CS151

ArrayList

Java: Inner Classes

Maps

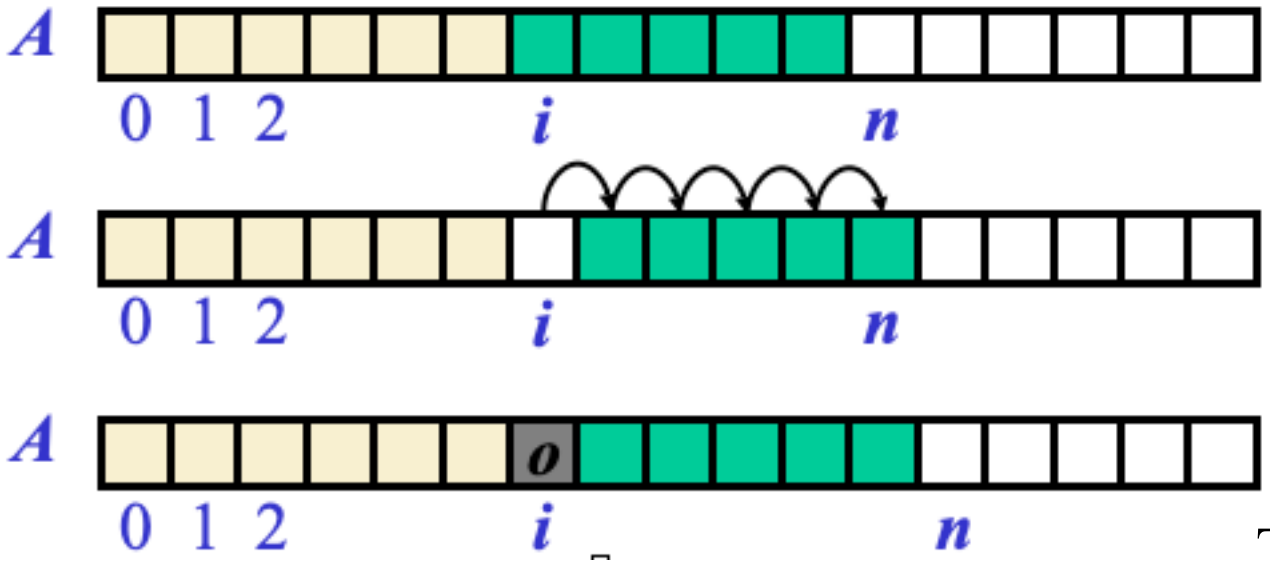
Lists

- A list is a bag in which the items are ordered.
  - No empty list items allowed!
  - Position in list is not fixed, but relative order is
- Actions with lists
  - Add item at location N
  - Get Nth item
  - Change Nth item
  - Remove Nth item
  - Others from BagOfStuff

clear, count, empty?, contains?, display

add(int index, W t)

- Tasks
  - Check location to ensure it is valid
  - Make space for new item



To make a space start at nth item move it to n+1

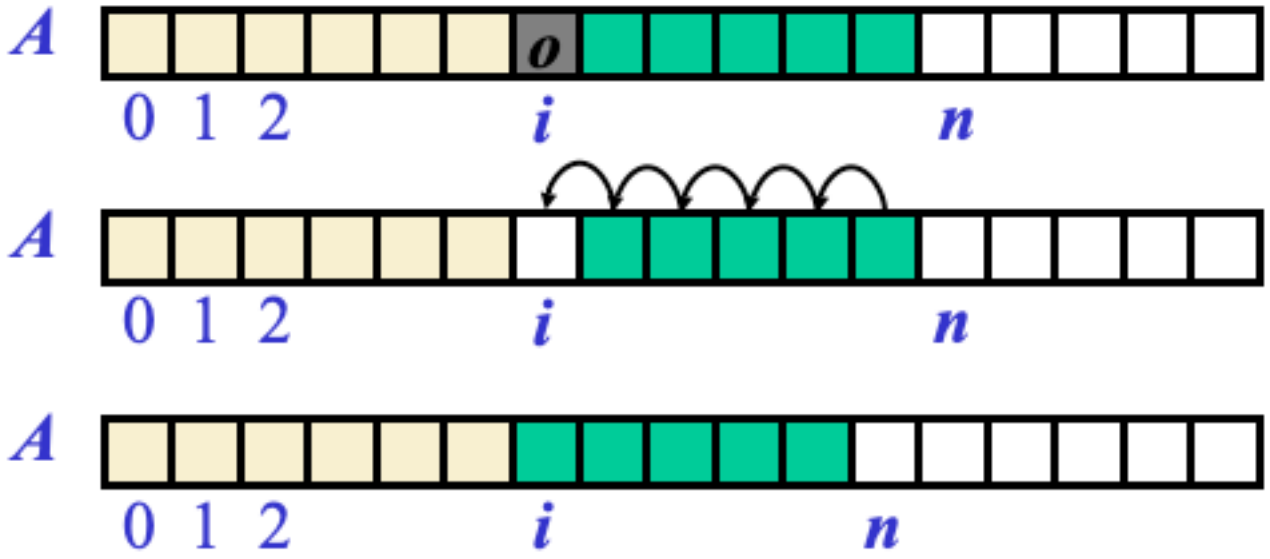
Time Complexity?

