데이터사이언스

Decision Tree implement assignment

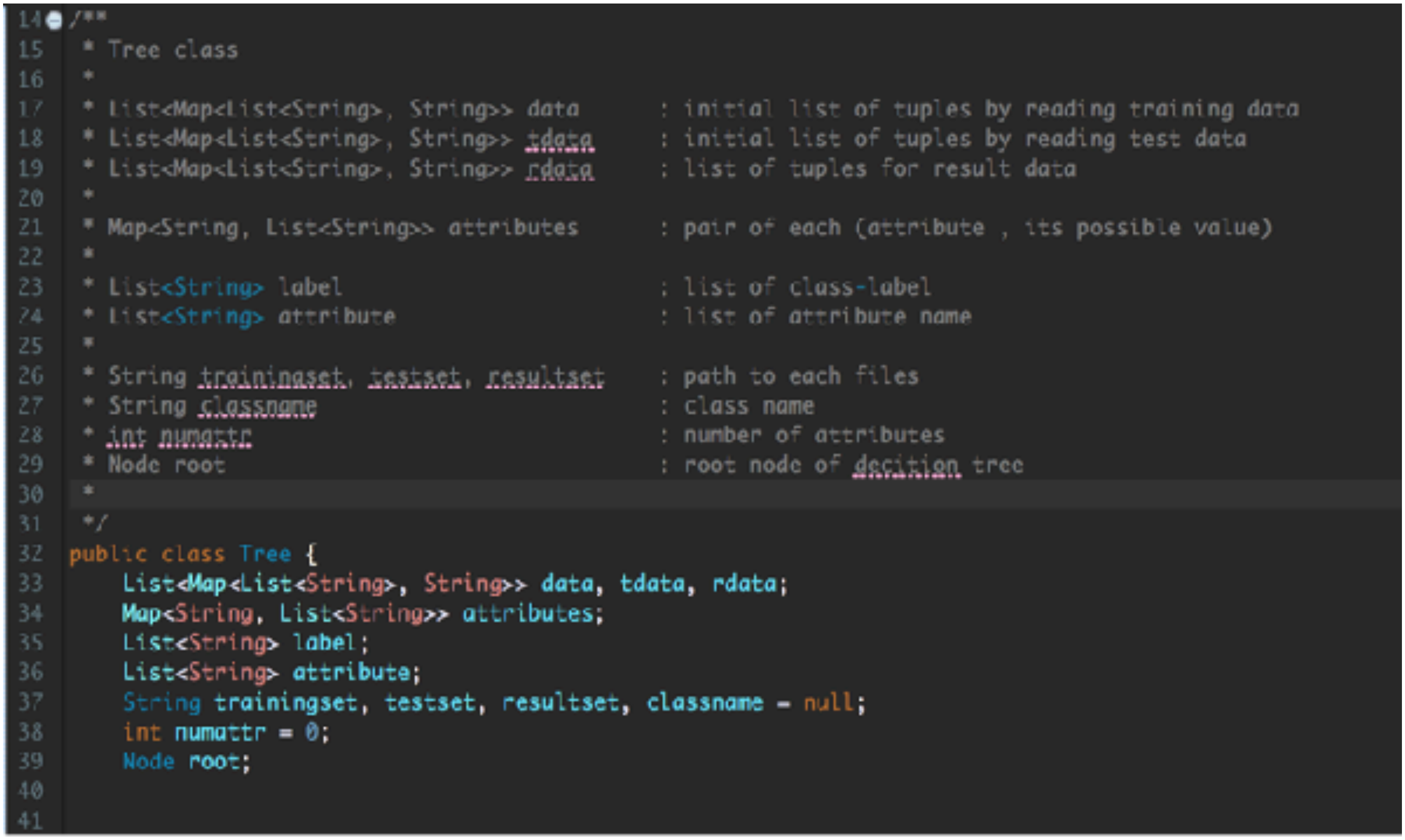
2013011800 구장회

순서

1. data structure - tree class , node class
2. summary of algorithm for crucial method
3. summary of rest algorithm
4. instruction for compile

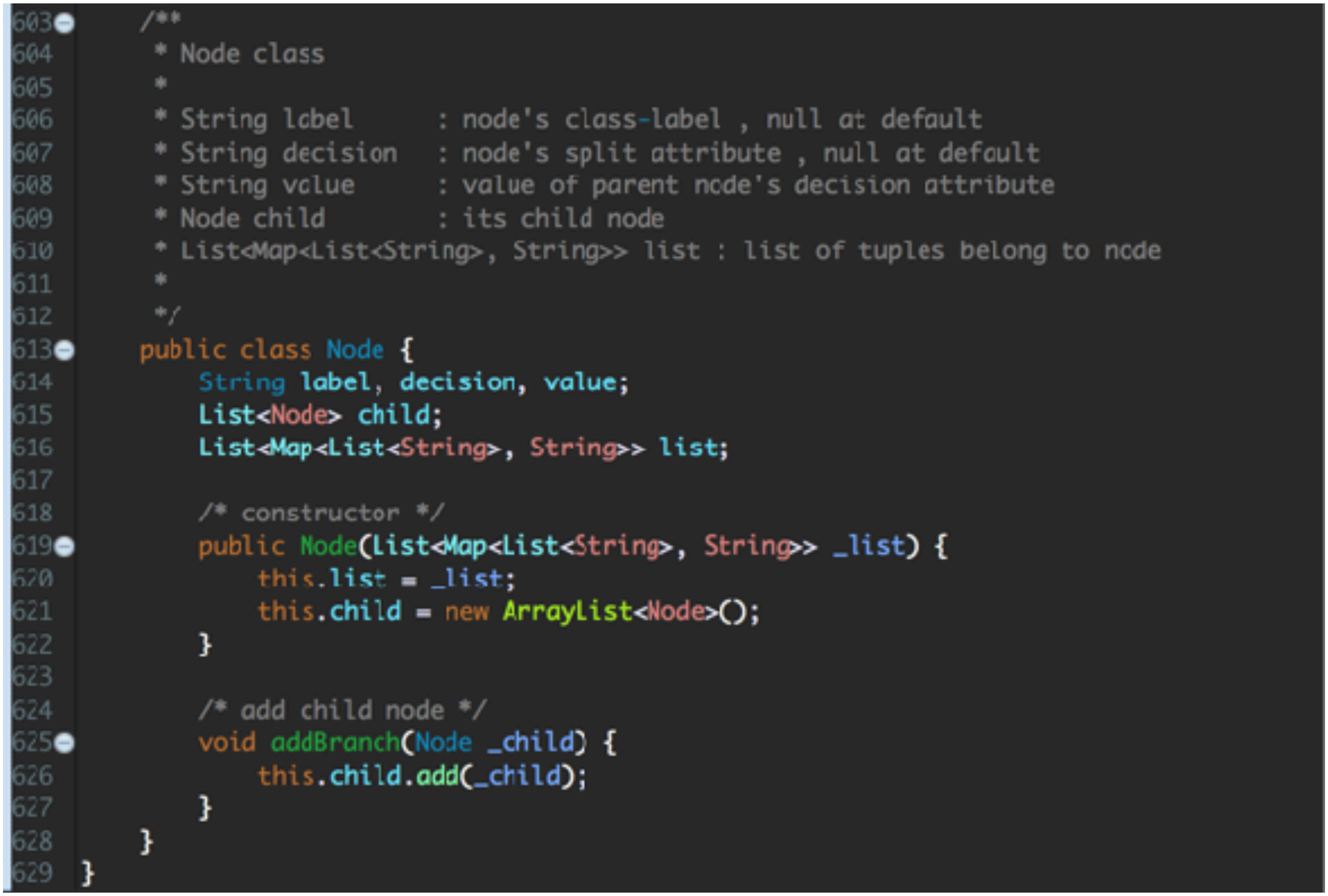
# data structure

[tree class]



```
/**
 * Tree class
 *
 * List<Map<List<String>, String>> data      : initial list of tuples by reading training data
 * List<Map<List<String>, String>> tdata     : initial list of tuples by reading test data
 * List<Map<List<String>, String>> rdata     : list of tuples for result data
 *
 * Map<String, List<String>> attributes      : pair of each (attribute , its possible value)
 *
 * List<String> label                        : list of class-label
 * List<String> attribute                    : list of attribute name
 *
 * String trainingset, testset, resultset    : path to each files
 * String classname                          : class name
 * int numattr                               : number of attributes
 * Node root                                 : root node of decition tree
 *
 */
public class Tree {
    List<Map<List<String>, String>> data, tdata, rdata;
    Map<String, List<String>> attributes;
    List<String> label;
    List<String> attribute;
    String trainingset, testset, resultset, classname = null;
    int numattr = 0;
    Node root;

