Grado en Ingeniería Informática y Grado en Estadística
Fundamentos de Programación

Examen convocatoria ordinaria. 14 de enero de 2020.

Apellidos _____

Nombre _____ **Grupo** _____

DNI y Firma

Duración del examen: 3,5 horas.	Se valorará la presentación y la claridad en la exposición.
Empezar cada problema en una cara distinta.	Se valorará la adecuación de las estructuras utilizadas al problema a resolver.
Poner nombre y apellidos en todas las páginas.	No se calificarán las respuestas escritas a lápiz.
Las hojas del enunciado también deben entregarse.	

1. [2 ptos] Para almacenar los NIF de los usuarios de un determinado sistema se utilizan registros de tipo **`regNIF`**, que contienen tres campos: uno de tipo **`long`** para almacenar el número del DNI, otro de tipo **`int`** que guarda el número de dígitos significativos de ese DNI y otro de tipo **`char`** que almacena la letra de ese DNI. Por ejemplo, la cadena "**`07123456L`**", que representa un NIF correcto, se almacenaría en un registro de tipo **`regNIF`** guardando el entero **`7123456`** en el campo de tipo **`long`**, el valor **`7`** en el campo de tipo **`int`** y el carácter '**`L`**' en el campo de tipo **`char`**. Dadas estas condiciones, se pide:

a) Definir el tipo de datos **`regNIF`**

```
public class RegNIF {
      long num;
      int dig;
      char letra;

      public RegNIF (long x, int n, char c){
            this.num = x;
            this.dig = n;
            this.letra = c;
      }
      public RegNIF (){
            this.num = 0;
            this.dig = 0;
            this.letra = ' ';
      }
}
```

b) Elaborar un método Java con las siguientes características:

Parámetros de entrada: una cadena de caracteres.

Valor devuelto: el registro **`regNIF`** correspondiente.

Precondición: la cadena de entrada se corresponde con un NIF correcto en el sentido de que contiene 9 caracteres, de los cuales los 8 primeros son dígitos, y el último una letra mayúscula del alfabeto inglés.

```
public static RegNIF conversor (String cad){
      int dig, ind = 0;
      long num=0, pot=1;

      while (cad.CharAt(ind) == 0)
            ind++;
      dig = 8-ind;

      for (int i=7; i>=ind; i--){
            num = num + (cad.CharAt(i)-'0')*pot;
            pot = pot * 10;
      }

      RegNIF reg = new RegNIF (num, dig, cad.CharAt(8));
      return reg;
}
```

```
//Otra forma de hacerlo
public static RegNIF conversor (String cad){
      int dig, ind = 0;
      long num=0;

      while (cad.CharAt(ind) == 0)
            ind++;
      dig = cad.length-1-ind;

      for (int i=ind; i>cad.length-1; i++)
            num = num*10 + (cad.CharAt(i)-'0');

      RegNIF reg = new RegNIF (num, dig, cad.CharAt(cad.length-1));
      return reg;
}
```

2. [2 ptos] Crear un programa Java que escriba en el fichero de texto **`salida.txt`** el resultado de concatenar un número indeterminado de ficheros de texto. Los nombres de los ficheros de texto de entrada se le irán pidiendo al usuario. Para terminar el usuario debe teclear la cadena **`FIN`**. En el fichero de salida debe copiarse la información contenida en todos los ficheros de entrada, cuyo nombre haya indicado el usuario, en el mismo orden en el que se hayan ido introduciendo los nombres. Supóngase que están importados los paquetes necesarios.

RECORDATORIO: Para declarar y abrir en modo lectura un fichero de texto:

`Scanner <`*`id_fich`*`> = new Scanner (new File(<`*`nombre_fich`*`>));`

Al abrir el fichero se puede producir la excepción **`FileNotFoundException`**

Para declarar y abrir en modo escritura un fichero de texto:

`PrintWriter <`*`id_fich`*`> = new PrintWriter (new FileWriter (<`*`nombre_fich`*`>));`

Al abrir el fichero se puede producir la excepción **`IOException.`**

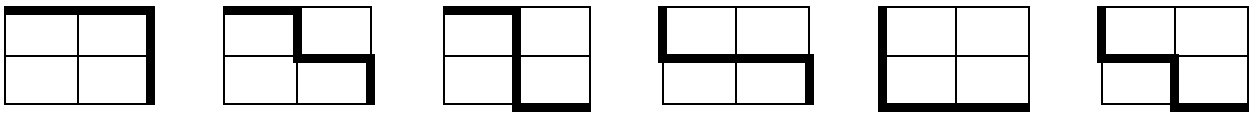
```
public class Concatena{
   public static void main (String[] args){
         Scanner in = new Scanner (System.in);
         String linea, nombre;

         Scanner entrada;
         PrintWriter salida;

         try {
               resultado = new PrintWriter ("salida.txt");
         }
         catch (IOException e) {
               System.out.println ("No puede abrirse el fichero");
               return;
         }

         System.out.println ("Teclee el nombre del siguiente fichero");
         nombre = in.next();



         while (!nombre.equals("FIN")){
               try {
                     entrada = new Scanner (new File (nombre));
               }
               catch (FileNotFoundException e) {
                     System.out.println ("No se encontró el fichero");
                     return;
               }
               while (entrada.hasNextLine()){
                     linea = entrada.nextLine();
                     resultado.println (linea);
               }
               entrada.close();
               System.out.println ("Teclee el nombre del siguiente fichero");
               nombre = in.next();
         }
         salida.close();
   }
}
```

