07.07 Virtual Lecture Notes

Array Deletion

Performing a deletion from an array is similar to an 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. Look over the TestInventory7.java program to see the changes.

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`. To delete from our inventory, we could write a method like this:

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

        itemList[itemList.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.

Our next deletion method is based upon the name of an inventory item. The following method does that:

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

   // find location of item you want to delete
   for(index = 0; index < itemList.length; index++)
   {
      if((itemList[index] != null) &&
(itemList[index].getName().equals(find)))
      {
         location = index;
         break;
      }
      else if(itemList[index] == null)
      {
         location = -1;
         break;
      }
   }
   if ((index != itemList.length) && (location >= 0))
   {
     //move items up in the array
     for(index = location; index < itemList.length -1;
index++)
       itemList[index] = itemList[index + 1];

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

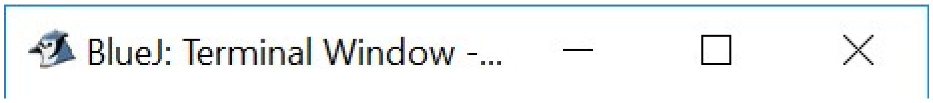
In this method, we have to be careful since a previous deletion may have inserted a `null` value into the array. So we added a break to stop either when we reach the end of the items in the array (due to finding a `null` value) or we actually find the item. Since a `null` item cannot have a `getName`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

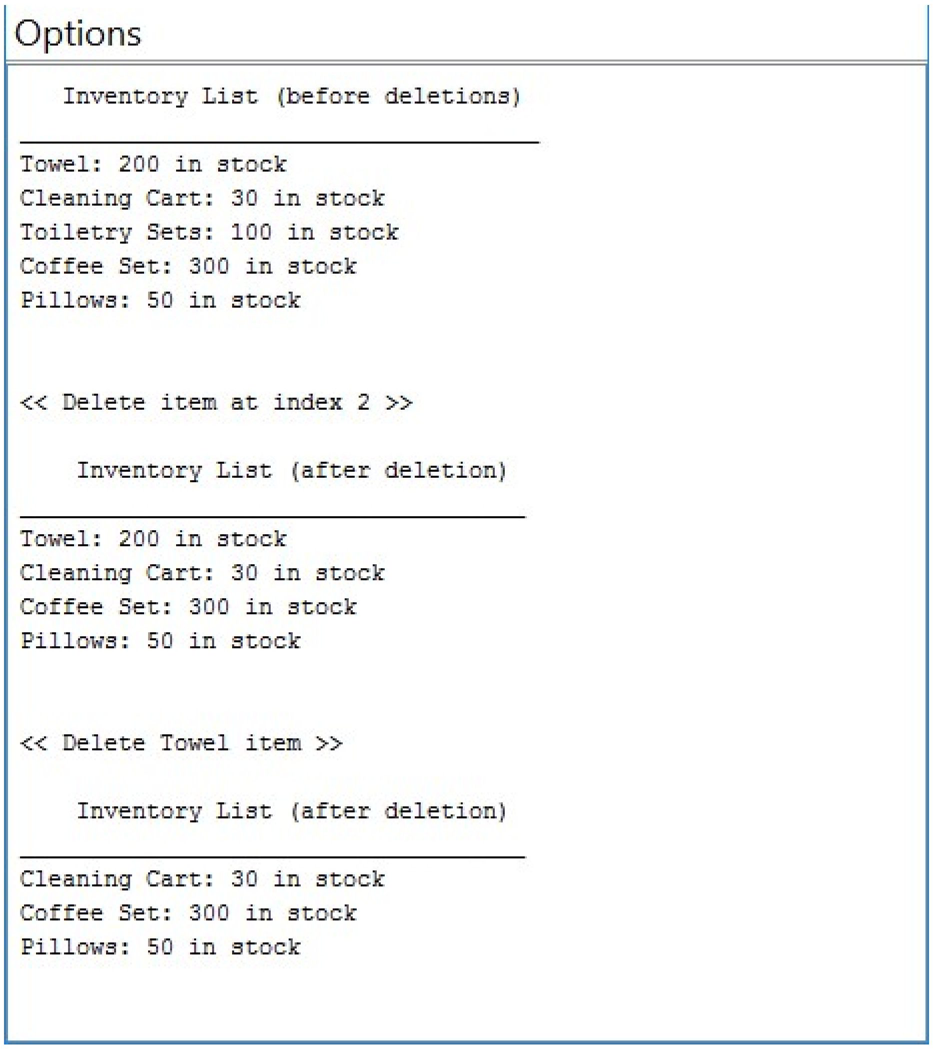
If we search through the array and find a `null` item, that means the item we were looking for is not in the list, so we set `location` to `-1` to guarantee that we will not accidentally delete the wrong item. Also, so that we can tell that we did not go through an entire array and find nothing, `index` 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[] itemList)
{
   for(int i = 0; i < itemList.length; i++)
     if(itemList[i] != null)
       System.out.println(itemList[i]);
}
```

Run the program and observe the output after deletions are performed on the item in index 2 and the Towel item.


BlueJ: Terminal Window -...



```
   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


<< Delete item at index 2 >>

    Inventory List (after deletion)
____________________________________
Towel: 200 in stock
Cleaning Cart: 30 in stock
Coffee Set: 300 in stock
Pillows: 50 in stock


<< Delete Towel item >>

    Inventory List (after deletion)
____________________________________
Cleaning Cart: 30 in stock
Coffee Set: 300 in stock
Pillows: 50 in stock
```

ArrayList Deletion

When 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 for deleting by location would look like this:

```
public static void deleteByLoc(List<InventoryItem> itemList, int
location)
{
   // delete item from ArrayList
   itemList.remove(location);
}
```

Only the `remove` method is needed to perform the deletion by location.

The second method for deleting is also simplified when using an `ArrayList`.

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

   // find location of item you want to delete
   for(index = 0; index < itemList.size(); index++)
   {
      if(itemList.get(index).getName().equals(find))
      {
         location = index;
         break;
      }
   }
   // delete item from ArrayList
   if(index != itemList.size())
      itemList.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 TestInventory8.java and run it.

Print