Lecture 9

Iterators

Iterators are a convenient way of moving through data in a Collection. They move through the data in a predictable order.

Iterator is an interface, it requires

- hasNext() – returns true if there are more elements.
- next() – returns the next item in the collection.
- remove() – Removes from the underlying collection the last element returned by the iterator.

Note that `remove()` is an optional method, and thus does not have to be implemented.

`remove()` must be called after `next()` is called, and can only be called once. Behavior of the iterator is undefined for multiple calls.

Also, we should never change the Collection in the middle of an iteration. This behavior is also undefined.

Examples:

- `arraylistIterator()`:
  - `ArrayList` implements the `Iterator` interface, so it provides an `iterator()` method.
  - Our types have to match, so if we have an `ArrayList` of type `ArrayList<Integer>` then our `Iterator` is `Iterator<Integer>`
  - Once we have the `Iterator`, we call `hasNext()` until it is false.
  - If `hasNext()` returns true, we call `next()` to get the next object in the iteration.
- `arraylistIteratorConcern()`:
  - Notice that the `iterator()` method in ArrayList returns a new instance of an `Iterator` with every call.
  - Each iterator keeps track of where it is in the iteration
- `arraylistIteratorConcern2()`, we see we can't modify in the middle of an iteration
  - Throws `ConcurrentModificationException` if we add to the list, then call `next()`
  - works fine if we don't call `next()` again (because we are not in the middle of an iteration). iteration, we get a ConcurrentModificationException
- `arraylistConcern3()`:
  - We can change a field inside an object on the list, since the object (from the point of view of the list) hasn't changed.

`AlphabetSoup` and `SoupIterator` show a full implementation of our own custom iterator.

`AlphabetSoup`

- implements `Iterable<String>`
- implements the `iterator()` method, which returns a new instance of `SoupIterator`
- return type of `iterator()` is `Iterator<String>`

SoupIterator class

- implements `Iterator<String>`
- constructor takes the type that it will iterate over
- maintains a `currentIndex` instance variable to save the index of the next item to return
- `hasNext()` checks `currentIndex` against the size of the list
- `next()` returns the item at `currentIndex`
- `next()` also calls `hasNext()`, just to be safe.

`alphabetIterable()` in IteratorExamples uses the Java's iterator `for` loop. We can't do this unless the type specified in the `for` loop implements `Iterable`

File I/O

You will be doing an on-disk sorting program for Assignment 3, so you will need to be creating, moving, merging, and deleting files.

Here's a few helpful hints.

File class:

- represents a file or directory
- doesn't have to exist
- use the File.separator so that it doesn't matter what system we run on.
- Some methods that may be helpful:
  - delete()
  - exists()
  - createNewFile()
  - isFile()
  - isDirectory()
  - listFiles()
  - mkdir()
  - renameTo(...)

Use the `BufferedReader` and `PrintWriter` classes for reading and writing to file, respectively. They have lots of useful methods to make things easy on you.

```
PrintWriter out = new PrintWriter(new FileWriter(...));
BufferedReader in = new BufferedReader(new FileReader(...));
```