Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Septiembre 2006

Cuestiones (5 cuestiones, 5 puntos en total)

1) Se dispone de la siguiente clase enumerada. Escribir un método, perteneciente a una clase separada, que muestre en pantalla todos los valores de la clase enumerada, uno por línea. Observar que en la descripción de la clase, los puntos suspensivos nos indican que hay más valores definidos, además de los que se describen.

```
public enum Deporte
{
   futbol, baloncesto, ciclismo, natacion, ...
}
```

2) El siguiente pseudocódigo corresponde a un algoritmo (no óptimo) que calcula las combinaciones sin repetición de *n* elementos tomados de *m* en *m*

```
método combinaciones (m,n  enteros) retorna entero
    si  m>n  ó si  n<1 ó si  m<1 lanza la excepción Incorrecto
    si  m coincide con n retorna 1
    en otros casos:
       entero var=1;
       lazo para cada i decrementándose desde n hasta m+1
         var=var*i
       fin de lazo
       lazo para cada i decrementándose desde n-m hasta 2
         var=var/i;
       fin de lazo
       retorna var
fin de método
```

Escribir este algoritmo como un método estático en Java.

3) Indicar cuál es el ritmo de crecimiento del tiempo de ejecución del algoritmo anterior para el caso $m=n/2$, utilizando la notación $O(n)$.

4) Haz un método Java que permita escribir en un fichero de texto llamado `letras.txt` las letras de un `String` que se le pasa como parámetro, una letra por línea.

5) Indica qué orden podemos dar a un intérprete de órdenes de un sistema operativo tipo Linux/Unix para copiar todos los ficheros y subdirectorios de nuestro directorio de trabajo al directorio `copias`, que está situado dentro del árbol de ficheros en la ruta `/usr/local`

Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Septiembre 2006

Problema (5 puntos)

Se desea hacer un sistema para controlar una planta compuesta por un conjunto de hasta 20 paneles solares que producen electricidad. Cada panel solar individual se representa en el computador por medio de un objeto de la clase `PanelSolar`, cuya interfaz es:

```
/**
 * Clase que representa un panel solar
 */
public class PanelSolar
{

    /**
     * Constructor al que se le pasa el identificador del panel
     */
    public PanelSolar(String id) {...}

    /**
     * Retorna el identificador del panel
     */
    public String id() {...}

    /**
     * Método para alinear el panel; retorna true si se ha conseguido
     * alinear con el sol, y false en caso contrario
     */
    public boolean panelAlineado() {...}

    /**
     * Retorna el acimut del panel, en grados, que nos indica
     * hacia dónde apunta en la dirección horizontal.
     * El Sur se representa con el valor 180 grados.
     * Lanza Averiado si el panel está averiado
     */
    public double acimut() throws Averiado{...}

    /**
     * Retorna la elevación del panel, en grados, sobre el horizonte.
     * Cuando el panel apunta al horizonte el valor es 0 grados
     * Lanza averiado si el panel está averiado
     */
    public double elevacion() throws Averiado {...}

    /**
     * Retorna la potencia eléctrica, en vatios, que el panel
     * está produciendo en este instante
     */
    public double potencia() {...}
}
```

Lo que se pide es implementar la clase `PlantaSolar`, que debe responder a la siguiente interfaz:

```
public class PlantaSolar
{
    public void anadePanel(PanelSolar p) throws NoCabe, YaExiste {...}
    public void alinea() {...}
}
```

La clase debe contener una constante pública y estática, llamada `max`, que represente el número máximo de paneles solares, que es 20.

La clase debe contener los siguientes atributos privados

- `panel`: un array de hasta `max` objetos de la clase `PanelSolar`
- `num`: un entero que indica cuántos paneles solares tenemos guardados en el array `panel`; los paneles válidos serán por tanto los contenidos en las casillas de 0 a `num`-1; valor inicial 0
- `acimutMedio`: un número real que indica el acimut medio en grados; valor inicial 0.0
- `elevMedia`: un número real que indica la elevación media en grados; valor inicial 0.0
- `potenciaTotal`: un número real que indica la potencia total en vatios; valor inicial 0.0

Los métodos de la clase `PlantaSolar` deben hacer lo siguiente:

- `anadePanel`: Añade un nuevo panel solar, que se pasa como parámetro, siguiendo los siguientes pasos. En primer lugar, si no caben más paneles, lanza `NoCabe`. En segundo lugar comprueba si ya existe un panel con el mismo identificador, y en ese caso lanza `YaExiste` (para ello habrá que recorrer los paneles que tenemos almacenados, comparando sus identificadores con el de `p`). Por último, si no hubo errores, añade el panel `p` en la primera casilla libre del array de paneles, e incrementa `num`.
- `alinea`: Alinea todos los paneles de la planta y, con los que resulten correctamente alineados y no estén averiados, calcula el acimut y elevacion medios, así como la potencia total, almacenándolos en los respectivos atributos. Para ello, después de inicializar a cero el atributo de la potencia total, un par de variables para guardar en ellas respectivamente las sumas de los acimuts y elevaciones de los paneles (para calcular luego la media), y un contador de los paneles solares que están correctos, recorre los paneles solares que tenemos guardados en el array `panel`, y con cada uno de ellos hace lo siguiente:
  - alinea el panel llamando a `panelAlineado` y guarda el resultado obtenido, que indica si el panel quedó bien alineado
  - añade a la potencia total la potencia de este panel solar
  - si el panel resultó bien alineado, obtener su acimut y elevación y, si no se lanza `Averiado` añadirlos respectivamente a la suma de acimut y elevación e incrementar el contador de paneles correctos; si se lanza `Averiado`, no hacer nada y seguir con el siguiente panel

  al finalizar, calcular el acimut medio y la elevación media (como la suma entre el contador de paneles correctos) y almacenarlos en los respectivos atributos

Las excepciones están definidas en clases aparte de la forma:

```
public class Averiado extends Exception{}
public class NoCabe extends Exception{}
public class YaExiste extends Exception{}
```

Nota: Cada parte de la planta solar se valorará según su dificultad, del siguiente modo:

- constante y atributos: 15%
- `anadePanel`: 35%
- `alinea`: 60%