Solucion al Examen de Fundamentos de Computadores y Lenguajes

Examen Parcial. Junio 2006

Cuestiones (5 cuestiones, 5 puntos en total)

1) Escribir una clase que permita almacenar una lista de jugadores y sus correspondientes equipos. Los jugadores se representan por su nombre (un `String`), mientras que los equipos son objetos de la clase `Equipo`. La clase dispone de tres atributos privados:
   - `jugador`: array de strings que representan nombres de jugadores
   - `pertenece`: array de equipos
   - `num`: número actual de jugadores almacenados

   Ambos arrays se organizarán de modo que `pertenece[i]` es el equipo al que pertenece `jugador[i]`

   La clase debe tener un constructor que pone `num` a cero y crea los dos arrays del mismo tamaño, dato que se pasa como parámetro.

```java
/**
 * Clase que almacena una lista de jugadores y sus equipos
 */
public class ListaJugadores
{
    // atributos privados
    private String[] jugador;
    private Equipo[] pertenece;
    private int num;

    /**
     * Constructor al que se le pasa el tamaño de la lista
     */
    public ListaJugadores(int tamano)
    {
        jugador = new String[tamano];
        pertenece = new Equipo[tamano];
        num=0;
    }
}
```

2) Añadir a la clase anterior un método que sirve para añadir un jugador y su correspondiente equipo a la lista. Ambos datos se pasan como parámetros. Si hay espacio, el nuevo jugador y equipo se insertan en la casilla `num` de ambos arrays, y luego se incrementa `num`. Si no hay espacio en los arrays para hacer esto, se lanza la excepción `NoCabe`, que está declarada en una clase aparte como

```java
public class NoCabe extends Exception {}
```

```java
    /** Añade un jugador dado su nombre y su equipo */
    public void anade(String nombre, Equipo equipo) throws NoCabe
    {
        if (num<jugador.length) {
            jugador[num]=nombre;
            pertenece[num]=equipo;
            num++;
        } else {
            throw new NoCabe();
        }
    }
```

3) Se dispone de la clase que se indica a continuación y que representa una matriz de números reales.

```
public class Matriz
{
   // atributos privados
   private double mat[][];
   ...
}
```

Escribir un método de esta clase que permita trasponer la matriz si ésta es cuadrada, intercambiando sus filas por sus columnas. Hacerlo a partir del siguiente pseudocódigo.

```
método trasponer() lanza NoCuadrada
   si no coincide mat.filas  con mat.columnas
      lanzar excepción NoCuadrada
   fin si
   para todo i desde 0 hasta  mat.filas-2 lazo
       para todo j desde i+1 hasta  mat.columnas-1 lazo
           intercambiar mat[i][j] con mat[j][i];
       fin lazo
   fin lazo
fin trasponer
```

Nota: `mat.filas` representa en el pseudocódigo el número de filas de `mat`, y `mat.columnas` el de columnas. Usar en Java el atributo `length` (respectivamente `mat.length` y `mat[0].length`).

```
public void trasponer () throws NoCuadrada
{
    if (mat.length!=mat[0].length) {
        throw new NoCuadrada();
    }
    for (int i=0; i<=mat.length-2; i++) {
        for (int j=i+1; j<=mat[0].length-1; j++) {
            double temp=mat[i][j];
            mat[i][j]=mat[j][i];
            mat[j][i]=temp;
         }
     }
}
```

---

4) Indicar razonadamente el ritmo de crecimiento del tiempo de ejecución del algoritmo anterior mediante la notación $O(n)$.

En este caso, el tamaño del problema, n, es el número de filas de la matriz.

El método dispone de dos instrucciones en el nivel más externo: una instrucción condicional `if`, y un bucle `for`.

La instrucción `if` contiene una instrucción simple ($O(1)$) y por tanto es $O(1)$.

Para analizar el bucle `for` externo, multiplicamos el número de veces que se hace (n-1) por lo que tarda lo que tiene dentro, que es otro bucle `for`. Éste se hace un número de veces igual a n-i-1, que en el peor caso es n-1 veces. En su interior hay tres instrucciones simples, que por la regla de las sumas son $O(1)$. Por tanto, el bucle `for` externo tiene un ritmo de crecimiento que es $O(n\text{-}1)O(n\text{-}1)O(1)$, que por la regla de los productos es $O(n^2)$

El tiempo total es por tanto $O(1)+O(n^2)$, y por la regla de las sumas es $O(n^2)$.

