Second IIP mid term exam recovery (ETSInf)
January 19, 2018. Duration: 2 hours and a half

Note: The maximum grade of this exam is 10 points, but its weight in the final grade of IIP is **3,6 points**

NAME:	GROUP:

1. 6 points It is available the class `Appointment` for representing appointments scheduled in the agenda of a dental clinic. Each appointment is associated with an operation to be done in the clinic. The opening hours of the dental clinic is from 8 AM till 8 PM with no stops. A dental clinic can perform three operation types: orthodontic revision, mouth cleaning and extraction of a tooth. Orthodontic revisions have an estimated duration of 20 minutes, mouth cleanings 30 minutes and extractions of a tooth 60 minutes. To avoid staying open after 8 PM, the starting time of the last appointment to be scheduled in the agenda is 7 PM. So starting times should be in the range 8 AM to 7 PM. The class `Appointment` is already implemented and available to be used. Next figures show the parts of the documentation of classes `Appointment` and `TimeInstant` which are needed for completing the class `Agenda`.

Class `Appointment`:

Fields

Modifier and Type	Field and Description
static int	**CLEANING** Constant indicating a mooth cleaning, its value is 1.
static int	**EXTRACTION** Constant indicating an extraction of a tooth, its value is 0.
static int	**ORTHODONTIC** Constant indicating an orthodontic revision, its value is 2.

Constructors

Constructor and Description
Appointment(int type, java.lang.String patientName, int hour, int minutes) Creates an appointment of the operation type indicated by type, with the patient name patientName, for starting at the time specified by parameters hour y minutes.

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
int	**compareTo**(**Appointment** other) Returns > 0 if this starts before other, < 0 if this starts after other, and 0 when both times are the same.
int	**getDuration**() Returns the lapse of time in minutes an operation is going to last according to the operation type.
java.lang.String	**getName**() Returns the patient name.
TimeInstant	**getStartingTime**() Returns the scheduled starting time for the operation.
int	**getType**() Returns the operation type.

Class `TimeInstant`:

Constructors

Constructor and Description
TimeInstant(int hh, int mm) TimeInstant corresponding to hh hours and mm minutes.

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
int	**toMinutes**() Returns number of minutes from 00:00 until current TimeInstant object.

To be done: Write the class `Agenda` for representing the daily schedule of a dental clinic. The following attributes should be used in the class `Agenda`, and you have to implement the following methods:

a) (0,5 points) Attributes:

- `MAX_APPOINTMENTS`: constant, static and public attribute of type integer for indicating the maximum number of operations which can be scheduled per day. This maximum number is $34 = (19-8)*3+1$, which results of considering that the maximum number of operations per hour is 3 due to the fact that the minimum duration of an operation is 20 minutes. So, 33 operations can be scheduled from 8 AM till 7 PM. Then, the last one which can be scheduled for starting at 7 PM, because an extraction can last 60 minutes.
- `numAppointments`: instance private attribute of type integer but restricted to values in the interval `[0..MAX_APPOINTMENTS]`, that indicates the number of currently scheduled operations in the agenda.
- `appointments`: instance private attribute, an array of `Appointments` for storing the appointments of one day; the stored appointments should be placed together from position 0 till `numAppointments-1`, both included. And the stored appointments should be sorted in chronological order by considering the starting time. Additionally, no overlap between appointments is allowed.
- `numOrthodonticRevisions`: instance private attribute of type integer for indicating the number of scheduled orthodontic revisions stored in the array.

b) (0,5 points) A default constructor (with no parameters) for creating the array and setting to zero the number of appointments and the number of appointments which are orthodontic revisions.

c) (2,0 points) A method with the profile:

```
public boolean insert(Appointment a)
```

for inserting a new appointment `a` in the agenda, i.e., tries to insert the appointment in the array `appointments`. For simplifying this method, you can assume as a precondition that the new appointment `a` does not overlap with any of the existing appointments in the array. This method has to check if there are free places in the array and that the new appointment has an starting time in the range corresponding to the opening hours. Considering that the appointments already stored in the array are sorted in ascending order according to the starting time, the new appointment should be placed in the correct position, so, those existing appointments whose starting time is after the starting time of the new one should be shifted one place to the right. The method `insert` will return `true` when it has been possible to insert the new appointment, and `false` otherwise.

d) (1,0 points) Method with profile:

```
public int getNumberOfExtractions()
```

that returns an integer indicating the number of extractions of a tooth scheduled for the day.

e) (2,0 points) Method with profile:

```
public String[] getNamesOfOrthodonticRevisions()
```

that returns an array of `String` with the names of the patients with an orthodontic revision scheduled. The length of this array should be equal to the number of scheduled appointments for orthodontic revisions, or 0 if no orthodontic revisions are scheduled.

