```
    1: package Processing;
    2:
    3: public class SingleLinkage implements ClusterMethod {
    4:         DistanceMeasure method;
    5:
    6:         SingleLinkage (DistanceMeasure distanceMeasure) {
    7:                 method = distanceMeasure;
    8:         }
    9:
   10:         public double calculateDistance(Cluster cluster1, Cluster cluster2) {
   11:
   12:                 double minimum = 10000000000.0; // a value bigger than any result we
'd ever get
   13:
   14:                 for(int x = 0 ; x < cluster1.size() ; x++) {
   15:                         for(int y = 0 ; y < cluster2.size() ; y++) {
   16:                                 double distance = method.calculateDistance(cluster1.
get()[x], cluster2.get()[y]);
   17:                                 if(Math.abs(distance) < minimum) {
   18:                                         minimum = Math.abs(distance);
   19:                                 }
   20:                         }
   21:                 }
   22:
   23:                 return minimum;
   24:         }
   25: }
```

```
    1: package Processing;
    2:
    3: public class Node implements Cluster {
    4:         private Unit[] cluster;
    5:         private int size = 0;
    6:
    7:         Node (Cluster cluster1, Cluster cluster2) {
    8:                 this.size = cluster1.size() + cluster2.size();
    9:
   10:                 add(cluster1.get());
   11:                 add(cluster2.get());
   12:         }
   13:
   14:         private void add(Unit[] units) {
   15:                 for(int x = 0 ; x < units.length ; x++) {
   16:                         cluster[size] = units[x];
   17:                         size++;
   18:                 }
   19:         }
   20:
   21:         public boolean single() { // in case I make the mistake of adding empty unit
[] lists to the cluster, safety measure.
   22:                 if(size==1) return true ;
   23:                 else return false;
   24:         }
   25:
   26:         public int size() {
   27:                 return size;
   28:         }
   29:
   30:         public Unit[] get() {
   31:                 return cluster;
   32:         }
   33:
   34:         public double[] maximum() {
   35:                 double[] maximums = new double[cluster[0].numberRow.numberRow.length
];
   36:
   37:                 for(int x = 0 ; x < size ; x++) {
   38:                         maximums[x] = rowMaximum(x);
   39:                 }
   40:
   41:                 return maximums;
   42:         }
   43:
   44:         private double rowMaximum(int row) {
   45:                 double maximum = 0;
   46:
   47:                 for(int x = 0 ; x < size ; x++) {
   48:                         double value = cluster[x].numberRow.get(row);
   49:                         if(value > maximum) {
   50:                                 maximum = value;
   51:                         }
   52:                 }
   53:
   54:                 return maximum;
   55:         }
   56: }
```

```
1: package Processing;
2:
3: public class Cartesian implements View {
4:
5:         public void draw(ClusterRow cluster) {
6:
7:         }
8: }
```

```
    1: package Processing;
    2:
    3: import java.util.Arrays;
    4:
    5: public class Dataset {
    6:         int clusters, elements, variables, type;
    7:         public String[] variableNames;
    8:         private UnitRow unitRow;
    9:
   10:         Dataset(int clusters, int elements, int variables, String[] variableNames) {
   11:                 this.clusters = clusters;
   12:                 this.elements = elements;
   13:                 this.variables = variables;
   14:                 this.variableNames = variableNames;
   15:                 this.unitRow = new UnitRow(elements);
   16:                 process(); // move the type ( first row name ) to its own constant.
set variableNames to containt the rest
   17:         }
   18:
   19:         private void process() {
   20:                 String[] names = new String[variables];
   21:
   22:                 for(int x = 0 ; x < variables ; x++) {
   23:                         names[x] = variableNames[x+1];
   24:                 }
   25:
   26:                 variableNames = names;
   27:         }
   28:
   29:         public void addUnit(Unit unit) {
   30:                 unitRow.add(unit);
   31:         }
   32:
   33:         public Unit getUnit(int index) {
   34:                 return unitRow.get(index);
   35:         }
   36:
   37:         public void normalize() {
   38:                 unitRow.normalize(variables);
   39:         }
   40:
   41:         public void preselect() {
   42:                 // we need a way to get names after sorting through an array.
