CST Part IA: OOP, SV 3
Joe Yan
2016-11-21

1 Exercise 40

```
public class MarksBoard {
        private Map<String, Float> mMapName;
        // hash fast to add new element but slow when doing sorting.
        // tree slow to add new element but fast to request sorted list.

        public MarksBoard() {
                mMapName = new HashMap<>();
        }

        public boolean addStudent(String name, float score) {
                if (mMapName.containsKey(name))
                        return false;
                else {
                        mMapName.put(name, score);
                        return true;
                }
        }

        public List<String> allStudents() {
                List<String> students = new ArrayList<String>(mMapName.keySet());
                Collections.sort(students);
                return students;
        }

        public float getPercentageBoundary(float p) {
                if (p <= 0f)
                        return 100;
                if (p > 100f)
                        return 0;
                float size = mMapName.size();
                int nth = (int) Math.ceil(size * p / 100);
                ArrayList<Float> scoreList = new ArrayList<>(mMapName.values());
                Collections.sort(scoreList);
                return scoreList.get((int) size - nth);
        }

        public List<String> getTopPercentageStudents(float p) {
                float n = mMapName.size();
                float boundary = getPercentageBoundary(p);
                ArrayList<String> nameList = new ArrayList<>();
                for (java.util.Map.Entry<String, Float> pair : mMapName.entrySet()) {
                        if (pair.getValue() >= boundary)
                                nameList.add(pair.getKey());
                }
                Collections.sort(nameList);
```

```
                return nameList;
        }

        public float getMedianMark() {
                int size = mMapName.size();
                int pos = size / 2;
                ArrayList<Float> scoreList = new ArrayList<>(mMapName.values());
                Collections.sort(scoreList);
                if (size % 2 == 1) {
                        return scoreList.get(pos);
                } else {
                        return (scoreList.get(pos) + scoreList.get(pos - 1)) / 2;
                }
        }
}
```

2 Exercise 47

```
public class IntPair implements Comparable<IntPair> {
        private int first;
        private int second;

        public IntPair(int a, int b) {
                first = a;
                second = b;
        }

        @Override
        public int compareTo(IntPair o) {
                if (this.first > o.first) {
                        return 1;
                } else if (this.first < o.first) {
                        return -1;
                } else {
                        if (this.second > o.second) {
                                return 1;
                        } else if (this.second < o.second) {
                                return -1;
                        } else {
                                return 0;
                        }
                }
        }

        public int first() {
                return first;
        }

        public int second() {
                return second;
        }
}
```

```
public class ReadAndSort {
        public static void main(String args[]) throws Exception {
                try {
                        Reader r = new FileReader
                        ("D:/Java_Workspace/OOPSV/src/S47B/test.txt");
                        BufferedReader b = new BufferedReader(r);
                        String firstLine = b.readLine();
                        String secondLine = b.readLine();
                        String[] firsts = firstLine.split(",");
                        String[] seconds = secondLine.split(",");
                        if (firsts.length != seconds.length)
                                throw new Exception("The data is not in pair!");
                        List<IntPair> data = new LinkedList<IntPair>();
                        for (int i = 0; i != firsts.length; ++i) {
                                data.add(new IntPair(Integer.parseInt(firsts[i]),
                                Integer.parseInt(seconds[i])));
                        }
                        Collections.sort(data);
                        for (IntPair pair : data) {
                                System.out.print(pair.first() + "  ");
                                System.out.println(pair.second());
                        }
                        b.close();
                } catch (FileNotFoundException e) {
                        System.out.println("File not found!");
                        e.printStackTrace();
                } catch (IOException e) {
                        System.out.println("Exception happens when reading file.");
                        e.printStackTrace();
                } catch (NumberFormatException e) {
                        System.out.println("Parsing to int failed.");
                }
        }
}
```

3 Exercise 49

- The main purpose of the state pattern design is for several states which extends or implements the same base class are able to switch to each other swiftly. If I use a abstract base class and subclasses simply inherit from it the design will lose the flexibility in the state changing, I have to write something like

```
Academic someone = new lecturer();
Academic someone = (professor) someone; // I guess complier will accept this?
```

  which is clumsy.

- Also the trivial design is bad for encapsulation. Instead of using a context class accepting the request of function and change of state from package user, a abstract base class will explode the inner design to the user which is not expected.