Client.Part1:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Client;

import DataStructures.LinkedQueue;

import Sort.\*;

import java.util.Comparator;

/\*\*

\* Client that will sort a sequence of items as specified in the part 1 of the assignment.

\* @author Jacob Huesman

\*/

public class Part1 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

/\* Create Arrays and Queues \*/

String[][] seq = genSeq();

LinkedQueue queue = new LinkedQueue();

/\* Generate Comparators \*/

Compare1 comp1 = new Compare1();

Compare2 comp2 = new Compare2();

Compare3 comp3 = new Compare3();

/\* Add all of the Strings to a queue and print out current ordering \*/

System.out.println("Initial Unsorted List:");

for(int i=0; i<seq.length; i++){

System.out.println(seq[i][0] + ", " + seq[i][1] + ", " + seq[i][2]);

queue.enqueue(seq[i]);

}

/\* Sort the first column using MergeSort \*/

Sort.mergeSort(queue, comp1);

/\* Sort the second columns using quickSort \*/

/\* Break the sequence into groups to be sorted \*/

LinkedQueue tempQueue1 = new LinkedQueue();

LinkedQueue tempQueue2 = new LinkedQueue();

LinkedQueue sortedQueue = new LinkedQueue();

while(!queue.isEmpty()){

String[] current = (String[]) queue.dequeue();

tempQueue1.enqueue(current);

while(!queue.isEmpty() && current[0].equals(((String[])queue.first())[0])){

current = (String[]) queue.dequeue();

tempQueue1.enqueue(current);

}

Sort.quickSort(tempQueue1, comp2);

/\* Nested bubbleSort sorts the third columns \*/

while(!tempQueue1.isEmpty()){

String[] current2 = (String[]) tempQueue1.dequeue();

tempQueue2.enqueue(current2);

while(!tempQueue1.isEmpty() && current2[1].equals(((String[])tempQueue1.first())[1])){

current2 = (String[]) tempQueue1.dequeue();

tempQueue2.enqueue(current2);

}

/\* Convert Queue to an ArrayList \*/

String[][] array = new String[tempQueue2.size()][];

int i = 0;

while(!tempQueue2.isEmpty()){

array[i] = (String[]) tempQueue2.dequeue();

i++;

}

/\* Sort \*/

Sort.bubbleSort(array, comp3);

/\* Covert back to an ArrayList \*/

for(i=0; i<array.length; i++){

sortedQueue.enqueue(array[i]);

}

/\* Add full sorted portion to sortedQueue \*/

while(!tempQueue2.isEmpty()){

sortedQueue.enqueue(tempQueue2.dequeue());

}

}

}

queue = sortedQueue;

/\* Print out sorted list \*/

System.out.println("\nSorted List:");

for(int i=0; i<seq.length; i++){

String[] str = (String[]) queue.dequeue();

System.out.println(str[0] + ", " + str[1] + ", " + str[2]);

}

}

/\* Method was created to reduce clutter in the main method \*/

private static String[][] genSeq(){

String[][] seq = {{"Freshman","Computer-Science","Emily"},

{"Sophomore","MIS","John"},

{"Freshman","Math","Ayushi"},

{"Sophomore","MIS","Miranda"},

{"Freshman","Computer-Science","Jacob"},

{"Freshman","Electrical-Engineering","Jacob"},

{"Junior","Math","Loran"},

{"Junior","Computer-Science","Ryan"},

{"Freshman","Computer-Science","Amy"},

{"Masters","Psychology","Jack"},

{"Freshman","Math","Emily"},

{"Freshman","Computer-Science","Chengyao"},

{"PhD","Math","Loran"},

{"Masters","History","James"},

{"PhD","Computer-Science","Emily"},

{"Masters","Biology","James"},

};

return seq;

}

private static String[] year = {"Freshman", "Sophomore", "Junior", "Senior", "Masters", "PhD"};

/\* Simple Comparison Classes \*/

private static class Compare1 implements Comparator<String[]> {

@Override

public int compare(String[] o1, String[] o2) {

int j=0,k=0;

for(int i=0; i<year.length; i++){

if(year[i].equals(o1[0])){

j = i;

} if (year[i].equals(o2[0])){

k = i;

}

}

return j - k;

}

}

private static class Compare2 implements Comparator<String[]> {

@Override

public int compare(String[] o1, String[] o2) {

return o1[1].compareTo(o2[1]);

}

}

private static class Compare3 implements Comparator<String[]> {

@Override

public int compare(String[] o1, String[] o2) {

return o1[2].compareTo(o2[2]);

}

}

}

Client.Part2:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Client;

import Employee.Employee;

import java.util.ArrayList;

/\*\*

\* Client that will sort a sequence of items as specified in the part 2 of the assignment.

