Grado en Ingeniería Informática y Grado en Estadística
Fundamentos de Programación

Examen de convocatoria extraordinaria. 3 de febrero 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: **`num`** de tipo **`long`** para almacenar el número del DNI, **`dig`** de tipo **`int`** que guarda el número de dígitos significativos de ese DNI y **`letra`** de tipo **`char`** que almacena la letra de ese DNI. Por ejemplo, la cadena "**`07123006L`**", que representa un NIF correcto, se almacenaría en un registro de tipo **`regNIF`** guardando el entero **`7123006`** en el campo **`num`**, el valor **`7`** en el campo **`dig`** y el carácter '**`L`**' en el campo **`letra`**. Dadas estas condiciones, se pide:

a) Elaborar un método Java que determine si una determinada cadena de caracteres puede ser 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 boolean validacion (String cad){
   boolean correcto = true;
   int i=0;

   if (cad.length != 9)
         correcto = false;
   else
         if ((cad.CharAt(8) <= 'A') || (cad.CharAt(8) >= 'Z')
               correcto = false;
         else
             while ((i<cad.length-1) && correcto){
               if ((cad.CharAt(i)<='0') || (cad.CharAt(i)>='9')
                     correcto = false;
               i++;
             }
   return correcto;
 }
```

b) Elaborar un método Java con las siguientes características:

Parámetros de entrada: un registro de tipo **`regNIF`**

Valor devuelto: la cadena de caracteres correspondiente.

Precondición: el registro de entrada es correcto, es decir, su campo **`letra`** es un carácter que se corresponde con una letra mayúscula del alfabeto inglés y su campo **`num`** es un número entero positivo que tiene tantos dígitos significativos como el valor almacenado en el campo **`dig`**.

```
public static String conversion (regNIF reg){
      long num = reg.num;
      long pot = 1;
      String cad = "";

      for (int i=1; i<=(8-reg.dig); i++)
            cad = cad + "0";

      for (int i=1; i<=(reg.dig-1); i++){
            pot = pot*10;

      while (num >0){
            cad = cad + (char)((num/pot)+'0');
            num = num%pot;
            pot = pot/10;
      }

      cad = cad + reg.letra;
      return cad;
}
```

2. **[2 ptos]** El fichero de texto **`valores.txt`** contiene números reales a razón de uno por línea. Crear un programa Java que elimine de dicho fichero el número que se obtenga del usuario a través del teclado, todas las veces que aparezca. Suponer 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 Borra{
   public static void main (String[] args){
         Scanner in = new Scanner (System.in);
         double num, dato;

         Scanner valores;
         PrintWriter auxiliar;

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

```
        try {
              valores = new Scanner (new File ("valores.txt"));
        }
        catch (FileNotFoundException e) {
              System.out.println ("No se encontró el fichero");
              return;
        }

        System.out.println ("Teclee el dato a borrar: ");
        num = in.nextDouble();

        while (valores.hasNextDouble()){
              dato = valores.nextDouble();
              if (dato != num)
                    auxiliar.println (dato);
        }
        valores.close();
        auxiliar.close();

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



        try {
              auxiliar = new Scanner (new File ("auxiliar.txt"));
        }
        catch (FileNotFoundException e) {
              System.out.println ("No se encontró el fichero");
              return;
        }

        while (auxiliar.hasNextDouble()){
              dato = auxiliar.nextDouble();
              valores.println (dato);
        }
        valores.close();
        auxiliar.close();
   }
}
```

3. **[1 pto]** Una forma de calcular la potencia b^n, suponiendo que **n** es un entero positivo y que **b** y **n** no son simultáneamente nulos, es utilizar las siguientes propiedades de la potencia:

   a) Si **n=0** , el resultado es **1**

   b) Si **n** es par, $b^n = b^{n/2} * b^{n/2}$

   c) Si **n** es impar, $b^n = b^{n-1} * b$

Escriba una función recursiva en Java que calcule la potencia de esta manera.

