IIP Second Partial - ETSInf
January 15th of 2016. Time: 2 hours and 30 minutes.

1. 6.5 points You have available the `Thermostat` class, that represents a temperature controller for a thermic device installed in a zone of a give space (house, office building, ...). Each thermostat is defined based on four data items: identifier (name of the zone where it is situated), mode (`COOL` for refrigeration, `HOT` for heating), current temperature of the zone, and comfort temperature desired by the user.

   This class (which is known from the previous partial) documentation summary is:

Field Summary

Fields

Modifier and Type	Field and Description
`static int`	`COOL` Constant that represents refrigeration mode for a thermostat
`static int`	`HOT` Constant that represents heating mode for a thermostat
`static int`	`T_IDEAL_COOL` Constant that represents ideal temperature for `COOL` mode in a thermostat
`static int`	`T_IDEAL_HOT` Constant that represents ideal temperature for `HOT` mode in a thermostat

Constructor Summary

Constructors

Constructor and Description
`Thermostat()` Creates a standard thermostat, with `COOL` mode, name `"living room"`, comfort temperature `T_IDEAL_COOL`, and current temperature as random value in `[20.0, 40.0[`
`Thermostat(int m, java.lang.String n, int tCo, double tCu)` Creates a thermostat with mode `m`, name `n`, comfort temperature `tCo`, and current temperature `tCu`

Method Summary

Methods

Modifier and Type	Method and Description
`int`	`differenceWithIdeal()` Returns an integer value which is: - Zero when comfort temperature is appropriate to the mode, i.e., greater than or equal to ideal in `COOL`, or lower than or equal to ideal in `HOT` mode - Absolute difference between comfort and ideal temperature in any other case
`java.lang.String`	`getName()` Returns the thermostat name

   You must: implement the `ThermostatManager` class, that represents the thermostats in a space, by using the attributes and methods defined below.

   Remember you have to use the constants for `Thermostat` and `ThermostatManager` whenever required.

a) (0.5 points) Attributes:

- `MAX_THERMS`, class (`static`) constant atttribute that represents the maximum number of thermostates in the space; its value is 15
- `numTherms`, integer in the interval `[0 - MAX_THERMS]` which represents the number of thermostats in the current moment
- `therms`, array of `Thermostat`, with length `MAX_THERMS`, to store the thermostats in the space in a given moment; they are placed in consecutive positions of the array, from 0 to `numTherms - 1` (both included)
- `noEfficients`, integer that represents the number of thermostates in the space that do not fulfil efficiency requirements, i.e., those whose comfort temperature is not appropriate for its working mode (as the `differenceWithIdeal` method in `Thermostat` class checks)

b) (0.5 points) Implement a default constructor (with no parameters) that creates a manager with 0 thermostats

c) (1 point) Write a method with header:

```
private int thermostatInZone(String zoneName)
```

such that, given the name of a zone in the space (`zoneName`), returns the position in the array where the thermostat with name equal to that zone name is present, or -1 if not present in the space

d) (1.5 points) Write a method with header:

```
public boolean install(Thermostat t)
```

such that adds thermostat `t` to the manager if no thermostat with the same name is present; in case a thermostat with the same name is present, `t` will replace it. The method must return `true` when installation or updating was successful, and `false` otherwise (no more thermostats can be added). Notice that attribute `noEfficients` must be properly updated, and that private method `thermostatInZone` must be used for searching for the thermostat `t` position by using its name

e) (1.5 points) Write a method with header:

```
public Thermostat highestDif()
```

such that returns the `Thermostat` with highest difference, in absolute value, between comfort and ideal temperatures, or `null` when no `Thermostat` is present

f) (1.5 points) Write a method with header:

```
public Thermostat[] thermostatNoEff()
```

such that returns an array of `Thermostat` with the non-efficient thermostats; this array length must be equal to the number of non-efficient thermostats, or 0 when no non-efficient thermostat is present