```

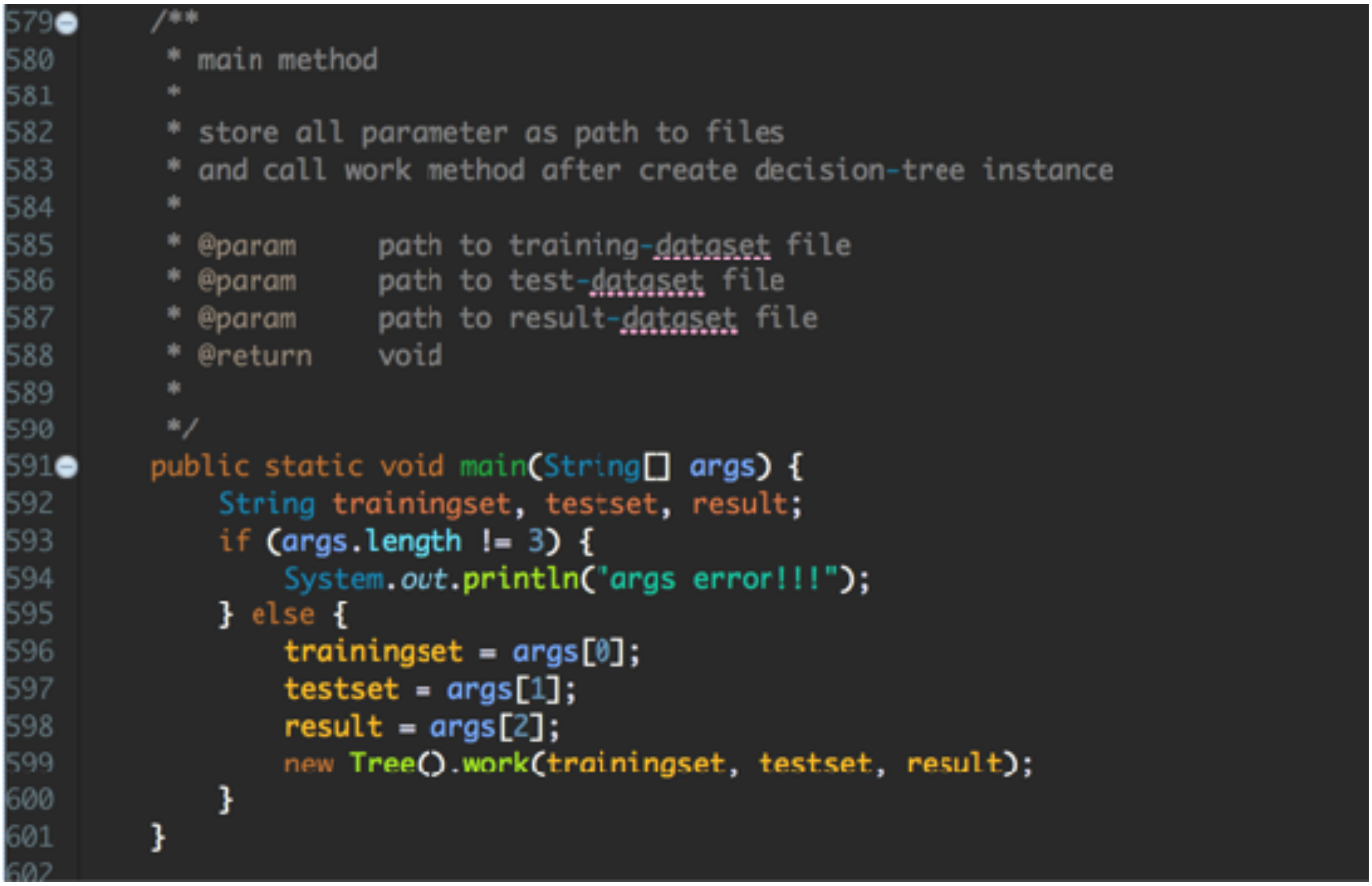
[node class]



```java
    /**
     * Node class
     *
     * String label      : node's class-label , null at default
     * String decision   : node's split attribute , null at default
     * String value      : value of parent node's decision attribute
     * Node child        : its child node
     * List<Map<List<String>, String>> list : list of tuples belong to node
     *
     */
    public class Node {
        String label, decision, value;
        List<Node> child;
        List<Map<List<String>, String>> list;

        /* constructor */
        public Node(List<Map<List<String>, String>> _list) {
            this.list = _list;
            this.child = new ArrayList<Node>();
        }

        /* add child node */
        void addBranch(Node _child) {
            this.child.add(_child);
        }
    }
}
```

# summary of algorithm for crucial method

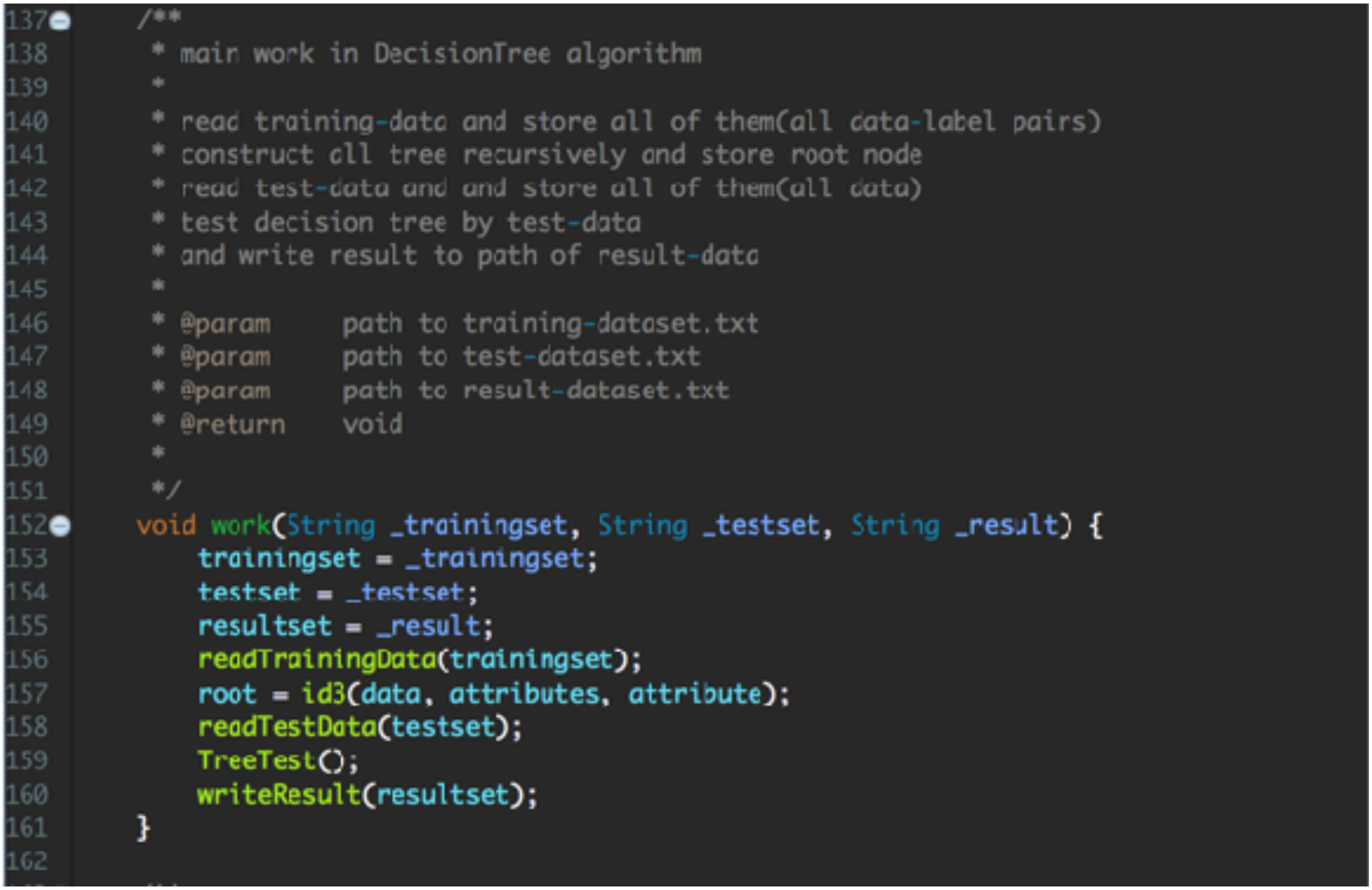
[main method]



```
    /**
     * main method
     *
     * store all parameter as path to files
     * and call work method after create decision-tree instance
     *
     * @param     path to training-dataset file
     * @param     path to test-dataset file
     * @param     path to result-dataset file
     * @return    void
     *
     */
    public static void main(String[] args) {
        String trainingset, testset, result;
        if (args.length != 3) {
            System.out.println("args error!!!");
        } else {
            trainingset = args[0];
            testset = args[1];
            result = args[2];
            new Tree().work(trainingset, testset, result);
        }
    }
```

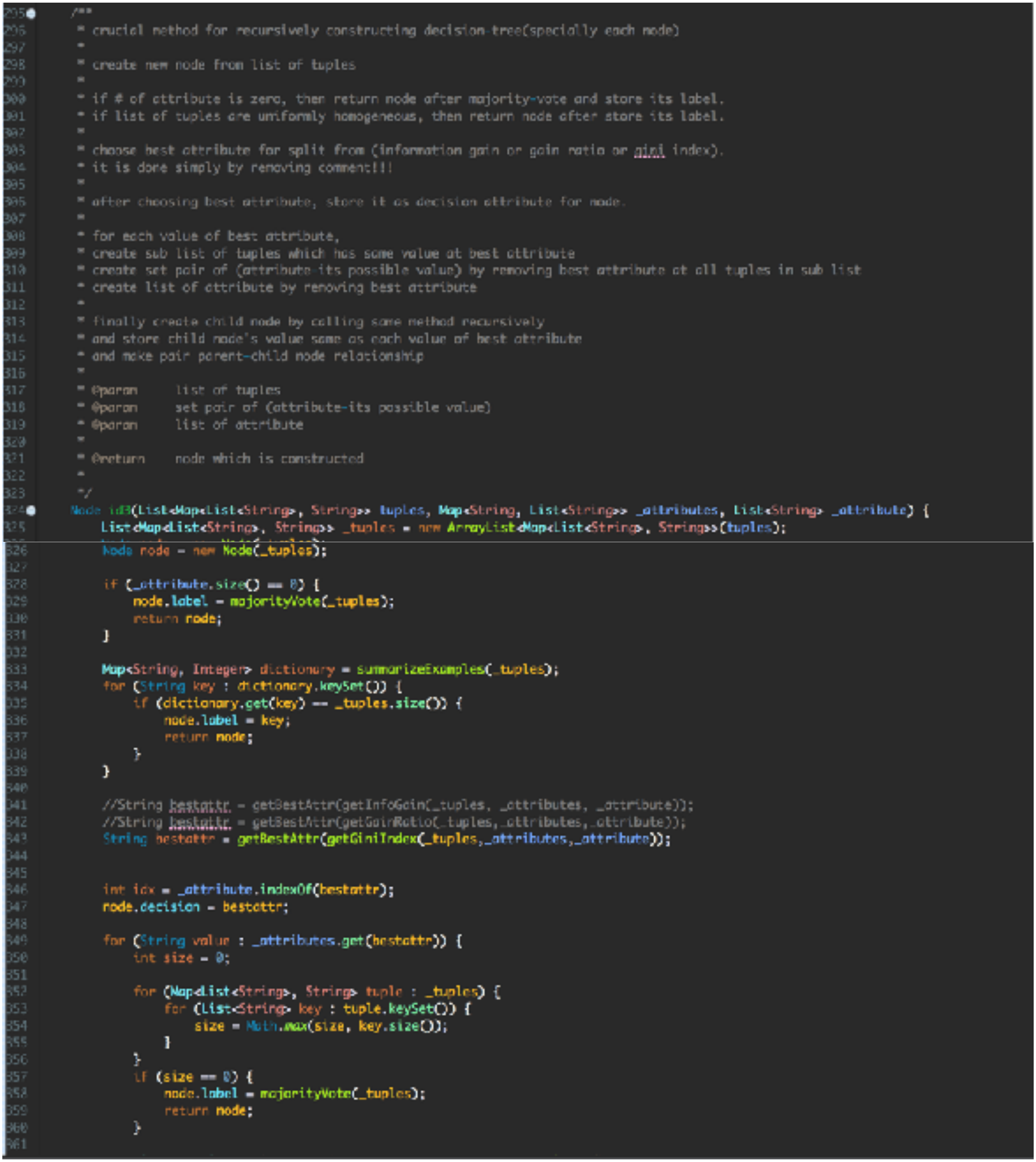
[work() in tree class]

