Recuperación del Segundo Parcial de IIP - ETSInf
Fecha: 28 de enero de 2014. Duración: 2:30 horas.

1. 6.5 puntos Se dispone de la clase `Sensor` que representa un sensor de temperatura conectado a un dispositivo térmico de manera que, además de medir la temperatura actual, puede aumentar y reducir la temperatura (medida en grados centígrados) de dicho dispositivo. Además, es posible definir la temperatura ideal a la que el usuario desea que esté la zona donde se encuentra el sensor. A continuación se muestra un resumen de su documentación:

Constructor Summary

Constructors

Constructor and Description
`Sensor(java.lang.String nombre, int tempIdeal)` Crea un `Sensor` con un nombre y una temperatura ideal.

Method Summary

Methods

Modifier and Type	Method and Description
`void`	`aumentaTempActual()` Aumenta la temperatura actual en un determinado número (aleatorio) de grados.
`boolean`	`equals(java.lang.Object o)` Comprueba si el `Sensor` en curso es igual a otro dado; i.e. si coinciden en nombre.
`java.lang.String`	`getNombre()`
`double`	`getTempActual()`
`double`	`getTempIdeal()`
`void`	`reduceTempActual()` Disminuye la temperatura actual en un determinado número (aleatorio) de grados.
`java.lang.String`	`toString()` Devuelve un `String` con la información del `Sensor` en curso.

Se pide: implementar la clase `CasaDomotica` que, como su nombre indica, representa una casa domótica en la que pueden existir múltiples sensores en diferentes ubicaciones. Se representa mediante las componentes (atributos y métodos) que se indican a continuación.

a) (0.5 puntos) Atributos:
   - `MAX_SENSORES`, una constante que representa el número máximo de sensores de una casa domótica: 100.
   - `numSensores`, un entero en el intervalo `[0..MAX_SENSORES]` que representa el número de sensores que tiene en cada momento la casa domótica.
   - `sensores`, un array de tipo base `Sensor`, de capacidad `MAX_SENSORES` y cuyas componentes se almacenan secuencialmente, en posiciones consecutivas desde la `0` hasta la `numSensores-1`.
   - `maxGradosDesviacion`, un valor decimal que representa la máxima desviación en grados que el usuario desea para los sensores de la casa domótica.
   - `numSensoresConDesviacion`, que representa el número de sensores de la casa domótica que tienen actualmente una desviación en grados, en valor absoluto, superior a `maxGradosDesviacion`.

b) (0.5 puntos) Un constructor general que, dado un valor de máxima desviación en grados, crea una casa domótica vacía, con 0 sensores, y el valor de máxima desviación en grados indicado.

c) (1 punto) Un método con perfil: `private boolean existeSensor(Sensor s)` que, dado un `Sensor s`, indica si existe o no dicho `Sensor` en la casa domótica.

d) (0.5 puntos) Un método con perfil: `private boolean sensorConDesviacion(Sensor s)` que, dado un `Sensor s`, devuelve `true` si el `Sensor` presenta una desviación de su temperatura actual frente a su temperatura ideal superior a la máxima desviación en grados definida para la casa domótica. En caso contrario, devuelve `false`.

e) (1 punto) Un método con perfil: `public int registraSensor(Sensor s)` que devuelve la posición en la que se ha añadido el `Sensor s` dado en el array de la casa domótica. Si `s` no cabe o ya está en el array, el método devuelve `-1` para advertir que no se ha podido añadir al array. Se debe usar el método privado `existeSensor(Sensor)` para buscar el `Sensor`. También se deberá usar el método privado `sensorConDesviacion(Sensor)` para actualizar, si es necesario, el atributo `numSensoresConDesviacion`.

f) (1 punto) Un método con perfil: `public Sensor[] sensoresConDesviacion()`
que devuelve un array de `Sensor` con los sensores que presentan desviación de su temperatura actual frente a su temperatura ideal, considerando la máxima desviación en grados definida para la casa domótica. La longitud de dicho array será igual al número de sensores que presentan dicha desviación, o 0 si no hay ninguno. Se debe usar el método `sensorConDesviacion(Sensor)`.

g) (1 punto) Un método con perfil: `public void ajustaTemperatura()`
que regula la temperatura de todos los `Sensor` que presentan desviación de su temperatura frente a su temperatura ideal de acuerdo a la desviación definida para la casa domótica. La regulación se hará para cada `Sensor` que presente desviación, usando los correspondientes métodos de la clase `Sensor` para ajustar la temperatura actual, hasta que ya no presente desviación.

h) (1 punto) Un método con perfil: `public Sensor sensorConMayorTemperatura()`
que devuelve el `Sensor` que mide la mayor temperatura actual en el momento de la invocación del método, o `null` si la casa domótica no tiene ningún `Sensor` registrado.