\* @author Jacob Huesman

\*/

public class Part2 {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

ArrayList<Employee> emps = generateEmployees();

/\* Lab10 Part II \*/

System.out.println("Lab10 Part II");

/\* Unsorted List \*/

System.out.println("Unsorted List:");

printEmps(emps);

System.out.println("");

/\* Sort By Name \*/

System.out.println("Sorted by Name:");

printEmps(Employee.sortByName(emps));

System.out.println("");

/\* Sort By Gender \*/

System.out.println("Sorted by Gender:");

emps = generateEmployees();

printEmps(Employee.sortByGender(emps));

System.out.println("");

/\* Sort By EmpID \*/

System.out.println("Sorted by EmpID:");

emps = generateEmployees();

printEmps(Employee.sortByEmpID(emps));

System.out.println("");

/\* Sort By Office \*/

System.out.println("Sorted by Office:");

emps = generateEmployees();

printEmps(Employee.sortByOffice(emps));

}

/\* Method to generate employees \*/

private static ArrayList<Employee> generateEmployees(){

ArrayList<Employee> emps = new ArrayList<>(10);

emps.add(new Employee("Oliver", 'm', 1111, 111));

emps.add(new Employee("Alicia", 'f', 2111, 161));

emps.add(new Employee("Jaci", 'f', 1561, 112));

emps.add(new Employee("Jay", 'm', 7161, 911));

emps.add(new Employee("Sam", 'm', 8213, 541));

emps.add(new Employee("Krista", 'f', 2811, 011));

emps.add(new Employee("Mila", 'f', 7151, 179));

emps.add(new Employee("Pratap", 'm', 3213, 781));

emps.add(new Employee("Kevin", 'm', 4568, 843));

emps.add(new Employee("Josh", 'm', 2258, 159));

return emps;

}

/\* Method to print the list of employees \*/

private static void printEmps(ArrayList<Employee> emps){

for(Employee emp : emps){

System.out.println(emp.getName() + ", " + emp.getGender() + ", " + emp.getEmpID() + ", " + emp.getOffice());

}

}

}

Employee.Employee:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Employee;

import Sort.Sort;

import java.util.ArrayList;

import java.util.Comparator;

/\*\*

\* Barebones Employee class. Methods are fairly self explanatory.

\* @author Jacob Huesman

\*/

public class Employee {

/\* Instance Variables \*/

private String name;

private char gender;

private int empID, office;

/\* Constructors \*/

public Employee(String name, char gender, int empID, int office){

this.name = name;

this.gender = gender;

this.empID = empID;

this.office = office;

}

/\* Accessor Methods \*/

public String getName(){ return name; }

public char getGender(){ return gender; }

public int getEmpID(){ return empID; }

public int getOffice(){ return office; }

/\* Mutator Methods \*/

public void setName(String name){ this.name = name; }

public void setGender(char gender){ this.gender = gender; }

public void setEmpID(int empID){ this.empID = empID; }

public void setOffice(int office){ this.office = office; }

/\* Sorting Methods \*/

public static ArrayList<Employee> sortByName(ArrayList<Employee> emps){

return sort(emps, new CompareName());

}

public static ArrayList<Employee> sortByGender(ArrayList<Employee> emps){

return sort(emps, new CompareGender());

}

public static ArrayList<Employee> sortByEmpID(ArrayList<Employee> emps){

return sort(emps, new CompareEmpID());

}

public static ArrayList<Employee> sortByOffice(ArrayList<Employee> emps){

return sort(emps, new CompareOffice());

}

/\* General sorting method to sort based on given comparator \*/

private static ArrayList<Employee> sort(ArrayList<Employee> emps, Comparator comp){

