Problem

Read gradesClasses.txt (it contains the grades from test 1 and an associated class) and save the learning set in an appropriate data structure

e.g:
3.5, insufficient
8, good
9.7, very good

Requirements:

a) Find the class of the following grades: 3,80, 5.75,6.25, 7.25, 8.5 using k-NN rule for the following values of k : 1,3,5,7,9,13,17
b) Identify what is the maximum value of k for which the result is accurate
c) Create J-unit test

What is k-NN rule ?

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification. The input consists of the k closest training examples in the feature space. (Wikipedia)

Example:

Let's consider the following grade for which we want to find its class: **5.75**

1-NN
Nearest neighbor: 5.8
Nearest neighbor class: sufficient
Searched grade class: **sufficient**

3-NN
Nearest neighbors: 5.8, 6, 6
Nearest neighbor class: sufficient, sufficient, sufficient
Searched grade class: **sufficient (3 sufficient)**

5-NN

Nearest neighbors: 5.8, 6, 6, 5, 6.65

Nearest neighbor class: sufficient, sufficient, sufficient, sufficient, sufficient

Searched grade class: **sufficient (5 sufficient)**

7-NN

Nearest neighbors: 5.8, 6, 6, 5, 6.65, 4.8, 4.8

Nearest neighbor class: sufficient, sufficient, sufficient, sufficient, sufficient, insufficient, insufficient

Searched grade class: **sufficient (5 sufficient and 2 insufficient)**

9-NN

Nearest neighbors: 5.8, 6, 6, 5, 6.65, 4.8, 4.8, 4.75, 6.8

Nearest neighbor class: sufficient, sufficient, sufficient, sufficient, sufficient, insufficient, insufficient, insufficient, sufficient

Searched grade class: **sufficient (6 sufficient and 3 insufficient)**

13-NN

Nearest neighbors: 5.8, 6, 6, 5, 6.65, 4.8, 4.8, 4.75, 6.8, 6.85, 4.5, 4.5, 4.5

Nearest neighbor class: sufficient, sufficient, sufficient, sufficient, sufficient, insufficient, insufficient, insufficient, sufficient, sufficient, insufficient, insufficient, insufficient

Searched grade class: **sufficient (7 sufficient and 6 insufficient)**

15-NN

Nearest neighbors: 5.8, 6, 6, 5, 6.65, 4.8, 4.8, 4.75, 6.8, 6.85, 4.5, 4.5, 4.5 , 4.5, 4.25

Nearest neighbor class: sufficient, sufficient, sufficient, sufficient, sufficient, insufficient, insufficient, insufficient, sufficient, sufficient, insufficient, insufficient, insufficient, insufficient, insufficient

Searched grade class: **insufficient (7 sufficient and 8 insufficient)**

We notice that we were able to identify the correct class of the searched grade using the following k values : 1,3,5,7,9,13

Steps & Hints

1. **Be organized.**

Keep classes small

So far you've created a few classes. It's not uncommon (and it's unfortunate) to see classes with 50 or 100 methods and a thousand lines or more of source. Some classes might be that large out of necessity, but most likely they need to be refactored. Refactoring is changing the design of existing code without changing its results. I recommend that you follow this best practice.

In general, a class represents a conceptual entity in your application, and a class's size should reflect only the functionality to do whatever that entity needs to do. Keep your classes tightly focused to do a small number of things and do them well.

Keep only the methods that you need. Be sure to limit the list of methods to what you need, and no more.

Name methods carefully

A good coding pattern when it comes to method names is the intention-revealing method-names pattern. This pattern is easiest to understand with a simple example. Which of the following method names is easier to decipher at a glance?

- cED()
- calculateEuclidianDistance()

The answer should be obvious, yet for some reason, programmers have a tendency to give methods (and variables, for that matter) small, abbreviated names. Six months after you write a bunch of code, you might not remember what you meant to do with a method called cED (), but it's obvious that a method called calculateEuclidianDistance (), well, probably calculates the euclidian distance.

Keep methods small

Small methods are as preferable as small classes. One idiom I try to follow is to keep the size of a method to one page as you look at it on your screen.

If a method grows beyond one page, refactor it. Usually, a long method contains subgroups of functionality bunched together. Take this functionality and move it to another method (naming it accordingly) and pass in parameters as needed.

Limit each method to a single job. A method doing only one thing well doesn't usually take more than about 20-25 lines of code (without brackets and comments).

Use comments

Please, use comments. The people who follow along behind you (or even you, yourself, six months down the road) will thank you.

2. **Update the method that reads from file to be able to save strings. Think of what other data structures you may use**
3. **Use a List to keep the grades for which we want to find the class**
4. **Use a Map to keep the found class for each k value**

   e.g : For the previous example we can use the following Map:

Key	Value
1	**sufficient**
3	**sufficient**
5	**sufficient**
7	**sufficient**
9	**sufficient**
13	**sufficient**
15	**insufficient**

5. **Create Junit tests**

   JUnit is an open source Unit Testing Framework for JAVA. It is useful for Java Developers to write and run repeatable tests. As the name implies, it is used for Unit Testing of a small chunk of code.

   Once you are done with code, you should execute all tests, and it should pass. Every time any code is added, you need to re-execute all test cases and makes sure nothing is broken.

   http://www.vogella.com/tutorials/JUnit/article.html
   https://courses.cs.washington.edu/courses/cse143/11wi/eclipse-tutorial/junit.shtml
   https://www.tutorialspoint.com/junit/junit_test_framework.htm