Problemas de tratamiento de archivos

Problema 1.

Implementar la clase CharCounter descrita en los apuntes.

Problema 2.

A partir de la solución al problema del cifrado de César, implementad un programa tal que dado dos nombres de ficheros (uno para la entrada y otro para la salida) y un valor para la clave, encripte el contenido del fichero de entrada y lo guarde en el segundo.
Podéis hacer dos versiones:

- Una que trabaje carácter a carácter usando FileReader
- Una que trabaje línea a línea usando BufferedReader

Ampliad la versión del programa de cifrado, para que mantenga mayúsculas y minúsculas (la solución que vimos transformaba todo a mayúsculas).

Problema 3.

La foto-finish. En cualquiera de las carreras (ya sea tortugas, tortugas 2.0 o pacmans) haced un programa para guardar la foto-finish, es decir, cuando la carrera acabe, guardad en un fichero las posiciones de cada uno de los corredores. Haremos dos versiones, dependiendo del formato.

En la primera, el formato será:

- los datos de cada corredor acabarán con el carácter ';'
- cada dato estará separado del siguiente con un carácter ','

Por ejemplo: 234.8,0.0; 312.3,78-0;

En la segunda, el formato será:

- la información de cada corredor estará en una línea
- en cada línea separaremos los datos de un corredor con el carácter ','

Para escribir en el fichero, como se trata de ficheros de texto, deberéis escribir tanto enteros como números en coma flotante (double) pasándolos previamente a String. Para ello podéis usar los métodos de la clase String:

- static String valueOf(int i)
- static String valueOf(double d)

Problema 4.

Haced el programa "revele" las fotos creadas por el programa anterior, es decir, que lea el fichero y reconstruya la foto.
Cuando leáis el fichero obtendréis cadenas, para obtener, a partir de la cadena, un entero o un double, podéis usar los métodos:

- static int parseInt(String s), de la clase Integer
- static double parseDouble(String s), de la clase Double

Haced una versión para cada uno de los formatos.

Problema 5.

Haced un programa que escriba 10 objetos gráficos al azar (rectángulos, ovales, de diversos tamaños, posiciones y colores) en la pantalla y que los guarde en un fichero de texto (con un formato similar al del problema de la foto-finish).
Haced también el programa que lee el fichero generado y reconstruye la imagen.
Como el programa es independiente de la manera de crear el fichero de texto, probar de usarlo con un fichero que habéis "creado" usando un editor de texto.

Problema 6.

Dado un fichero de texto separado por líneas que representa una tabla de datos, haced un programa que lo transforme en un fichero de HTML que podamos visualizar en el navegador.
Por ejemplo, el fichero de datos de entrada tendrá la forma:

```
Nombre, Problema 1, Problema 2, Problema 3
Juan,8.3,6.4,8.2
.....
```

Es decir:

- la primera línea contendrá, separadas por comas, las cabeceras de cada una de las columnas
- las posteriores líneas contiene, separadas por comas, los valores para todas las columnas

El fichero de salida será:

```
 1 <html>
 2   <head><title>Notes</title></head>
 3   <body>
 4    <table>
 5      <tr>
 6         <th>Nombre</th>
 7         <th>Problema 1</th>
 8         <th>Problema 2</th>
 9         <th>Problema 3</th>
10      </tr>
11      <tr>
12         <td>Juan</td>
13         <td>8.3</td>
14         <td>6.4</td>
15         <td>8.2</td>
16      </tr>
17         ...
18       </table>
19   </body>
20 </html>
```

Es decir:

- 1-4: parte inicial
- 5-10: transformación de la primera línea del fichero con tantas <th> y </th> como elementos haya en ella
- 11-16: transformación de cada línea subsiguiente del fichero, con tantos <td> y </td> como datos haya en ella
- 18-20: parte final

Consideraciones:

- Podéis suponer que el fichero está bien formado, es decir, que contiene como mínimo una línea de cabeceras y una de datos, y que cada línea de datos contiene tantos datos como cabeceras hay
- Aplicad diseño descendente y descomponed, descomponed, ...

