ASDV 2520
Data Structures and Algorithms
Comparator, Stack, Queue, PriorityQueue , Vector, LinkedList, Collection and Collections

1. Create appropriate classes in Netbeans and test/run the posted under Modules chapter 20 HTMLs: TestArrayAndLinkedList, GeometricObjectComparator-TestComparator, SortStringByLength, PrirityQueueDemo and EvaluateExpression. Download and test MultipleBallsWithComparator. Study the method of class Ball public int compareTo(Ball b), which is used to remove or add a ball.

2. Create a class A1 as shown below. We will create a List of type A1 and sort it using the Collections. Make A1 public class A1 implements Comparable<A1> , as the List NEEDS a comparator to sort it.

```
6
7   public class A1 implements Comparable<A1>
8   {
9       int x;
10      public A1() {}
11      public A1( int x ) { this.x = x;}
12      @Override
13      public int compareTo(A1 o)
14      {
15          return  this.x - o.x ;
16      }
17
18      @Override
19      public String toString()
20      {
21          return "A{" + "x=" + x + '}';
22      }
23
24      public static void main(String[] args)
25      {
26          System.out.println("Sorting in ascending order");
27          List<A1> list1 = Arrays.asList(new A1(3), new A1(), new A1(2));
28
29          Collections.sort(list1);
30          System.out.println(list1);
31
32          System.out.println("Sorting in decending order");
33          Collections.sort(list1, Collections.reverseOrder());
34          System.out.println(list1);
35      }
36  }
37
```

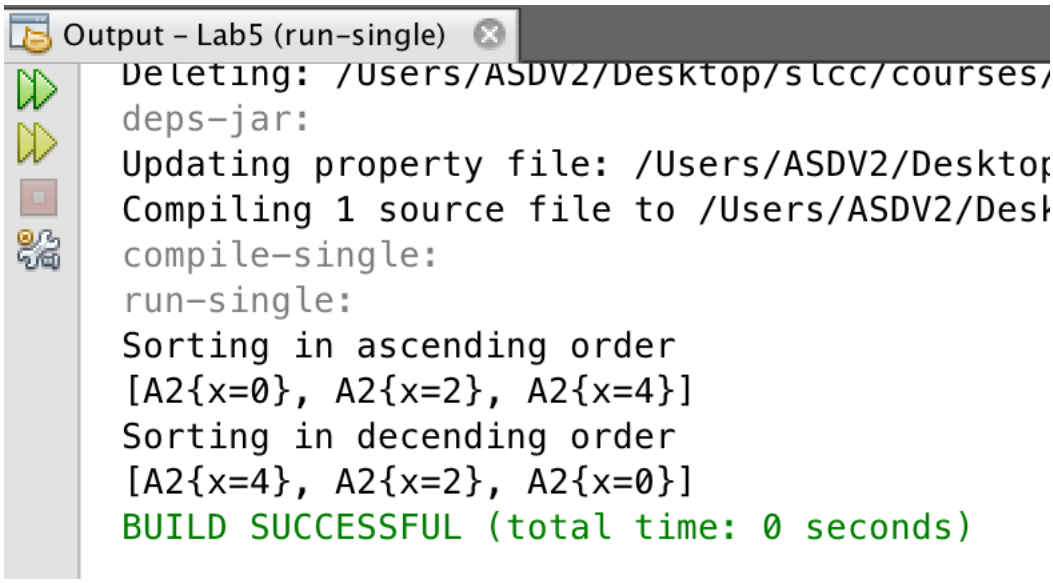
```
Output - Lab5 (run-single)
Deleting: /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/Lab5
deps-jar:
Updating property file: /Users/ASDV2/Desktop/slcc/courses/2017
Compiling 1 source file to /Users/ASDV2/Desktop/slcc/courses/2
compile-single:
run-single:
Sorting in ascending order
[A{x=0}, A{x=2}, A{x=3}]
Sorting in decending order
[A{x=3}, A{x=2}, A{x=0}]
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Copy and paste class A1 and REFACTOR it as class A2. This time will will provide a comparator manually as A2 does not implement the interface Comparable. Understand lines 21-27 that create the Comparator object and pass it as an argument to Collections.sort(). When we crate a a new Comparator<A2> Interface and we implement all abstract methods we implement the method compare(A2 o1, A2 o2) and not the compareTo(A2 o).