```
public static float potencia (float b, int n){

   if (n == 0)
         return 1;
   else
         if (n%2 == 0)
               return (potencia(b, n/2) * potencia(b, n/2));
         else
               return (potencia(b, n-1) * b);
}
```

4. **[1,5 ptos]** Dada **list** (de tipo **Nodo**), una referencia a una **lista dinámica** que contiene números enteros en el campo **dato**, crear un método Java que devuelva cuántos números pares hay en posiciones pares de la lista suponiendo que se empieza a contar desde 1. Considere definida la clase **Nodo** como:

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

Por ejemplo, si la lista contuviera los valores **4 12 15 17 45 46**, el método debería devolver **2**, ya que hay 2 números pares que ocupan posición par (el 12 en la posición 2 y el 46 en la posición 6).

```
public static int contar (Nodo list, int num){
   lista q = list;
   boolean esta = False;
   int cont = 0;
   int pos = 0;

   while (q != null){
         pos++;
         if ((q.dato%2 == 0) && (pos%2 == 0))
               cont++;
         q = q.sgte;
   }
   return cont;
}
```

[2 ptos] Escribir un método Java que devuelva cuántas veces aparece en una matriz de números reales el valor máximo de la suma de sus diagonales.

Por ejemplo. Si la matriz de entrada fuese la siguiente:

	0	1	2
0	5,20	6,10	-4,00
1	6,10	-8,90	4,20
2	4,60	5,70	8,00

La suma de la diagonal principal es 6,1 (=5,2-8,9+8) y la de la diagonal secundaria -8,3 (=-4-8,9+4,6), por lo que el método debería devolver **2**, ya que el valor 6,1 (el máximo de 6,1 y -8,3) aparece dos veces en la matriz.

```
//Posible solución: Sin precondiciones sobre la matriz
public static double diagonales (double [][] m){
   int fil = notas.length;
   int col = notas[0].length;
   int j;
   double suma, tamMax, i, j, diag1=0, diag2=0;

   if (fil<col)
         tamMax = fil;
   else
         tamMax = col;

   for (i=0; i<tamMax; i++)
         diag1 = diag1 + m[i][i];
   j = col;
   for (i=0; i<tamMax; i++){
         diag2 = diag2 + m[i][j];
         j--;
   }

   if (diag1>diag2)
         return diag1;
   else
         return diag2;
}
```

```
public static int numDiagonales (double [][] m){
   int cont = 0;
   for (int i=0; i<m.length; i++)
         for (int j=0; j<m[0].length; j++)
               int (m[i][j] == diagonales(m))
                     cont++;
   return cont;
}
```

```java
//Otra solución: Suponiendo que la matriz es cuadrada
public static double diagonales (double [][] m){
// PRECONDCIÓN: La matriz es cuadrada, Número de filas = Número de columnas

   int fil = notas.length;
   int col, i, j;
   double suma, diag1=0.0, diag2=0.0;

   for (i=0; i<fil; i++)
         diag1 = diag1 + m[i][i];
   j = fil;
   for (i=0; i<fil; i++){
         diag2 = diag2 + m[i][j];
         j--;
   }

   if (diag1>diag2)
         return diag1;
   else
         return diag2;
}


public static int numDiagonales (double [][] m){
// PRECONDCIÓN: La matriz es cuadrada, Número de filas = Número de columnas

   int cont = 0;
   for (int i=0; i<m.length; i++)
         for (int j=0; j<m[0].length; j++)
               int (m[i][j] == diagonales(m))
                     cont++;
   return cont;
}
```

6. **[1,5 ptos]** Elaborar un método Java que, tomando como parámetro de entrada una cadena de caracteres, devuelva otra cadena cuyos caracteres sean los de la cadena de entrada pero en el siguiente orden: el primero, el último, el segundo, el penúltimo, el tercero, el antepenúltimo, etc.

Por ejemplo. Si la cadena de entrada fuese "**hola mundo!**", la cadena de salida debería ser "**h!ooldan um**"

```java
public static String combinada (String cad){
   String nueva = "";
   int tam = cad.length;

   for (int i=0; i < tam/2; i++)
         nueva = nueva + cad.charAt(i) + cad.charAt(tam-1+i);

   if ((tam/2)%2 == 1)
         nueva = nueva + cad.charAt(tam/2);

   return nueva;
}
```