- read training data
- construct decision tree
- read test data
- tree test from test data
- write result data

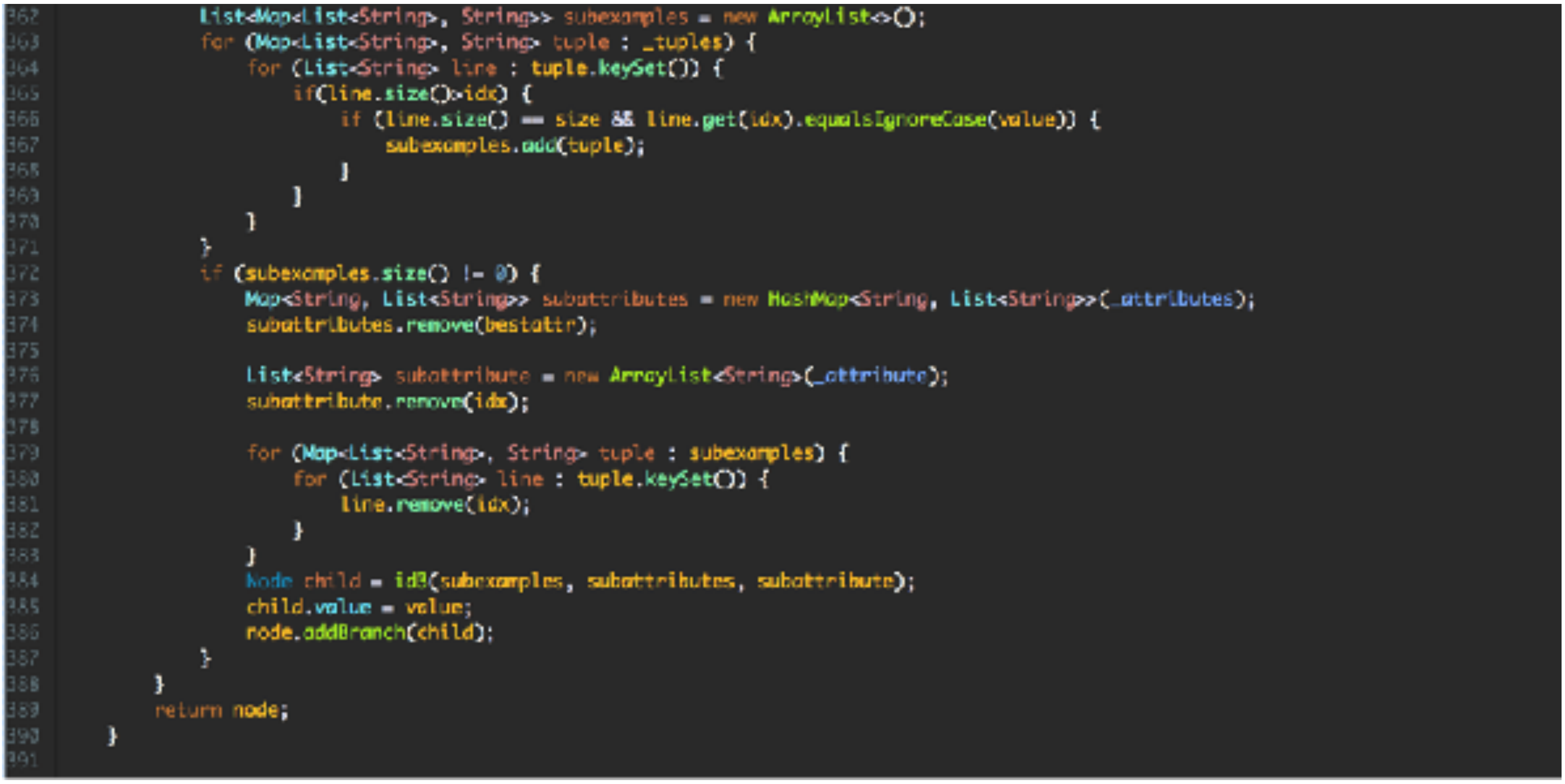


```
    /**
     * main work in DecisionTree algorithm
     *
     * read training-data and store all of them(all data-label pairs)
     * construct all tree recursively and store root node
     * read test-data and and store all of them(all data)
     * test decision tree by test-data
     * and write result to path of result-data
     *
     * @param     path to training-dataset.txt
     * @param     path to test-dataset.txt
     * @param     path to result-dataset.txt
     * @return    void
     *
     */
    void work(String _trainingset, String _testset, String _result) {
        trainingset = _trainingset;
        testset = _testset;
        resultset = _result;
        readTrainingData(trainingset);
        root = id3(data, attributes, attribute);
        readTestData(testset);
        TreeTest();
        writeResult(resultset);
    }

```

[id3() in tree class]



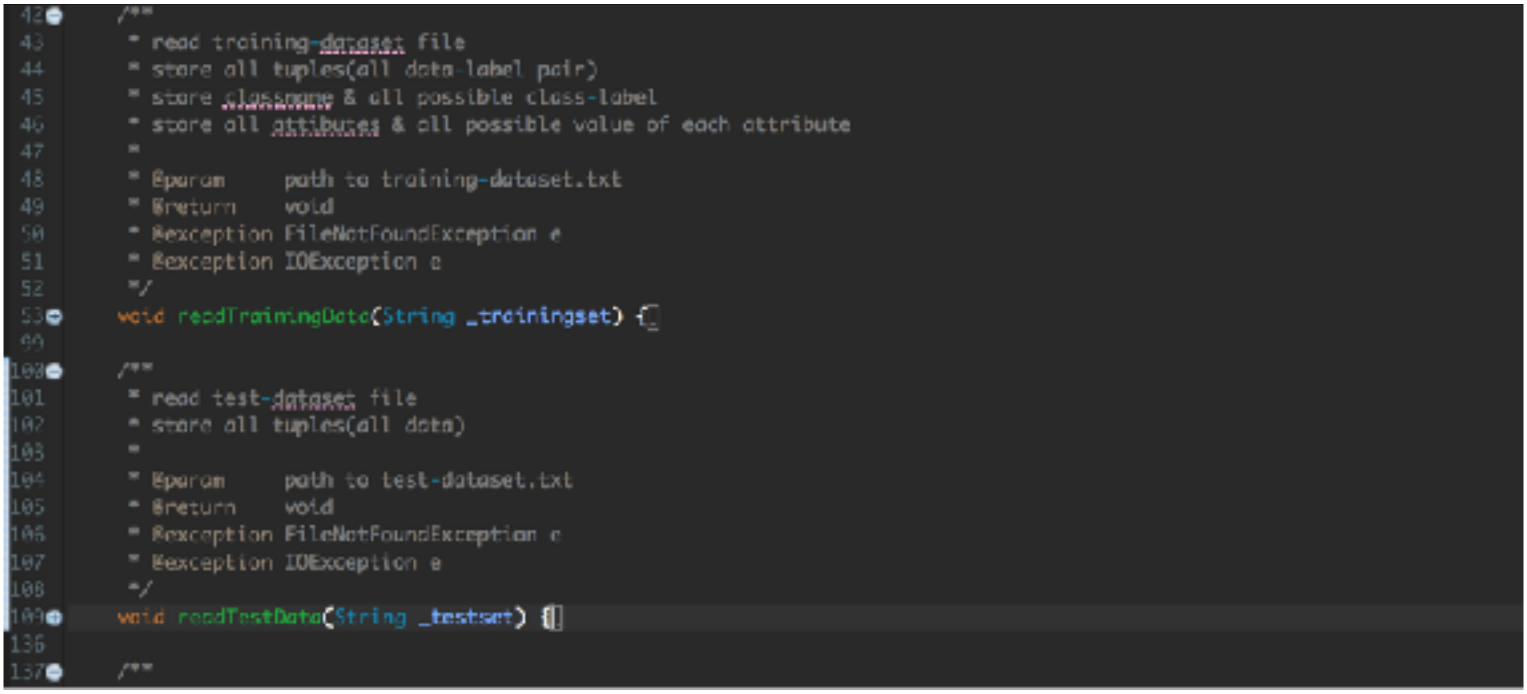
```
    /**
     * crucial method for recursively constructing decision-tree(specially each node)
     *
     * create new node from list of tuples
     *
     * if # of attribute is zero, then return node after majority-vote and store its label.
     * if list of tuples are uniformly homogeneous, then return node after store its label.
     *
     * choose best attribute for split from (information gain or gain ratio or gini index).
     * it is done simply by removing comment!!!
     *
     * after choosing best attribute, store it as decision attribute for node.
     *
     * for each value of best attribute,
     * create sub list of tuples which has same value at best attribute
     * create set pair of (attribute-its possible value) by removing best attribute at all tuples in sub list
     * create list of attribute by removing best attribute
     *
     * finally create child node by calling same method recursively
     * and store child node's value same as each value of best attribute
     * and make pair parent-child node relationship
     *
     * @param     list of tuples
     * @param     set pair of (attribute-its possible value)
     * @param     list of attribute
     *
     * @return    node which is constructed
     *
     */
    Node id3(List<Map<List<String>, String>> tuples, Map<String, List<String>> _attributes, List<String> _attribute) {
        List<Map<List<String>, String>> _tuples = new ArrayList<Map<List<String>, String>>(tuples);
        Node node = new Node(_tuples);

        if (_attribute.size() == 0) {
            node.label = majorityVote(_tuples);
            return node;
        }

        Map<String, Integer> dictionary = summarizeExamples(_tuples);
        for (String key : dictionary.keySet()) {
            if (dictionary.get(key) == _tuples.size()) {
                node.label = key;
                return node;
            }
        }

        //String bestattr = getBestAttr(getInfoGain(_tuples, _attributes, _attribute));
        //String bestattr = getBestAttr(getGainRatio(_tuples,_attributes,_attribute));
        String bestattr = getBestAttr(getGiniIndex(_tuples,_attributes,_attribute));


        int idx = _attribute.indexOf(bestattr);
        node.decision = bestattr;

        for (String value : _attributes.get(bestattr)) {
            int size = 0;

            for (Map<List<String>, String> tuple : _tuples) {
                for (List<String> key : tuple.keySet()) {
                    size = Math.max(size, key.size());
                }
            }
            if (size == 0) {
                node.label = majorityVote(_tuples);
                return node;
            }
```



```java
            List<Map<List<String>, String>> subexamples = new ArrayList<>();
            for (Map<List<String>, String> tuple : _tuples) {
                for (List<String> line : tuple.keySet()) {
                    if(line.size()>idx) {
                        if (line.size() == size && line.get(idx).equalsIgnoreCase(value)) {
                            subexamples.add(tuple);
                        }
                    }
                }
            }
            if (subexamples.size() != 0) {
                Map<String, List<String>> subattributes = new HashMap<String, List<String>>(_attributes);
                subattributes.remove(bestattr);

                List<String> subattribute = new ArrayList<String>(_attribute);
                subattribute.remove(idx);

                for (Map<List<String>, String> tuple : subexamples) {
                    for (List<String> line : tuple.keySet()) {
                        line.remove(idx);
                    }
                }
                Node child = id3(subexamples, subattributes, subattribute);
                child.value = value;
                node.addBranch(child);
            }
        }
        return node;
    }
```