Como siempre, podéis complicar el problema añadiendo una columna nueva para el promedio de cada línea y que sea calculada por vuestro programa.

Problema 7.

Dado un fichero de datos con el formato:

$$x_{min}, x_{max}, y_{min}, y_{max}$$
$$x_0, y_0$$
$$\ldots$$

Es decir,

- Una primera línea que indica los valores máximos y mínimos para los dos ejes de coordenadas
- Las líneas posteriores contienen los pares de coordenadas para cada punto

Se pide que construyáis un programa que muestre un gráfico con los puntos del fichero.
Como siempre podéis hacer varias versiones con diferentes grados de dificultad:

- Sin dibujar ejes y ocupando toda la pantalla
- Añadid una columna a los valores que sea un entero y que indique el tipo (p.e. 0, 1, 2, ...) y usar diferentes colores para tipos diferentes.
- Dibujando unos ejes (dejando márgenes pequeños)
- Dibujando ejes y añadiendo algunos valores de referencia (p.e. los mínimos y máximos o alguno intermedio)

Problema 8.

La búsqueda binaria en archivos de acceso directo. Haced un programa que, aprovechando el ejemplo visto en los apuntes:

- cree un archivo de personas ordenado crecientemente por identificador. Como luego usaremos ese identificador para hacer búsquedas binarias:
  - no uséis identificadores consecutivos ya que así no podremos buscar cosas que no existan. Algo tan simple como que el registro de posición i tiene identificador 2*i ya deja huecos.
  - no dejéis espacios registros vacíos en medio del archivo.
- implementad el método:

  Person findById(long id)

  tal que dado un id, devuelva la persona correspondiente a ese id (si existe) y, en caso de no existir, devuelva null.

Problema 9.

Basándoos en la implementación dada en los apuntes del MergeSort, haced una versión que funcione con un fichero de acceso directo de personas. Usad como criterio de ordenación, el orden creciente por id.

Problema 10.

