IIP Second Partial - ETSInf - Retake exam
January 24th, 2017. Time: 2 hours and 30 minutes.

1. [6 points] You have available the `Station` class, that represents a train station for a given trip. The data on the station is given by its town name, the time when the stop is done, and the type of station. Station can be of type stop, halt, or service (that does not allow passengers to get aboard).

   Below there is a summary of its documentation, including constants and some public methods:

Field Summary

Modifier and Type	Field and Description
static int	HALT
static int	SERVICE
static int	STOP

Constructor Summary

Constructor and Description
Station(java.lang.String to, iipUtil.TimeInstant ti, int ty) Creates a Station in town to at time ti and type ty.

Method Summary

Modifier and Type	Method and Description
iipUtil.TimeInstant	getStopTime() Devuelve stop time for this station.
java.lang.String	getTown() Returns town of the this station.
int	getType() Returns type of this station: STOP, HALT or SERVICE.

   You must: Implement the class `TrainTrip` that represents the stations sequence that a train makes between an origin and a destination. For that, you should employ the following attributes and methods:

   a) (0.5 points) Attributes (only first one is public):
   - `MAX_STATIONS`, class (`static`) constant that represents the maximum number of stations of any trip, with value 25. This constants and those of the `Station` class must be used whenever required.
   - `numStations`, integer in the interval `[0..MAX_STATIONS]` that represents the actual number of stations in the trip in each moment.
   - `trip`, array with base datatype `Station`, of size `MAX_STATION`, that stores the stations in the trip in each moment, in such a way that they are in consecutive positions in the array, from `0` to `numStations - 1` both included, **in ascendent chronological order according to stop time**, being `trip[0]` origin and `trip[numStations - 1]` destination.
   - `numBoardings`, non-negative integer that represents the number of stations in the trip that allow passengers to get aboard, i.e., those that are full or halt.

   b) (1 point) A constructor that given `Station origin` and `Station destination`, where `destination` stop time is posterior to `origin` stop time, creates a `TrainTrip` object whose trip has only those two stations.

   c) (1.5 points) A method with header:

```
private int position(TimeInstant tS)
```

   that returns -1 if time `tS` is previous to stop time at origin, or posterior to stop time at destination, or there exist a station with stop time equal to `tS`; otherwise, it returns the index of the first station in `trip` whose stop time is posterior to `tS`.
   Remember that two `TimeInstant` objects `t1` and `t2` may be compared by using its `compareTo` method, in such a way that `t1.compareTo(t2)` is < 0, 0, or > 0 when `t1` is previous, equal, and posterior to `t2`, respectively.

   d) (1.5 points) A method with header:

```
public boolean add(Station s)
```

   that adds `s` to the trip. If time stop of `s` is not equal to any other station in the trip, it is posterior to origin time stop, and it is previous to destination time stop, `s` is added to keep the chronological order and `true` is returned. If `s` cannot be added, because time stop incompabilities or because no more stations are allowed, `false` is returned.

In the case that you can add the station, the `position` method must be used to know where `s` must be placed into `trip`. Once you have found that position, you have to make an empty space in the array. For that you can employ a private method, already implemented, with header:

```
private void moveRight(int begin, int end)
```

that moves a position to the right the elements of `trip[begin..end]`, where `0` $\leq$ `begin` $\leq$ `end` $\leq$ `numStations - 1` $<$ `trip.length - 1`.

e) (1.5 points) A method with header:

```
public Station[] aboards()
```

that returns an array of `Station` with the stations of the trip that allow passengers to get aboard, i.e., that are full or halt stations. The length of this array would be equal to the number of stations where passengers are allowed to get aboard or 0 if no station allows it.

Solution:

```
import iipUtil.TimeInstant;

public class TrainTrip {
    public static final int MAX_STATIONS = 25;
    private Station[] trip;
    private int numStations;
    private int numBoardings;

    /** Precondition: origin and destination are different stations,
     *  stop time in destination is posterior to that of origin */
    public TrainTrip(Station origin, Station destination) {
        trip = new Station[MAX_STATIONS];
        trip[0] = origin;
        trip[1] = destination;
        numStations = 2;
        if (origin.getType() != Station.SERVICE) { numBoardings++; }
        if (destination.getType() != Station.SERVICE) { numBoardings++; }
    }

    private int position(TimeInstant tS) {
        if (trip[0].getStopTime().compareTo(tS) >= 0
            || trip[numStations - 1].getStopTime().compareTo(tS) <= 0) {
            return -1;
        }
        int i = 1;
        while (i < numStations - 1 && trip[i].getStopTime().compareTo(tS) < 0) {
            i++;
        }
        if (trip[i].getStopTime().compareTo(tS) == 0) { return -1; }
        else { return i; }
    }

    /** Precondition: 0 <= begin <= end <= numStations - 1 < trip.length - 1 */
    private void moveRight(int begin, int end) {
        for (int pos = end + 1; pos > begin; pos--) {
            trip[pos] = trip[pos - 1];
        }
    }

    public boolean add(Station s) {
        if (numStations == MAX_STATIONS) { return false; }
        int pos = position(s.getStopTime());
        if (pos == -1) { return false; }
        moveRight(pos, numStations - 1);
        trip[pos] = s;
        numStations++;
        if (s.getType() != Station.SERVICE) { numBoardings++; }
        return true;
    }

    public Station[] aboard() {
        Station[] result = new Station[numBoardings];
        int j = 0;
```

```
        for (int i = 0; i < numStations && j < numBoardings; i++) {
            if (trip[i].getType() != Station.SERVICE) {
                result[j] = trip[i];
                j++;
            }
        }
        return result;
    }
}
```

2. 2 points Let be an integer `a` $\geq$ 1. **You must:** implement a class (`static`) method that, using `'*'`, shows on the screen a figure composed of an isosceles rectangle triangle of height `a` and its symmetric image, opposed by their hypotenuse and with their basis separated by a blank space. For example, for `a = 4`, the method must show the following figure:

```
*       *
**     **
***   ***
**** ****
```

Solution:

```
/** Precondition: a > 1 */
public static void draw(int a) {
    int b = a * 2 + 1; // basis of figure to draw
    for (int i = 1; i <= a; i++) {
        for (int j = 1; j <= i; j++) { System.out.print('*'); }
        int blanks = b - i;
        for (int j = i; j < blanks; j++) { System.out.print(' '); }
        for (int j = 1; j <= i; j++) { System.out.print('*'); }
        System.out.println();
    }
}
```

3. 2 points **You must:** implement a class (`static`) method such that, given two arrays of `char` `a` and `b`, both them without repeated elements, returns the number of common characters. For example, if `a` is {`'C','T','A','G'`} and `b` is {`'T','U','C'`}, the method must return 2.

Solution:

```
/** Precondition: a and b without repeated elements */
public static int numCommonChar(char[] a, char[] b) {
    int commmon = 0;
    for (int i = 0; i < a.length; i++) {
        int j = 0;
        while (j < b.length && a[i] != b[j]) { j++; }
        if (j < b.length) { common++; }
    }
    return common;
}
```