



The University of the West Indies, St. Augustine
COMP 2603 Object Oriented Programming 1
2022/2023 Semester 2
Lab 11

In this lab, we will continue with the GUI and a domain class from Lab 10 using TreeSets and HashSets, and the Comparable interface. You may use either Netbeans or the BlueJ IDE for this lab

The screenshot shows a Java Swing window titled "Lab10" with a standard macOS-style title bar (red, yellow, and green buttons). The window contains a form titled "Green House Plant Details". The form has three text input fields labeled "Plant Name", "Price", and "Quantity in Stock". Below these fields are four buttons: "Add Plant", "Find Plant", "Update Plant", and "Delete Plant". Below the buttons is a large empty text area, and below that is a "Clear" button. The form is divided into two sections by a horizontal line. The bottom section is titled "Display Green House Details" and contains a large empty text area. At the bottom of the window are two buttons: "Display by Name" and "Display by Price".

Part 6: HashSet as a Collection - Duplicates not allowed, unsorted, reliance on hashCode() and equals()

Let's deal with the unwanted duplication of the plants now. We can fix this easily using a different collection - a HashSet. A **HashSet** is backed by a hash table (actually a HashMap instance) and it does not allow duplicate objects to be stored.

1. In the **Nursery** class, change the *dynamic* type of **plants** to **HashSet**. Why can we do this without needing to change any of the code in the **Nursery** class?

Answer:

2. Try adding the following plants using the GUI:

Plant Name: Aloe, price: 10.00, quantity: 50

Plant Name: Aloe, price: 10.00, quantity: 50

Did it work? Explain what you observe and why it occurs

Answer:

3. Override the **hashCode()** method in the **Plant** class so that it generates a hash code using the String produced by the **toString()** method. Repeat step 2. What do you observe?

Answer:

4. Override the **hashCode()** method in the **Plant** class so that it generates a hash code using the same criteria that is used to test equality in the **equals()** method. Repeat step 2 again. What do you observe this time? Why must these methods use the same criteria on which to base their functionality?

Answer:

5. Add a few plants:

Plant Name: Aloe, price: 10.00, quantity: 50
Plant Name: Penta, price: 10.00, quantity: 12
Plant Name: Hosta, price: 6.00, quantity: 10
Plant Name: Aloe, price: 10.00, quantity: 50

Click on the **Display by Name** button. Is the list sorted by name? Why not?

Answer:

Part 7: TreeSet as a Collection - Duplicates not allowed, sorted, reliance on hashCode() and equals(), requirement of Comparable objects

Let's deal with the sorting of the plants by name now. We can fix this easily again using a different collection - a **TreeSet**. A **TreeSet** does not allow duplicate objects to be stored. The elements are ordered using their [natural ordering](#), or by a [Comparator](#) provided at set creation time, depending on which constructor is used.

1. In the **Nursery** class, change the *dynamic* type of **plants** to **TreeSet**. Why can we do this again without needing to change any of the code in the **Nursery** class?

Answer:

2. Try adding the following plants using the GUI:

Plant Name: Aloe, price: 10.00, quantity: 50
Plant Name: Aloe, price: 10.00, quantity: 50

Did it work? Explain what you observe and why it occurs.

Answer:

3. Modify the **Plant** class so that it implements the **Comparable** interface. What method are you required to add to the **Plant** class now?

Answer:

4. Add a **int compareTo(Object obj)** method to the **Plant** class that compares the name of two **Plant** objects and returns 0 if the names are identical, returns 1 if the plant's name is alphabetically higher than the one being compared to, or returns -1 if the plant's name is alphabetically lower than the one being compared to. A **java.lang.IllegalArgumentException()** should be thrown if a non-Plant object is supplied for comparison.
5. Try adding the following plants using the GUI:
Plant Name: Aloe, price: 10.00, quantity: 50
Plant Name: Aloe, price: 10.00, quantity: 50
Plant Name: Penta, price: 10.00, quantity: 12
Plant Name: Hosta, price: 6.00, quantity: 10

Did it work? Why did this work now?

Answer:

6. Click on the **Display by Name** button. Is the list sorted by name? How about if we wanted to make it sorted in descending order instead. What would you change in the **compareTo()** method?

Answer:

Part 8: TreeSet as a Collection - Duplicates not allowed, sorted, reliance on hashCode() and equals(), use of Comparators for sorting

Let's deal with the sorting of the plants by price now. We can do this using a separate Collection to the plants collection, and a **Comparator** class.

1. In the **Nursery** class, create a private inner class called **PriceComparator** that implements the **Comparator** interface. Which method must the **PriceComparator** class provide?

Answer:

2. Write the code for the **compare(..)** method in the **PriceComparator** class with the signature: **public int compare(Object o1, Object o2)**

The method should compare the prices of two **Plant** objects and return 0 if the prices are identical, return 1 if plant 1's price is larger than plant 2's, or return -1 if the plant 1's price is smaller than plant 2's.

A **java.lang.IllegalArgumentException()** should be thrown if a non-Plant object is supplied for comparison.

3. In the **Nursery** class, in **getPlantsByPrice()** method, create a new **PriceComparator** object. Create a new **TreeSet** collection called **plantsByPrice** that orders elements based on the new **PriceComparator** object. Add all of the elements in the current **plants** collection to the **plantsByPrice** collection. The method should now return the result of invoking the **toString()** method on the **plantsByPrice** collection if it is not empty.

What does the **getPlantsByPrice()** method do now?

Answer:

4. Try adding the following plants using the GUI:

Plant Name: Aloe, price: 10.00, quantity: 50

Plant Name: Penta, price: 10.00, quantity: 12

Plant Name: Hosta, price: 6.00, quantity: 10

Does everything still work? Try out the Display by Price button. Did it sort by price? Why not? What do you need to do?

Answer:

5. In the **GreenhouseGUI** class, add functionality so that the **Display by Price** button presents a sorted list (by price) of all the plants (and their details) stored in the collection. This requires code to be added to the **sortByPriceButtonActionPerformed(..)** method that invokes the **getPlantsByPrice()** on the nursery object and displays the String returned by the in the **displayArea** JTextArea in the GUI.
6. Repeat step 4. Toggle between clicking on the **Display by Price** button and the

Display by Name button. What do you observe? Explain what is happening.

Answer:

7. We need to fix the **getPlantsByPrice()** method so that the new **PriceComparator** object used with a different collection. Delete the **TreeSet** from step 3 and all related code for that object. Create a new **ArrayList** collection called **plantsByPrice** that is initialised with all of the elements in the current **plants** collection.

TIP: There is a difference between **Collection** and **Collections** (plural) interfaces in Java

Explore the [Collections](#) interface for a method that will sort the **ArrayList** using the **PriceComparator**.

What is the name of this method and how is it invoked?

Answer:

Add code to the **getPlantsByPrice()** method based on your answer above.

8. Repeat step 4 again. Toggle between clicking on the **Display by Price** button and the **Display by Name** button. What do you observe now?

Answer:

Additional Activities - At Home Practice

- Add code to the application to make the Update Plant button work
- Add code to the application to make the Delete Plant button work.