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.
- To get "all elements in the list", we can use an ***iterator*** object.
  - Ask the array list for its iterator object.
  - Ask the iterator object for each element, in turn, as part of a while loop.
- We don't have to know how many elements are in the list!

Iterator Operations

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

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

- Here are the methods provided by Iterator:

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

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


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( );          // get the iterator for the collection.

while ( iter.hasNext( ) ) {                 // while there is another element…
    String name = (String) iter.next( );    // get the element (and cast it if needed)
    System.out.println(name);               // do something using the element.
}                                           // then go back to the top.

// Otherwise, we're done.
```


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 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)
  - 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