Soluciones al Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Septiembre 2005

Cuestiones (5 cuestiones, 5 puntos en total)

1) Crear una clase para almacenar los datos de una partícula subatómica medidos en un experimento. Los atributos de la clase serán privados y representan el nombre de la partícula (texto), la masa en Kg, y 10 tiempos de vida medidos en segundos. Estos últimos se almacenarán en un array.

   La clase tendrá un constructor al que se le pasan como parámetros el nombre de la partícula, su masa, y su vida media. Los dos primeros se guardarán en los respectivos atributos. El tercero se guardará en cada una de las casillas del array de tiempos de vida.

```
public class Particula
{
   private String nombre;
   private double masa; //Kg
   private double[] tiempoVida; // seg

   **
    * Constructor
    */
   public Particula(String nombre, double masa,  double tiempoMedio)
   {
       this.nombre=nombre;
       this.masa=masa;
       tiempoVida=new double[10];
       for (int i=0; i<10; i++) {
          tiempoVida[i]=tiempoMedio;
       }
   }

}
```

2) Se dispone de la clase `Vector` indicada abajo, cuyos objetos permiten guardar un vector de números reales. Escribir un método estático de una clase separada al que se le pase como parámetro un objeto de la clase `Vector`, y retorne la media de todos sus valores reales. Si el número de elementos del vector es cero, deberá lanzar la excepción `NoExiste`, compilada aparte y declarada como se indica.

```
public class Vector {

   /**
    * Constructor al que se le pasa el máximo tamaño del vector
    */
   public Vector(int numMax){...}

   /**
    * Retorna el elemento cuyo índice se pasa como parámetro.
    * Los elementos se numeran empezando en uno
    * Lanza NoExiste si el índice es de un elemento inexistente
    */
   public double elemento(int indice) throws NoExiste {...}

   /**
    * Retorna el número actual de elementos del vector
```

```
    */
    public int numElementos() {...}
}

public class NoExiste extends Exception {}
```

```
    public static double media(Vector v) throws NoExiste {
       int num=v.numElementos();
       if (num==0) {
           throw new NoExiste();
       }
       double suma=0.0;
       try {
          for (int i=0; i<num; i++) {
              suma=suma+v.elemento(i);
          }
       } catch (NoExiste e) {
          System.out.println("error inesperado");
       }
       return suma/num;
    }
```

---

3) Indicar qué órdenes habría que dar a un intérprete de órdenes UNIX para copiar tres ficheros denominados `Uno.java`, `Dos.java`, y `Tres.java` situados en el directorio `/usr/local/proyecto1` poniendo las copias en el directorio de trabajo, que es `/home/usr31`. Asimismo se deberán cambiar los nombres de las copias para que estén en minúsculas.

```
cp /usr/local/proyecto1/Uno.java uno.java
cp /usr/local/proyecto1/Dos.java dos.java
cp /usr/local/proyecto1/Tres.java tres.java
```

---

4) Indicar utilizando la notación $O(n)$, siendo n el tamaño de la lista, el tiempo de ejecución del algoritmo cuyo pseudocódigo se indica a continuación. ¿Cuál será el tiempo en el mejor caso? ¿Y en el peor?

```
algoritmo buscar nombre en lista indicando si está o no
   retorna true si lo encuentra y falso si no
   prueba con mayúsculas y minúsculas
comienzo algoritmo
   pasar nombre a minúsculas
   lazo para i desde 1 hasta el número de valores de la lista
      si nombre coincide con valor i de la lista
        retorna true
      fin de si
   fin lazo
   pasar nombre a mayúsculas
   lazo para i desde 1 hasta el número de valores de la lista
      si nombre coincide con valor i de la lista
        retorna true
      fin de si
   fin lazo
```

```
   retorna falso
fin algoritmo
```

El algoritmo tiene estas cinco partes:

- *pasar el nombre a minúsculas*: Aunque esta operación debe hacerse con totas las letras del nombre, en principio la longitud del nombre no depende del número de valores de la lista (n), por lo que es de tiempo de ejecución constante, O(1)
- *primer lazo, para todos los valores de la lista*: Aunque el lazo puede acabar prematuramente si se encuentra el nombre, en el peor caso el lazo se hace n veces. En su interior hay una instrucción condicional, que en su interior contiene una instrucción simple (O(1)). Por tanto, el tiempo de esta parte será nO(1), es decir, O(n).
- *pasar el nombre a mayúsculas*: Igual que en el paso a minúsculas, es O(1)
- *segundo lazo, para todos los valores de la lista*: el razonamiento es idéntico al del lazo anterior, O(n)
- *retornar falso*: es una instrucción simple, O(1)

Sumando todos estos tiempos, por la regla de las sumas, se obtiene una dependencia O(n) en el peor caso.

