Implementing Collections II

Lecture 8

Summary

- Implementing Collections:
  - Interfaces, Abstract Classes, Classes

Defining ArrayList

- Design the data structures to store the values
  - array of items
  - count

- Define the fields and constructors

```
public class ArrayList <E> implements List<E> {
    private E [ ] data;
    private int count;
```

- Define all the methods specified in the List interface

size()	add(E o)	add(int index, E element)	contains(Object o)	get(int index)
isEmpty()	clear()	set(int index, E element)	indexOf(Object o)	remove(int index)
remove(Object o)		lastIndexOf(Object o)	iterator()	
equals(Object o)		hashCode()	listIterator()	… … …

Defining ArrayList: too many methods

- Problem: There are a lot of methods in List that need to be defined in ArrayList, and many are complicated.

- But, many could be defined in terms of a few basic methods: (size, add, get, set, remove)
  - eg,

```
public boolean addAll(Collection<E> other){
   for (E item : other)
        add(item);
}
```

- Solution: **an Abstract class**
  - Defines the complex methods in terms of the basic methods
  - Leaves the basic methods “abstract” (no body)
  - classes implementing List can extend the abstract class.

Interfaces and Classes

- Interface
  - specifies type
  - defines method headers
- List <E>
  - Specifies sequence of E *type*
- Abstract Class
  - implements Interface
  - defines some methods
  - leaves other methods "abstract"
- AbstractList <E>
  - implements List <E>
  - defines array of <E>
  - defines addAll, subList, …
  - add, set, get, … are left **abstract**
- Class
  - extends Abstract Class
  - defines data structures
  - defines basic methods
  - defines constructors
- ArrayList <E>
  - extends AbstractList
  - implements fields & constructor
  - implements add, get, …

AbstractList

```
public abstract class AbstractList <E> implements List<E>{

  public abstract int size();

  public boolean isEmpty(){
    return (size() == 0);
  }

  public abstract E get(int index);

  public void add(int index, E element){
    throws new UnsupportedOperationException();
  }

  public boolean add(E element){
    add(size(), element);
```

No constructor or fields

declared abstract - must be defined in a real class

defined in terms of size()

declared abstract - must be defined in a real class

defined to throw exception should be defined in a real class

defined in terms of other add

AbstractList continued

```
public boolean contains(Object ob){
   for (int i = 0; i<size(); i++)
      if (get(i).equals(ob) ) return true;
   return false;
}
```

defined in terms of size and get

```
public void clear(){
   while (size() > 0)
      remove(0);
}
```

defined in terms of size and remove

:

:

- AbstractList cannot be instantiated.

ArrayList extends AbstractList

public class ArrayList <E> **extends** AbstractList<E>{

// fields to hold the data:

need an array for the items and an int for the count.

// constructor(s)

Initialise the array

// definitions of basic methods not defined in AbstractList

size()
get(index)
set(index, value)
remove(index)
add(index, value)
iterator()

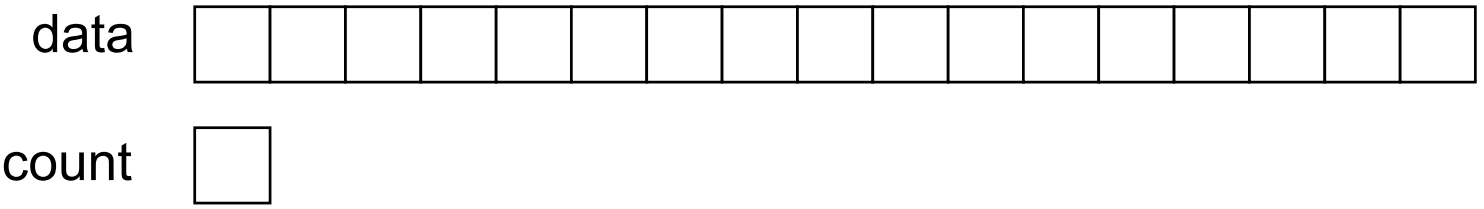
Can give other methods to override the inherited methods, if it would be more efficient

// (other methods are inherited from AbstractList)

// definition of the Iterator class

Implementing ArrayList

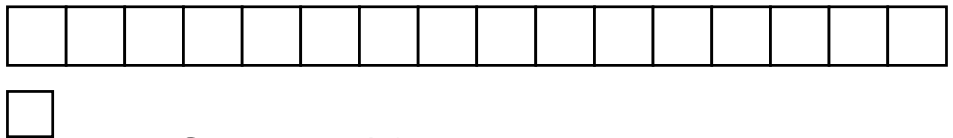
- Data structure:



- size:
  - returns the value of count
- get and set:
  - check if within bounds, and
  - access the appropriate value in the array
- add(index, elem):
  - check if within bounds, (0..size)
  - move other items up, and insert
  - as long as there is room in the array !

