Soluciones del Examen de Fundamentos de Computadores y Lenguajes

Examen Parcial. Junio 2003

Cuestiones (5 cuestiones, 5 puntos en total)

1) Se dispone de una clase para guardar pares de datos reales, con dos campos públicos:

```
public class Par {
   public double x,y;
}
```

Crear otra clase llamada `ListaPares` que permita guardar varios objetos de la clase `Par`, en un array. El array será un campo privado de la clase. La clase tendrá tres métodos con la siguiente interfaz:

```
public ListaPares (int num) {...}
public Par dato(int i) {...}
public void inserta (Par p, int i) {...}
```

`ListaPares` es el constructor: debe crear el array del tamaño igual a `num`. El método `dato` retorna el objeto de la clase `Par` que ocupa la posición `i` en el array. El método `inserta` pone el campo `i` del array igual al objeto `p`.

```
public class ListaPares {

  private Par [] lista;

  public ListaPares (int num) {
    lista=new Par[num];
  }

  public Par dato(int i) {
    return lista[i];
  }

  public void inserta (Par p, int i) {
    lista[i]=p;
  }

}
```

2) En Java los ficheros se manejan mediante *streams*. Describir lo que es uno de estos *streams*. Indicar los nombres de dos de ellos indicando brevemente para qué sirven y cómo se crean. No usar más de 8 líneas en total para la respuesta a este ejercicio.

Un stream es un objeto que se utiliza en la entrada/salida con ficheros en Java y que opera como un canal de comunicación, con un origen y un destino determinados, y en el que la información se transforma o se cambia de formato según la necesidad. Dos ejemplos:

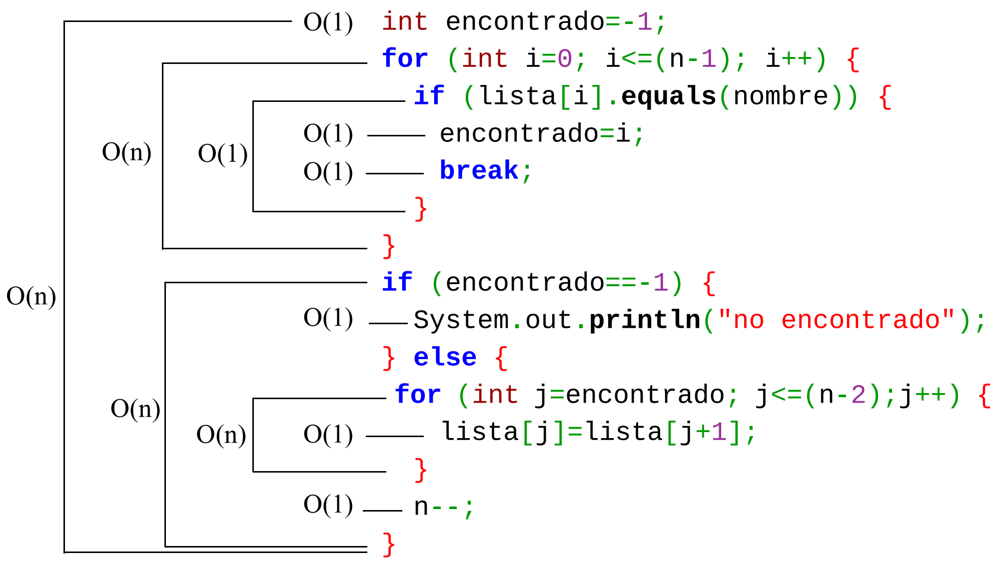
- FileOutputStream: Escribe bytes en un fichero. Ejemplo de creación:
  `FileOutputStream fos=new FileOutputStream("nombre-fichero");`
- ObjectOutputStream: Escribe objetos en un OutputStream. Ejemplo de creación:
  `ObjectOutputStream oos=new ObjectOutputStream(fos);`

3) Escribir en java un fragmento de programa que corresponda al siguiente pseudocódigo que busca un nombre (llamado `nombre`) en un array de strings (llamado `lista`) del que se usan `n` casillas, siendo `n` un entero; si encuentra el nombre en la lista lo borra, moviendo las casillas posteriores una posición hacia arriba:

```
encontrado es un entero: inicialmente vale -1
for i desde 0 hasta n-1 lazo
   if lista[i] igual a nombre then
      encontrado=i
      salir del lazo
   fin if
fin lazo
if encontrado igual a -1 then
   poner mensaje "no encontrado"
else
   for j desde encontrado hasta n-2 lazo
       lista[j]=lista[j+1]
   fin de lazo
   decrementa n
fin if
```

```java
int encontrado=-1;
for (int i=0; i<=(n-1); i++) {
  if (lista[i].equals(nombre)) {
    encontrado=i;
    break;
  }
}
if (encontrado==-1) {
  System.out.println("no encontrado");
} else {
  for (int j=encontrado; j<=(n-2);j++) {
    lista[j]=lista[j+1];
  }
  n--;
}
```