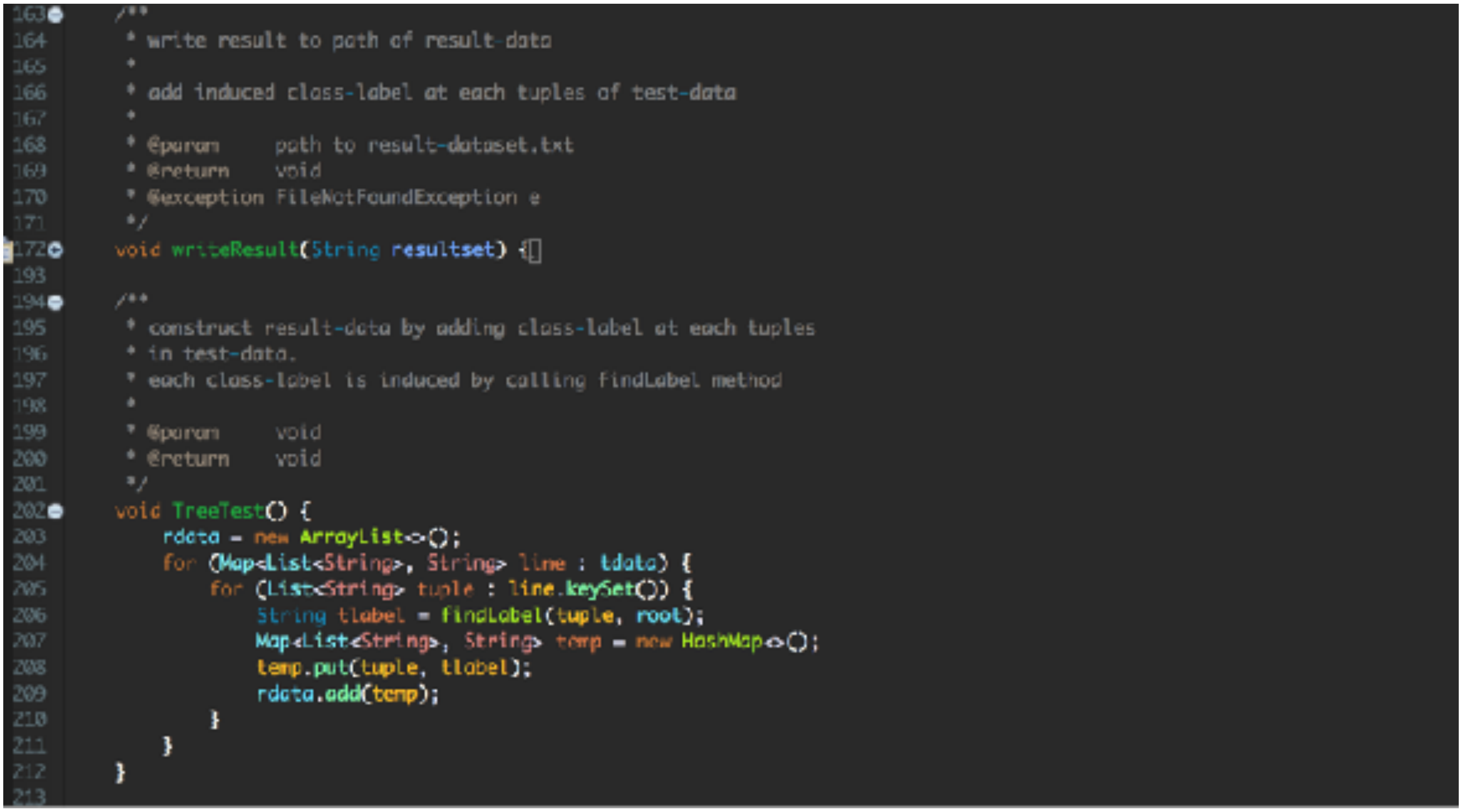
# summary of rest algorithm

[readTrainingData & readTestData]



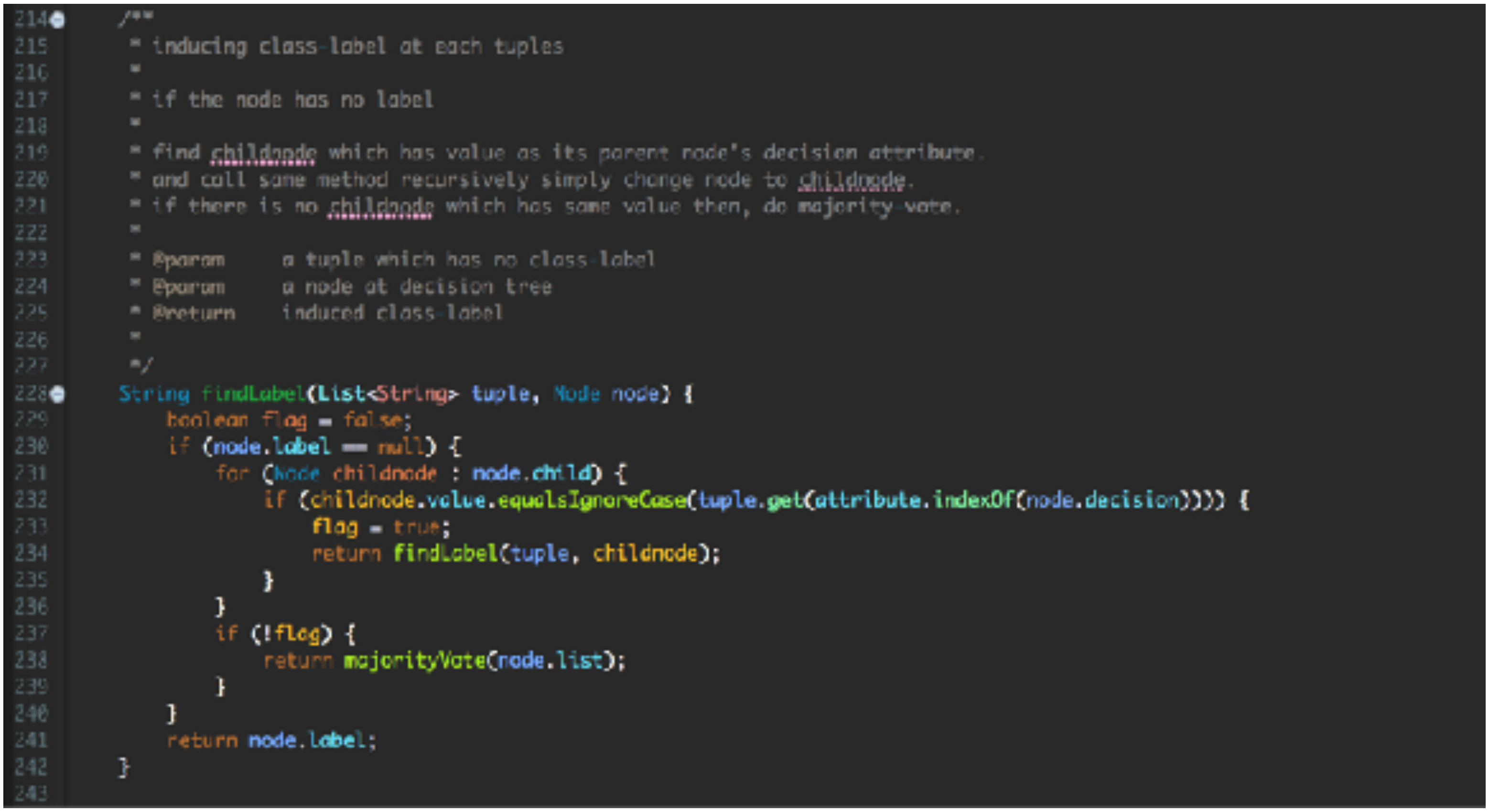
```
    /**
     * read training-dataset file
     * store all tuples(all data-label pair)
     * store classname & all possible class-label
     * store all attibutes & all possible value of each attribute
     *
     * @param     path to training-dataset.txt
     * @return    void
     * @exception FileNotFoundException e
     * @exception IOException e
     */
    void readTrainingData(String _trainingset) {

    /**
     * read test-dataset file
     * store all tuples(all data)
     *
     * @param     path to test-dataset.txt
     * @return    void
     * @exception FileNotFoundException e
     * @exception IOException e
     */
    void readTestData(String _testset) {

    /**
```

[writeResult & TreeTest]