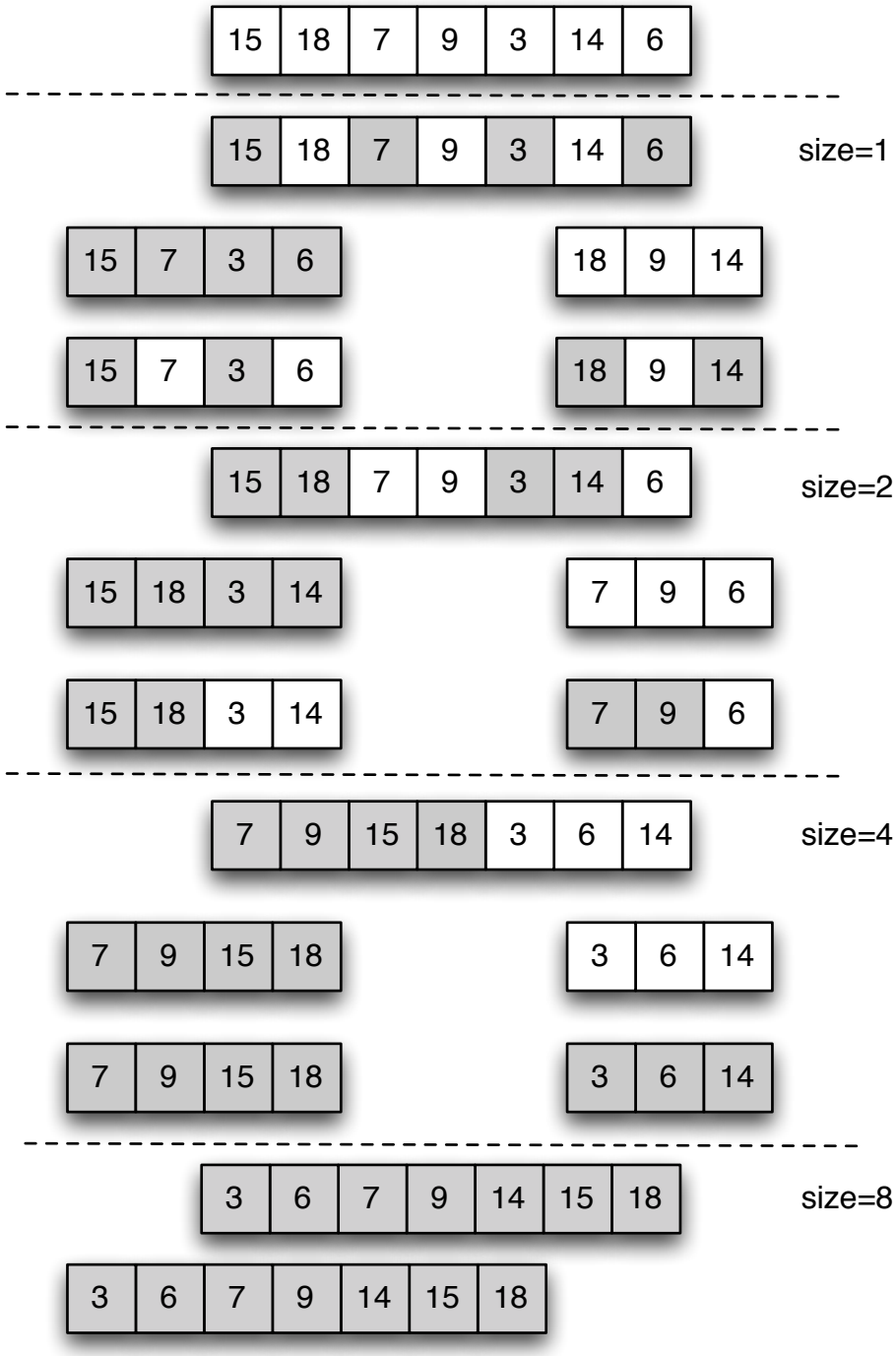
Modificad el algoritmo de MergeSort para que sea también capaz de detectar si el resultado ya está ordenado en el método merge (ahorrándonos así un split adicional).

Problema 11.

El MergeSort balanceado. Como se comenta en los apuntes, la versión que hemos presentado del MergeSort, aunque intuitiva, no es la mejor. En este ejercicio proponemos una variación que aumenta espectacularmente el rendimiento.
La idea es muy simple,

- En el paso Split, en vez de tener en cuenta si dos elemtos estás desordenado, agrupamos por bloques de tamaño fijo. En el primer Split el tamaño es 1, en el siguiente 2, en el siguiente 4, etc.
- En el paso de merge, mezclamos los elementos, pero manteniendo los grupos. Es decir, el primer merge fusionará un grupo de tamaño 1 del primer fichero con otro de tamaño 1 del segundo fichero, así hasta acabarlos. En el siguiente merge se fusionarán un grupo de tamaño 2 del primer fichero, con otro de tamaño 2 del segundo fichero, hasta que no queden más grupos.
- La secuencia de Split y merge se irá repitiendo, doblando cada vez el tamaño de los grupos, hasta que solamente quede uno.

Fijaos en que, si procedemos de esta manera: **en cada paso de split+merge nos aseguramos que doblamos el tamaño de las subsecuencias ordenadas.**


15 18 7 9 3 14 6
15 18 7 9 3 14 6
size=1
15 7 3 6
18 9 14
15 7 3 6
18 9 14
15 18 7 9 3 14 6
size=2
15 18 3 14
7 9 6
15 18 3 14
7 9 6
7 9 15 18 3 6 14
size=4
7 9 15 18
3 6 14
7 9 15 18
3 6 14
3 6 7 9 14 15 18
size=8
3 6 7 9 14 15 18