

16.04 Virtual Lecture Notes

At first glance, deletion from an array seems as easy as insertion. You simply have to find the item to be deleted and then switch any items after it ahead by one position. Then you add null to the end. This is all there is to the deletion algorithm.

While the process itself sounds easy, it has to be carefully handled. Because deletion from an array means that a null value may be entered into the array (usually at the end), all of your methods for processing the array have to be modified to allow for there being null entries. Modifying your other methods is not part of the deletion algorithm, but should be done so that they will still work. If you do not make the changes, your other methods may not function properly when used after you have deleted items.

Let us try to modify our inventory to include the deletion algorithm.

Let us assume we start with the same five items as in the previous lessons.

```
InventoryItem[] inventory = new InventoryItem[5];

// create inventory
inventory[0] = new InventoryItem("Towel", 200);
inventory[1] = new InventoryItem("Cleaning Cart", 30);
inventory[2] = new InventoryItem("Toiletry Sets", 100);
inventory[3] = new InventoryItem("Coffee Set", 300);
inventory[4] = new InventoryItem("Pillows", 50);
```

The first type of deletion is to delete an item based upon its **location**. So, to delete from our inventory, we could write a method like this:

```
public static void deleteByLoc(InventoryItem[] list,
                               int location)
{
    if ((location > 0) && (location < list.length))
    {
        //move items up in the array
        for(int index = location; index < list.length -1;
            index++)
            list[index] = list[index + 1];
        list[list.length-1] = null;
    }
}
```

Notice that to make sure we do not experience any boundary issues, we check to make sure that the **location** to be deleted is in range. We simply move items ahead by one position, overwriting the item in the location to be deleted. This removes reference to the item at that

location and it will be deleted by Java. Finally, we make the last position in the array equal to null. This is so that there are not two references to the same element after we delete an item.

The second way is to delete an item based upon the name of an inventory item. The following method does that:

```
public static void deleteByName(InventoryItem[] list,
                                String find)
{
    int location = 0;
    int index;

    // find location of item you want to delete
    for(index = 0; index < list.length; index++)
        if ((list[index] != null) &&
            (list[index].getName().equals(find)))
        {
            location = index;
            break;
        }
    else if (list[index] == null)
    {
        location = -1;
        break;
    }

    if ((index != list.length) && (location >= 0))
    { //move items up in the array
        for(index = location; index < list.length - 1; index++)
            list[index] = list[index + 1];

        list[list.length-1] = null;
    }
}
```

Notice that, in this method, we have to be careful since a previous deletion may have inserted a null value into the array. So we added a break to stop either when we reach the end of the items in the array (due to finding a null value) or we actually find the item. Since a null item cannot have a **getName()** method, we have to be careful not to call it when we should not.

If we search through the array and find a null item, that means the item we were looking for is not in the list, so we set **location** to -1 to guarantee that we will not accidentally delete the wrong item. Also, so that we can tell that we did not go through an entire array and find nothing, **index** is declared outside the for loops. If, after the first loop, **index** is equal to the length of the array, then we know that we did not find the item to delete. If **location** is -1, then we know that we did not find the item to delete.

Finally, we add null checks to the other methods to make sure they will not try to operate incorrectly on null items:

```
public static void printInventory(InventoryItem[] list)
{
    for(int i = 0; i < list.length; i++)
        if (list[i] != null)
            System.out.println(list[i]);
}

public static void changeItem(InventoryItem[] list,
                              String find, String replace)
{
    for(int index = 0; index < list.length; index++)
        if ((list[index] != null) &&
            (list[index].getName().equals(find)))
            list[index].setName(replace);
}

public static void insertItem1(InventoryItem[] list,
                               int location, String addN,
                               int addS)
{
    //move items up in the array - last item is lost
    for(int index = list.length - 1; index > location;
        index--)
        list[index] = list[index-1];

    list[location] = new InventoryItem(addN, addS);
}

public static void insertItem2(InventoryItem[] list,
                               String find, String addN,
                               int addS)
{
    int location = 0;

    // find location of item you want to insert
    for(int index = 0; index < list.length; index++)
        if ((list[index] != null) &&
            (list[index].getName().equals(find)))
        {
            location = index;
            break;
        }
        else if (list[index] == null)
        {
```