ArrayList: fields and constructor

```
public class ArrayList <E> extends AbstractList <E> {
    private E[ ] data;
    private int count=0;
    private static int INITIALCAPACITY = 16;
```

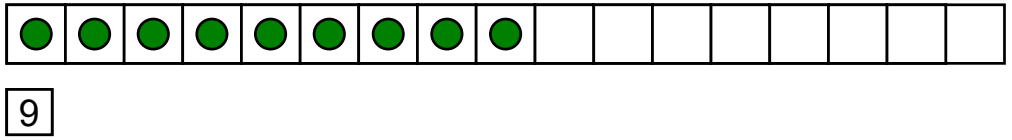


```
public ArrayList(){
    data =      (E[ ]) new Object[INITIALCAPACITY];
}
```

- Can't use type variables as array constructors!!!!
- Must Create as Object[ ] and cast to E[ ]
- The compiler will return a warning!
  - " …. uses unchecked or unsafe operations" (why it is 'unchecked'?)

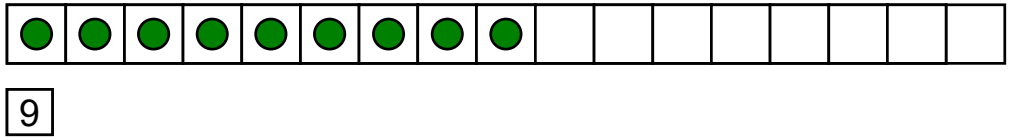
ArrayList: size, isEmpty

```
public class ArrayList <E> extends AbstractList <E> {
    private E[ ] data;
    private int count=0;
        :

/** Returns number of elements in collection as integer */
    public int size () {
        return count;
    }

/** Returns true if this set contains no elements. */
    public boolean isEmpty(){
        return count==0;
    }
```



ArrayList: get

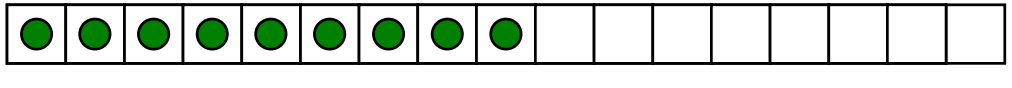
```
public class ArrayList <E> extends AbstractList<E> {
    private E[ ] data;
    private int count=0;
        :
```




```
 /**Returns the value at the specified index.
  * Throws an IndexOutOfBoundsException is index is out of
  bounds */
   public E get(int index){
     if (index < 0 || index >= count)
     throw new IndexOutOfBoundsException();
     return data[index];
   }
```

ArrayList: set

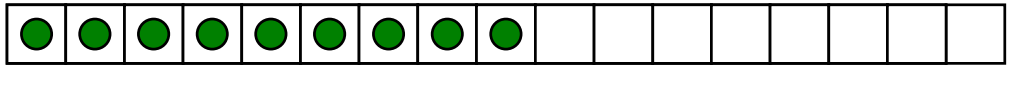
```
public class ArrayList <E> extends AbstractList<E> {
    private E[ ] data;
    private int count=0;
        :
```




```
/**Replaces the value at the specified index by the specified value
 * Returns the old value.
 * Throws an IndexOutOfBoundsException if index is out of
 bounds */
 public E set(int index, E value){
    if (index < 0 || index >= count)
        throw new IndexOutOfBoundsException();
    E ans = data[index];
    data[index] = value;
    return ans;
  }
```

ArrayList: remove

```
public class ArrayList <E> extends AbstractList <E> {
    private E[] data;
    private int count=0;
        :
```

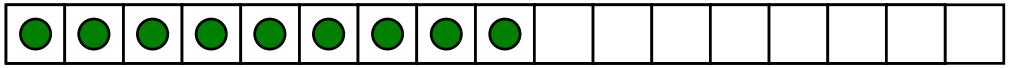



```
 /** Removes the element at the specified index, and returns it.
   * Throws an IndexOutOfBoundsException if index is out of
   bounds */
public E remove (int index){
      if (index < 0 || index >= count)
        throw new IndexOutOfBoundsException();
      E ans = data[index];                    ←remember
      for (int i=index; i< count; i++)        ←move items down
        data[i]=data[i+1];
      count--;                                ←decrement
      return ans;                             ←return
    }
```

problem?

ArrayList: remove (fixed)

```
public class ArrayList <E> extends AbstractList <E> {
   private E[] data;
   private int count=0;
        :
```

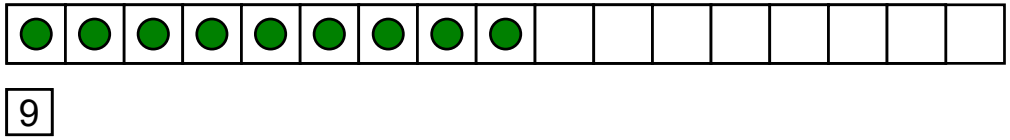


9

```
 /** Removes the element at the specified index, and returns it.
  * Throws an IndexOutOfBoundsException if index is out of
  bounds */ public E remove (int index){
     if (index < 0 || index >= count)
       throw new IndexOutOfBoundsException();
     E ans = data[index];                    ←remember
     for (int i=index+1; i< count; i++)      ←move items down
        data[i-1]=data[i];
     count--;                                ←decrement
     data[count] = null;                     ←delete previous last element
     return ans;                             ←return
   }
```