add(int index, W t)

```
    public boolean add(int index, Y t) throws IndexOutOfBoundsException
{
        if (index > count) {
            throw new IndexOutOfBoundsException("Can only add where
there are  already items");
        }
        if (index < 0) {
            throw new IndexOutOfBoundsException("Canot store to
negative location");
        }
        count++;
        if (count >= arra.length)
           grow();
        for (int i = (count - 1); i >= index; i--) {
            arra[i] = arra[i - 1];
        }
        arra[index] = t;
        return true;
    }
```

remove(index)

- Tasks
  - check to see if index is valid
  - move remaining items over to fill hole



Groups

- For the List151Impl class write

```
/** Removes the element at the specified position in this list. Shifts any
  * subsequent elements to the left (subtracts one from their indices).
  *
  * @param index the index of the element to be removed
  */
 void remove(int index) throws IndexOutOfBoundsException;

/** Removes the given item from item from the list. Compare using
  * equals. If more than one equals, will remove only one.
  * The one removed is unspecified.
  * @param index the index of the element to be removed
  */
 void remove(Y y);
```

getInstance(Y toget)

```
public Y getInstance(Y toget) {
    for (int i = 0; i < arra.length; i++) {
        if (arra[i]!=null && arra[i].equals(toget)) {
            return arra[i];
        }
    }
    return null;
}
```

- Why does this code make any sense??

2 dimensional List151Impl

```
public class AL2d {
    public static void main(String[] args) {
        List151Impl<List151Impl<String>> al2d = new List151Impl<>();
        al2d.add(new List151Impl<String>());
        // etc
        al2d.get(0).add("Hello");
        al2d.get(0).add(1);
    }
}
```

a real mouthful!

Add an AL to the "outer" AL

add a string to the inner AL

Not legal!

Testing List151Impl

- Perfect testing would exercise and validate every line of code
  - A perfect test suite can be as hard to write as the code it is testing
  - Alternative: test-driven development
    - write the tests first, then write code that satisfies all tests
  - Tests should be written pretending you do not have the code, but rather only a pseudocode
- Tests:
  - Construct: Make different capacities
  - Construct: Hold different object types
  - Add(item): Add 1 item? Two items, Three items (once you get to three you can assume more — kind of proof by induction.)
    - how do you know they are added?
    - Is order preserved?
  - Add(item): what happens when you run out of space?
  - Add(item): wrong type addition should be caught by compiler.
  - Add(index, item): what happens in each index of out range condition?
  - Add(index, item): what happens when there is no room to add?
  - ETC.

Test Code

```
    public static void main(String[] args) {
        System.out.println("\nTest A: adding consecutive integers to List151 with capacity of
10\nResult should be 0; 0,1; 0,1,2; etc");
        for (int i = 0; i < 4; i++) {
            List151Impl<Integer> test = new List151Impl<>(10);
            for (int j = 0; j <= i; j++) {
                test.add(j);
            }
            System.out.println("\n"+i+":");
            test.display();
        }

        System.out.println("\nTest B: Fill a list to capacity, then overfill");
        List151Impl<Integer> test = new List151Impl<>(10);
        for (int i = 10; i < 20; i++) {
            test.add(i);
        }
        System.out.println("Should be numbers 10..19 in positions 0..9");
        test.display();
        System.out.println("\nOverfill!!");
        for (int i = 100; i < 105; i++) {
            if (test.add(i)) {
                System.out.println("Should have returned false!!");
            }
        }
        System.out.println("Should Still be numbers 10..19 in positions 0..9");
        test.display();
    }
```

