```java
package com.kmeans;

import java.math.BigDecimal;

public class Point {
    private int category;
    private double x;
    private double y;
    private int predictedCenter = -20;

    public Point(String[] args) {
        this.category = Integer.parseInt(args[0]);
        this.x = Double.parseDouble(args[1]);
        this.y = Double.parseDouble(args[2]);
    }

    public Point(double x, double y) {
        this.category = 0;
        this.x = x;
        this.y = y;
    }

    public void setPredictedCenter(int predictedCenter) {
        this.predictedCenter = predictedCenter;
    }

    public int getPredictedCenter() {
        return predictedCenter;
    }

    public int getCategory() {
        return category;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public boolean compareCenters(Point center)
    {
        double xDif = Math.abs(x - center.getX());
        double yDif = Math.abs(y - center.getY());

        if(xDif > 0.001 && yDif > 0.001)
            return false;
        return true;
    }

    private double round(double value, int places) {
        if (places < 0) throw new IllegalArgumentException();

        BigDecimal bd = new BigDecimal(value);
        bd = bd.setScale(places, RoundingMode.HALF_UP);
```

```
59          return bd.doubleValue();
60      }
61
62      @Override
63      public String toString() {
64          return round(x,4) + " " + round(y,4);
65      }
66  }
67
```

```java
package com.kmeans;

import java.io.BufferedReader;

public class KMeans {
    private static final int K = 3;
    private static ArrayList<String> filenames = new ArrayList<>();
    private static ArrayList<Point> setOfPoints = new ArrayList<>();
    private static ArrayList<Point> centers = new ArrayList<>();
    private static ArrayList<Integer> countPlus = new ArrayList<>();
    private static ArrayList<Integer> countMinus = new ArrayList<>();
    private static int sumPos = 0;
    private static int sumNeg = 0;

    public static void main(String[] args) throws IOException {
        if(args.length == 0)
        {
            System.err.println("Specify file names");
            System.exit(1);
        }
        filenames.addAll(Arrays.asList(args));

        for(String file : filenames)
        {
                System.out.println("*** File : " + file + " ***");
                readFile(file);
                assignCenters();
                kMeans();
                writeCentersToFile(file);
                purity();
                fMeasure();
                System.out.println("*** *** ***");
                System.out.println();
                centers.clear();
                setOfPoints.clear();
        }
    }

    private static void readFile(String filename) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(filename));

        String line = null;
        while((line = reader.readLine()) != null)
        {
            String[] args = line.split(" ");
            Point p = new Point(args);
            setOfPoints.add(p);

            if(Integer.parseInt(args[0]) > 0)
                sumPos++;
            else
                sumNeg++;
        }
    }

    private static void assignCenters()
    {
```

```
65          Random rand = new Random();
66
67          for(int i=0; i<K; i++)
68          {
69              int r = rand.nextInt(setOfPoints.size());
70              Point p = new Point(setOfPoints.get(r).getX(), setOfPoints.get(r).getY());
71              centers.add(p);
72          }
73      }
74
75      private static void kMeans()
76      {
77          ArrayList<Double> sumX = new ArrayList<>();
78          ArrayList<Double> sumY = new ArrayList<>();
79          ArrayList<Integer> count = new ArrayList<>();
80          ArrayList<Point> previousCenters = new ArrayList<>();
81          double min;
82
83          for (int i = 0; i < K; i++) {
84              sumX.add(0.0);
85              sumY.add(0.0);
86              count.add(0);
87          }
88
89          whileloop:
90          while (true) {
91              for (Point p : setOfPoints) {
92                  min = 1000;
93                  for (int i = 0; i < K; i++) {
94                      double dist = euclideanDistance(p, centers.get(i));
95                      if (dist < min) {
96                          min = dist;
97                          p.setPredictedCenter(i);
98                      }
99                  }
100
101                 int i = p.getPredictedCenter();
102                 double x = sumX.get(i) + p.getX();
103                 sumX.set(i, x);
104                 double y = sumY.get(i) + p.getY();
105                 sumY.set(i, y);
106                 count.set(i, count.get(i) + 1);
107             }
108
109             previousCenters = new ArrayList<>(centers);
110
111             for (int i = 0; i < K; i++) {
112                 Point p = new Point(sumX.get(i) / count.get(i), sumY.get(i) / count.get(i));
113                 centers.set(i, p);
114             }
115
116             for (int i = 0; i < K; i++) {
117                 if(previousCenters.get(i).compareCenters(centers.get(i)))
118                     break whileloop;
119             }
120         }
121     }
```

```
122
123     private static void writeCentersToFile(String file) throws IOException
124     {
125         String[] name = file.split("\\.");
126         BufferedWriter writer = new BufferedWriter(new FileWriter(name[0] + "_centers.txt"));
127
128         for(Point c : centers)
129         {
130             writer.write(c.toString() + "\n");
131             writer.flush();
132         }
133
134         BufferedWriter writer2 = new BufferedWriter(new FileWriter(name[0] + "_data.txt"));
135
136         for(Point d : setOfPoints)
137         {
138             writer2.write(d.toString() + "\n");
139             writer2.flush();
140         }
141
142         writer.close();
143         writer2.close();
144     }
145
146     private static double euclideanDistance(Point data, Point center)
147     {
148         double x = Math.pow((data.getX() - center.getX()),2);
149         double y = Math.pow((data.getY() - center.getY()),2);
150         return x + y;
151     }
152
153     public static void purity()
154     {
155         double sumMax = 0;
156         for(int i=0; i<K; i++)
157         {
158             int tempPlus = 0;
159             int tempMinus = 0;
160             for(Point p : setOfPoints)
161             {
162                 if(p.getPredictedCenter() == i)
163                 {
164                     if(p.getCategory() > 0)
165                         tempPlus++;
166                     else
167                         tempMinus++;
168                 }
169             }
170             if(tempPlus >= tempMinus)
171                 sumMax += tempPlus;
172             else
173                 sumMax += tempMinus;
174
175             countPlus.add(i,tempPlus);
176             countMinus.add(i,tempMinus);
177         }
178
```

```
179         System.out.print("Purity is : " + sumMax/setOfPoints.size());
180     }
181 
182     public static void fMeasure()
183     {
184         double f = 0;
185 
186         for (int i = 0; i < K; i++)
187         {
188             double precision = 0;
189             double recall = 0;
190 
191             if (countPlus.get(i) > countMinus.get(i)) {
192                 precision = countPlus.get(i) / (double)(countPlus.get(i) + countMinus.get(i));
193                 recall = countPlus.get(i) / (double)sumPos;
194             } else {
195                 precision = countMinus.get(i) / (double)(countPlus.get(i) +
    countMinus.get(i));
196                 recall = countMinus.get(i) / (double)sumNeg;
197             }
198 
199             f += 2 / ((1 / precision) + (1 / recall));
200         }
201 
202         System.out.println("\tF-measure is : " + f);
203     }
204 }
```

```java
package com.hac;

import java.io.BufferedReader;

public class HierarchicalClustering {
    private final static int K = 4;
    private final static int N = 500;
    private final static int D = 2;
    private static double[][] C = new double[N][D];
    private static int[] category = new int[N];
    private static int[] I = new int[N];
    private static HashMap<Integer,ArrayList<Integer>> clusters = new HashMap<>();
    private static ArrayList<Integer> countPlus = new ArrayList<>();
    private static ArrayList<Integer> countMinus = new ArrayList<>();
    private static int sumPos;
    private static int sumNeg;

    public static void main(String[] args) throws IOException {
        if(args.length == 0){
            System.out.println("Missing input files");
            System.exit(1);
        }

        for (String file : args) {
            readFile(file);
            hc();
            purity();
            fMeasure();
        }
    }

    private static void readFile(String filename) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(filename));

        String line = null;
        int counter = 0;
        while((line = reader.readLine()) != null)
        {
            String[] args = line.split(" ");
            C[counter][0] = Double.parseDouble(args[1]);
            C[counter][1] = Double.parseDouble(args[2]);
            category[counter] = Integer.parseInt(args[0]);
            I[counter] = 1;
            counter++;

            if(Integer.parseInt(args[0]) > 0)
                sumPos++;
            else
                sumNeg++;
        }
    }

    private static void hc() {
        int remainingClusters = N;

        while (remainingClusters > K) {
            int minPosI = -1;
```

```
62              int minPosJ = -1;
63              double minDist = 1000;
64              for (int i = 0; i < N; i++) {
65                  if (I[i] == 1) {
66                      for (int j = i + 1; j < N; j++) {
67                          if (I[j] == 1) {
68                              double dist = euclideanDistance(i, j);
69                              if (dist < minDist) {
70                                  minDist = dist;
71                                  minPosI = i;
72                                  minPosJ = j;
73                              }
74                          }
75                      }
76                  }
77              }
78
79              ArrayList<Integer> tmp = new ArrayList<>();
80              if(clusters.containsKey(minPosJ)) {
81                  tmp.addAll(clusters.get(minPosJ));
82                  tmp.add(minPosJ);
83                  if(clusters.containsKey(minPosI))
84                      tmp.addAll(clusters.get(minPosI));
85              } else {
86                  tmp.add(minPosJ);
87                  if(clusters.containsKey(minPosI))
88                      tmp.addAll(clusters.get(minPosI));
89              }
90
91              clusters.put(minPosI, tmp);
92              clusters.remove(minPosJ);
93              I[minPosJ] = 0;
94              remainingClusters--;
95          }
96      }
97
98      private static double euclideanDistance(int i, int j)
99      {
100         double x = Math.pow((C[i][0] - C[j][0]),2);
101         double y = Math.pow((C[i][1] - C[j][1]),2);
102         return x + y;
103     }
104
105     private static void purity()
106     {
107         double sumMax = 0;
108         int i=0;
109         for(ArrayList<Integer> list : clusters.values())
110         {
111             int tempPlus = 0;
112             int tempMinus = 0;
113
114             for(Integer key : list)
115             {
116                 if(category[key] > 0)
117                     tempPlus++;
118                 else
```

```
119                     tempMinus++;
120             }
121
122             if(tempPlus >= tempMinus)
123                 sumMax += tempPlus;
124             else
125                 sumMax += tempMinus;
126
127             countPlus.add(i,tempPlus);
128             countMinus.add(i,tempMinus);
129             i++;
130         }
131
132         System.out.println("Purity is : " + sumMax/N);
133     }
134
135     private static void fMeasure()
136     {
137         double f = 0;
138
139         for (int i = 0; i < K; i++)
140         {
141             double precision = 0;
142             double recall = 0;
143
144             if (countPlus.get(i) > countMinus.get(i)) {
145                 precision = countPlus.get(i) / (double)(countPlus.get(i) + countMinus.get(i));
146                 recall = countPlus.get(i) / (double)sumPos;
147             } else {
148                 precision = countMinus.get(i) / (double)(countPlus.get(i) +
    countMinus.get(i));
149                 recall = countMinus.get(i) / (double)sumNeg;
150             }
151
152             f += 2 / ((1 / precision) + (1 / recall));
153         }
154
155         System.out.println("F-measure is : " + f);
156     }
157
158 }
159
```