Wentworth Institute of Technology
COMP1050 – Computer Science II
Spring 2015, Derbinsky

Part 3

For this assignment you are to implement a set of classes in Java. You must document the classes using Javadoc and turn in the source files, in the appropriate folder structure, as a zip file. You will be evaluated based upon correctness (adhering to the problem description, passing test cases), object-oriented design (e.g. appropriate usage of inheritance, member access control, etc.), and source code documentation (via Javadoc).

Assignment Overview

This assignment is the third building block to your final project and involves implementing classes given an API, supporting source/jars, and a set of unit tests. You are going to implement a hierarchy of classes supporting datasets.

0. Assignment Setup

First, download and unzip the `p3-starter.zip` file.

Next, import the contents of the file as a project in Eclipse. You will see red error icons – this is to be expected.

Finally, you should have available for reference the JavaDoc for the whole project (`final-doc.zip`), which includes the classes you must write for this part. (Note: it also includes classes you won't need for this particular project, but that make up the final project.)

1. Your Task

You are required to implement several classes. In doing so, you must adhere to the following general restrictions (additional notes are in subsections below):

- You cannot modify the provided public/protected API in any way.
- You may create variables/methods, but they must be `private`.

The methods you are to implement are all documented, and the test cases provide example usage. The following sections discuss the classes you are to implement.

If you pass all of the test cases, your implementation is in good shape. You may wish to create additional test cases in a separate file, but only turn in the source code for the classes you required to implement.

1.0 `Iterable`

The `Iterable` interface simply requires that a class return an iterator, which is an object that itself implements the `Iterator` interface (note that both of these interfaces are typed via generics; note also that these can be the same object – hint hint!). One of the syntactic reasons for a class to be iterable is so that you can use the foreach loop (e.g. `for (Foo f : bar)`, assuming bar implements `Iterable<Foo>`).

So, to make a class iterable, you are going to have to implement a class (typically a private inner class) that implements `Iterator` and return it inside the required `iterator` method. Your custom iterator must only implement two methods: `hasNext` (returns true if there is anything left to iterate over) and `next` (returns the next object to be iterated over). All Java collections are iterable, which is why you can easily loop over their contents (hint hint – this might be useful if you have a list of examples laying around...). You will be making your datasets iterable as well, so you can easily loop over their examples (and in a later assignment, you will make each example iterable, so you can loop over the features).

1.1 `DataSet`

The `DataSet` class provides an abstraction for iterating over examples, without committing to a particular backend mechanism of storage/representation.

1.2 `HiddenPatternDataSet`

The `HiddenPatternDataSet` allows you to define a dataset via simple patterns. Each pattern states that for a particular result object, there are a certain set of features that will have a fixed value. For example, all "dog" examples will have feature 1 equal to 1 and feature 3 equal to 2, whereas all "cat" examples will have feature 2 equal to 1 and feature 4 equal to 2. Any unspecified features will either be 0 or a random value in a known range, as defined by class configuration.

This dataset has [at least] two interesting aspects. First, the object **should not store any examples**, but should instead generate them reliably and repeatedly on demand. Thus, no matter how many examples are generated, the amount of memory needed by the class should only be as big as the number of patterns provided by the user. Second, if examples are made that use 0 for unspecified features, the class should generate `SparseExample` objects; if instead the features are random-valued, the class should generate `DenseExample` objects. Importantly, for random values, you will need to make use of a random seed such that two consecutive iterations over the dataset yield the same results.

To implement this class you may find it useful to use the `LinkedHashMap`[1] class. This class implements the `Map` interface[2] and its implementation is such that you are able to iterate over the keys in the order they were added into the map.

1.3 `ListDataSet`

Many sources of examples (e.g. files) would work well in a list, and so a `ListDataSet` is an abstract class that facilitates such storage, using an `ArrayList` backend.

[1] See `http://docs.oracle.com/javase/8/docs/api/java/util/Map.html`
[2] See `http://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html`

1.4 `LibSVMDataSet`

The `LibSVMDataSet` builds on the `ListDataSet`: it parses an input stream, in LibSVM format, and adds each example to the dataset. Each example starts with the result object and, to provide maximum flexibility, the LibSVMDataSet takes as an argument a `ResultInterpreter` object to convert the first string on each file into an appropriate Java object (e.g. an integer). The `LibSVMDataSet` provides the `ResultInterpreter` interface, as well as two static implementations, one for strings (`StringInterpreter`) and the other for integers (`IntegerInterpreter`).

When parsing the input stream, which may be from a file or some other source (e.g. network transfer), you may find it useful to use a `BufferedReader`[3], with an `InputStreamReader`[4], as well as a `StringTokenizer`[5].

2. Opportunities

For extra credit, write a separate console program that allows a user to specify an arbitrary `HiddenPatternDataSet` (via interactive questions) and then writes the resulting dataset to a file the user specifies.

[3] See `http://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html`
[4] See `http://docs.oracle.com/javase/8/docs/api/java/io/InputStreamReader.html`
[5] See `https://docs.oracle.com/javase/8/docs/api/java/util/StringTokenizer.html`