Escuela de Ingeniería y Arquitectura - Depto. de Informática e Ingeniería de Sistemas

Examen práctico de Programación 2 - 27 de junio de 2014 - Turno 1º

Este examen práctico individual en laboratorio forma parte de la evaluación de la asignatura. Su calificación tiene un peso del 15 % en la calificación final de la asignatura en la convocatoria de junio.

En esta prueba se debe diseñar la clase ***examenJunio.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 ***Bueno Sevilla*** sería ***examenJunio.BuenoSevilla***.

```
package examenJunio;

/**
 * Esta clase almacena los métodos 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 int quitarRepesConsecutivos (int n)

    /**
     * Pre: T.length>0 AND (PT alfa En [0,T.length-2].T[alfa]<=T[alfa+1])
     * Post: elMasRepetido(T) = R AND esElMasRepetido(T, 0, T.length-1, R)
     */
    public static int elMasRepetido (int [] T)

}
```

1. Se pide un diseño sin bucles del método **quitarRepesConsecutivos(n)**. **[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.

   Si n es 0 entonces devuelve 0. En otro caso, devuelve el entero que, escrito en base 10, es igual al entero resultante de eliminar del valor **n** de su parámetro todas sus cifras que sean iguales a la cifra que le sigue, cuando se escribe también en base 10.

   Ejemplos:

   - quitarRepesConsecutivos(1033720001) = 1037201
   - quitarRepesConsecutivos(-1033720001) = -1037201
   - quitarRepesConsecutivos(30007000) = 3070
   - quitarRepesConsecutivos(-30007000) = -3070
   - quitarRepesConsecutivos(0) = 0
   - quitarRepesConsecutivos(999999111) = 91
   - quitarRepesConsecutivos(-999999111) = -91

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.

2. Se pide un diseño sin bucles del método **elMasRepetido(T)** cuyo coste sea lineal en el número de datos de la tabla referenciada por **T**. **[5 puntos]**

   En la especificación del método se ha hecho uso del siguiente predicado que define las relaciones que satisface el dato más repetido **R** de los datos almacenados en la subtabla ***T[desde..hasta]***:

   $$esElMasRepetido(T, desde, hasta, R) \equiv (\forall \alpha \in [desde, hasta].$$
   $$(Núm\ \beta \in [desde, hasta].T[\beta] = T[\alpha]) \leq (Núm\ \beta \in [desde, hasta].T[\beta] = R))$$

   Es obligatorio que todos los métodos desarrollados estén adecuadamente especificados, aunque no es necesario utilizar especificaciones formales (predicados matemáticos). En las especificaciones de los métodos auxiliares puede hacerse uso del predicado $esElMasRepetido(T, desde, hasta, R)$ definido anteriormente.

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

   Al calificar este problema se tendrá muy en cuenta la eficiencia en tiempo y en memoria del diseño recursivo realizado.

Entrega del trabajo

Como resultado del trabajo se entregará el fichero ***ApellidosPropios*.java**, por ejemplo, ***BuenoSevilla.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 examenJunio;

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

    /**
     * Pre: cierto
     * Post: Si n es 0 entonces devuelve 0. En otro caso, devuelve el entero que, escrito en
     *       base 10, es igual al resultante de eliminar de "n" todas sus cifras cuyo valor
     *       sea igual a la cifra que le sigue, cuando se escribe también en base 10.
     */
    public static int quitarRepesConsecutivos (int n) {
        return quitarRepesConsecutivos(n/10, n%10);
    }

    /**
     * Pre: ultima>=0 AND ultima<=9
     * Post: Devuelve el entero que, escrito en base 10, es igual al resultante de eliminar del
     *       número 10*n+ultima todas sus cifras cuyo valor sea igual a la cifra que le sigue,
     *       cuando se escribe también en base 10, salvo que el valor de número 10*n+ultima
     *       sea 0 en cuyo caso devuelve 0.
     */
    private static int quitarRepesConsecutivos (int n, int ultima) {
        if (n==0) {
            return ultima;
        }
        else {
            if (n%10==ultima) {
                return quitarRepesConsecutivos(n/10, ultima);
            }
            else {
                return 10*quitarRepesConsecutivos(n/10, n%10) + ultima;
            }
        }
    }
```