Employee[] array = new Employee[emps.size()];

emps.toArray(array);

Sort.bubbleSort(array, comp);

ArrayList<Employee> list = new ArrayList<>(emps.size());

for(int i=0; i<emps.size(); i++){

list.add(array[i]);

}

return list;

}

/\* Comparators \*/

private static class CompareName implements Comparator<Employee> {

@Override

public int compare(Employee o1, Employee o2) {

return o1.getName().compareTo(o2.getName());

}

}

private static class CompareGender implements Comparator<Employee> {

@Override

public int compare(Employee o1, Employee o2) {

return o1.getGender() - o2.getGender();

}

}

private static class CompareEmpID implements Comparator<Employee> {

@Override

public int compare(Employee o1, Employee o2) {

return o1.getEmpID() - o2.getEmpID();

}

}

private static class CompareOffice implements Comparator<Employee> {

@Override

public int compare(Employee o1, Employee o2) {

return o1.getOffice() - o2.getOffice();

}

}

}

Sort.Sort:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Sort;

import DataStructures.\*;

import java.util.Comparator;

/\*\*

\* A collection of sorting algorithms based off of algorithms presented in Data Structures & Algorithms by Goodrich, Tamassia and Goldwasser.

\* @author Jacob Huesman

\*/

public class Sort {

/\*\*

\* Merge the contents of sorted queues S1 and S2 into an empty queue S

\* @param <K> Class of the object contained in the queues

\* @param S1 First queue

\* @param S2 Second queue

\* @param S Queue to have the queues S1 and S2 merged into

\* @param comp Comparator to use

\*/

public static <K> void merge(Queue<K> S1, Queue<K> S2, Queue<K> S, Comparator<K> comp){

while(!S1.isEmpty() && !S2.isEmpty()){

if(comp.compare(S1.first(), S2.first()) < 0){

S.enqueue(S1.dequeue());

} else {

S.enqueue(S2.dequeue());

}

}

while(!S1.isEmpty()){

S.enqueue(S1.dequeue());

}

while(!S2.isEmpty()){

S.enqueue(S2.dequeue());

}

}

/\*\*

\* Sorts a Queue S using a recursive mergeSort technique

\* @param <K> Class of the objects contained in the Queue

\* @param S Queue to be sorted

\* @param comp Comparator to be used

\*/

public static <K> void mergeSort(Queue<K> S, Comparator<K> comp){

int n = S.size();

if(n < 2){

return;

}

Queue<K> S1 = new LinkedQueue<>();

Queue<K> S2 = new LinkedQueue<>();

while(S1.size() < n/2){

S1.enqueue(S.dequeue());

}

while(!S.isEmpty()){

S2.enqueue(S.dequeue());

}

mergeSort(S1, comp);

mergeSort(S2, comp);

merge(S1, S2, S, comp);

}

/\*\*

\* Sorts a Queue S using a recursive quickSort technique

\* @param <K> Class of the objects contained in the Queue

\* @param S Queue to be sorted

\* @param comp Comparator to be used

\*/

public static <K> void quickSort(Queue<K> S, Comparator<K> comp){

int n = S.size();

if(n < 2){

return;

}

K pivot = S.first();

Queue<K> L = new LinkedQueue<>();

Queue<K> E = new LinkedQueue<>();

Queue<K> G = new LinkedQueue<>();

while(!S.isEmpty()){

K element = S.dequeue();

int c = comp.compare(element, pivot);

if(c < 0){

L.enqueue(element);

} else if(c == 0){

E.enqueue(element);

} else{

G.enqueue(element);

}

}

quickSort(L, comp);

quickSort(G, comp);

while(!L.isEmpty()){

S.enqueue(L.dequeue());

}

while(!E.isEmpty()){

S.enqueue(E.dequeue());

}

while(!G.isEmpty()){

S.enqueue(G.dequeue());

}

}

/\*\*

\* Sorts a generic array of objects using the bubbleSort technique

\* @param <K> Class of the objects contained in the Queue

\* @param S Array of objects to be sorted

\* @param comp Comparator to be used

\*/

public static <K> void bubbleSort(K[] S, Comparator<K> comp){

int i, j;

