a) Read the entry data from the attached file: iris.csv

   Attribute Information:
   1. sepal length in cm
   2. sepal width in cm
   3. petal length in cm
   4. petal width in cm
   5. class:
      -- Iris Setosa
      -- Iris Versicolour
      -- Iris Virginica

b) Split the entry set into Training Set and Evaluation set

   Training set: 85 % of patterns
   Evaluation set: 15 % of patterns , randomly selected from each class
   IMPORTANT NOTE: make sure to keep the initial proportions between classes when choosing the evaluation set (for each class you will have to randomly select 15% of the patterns for the evaluation set; we don't want to have the evaluation set composed with patterns from only one or two classes)

c) using the above training and evaluation sets, apply KNN algorithm to the evaluation set and study the error for different values of K (1,3,5,7,9,11,13,15). Which is the optimal value for K ?

d) Use the type 3 classifier (minimum distance classifier) to classify the patterns from the evaluation set. Compare the accuracy for the two methods.

IMPORTANT NOTE: Use the optimal complexity for KNN and minimum distance classifier algorithms

For KNN algorithm, you can use a KD Tree (optimal complexity - https://www.sanfoundry.com/java-program-find-nearest-neighbour-using-k-d-tree-search/) or Heap Structure. If you plan to use a heap structure, in Java you can look at “priority queue” and “tree map” (with tree maps we will have to be careful because it does not allow duplicate keys. If we want to keep multiple equal distances in the map, we can create a map having the distance as a key and a list of classes as values: Map<Double, List<String>>). For heap structures you can use the following algorithm:

```
mh = new MxHeap()
for i=1,n execute
     *) calculate dz=dist(z, x[i])l
      If mh.size()<k then
              mh.add( {dx, C[i]} )
      else
              If dz < mh.top()  then
                     mh.remove()
                     mh.add( {dx, C[i]} )
              @
       @
@
```

For minimum distance classifier (type 3 classifier), make sure you iterate only once over the entry set (the optimal complexity: O((n+M)p)):

```
procedure  Class (n, p,  M, x[n][p], f[n], iclass[n], w[M][p+1]
   *)  init(miu[k]=0,  (w[k][j]=0, j=1,p+1), k=1,M)
   for i=1,n  execute
      k=iclass[i]
      for j=1,p  execute
          w[k][j] += x[i][j]*f[i]
      @
      miu[k] +=f[i]
  @
   for k=1,M  execute
      for j=1,p  execute
           w[k][j] /= miu[k]
           w[k][p+1] -= w[k][j]*w[k][j]/2
     @
  @
End
```

For creating the training and evaluation sets, you can use the following code (it can be optimized)

```
String[][] learningSet;
try {
        learningSet = FileUtils.readLearningSetFromFile("in.txt");
        int numberOfPatterns = learningSet.length;
        int numberOfFeatures = learningSet[0].length-1;

        Map<String, Integer> classesMap = new HashMap<String, Integer>();

        //create map with distinct classes and number of occurence for each class
        for (int i=0; i<numberOfPatterns; i++)
        {
                String clazz = learningSet[i][learningSet[i].length-1];
                if (classesMap.containsKey(clazz))
                {
                        Integer nrOfClassPatterns = classesMap.get(clazz);
                        classesMap.put(clazz, nrOfClassPatterns + 1);
                }
                else
                {
                        classesMap.put(clazz, 1);
                }
        }
        Random random = new Random();
        //map that keeps for each class the random patterns selected for evaluation set
        Map<String, List<Integer>> classesEvaluationPatterns = new HashMap<String, List<Integer>>();
        Integer evaluationSetSize = 0;
        for (Map.Entry<String, Integer> entry: classesMap.entrySet())
        {
                String className = entry.getKey();
                Integer classMembers = entry.getValue();
                Integer evaluationPatternsNr = Math.round(classMembers *15/100);
                evaluationSetSize += evaluationPatternsNr;
                List<Integer> selectedPatternsForEvaluation = new ArrayList<Integer>();
                for (int i=0; i<evaluationPatternsNr; i++)
                {
                        Integer patternNr = random.nextInt(classMembers ) +1;
                        while (selectedPatternsForEvaluation.contains(patternNr))
                        {
                                patternNr = random.nextInt(classMembers ) +1;
                        }
                        selectedPatternsForEvaluation.add(patternNr);
                }
                classesEvaluationPatterns.put(className, selectedPatternsForEvaluation);
        }

        String[][] evaluationSet = new String[evaluationSetSize][numberOfPatterns];
        String[][] trainingSet = new String[numberOfPatterns-evaluationSetSize][numberOfPatterns];
        int evaluationSetIndex = 0;
        int trainingSetIndex = 0;
        Map<String, Integer> classCurrentIndex = new HashMap<String, Integer>();
        for (int i=0; i<numberOfPatterns; i++)
        {
                String className = learningSet[i][numberOfFeatures];
                if (classCurrentIndex.containsKey(className))
                {
                        int currentIndex = classCurrentIndex.get(className);
                        classCurrentIndex.put(className, currentIndex+1);
                }
                else
                {
                        classCurrentIndex.put(className, 1);
                }
                if (classesEvaluationPatterns.get(className).contains(classCurrentIndex.get(className)))
                {
                        evaluationSet[evaluationSetIndex] = learningSet[i];
                        evaluationSetIndex++;
                }
                else
                {
                        trainingSet[trainingSetIndex] = learningSet[i];
                        trainingSetIndex++;
                }
        }

        FileUtils.writeLearningSetToFile("eval.txt", evaluationSet);
        FileUtils.writeLearningSetToFile("train.txt", trainingSet);

        } catch (USVInputFileCustomException e) {
                e.printStackTrace();
        } finally {
                System.out.println("Finished learning set operations");
        }
```