More on Implementing Collections III

Lecture 9

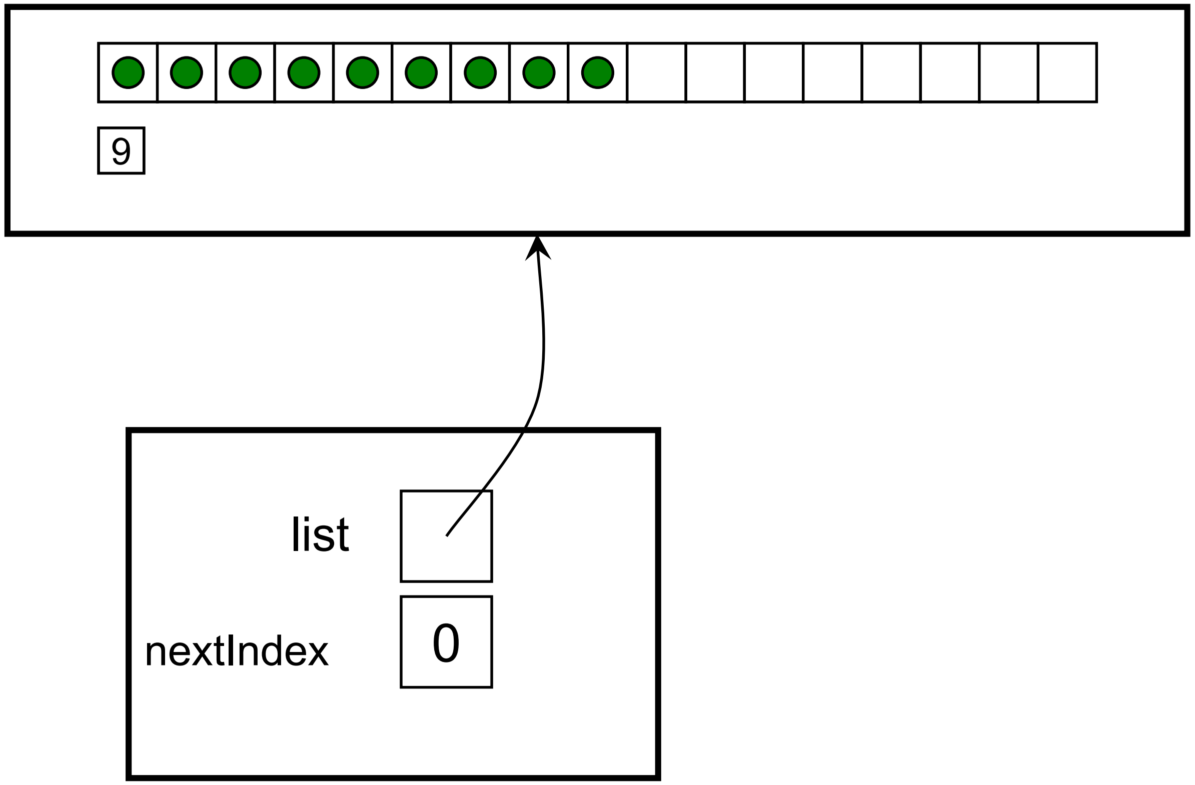
Menu

- Implementing ArrayList:
    - Iterators
    - Cost of adding and removing

ArrayList: What else?

- iterator():
  - defining an iterator for ArrayList.
- Cost:
  - What is the cost (time) of adding or removing an item?
  - How expensive is it to increase the size?
  - How do we increase the size?

Iterator



ArrayList: iterator

```
/** Returns an iterator over the elements in the List */
public Iterator <E> iterator(){
     return new ArrayListIterator<E>(this);
}

/** Definition of the iterator for an ArrayList
   * Defined inside the ArrayList class, and can therefore  access
     *   the private fields of an ArrayList object.  */
    private class ArrayListIterator <E> implements Iterator <E>{
      //  fields to store state
      //  constructor
      //  hasNext(),
      //  next(),
      //  remove()     (an optional operation for Iterators)
   }
```