```
1  package collections;
2  import java.util.Arrays;
3  import java.util.Collections;
4  import java.util.Comparator;
5  import java.util.List;
6
7  public class A2
8  {
9      int x;
10
11     public A2() {}
12     public A2( int x ) { this.x = x;}
13
14     @Override
15     public String toString(){  return "A2{" + "x=" + x + '}'; }
16
17     public static void main(String[] args)
18     {
19         System.out.println("Sorting in ascending order");
20         List<A2> list1 = Arrays.asList(new A2(4), new A2(), new A2(2));
21         Comparator<A2> c = new Comparator<A2>() {
22             @Override
23             public int compare(A2 o1, A2 o2)
24             {
25                 return o1.x - o2.x;
26             }
27         };
28         Collections.sort(list1, c);
29         System.out.println(list1);
30     }
31 }
```

```
Output - Lab5 (run-single)
ant -f /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/Lab5 -Djavac.includes=collections/A2.java -Dnb.inter
init:
Deleting: /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/Lab5/build/built-jar.properties
deps-jar:
Updating property file: /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/Lab5/build/built-jar.properties
Compiling 1 source file to /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/Lab5/build/classes
compile-single:
run-single:
Sorting in ascending order
[A2{x=0}, A2{x=2}, A2{x=4}]
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Modify the class A2 inside main and provide a comparator to sort in reversed order and print the output displayed below:



4. Copy and Refactor class A2 into class A3 as shown below. This time class A3 provides the comparator from within a static method.

```
1   package collections;
2   import java.util.Arrays;
3   import java.util.Collections;
4   import java.util.Comparator;
5   import java.util.List;
6   public class A3
7   {
8       int x;
9       public A3() {}
10      public A3( int x ) { this.x = x;}
11
12      public static Comparator<A3> comparator()
13      {
14        Comparator<A3> c =  new Comparator<A3>() {
15            @Override
16            public int compare(A3 o1, A3 o2)
17            {
18                return o1.x - o2.x;
19            }
20        };
21        return c;
22      }
23      @Override public String toString(){return "A3{" + "x=" + x + '}'; }
24      public static void main(String[] args)
25      {
26          System.out.println("Sorting in ascending order");
27          List<A3> list1 = Arrays.asList(new A3(4), new A3(), new A3(2));
28          Collections.sort(list1, A3.comparator());
29          System.out.println(list1);
30      }
31  }
```

```
Output - Lab5 (run-single)
ant -f /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/Lab
init:
Deleting: /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/
deps-jar:
Updating property file: /Users/ASDV2/Desktop/slcc/courses/
Compiling 1 source file to /Users/ASDV2/Desktop/slcc/cours
compile-single:
run-single:
Sorting in ascending order
[A3{x=0}, A3{x=2}, A3{x=4}]
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Add a new static method to class A3 and call it called comparatorReverse. Use the method inside main () so it displays the list in reverse order.

6. Now we will utilize the power of Generics to create a class A4 that takes and uses a generic type E. Copy class A3 and refactor it as class A4. You can see in main that we create lists of all types as long as these types implement the interface Comparable.

```
1   package collections;
2
3   import java.util.Arrays;
4   import java.util.Collections;
5   import java.util.Comparator;
6   import java.util.List;
7
8   public class A4<E extends Comparable<E>>
9   {
10      E x;
11      public A4(){ }
12      public A4(E x){this.x = x;}
13
14      public static Comparator<A4> comparator()
15      {
16          Comparator<A4> c = new Comparator<A4>() {
17              @Override
18              public int compare(A4 a1, A4 a2)
19              {
20                  return a1.x.compareTo(a2.x);
21              }
22          };
23
24          return c;
25      }
26
27      @Override public String toString(){ return "A4{" + "x=" + x + '}';}
28
29      public static void main(String[] args)
30      {
31          System.out.println("Sorting in ascending order");
32          List<A4> list1 = Arrays.asList(
                                    new A4( new Integer(4) ),
                                    new A4( new Integer(1)),
                                    new A4(new Integer(2))
36                                  );
37          Collections.sort(list1, A4.comparator());
38          List<A4> list2 = Arrays.asList(
                                    new A4( new String("once") ),
                                    new A4( new String("upon")),
                                    new A4(new String("a")),
                                    new A4(new String("time")),
                                    new A4(new String("in")),
                                    new A4(new String("America"))
45                                  );
46          Collections.sort(list2, A4.comparator());
47          System.out.println(list2);
48
49      }
50  }
```