Solution:

```
/**
 * Class ThermostatManager: represents a device that manages all
 * thermostates of a space
 *
 * @author IIP Exam
 * @version Second Partial - Year 2015-2016
 */
public class ThermostatManager {
    /** Constant that represents the maximum number of
     *  thermostats in a space */
    public static final int MAX_THERMS = 15;
    // integer in [0..MAX_THERMS] that represents the number of
    // thermostats in the space in the current time
    private int numTherms;
    // array of Thermostat objects, with length MAX_THERMS, where
    // thermostats are sequentially stored in consecutive positions,
    // from 0 to numTherms - 1
    private Thermostat[] therms;
```

```
    // integer that represents the number of thermostats that
    // do not fulfill efficiency requirements
    private int noEfficients;

    /** Default constructor
     *  Empty manager (no thermostats) */
    public ThermostatManager() {
        therms = new Thermostat[MAX_THERMS];
        numTherms = 0;
        noEfficients = 0;
    }

    /** Checks if a thermostat is installed in a given zone of the space.
     *  @param zoneName String, name of the space zone.
     *  @return int, position in array therms or -1 if not present.
     */
    private int thermostatInZone(String zoneName) {
        int i = 0;
        while (i < numTherms && !therms[i].getName().equals(zoneName)) { i++; }
        if (i < numTherms) { return i; }
        else { return -1; }
    }

    /** Installs a new thermostat or updates it in the zone if previously installed.
     *  Returns true if successfully installed/updated, false when no more thermostats
     *  can be managed. Updates noEfficients if necessary.
     *  @param t Thermostat, thermostat to install/update.
     *  @return boolean.
     */
    public boolean install(Thermostat t) {
        boolean fits = true;
        int pos = thermostatInZone(t.getName());
        if (pos != -1) {
            if (therms[pos].differenceWithIdeal() != 0) { noEfficients--; }
            therms[pos] = t;
        }
        else if (numTherms < MAX_THERMS) { therms[numTherms++] = t; }
        else { fits = false; }
        if (fits && t.differenceWithIdeal() != 0) { noEfficients++; }
        return fits;
    }

    /** Returns Thermostat with highest absolute difference between comfort and
     *  ideal temperatures, or null if no thermostats present in the space.
     *  @return Thermostat.
     */
    public Thermostat highestDif() {
        Thermostat tMax = null;
        if (numTherms != 0) {
            tMax = therms[0];
            int difMax = tMax.differenceWithIdeal();
            for (int i = 1; i < numTherms; i++) {
                int dif = therms[i].differenceWithIdeal();
                if (dif > difMax) { tMax = therms[i]; difMax = dif; }
            }
        }
        return tMax;
    }

    /** Returns Thermostat array with the non-efficient thermostats of the space.
     *  Array length must be equal to the number of non-efficient thermostats,
     *  or 0 when no thermostat is present.
     *  @return Thermostat[].
     */
    public Thermostat[] thermostatNoEff() {
        Thermostat[] aux = new Thermostat[noEfficients];
        int k = 0;
        for (int i = 0; i < numTherms && k < noEfficients; i++) {
            if (therms[i].differenceWithIdeal() != 0) {
                aux[k] = therms[i];
                k++;
            }
        }
        return aux;
    }
}
```

2. 1.75 points Write a Java class (`static`) method that, given an integer `n` ≥ 2 as parameter, shows on the screen a drawing with `n` lines with two diagonals that join in the last line, over a background of `'-'` symbols, such that:
   - In each line, two `'V'` symbols must appear separated every line by a lower number of `'-'` symbols
   - In first line, first `'V'` is in the leftmost position and last `'V'` is in the rightmost position

For example, for `n`=5:

```
V--------V
-V------V-
--V----V--
---V--V---
----VV----
```

Solution:

```
/** Write on standard output a drawing with n lines (n >= 2),
 *  formed by two diagonals of 'V' such that join in the last line,
 *  over a rectangular background of '-'.
 */
public static void writeV(int n) {
    // blank1 number of '-' that must be written in each line before
    // first 'V' and after second one;
    // blank2 number of '-' that must be written in each line between
    // the two 'V';
    int blank1 = 0, blank2 = 2 * n - 2;
    while (blank1 < n) {
        for (int i = 1; i <= blank1; i++) { System.out.print('-'); }
        System.out.print('V');
        for (int i = 1; i <= blank2; i++) { System.out.print('-'); }
        System.out.print('V');
        for (int i = 1; i <= blank1; i++) { System.out.print('-'); }
        System.out.println();
        blank1++; blank2 = blank2 - 2;
    }
}
```

3. 1.75 points Write a Java class (`static`) method that, given an array of `double` as parameter, checks that its components with even index appear sorted in ascendent order. For example:

For

0	1	2	3	4	5
1.5	0.0	**3.0**	-1.0	**3.5**	2.0

and

0	1	2	3	4	5	6
3.0	0.0	**4.5**	-1.0	**6.5**	2.0	**8.5**

must return `true`.

For

0	1	2	3	4	5
1.5	0.0	**3.0**	-1.0	**1.5**	2.0

and

0	1	2	3	4	5	6
3.0	0.0	**1.0**	-1.0	**6.5**	2.0	**8.5**

must return `false`.

Solution:

```
/** Checks that elements with even index of a are sorted
 *  in ascendent order
 */
public static boolean inOrderEven(double[] a) {
    int i = 0;
    while (i < a.length - 2 && a[i] <= a[i + 2]) { i = i + 2; }
    return (i >= a.length - 2);
}
```