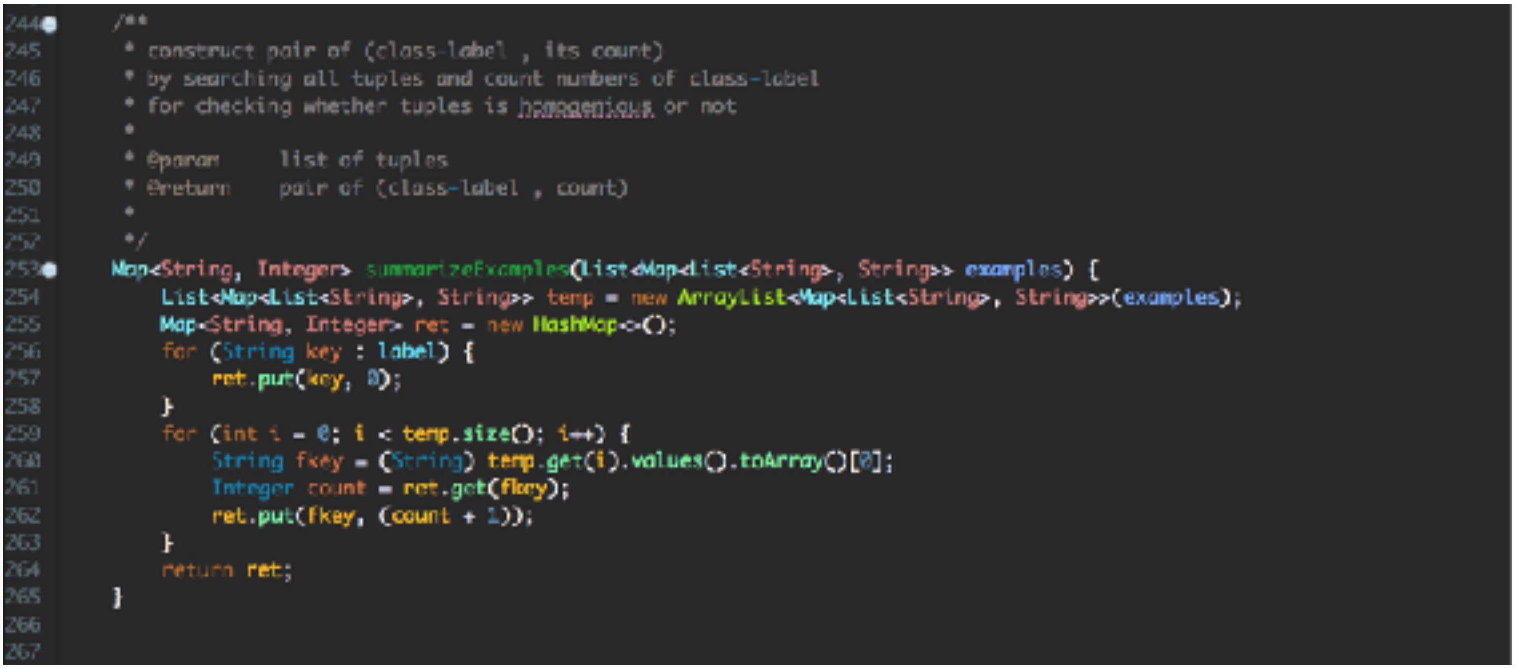
```
/**
 * write result to path of result-data
 *
 * add induced class-label at each tuples of test-data
 *
 * @param     path to result-dataset.txt
 * @return    void
 * @exception FileNotFoundException e
 */
void writeResult(String resultset) {[]

/**
 * construct result-data by adding class-label at each tuples
 * in test-data.
 * each class-label is induced by calling findLabel method
 *
 * @param     void
 * @return    void
 */
void TreeTest() {
    rdata = new ArrayList<>();
    for (Map<List<String>, String> line : tdata) {
        for (List<String> tuple : line.keySet()) {
            String tlabel = findLabel(tuple, root);
            Map<List<String>, String> temp = new HashMap<>();
            temp.put(tuple, tlabel);
            rdata.add(temp);
        }
    }
}
```

[findLabel]



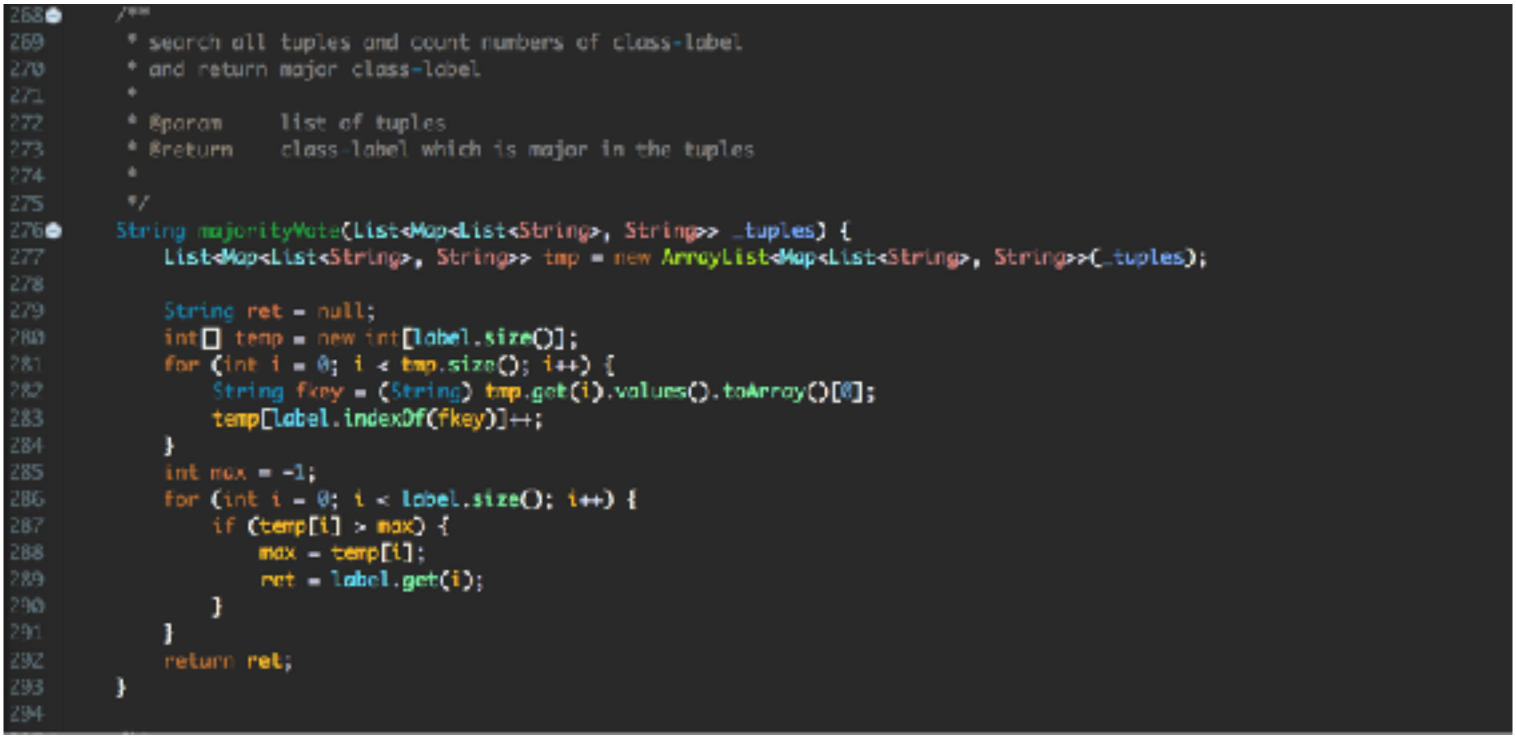
```java
/**
 * inducing class-label at each tuples
 *
 * if the node has no label
 *
 * find childnode which has value as its parent node's decision attribute.
 * and call same method recursively simply change node to childnode.
 * if there is no childnode which has same value then, do majority-vote.
 *
 * @param     a tuple which has no class-label
 * @param     a node at decision tree
 * @return    induced class-label
 *
 */
String findLabel(List<String> tuple, Node node) {
    boolean flag = false;
    if (node.label == null) {
        for (Node childnode : node.child) {
            if (childnode.value.equalsIgnoreCase(tuple.get(attribute.indexOf(node.decision)))) {
                flag = true;
                return findLabel(tuple, childnode);
            }
        }
        if (!flag) {
            return majorityVote(node.list);
        }
    }
    return node.label;
}
```

[summarizeExamples]



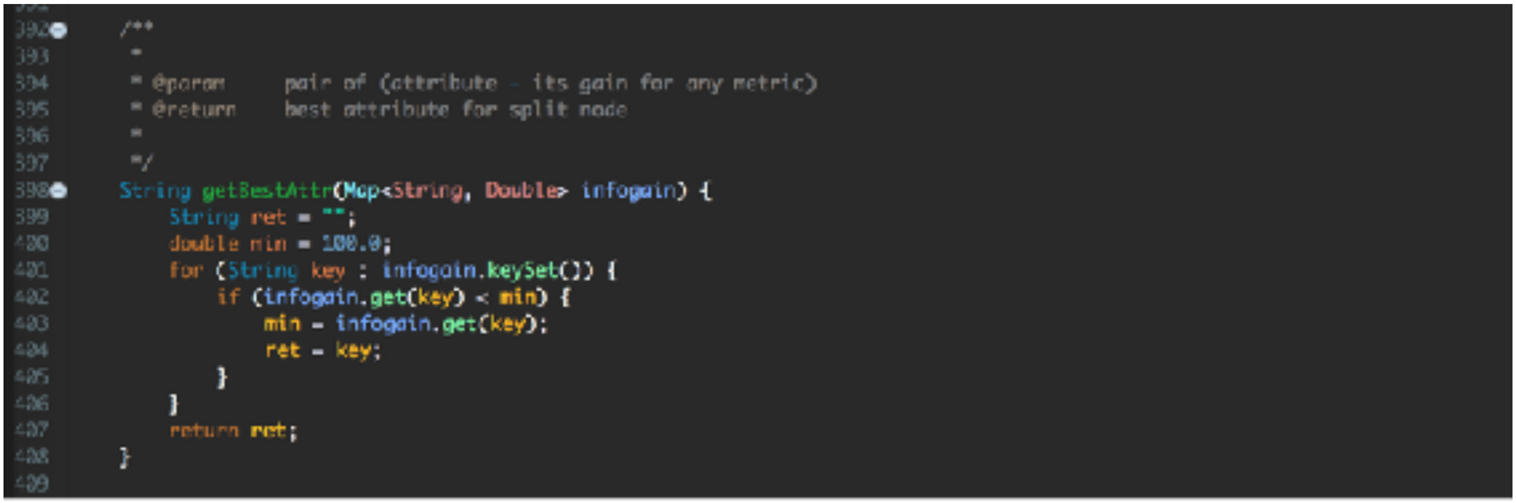
```java
    /**
     * construct pair of (class-label , its count)
     * by searching all tuples and count numbers of class-label
     * for checking whether tuples is homogenious or not
     *
     * @param     list of tuples
     * @return    pair of (class-label , count)
     *
     */
    Map<String, Integer> summarizeExamples(List<Map<List<String>, String>> examples) {
        List<Map<List<String>, String>> temp = new ArrayList<Map<List<String>, String>>(examples);
        Map<String, Integer> ret = new HashMap<>();
        for (String key : label) {
            ret.put(key, 0);
        }
        for (int i = 0; i < temp.size(); i++) {
            String fkey = (String) temp.get(i).values().toArray()[0];
            Integer count = ret.get(fkey);
            ret.put(fkey, (count + 1));
        }
        return ret;
    }
```