```
Output - Lab5 (run-single)
Deleting: /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/Lab5/build/bui
deps-jar:
Updating property file: /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/l
Compiling 1 source file to /Users/ASDV2/Desktop/slcc/courses/2017Fall/252
Note: /Users/ASDV2/Desktop/slcc/courses/2017Fall/2520/Lab5/src/collectior
Note: Recompile with -Xlint:unchecked for details.
compile-single:
run-single:
Sorting in ascending order
[A4{x=America}, A4{x=a}, A4{x=in}, A4{x=once}, A4{x=time}, A4{x=upon}]
BUILD SUCCESSFUL (total time: 0 seconds)
```

7. Add a new static method to class A4 and call it called comparatorReverse. Use the method inside main () so it displays both of the lists in reverse order.

8. Test the methods of the Collections class below. Be aware of what you type and learn.

```
ackage collections;

mport java.util.Arrays;
mport java.util.Collection;
mport java.util.Collections;
mport java.util.List;
mport java.util.Random;

ublic class TestTheCollections


    public static void main(String[] args)
    {
        System.out.println("Sorting in ascending order");
        List<String> list1 = Arrays.asList("red", "green", "blue");
        Collections.sort(list1);
        System.out.println(list1);

        System.out.println("Sorting in decending order");
        List<String> list2 = Arrays.asList("yellow", "red", "green", "blue");
        Collections.sort(list2, Collections.reverseOrder());
        System.out.println(list2);

        System.out.println("\nBinary search");
        List<Integer> list3
                = Arrays.asList(2, 4, 7, 10, 11, 45, 50, 59, 60, 66);
        System.out.println(list3 + " 7 is at  index: " + Collections.binarySearch(list3, 7));
        System.out.println(list3 + " 9 is at  index: " + Collections.binarySearch(list3, 9));
        System.out.println(list3 + " 100 is at  index: " + Collections.binarySearch(list3, 100));

        List<String> list4 = Arrays.asList("blue", "green", "red");
        System.out.println(list4 + " red is at  index: " + Collections.binarySearch(list4, "red"));
        System.out.println(list4 + " amber is at  index: " + Collections.binarySearch(list4, "amber"));
        System.out.println(list4 + " brown is at  index: " + Collections.binarySearch(list4, "brown"));

        System.out.println("\nReverse the list");
        List<String> list5 = Arrays.asList("yellow", "red", "green", "blue");
        System.out.println("Original list: " + list5);
        Collections.reverse(list5);
        System.out.println("Reversed list: " + list5);

        System.out.println("\nSuffle lists");
        List<String> list6 = Arrays.asList("yellow", "red", "green", "blue");
        List<String> list7 = Arrays.asList("yellow", "red", "green", "blue");

        System.out.println("Original list: " + list6);
        Collections.shuffle(list6, new Random(20));
        System.out.println("Suffled list: " + list6);

        System.out.println("Original list: " + list7);
        Collections.shuffle(list7, new Random(20));
        System.out.println("Suffled list: " + list6);
```

```

        List<String> list8 = Arrays.asList("yellow", "red", "green", "blue");
        List<String> list9 = Arrays.asList("white", "black");
        System.out.println("\nCopy into " + list8 + " the list " + list9);
        Collections.copy(list8, list9);
        System.out.println(list8);
        System.out.println("The output for list8 is [white, black, green, blue].\n"
                + "The copy method performs a\n"
                + "shallow copy: only the references of the elements from the source list are copied.");

        List<String> list10 = Arrays.asList("red", "green", "blue");
        System.out.println("\nFiil the list " + list10 + " with \'black\' ");
        Collections.fill(list10, "black");
        System.out.println(list10);