3. **[1 pto]** Dada **`list`** (de tipo **`Nodo`**), una referencia a una **lista dinámica** que contiene números enteros en su campo **`dato`**, crear un método Java que devuelva si aparece o no en esa lista un número dado (de tipo **`int`**). Supóngase definida la clase Nodo como:

```
public class Nodo {
      int dato;
      Nodo sgte;
      // constructores, etc.
}
```

```
public static boolean buscar (Nodo list, int num){
   lista q = list;
   boolean esta = False;
   while ((q != null) && (!esta)){
         if (q.dato == num)
               esta = True;
         q = q.sgte;
   }
   return esta;
}
```

4. **[1 pto]** Comenzando en la esquina superior derecha de una cuadrícula de tamaño 2x2, y realizando exclusivamente movimientos hacia abajo y a la derecha, hay exactamente 6 rutas para llegar a la esquina inferior derecha. Gráficamente, las rutas posibles son:



Escriba una función **recursiva** en Java, de nombre **`numRut`**, que devuelva el número de rutas que existen para llegar desde un punto de la cuadrícula a la esquina inferior derecha, en una cuadrícula de tamaño **`TAMxTAM`**, siendo **`TAM`** una constante predefinida.

Por ejemplo, para la cuadrícula de tamaño **`2x2`**, y siendo **`(0,0)`** la coordenada de la esquina superior izquierda, la función **`numRut`** debería devolver 6. Es decir, **`numRut(0,0)`** vale **`6`**.

PISTA: El número de rutas hasta la esquina inferior derecha desde cualquier punto de la cuadrícula, se puede calcular sumando el número de rutas desde punto inferior a ese punto (si lo hay) y el número de rutas desde el punto a la derecha de ese punto (si es que existe).

```
public static int numRut (int fil, int col){
   if ((fil == TAM) && (col == TAM))
         return 1;
   else
         if (fil == TAM)
               return (numRut(fil, col+1);
         else
               if (col == TAM)
                     return (numRut(fil+1, col);
               else
                     return ((numRut(fil+1, col) + (numRut(fil, col+1);
}
```

5. **[2 ptos]** En un centro educativo se almacenan las **notas** de los alumnos de un determinado curso en una matriz bidimensional de tamaño *<número de alumnos> x <número de asignaturas>* y en cada casilla la nota del alumno en la asignatura correspondiente. Elabore un método Java que construya y devuelva una matriz de **medias** que contenga una columna y una fila más que la matriz de notas y cuyo contenido se calcule a partir de la matriz de **notas** con el siguiente proceso:

- En la columna adicional debe almacenar la media de cada alumno y en la fila adicional la media de cada asignatura.
- En la posición correspondiente a la última fila y última columna se debe almacenar la nota media global de todo el curso.
- El resto de las posiciones de la matriz de medias debe ser el mismo que el de la matriz de notas.

Por ejemplo. Suponiendo que hay 5 alumnos y 3 asignaturas y que la matriz de **notas** es la de la izquierda, el método debería devolver una matriz de **medias** como la de la derecha:

notas

	0	1	2
0	5,20	6,40	4,00
1	7,60	8,90	4,20
2	4,60	5,70	8,00
3	3,00	7,50	6,50
4	6,30	1,00	2,00

medias

	0	1	2	3
0	5,20	6,40	4,00	5,20
1	7,60	8,90	4,20	6,90
2	4,60	5,70	8,00	6,10
3	4,00	7,50	6,50	6,00
4	6,30	1,00	2,00	3,10
5	5,54	5,90	4,94	5,46

```
public static double [][] matrices (double [][] notas){
   int fil = notas.length;
   int col = notas[0].length;
   double suma;
   double [][] medias = new double [fil+1][col+1];

   //copia de los datos originales y cálculo de la media global
   suma = 0;
   for (int i=0; i<fil; i++)
         for (int j=0; j<col; j++){
               medias[i][j] = notas[i][j];
               suma = suma + notas[i][j];
         }
   medias[fil][col] = suma/(fil*col);

   //cálculo de la nueva columna, recorrido por filas de la matriz notas
   for (int i=0; i<fil; i++){
         suma = 0;
         for (int j=0; j<col; j++)
               suma = suma + notas[i][j];
         medias[i][col] = suma/col;
   }

   //cálculo de la nueva fila, recorrido por columnas de la matriz notas
   for (int j=0; j<col; j++){
         suma = 0;
         for (int i=0; i<fil; i++)
               suma = suma + notas[i][j];
         medias[fil][j] = suma/fil;
   }
   return medias;
}
```

6. **[2 pto]** Elaborar un método Java que, tomando como parámetro de entrada una cadena de caracteres, devuelva el número de dígitos **distintos** que hay en esa cadena. Por ejemplo, si la cadena de entrada fuese **"14 de enero de 2020"**, el método debería devolver **4**, ya que contiene cuatro dígitos diferentes (**'1'**, **'4'**, **'0'** y **'2'**).

```
public static int cadenas (String cad){
   int [] cont = new int [10];
   int i = 0;
   int dig = 0;

   for (i=0; i<10; i++)
         cont[i] = 0;

   for (i=0; i<cad.length; i++)
         if ((cad.charAt(i) >= '0') && (cad.charAt(i) <= '9'))
               cont [cad.charAt(i)-'0']++;

   for (i=0; i<10; i++)
         if (cont[i] > 0)
               dig++;
   return dig;
}
```

```
//Otra forma de hacerlo
public static int cadenas (String cad){
   int dig = 0;

   for (int i=0; i<10; i++)
         if (cad.indexOf((char)('0'+i)) >= 0)
               dig++;

   return dig;
}
```