COS 126	General Computer Science	Fall 2012

Practice Programming Exam 4

This test has 1 question with 3 parts. You have 90 minutes. The exam is open book, open note, and open course website and booksite. You may use code from your programming assignments or the Introduction to Programming in Java booksite. No communication with any non-staff members is permitted. IF this were an actual exam, you would be directed to submit your solution via Dropbox. **Write out and sign the Honor Code pledge before turning in the test.**

"I pledge my honor that I have not violated the Honor Code during this examination."

_____
Signature

Name:

NetID:

Precept:

Do not remove this exam from the exam room.

Problem. Your task is to create a calendar that can track events on a per-day basis. Client programs need to be able to schedule events and detect conflicts between events. Test `main()` methods are provided for each class.

Part 1 (12 points). First, write a class `Event` that implements the following API:

API Specification

```
public class Event
-----------------------------------------------------------------------------------------------
        Event(int start, int end, String info) // construct an Event using the arguments
   int  getStart()                             // return start time of event
   int  getEnd()                               // return end time of event
String  getInfo()                              // return information about event
boolean conflict(Event that)                   // Does this Event conflict with that Event?
String  toString()                             // return String representation of Event
  void  main()                                 // test main
```

The constructor should simply save the argument values in instance variables. The start and end times are based on a 24 hour clock. It is not necessary to write leading zeroes. Do not worry about checking for illegal times. (i.e., hours over 23 or minutes over 59)

The accessor methods `getStart()`, `getEnd()` and `getInfo()` should each return the appropriate instance variable.

The `conflict()` method should return `true` if the argument `Event` conflicts with this `Event`. Two events conflict if their times overlap. (It is OK if the start time of the later one is equal to the end time of the earlier one. That is not a conflict.)

The `toString()` method should return a `String` consisting of start time and end time separated by a dash, followed by a colon, a space and the information about the event. (e.g., 1000-1050: COS126 lecture)

Part 2 (12 points). Your next task is to implement a `Day` class. A `Day` is a linked list of `Event` objects linked according to start times, from earliest to latest. If events have the same start time, it does not matter which comes first, but they should be next to each other.

You will need a private inner `Node` class defined as follows:

```
private class Node {
    private Event ev;
    private Node next;
}
```

API Specification

```
public class Day
-------------------------------------------------------------------------------------------------
        Day()                       // construct an empty Day
  void  addEvent(Event ev)          // add Event ev to this Day
                                    // insert it in the linked list in start time order
  void  show()                      // output the Events of this Day, one event per line
                                    // output a warning about conflicts between adjacent events
                                    // as they are printed.
  void  main()                      // test main
```

Part 3 (6 points). Your final task is to implement a `Calendar` class. The `Calendar` class should store a symbol table that uses an 8-digit integer date (yyyymmdd) as the key and a `Day` object as the value.

Your `Calendar` should be a sparse calendar. That is, you should only add an entry to your symbol table if there is at least one `Event` on that particular date.

API Specification

```
public class Calendar
-----------------------------------------------------------------------------------------------
         Calendar()                       // construct an empty Calendar instance
                                          // by initializing its symbol table
 void    schedule(int date, Event ev)     // schedule an Event on a particular date
 void    show()                           // output the Calendar
 void    main()                           // test main
```

The `schedule` method should use the `addEvent()` method for the correct `Day` object. If there is no `Day` instance for that date, then `schedule()` should construct one and add it to the symbol table.

Testing and Sample Runs. Below is a test `main()` for `Event`. You may copy it from the precepts page.

```
    // test main for Event
    public static void main(String[] args) {
        // set up some events
        Event ev0 = new Event(900, 1000, "Breakfast");
        Event ev1 = new Event(1000, 1050, "COS126 lecture");
        Event ev2 = new Event(1230, 1320, "COS126 precept");
        Event ev3 = new Event(1200, 1300, "lunch with Bob");

        StdOut.println(ev0);
        StdOut.println(ev1);
        StdOut.println("Does " + ev0.getInfo() + " conflict with "
                           + ev1.getInfo() + "?");
        if (ev0.conflict(ev1)) StdOut.println("Yes.");
        else                   StdOut.println("No.");
        StdOut.println("Does " + ev2.getInfo() + " conflict with "
                           + ev3.getInfo() + "?");
        if (ev2.conflict(ev3)) StdOut.println("Yes.");
        else                   StdOut.println("No.");
    }
```

Here is our output for the `Event` test `main()` given above:

```
> java Event
900-1000: Breakfast
1000-1050: COS126 lecture
Does Breakfast conflict with COS126 lecture?
No.
Does COS126 precept conflict with lunch with Bob?
Yes.
```

Here is a test `main()` for `Day`. You may copy it from the precepts page.

```
// test main for Day
public static void main(String[] args) {
    // set up some events
    Event ev1 = new Event(1000, 1050, "COS126 lecture");
    Event ev2 = new Event(1230, 1320, "COS126 precept");
    Event ev3 = new Event(1200, 1300, "lunch with Bob");

    // Add to the same day
    Day tues = new Day();
    tues.addEvent(ev1);
    tues.addEvent(ev2);
    tues.addEvent(ev3);

    // Output the events for the Day
    tues.show();
}
```

Here is our output for the `Day` test `main()` given above:

```
> java Day
1000-1050: COS126 lecture
1200-1300: lunch with Bob
Warning: 1200-1300: lunch with Bob conflicts with 1230-1320: COS126 precept
1230-1320: COS126 precept
```

Here is a test `main()` for `Calendar`. You may copy it from the precepts page.

```
// test main for Calendar
public static void main(String[] args) {
    // set up some events
    Event ev1 = new Event(1000, 1050, "COS126 exam");
    Event ev2 = new Event(1930, 2100, "COS126 programming exam");
    Event ev3 = new Event(1200, 1300, "lunch with Bob");
    Event ev4 = new Event(1000, 1050, "COS126 lecture");
    Event ev5 = new Event(1230, 1320, "COS126 precept");

    // set up some dates
    int d1 = 20120501;
    int d2 = 20120503;

    Calendar c = new Calendar();
    c.schedule(d1, ev2);
    c.schedule(d1, ev1);
    c.schedule(d2, ev3);
    c.schedule(d2, ev4);
    c.schedule(d2, ev5);

    // Output the events for each date
    c.show();
}
```

Here is our output for the `Calendar` test `main()`:

```
> java Calendar
20120501
1000-1050: COS126 exam
1930-2100: COS126 programming exam

20120503
1000-1050: COS126 lecture
1200-1300: lunch with Bob
Warning: 1200-1300: lunch with Bob conflicts with 1230-1320: COS126 precept
1230-1320: COS126 precept
```

Your programs should work for other clients as well.

Submission. If this were an actual programming exam, you would be directed to submit the three files `Event.java`, `Day.java` and `Calendar.java` via `Dropbox` and reminded to click the *Check all submitted files* button to verify your submission.

Grading. *If this were an actual exam, your program would be graded on correctness and clarity. You would receive partial credit for correctly implementing the following components:*

- *Header with name, login, precept*
- *Constructor and each method for* `Event` *class.*
- *Constructor and each method for* `Day` *class.*
- *Constructor and each method for* `Calendar` *class.*

You will receive a substantial penalty if your program does not compile or if you do not follow the prescribed API or output specification.