Java Inner Classes

- A class defined WITHIN another class
  - Cannot be public (so private or protected)
- Reason
    - Encapsulation!!!!!
    - Class writer can change it as needed
    - group together data items
      - for example, key-value pairs

Inner classes

- Are real classes
- Are usually very simple
- They can inherit for other external classes or other internal classes
- Variables are “public” to the containing class
    - they are only “public” to the containing class so no encapsulation violation
    - No need for get/set accessors
        - just use . accessors

Inner class Example

```
public class OutCl {

    private class InnCl {
        private int value1;
        private String value2;
        public InnCl(int v1, String v2) {
            this.value1 = v1;
            this.value2 = v2;
        }
        @Override
        public String toString() {
            return value1 + " " + value2;
        }
}
```

INNER CLASS DEFINED … NOTe IT IS PRIVATE

Inner class used .. just like any other class from within the class. However, cannot be used from static context, so cannot be used in Main

```
    public void worker() {
        InnCl icl1 = new InnCl(1, "Bob");
        InnCl icl2 = new InnCl(2, "Carol"
        icl1.value1 = 3;
        icl2.value2 = "Alice";
        System.out.println(icl1 + "\n" +
    }

    public static void main(String[] args
        OutCl ocl = new OutCl();
        ocl.worker();
    }}
```

Generic Inner Class!

Inner class has different generic parameters than its surrounding class. Realistically, they will almost always be the same, but this allows for difference

```
public class OutCLGen<R,S> {
    /**
     * The inner class, Generically
     */
    private class InnCl<Y,Z> {
        private Y value1; // a value
        private Z value2; // another value

        public InnCl(Y v1, Z v2) {
            this.value1 = v1;
            this.value2 = v2;
        }

        public String toString() {
            return value1 + " " + value2;
        }
    }
```

```
    public void worker(R rValue, S sValue) {
        InnCl<String, String> icl1 = new InnCl<>("Alice"
        InnCl<R,S> icl2 = new InnCl<>(rValue, sValue);
        icl1.value1 = 3;
        System.out.println(icl1 + "\n" + icl2);
    }

    public static void main(String[] args) {
        OutCLGen<Integer, String> ocl = new OutCLGen<>()
        ocl.worker(42, "Carol");
    }
```

Dictionary (Map)

- A searchable collection of key-value pairs
  - A lot of this course will be involve key value pairs
    - A lot of life is about key value pairs
      - SSN, tax history
      - BMID no, student record
      - …..
- Multiple entries with the same key are not allowed
- AKA associative array

What do you do with dictionaries (Physical)

- Look up based on a key item (word)
  - to get defintion
- Add items (word and definition)
- Remove Items (word)
- Others??

Count, list keys, iterators, contains, clear

Map Interface

- https://docs.oracle.com/javase/7/docs/api/java/util/Map.html

```
public interface Map151<K, V> {
    public void put(K key, V val);
    public V get(K key);
    public boolean containsKey(K key);
    public int size();
    public Set<K> keySet();
}
```

Map Implementation

```
public class Map151Impl<K,V> implements
Map151Interface<K,V>{

    private ArrayList<Pair<K,V>> underlying = new
ArrayList<>();

    private class Pair<L,W> {
        public L ky;
        public W vl;
        Pair(L key, W value) {
            ky=key;
            vl=value;
        }
        // if needed, override equals
    }
```

Use Java standard class rather than List151Impl

Using the book's terminology, this is an unsorted ArrayList based dictionary

Map Implementation (pt 2)

```
   public boolean containsKey(K key) {
        return null != getKV(key);
    }

    private Pair<K,V>getKV(K ky) {
```

```
    }

    /**
     * The number of items in the map
     * @return The number of items in
the map
     */
    public int size() {
        return underlying.size();
    }
```

```
public void put(K key, V val) {
    Pair<K,V> pair =getKV(key);
    if (pair==null) {
        Pair<K,V> np = new Pair<>(key, val
        underlying.add(np);
    } else {
        pair.value=val;
    }
}

public V get(K key) {
    Pair<K,V> pair = getKV(key);
    if (pair!=null)
        return pair.value;
    return null;
}
```