Escuela de Ingeniería y Arquitectura - Depto. de Informática e Ingeniería de Sistemas
Examen práctico de Programación 2 - 11 de septiembre de 2014

Este examen práctico individual en laboratorio forma parte de la evaluación de la asignatura. Su calificación tiene un peso del 30 % en la calificación final de la asignatura en la convocatoria de septiembre.

En esta prueba se debe diseñar la clase ***examenSeptiembre.Apellidos*** en la que debe constar el código de los métodos que se piden a continuación, junto con el de sus métodos auxiliares. El nombre de la clase es el resultado de concatenar los apellidos propios de cada alumno. Así, por ejemplo, el nombre de la clase que debiera diseñar un alumno que se apellidara ***Ortiz Cuenca*** sería ***examenSeptiembre.OrtizCuenca***.

```
package examenSeptiembre;

/**
 * Esta clase almacena los método públicos pedidos en este examen así como los métodos
 * privados auxiliares resultantes del diseño de los anteriores
 */
public class ApellidosPropiosDeCadaAlumno {

    /**
     * Pre: escribir aquí su precondición
     * Post: escribir aquí su postcondición
     */
    public static String letrasOrdenadas (String s)

    /**
     * Pre: n>=0
     * Post: Devuelve un entero que, escrito en base 10, consta del mismo conjunto
     *       de cifras significactivas que n, tiene todas sus cifras ordenadas
     *       de mayor a menor valor, y, cada una de ellas, está en él repetida
     *       el mismo número de veces que en n
     */
    public static int ordenarSusCifras (int n)

}
```

1. Se pide un diseño sin bucles del método **letrasOrdenadas(s)**. **[5 puntos]**

   Se debe hacer un diseño sin bucles, desarrollando cuantos métodos auxiliares privados sean necesarios. Estos métodos auxiliares tampoco podrán presentar bucles. Su comportamiento se describe e ilustra a continuación.

   Devuelve la secuencia de letras del alfabeto inglés [A,B,C,...,W,Y,Z], ordenadas alfabéticamente, que forman parte del string referenciado por **s**. El comportamiento del método es el mismo tanto si las letras presentes en el string son mayúsculas o minúsculas.

   Ejemplos:

   - letrasOrdenadas("`ABECEDARIO`") = "`ABCDEIOR`"
   - letrasOrdenadas("`abecedario`") = "`ABCDEIOR`"
   - letrasOrdenadas("`Aragon`") = "`AGNOR`"
   - letrasOrdenadas("`Zaragoza`") = "`AGORZ`"
   - letrasOrdenadas("`Hoy es 11 de septiembre`") = "`BDEHIMOPRSTY`"

- letrasOrdenadas("11-09-2014") = ""

Es obligatorio que todos los métodos desarrollados estén adecuadamente especificados, aunque no es necesario utilizar especificaciones formales (predicados matemáticos).

2. Se pide un diseño sin bucles del método **ordenarSusCifras (n)** de acuerdo con la especificación antes mostrada. **[5 puntos]**

   Se debe hacer un diseño sin bucles, desarrollando cuantos métodos auxiliares privados sean necesarios. Estos métodos auxiliares tampoco podrán presentar bucles.

   Algunos ejemplos que ilustran el comportamiento del método **ordenarSusCifras (n)** se muestran a continuación.

   - ordenarSusCifras(0) = 0
   - ordenarSusCifras(6) = 6
   - ordenarSusCifras(161) = 611
   - ordenarSusCifras(60041) = 64100
   - ordenarSusCifras(40776271) = 77764210
   - ordenarSusCifras(229099202) = 999222200

   Es obligatorio que todos los métodos desarrollados estén adecuadamente especificados, aunque no es necesario utilizar especificaciones formales (predicados matemáticos).

   No se permite definir ningún dato fuera del ámbito local de cada uno de los métodos diseñados.

Entrega del trabajo

Como resultado del trabajo se entregará el fichero ***Apellidos*.java**, por ejemplo, ***OrtizCuenca.java***, que almacena el código de la clase Java desarrollada, a través de la plataforma ***Moodle2*** (moodle2.unizar.es). Se recomienda no hacerlo hasta que haya sido verificado de forma exhaustiva que el comportamiento del código dieñado es el adecuado.

Se recuerda que, en caso de detectarse coincidencias significativas en una parte del trabajo de dos o más alumnos, todos ellos serán calificados con un cero en esta prueba.