[majority-vote]



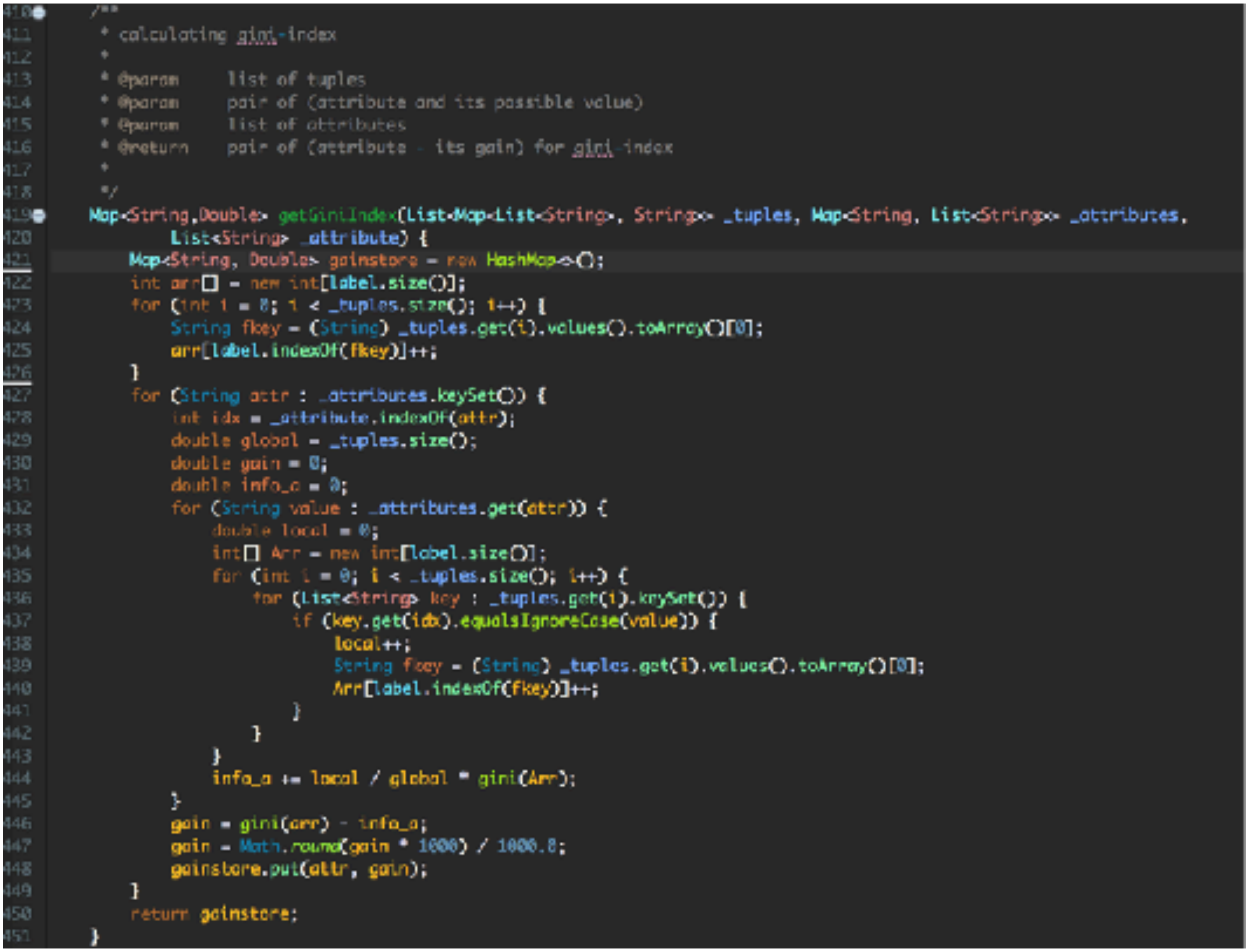
```
/**
 * search all tuples and count numbers of class-label
 * and return major class-label
 *
 * @param     list of tuples
 * @return    class-label which is major in the tuples
 *
 */
String majorityVote(List<Map<List<String>, String>> _tuples) {
    List<Map<List<String>, String>> tmp = new ArrayList<Map<List<String>, String>>(_tuples);

    String ret = null;
    int[] temp = new int[label.size()];
    for (int i = 0; i < tmp.size(); i++) {
        String fkey = (String) tmp.get(i).values().toArray()[0];
        temp[label.indexOf(fkey)]++;
    }
    int max = -1;
    for (int i = 0; i < label.size(); i++) {
        if (temp[i] > max) {
            max = temp[i];
            ret = label.get(i);
        }
    }
    return ret;
}
```

[getBestAttr]



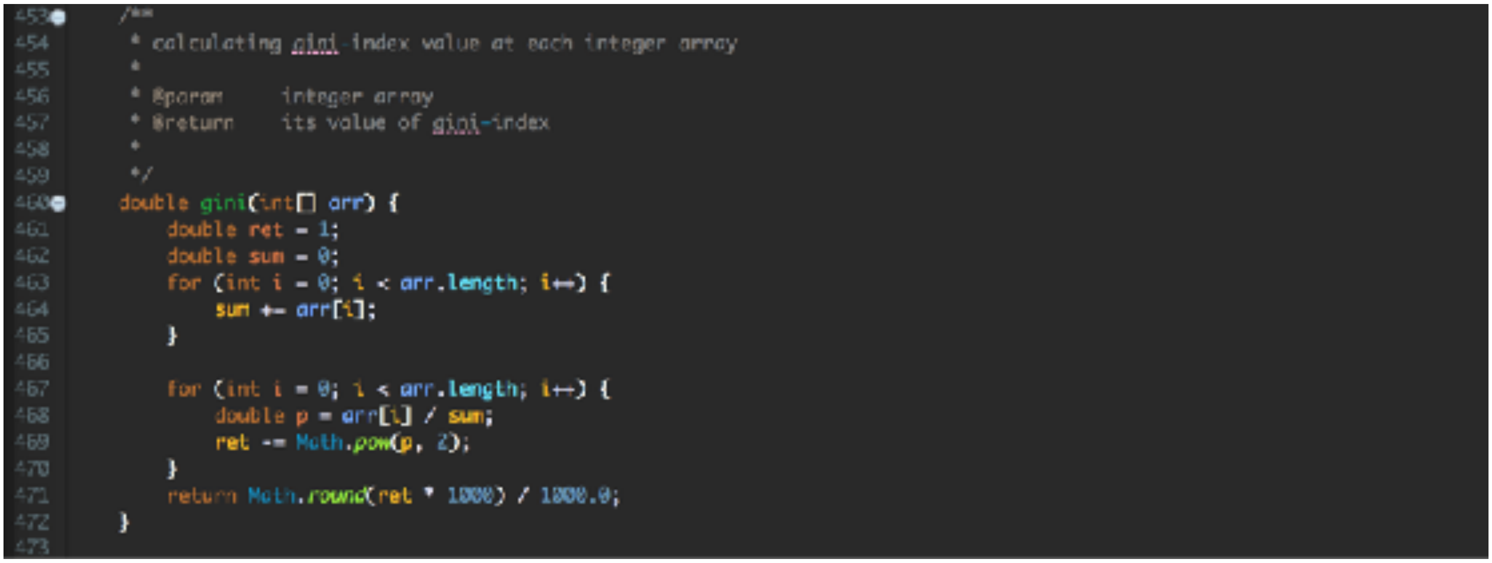
```
/**
 *
 * @param     pair of (attribute - its gain for any metric)
 * @return    best attribute for split node
 *
 */
String getBestAttr(Map<String, Double> infogain) {
    String ret = "";
    double min = 100.0;
    for (String key : infogain.keySet()) {
        if (infogain.get(key) < min) {
            min = infogain.get(key);
            ret = key;
        }
    }
    return ret;
}
```

[getGiniIndex]