Solución:

CasaDomotica.java

```
/**
 * Representa una CasaDomotica, que incluye un array de Sensores
 * que obtienen la temperatura de ciertas zonas de la casa.
 */
public class CasaDomotica{
    private Sensor[] sensores;
    private int numSensores;
    private int numSensoresConDesviacion;
    private double maxGradosDesviacion;
    public static final int MAX_SENSORES = 100;

    public CasaDomotica(double maxGradosDesviacion) {
      sensores = new Sensor[MAX_SENSORES];
      numSensores = numSensoresConDesviacion = 0;
      this.maxGradosDesviacion = maxGradosDesviacion;
    }

    private boolean existeSensor(Sensor s) {
      int i;
      for (i=0; i<numSensores && !sensores[i].equals(s); i++);
      return i<numSensores;
    }

    private boolean sensorConDesviacion(Sensor s) {
      return Math.abs(s.getTempActual() - s.getTempIdeal()) > maxGradosDesviacion;
    }

    public int registraSensor(Sensor s) {
      int res = -1;
      if (!existeSensor(s) && numSensores<MAX_SENSORES){
        res = numSensores; sensores[numSensores++] = s;
        if (sensorConDesviacion(s)) numSensoresConDesviacion++;
      }
      return res;
    }
```

```
    public Sensor[] sensoresConDesviacion() {
        Sensor[] aux = new Sensor[numSensoresConDesviacion];
        int k = 0;
        for (int i=0; i<numSensores; i++)
            if (sensorConDesviacion(sensores[i])) {
                aux[k] = sensores[i];
                k++;
            }
        return aux;
    }

    public void ajustaTemperatura() {
      for (int i=0; i<numSensores; i++) {
         Sensor s = sensores[i];
         while (sensorConDesviacion(s))
           if (s.getTempActual() > s.getTempIdeal()) s.reduceTempActual();
           else s.aumentaTempActual();
      }
      numSensoresConDesviacion = 0;
    }

    public Sensor sensorConMayorTemperatura() {
      Sensor max = sensores[0];
      for (int i=0; i<numSensores; i++)
         if (sensores[i].getTempActual() > max.getTempActual()) max = sensores[i];
      return max;
    }
}
```

CasaDomotica.java

2. 1.5 puntos Dado un entero h en el rango [0..23], se pide un método público y estático que escriba en la salida estándar, hora a hora y minuto a minuto, todas las representaciones horarias en el formato de cinco caracteres `hh:mm`, desde la `00:00` hasta la correspondiente al último minuto de la hora h. Cada una de ellas se debe escribir en una línea diferente. Ejemplo: Si $h = 13$, se debería mostrar:

```
00:00
00:01
00:02
...
00:59
01:00
01:01
01:02
...
01:59
...
13:00
13:01
13:02
...
13:59
```

Nota: Por motivos de espacio en este enunciado, el ejemplo anterior se ha abreviado sustituyendo por puntos suspensivos buena parte de las representaciones que el método debe mostrar.

Solución:

Primera versión:

```
/** h está en el rango [0..23] */
public static void displayHoras(int h) {
   for (int i=0; i<=h; i++) {
      String display = i<10 ? "0"+i+":" : i+":";
      for (int j=0; j<=9; j++) System.out.println(display+"0"+j);
      for (int j=10; j<=59; j++) System.out.println(display+j);
   }
}
```

Segunda versión:

```
/** h está en el rango [0..23] */
public static void displayHoras(int h) {
    for (int i=0; i<=h; i++)
        for (int j=0; j<60; j++)
            System.out.printf("%02d:%02d\n", i, j);
}
```

3. 2 puntos **Se pide:**

a) Indicar qué muestra por pantalla el siguiente código y explicar qué hace en dos o tres líneas como máximo.

b) Indicar qué ocurriría si la condición del bucle interno en el método `funcion(int[][])` fuese `j<m.length`.

```
public class Traza {
    public static void main(String[] args) {
        int [][] a = { { 1, 2, 2 }, { 0, 1, 2 }, { 0, 0, 1 } };
        funcion( a );
        for (int i=0; i<a.length; i++) {
            for (int j=0; j<a.length; j++)
                System.out.printf( " %3d ", a[i][j] );
            System.out.println();
        }
    }
    public static void funcion(int[][] m ) {
        for (int i=0; i<m.length; i++)
            for (int j=0; j<i; j++)
                intercambia( m, i, j );
    }
    public static void intercambia(int[][] m, int a, int b) {
        int aux = m[a][b]; m[a][b] = m[b][a]; m[b][a] = aux;
    }
}
```

Solución:

a) Por pantalla se muestra:

```
1    0    0
2    1    0
2    2    1
```

Calcula la traspuesta de la matriz `a`.

b) Dejaría la matriz como estaba originalmente, pues realiza la traspuesta dos veces.