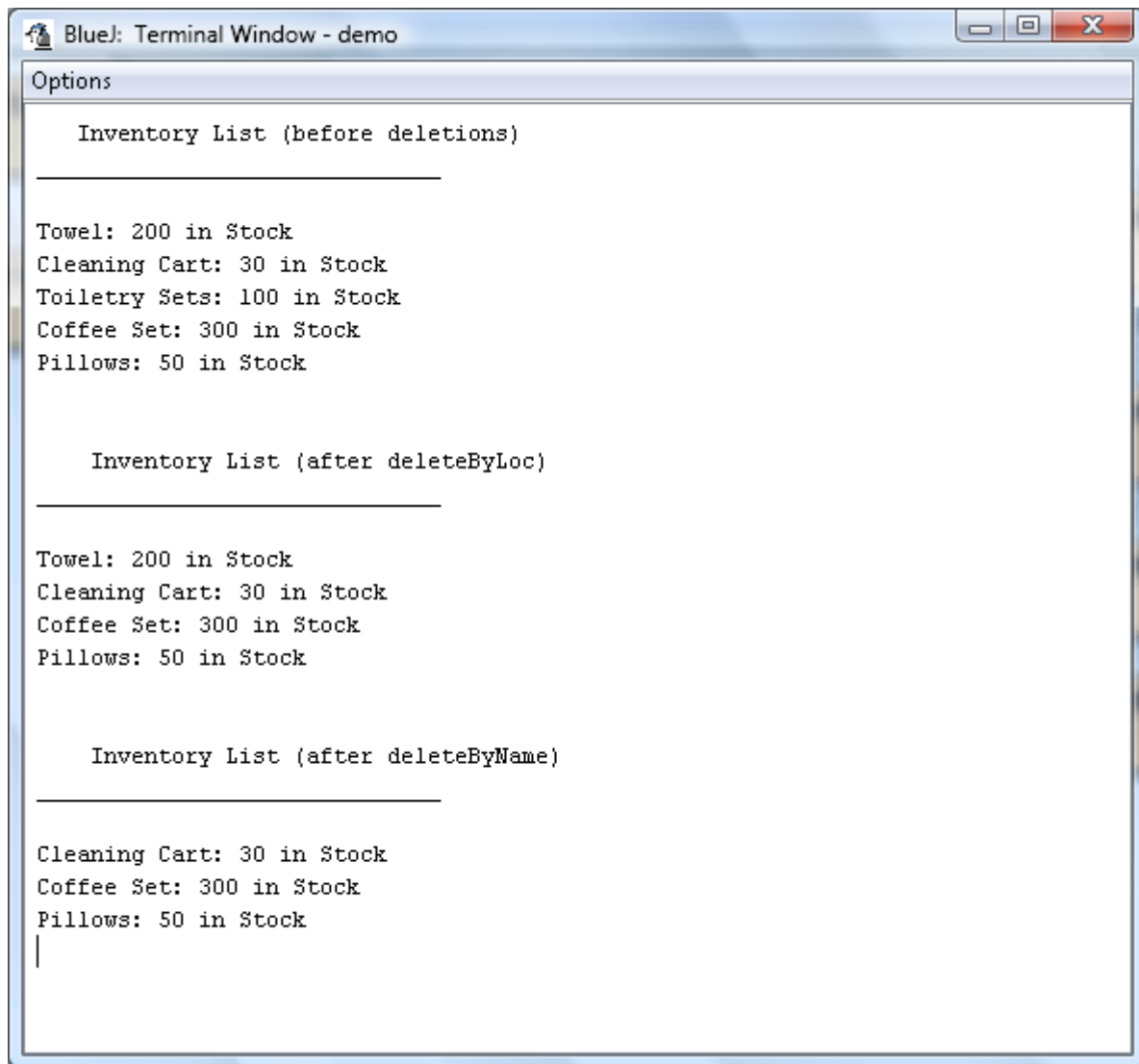
```
        location = index;
        break;
    }

    //move items down in the array - last item is lost
    for(int index = list.length - 1; index > location;
        index--)
        list[index] = list[index-1];

    list[location] = new InventoryItem(addN, addS);
}
```

You will need to really study the changes, to see that all we did was add null checks.

Anyhow, test deletions for inventory item 2 and inventory item Towel result in the following output from our **TestInventory7.java**:



```
BlueJ: Terminal Window - demo
Options

Inventory List (before deletions)
-----
Towel: 200 in Stock
Cleaning Cart: 30 in Stock
Toiletry Sets: 100 in Stock
Coffee Set: 300 in Stock
Pillows: 50 in Stock

Inventory List (after deleteByLoc)
-----
Towel: 200 in Stock
Cleaning Cart: 30 in Stock
Coffee Set: 300 in Stock
Pillows: 50 in Stock

Inventory List (after deleteByName)
-----
Cleaning Cart: 30 in Stock
Coffee Set: 300 in Stock
Pillows: 50 in Stock
|
```

- Download the TestInventory7.java file to your module 16 demo programs directory and open it.
- Run the file and make sure you understand it before you continue.

Now, how about an ArrayList? Since we are using an ArrayList, we do not have to worry about null values. We can delete an item and the ArrayList will automatically resize itself. This also means that we will not have to add any null checks to the rest of the methods. We will have to be careful when deleting by name, to make sure that we only delete if we find the item. With insertion, we did not worry about failing to find the item, as we could still insert. But there is nothing to delete if we do not find a matching item.

The first method would be modified like this:

```
public static void deleteByLoc(ArrayList<InventoryItem>  
list,
```

```
int location)
{
    // delete item from ArrayList
    list.remove(location);
}
```

Since we are using an ArrayList in this first version, we only need to use the **remove ()** method of the ArrayList. So insertion in this case is extremely easy.

The second way of deleting is also simplified when using an ArrayList.

```
public static void deleteByName (ArrayList<InventoryItem>
                                list, String find)
{
    int location = 0;
    int index;

    // find location of item you want to delete
    for(index = 0; index < list.size(); index++)
        if (list.get(index).getName().equals(find))
        {
            location = index;
            break;
        }

    // delete item from ArrayList
    if (index != list.size())
        list.remove(location);
}
```

Notice the method begins exactly as in the previous example for arrays, and then the item is deleted by using the **remove ()** method.

The output from the ArrayList test program is the same as for the array version.

Take a look at the demo program **TestInventory6.java**.

- You know the drill. Do a desk check of the program so you know what happens on each line.
- Run the file and make sure you understand it before you continue.

The deletion algorithm has an added degree of difficulty because of the need to deal with nulls. Be sure you understand how these demo programs work before proceeding to the assignment.