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



La primera instrucción es O(1)

El primer lazo es n veces lo que tiene dentro, que es O(1): por tanto, O(n)

El segundo *if* tiene dos ramas la primera es O(1) la segunda veremos que es O(n). Por tanto, en el peor caso: O(n)

El segundo lazo se hace *n*-1-*encontrado* veces: en el peor caso, *n*-1 veces. Lo de dentro es O(1). Por tanto, en total es O(n)

Por la regla de las sumas, el algoritmo completo es O(n)

5) Se dispone de un método estático de la clase `Radio` con la siguiente interfaz:

```
public static void transmite (String s) throws SinSenal {...}
```

donde `SinSenal` es una excepción declarada como una clase independiente. Escribir un método estático denominado `reintentaTransmision` con un parámetro de tipo `String`, y que retorne un booleano. El nuevo método debe llamar a `transmite` con su parámetro y reintentar la llamada hasta dos veces más si se lanza `SinSenal`. Si la llamada tiene éxito en alguna de las ocasiones, retorna `true` inmediatamente; en caso contrario, es decir si a la tercera llamada también falla, retornará `false`.

```
public static boolean reintentaTransmision (String s) {
   for (int i=0; i<3; i++) {
     try {
       Radio.transmite(s);
       return true; // ha transmitido bien
     } catch (SinSenal e) {
       // no hace falta hacer nada; seguimos en el lazo
     }
   }
   return false; // si llega aquí ha fallado 3 veces
}
```

Examen de Fundamentos de Computadores y Lenguajes

Examen Parcial. Junio 2003

Problema (5 puntos)

Se desea escribir parte del software de un sistema para la monitorización de datos relativos a un accidente marítimo con vertido de hidrocarburos en la costa. Se pide escribir la clase `DatosPlaya`, que sirve para guardar los datos asociados a una playa o zona de la costa concreta. Debe presentar la siguiente interfaz:

```
public class DatosPlaya {

  // terrenos
  public static final int ARENA=1, ROCA=2, RIA=3, MARISMA=4;

  // estados del terreno
  public static final int LIMPIO=1, LEVE=2, MODERADO=3, GRAVE=4;

  // estado de las aves encontradas
  public static final int LIMPIA=1, CONMANCHAS=2;

  public DatosPlaya (String nombre) {...}

  public void cambiaEstado (int terreno, int estado) {...}

  public int leeEstado (int terreno) {...}

  public void aveEncontrada (String especie, boolean muerta,
     int estado) throws NoCabe {...}

  public int avesLimpias (String especie) {...}
  public int avesConManchas (String especie) {...}
  public int avesMuertas (String especie) {...}
  public int avesVivas   (String especie) {...}

}
```

La clase debe definir los siguientes campos, todos privados, y con los nombres indicados:

- `nombrePlaya`: Un string para el nombre de la playa.
- `estadoTerreno`: un array de cuatro enteros para almacenar el estado del terreno (ver constantes "estado del terreno") para cada tipo de terreno (ver constantes "terrenos"); el índice del array representa el tipo de terreno, y el contenido es el estado.
- `nombreAve`: un array de 100 strings para los nombres de las especies de ave encontradas en la playa.
- `numAvesVivas`: un array de 100 números enteros para almacenar el número de aves de cada especie encontradas vivas.
- `numAvesMuertas`: un array de 100 números enteros para almacenar el número de aves de cada especie encontradas muertas.

- `numLimpias`: un array de 100 números enteros para almacenar el número de aves de cada especie encontradas limpias.
- `numConManchas`: un array de 100 números enteros para almacenar el número de aves de cada especie encontradas con manchas de hidrocarburo.
- `numAves`: un número entero que indica cuántas casillas de los arrays de tamaño 100 son válidas.

Los cinco arrays de 100 casillas están relacionados entre sí: la casilla i-ésima de cada uno de ellos se refiere a la misma especie de ave. Por ello sólo es necesario un único número entero que indique el número de casillas útiles del conjunto de los cinco arrays.

Inicializar los campos a los valores que creas adecuados.

