Examen de Fundamentos de Computadores y Lenguajes

2º Examen Parcial. Junio 2001

Cuestiones (5 cuestiones, 5 puntos en total)

1) Dada la clase que se muestra abajo y que permite almacenar las tres coordenadas de un vector en el espacio, añadirle un método que retorne el módulo del vector resultante de multiplicar el vector almacenado en el objeto (*x*,*y*,*z*) por un escalar. El escalar se pasará como parámetro. El módulo de un vector es $\sqrt{x^2+y^2+z^2}$.

```
public class Vector3D {
   public double x,y,z;
}
```

2) Escribir un fragmento de programa que, dado un array `a` de enteros (del tipo `int[]`) ya creado, cree un nuevo array del mismo tipo y tamaño llamado `b` y copie en él todos los datos del array original `a`.

3) Se dispone de la clase `Ecuacion` que permite almacenar una ecuación de segundo grado y define las siguientes operaciones:

```
public  Ecuacion (double a, double b, double c)
public double raiz1() throws IllegalArgumentException
public double raiz2() throws IllegalArgumentException {
```

El constructor permite definir los coeficientes (*a*,*b*,*c*) de la ecuación. Las operaciones `raiz1`() y `raiz2`() devuelven respectivamente cada una de las dos raíces reales de la ecuación, si las hay, o arrojan `IllegalArgumentException` si las raíces son complejas.

El siguiente fragmento de programa usa la clase `Ecuacion` anterior para obtener la solución de una ecuación y mostrarla en la pantalla.

```
Ecuacion ec=new Ecuacion(a,b,c);
System.out.println("R1: "+ec.raiz1()+"  R2: "+ec.raiz2());
```

Se pide modificar ese fragmento de programa para que si se lanzase la excepción `IllegalArgumentException`, se trate la excepción y se ponga en la pantalla el texto "Las raíces son complejas".

4) Indica brevemente (máximo 8 líneas) qué es y para qué sirve el ocultamiento de información en la programación de clases.

5) Entre los algoritmos de ordenación de tablas (arrays) estudiados en clase, tanto el método de la burbuja como el método de ordenación por selección, tienen una función de eficacia del tipo O(N^2), siendo *N* el tamaño de la tabla. También es sabido que el método de ordenación rápida o *quicksort* tiene una eficacia en el caso promedio de O(*N log N*). Sin embargo, en ocasiones es posible, por las condiciones iniciales del problema, dar soluciones más eficaces. Este es el caso en la situación que describimos a continuación.

Se desea ordenar una tabla *a*[1..*N*] de cifras binarias, *N* >0. Es decir, para cada índice de la tabla, *i*, 1 ≤ *i* ≤ *N* la componente *i*-ésima, *a*[*i*] es o bien un *cero* o bien un *uno* y ningún

otro valor. Para ordenar una tabla como la descrita basta diseñar un bucle controlado por dos variables i, j, $1 \leq i,\ j \leq N$ de tal manera que tras cada iteración todos las componentes de la tabla de índice menor que el valor i contienen un cero y todas los componentes de índice mayor que j contienen un uno. Inicialmente, i se hace igual a 1, y j igual a N.

Para diseñar el cuerpo del bucle basta hacer el siguiente análisis. Si la componente $a[i]$ contiene el valor cero se incrementa el contador i; si la componente $a[j]$ contiene el valor uno se decrementa el contador j. Si ninguna de las dos condiciones era cierta deberán intercambiarse los valores de ambas componentes. El bucle acaba cuando $i \geq j$.

Se pide especificar, diseñar en pseudocódigo y analizar la eficacia del método de ordenación descrito anteriormente (observar que **no** se pide la implementación en Java).

Examen de Fundamentos de Computadores y Lenguajes

2º Examen parcial. Junio 2001

Problema (5 puntos)

Se desea construir una aplicación para procesar y mostrar datos de varias estaciones meteorológicas situadas en un Cantabria. Estas estaciones están conectadas a Internet y pueden suministrar datos bajo petición desde un computador central. Esta conexión está resuelta en la clase `EstacionMet`, que se supone ya realizada, y cuya interfaz aparece a continuación

```
public class EstacionMet {

   /** Excepción que lanza conecta() si no se puede establecer
       la conexión con la estación */
   public class NoHayConexion extends Exception {}

   /** Constructor. Requiere el nombre de la estación */
   public EstacionMet(String nombre) {...}

   /** Retorna el nombre de la estación */
   public String nombre() {...}

   /** Establece la conexión con la estación remota.
       Requiere la dirección de Internet  */
   public void conecta(String direcIP) throws NoHayConexion {...}

   /** Retorna la temperatura en grados centigrados */
   public double temperatura() {...}

   /** Retorna la presión en milibares */
   public double presion() {..}
}
```

Los atributos de `EstacionMet` son privados. La descripción de cada operación aparece en los comentarios de documentación.

Se pide utilizando esta clase hacer un programa que cree varios objetos de la clase `EstacionMet`, se conecte a las estaciones remotas, tome medidas de temperatura, realice algunos cálculos con ellas, y muestre información en una pantalla. Los nombres y direcciones de las estaciones se muestran en la siguiente tabla:

Nombre	Dirección IP
reinosa	123.548.22.13
santander	123.548.22.14
laredo	123.548.22.15
comillas	123.548.22.16
potes	123.548.22.17

Los pasos que debe realizar esta aplicación son los siguientes (*Nota*: cada apartado se valorará en función de su dificultad):

a) *Declaraciones de los siguientes datos*:
   - un array de strings con los nombres de las estaciones
   - un array de strings con las direcciones de IP
   - un array de objetos del tipo `EstacionMet`, para las estaciones
   - un array de números reales, para almacenar las temperaturas de cada estación
   - un entero para el número de estaciones con conexión
   - tres números reales para almacenar la temperatura media, mínima y máxima
   - dos strings para almacenar los nombres de las estaciones donde se han registrado las temperaturas mínima y máxima

b) *Creación y conexión de las estaciones*: mediante un lazo crear cada una de las estaciones y conectarla llamando a `conecta`(). Si se arroja `EstacionMet.NoHayConexion` mostrar un mensaje de error. Al final del lazo, el entero con el número de estaciones con conexión deberá estar a su valor correcto.

c) *Leer los datos*: mediante un lazo, recorrer todas las estaciones que tienen conexión y leer su temperatura almacenándola en el array de temperaturas. Ignorar las estaciones para las que no se pudo establecer la conexión.

d) *Procesar datos*: mediante un lazo en el que recorremos el array de temperaturas, calcular las temperaturas media, máxima y mínima. Además, almacenar los nombres de las estaciones en las que se han registrado las temperaturas mínima y máxima en las variables creadas al efecto.

e) *Mostrar información*: Crear un objeto de la clase `Escritura` o similar, para escribir información en una ventana en pantalla. Mediante un lazo mostrar las temperaturas de todas las estaciones con conexión, junto a sus nombres, en la ventana creada. Después mostrar la temperatura media, máxima y mínima, las dos últimas con las estaciones respectivas donde se registraron esas temperaturas. Cuando el usuario cierre la ventana (es decir después de la operación `Escritura.espera`() o similar), volver al paso c).