Problems relating to Unit 6 (iterators, comparators, filtering and wrappers):

1. Iterators: Go back to the original NameSet.java, and add code to it that gets an iterator for the set and uses it to display the all the names, with the length of each name alongside it. Then add a block of code which will find and display the longest name.
2. An Iterator for Players: If you have written a program to set up a set of Players (Darts.java), add code to it which will use an iterator to display a table of player names and scores.
3. The sample program FilteredFriendsList: copy, compile, run and examine it. Note that it uses the Iterator method remove() to remove all the under-10s from the list. Change it so that instead, it removes all friends whose names start with J.
4. Filtering a Collection: Copy the sample program GenericFriendsList2.java, compile and run it, add a few more friends, both males and females, then add code which will filter out all males, leaving you with an all-female list of friends.
5. Programming Exercise, filters: Make sure you have a copy of BankAccount.java in your current folder. Write a program which will create a set and add 4 BankAccounts to it: give the accounts any account number you like, and lodge the following sums to the accounts: 200 to the first, 80 to the second, 120 to the third, and 70 to the fourth. Your program should then split your accounts into two sets: those which have balances of at least 100 euro, and those which don’t. [Hint: add all the accounts in the original set to a new set, so that you have two sets each containing the original accounts, then make use of an iterator to filter both sets.]
6. SortedSets of objects: Copy the sample program PersonSet.java, compile and run it. Note that the friends don’t appear in the same order as they were added to the Set. The order within a Set is arbitrary. Now add the following code (similar to what was in NameSet) to it, and see what happens (read the error message carefully):

Set<Person> sortedFriends = new TreeSet<Person>(friends);

System.out.println(sortedFriends);

Comment out this code, but don’t delete it.

Adding a ‘compareTo’ method: Open your Person class and add the words implements Comparable <Person> to its header. Note the error message you get when you compile.

Add a compareTo() method to Person as follows:

public int compareTo(Person other){

if (this.age< other.getAge())

return -1;

else if (this.age ==other.getAge())

return 0;

else

return 1;

}

Put back the code you commented out earlier in PersonSet.java, compile, and re-run. Does it work now?

Comparing by name: Perhaps you’d prefer to have your friends listed alphabetically. Change the compareTo() method so that it does a String comparison on the names instead, then rerun PersonSet. Check that your friends are being listed in alphabetical order. [ Hint: this method can be very short if it invokes the String compareTo method.]

1. The sample ComparePlayers classes. Copy the sample classes Player, ComparePlayerNames, ComparePlayersByScore and ComparePlayersTester, examine them, then run ComparePlayersTester. Note that it displays the players sorted first by name, then by score.
   1. What attributes does a Player have?
   2. What interface is being implemented by ComparePlayerNames?
   3. How many methods are there in this interface?
   4. Why does ComparePlayerName compile without an implementation of the second method?
   5. The compareTo() method within ComparePlayerNames works by invoking the compareTo() method for which other class?
   6. What does compareTo() return?
   7. What Set implementation class is used to hold the original list of Players?
   8. What Set implementation class is used to hold the list of players ranked by high score?
   9. How many constructors are there for a TreeSet? Which one is being used in ComparePlayersTester?
2. Writing Comparators. Copy the sample entity class Bicycle.java, write Comparators for it that will compare bicycles by (a) owner name (b) make (c) value, then write an application that will declare and populate a Set of bicycles (use a version of the OOP2 BicycleFrame sample class that holds its data in a set rather than an array if you wish), ask the user would s/he like the bicycle list to be displayed in order of owner name or in order of value, then construct, populate and display the corresponding TreeSet. Use an iterator for the display.
3. The Wrapper classes. Check the API documentation for the Wrapper classes Integer and Double. Do they implement Comparable? Do they contain a compareTo() method?
4. Write a program that wraps each int from 1 to 10 into an Integer object, adds them to a Set and displays the set. Use an iterator to traverse the set, extracting the int values from the Integer objects and totalling them, outputting the total of the numbers at the end. Check by using a calculator or a spreadsheet function that you are getting the right answer. Note that in the latest versions of java, ‘autoboxing’ eliminates the need for the wrapping and the extraction. Make a simpler version of your program that takes advantage of this.
5. Set up a short text file using Notepad or Word pad, then write a program that reads words from the text file one by one, using a Scanner and the next() method, and adds them to a set: the program should then display a list of the words in the file sorted alphabetically and tell you how many different words were used in the file. Optional extra: by taking this approach, write a program which will also tell you whether a particular keyword (entered by the user) is contained in a given text file.