```
    /**
     * Pre: T.length>0 AND (PT alfa En [0,T.length-2].T[alfa]<=T[alfa+1])
     * Post: elMasRepetido(T) = R AND esElMasRepetido(T, 0, T.length-1, R)
     */
    public static int elMasRepetido (int[] T) {
        return elMasRepetido(T, T[0], 1, T[0], 1, 1);
    }

    /**
     * Pre: i>=0 AND i<=T.length AND (PT alfa En [0,T.length-2].T[alfa]<=T[alfa+1])
     *      AND elMasRepetido(T, 0, i-vecesAspirante, candidato)
     *      AND vecesCandidato = (Núm alfa EN [0,i-1].T[alfa]=candidato]
     *      AND aspirante = T[i-1]
     *      AND vecesAspirante = (Núm alfa EN [0,i-1].T[alfa]=aspirante]
     * Post: elMasRepetido(T,candidato,vecesCandidato, aspirante, vecesAspirante, i) = R
     *       AND esElMasRepetido(T, 0, T.length-1, R)
     */
    private static int elMasRepetido (int[] T, int candidato, int vecesCandidato,
                                      int aspirante, int vecesAspirante, int i) {
        if (i==T.length) {
            if (vecesCandidato>=vecesAspirante) {
                return candidato;
            }
            else {
                return aspirante;
            }
        }
        else if (T[i]==aspirante) {
            return elMasRepetido(T, candidato, vecesCandidato, aspirante, vecesAspirante+1, i+1);
        }
        else {
            if (vecesCandidato<vecesAspirante) {
                return elMasRepetido(T, aspirante, vecesAspirante, T[i], 1, i+1);
            }
            else {
                return elMasRepetido(T, candidato, vecesCandidato, T[i], 1, i+1);
            }
        }
    }

}
```

Escuela de Ingeniería y Arquitectura - Depto. de Informática e Ingeniería de Sistemas
Examen práctico de Programación 2 - 27 de junio de 2014 - Turno 2º

Este examen práctico individual en laboratorio forma parte de la evaluación de la asignatura. Su calificación tiene un peso del 15 % en la calificación final de la asignatura en la convocatoria de junio.

En esta prueba se debe diseñar la clase ***examenJunio.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 ***Bueno Sevilla*** sería ***examenJunio.BuenoSevilla***.

```
package examenJunio;

/**
 * Esta clase almacena los métodos 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 int cifrasOrdenadas (int n)

    /**
     * Pre: T.length>0
     * Post: Devuelve la longitud de la subsecuencia más larga de datos consecutivos de T de valor
     *       no decreciente .
     */
    public static int longSecuenciaMasLarga (int[] T)

