Parcial 1 - PRÁCTICAS - PRG - ETSInf. Curso 2015-16
11 de abril de 2016. Duración: 1 hora

(Nota: El examen se evalua sobre 10 puntos, pero su peso específico en la nota final de la asignatura es de 0,8 puntos)

NOMBRE:	GRUPO DE PRÁCTICAS:

1. 2 puntos **Se pide:** responder a las siguientes preguntas sobre la práctica 1.

   a) Si sobre la torre `origen` hay 30 discos, ¿qué debe hacerse con los 29 discos que hay sobre el disco de mayor diámetro antes de poder moverlo a la torre `destino`?

   Solución: Moverlos a la torre `auxiliar`.

   b) En el método `hanoi`, para un número de discos mayor que 1, la primera llamada `moverDisco(origen, destino)` que se ejecuta, ¿está asociada a su caso base o al general?

   Solución: Está asociada a su caso base.

2. 3 puntos El método `posFracSep(String)`, visto en la práctica 2, determina recursivamente la posición del separador de la parte fraccionaria de un número en coma flotante bien formado contenido en una `String`, esto es, la posición del carácter punto `'.'` o coma `','` que aparezca o -1 en caso de que no exista.

   Se pide: completar con las instrucciones necesarias el método recursivo `posFracSep(String)` siguiente para que sea correcto, teniendo en cuenta que se ha optado por una descomposición recursiva descendente de la `String`, es decir, se ha considerado el último carácter de la `String` y la substring inicial (desde la posición `0` hasta la `s.length()-2`).

```
/** Devuelve la posición donde se encuentra el separador de la parte
 * fraccionaria o -1 si no se encuentra.
 * @param s String que contiene el valor en coma flotante.
 * @return int posición del separador o -1 si no se encuentra.
 * PRECONDICIÓN: s contiene un número en coma flotante bien formado. */
public static int posFracSep(String s) {
    int ult = s.length() - 1;
    if (s.charAt(ult) == '.' || s.charAt(ult) == ',') { return ult; }
    else { return posFracSep(s.substring(0, ult)); }
}
```

   Recuerda que `s.substring(i, j)` devuelve un objeto `String` que representa la substring de `s` formada con los caracteres comprendidos entre el `i` y el `j-1`.

Solución:

```
public static int posFracSep(String s) {
    if (s.length() == 0) { return -1; }
    else {
        int ult = s.length() - 1;
        if (s.charAt(ult) == '.' || s.charAt(ult) == ',') { return ult; }
        else { return posFracSep(s.substring(0, ult)); }
    }
}
```

3. [5 puntos] Se desea medir el tiempo promedio de ejecución de un algoritmo de ordenación de arrays de enteros, cuya ejecución, para un array `a`, se realiza mediante la llamada:

```
AlgoritmosMedibles.ordena(a);
```

Además, para mejorar la precisión de la medida, se deberá repetir la misma un número `numReps` de repeticiones.

Adicionalmente, se dispone de los métodos siguientes para inicializar un array según lo deseado:

```
private static int[] crearArrayAleatorio(int t)
private static int[] crearArrayOrdCreciente(int t)
private static int[] crearArrayOrdDecreciente(int t)
```

y se puede utilizar la rutina de temporización del sistema, tal como se ha visto en la práctica 3:

```
long t = System.nanoTime();
```

Se pide: escribir un método con la cabecera siguiente:

```
public static double tiempoMilisPromedio(int t, int numReps)
```

que devuelva como resultado el tiempo de ejecución en el caso promedio, en milisegundos, de la ordenación con el método `ordena(int[])` para una talla `t` y con un número de repeticiones `numReps`.

Debe tenerse en cuenta que se ha de elegir el método más conveniente para crear el array según el caso a medir y que no se debe volver a ordenar un array ya ordenado.

Solución:

```
public static double tiempoMilisPromedio(int t, int numReps) {
    final double NANOS_MILIS = 1e6;
    int[] a;
    long tIni, tFin;
    double acum = 0;
    for (int i = 0; i < numReps; i++) {
        a = crearArrayAleatorio(t);
        tIni = System.nanoTime();
        AlgoritmosMedibles.ordena(a);
        tFin = System.nanoTime();
        acum += tFin - tIni;
    }
    return acum / numReps / NANOS_MILIS;
}
```