Solution:

```
/**
 * Class <code>Agenda</code> for managing the appointments of a dental clinic.
 * The appointments are sorted by the estimated starting time.
 * The opening time slot starts at 08:00 AM and ends and 08:00 PM.
 * The last appointment that can be scheduled is at 07:00 PM
 * in order to ensure the dental clinic closes at 08:00 PM,
 * this is because an extraction of a tooth lasts 60 minutes.
 *
 * They are three types of appointments, which are,
 * extraction of a tooth, mouth cleaning and orthodontic revision.
 * Each operation has an averaged duration depending on his type,
 * orthodontic revision 20 minutes, mouth cleanings 30 minutes,
 * and extractions of a tooth 60 minutes.
 *
 * @author IIP
 * @version Academic year 2017-18
 */
public class Agenda {
    /** Constant indicating the maximum amount of apppoints per day. */
    public static final int MAX_APPOINTMENTS = (19 - 8) * 3 + 1;
```

```
    /** Array for storing the appointments scheduled in the dental clinic. */
    private Appointment[] appointments;
    /** Number of valid appointments stored in the array appointments. */
    private int numAppointments;
    /** Number of appointments sotred in the array appointments
        whose operation type is Appointment.ORTHODONTIC. */
    private int numOrthodonticRevisions;

    /**
     * Creates an agenda for managing the possible operations of a dental
     * clinic scheduled for a unique day.
     * At the beginning the agenda will be empty, i.e., it will not store
     * any appointments. Appointments will be inserted progressively.
     */
    public Agenda() {
        this.numAppointments = 0;
        this.numOrthodonticRevisions = 0;
        this.appointments = new Appointment[MAX_APPOINTMENTS];
    }

    /**
     * Inserts a new appointment into the agenda at the correct position in
     * order to maintain all the appointments sorted in chronological order.
     * Precondition: the appointment <code>a</code> does not overlap with
     * any of the other ones already stored in the array <code>appointments</code>.
     */
    public boolean insert(Appointment a) {
        // If there is no free space returns false.
        if (numAppointments == appointments.length) { return false; }

        // If the starting time of the new appointment is outside opening hours
        // returns false.
        if (a.getStartingTime().toMinutes() < 8 * 60
            || a.getStartingTime().toMinutes() > 19 * 60) { return false; }

        // Searches the correct position for the new appointment.
        int pos = 0;
        while (pos < numAppointments && a.compareTo(appointments[pos]) < 0) { pos++; }

        // All the appointments whose starting time is after the new one are
        // shifted one place to the right, i.e., all the appointments that
        // are located in the subarray [pos, numappointments - 1]
        for (int i = numAppointments; i > pos; --i) {
            appointments[i] = appointments[i - 1];
        }

        // The new appointment is placed at its position in the array.
        appointments[pos] = a;

        // The number of stored appointments is increased.
        ++numAppointments;

        // The number of stored appointments of the type orthodontic revision
        // is increased if the type of the new appointment is Appointment.ORTHODONTIC.
        if (a.getType() == Appointment.ORTHODONTIC) { ++numOrthodonticRevisions; }

        return true;
    }

    /** Returns the number of appointments corresponding to extractions of a tooth. */
    public int getNumberOfExtractions() {
        int counter = 0;
        for (int i = 0; i < numAppointments; i++) {
            if (appointments[i].getType() == Appointment.EXTRACTION) { ++counter; }
        }
        return counter;
    }

    /** Returns an array of String with the names of the patients
     *  with an appointment for orthodontic revision. */
    public String[] getNamesOfOrthodonticRevisions() {
        String[] names = new String[numOrthodonticRevisions];
        int k = 0;
        for (int i = 0; i < numAppointments && k < numOrthodonticRevisions; i++) {
            if (appointments[i].getType() == Appointment.ORTHODONTIC) {
                names[k++] = appointments[i].getName();
            }
        }
        return names;
    }
}
```

2. [2 points] **To be done:** implement an static method that given an integer `n` $\geq$ `3`, shows on standard output a figure of `n` lines. Three `N` letters should appear per line, with the appropriate separation for representing a capital N. If `n = 5` the output should be:

```
NN    N
N N   N
N  N  N
N   N N
N    NN
```

Solution:

```
/** Precondition: n >= 3 */
public static void letter(int n) {
    for (int i = 1; i <= n; i++) {
        System.out.print('N');
        for (int j = 1; j < i; j++) {
            System.out.print(' ');
        }
        System.out.print('N');
        for (int j = 1; j <= n - i; j++) {
            System.out.print(' ');
        }
        System.out.println('N');
    }
}
```

3. [2 points] **To be done:** implement an static method that given an array of integer values checks if it is **Even Dominant**, i.e., whether the sum of the absolute values stored at even positions in the array is greater than all individual absolute values stored at odd positions.

Solution:

```
public static boolean evenDominant(int[] v) {
    boolean toContinue = true;
    int sum = 0, j = 1;
    for (int i = 0; i < v.length; i = i + 2) {
        sum = sum + Math.abs(v[i]);
    }
    while (j < v.length && toContinue) {
        if (Math.abs(v[j]) >= sum) { toContinue = false; }
        else { j = j + 2; }
    }
    return toContinue;
}
```