PROGRAMMING PROJECT 3

Posting ID: 5439-820

1 CONTENTS

2 PLEDGE

I pledge that this submission is entirely my own work. I have not attempted to find other solutions to the same problem, and I have not looked at or shown my code or write-up to any person other than the grader, tutors, or instructor. I understand that I may discuss ideas with others but I may not share code. I understand that, should I accidentally violate any of these conditions I must inform the grader and instructor immediately before submitting my work. I understand that if I am unable to explain aspects of my code to a grader when I am asked, then it will be considered cheating, and I will receive a grade of EX (failure due to a breach of academic integrity) in the course.

Signed: [Armando Minor] [02-21-16]

3 REFLECTION

Beginning my project one of the first things I did was understand the desired outcome. I took into consideration what would be the best approach for myself tackling this project. My best approach was to take the whole project and divide it into pieces to make the workload easier. Each method needed to complete the project was done individually. This approach helped me compile the desired outcome with less effort and less errors, although I did have a few. I researched a lot of my errors since many were new to me and I wasn't sure how to fix them. Once fixing them I saved the information on how to fix the errors going forward. I ran a few tests

during and after my project was completed to ensure the desired result was achieved. One of my biggest challenges was to get the desired outcome for the methods. Making each one was working correctly and efficiently. The time taken on this assignment exceeded those done earlier in the semester.

4 SOURCE CODE

```
1 import java.util.ArrayList;
2
3 /**
4  * Implements various sorting algorithms(mergesort and quicksor
     t).
5  *@author Armando Minor
6  */
7 public class Minor_PP4
8 {
9     /**
10         * Entry point for sample output.
11         *
12         * @param args the command line arguments
13         */
14        public static void main(String[] args) {
15            //Q1
16            String[] data = {"S", "O", "R", "T", "E", "X", "A",
      "M", "P", "L", "E"};
17            quicksortmid(data);
18            assert isSorted(data); //requires assertions enable
     d.
19            show(data);
20
21            //Q2
22            //Create new array called demo
23            ArrayList<Comparable> demo=new ArrayList<Comparable
     >();
24            // add elements to the list
25            demo.add(32);
26            demo.add(2);
27            demo.add(3);
28            demo.add(352);
29            demo.add(872);
30            demo.add(7);
31
32            mergeSort(demo);
33
34            System.out.println(mergeSort(demo));
35
36        }
37
```

```
38      /**
39       * Sorts the specified array of objects using the merge
     sort
40       * algorithm.
41       *
42       *
43       */
44      public static ArrayList<Comparable> mergeSort(ArrayList
    <Comparable> l){
45          if (l.size()<=1)
46              return l;
47          else{
48              ArrayList<Comparable> a=new ArrayList<Comparabl
    e>();
49              ArrayList<Comparable> b=new ArrayList<Comparabl
    e>();
50              for(int i=0;i<l.size()/2;i++)
51                  a.add(l.get(i));
52              for(int i=0;i<l.size()-a.size();i++)
53                  b.add(l.get(i+a.size()));
54              return  merge(mergeSort(a),mergeSort(b));
55          }
56      }
57
58
59      /**
60       * Merges two sorted subarrays of the specified array.
61       *@param a is first list
62       *@param b is second list
63       */
64      @SuppressWarnings("unchecked")
65      public static ArrayList<Comparable> merge( ArrayList<Co
    mparable> a,  ArrayList<Comparable> b){
66          int aMax=0;
67          int bMax=0;
68
69          ArrayList<Comparable> c=new ArrayList();
70
71          while(aMax<a.size() &&bMax<b.size()){
72              if (a.get(aMax).compareTo(b.get(bMax))< 0){
73                  c.add(a.get(aMax));
74                  aMax++;
75              }
76              else{
77                  c.add(b.get(bMax));
78                  bMax++;
79              }
```