Una solución

```
package examenSeptiembre;

/**
 * Esta clase almacena los método públicos pedidos en este examen así como los métodos
 * privados auxiliares resultantes del diseño de los anteriores
 */
public class ApellidosPropiosDeCadaAlumno {

    /**
     * Pre: cierto
     * Post: Devuelve la secuencia de letras, ordenadas alfabéticamente, que forman parte del
     *       string referenciado por s
     *       Ejemplos:
     *         letrasOrdenadas("ABECEDARIO") = "ABCDEIOR"
     *         letrasOrdenadas("abecedario") = "ABCDEIOR"
     *         letrasOrdenadas("Aragon") = "AGNOR"
     *         letrasOrdenadas("Zaragoza") = "AGORZ"
     *         letrasOrdenadas("Hoy es 11 de septiembre") = "BDEHIJMOPRSTUVY"
     *         letrasOrdenadas("11-09-2014") = ""
     */
    public static String letrasOrdenadas (String s) {
        boolean[] letras =
            { false, false, false, false, false, false, false, false, false, false,
              false, false, false, false, false, false, false, false, false, false,
              false, false, false, false, false, false };
        s = s.toUpperCase();
        anotarLetras (s, letras );
        return presentarLetras ( letras , 'A', "");
    }

    /**
     * Pre: letras.length=26 y el elemento de índice I de letras hace referencia al carácter
     *      cuya posición en el alfabeto anglosajón (A, B, ..., Y, Z) es la I+1
     * Post: Asigna el valor true a los datos de la tabla referenciada por letras cuyos índices
     *       hacen referencia a las letras del alfabeto anglosajón (A, B, ..., Y, Z) contenidas
     *       en el string referenciado por s
     */
    private static void anotarLetras (String s, boolean[] letras ) {
        if (s.length()>0) {
            if (s.charAt(0)>='A' && s.charAt(0)<='Z') {
                letras [s.charAt(0)-'A'] = true;
            }
            anotarLetras (s. substring (1), letras );
        }
    }
```

```
/**
 * Pre: parcial es una referencia a un string con las letras, en orden alfabético, del alfabeto
 *      tales que los valores que les corresponden en letras [0..desde-'A'-1] son true.
 * Post: Devuelve una referencia a un string con las letras, en orden alfabético, del alfabeto
 *       tales que los valores que les corresponden en letras [0..'Z'-'A'-1] son true.
 */
private static String presentarLetras (boolean[] letras, char desde, String parcial) {
    if (desde<='Z') {
        if ( letras [desde-'A']) {
            String letra = "" + desde;
            return presentarLetras ( letras , (char) (desde+1), parcial .concat( letra ));
        }
        else {
            return presentarLetras ( letras , (char) (desde+1), parcial );
        }
    }
    else {
        return parcial ;
    }
}


/**
 * Pre: n>=0
 * Post: Devuelve un entero que escrito en base 10 consta del mismo conjunto
 *       de cifras significactivas que n, tiene todas sus cifras ordenadas
 *       de mayor a menor valor, y, cada una de ellas, está en él repetida
 *       el mismo número de veces que en n
 */
public static int ordenarSusCifras (int n) {
    return considerarCifras (n ,9,0);
}

/**
 * Pre: n>=0 AND cifra>=0 AND cifra<=9 AND parcial es un entero que, escrito en base 10,
 *      solo contiene las cifras del intervalo [cifra+1,9] y, cada una de ellas, está en
 *      él repetida el mismo número de veces que en n
 * Post: Devuelve un entero que escrito en base 10 tiene todas sus cifras ordenadas
 *       de mayor a menor valor y, cada una de ellas, está en él repetida
 *       el mismo número de veces que en n
 */
private static int considerarCifras (int n, int cifra, int parcial) {
    if ( cifra >=0) {
        parcial = acumularCifra( cifra , contarVeces(n, cifra ), parcial );
        return considerarCifras (n, cifra -1, parcial );
    }
    else {
        return parcial ;
    }
}
```

```
    /**
     * Pre:  cifra >=0 AND cifra<=9 AND veces>=0 AND parcial>=0
     * Post: acumular(cifra, veces, parcial) = 10^veces*parcial
     *                                        + (SIGMA alfa EN [0,veces-1].10^alfa)* cifra
     */
    private static int acumularCifra (int cifra, int veces, int parcial) {
        if (veces==0) {
            return parcial;
        }
        else {
            return acumularCifra (cifra, veces-1, 10*parcial+cifra);
        }
    }

    /**
     * Pre: n>=0 AND cifra>=0 AND cifra<=9
     * Post: Devuelve el número de veces que está repetida cifra como cifra significativa
     *       de n cuando este valor es expresado en base 10
     */
    private static int contarVeces (int n, int cifra) {
        if (n>0) {
            if (n%10==cifra) {
                return 1 + contarVeces(n/10, cifra);
            }
            else {
                return contarVeces(n/10, cifra);
            }
        }
        else {
            return 0;
        }
    }

}
```