Por otro lado, la clase debe desarrollar los métodos de la interfaz. Sus respectivas funciones serán las siguientes:

- `DatosPlaya`: Constructor que pone el campo con el nombre de la playa igual a `nombre`.
- `cambiaEstado`: Cambia el estado del terreno indicado por `terreno` al valor `estado`.
- `leeEstado`: Retorna el estado del terreno indicado.
- `aveEncontrada`: Anota los datos de un ave nueva encontrada, cuya especie se identifica por `especie`, realizando para ello los siguientes pasos:
  - Busca en el array de nombres de aves la especie indicada. Si no la encuentra, es preciso añadir una nueva, si cabe. Si no cabe se lanzará la excepción `NoCabe`. Al añadir la nueva especie se anota su nombre en el array de nombres, y se inicializan los números de aves vivas, muertas, limpias, y con manchas de esa especie a cero.
  - Posteriormente, tanto si la especie es nueva como si es una existente, se actualizan sus datos añadiendo el nuevo avistamiento. Para ello si `muerta` es `true` se incrementa en uno el número de aves muertas de esa especie, y si no el de aves vivas. Si `estado` vale `LIMPIA`, se incrementa en uno el número de aves limpias de esa especie, y si vale `CONMANCHAS` el de aves con manchas de hidrocarburo.
- `avesLimpias`: Retorna el número de aves limpias de la especie indicada. Si la especie no se encuentra almacenada, se retorna cero.
- `avesConManchas`: Retorna el número de aves con manchas de hidrocarburo de la especie indicada. Si la especie no se encuentra almacenada, se retorna cero.
- `avesMuertas`: Retorna el número de aves muertas de la especie indicada. Si la especie no se encuentra almacenada, se retorna cero.
- `avesVivas`: Retorna el número de aves vivas de la especie indicada. Si la especie no se encuentra almacenada, se retorna cero.

Puesto que los últimos cuatro métodos son casi iguales, desarrollar solamente uno de ellos.

Solución

```
public class DatosPlaya {

  // terrenos
  public static final int ARENA=1, ROCA=2, RIA=3, MARISMA=4;

  // estados del terreno
  public static final int LIMPIO=1, LEVE=2, MODERADO=3, GRAVE=4;

  // estado de las aves encontradas
  public static final int LIMPIA=1, CONMANCHAS=2;

  private String nombrePlaya;

  private int estadoTerreno[] = new int[5];
  // cinco elementos, ya que la casilla 0 no se usa

  private static final int maxEspecies=100;

  private String  nombreAve[] =new String [maxEspecies];
  private int numAvesVivas[]  =new int [maxEspecies];
  private int numAvesMuertas[]=new int [maxEspecies];
  private int numLimpias []   =new int [maxEspecies];
  private int numConManchas[] =new int [maxEspecies];

  private int numAves=0;

  public DatosPlaya (String nombre) {
    nombrePlaya=nombre;
  }

  public void cambiaEstado (int terreno, int estado) {
    estadoTerreno[terreno]=estado;
  }

  public int leeEstado (int terreno) {
    return estadoTerreno[terreno];
  }

  // definimos el mÈtodo busca ya que se usa tanto desde aveEncontrada
  // como desde avesLimpias y el resto de mÈtodos similares
  // su misiÚn es buscar una especie en la lista de nombres

  private int busca (String especie) {
    for (int i=0; i<numAves; i++) {
      if (especie.equals(nombreAve[i])) {
        return i; // encontrada
      }
    }
    return -1; // no encontrada
  }

  public void aveEncontrada (String especie, boolean muerta, int estado)
    throws NoCabe
  {
    int i=busca(especie);
    if (i==-1) {
      // No encontrada
      if (numAves==maxEspecies) {
        // No caben mas
        throw new NoCabe();
      } else {
        // nueva especie
         i=numAves;
        numAves++;
        nombreAve[i]=especie;
        numAvesMuertas[i]=0;
        numAvesVivas[i]=0;
```

```
          numLimpias[i]=0;
          numConManchas[i]=0;
        }
      }
      // anadir datos de la nueva ave
      if (muerta) {
        numAvesMuertas[i]++;
      } else {
        numAvesVivas[i]++;
      }
      switch (estado) {
      case LIMPIA:
        numLimpias[i]++;
        break;
      case CONMANCHAS:
        numConManchas[i]++;
      }
    }

    public int avesLimpias (String especie) {
      int i=busca(especie);
      if (i==-1) {
        // no encontrada
        return 0;
      }
      return numLimpias[i];
    }

    public int avesConManchas (String especie) {...}

    public int avesMuertas (String especie) {...}

    public int avesVivas   (String especie) {...}

}
```