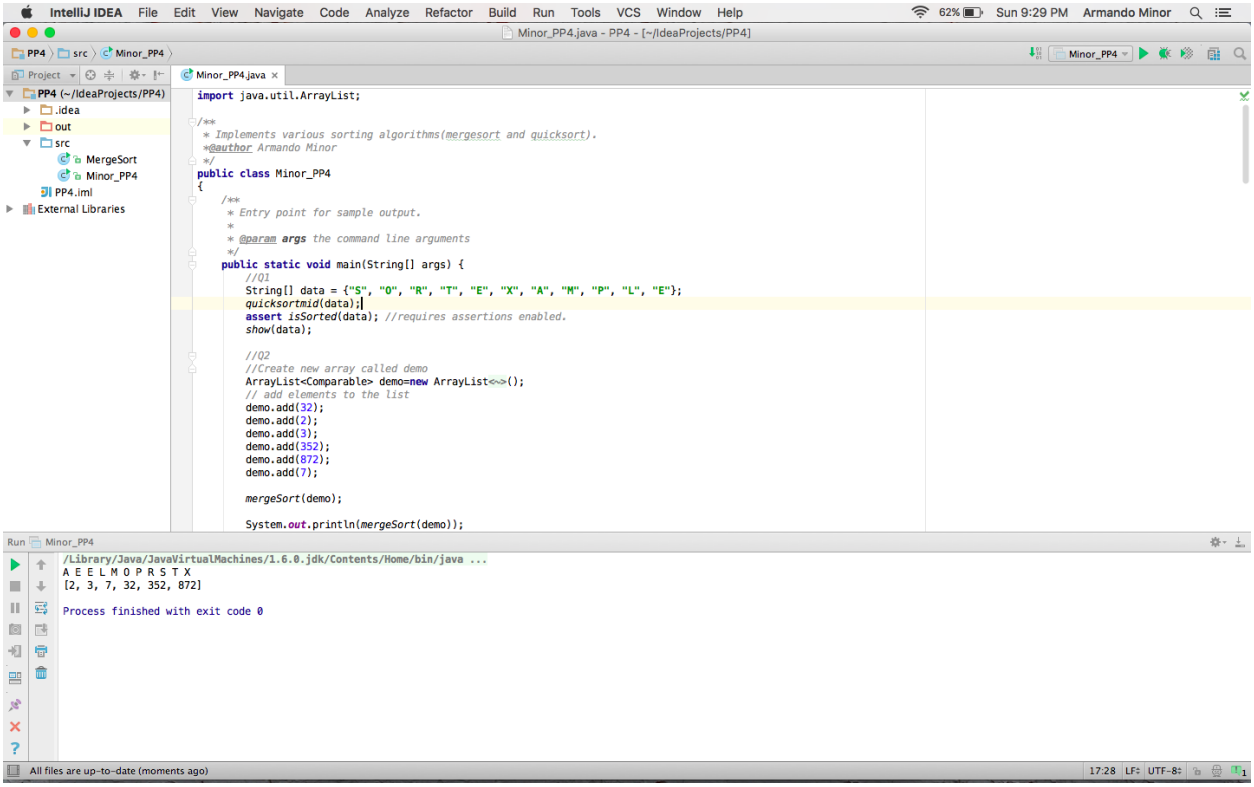
```
80          }
81          if (aMax==a.size()){
82              for(int x=bMax;x<b.size();x++){
83                  c.add(b.get(x));
84              }
85          }
86          else if(bMax==b.size()){
87              for(int x=aMax;x<a.size();x++){
88                  c.add(a.get(x));
89              }
90          }
91          return c;
92      }
93
94      /**
95       * Displays contents of array, space separated.
96       * @param data Array to display.
97       */
98      private static void show(Comparable[] data) {
99          for (Comparable a1 : data)
100             System.out.print(a1 + " ");
101
102         System.out.println();
103     }
104     //
105     /**
106      * Checks if array is in sorted order.
107      * @param data Array to be checked.
108      * @return Returns true if array is sorted.
109      */
110     public static boolean isSorted(Comparable[] data) {
111         for (int i = 1; i < data.length; i++)
112             if (less(data[i], data[i-1]))
113                 return false;
114
115         return true;
116     }
117
118     @SuppressWarnings("unchecked")
119     //Method ensures objects are less than each other
120     private static boolean less(Comparable v, Comparable w)
    {
121         return v.compareTo(w) < 0;
122     }
123     /*Quick sort method for array
124     *@param data is the array to be sorted
125     */
```

```
126
127     public static <T extends Comparable<T>> void quicksortm
    id(T[] data) {
128         qsort(data, 0, data.length-1);
129     }
130     /*Quick sort method for array
131      *@param data is the array to be sorted
132      * @param low is the lowest value of data
133      * @param hi is the highest value of data
134      */
135     private static <T extends Comparable<T>> void qsort(T[]
     data, int low, int hi) {
136         if(low >= hi) return;
137         int pi = partition(data, low, hi);
138         qsort(data, low, pi-1);
139         qsort(data, pi+1, hi);
140     }
141     /*Partition method finds partition for the array
142      *@param data is the array to be sorted
143      * @param low is the lowest value of data
144      * @param hi is the highest value of data
145      */
146     private static <T extends Comparable<T>> int partition(
    T[] data, int low, int hi) {
147         int i = low + 1;
148         int j = hi;
149
150         while(i <= j) {
151             if(data[i].compareTo(data[low]) <= 0) {
152                 i++;
153             }
154             else if(data[j].compareTo(data[low]) > 0) {
155                 j--;
156             }
157             else if(j < i) {
158                 break;
159             }
160             else
161                 swap(data, i, j);
162         }
163         swap(data, low, j);
164         return j;
165     }
166     /*swap method enables exchange of paramaters if needed
167      *@param data is the array to be sorted
168      * @param low is the lowest value of data
169      * @param hi is the highest value of data
```

```
170         */
171        private static void swap(Object[] data, int i, int j) {

172            Object temp = data[i];
173            data[i] = data[j];
174            data[j] = temp;
175        }
```

5 } COMPILATION OUTPUT



6 TESTING STRATEGY

My testing strategy for this assignments was to make sure each method was invoked properly. With each call one has to be careful and ensure each step is processed properly. Without these precautions I would have been unable to complete the project successfully. The biggest challenges were to invoke these methods properly. With the helper methods the correct structure has to be in place to make the correct calls to the methods. Once the methods are place correctly one looks for any compile errors to ensure the quality of the project. Once I finished the correct code I detailed the code with comments to enhance readability.