K temp;

for(i = (S.length - 1); i >= 0; i--){

for(j = 1; j <= i; j++){

if(comp.compare(S[j-1], S[j]) > 0){

temp = S[j-1];

S[j-1] = S[j];

S[j] = temp;

}

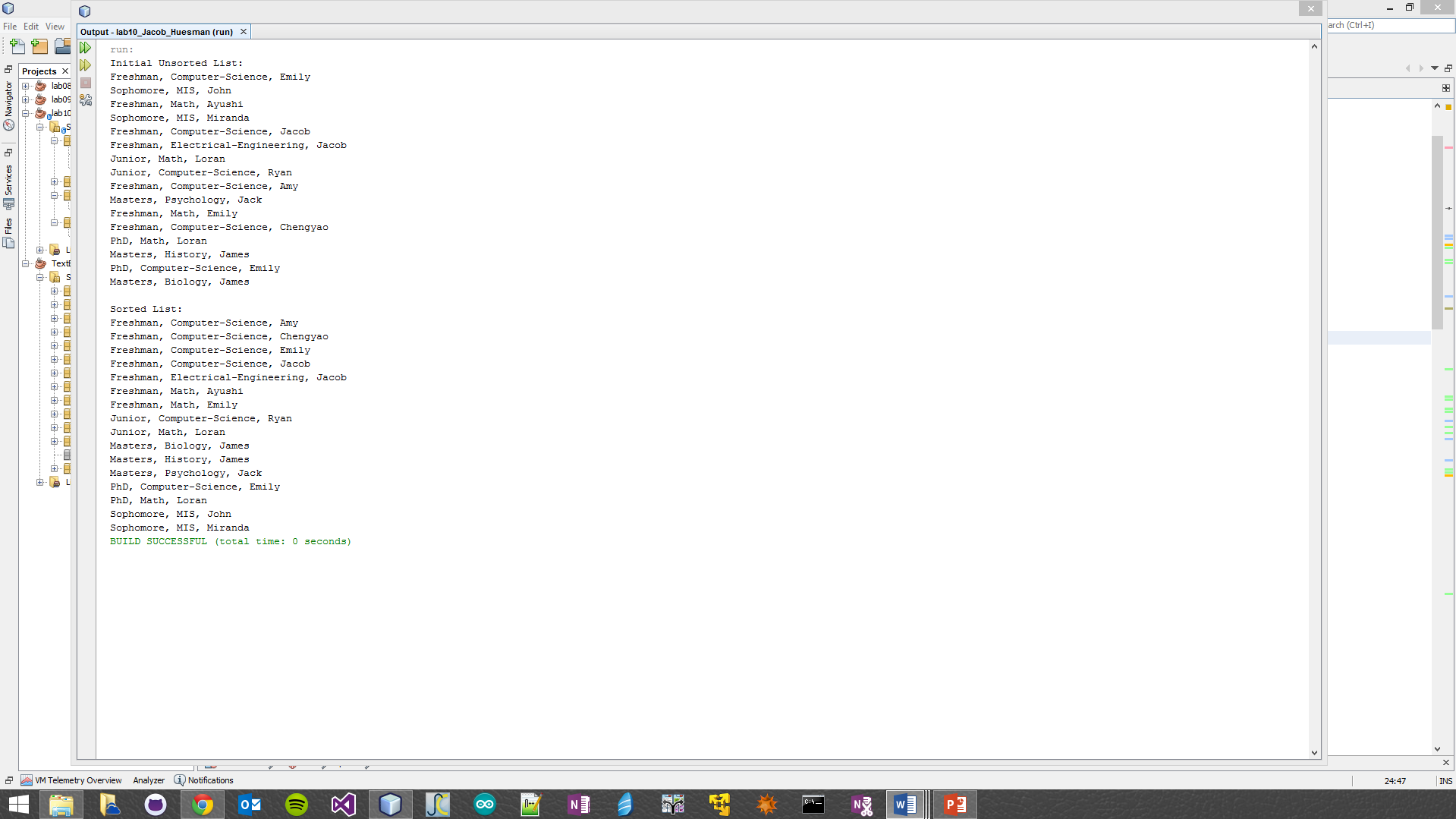
}

}

}

}

Part I:



Part II:

