INFO1105/1905/9105 2017 Semester 2, Assignment 1

September 8, 2017

Submission details

- Due: Friday 22 September 2017, at 5pm
- Submit your **report** on eLearning (through the Turnitin link). The report must be in pdf format, and cannot be handwritten. Double check that your report has actually been submitted successfully (you need to get an email receipt from the Turnitin system)! **Note that the instructions and marking scheme for the report are different for students enrolled in the postgraduate unit INFO9105, compared to the scheme for undergraduates in INFO1105/1905.**
- Submit your **source code** on Ed, *including* any JUnit tests you make.
- Also, submit a tar file of your source code including tests, via the link on eLearning. This is available to us as an offical record of your work.
- The policy for late submissions is described on slides 23–24 of Lecture 1a.
- This is an **individual assignment**, so your code and your report should be entirely your own work. We recommend that you use data structures that from the JCF (i.e. java.util.*), but you can implement your own if you prefer. If you re-use any code from the textbook, this must be acknowledged, just like any other sources. **We will be using similarity matching software, on both the code and the report. Please do not plagiarize; make sure that you follow the University policy on academic honesty.**

1 Specification of the coding

1.1 Description

You will write the code for a class `Assignment` which implements the `SubmissionHistory` interface. This interface describes a system for keeping track of student assignment submissions.

1.2 The `SubmissionHistory` interface

`SubmissionHistory` objects allow you to keep track of `Submission` objects. The `SubmissionHistory` interface has the following methods:

```
// Find the highest grade of any submission for a given student
public Integer getBestGrade(String unikey);

// The most recent submission for a given student
public Submission getSubmissionFinal(String unikey);

// The most recent submission for a given student, prior to a given time
public Submission getSubmissionBefore(String unikey, Date deadline);

// Add a new submission (can assume submissions from one student have different times)
public Submission add(String unikey, Date timestamp, Integer grade);

// Remove a submission (can assume submissions from one student have different times)
public void remove(Submission submission);

// Get all the students who have the highest grade
public List<String> listTopStudents();

// Get all the students whose most recent submission has lower grade than their best submission
public List<String> listRegressions();
```

For all of the interface methods, if any of the arguments passed to the method are `null`, then you should `throw new IllegalArgumentException();`

The full skeleton code for the assignment (with more detailed comments than those given here) is available for download on the resources section of Ed, and is also listed in the appendix.

Think carefully about which data structure(s) are most suitable. A simple solution can work correctly (and get reasonable marks) but for efficiency, you may find it useful to store data in more than one data structure or to nest structures.

To achieve *maximum* marks, your solution should be efficient. All methods should run in sub-linear time (i.e. *better* than $O(n)$, where n is the number of submissions. You can still get good marks (i.e. a credit) for simpler, less efficient solutions (provided that they are correct, and are correctly analyzed).

Exceptions:

- `listRegressions` should be better than $O(n^2)$ time
- `listTopStudents` should be better than $O(n)$ time, with the assumption that the number of students being returned is sub-linear.

1.3 Submission objects

Each `Submission` object has a unikey (`String`), a timestamp (`java.util.Date`), and a grade (`Integer`). You may assume that `Submission` objects are *immutable*. That means that once an submission has been added, it cannot be modified (but it could be removed.) You may assume that no student will have multiple submissions with the same timestamp. You may assume that any Submission object has non-null values for its three fields.

