INFO1105 2016 Semester 2, Assignment 1

September 2, 2016

Submission details

- Due: **5pm on Friday, 16th September, 2016**
- Submit your **report** via Blackboard (turnitin). The report must be in pdf format, and cannot be handwritten. Note that your submission is not complete until you see the "Congratulations - your submission is complete!" message.
- Submit your **source code** on Ed, *including* any JUnit tests you make.
- Submit your **source code** on eLearning, *including* any JUnit tests you make, and a **signed cover sheet** as a zip file. The cover sheet is downloadable on Ed.
- The policy for late submissions is described on slide 21 of Lecture 1a.
- This is an **individual assignment**, so your code and report should be entirely your own work. We recommend that you use data structures implemented in 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.

1 Calendar

1.1 Description

In this assignment you will write the code for a class `Assignment` which implements the `Calendar` interface. The full skeleton code for the assignment is available for download on the resources section of Ed, and is also listed in the appendix.

1.2 The Calendar interface

Objects implementing the `Calendar` interface allow you to keep track of `Appointment` objects. The `Calendar` interface has the following methods:

```
// Return a list of all appointments at the given location (at any time)
// If there are no such appointments, return an empty list
// Throws IllegalArgumentException if the argument is null
public List<Appointment> getAppointments(String location);

// Return the next appointment at or after the given time (in any location)
// If there is no such appointment, return null
// Throws IllegalArgumentException if the argument is null
public Appointment getNextAppointment(Date when);

// Return the next appointment at or after the given time, at that location
// If there is no such appointment, return null
// Throws IllegalArgumentException if any argument is null
public Appointment getNextAppointment(Date when, String location);

// Create a new appointment in the calendar
// Throws IllegalArgumentException if any argument is null
public void add(String description, Date when, String location);

// Remove an appointment from the calendar
// Throws IllegalArgumentException if the argument is null
public void remove(Appointment appointment);
```

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

Think carefully about which data structure(s) are most suitable. It may be useful to store data in more than one data structure. To achieve full marks, it is a requirement that each method runs in sub-linear time (i.e. they are *better* than $O(n)$).

1.3 Appointment objects

Each `Appointment` object has a description (`String`), a start time (`java.util.Date`), and a location (`String`). You may assume that `Appointment` objects are *immutable*. That means that once an `Appointment` has been created, it cannot be modified (but it could be removed.)

1.4 Example

```
SimpleDateFormat df = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
Calendar calendar = new Calendar();
calendar.add("A", df.parse("2016/09/03 09:00:00"), "SIT lab 117");
calendar.add("B", df.parse("2016/09/03 16:00:00"), "SIT lab 117");
calendar.add("C", df.parse("2016/09/03 16:00:00"), "SIT lab 118");
calendar.add("D", df.parse("2016/09/03 18:00:00"), "SIT lab 117");
```

```
// This will return a list containing Appointments A, B and D
List<Appointment> example1 = calendar.getAppointments("SIT lab 117");

// This can return either Appointment B, or Appointment C
// It does not matter which one, as they are both correct.
Appointment example2 = calendar.getNextAppointment(df.parse("2016/09/03 13:00:00"));

// This will return Appointment B
Appointment example3 = calendar.getNextAppointment(df.parse("2016/09/03 13:00:00"), "SIT lab 117");

// This would return null
Appointment example4 = calendar.getNextAppointment(df.parse("2020/01/01 13:00:00"));

// This would cause an IllegalArgumentException to be thrown
Appointment example5 = calendar.getNextAppointment(null);
```

1.5 Report

You must write a short report (please try to keep it under 2 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 you chose it (one paragraph each, or bullet points.)
- For each of the five methods declared in the `Calendar` interface, state the running time of your implementation in big-Oh notation, and give a brief argument justifying that this is correct.
- Describe how you tested your code. List the test cases you wrote, stating briefly the purpose of each test.

2 Marking

2.1 Code automarking [30%]

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.

- Pass level: The code passes all of the publically visible tests.
- Distinction level: The code also passes a majority of the private tests.

2.2 Code design [30%]

This is based on your report and on examination of the code.

- Pass level: The central decisions about data structures and algorithms are reasonable, would provide the basis for meeting the required functionality, and are explained in the report.

- Distinction level: The central decisions about data structures and algorithms are well made, executed appropriately in the code to provide required functionality at good efficiency, and are explained clearly in the report.

2.3 Code style [10%]

This is based on examination of the the code.

- Pass level: Most of the code is understandable without particular effort, with mostly sensible layout, and meaningful comments where needed.
- Distinction level: The code is easy to follow, helping the reader and avoiding distraction. The code has a good choice of layout and comments; well chosen names; appropriate use of helper methods; and idiomatic use of Java language.

2.4 Testing [20%]

This is based on your report and on examination of the code.

- Pass level: There is a reasonable range of tests for normal cases of the public methods, and are listed in the report.
- Distinction level: The tests are well chosen and justified, covering a significant range of both normal and corner cases.

2.5 Analysis of runtime [10%]

This is based on your report (the implementation is assessed separately in the Code Design section).

- Pass level: Correct big-Oh for majority of public methods.
- Distinction level: Convincing and valid arguments in most cases.

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 Calendar.java

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

public interface Calendar {

        // Return a list of all appointments at the given location (at any time)
        // If there are no such appointments, return an empty list
        // Throws IllegalArgumentException if the argument is null
        public List<Appointment> getAppointments(String location);

        // Return the next appointment at or after the given time (in any location)
```

```
        // If there is no such appointment, return null
        // Throws IllegalArgumentException if the argument is null
        public Appointment getNextAppointment(Date when);

        // Return the next appointment at or after the given time, at that location
        // If there is no such appointment, return null
        // Throws IllegalArgumentException if any argument is null
        public Appointment getNextAppointment(Date when, String location);

        // Create a new appointment in the calendar
        // Throws IllegalArgumentException if any argument is null
        public void add(String description, Date when, String location);

        // Remove an appointment from the calendar
        // Throws IllegalArgumentException if the argument is null
        public void remove(Appointment appointment);
}
```

3.2 Appointment.java

```
import java.util.Date;

public interface Appointment {

        public String getDescription();
        public String getLocation();
        public Date getStartTime();

}
```

3.3 Assignment.java

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

public class Assignment implements Calendar {

        // The default constructor for the class should be public
        // We will use this when we test your code!
        public Assignment() {
        }

        @Override
        public List<Appointment> getAppointments(String location) {
                // TODO Implement this! (then remove this TODO comment)
                return null;
        }

        @Override
        public Appointment getNextAppointment(Date when) {
                if(when == null) {
                        throw new IllegalArgumentException("time was null");
                }
                // TODO Implement this! (then remove this TODO comment)
                return null;
        }

        @Override
        public Appointment getNextAppointment(Date when, String location) {
                // TODO Implement this! (then remove this TODO comment)
                return null;
        }

        @Override
        public void add(String description, Date when, String location) {
                // TODO Implement this! (then remove this TODO comment)
        }
```

```
        @Override
        public void remove(Appointment appointment) {
                // TODO Implement this! (then remove this TODO comment)
        }
}
```

3.4 ExampleTests.java

This JUnit test class has a few tests which correspond to the example code. It includes an example of testing for thrown exceptions, and one possible approach for comparing `List` objects. It is not part of the skeleton code, but you are welcome to re-use code from this example file to help you get started on writing your own test cases. It is available as a separate download on Ed.