Problems relating to Unit 5

Note: problems 1 – 4 are very straightforward. Problem 5 will take more time, because you have to work out how to handle some GUI objects, Problems 6 – 8 are short and straightforward, as is the first part of Problem 9. Parts b and c of problem 9, and problems 10 and 11, focus on using standard algorithms on a set instead of on an array: they are straightforward, but you should make sure you understand the algorithms before you start to code. Problems 12 and 13 are a little more challenging.

1. Collection: From the Java API documentation, find out the following:
   1. Is Collection a class or an interface?
   2. Does it extend any other class or interface? If so, which?
   3. Why are there no constructors listed for Collection?
   4. To what package does it belong?
   5. How many subinterfaces does it have? Name one of them.
   6. List all the Collection methods that return information about the collection without making any changes in it
   7. List all the Collection methods that make changes in the collection
2. AbstractCollection: Examine the documentation for AbstractCollection
   1. Is it a class or an interface?
   2. Is it a subclass of any class? If so, which?
   3. Does it implement any interface? If so, name it / them.
   4. How can you tell it is an abstract class?
   5. Two of its methods are abstract. Which?
   6. What makes a method an abstract method?
   7. Would this line of code compile within a program: AbstractCollection <String> c = new AbstractCollection<String>();? Give a reason for your answer
3. The sample program NameSet: Copy the sample program NameSet.java, compile and run it. How many ‘add’ commands were issued for the set? How many names appear in the display? Can you explain the difference? Add a few more names to the TreeSet, then display again: do they come up in alphabetic order? What if some of them start with lower-case letters?
4. The ‘contains()’ method: Add some code to NameSet.java after the set of names has been set up to allow the user to enter a name into a dialog: the program should then tell you whether the name is in the set (use the ‘contains’ method: this works because Strings have a .equals() method).
5. Developing a program which uses a set: write a program which lets you enter the names of the ITT clubs you might like to join, displays them in a clickable list within a message dialog so that you can click one to remove, then displays the clubs that are left when you have removed one: it should also tell you how many clubs are left. Get them organised alphabetically within the clickable list.

The program should work as follows: use an input dialog, and a loop controlled either by sentinel – finishing when you enter Stop – or by a confirmDialog, to get your club choices, adding each club name to a set as you type in their names. The program should then use a JComboBox to display the clubs, with the title ”Click the club you want to remove from the list”.

Optional Extra: click more than one club to remove, all at the same time. JLists allow you to use multiple selections, returning an Object array: if you opt for this approach, you will need to convert the array to a set in order to remove them all at the same time.

Hint: you will need to examine the constructors for JList or JComboBox to find the most suitable (none of them accepts a set): then construct an intermediate object (e.g. DefaultComboBoxModel, DefaultListModel) from the set to pass to the JList/JComboBox constructor. All the information you will need for this can be obtained from the docs for JList, JComboBox. Once a club has been selected for removal, you will need to consult the JList or JComboBox documentation to see what method to use in order to make sure that you remove the correct item from your set. If your code generates any compiler warnings, make sure you have specified a type for any collection you create.

1. The bulk methods: Make a copy of NameSet.java, then add code to set up and populate a second set called friends, containing the names of your friends. The program should do the following: (a) add your friends to the set of names and display the resulting set (b) check that this has worked by invoking containsAll() (c) remove all your friends from the set of names (use removeAll()) (d) remove Elizabeth from the list of names (e) tell you how many friends you have.
2. A Set of Players (Darts.java): Copy the instantiable class Player.java into your folder and examine it. Then write a program called Darts.java which will declare and create a HashSet of Players, add some players to it, and display a table of their names and scores. Make sure to use generic code for the set declaration and creation, and for-each for looping. What happens if you add two players with the same name? Note that you will need to use your solution to this problem as a starting-point for problems 10 and 11.
3. Non-generic collections: Copy the sample program PreJava5FriendsList.java.
   1. Compile and run it. Note the compiler warning method: the program should run just the same. Nb it needs Person.java in the same folder.
   2. Change the line which creates the list from friends = new ArrayList() to friends = new List(); and recompile. What happens now? Can you explain why this is?
   3. Could you use the ‘for-each’ construct to display the list of names? Try it and see.
4. Generic collections, also counting and averaging algorithms: Copy the sample program GenericFriendsList1.java, compile and run it. Note that it uses three different ways to reach each friend in the list. Make sure you understand the syntax of each of them.
   1. What would happen if you remove <Person> from the List declaration and creation? Try it and see. Can you solve the resulting problems using a typecast?
   2. Undo any changes you made for part (a) then add a block of code (work out the logic by writing pseudocode first) to GenericFriendsList1 which will count the number of males and the number of females, and tell you how many of each there are. Get the program to check that number of males + the number of females is the same as the number of persons in the list.
   3. Add a block of code to this program which displays all the ages (nb only the ages) and the average age. Do this in the simplest way possible, but in a way that will still work when you add more friends: think the process through first by writing pseudocode
5. Algorithm for finding biggest: Make a copy of your Darts.java program called Darts2.java, then add a section to Darts2.java which will find and display the player with the highest score. Write a pseudocode algorithm first
6. Linear Search algorithm: Add a section to Darts2.java which will ask you to enter a name, then will search the set of players to see if that player is in it, and if so, display the corresponding score.
7. (Advanced, optional) Approach the previous question differently by adding an ‘equals()’ method to Player, similar to that for Strings, which returns true if two players have the same name, false otherwise: within Darts2.java construct a ‘dummy’ Player object with the name which is to be found: the ‘contains()’ method will tell you whether there is a player of that name in the set. If you want to get the player’s score, you will need to create a List from the set, and make use of ‘get’.(Note: easier if you change to TreeSet and implement comparable. Otherwise you need to override the hashcode method.)
8. Manipulating Sets: Write a program containing (a) a static method to construct the union of two sets (b) a static method to return the intersection of two sets (c) a main which sets up and populates two sets then tests the union and the intersection method. Write pseudocode first for each of the methods.