En el caso mejor, se encuentra el nombre en el primer valor de la lista, por lo que se habrá ejecutado un número fijo de instrucciones: O(1).

---

5) Escribir el método `escribeFichero` de la clase indicada a continuación, de modo que almacene en el fichero de texto cuyo nombre se indica como parámetro los dos atributos de la clase, uno por línea, precedidos cada uno de ellos de un texto que indique el nombre del atributo, y seguidos de un espacio en blanco y un texto que indique las unidades.

```
import java.io.*;
public class EstacionMeteo {

   private double temperatura; // grados Kelvin
   private double presionAtmosferica; // mm de mercurio

   public void escribeFichero (String nombreFichero) ...
}
```

```
public void escribeFichero (String nombreFichero)
{
   try {
      FileWriter fileout = new FileWriter(nombreFichero);
      PrintWriter out = new PrintWriter(fileout);
      out.println("Temperatura: "+temperatura+" K");
      out.println("Presion Atmosf.: "+presionAtmosferica+" mm Hg");
      out.close();
   } catch (Exception e) {
       System.out.println("Error "+e);
   }
}
```

Soluciones al Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Septiembre 2005

Problema (5 puntos)

Se desea hacer parte de un programa que sea capaz de almacenar y mostrar imágenes de un experimento científico. Las imágenes se almacenan con indicación de su nombre y de la fecha en que se tomaron. Las fechas se manipulan mediante objetos de la clase `Fecha`, que contiene un constructor y métodos para comparar fechas, que ya está implementada, y cuya especificación es:

```
public class Fecha {

    /**
     * Constructor al que se le pasa el dia, mes y año.
     * Lanza NoValida si la fecha resultante es incorrecta
     */
    public Fecha(int dia, int mes, int agno) throws NoValida {...}

    /**
     * a mayor o igual que b
     */
    public static boolean mayorOIgualQue(Fecha a, Fecha b) {...}

    /**
     * a menor o igual que b
     */
    public static boolean menorOIgualQue(Fecha a, Fecha b) {...}

}
```

Las imágenes se guardan en objetos de la clase `Imagen`, ya implementada, y cuya especificación es:

```
/**
 * La clase Imagen sirve para almacenar una imagen en color
 * que se puede pintar en una pantalla. Tiene asociado un nombre y fecha
 */
public class Imagen
{

    /**
     * Constructor al que se le pasa el nombre de la imagen y la fecha
     * en que se tomó
     */
    public Imagen(String nombre, Fecha dia) {...}

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

    /**
     * Retorna la fecha de la imagen
     */
    public Fecha dia() {...}

    /**
     * Pinta la imagen en una nueva ventana en la pantalla
     */
    public void pinta() {...}

 }
```

La excepción `NoValida` está declarada de la forma indicada a continuación. Se muestran también otras excepciones que usaremos más adelante, cada una de ellas definida como una clase aparte:

```
public class NoValida extends Exception {}

public class NoCabe extends Exception {}

public class NoExiste extends Exception {}
```

Lo que se pide es implementar la clase `Catalogo`, **con excepción de su segundo constructor**. Los objetos de esta clase sirven para almacenar un catálogo de imágenes de satélite. Presentan operaciones para obtener imágenes individuales y grupos de imágenes seleccionados por su fecha. La especificación de esta clase es:

```
public class Catalogo {

    /**
     * Constructor que crea el catálogo vacío, con un máximo de fotos
     * igual a numMax
     */
    public Catalogo(int numMax) {...}

    /**
     * Crea el catálogo leyendolo del fichero cuyo nombre se indica
     */
    public Catalogo(String nombreFichero) throws NoExiste {...}

    /**
     * Retorna el número actual de imágenes
     */
    public int numElementos() {...}

    /**
     * La primera vez retorna la primera imagen almacenada
     * Luego retorna la imagen siguiente a la ultima retornada con este metodo
     * Despues de la ultima se retorna la primera imagen
     * Lanza NoExiste si no hay ninguna imagen
     */
    public Imagen proximaImagen() throws NoExiste {...}

    /**
     * Inserta una imagen al final del catalogo.
     * Lanza NoCabe si no hay espacio
     */
    public void inserta(Imagen foto) throws NoCabe{...}

    /**
     * Filtra las imagenes retornando un catalogo que tiene las imagenes del
     * original cuyas fechas estan en el rango de fechas [desde,hasta]
     */
    public Catalogo filtraPorFechas(Fecha desde, Fecha hasta) {...}
}
```

La clase debe declarar tres atributos privados:

- Un array de objetos de la clase `Imagen` para guardar imágenes. Usaremos una parte del array para almacenar las imágenes, y otra parte quedará vacía.
- Un número entero que indica cuántas imágenes hay actualmente almacenadas en el array. Lo llamaremos `num`.
- Un número entero que indica cuál es la próxima imagen que devolverá `proximaImagen`(). Inicialmente se pone a cero. Lo llamaremos `proximo`

Los métodos que se piden deben hacer lo siguiente:

- Primer constructor: crea el array de imágenes con el tamaño indicado, y pone el número actual de imágenes (`num`) a cero.
- `numElementos`: retorna el valor del atributo que contiene el número actual de imágenes (`num`).
- `proximaImagen`: si no hay ninguna imagen lanza `NoExiste`. En caso contrario, incrementa `proximo` y lo pone a cero si ha alcanzado el valor de `num`. Después, retorna la imagen que está en la casilla del array indicada por el valor que tenía `proximo` antes de ser modificado.
- `inserta`: si el número actual de imágenes (`num`) es igual al tamaño del array esto indica que no hay hueco para la nueva imagen, y se lanza `NoCabe`. En caso contrario, se mete la nueva imagen en la primera casilla libre del array y se incrementa el número actual de imágenes.
- `filtraPorFechas`: Crea un nuevo catálogo vacío, con un número máximo de imágenes igual al del objeto actual. Luego recorre en un lazo todas las imágenes almacenadas en el array, y si la fecha de la imagen es mayor o igual que la fecha `desde` y menor o igual que la fecha `hasta`, añade (con `inserta`) la imagen al nuevo catálogo. Si se lanza `NoCabe`, indica en pantalla el mensaje `"Error inesperado"`. Al finalizar, retorna el nuevo catálogo que contendrá las imágenes cuyas fechas estén en el rango solicitado.

Se pide también escribir un pequeño programa (una clase con `main`) para probar la clase `Catalogo`. El programa debe hacer lo siguiente:

- Crea un catálogo usando el segundo constructor de la clase (el que no hemos hecho), indicándole como nombre del fichero `"cat.txt"`
- Obtiene otro catálogo filtrando el primero para las fechas entre el 1 de enero de 2002 y el 31 de diciembre de ese mismo año.
- Muestra en pantalla (con `pinta`) todas las imágenes del nuevo catálogo. Para ello, las obtiene sucesivamente desde un lazo con el método `proximaImagen`.
- Si en algún momento se lanzan las excepciones `NoValida` o `NoExiste`, se pone un mensaje indicativo del error en pantalla y se finaliza el programa.

La clase `Catalogo` se valora con 4 puntos, y el programa de prueba con 1 punto.

```
public class Catalogo
{

    private Imagen[] imag;
    private int num;
    private int proximo=0;

    /**
     * Constructor que crea el catalogo vacio, con un maximo de fotos
      * igual a numMax
      */
    public Catalogo(int numMax) {
        imag=new Imagen[numMax];
        num=0;
    }
```

```java
    /**
     * Numero actual de elementos
     */
    public int numElementos() {
        return num;
    }

    /**
     * Retorna la imagen siguiente
     */
    public Imagen proximaImagen() throws NoExiste {
        int actual=proximo;
        if (num==0) {
            throw new NoExiste();
        } else {
            proximo++;
            if (proximo==num) {
               proximo=0;
            }
            return imag[actual];
        }
    }

    /**
     * Inserta una imagen al final del catalogo
     */
    public void inserta(Imagen foto) throws NoCabe{
        if (num==imag.length) {
            throw new NoCabe();
        } else {
            imag[num]=foto;
            num++;
        }
    }

    /**
     * Filtra las imagenes para fechas en el rango [desde,hasta]
     */
    public Catalogo filtraPorFechas(Fecha desde, Fecha hasta) {
        Catalogo nuevo=new Catalogo(imag.length);
        try {
           for (Imagen foto:imag) {
               if (Fecha.mayorOIgualQue(foto.dia(),desde) &&
                   Fecha.menorOIgualQue(foto.dia(),hasta))
               {
                   nuevo.inserta(foto);
               }
           }
        } catch (NoCabe e) {
            System.out.println("Error inesperado "+e);
        }
        return nuevo;
    }
}
```

```
public class PruebaCatalogo
{
   public static void main (String args[]) {
      try {
         Catalogo cat=new Catalogo("cat.txt");
         Fecha inicial = new Fecha(1,1,2002);
         Fecha fin   = new Fecha(31,12,2002);
         Catalogo resultado=cat.filtraPorFechas(inicial,fin);
         for (int i=0; i<resultado.numElementos(); i++) {
             resultado.proximaImagen().pinta();
         }
     } catch (NoExiste e) {
         System.out.println("Fichero no existe");
     } catch (NoValida e) {
         System.out.println("Fecha no valida");
     }
   }
}
```