1.4 Example

Suppose we added the following submissions:

```
SimpleDateFormat df = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
SubmissionHistory history = new SubmissionHistory();
history.add("aaaa1234", df.parse("2016/09/03 09:00:00"), 66); //submission A
history.add("aaaa1234", df.parse("2016/09/03 16:00:00"), 86); //submission B
history.add("cccc1234", df.parse("2016/09/03 16:00:00"), 73); //submission C
history.add("aaaa1234", df.parse("2016/09/03 18:00:00"), 40); //submission D

// This will return an Integer corresponding to the number 86
Integer example1 = history.getBestGrade("aaaa1234");

// This will return null
Integer example2 = history.getBestGrade("zzzz1234");

// This will throw new IllegalArgumentException();
Integer example3 = history.getBestGrade(null);

// This will return a Submission corresponding to submission D
Submission example4 = history.getSubmissionFinal("aaaa1234");

// This will return a Submission corresponding to submission C
Submission example5 = history.getSubmissionFinal("cccc1234");

// This will return a Submission corresponding to submission A
Submission example6 = history.getSubmissionBefore("aaaa1234", df.parse("2016/09/03 13:00:00"));

// This will return null
Submission example7 = history.getSubmissionBefore("cccc1234", df.parse("2016/09/03 13:00:00"));

// This will return a list containing only {"aaaa1234"}
// because that student's final submission had grade 40, but their best was 86
List<String> example8 = history.listRegressions();

// This will return a list containing only {"aaaa1234"}
// because that was the only student with the highest grade
List<String> example9 = history.listTopStudents();

// If we added another student with the same highest mark, they would both be returned
// If we instead removed submission B, then {"cccc1234"} would become the top student
```

2 Marking

2.1 Code automarking [40%]

Part of the marking of your code will be done automatically on Ed. You are encouraged to submit early and often – there is no penalty for making multiple submissions.

There will be *some* visible tests to help you verify that your solution is configured correctly. The remaining test cases will be invisible until *after* the submission deadline. Test your code carefully to ensure that your solution works correctly. The score on this component will be based on the number of tests passed. If you pass every visible test, you will get at least half of the available marks.

2.2 Code hand marking [30% for students in INFO1105 or INFO1905; 20% for students in INFO9105]

Part of the marking of your code will be done by hand. We will grade your code on its readability and overall quality. Your code should be well laid out, with appropriate use of whitespace. Comments should be used where it makes your code easier to understand. Variables and methods should be named appropriately. If appropriate, helper methods

should be used to avoid duplicating code, and to make any very complex methods easier to understand.

To gain half the marks for this component, your code needs to be understandable without great effort by a reader. Three-quarters of the marks for this component would be awarded for code where, in addition, the design makes some of the methods efficient, and where the structures used are adequately documented in the code itself, and also you have written a set of tests (besides those we provided) that check for the main aspects of correctness of several of the public methods. To get full marks, your design needs to make every method efficient, and it must still be easy to understand, and also you have written a very thorough set of tests (besides those we provided) that check for correctness of all the public methods.

Note that if you have a complicated efficient design, and it is not easy to understand, then you will get less than half the marks.

2.3 Report [30%] for students in INFO1105 or INFO1905

You should write a brief report (please try to keep it under 1 page). The report should include the following:

- A high level overview of your implementation (one paragraph, or bullet points.)
- For each data structure used, state what you used it for (one paragraph each, or bullet points.)
- List each method from the description, stating the running time in big-Oh notation, and giving a brief justification (for example, a justification could say how often various methods of the underlying data structure are called, and what is the big-Oh cost of each call).

An easy-to-read report that gives correct running times for the methods, with reasonable justification, will gain at least half the marks for this component. To gain three-quarters of the marks on this component, you should also have a design where several of the methods are efficient. To gain full marks for this component, you should also have a design where every method is efficient.

Note that if you have a complicated design, and do not analyse its methods correctly, then you will get less than half the marks.

2.4 Report [40%] for students in INFO9105

You should write a report (please try to keep it under 3 pages). The report should include the following:

- A high level overview of your implementation (one paragraph, or bullet points.)
- For each data structure used, state what you used it for and why (one paragraph each, or bullet points.)
- List each method from the description, stating the running time in big-Oh notation, and giving a brief justification (for example, a justification could say how often various methods of the underlying data structure are called, and what is the big-Oh cost of each call).

- A discussion that compares your design to an alternative design thast you considered, and explains the advantages and disadvantages of each of these two approaches. (1-2 pages)

The first three aspects will be scored out of 30, using the same scheme as for the undergraduate students. That is: An easy-to-read report that gives correct running times for the methods, with reasonable justification, will gain at least half the marks for this component. To gain three-quarters of the marks on this component, you should also have a design where several of the methods are efficient. To gain full marks for this component, you should also have a design where every method is efficient. Note that if you have a complicated design, and do not analyse its methods correctly, then you will get less than half the marks.