ArrayList: add

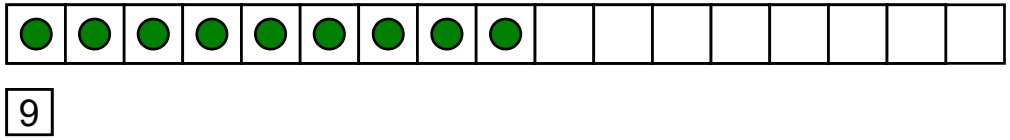
```
public class ArrayList <E> extends AbstractList <E> {
    private E[] data;
    private int count=0;
        :

 /** Adds the specified element at the specified index */
public void add(int index, E item){
    if (index < 0 || index >= count)
        throw new IndexOutOfBoundsException();
    for (int i=count; i > index; i--)        ←move items up
        data[i]=data[i-1];
    data[index]=item;                        ←insert
    count++;                                 ←increment
}
```



- What's wrong???

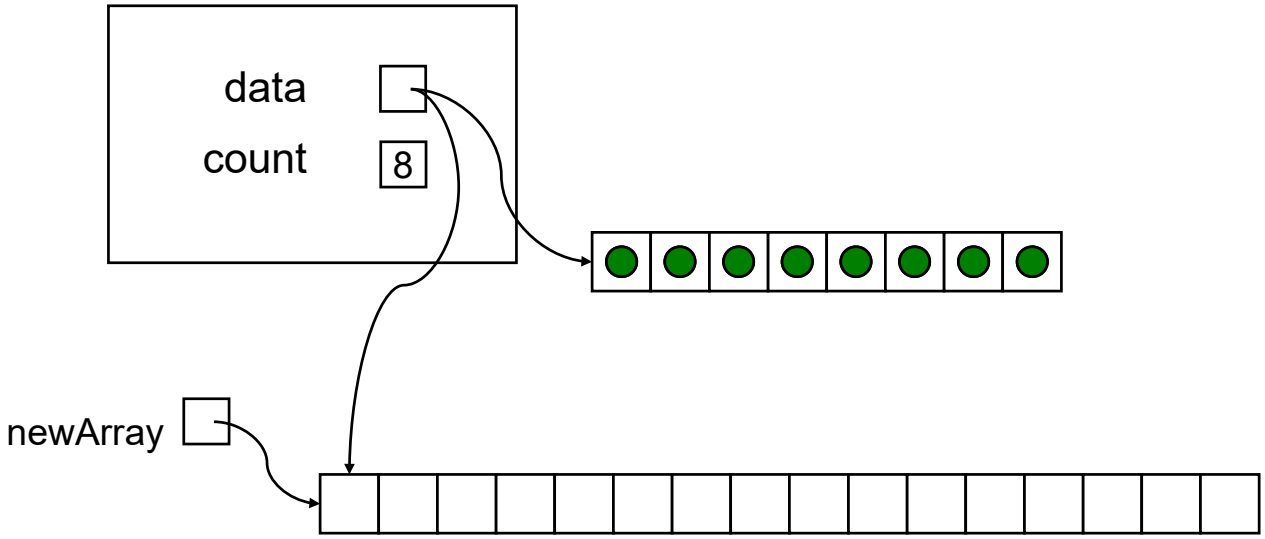
ArrayList: add (fixed)

```
public class ArrayList <E> extends AbstractList <E> {
    private E[] data;
    private int count=0;
        :

 /** Adds the specified element at the specified index.*/
public void add(int index, E item){
     if (index < 0 || index > count)      ←can add at end?
        throw new IndexOutOfBoundsException();
     ensureCapacity();                    ←make room
     for (int i=count; i > index; i--)    ←move items up
       data[i]=data[i-1];
     data[index]=item;                    ←insert
     count++;                             ←increment
}
```



Increasing Capacity

- ensureCapacity():



- How big should the new array be?

ArrayList: ensureCapacity

```
/**Ensure data array has sufficient number of elements
   * to add a new element */
private void ensureCapacity () {
  if (count < data.length) return;          ← room already
  E [ ] newArray = (E[]) (new Object[data.length+INITIALCAPACITY]);
    for (int i = 0; i < count; i++)         ← copy to new array
        newArray[i] = data[i];
    data = newArray;                        ← replace (replace what?)
}
```

OR

```
private void ensureCapacity () {
    if (count < data.length) return;        ← room already
    E [ ] newArray = (E[ ]) (new Object[data.length * 2]);
    for (int i = 0; i < count; i++)         ← copy to new array
        newArray[i] = data[i];
    data = newArray;                        ← replace
}
```

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 should we increase the size?

Q&A

- What are the key features of an abstract class?
- Can an abstract class be instantiated?
- Abstract methods can be defined within a class to save implementation efforts. (T or F)

- What are the key issues of implementation when we remove an element from an ArrayList?
- What are the key issues of implementation when we add an element from an ArrayList?

Menu

- Implementing Collections:
  - Interfaces, Abstract Classes, Classes