---

5) Se dispone de la clase enumerada que se muestra a continuación. Escribir un método estático para una clase aparte, con un parámetro del tipo `String`. El método debe invocar a `construye()` con el `String` que se le ha pasado como parámetro, y tratar la excepción `Incorrecto` de modo que se ponga en pantalla un mensaje de error en el que se incluya el valor del parámetro (ej., "El texto xxx es incorrecto", siendo xxx el valor del parámetro).

```
public enum DiaSemana
{
    monday, tuesday, wednesday, thursday, friday, saturday, sunday;

   public static DiaSemana construye(String texto) throws Incorrecto
    {
       ...
    }
}
```

la excepción `Incorrecto` está declarada en una clase aparte como

```
public class Incorrecto extends Exception {}
```

```
public static DiaSemana construyeTratado (String texto) {
    try {
        return DiaSemana.construye(texto);
    } catch (Incorrecto e) {
        System.out.println("El texto "+texto+" es incorrecto");
        return null;
    }
}
```

Examen de Fundamentos de Computadores y Lenguajes

Examen Parcial. Junio 2006

Problema (5 puntos)

Se desea hacer un programa para gestionar los datos de la estadística de un vivero de árboles. En el vivero se plantan árboles en tres tipos de tierra, para experimentar cuál es la mejor tierra para cada especie de árbol.

El tipo de tierra se representa con un tipo enumerado:

```
public enum Tierra
{
   normal, silicea, calcarea
}
```

Para cada especie de árbol, se guardan sus datos en la clase `EstadisticaArbol` que tiene la siguiente interfaz

```
public class EstadisticaArbol
{
  /**
   * Constructor al que se le pasa el nombre de la especie.
   * Pone los datos a cero
   */
  public EstadisticaArbol(String especie)  {...}

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

  /**
   * anade números de árboles plantados y crecidos,
   * para el tipo de tierra indicado
   */
  public void anade(int plantados, int crecidos, Tierra tipoTierra) {...}

  /**
   * Retorna el numero de arboles plantados para el tipo de tierra indicado
   */
  public int plantados(Tierra tipoTierra) {...}

  /**
   * Retorna el numero de arboles que han crecido,
   * para el tipo de tierra indicado
   */
  public int crecidos(Tierra tipoTierra) {...}
}
```

Los datos del vivero se guardan en la clase `Vivero`, cuya interfaz es:

```
public class Vivero
{
  /**
   * Busca el arbol cuyo nombre se indica y retorna sus datos
   * estadisticos. Si no lo encuentra, retorna null
   */
  public EstadisticaArbol busca (String especieArbol) {...}

}
```

Lo que se pide es escribir la clase `DatosVivero`, cuya interfaz se muestra a continuación

```
public class DatosVivero
{
  public DatosVivero(Vivero v) {...}

  public double conExito(String[] especie, Tierra tipoTierra)
     throws NoEncontrado
  {...}

  public Tierra mejorTierra(String especie) throws NoEncontrado
  {...}

  public void escribeEnFichero(String nombreFichero, String especie)
  {...}
}
```

La clase debe tener un atributo que es un objeto de la clase vivero. También debe tener los siguientes métodos:

- *Constructor*. Se le pasa el vivero como parámetro y lo copia en el atributo
- `conExito`: Retorna el porcentaje de arboles con éxito (crecidos*100/plantados) para el tipo de tierra indicado, dada una lista de nombres de especies de arboles. La lista es un array de Strings. Si alguno de los árboles no se encuentra, lanza `NoEncontrado`.
- `mejorTierra`: Retorna el mejor tipo de tierra (la que tiene un porcentaje de éxito mayor) para la especie indicada. Si no se encuentra, lanza `NoEncontrado`.
- `escribeEnFichero`: Escribe en un fichero de texto cuyo nombre se le pasa como parámetro todos los datos de la especie cuyo nombre se pasa también como parámetro. Pone un dato por línea, anteponiendo un breve texto descriptivo. En la ultima línea pone el mejor tipo de tierra para esa especie. Si no se encuentra la especie, pone en el fichero el nombre de la especie y un mensaje de error.