The fourth aspect (the discussion of the comparison with an alternative design) will be scored out of 10. To get half the marks, you need to present an alternative design that could work correctly, and identify one substantial advantage of the design you actually coded. For three-quarters of this aspect, you need to also identify one valid disadvantage of the design you coded. For full marks, you need a reasonable and balanced discussion that shows a good understanding of the alternatives and their implications.

3 Appendix: Skeleton code

The skeleton code is included here for your reference, but we **strongly** recommend that you download it from Ed instead, to avoid any text encoding artifacts that might be introduced by copying it from this pdf.

3.1 SubmissionHistory.java

```
import java.util.Date;
import java.util.List;

public interface SubmissionHistory {

        /**
         * Find the highest grade of any submission by this student
         *
         * @param unikey
         *             The student to filter on
         * @return the best grade by this student, or null if they have made no
         *         submissions
         * @throws IllegalArgumentException
         *              if the argument is null
         */
        public Integer getBestGrade(String unikey);

        /**
         * The most recent submission for a given student
         *
         * @param unikey
         *             The student to filter on
         * @return Submission made most recently by that student, or null if the
         *         student has made no submissions
         * @throws IllegalArgumentException
         *              if the argument is null
         */
        public Submission getSubmissionFinal(String unikey);
```

```
    /**
     * The most recent submission for a given student, prior to a given time
     *
     * @param unikey
     *             The student to filter on
     * @param deadline
     *             The deadline after which no submissions are considered
     * @return Submission made most recently by that student, or null if the
     *         student has made no submissions
     * @throws IllegalArgumentException
     *              if the argument is null
     */
    public Submission getSubmissionBefore(String unikey, Date deadline);

    /**
     * Add a new submission
     *
     * For simplicity, you may assume that all submissions have unique times
     *
     * @param unikey
     * @param timestamp
     * @param grade
     * @return the Submission object that was created
     * @throws IllegalArgumentException
     *              if any argument is null
     */
    public Submission add(String unikey, Date timestamp, Integer grade);

    /**
     * Remove a submission
     *
     * For simplicity, you may assume that all submissions have unique times
     *
     * @param submission
     *             The Submission to remove
     * @throws IllegalArgumentException
     *              if the argument is null
     */
    public void remove(Submission submission);

    /**
     * Get all the students who achieved the highest grade (in any of their
     * submissions).
     *
     * For example, if the highest grade achieved by any student was 93, then
     * this would return a list of all the students who have made a submission
     * graded at 93.
  *
  * If no submissions have been made, then return an empty list.
     *
     * @return a list of unikeys
     */
    public List<String> listTopStudents();

    /**
     * Get all the students whose most recent submissions have lower grades than
     * their best submissions
     *
     * @return a list of unikeys
     */
    public List<String> listRegressions();

}
```

3.2 Submission.java

```
import java.util.Date;
```

```
public interface Submission {

        /**
         * @return the unikey (a String of the form "abcd1234")
         */
        public String getUnikey();

        /**
         * @return a Date object representing the time the submission was made
         */
        public Date getTime();

        /**
         * @return an integer grade
         */
        public Integer getGrade();
}
```

3.3 Assignment.java

```
import java.util.Date;
import java.util.List;

public class Assignment implements SubmissionHistory {

        /**
         * Default constructor
         */
        public Assignment() {
                // TODO initialise your data structures
        }

        @Override
        public Integer getBestGrade(String unikey) {
                // TODO Implement this, ideally in better than O(n)
                return null;
        }

        @Override
        public Submission getSubmissionFinal(String unikey) {
                // TODO Implement this, ideally in better than O(n)
                return null;
        }

        @Override
        public Submission getSubmissionBefore(String unikey, Date deadline) {
                // TODO Implement this, ideally in better than O(n)
                return null;
        }

        @Override
        public Submission add(String unikey, Date timestamp, Integer grade) {
                // TODO Implement this, ideally in better than O(n)
                return null;
        }

        @Override
        public void remove(Submission submission) {
                // TODO Implement this, ideally in better than O(n)

        }

        @Override
        public List<String> listTopStudents() {
                // TODO Implement this, ideally in better than O(n)
                // (you may ignore the length of the list in the analysis)
                return null;
        }
```

```
        @Override
        public List<String> listRegressions() {
                // TODO Implement this, ideally in better than O(n^2)
                return null;
        }

}
```