want is to be able to write:

```
For all elements in the list,
    Do something.
```

- This will be a loop, since we want to repeat the "do something" for each element in the list.
- For some collections, like lists, we could access the elements by position (e.g., get(i))
- More general – use an “iterator” object associated with the collection
  - Generalizes to other collections like set that are not ordered

Iterators

- To get "all elements in the list", we can use an ***iterator*** object
- Iterators
  - Know about the collection they are processing
  - Keep track of where they are in the collection
- Strategy
  - Ask the array list for an associated iterator object
  - Ask the iterator object for each element, in turn, in a while loop
- We don't have to count or even know how many elements are in the list!

Iterator Operations

- Getting an iterator object from an ArrayList (and other kinds of Java collections):

```
Iterator iter = names.iterator( );
```

- Here are the basic methods provided an Iterator:

```
// Return true if the iteration has more elements.
public boolean hasNext( )

// Return the next element in the iteration.
public Object next( )
```

- Note that we'll usually want (need) to cast the result of next() to a more specific kind of object

Using an Iterator, in English

- General algorithm:

```
Get the iterator for the collection [names.iterator()]
While the iterator has at least one more element [iter.hasNext()]
    Get the next element  [iter.next()]
    Do something using the element
    Then go back to the top
Otherwise, we're done
```


Using an Iterator, in Java

```
ArrayList names = ...;

System.out.println("The names are as follows:");

Iterator iter = names.iterator( );
while ( iter.hasNext( ) ) {
    String name = (String) iter.next( );
    System.out.println(name);
}

// done
```

•Footnote: While an iteration is in progress, you can't add or delete items from the list (with some exceptions, which you can look up in the docs)

Relationships Between Objects

Another Example: Finding the Longest Name

- Suppose we want to find the longest name. How would we do it?
  - **Recall:** "Bob".length() == 3
- What's the algorithm in English?
- What's the Java code?

Solution

Iterators vs Indexed Access

- We can also process an ArrayList and some other kinds of collections using get(index)

```
for (int k = 0; k < names.size( ); k++) {
    process names.get(k);
}
```

- Tradeoffs
  - Iterators are more general – work on all collections, even if the collection doesn't support indexed access (i.e., using get(k) to access elements directly), however
  - Iterators only support traversal of a collection from beginning to end – if we want to go backwards or in some other order, we need indexed access (and a container that supports it)
- General rule: use iterators (the more general solution) normally; use other traversals when iterators don't do what you need