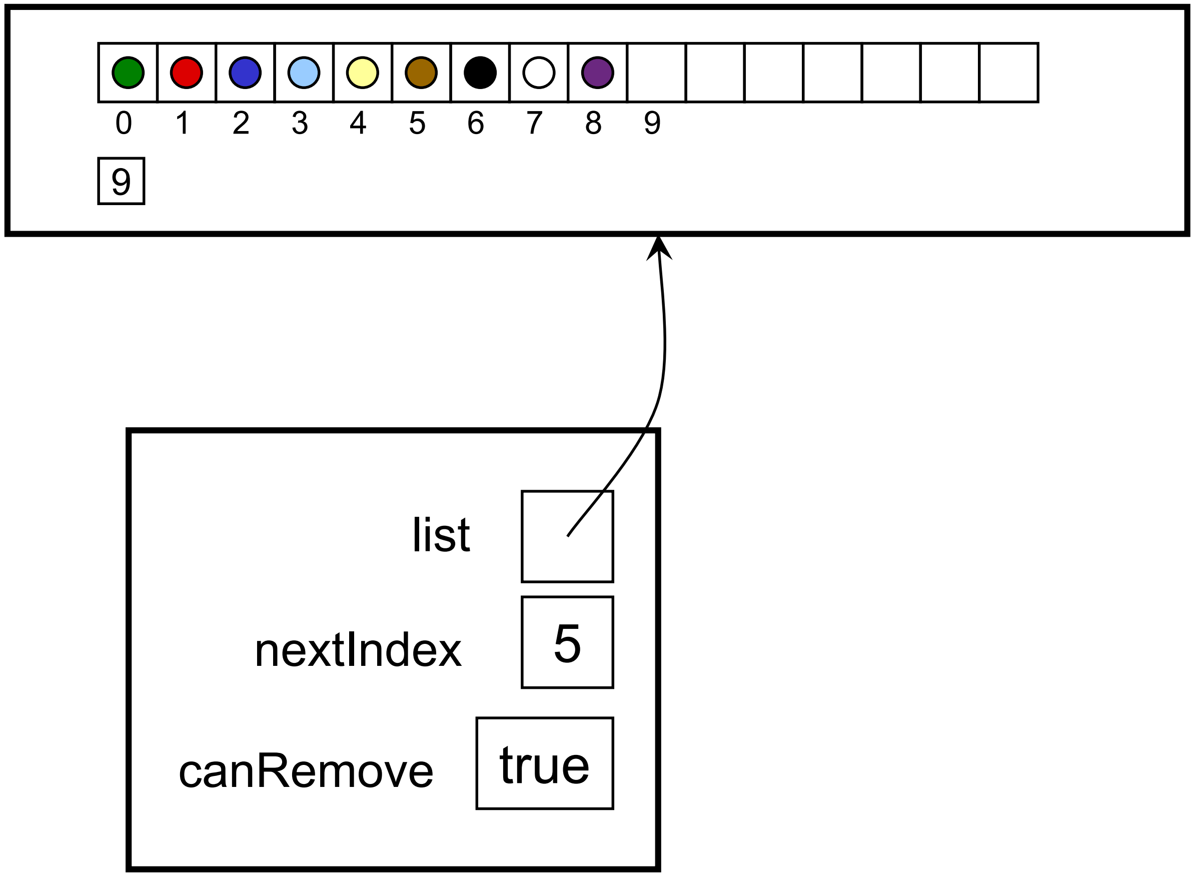
Iterator

```
private class ArrayListIterator <E> implements Iterator <E>{

    private ArrayList<E> list;// reference to the list it is iterating down
    private int nextIndex = 0;  // the index of the next value to return
    private boolean canRemove = false;
                    // to disallow the remove operation initially

    /** Constructor  */
    private ArrayListIterator (ArrayList <E> list) {
      this.list = list;
     }

    /** Return true if iterator has at least one more element  */
    public boolean hasNext () {
      return (nextIndex < list.count);
    }
```

Iterator: next, remove

```
/** Return next element in the List */
public E next () {
    if (nextIndex >= list.count)  throw new NoSuchElementException();
            return list.get(nextIndex++); ← increment and return
          }


/** Remove from the list the last element returned by the iterator.
  *  Can only be called once per call to next.  */
public void remove(){
     throw new UnsupportedOperationException();
  }
```