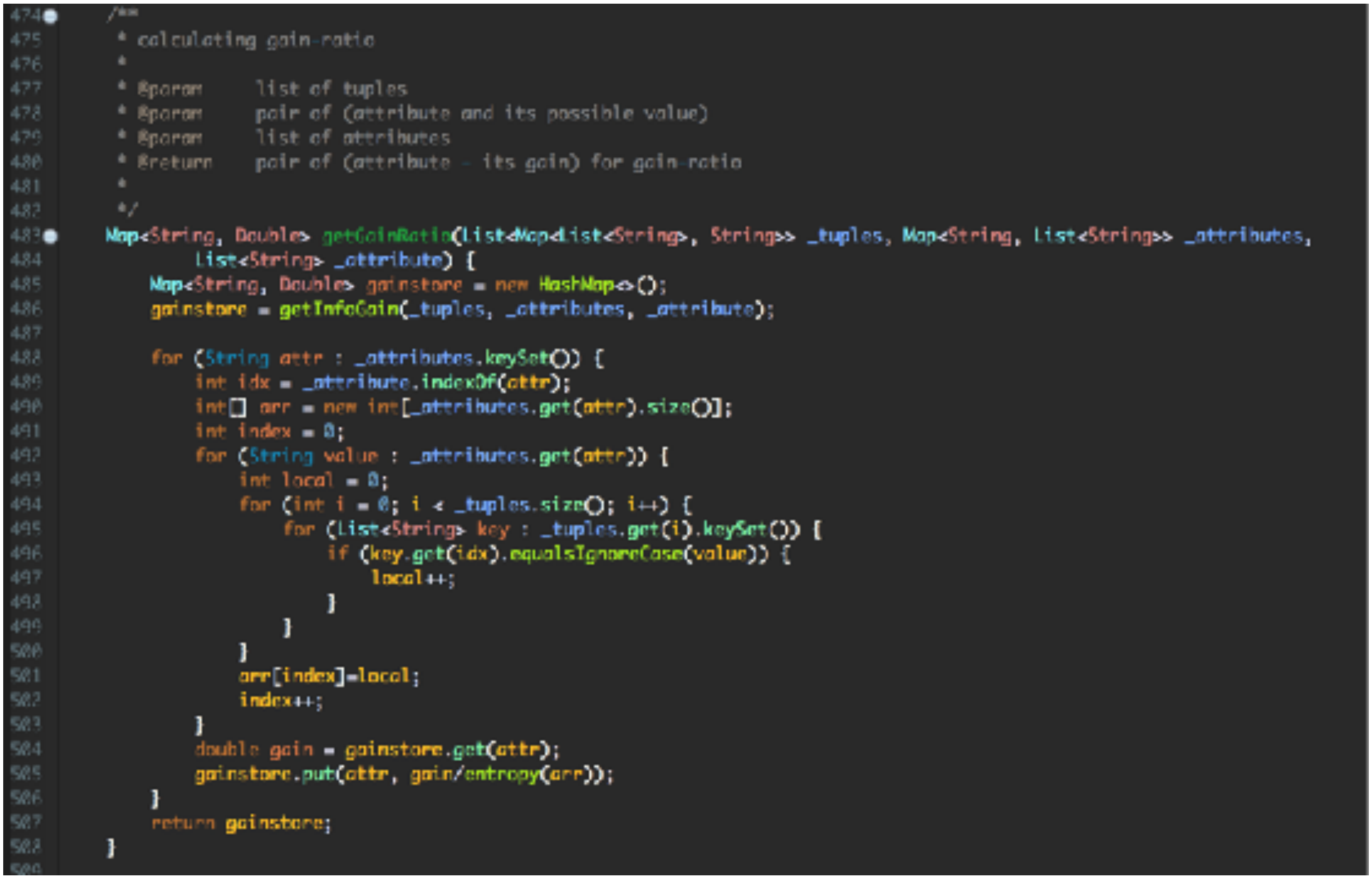
```
    /**
     * calculating gini-index
     *
     * @param     list of tuples
     * @param     pair of (attribute and its possible value)
     * @param     list of attributes
     * @return    pair of (attribute - its gain) for gini-index
     *
     */
    Map<String,Double> getGiniIndex(List<Map<List<String>, String>> _tuples, Map<String, List<String>> _attributes,
            List<String> _attribute) {
        Map<String, Double> gainstore = new HashMap<>();
        int arr[] = new int[label.size()];
        for (int i = 0; i < _tuples.size(); i++) {
            String fkey = (String) _tuples.get(i).values().toArray()[0];
            arr[label.indexOf(fkey)]++;
        }
        for (String attr : _attributes.keySet()) {
            int idx = _attribute.indexOf(attr);
            double global = _tuples.size();
            double gain = 0;
            double info_a = 0;
            for (String value : _attributes.get(attr)) {
                double local = 0;
                int[] Arr = new int[label.size()];
                for (int i = 0; i < _tuples.size(); i++) {
                    for (List<String> key : _tuples.get(i).keySet()) {
                        if (key.get(idx).equalsIgnoreCase(value)) {
                            local++;
                            String fkey = (String) _tuples.get(i).values().toArray()[0];
                            Arr[label.indexOf(fkey)]++;
                        }
                    }
                }
                info_a += local / global * gini(Arr);
            }
            gain = gini(arr) - info_a;
            gain = Math.round(gain * 1000) / 1000.0;
            gainstore.put(attr, gain);
        }
        return gainstore;
    }
```

[gini]



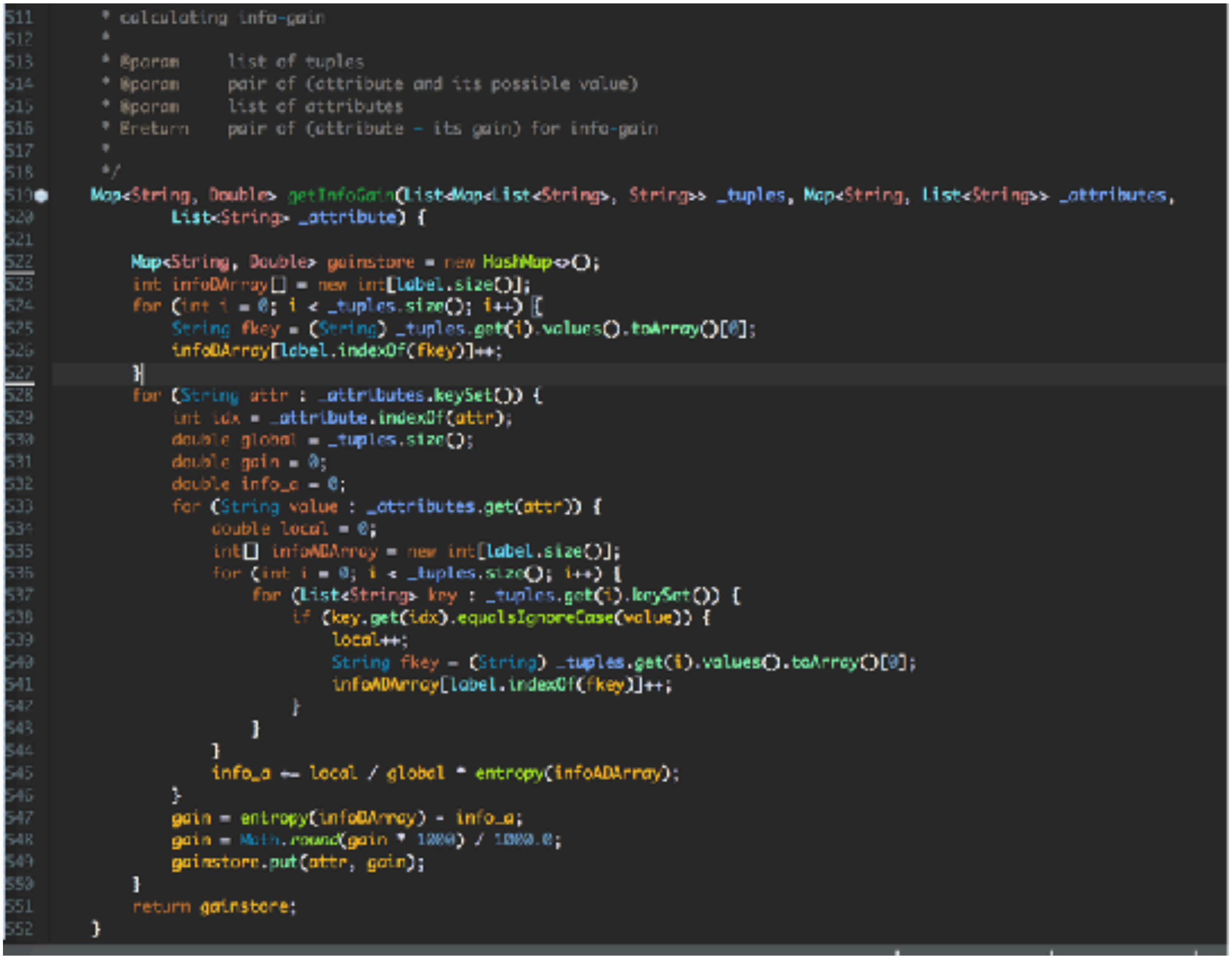
```java
    /**
     * calculating gini-index value at each integer array
     *
     * @param     integer array
     * @return    its value of gini-index
     *
     */
    double gini(int[] arr) {
        double ret = 1;
        double sum = 0;
        for (int i = 0; i < arr.length; i++) {
            sum += arr[i];
        }

        for (int i = 0; i < arr.length; i++) {
            double p = arr[i] / sum;
            ret -= Math.pow(p, 2);
        }
        return Math.round(ret * 1000) / 1000.0;
    }
```

[getGainRatio]



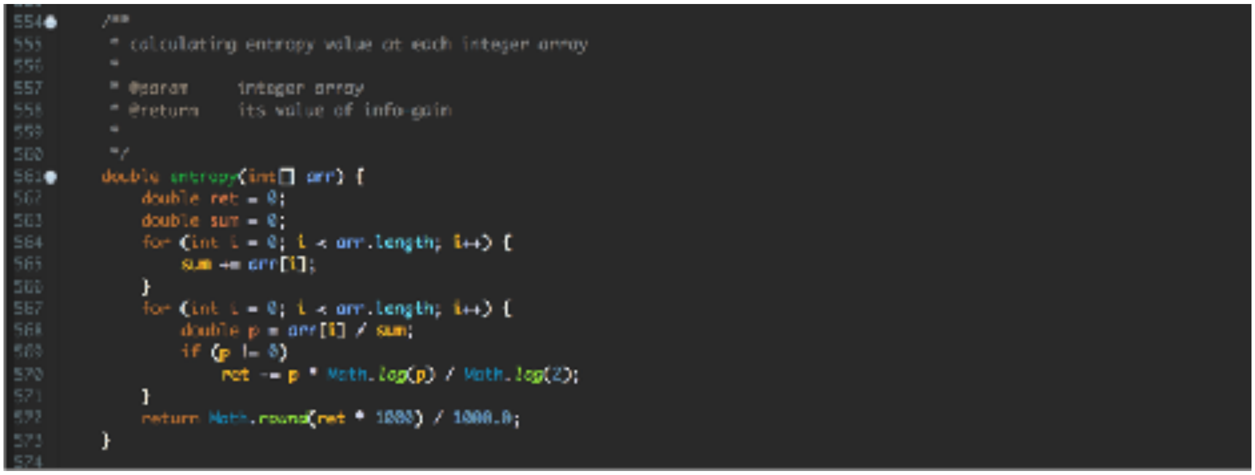
```java
    /**
     * calculating gain-ratio
     *
     * @param     list of tuples
     * @param     pair of (attribute and its possible value)
     * @param     list of attributes
     * @return    pair of (attribute - its gain) for gain-ratio
     *
     */
    Map<String, Double> getGainRatio(List<Map<List<String>, String>> _tuples, Map<String, List<String>> _attributes,
            List<String> _attribute) {
        Map<String, Double> gainstore = new HashMap<>();
        gainstore = getInfoGain(_tuples, _attributes, _attribute);

        for (String attr : _attributes.keySet()) {
            int idx = _attribute.indexOf(attr);
            int[] arr = new int[_attributes.get(attr).size()];
            int index = 0;
            for (String value : _attributes.get(attr)) {
                int local = 0;
                for (int i = 0; i < _tuples.size(); i++) {
                    for (List<String> key : _tuples.get(i).keySet()) {
                        if (key.get(idx).equalsIgnoreCase(value)) {
                            local++;
                        }
                    }
                }
                arr[index]=local;
                index++;
            }
            double gain = gainstore.get(attr);
            gainstore.put(attr, gain/entropy(arr));
        }
        return gainstore;
    }
```