   43:                 // need to bind names to the deviation value, then sort the array an
d get the first 50
   44:                 // keep in mind that variableNames also contains the type of the fir
st row ( which is names )
   45:                 if(variables <= 50) return;
   46:
   47:                 double borderValue = getBorderValue();
   48:                 String[] names = new String[50]; // contains used names
   49:                 int[] indexes = new int[50];     // contains the used indexes so tha
t we can alter our unitrow
   50:                 int index = 0;                   // current position in names/indexe
s list
   51:
   52:                 for(int x = 0 ; x < variables ; x++) { // now compare all elements+
their results.
   53:                         if(getDeviation(x) > borderValue) {
   54:                                 names[index]   = variableNames[x];
   55:                                 indexes[index] = x;
   56:                                 index++;
   57:                         }
   58:                 }
   59:                 variableNames = names;
   60:                 updateUnits(indexes);
   61:         }
   62:
   63:         private double getBorderValue() { // used to determine what the lowest value
 in our preselection is,
   64:                                                                                 // this wa
y we can filter by grabbing anything bigger than the borderValue
   65:                 double[] results = new double[variables];
   66:
   67:                 for(int x = 0 ; x < variables ; x++) { // for each row
   68:                         results[x] = getDeviation(x);      // generate  all results
   69:                 }
   70:                 Arrays.sort(results); // sort it
   71:
   72:                 return results[results.length - 51]; // we want the -50th index ( so
 length - 51 )
   73:         }
   74:
   75:
   76:         private double getDeviation(int row) {
   77:                 return Math.sqrt(getVariance(row));
   78:         }
   79:
   80:         private double getVariance(int row) {
   81:                 double mean = average(row);
   82:         double sum = 0.0;
   83:         for(int x = 0 ; x < elements ; x++){
   84:                 double value =  unitRow.get(x).getNumber(row);
   85:             sum += (mean-value)*(mean-value);
   86:         }
   87:         double cordivisor = elements - 1;
   88:         double correction = 1 / cordivisor; // 1 / n - 1
   89:         return correction * sum;
   90:     }
   91:
   92:
   93:         private double average(int row) {
   94:                 return unitRow.average(row);
   95:         }
   96:
   97:         private void updateUnits(int[] indexes) { // we update all units to only con
tain our preselected values
   98:                 UnitRow unitRow = new UnitRow(elements);
   99:
  100:                 for(int x = 0 ; x < elements ; x++) { // for each unit
  101:                         Unit originalUnit = this.unitRow.get(x);
  102:                         Unit unit = new Unit(originalUnit.name, 50);
  103:
  104:                         for(int y = 0 ; y < indexes.length ; y++) { // for each vari
able within a given unit
  105:                                 unit.addNumber(originalUnit.getNumber(indexes[y]));
  106:                         }
  107:                         unitRow.add(unit);
  108:                 }
  109:
  110:                 this.unitRow = unitRow;
  111:                 variables = indexes.length;
  112:         }
  113: }
```

```
 1: package Processing;
 2:
 3: public class ClusterRow {
 4:         public Cluster[] clusterRow;
 5:         private int size = 0;
 6:         Dataset data;
 7:
 8:         ClusterRow(Dataset data) {
 9:                 this.data = data;
10:                 this.data.normalize();
11:                 this.data.preselect();
12:
13:                 clusterRow = new Cluster[data.elements];
14:                 process(this.data);
15:         }
16:
17:         private void process(Dataset data) {
18:                 for(int x = 0 ; x < data.elements ; x++) {
19:                         Leaf leaf = new Leaf(data.getUnit(x));
20:                         clusterRow[size] = leaf;
21:                         size++;
22:                 }
23:         }
24: }
```

```
 1: package Processing;
 2:
 3: public class NumberRow {
 4:         double[] numberRow;
 5:         private int size;
 6:
 7:         NumberRow(int maximumSize) {
 8:                 numberRow = new double[maximumSize];
 9:         }
10:
11:         public void add(double entry) {
12:                 numberRow[size] = entry;
13:                 size++;
14:         }
15:
16:         public double get(int index) {
17:                 return numberRow[index];
18:         }
19: }
```

```
 1: package Processing;
 2:
 3: public class Leaf implements Cluster {
 4:         private Unit[] units = new Unit[1];
 5:         private int size = 1;
 6:
 7:         public Leaf (Unit unit) {
 8:                 this.units[0]= unit;
 9:         }
10:
11:         public boolean single() {
12:                 return true;
13:         }
14:
15:         public int size() {
16:                 return size;
17:         }
18:
19:         public Unit[] get() {
20:                 return units;
21:         }
22:
23:         public double[] maximum() {
24:                 return units[0].numberRow.numberRow;
25:         }
26: }
```

```
 1: package Processing;
 2:
 3: public class Unit {
 4:         public NumberRow numberRow;
 5:         public String name;
 6:
 7:         Unit(String newName, int size) {
 8:                 name = newName;
 9:                 numberRow = new NumberRow(size);
10:         }
11:
12:         public void addNumber(double number) {
13:                 numberRow.add(number);
14:         }
15:
16:         public NumberRow getRow() {
17:                 return numberRow;
18:         }
19:
20:         public double getNumber(int index) {
21:                 return numberRow.get(index);
22:         }
23: }
```

```
 1: package Processing;
 2:
 3: public class Pearson implements DistanceMeasure {
 4:
 5:         public double calculateDistance(Unit unit1, Unit unit2) {
 6:                 double[] numbers1 = getNumbers(unit1);
 7:                 double[] numbers2 = getNumbers(unit2);
 8:                 double average1   = average(numbers1);
 9:                 double average2   = average(numbers2);
10:                 double derivation1= derivation(numbers1);
11:                 double derivation2= derivation(numbers2);
12:                 double sum = 0.0;
13:
14:                 for(int x = 0 ; x < numbers1.length ; x++) {
15:                         double substraction  = numbers1[x] - average1;
16:                         double leftdivision  = substraction / derivation1;
17:                         substraction = numbers2[x] - average2;
18:                         double rightdivision = substraction / derivation2;
19:
20:                         double multiplication = leftdivision * rightdivision;
21:                         sum += multiplication;
22:                 }
23:                 double divisor = numbers1.length - 1;
24:                 double result  = sum / divisor;
25:
26:                 return (1 - result);
27:         }
28:
29:         private double average(double[] numbers) {
30:                 double sum = 0.0;
31:
32:                 for(int x = 0 ; x < numbers.length ; x++) {
33:                         sum += numbers[x];
34:                 }
35:
36:                 double result =  sum / numbers.length;
37:                 return result;
38:         }
39:
40:         private double derivation(double[] numbers) {
41:                 double sum = 0.0;
42:                 double average = average(numbers);
43:
44:                 for(int x = 0 ; x < numbers.length ; x++) {
45:                         double substraction = numbers[x] - average;
46:                         double squared      = substraction * substraction;
47:                         sum += squared;
48:                 }
49:
50:                 double divisor = numbers.length - 1;
51:                 double division = sum / divisor;
52:
53:                 return Math.sqrt(division);
54:         }
55:
56:         private double[] getNumbers(Unit unit) {
57:                 return unit.numberRow.numberRow;
58:         }
59: }
```

```
1: package Processing;
2:
3: public interface ClusterMethod {
4:         double calculateDistance(Cluster cluster1, Cluster cluster2);
5: }
```

```
 1: package Processing;
 2:
 3: public interface Cluster {
 4:         //constants
 5:         //method signatures
 6:         int size();