}
```

1. Se pide un diseño sin bucles del método **cifrasOrdenadas(n)**. **[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.

   El método ha de devolver un número que, escrito en base 10, presenta ordenadas de mayor a menor valor las cifras significativas de su parámetro **n**, cuando el valor de éste se escribe también en base 10.

   Ejemplos:

   - cifrasOrdenadas(0) = 0
   - cifrasOrdenadas(333) = 3
   - cifrasOrdenadas(-333) = 3
   - cifrasOrdenadas(609364) = 96430
   - cifrasOrdenadas(-609364) = 96430
   - cifrasOrdenadas(1033072001) = 73210
   - cifrasOrdenadas(-1033072001) = 73210

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.

2. Se pide un diseño sin bucles del método **longSecuenciaMasLarga (T)** cuyo coste sea lineal en el número de datos de la tabla referenciada por **T**. **[5 puntos]**

   La tabla referenciada por **T** almacena una secuencia de datos enteros. Por ejemplo:

   `T = < 8, 1, 4, 4, 4, 5, 0, 132, -2, -1, 10, 5, 3, 3, 8, 8, 8, 4 >`

   En esta secuencia podemos distinguir las siguientes subsecuencias de datos consecutivos de valor no decreciente.

   `< 8 >` subsecuencia de datos no decrecientes cuya longitud es de un dato.

   `< 1, 4, 4, 4, 5>` subsecuencia de datos no decrecientes cuya longitud es de 5 datos.

   `< 0, 132 >` subsecuencia de datos no decrecientes cuya longitud es de 2 datos.

   `< -2, -1, 10 >` subsecuencia de datos no decrecientes cuya longitud es de 3 datos.

   `< 5 >` subsecuencia de datos no decrecientes cuya longitud es de un dato.

   `< 3, 3, 8, 8, 8 >` subsecuencia de datos no decrecientes cuya longitud es de 5 datos.

   `< 4 >` subsecuencia de datos no decrecientes cuya longitud es de un dato.

   Una invocación **longSecuenciaMasLarga (T)** devolverá como resultado 5, la longitud de la subsecuencia más larga de datos consecutivos de **T** de valor no decreciente.

   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 ***ApellidosPropios*.java**, por ejemplo, ***BuenoSevilla.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 examenJunio;

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

    /**
     * Pre: cierto
     * Post: Devuelve un número que, escrito en base 10, presenta ordenadas de mayor a menor valor
     *       las cifras significativas de n, cuando éste se escribe también en base 10.
     */
    public static int cifrasOrdenadas (int n) {
        boolean[] cifras =
            { false , false , false , false , false , false , false , false , false , false };
        if (n<0) { n = -n; }
        anotarCifras (n, cifras );
        return presentarCifras ( cifras , 9, 0);
    }

    /**
     * Pre: cifras . length=10 AND n>=0
     * Post: Asigna el valor true a los datos de la tabla referenciada por cifras cuyos índices
     *       coinciden con las cifras significativas de n, cuando su valor de expresa en base 10.
     *       Deja inalterados los restantes datos de la tabla referenciada por cifras
     */
    private static void anotarCifras (int n, boolean[] cifras ) {
        if (n!=0) {
            cifras [n%10] = true;
            anotarCifras (n/10, cifras );
        }
    }

    /**
     * Pre: Sea n el número de elementos de la subtabla cifras [0.. cifra+1, 9] cuyo valor sea true
     *      Sea R el número formado por los índices de la subtabla cifras [0.. cifra ] cuyo valor
     *      sea true escritos en base 10 según valores decrecientes .
     * Post: Devuelve 10^N+R
     */
    private static int presentarCifras (boolean[] cifras , int cifra , int parcial ) {
        if ( cifra >=0) {
            if ( cifras [ cifra ]) {
                return presentarCifras ( cifras , cifra -1, 10* parcial + cifra );
            }
            else {
                return presentarCifras ( cifras , cifra -1, parcial );
            }
        }
        else {
            return parcial ;
        }
    }
```

```
    /**
     * Pre: T.length>0
     * Post: Devuelve la longitud de la subsecuencia más larga de datos consecutivos de T de valor
     *       no decreciente.
     */
    public static int longSecuenciaMasLarga(int[] T) {
        return longSecuenciaMasLarga(T, 0, 1, 1);
    }

    /**
     * Pre: T.length>0 AND hasta>=0 AND hasta<=T.length
     *      AND longMayor es la longitud de la subsecuencia más larga de datos consecutivos de
     *      valor no decreciente de la subtabla T[0, hasta-longActual]
     *      AND longActual es la longitud de la subsecuencia más larga de datos consecutivos de
     *      valor no decreciente de T cuyo último dato es el dato T[hasta]
     * Post: Devuelve la logitud de la subsecuencia más larga de datos consecutivos de T de valor
     *       no decreciente.
     */
    private static int longSecuenciaMasLarga(int[] T, int hasta, int longActual, int longMayor) {
        if (hasta==T.length-1) {
            if (longActual<longMayor) {
                return longMayor;
            }
            else {
                return longActual;
            }
        }
        else {
            if (T[hasta+1]>=T[hasta]) {
                return longSecuenciaMasLarga(T, hasta+1, longActual+1, longMayor);
            }
            else {
                if (longActual<longMayor) {
                    return longSecuenciaMasLarga(T, hasta+1, 1, longMayor);
                }
                else {
                    return longSecuenciaMasLarga(T, hasta+1, 1, longActual);
                }
            }
        }
    }

}
```