        /*
        The disjoint(collection1, collection2) method returns true if the two collections
        have no elements in common. For example, in the following code, disjoint(collection1,
        collection2) returns false, but disjoint(collection1, collection3) returns true.
        */
        System.out.println("\nCollentions.disjoint()");
        Collection<String> collection1 = Arrays.asList("red", "cyan");
        Collection<String> collection2 = Arrays.asList("red", "blue");
        Collection<String> collection3 = Arrays.asList("pink", "tan");
        System.out.println(Collections.disjoint(collection1, collection2));
        System.out.println(Collections.disjoint(collection1, collection3));


        System.out.println("\nFrequency");
        Collection<String> collection = Arrays.asList("red", "cyan", "red");
        System.out.println( collection + " red occurs " + Collections.frequency(collection, "red") + " times");
    }
```

```
Output - Lab5 (run-single)
run-single:
Sorting in ascending order
[blue, green, red]
Sorting in decending order
[yellow, red, green, blue]

Binary search
[2, 4, 7, 10, 11, 45, 50, 59, 60, 66] 7 is at  index: 2
[2, 4, 7, 10, 11, 45, 50, 59, 60, 66] 9 is at  index: -4
[2, 4, 7, 10, 11, 45, 50, 59, 60, 66] 100 is at  index: -11
[blue, green, red] red is at  index: 2
[blue, green, red] amber is at  index: -1
[blue, green, red] brown is at  index: -2

Reverse the list
Original list: [yellow, red, green, blue]
Reversed list: [blue, green, red, yellow]

Suffle lists
Original list: [yellow, red, green, blue]
Suffled list: [blue, yellow, red, green]
Original list: [yellow, red, green, blue]
Suffled list: [blue, yellow, red, green]

Copy into [yellow, red, green, blue] the list [white, black]
[white, black, green, blue]
The output for list8 is [white, black, green, blue].
The copy method performs a
shallow copy: only the references of the elements from the source list are copied.

Fiil the list [red, green, blue] with 'black'
[black, black, black]

Collentions.disjoint()
false
true

Frequency
[red, cyan, red] red occurs 2 times
BUILD SUCCESSFUL (total time: 0 seconds)
```

9. Test the Priority Queue

```
1   package test11;
2
3   import java.util.Collections;
4   import java.util.Comparator;
5   import java.util.PriorityQueue;
6
7   public class TestPriorityQueue
8   {
9
10      public static void main(String[] args)
11      {
12          PriorityQueue<String> queue1 = new PriorityQueue<>();
13          queue1.offer("Oklahoma");
14          queue1.offer("Indiana");
15          queue1.offer("Georgia");
16          queue1.offer("Texas");
17
18          System.out.println("Priority queue using Comparable:");
19          while (queue1.size() > 0)
20            {
21              System.out.print(queue1.remove() + " ");
22            }
23
24
25          Comparator<String> c = Collections.reverseOrder();
26          PriorityQueue<String> queue2 = new PriorityQueue<>(
27                  4, c);
28          queue2.offer("Oklahoma");
29          queue2.offer("Indiana");
30          queue2.offer("Georgia");
31          queue2.offer("Texas");
32
33          System.out.println("\n\nPriority queue using Comparator:");
            while (queue2.size() > 0)
35            {
36              System.out.print(queue2.remove() + " ");
37            }
38      }
39
40  }
41
```

***20.2** (Store numbers in a linked list) Write a program that lets the user enter numbers from a graphical user interface and displays them in a text area, as shown in Figure 20.17a. Use a linked list to store the numbers. Do not store duplicate numbers. Add the buttons Sort, Shuffle, and Reverse to sort, shuffle, and reverse the list.

***20.4** (Sort points in a plane) Write a program that meets the following requirements:

- Define a class named Point with two data fields x and y to represent a point's x- and y-coordinates. Implement the Comparable interface for comparing the points on x-coordinates. If two points have the same x-coordinates, compare their y-coordinates.
- Define a class named CompareY that implements Comparator<Point>. Implement the compare method to compare two points on their y-coordinates. If two points have the same y-coordinates, compare their x-coordinates.
- Randomly create 100 points and apply the Arrays.sort method to display the points in increasing order of their x-coordinates and in increasing order of their y-coordinates, respectively.

20.6 (Use iterators on linked lists) Write a test program that stores 5 million integers in a linked list and test the time to traverse the list using an iterator vs. using the get(index) method.