 7:         boolean single();  // for knowing if the cluster is single or grouped
 8:         Unit[] get();
 9:         double[] maximum();
10: }
```

```
    1: package Processing;
    2:
    3: import java.util.Scanner;
    4: import java.io.PrintStream;
    5: import ui.UIAuxiliaryMethods;
    6:
    7: public class Processing {
    8:
    9:         PrintStream out;
   10:         Dataset data;
   11:
   12:         Processing() {
   13:                 out = new PrintStream(System.out);
   14:                 UIAuxiliaryMethods.askUserForInput();
   15:         }
   16:
   17:         void Start() {
   18:                 data = getDataset();
   19:                 data.normalize();
   20:                 data.preselect();
   21:
   22:                 ClusterRow clusters = new ClusterRow(data);
   23:                 Pearson method = new Pearson();
   24:                 AverageLinkage linkage = new AverageLinkage(method);
   25:                 double distance = linkage.calculateDistance(clusters.clusterRow[0],
clusters.clusterRow[1]);
   26:                 System.out.println(distance); // outputs 1.1102230246251565E-16
   27:                 // I don't implement the exact output described, but this is close e
nough
   28:         }
   29:
   30:         Dataset getDataset() {
   31:                 try {
   32:                         //Scanner in = new Scanner(new File("src/Processing/milk.txt
"));
   33:                         Scanner in = new Scanner(System.in);
   34:                         // assuming we are dealing with the standard format describe
d in the pdf
   35:                         data = process(in);
   36:
   37:                 } catch (Exception e){
   38:                         /* handle it */
   39:                 }
   40:
   41:                 return data;
   42:         }
   43:
   44:         Dataset process(Scanner in){
   45:                 int clusters = Integer.parseInt(in.nextLine());
   46:                 int elements = Integer.parseInt(in.nextLine());
   47:                 int variables = Integer.parseInt(in.nextLine());
   48:                 String[] variableNames = in.nextLine().split("\t");
   49:
   50:                 data = new Dataset(clusters, elements, variables, variableNames);
   51:
   52:                 processLines(data, in);
   53:                 in.close();
   54:
   55:                 return data;
   56:         }
   57:
   58:         void processLines(Dataset data, Scanner in) {
   59:                 // elements == the amount of lines we have to add as units.
   60:                 for(int x = 0; x < data.elements ; x++) {
   61:                         Scanner tabScanner = new Scanner(in.nextLine());
   62:                         tabScanner.useDelimiter("\t");
   63:
   64:                         processTabs(data, tabScanner);
   65:                         tabScanner.close();
   66:                 }
   67:         }
   68:
   69:         void processTabs(Dataset data, Scanner tabScanner) {
   70:                 // we know that each line contains [variables] amount of numbers and
a name
   71:                 String name = tabScanner.next();
   72:                 Unit unit = new Unit(name, data.variables);
   73:
   74:                 for(int y = 0;  y < data.variables ; y++) {
   75:                         unit.addNumber(tabScanner.nextDouble());
   76:                 }
   77:
   78:                 data.addUnit(unit);
   79:         }
   80:
   81:         public static void main(String[] args)  {
   82:                 new Processing().Start();
   83:         }
   84: }
```

```
1: package Processing;
2:
3: public interface DistanceMeasure {
4:         double calculateDistance(Unit unit1, Unit unit2);
5: }
```

```
    1: package Processing;
    2:
    3: public class AverageLinkage implements ClusterMethod {
    4:         DistanceMeasure method;
    5:
    6:         AverageLinkage (DistanceMeasure distanceMeasure) {
    7:                 method = distanceMeasure;
    8:         }
    9:
   10:         public double calculateDistance(Cluster cluster1, Cluster cluster2) {
   11:
   12:                 double sum = 0.0;
   13:
   14:                 for(int x = 0 ; x < cluster1.size() ; x++) {
   15:                         for(int y = 0 ; y < cluster2.size() ; y++) {
   16:                                 double distance = method.calculateDistance(cluster1.