[getInfoGain]



```
     * calculating info-gain
     *
     * @param     list of tuples
     * @param     pair of (attribute and its possible value)
     * @param     list of attributes
     * @return    pair of (attribute - its gain) for info-gain
     *
     */
    Map<String, Double> getInfoGain(List<Map<List<String>, String>> _tuples, Map<String, List<String>> _attributes,
            List<String> _attribute) {

        Map<String, Double> gainstore = new HashMap<>();
        int infoDArray[] = new int[label.size()];
        for (int i = 0; i < _tuples.size(); i++) {
            String fkey = (String) _tuples.get(i).values().toArray()[0];
            infoDArray[label.indexOf(fkey)]++;
        }
        for (String attr : _attributes.keySet()) {
            int idx = _attribute.indexOf(attr);
            double global = _tuples.size();
            double gain = 0;
            double info_a = 0;
            for (String value : _attributes.get(attr)) {
                double local = 0;
                int[] infoADArray = new int[label.size()];
                for (int i = 0; i < _tuples.size(); i++) {
                    for (List<String> key : _tuples.get(i).keySet()) {
                        if (key.get(idx).equalsIgnoreCase(value)) {
                            local++;
                            String fkey = (String) _tuples.get(i).values().toArray()[0];
                            infoADArray[label.indexOf(fkey)]++;
                        }
                    }
                }
                info_a += local / global * entropy(infoADArray);
            }
            gain = entropy(infoDArray) - info_a;
            gain = Math.round(gain * 1000) / 1000.0;
            gainstore.put(attr, gain);
        }
        return gainstore;
    }
```

[entropy]



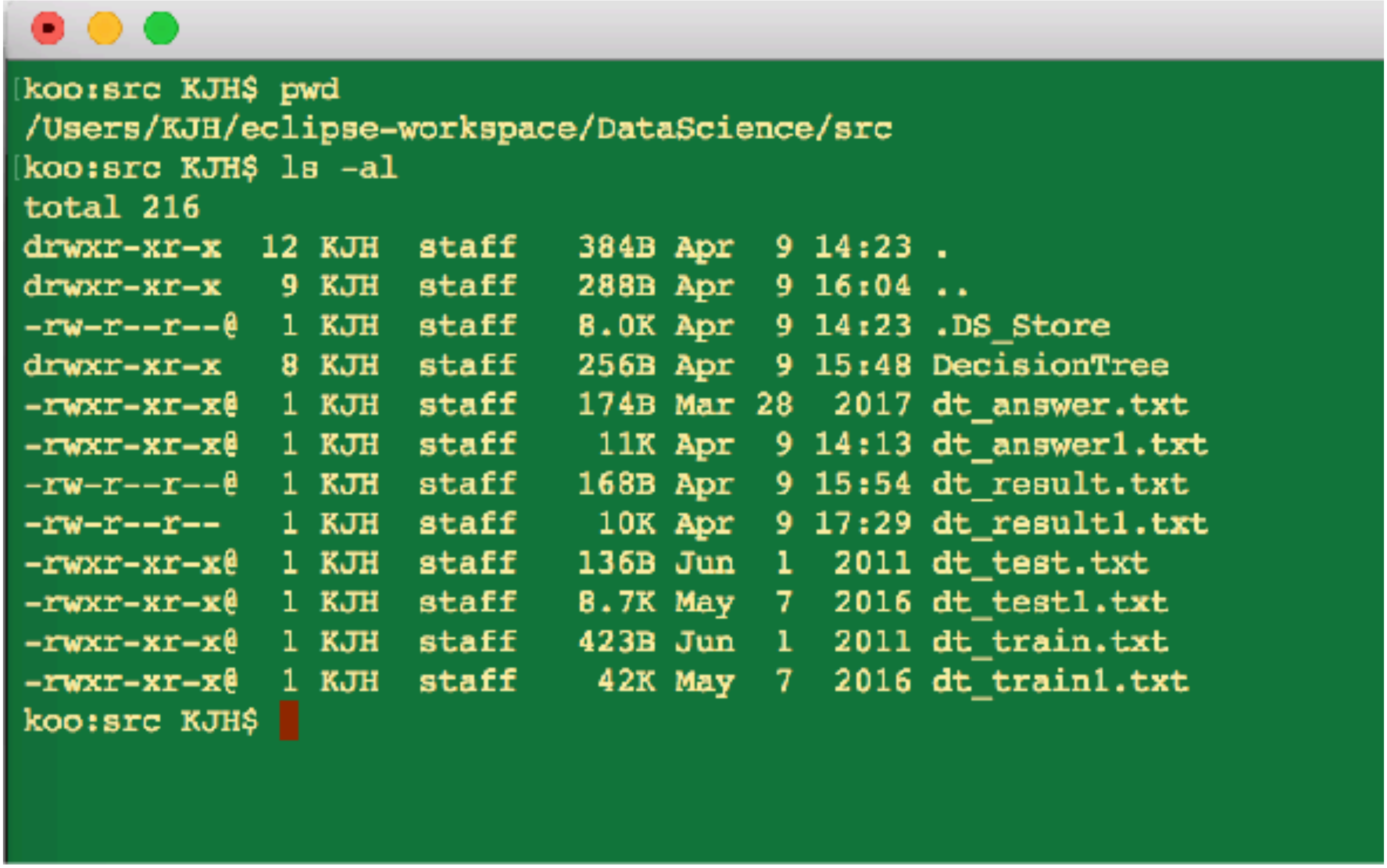
```
/**
 * calculating entropy value at each integer array
 *
 * @param      integer array
 * @return     its value of info-gain
 *
 */
double entropy(int[] arr) {
    double ret = 0;
    double sum = 0;
    for (int i = 0; i < arr.length; i++) {
        sum += arr[i];
    }
    for (int i = 0; i < arr.length; i++) {
        double p = arr[i] / sum;
        if (p != 0)
            ret -= p * Math.log(p) / Math.log(2);
    }
    return Math.round(ret * 1000) / 1000.0;
}
```

# instruction for complile

[Enviorment]

- OS : Mac OS
- Language : Java

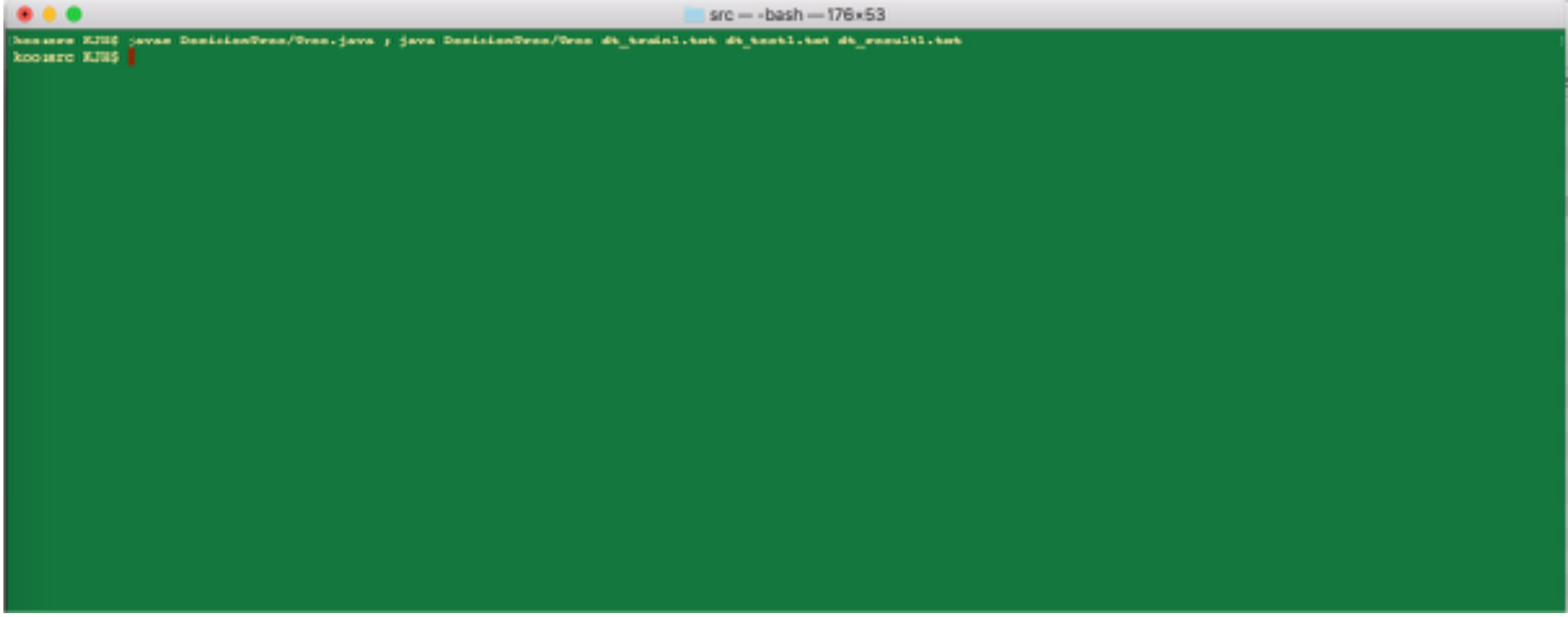
[Screenshot - 실행 전]



```
[koo:src KJH$ pwd
/Users/KJH/eclipse-workspace/DataScience/src
[koo:src KJH$ ls -al
total 216
drwxr-xr-x  12 KJH  staff   384B Apr  9 14:23 .
drwxr-xr-x   9 KJH  staff   288B Apr  9 16:04 ..
-rw-r--r--@  1 KJH  staff   8.0K Apr  9 14:23 .DS_Store
drwxr-xr-x   8 KJH  staff   256B Apr  9 15:48 DecisionTree
-rwxr-xr-x@  1 KJH  staff   174B Mar 28  2017 dt_answer.txt
-rwxr-xr-x@  1 KJH  staff    11K Apr  9 14:13 dt_answer1.txt
-rw-r--r--@  1 KJH  staff   168B Apr  9 15:54 dt_result.txt
-rw-r--r--   1 KJH  staff    10K Apr  9 17:29 dt_result1.txt
-rwxr-xr-x@  1 KJH  staff   136B Jun  1  2011 dt_test.txt
-rwxr-xr-x@  1 KJH  staff   8.7K May  7  2016 dt_test1.txt
-rwxr-xr-x@  1 KJH  staff   423B Jun  1  2011 dt_train.txt
-rwxr-xr-x@  1 KJH  staff    42K May  7  2016 dt_train1.txt
koo:src KJH$ 
```

[Screenshot - 실행 방법]

compile: $javac DecisionTree/Tree.java
execute : $java DecisionTree/Tree dt_train1.txt dt_test1.tst dt_result1.txt



```
koo:src KJH$ javac DecisionTree/Tree.java ; java DecisionTree/Tree dt_train1.txt dt_test1.txt dt_result1.txt
koo:src KJH$
```