La excepción `NoEncontrado` está definida en una clase escrita aparte de la forma:

```
public class NoEncontrado extends Exception  {}
```

Nota: cada parte de la clase `DatosVivero` se valorará según su dificultad.

```
import java.io.*;
/**
 * Clase con operaciones para explotar los datos de un vivero
 * de arboles
 */
public class DatosVivero
{
    // atributos privados
    private Vivero v;

    /**
     * Constructor al que se le pasa el vivero
     */
    public DatosVivero(Vivero v)
    {
        this.v=v;
    }
```

```java
/**
 * Retorna el porcentaje de arboles con exito
 * (crecidos*100/plantados)para el tipo de tierra
 * indicado, dada una lista de nombres de especies de arboles
 * Si alguno de los árboles no se encuentra, lanza NoEncontrado
 */
public double conExito(String[] especie, Tierra tipoTierra)
   throws NoEncontrado
{
   int plantados=0,crecidos=0;
   // recorre el array especie
   for (String esp:especie) {
       // busca la especie en en vivero
       EstadisticaArbol a=v.busca(esp);
       if (a==null) {
           throw new NoEncontrado();
       } else {
            // añade los plantados y crecidos al total
            plantados=plantados+a.plantados(tipoTierra);
            crecidos=crecidos+a.crecidos(tipoTierra);
       }
   }
   // hace el calculo y lo retorna
   return crecidos*100.0/plantados;
}

/**
 * Retorna el mejor tipo de tierra (la que tiene un porcentaje de éxito
* mayor) para la especie indicada. Si no se encuentra, lanza NoEncontrado
 */
public Tierra mejorTierra(String especie) throws NoEncontrado  {
    // Obtener la estadistica
    EstadisticaArbol a=v.busca(especie);
    if (a==null) {
        throw new NoEncontrado();
    }
    // calcular tasas de exito para cada tipo de tierra
    double exitoNormal=a.crecidos(Tierra.normal)*100.0/
                       a.plantados(Tierra.normal);
    double exitoSilicea=a.crecidos(Tierra.silicea)*100.0/
                       a.plantados(Tierra.silicea);
    double exitoCalcarea=a.crecidos(Tierra.calcarea)*100.0/
                       a.plantados(Tierra.calcarea);
    // calcular el maximo
    if (exitoNormal>exitoSilicea) {
        // el maximo esta entre normal y calcarea
        if (exitoNormal>exitoCalcarea) {
            return Tierra.normal;
        } else {
            return Tierra.calcarea;
        }
    } else {
        // el maximo esta entre silicea y calcarea
        if (exitoSilicea>exitoCalcarea) {
            return Tierra.silicea;
        } else {
            return Tierra.calcarea;
        }
    }
}

/**
 * Escribe en un fichero de texto cuyo nombre se le pasa
 * como parámetro todos los datos de la especie cuyo nombre
 * se pasa también como parámetro. Pone un dato por línea,
 * anteponiendo un breve texto descriptivo. En la ultima línea pone
 * el mejor tipo de tierra para esa especie.
 * Si no se encuentra la especie, pone en el fichero el
 * nombre de la especie y un mensaje de error.
 */
```

```
    public void escribeEnFichero(String nombreFichero, String especie)
    {
        try {
            // Abrir el fichero
            FileWriter f=new FileWriter(nombreFichero);
            PrintWriter p=new PrintWriter(f);
            // Obtener la estadistica
            EstadisticaArbol a=v.busca(especie);
            if (a==null) {
               // Escribir mensaje error
               p.println("La especie "+especie+" no se encuentra");
            } else {
               // Escribir los datos en el fichero
               p.println("Especie: "+especie);
               p.println("Plantados en t. normal: "+
                         a.plantados(Tierra.normal));
               p.println("Plantados en t. silicea: "+
                         a.plantados(Tierra.silicea));
               p.println("Plantados en t. calcarea: "+
                         a.plantados(Tierra.calcarea));
               p.println("Crecidos en t. normal: "+
                         a.crecidos(Tierra.normal));
               p.println("Crecidos en t. silicea: "+
                         a.crecidos(Tierra.silicea));
               p.println("Crecidos en t. calcarea: "+
                         a.crecidos(Tierra.calcarea));
               p.println("Mejor tierra: "+mejorTierra(especie));
            }
            p.close();
        } catch (Exception e) {
            System.out.println("Error imprevisto: "+e);
        }
    }
}
```