CS151

Lists

ArrayList
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
Java Interface for List

public interface List151<W> extends BagOfStuff {
    boolean add(int index, W t) throws IndexOutOfBoundsException;
    void remove(int index) throws IndexOutOfBoundsException;
    W get(int index) throws IndexOutOfBoundsException;
    boolean set(int index, W t) throws IndexOutOfBoundsException;
    int indexOf(W t);
}

From BagOfStuff

    public int numberOfItems();    public boolean isEmpty();
    public boolean add(S p);        public S remove();
    public boolean remove(S p);     public void clear();
    public int countOf(S p);        public boolean contains(S p);
    public void display();          

Why throws exceptions???

• Signal to user that something went wrong and the operation failed.
  • Alternative: have a special return value that indicates failure.
    • both approaches work
    • some DS have methods for both

• By throwing an exception you force the user (of your class) to do something or have the program die.
Implementing List151

- List151 looks a lot like BagOfStuff.
  - Order is important
    - both during set and remove
  - So internally again use an array but this time need to be sure there are no empty spaces
- Also, speed matters
Problem ... how can you compare equality of two generics

The only functions you can assume exist for a generic are those with Object.

We will discuss ways around this limitation later in semester

Solution: Use equals. Document this! Then it is the users responsibility to either use the default equals or override it appropriately
public class List151Impl<Y> implements List151<Y> {
    /** The actual number of items stored */
    private int count;
    /** The array in which all the data is actually stored */
    private Y[] arra;
    public List151Impl() {
        this(100);
    }
    @SuppressWarnings("unchecked")
    public List151Impl(int initialCapacity) {
        arra = (Y[]) new Object[initialCapacity];
        count = 0;
    }
}
Add

- recall from StuffBag had to look through the bag to find first space

```java
@Override
public boolean add(R p) {
    int loc=0;
    while (loc < stuffArray.length && stuffArray[loc] != null) {
        loc++;
    }
    if (loc == stuffArray.length){
        return false;
    } else {
        stuffArray[loc] = p;
        return true;
    }
}
```

- For list151, there are no spaces, and end is known, so just add to end

```java
public boolean add(Y t) {
    if (count >= arra.length)
        return false;
    arra[count] = t;
    count++;
    return true;
}
```

Time complexity of these add methods
get(index)

• Tasks
  • check to see if index is valid
  • return item at index in array

public Y get(int index) throws IndexOutOfBoundsException {
  if (index > count) {
    throw new IndexOutOfBoundsException("Can only get where there are already items");
  }
  if (index < 0) {
    throw new IndexOutOfBoundsException("Cannot get from a negative location");
  }
  return arra[index];
}
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)

• live write

Time complexity of this add
indexOf(item)

- loop through all items
  - if the provided item is equal current item stop and return index of current item
- if provided item not found return -1

```java
public int indexOf(Y t) {
    for (int i = 0; i < count; i++) {
        if (arra[i] == t)
            return i;
    }
    return -1;
}
```
remove(index)

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

• For the List151Impl class write
  • remove(index)
  • remove(item)

/**
 * 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;
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);
    }
}
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 always 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.
public static void main(String[] args) {
    System.out.println("Test 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("Test 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();
}