Iterator, with remove



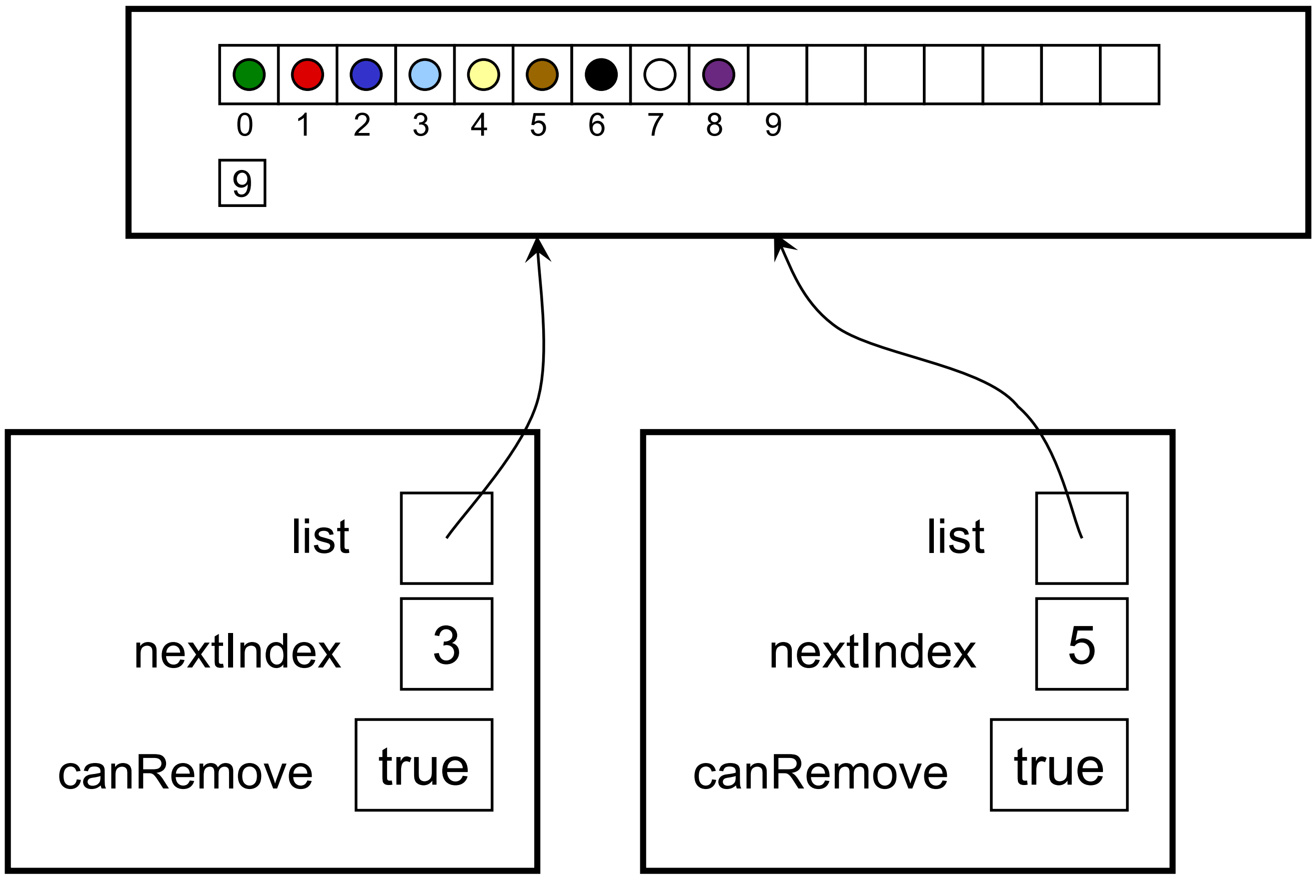
Iterator: next, remove

```
/** Return next element in the List */
  public E next () {
      if (nextIndex >= list.count)  throw new
  NoSuchElementException();
      canRemove = true;               ← for the remove method
      return list.get(nextIndex++);   ← increment and return
    }

/** Remove from the list the last element returned by the iterator.
  Can only be called once per call to next.  */
  public void remove(){
      if ( ! canRemove )   throw new IllegalStateException();
      canRemove = false;              ← can only remove once
      nextIndex--;           ← put counter back to last item
      list.remove(nextIndex);         ← remove last item
    }
```

what if we don't put counter back to last item?

After removal, nextIndex will be pointing at which item?

Multiple Iterators



Multiple Iterators: Summary

- Each iterator keeps track of its own position in the List
- Removing the last item returned is possible, but
- The implementation is not smart, and may be corrupted if any changes are made to the ArrayList that it is iterating down.
- Note that because it is an inner class, it has access to the ArrayList's private fields.

ArrayList: Cost

- What's the cost of get, set, remove, add?
- How should we implement ensureCapacity() ?
- How do you measure the cost of operations on collections?
- What is the "cost" of an algorithm or a program?
- Number of steps required if the list contains *n* items:
  - get:
  - set:
  - remove:
  - add:

Q&A

- remove() is compulsory in Iterator implementation. (T or F)
- How does ArrayList make use of the ‘type parameter’ in its implementation?
- Which element will be removed by ArrayList.remove()?
- What does ArrayList.next() check before returning the next element in the list?
- How does ArrayList.remove() ensure only 1 element can be removed after each call to next()?
- What can happen if 2 or more Iterators running concurrently under the same ArrayList? Name 2 scenarios.

Summary

- Implementing ArrayList:
  - Iterators
  - Costs of adding and removing

Readings

- [Mar07] Read 3.4
- [Mar13] Read 3.4