get()[x], cluster2.get()[y]);
   17:                                 sum += Math.abs(distance);
   18:                         }
   19:                 }
   20:                 double divisor = cluster1.size() + cluster2.size();
   21:                 double average = sum / divisor;
   22:
   23:                 return average;
   24:         }
   25: }
```

```
 1: package Processing;
 2:
 3: public class Manhattan implements DistanceMeasure {
 4:
 5:         public double calculateDistance(Unit unit1, Unit unit2) {
 6:
 7:                 double distanceSum = 0.0;
 8:
 9:                 for(int x = 0 ; x < unit1.numberRow.numberRow.length ; x++) {
10:                         double result = unit1.getNumber(x) - unit2.getNumber(x);
11:                         distanceSum += Math.abs(result);
12:                 }
13:
14:                 return distanceSum;
15:         }
16: }
```

```
    1: package Processing;
    2:
    3: public class UnitRow {
    4:         private Unit[] unitRow;
    5:         private int size = 0;
    6:
    7:         UnitRow(int maximumSize) {
    8:                 unitRow = new Unit[maximumSize];
    9:         }
   10:
   11:         public void add(Unit unit){
   12:                 unitRow[size] = unit;
   13:                 size++;
   14:         }
   15:
   16:         public Unit get(int index){
   17:                 return unitRow[index];
   18:         }
   19:
   20:         public void normalize(int variables){
   21:                 for(int row = 0 ; row < variables ; row++) {
   22:                         double max = maximum(row);
   23:                         double min = minimum(row);
   24:
   25:                         for(int unitNum = 0 ; unitNum < size ; unitNum++) {
   26:                                 double result = normal(max, min, unitRow[unitNum].ge
tNumber(row));
   27:
   28:                                 unitRow[unitNum].numberRow.numberRow[row] = result;
   29:                         }
   30:                 }
   31:         }
   32:
   33:         private double normal(double max, double min, double element) {
   34:                 double dividend = element - min;
   35:                 double divisor = max - min;
   36:
   37:                 return dividend / divisor;
   38:         }
   39:
   40:         private double maximum(int row) {
   41:                 double maximum = 0;
   42:
   43:                 for(int x = 0 ; x < size ; x++) {
   44:                         double value = get(x).getNumber(row);
   45:                         if(value > maximum) {
   46:                                 maximum = value;
   47:                         }
   48:                 }
   49:
   50:                 return maximum;
   51:         }
   52:
   53:         private double minimum(int row) {
   54:                 double minimum = get(0).getNumber(row);
   55:
   56:                 for(int x = 0 ; x < size ; x++) {
   57:                         double value = get(x).getNumber(row);
   58:                         if(value < minimum) {
   59:                                 minimum = value;
   60:                         }
   61:                 }
   62:
   63:                 return minimum;
   64:         }
   65:
   66:         public double average(int row) {
   67:                 double sum = 0.0;
   68:
   69:                 for(int x = 0 ; x < size ; x++) {
   70:                         sum += get(x).getNumber(row);
   71:                 }
   72:
   73:                 return sum/size;
   74:         }
   75: }
```

```
1: package Processing;
2:
3: public interface View {
4:         void draw(ClusterRow cluster);
5: }
```

```
1: package Processing;
2:
3: public class Clusterer {
4:
5: }
```

```
    1: package Processing;
    2:
    3: public class Euclidean implements DistanceMeasure {
    4:
    5:         public double calculateDistance(Unit unit1, Unit unit2) {
    6:
    7:                 double distanceSum = 0.0;
    8:
    9:                 for(int x = 0 ; x < unit1.numberRow.numberRow.length ; x++) {
   10:                         double substraction = unit1.getNumber(x) - unit2.getNumber(x
);
   11:                         double squared      = substraction * substraction;
   12:                         distanceSum += squared;
   13:                 }
   14:
   15:                 return Math.sqrt(distanceSum);
   16:         }
   17: }
```

```
    1: package Processing;
    2:
    3: public class CompleteLinkage implements ClusterMethod {
    4:         DistanceMeasure method;
    5:
    6:         CompleteLinkage (DistanceMeasure distanceMeasure) {
    7:                 method = distanceMeasure;
    8:         }
    9:
   10:         public double calculateDistance(Cluster cluster1, Cluster cluster2) {
   11:
   12:                 double maximum = 0.0;
   13:
   14:                 for(int x = 0 ; x < cluster1.size() ; x++) {
   15:                         for(int y = 0 ; y < cluster2.size() ; y++) {
   16:                                 double distance = method.calculateDistance(cluster1.
get()[x], cluster2.get()[y]);
   17:                                 if(Math.abs(distance) > maximum) {
   18:                                         maximum = Math.abs(distance);
   19:                                 }
   20:                         }
   21:                 }
   22:
   23:                 return maximum;
   24:         }
   25: }
```