Ejercicios del Tema 4: Entrada y salida. Flujos y ficheros

Profesores de PRG
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

1. Construir un programa Java que reciba como argumento de línea de comandos la ruta a un fichero y que muestre por pantalla información básica sobre el mismo (como mínimo el nombre del fichero, directorio donde se encuentra y su tamaño expresado en kbytes).

2. Escribir un método estático que escriba en un fichero binario los números del 1 al 999.

3. Escribir un método estático que lea el fichero generado por el programa del ejercicio 2 y sume dichos números. Comprobar que el resultado es correcto implementando un bucle adicional que realice dicha suma.

4. Construir un programa que escriba en un fichero de texto los números del 1 al 999 y posteriormente los vuelva a leer de ese fichero para realizar la suma de los mismos. Verificar que el resultado es correcto. Comprobar la diferencia de tamaños entre el fichero generado en el ejercicio 2 y el generado por este ejercicio.

5. Construir un programa que permita buscar palabras en un fichero de texto. Se debe mostrar el número de línea y su contenido, para cada línea que contenga la palabra buscada.

6. Desarrollar un programa que permita eliminar todas las ocurrencias de una palabra dada en un fichero de texto. El programa recibirá como argumentos de línea de comandos la ruta al fichero así como la palabra en cuestión. Este código producirá automáticamente un nuevo fichero con la siguiente nomenclatura: Si el fichero de entrada se llama *fichero.txt*, el fichero generado se llamará *fichero_2.txt*.

7. Escribir un método estático que reciba como entrada el nombre de un fichero de texto y devuelva estadísticas básicas sobre el mismo (como mínimo se debe incluir el número de palabras, el número de caracteres totales del texto y la longitud media de una palabra medida en nº de caracteres).

8. Modificar el programa ejemplo de la agenda telefónica (Capítulo 16 - Fig. 16.15, 16.16 y 16.17) de forma que:

   - un elemento individual de la agenda (un `ItemAgenda`) mantenga además del nombre, teléfono y código postal de un contacto su dirección postal. Tras hacerlo, ¿es necesario modificar algo en las operaciones de lectura y escritura en fichero de la clase `Agenda`?
   - Escribir operaciones en la clase `Agenda` para efectuar una búsqueda de un contacto por nombre o teléfono. Ambas operaciones devolverán el primer `ItemAgenda` que cumpla la condición en caso de que exista o `null` en el caso de que no sea así.
   - Crear un nuevo programa principal que, mediante el uso de un menú, permita almacenar la agenda en curso en un fichero, añadir un nuevo contacto cuyos datos se pedirán al usuario, eliminar un contacto dado su número de teléfono y, finalmente, recuperar una agenda desde un fichero dado.

9. Una estación meteorológica necesita gestionar las medidas diarias de la pluviosidad en una determinada zona a lo largo de un año con las siguientes características:

   - Se ha decidido construir una clase, denominada `Pluviometro` que tenga como atributos dos arrays, uno para almacenar el número de días de cada mes y otro para guardar las medidas de pluviosidad de dichos días. Por comodidad para el programador se ha decidido prescindir de usar la posición 0 de los arrays, para que el índice coincida con el número de mes o el número de día, de manera que las posiciones `[0]` de los arrays no se usarán.
   - `diasM` es un array de 13 `int` tal que `diasM[i]` es el numero total de días del mes `i` siendo `1<=i<=12`, de tal manera que como se ha comentado, `dia[0]` no se usará.
   - `lluvia` es un array bidimensional con 13 filas. `lluvia[i]` es un array de `diasM[i]+1` valores de tipo `double` (dicho de otra manera su longitud es `lluvia[i].length==diasM[i]+1`) tal que `lluvia[i][j]` representa la medida del día `j` del mes `i`, siendo `1<=i<=12` y `1<=j<=diasM[i]`. Las posiciones `lluvia[0][j]` y `lluvia[i][0]` no se usarán.

   Se pide escribir un programa con la siguiente funcionalidad:

   a) leer los datos de pluviosidad desde un fichero *pluvio.dat* en el que cada línea tiene el siguiente formato:

   ```
   dia mes medida
   ...
   ```

   y almacenarlos en la matriz `lluvia`, validando los valores de día y mes leídos. Las medidas no tienen por qué estar ordenadas cronológicamente. Un ejemplo de algunas líneas del fichero es el siguiente:

   ```
   ...
   24 11 312.12
   15 3 6.756
   14 8 12.5
   15 1 31.3
   16 3 212.0
   17 3 87.9
   18 3 3.56
   23 6 11.11
   ...
   ```

   b) dada la matriz `lluvia` y cierto mes `m`, determinar la cantidad máxima llovida en un solo día a lo largo de dicho mes así como el día en que esta se produjo.

   c) dada la matriz `lluvia`, cierto mes `m` y una cantidad `lt` de litros, determinar un día de dicho mes en que la pluviosidad haya superado dicha cantidad. Si no existe, indicarlo con un mensaje.

   d) dada la matriz `lluvia` y cierto mes `m`, determinar si hubo al menos tres días consecutivos en dicho mes con una pluviosidad mayor a 100 litros cada uno de ellos.

   e) mostrar por pantalla las medidas del fichero de entrada *pluvio.dat* pero ordenadas cronológicamente.

10. Se desea modificar la solución al problema anterior, de forma que la lectura de los datos se produzca de un fichero binario (un `DataInputStream` o un `ObjectInputStream`) en lugar de un fichero de texto, tal y como se planteó antes.

    La lectura de los datos se efectuará en tríadas de valores, enteros los dos primeros, que representarán, respectivamente, el día y mes de la medida; siendo el tercer valor uno en coma flotante (un `double`) que contendrá la cantidad llovida.

    Para determinar el momento en el que se produzca el final del fichero, se **deberá utilizar** la excepción `EOFException` tal y como se menciona en el Capítulo 16 (Fig. 16.18).

11. Se tienen los siguientes datos referentes a la última vuelta ciclista local:

    - `ciclistas`: array con los nombres de cada ciclista.
    - `tiempos`: matriz en la que en cada fila `i` se tienen los tiempos de `ciclistas[i]` en cada una de las cinco etapas, el tiempo máximo empleado en una etapa es 180 minutos (se consideran valores enteros).

    Se pide escribir un programa con la siguiente funcionalidad:

    a) diseñar la clase `VueltaCiclista` que tenga como atributos los arrays anteriormente mencionados.

    b) leer los datos de un fichero de texto con el formato:

    ```
    nº de participantes
    nombre t1 t2 t3 t4 t5
    otronombre t1 t2 t3 t4 t5
    ...
    ```

    c) dado el nombre de un ciclista, mostrar por pantalla los tiempos empleados por este en cada una de las etapas si ha participado o el mensaje *"No ha participado en esta vuelta"* en caso contrario.

    d) mostrar por pantalla el nombre del ciclista ganador de la vuelta y el tiempo que este empleó. Gana la vuelta el ciclista cuya suma de tiempos de las cinco etapas es menor.

    e) mostrar por pantalla los ciclistas y sus tiempos ordenados según el tiempo empleado.

    En todos los casos, los tiempos se mostrarán en horas y minutos.

12. Para resolver el problema anterior desde la *perspectiva de la programación orientada a objetos*, se plantea la siguiente organización de la información, que deberá ser implementada adecuadamente:

    - Una clase `Ciclista`, mediante la que se mantendrá información relativa a cada uno de los mismos, en particular su `nombre` así como sus `tiempos`, array de 5 elementos enteros en los que, en cada uno de ellos, se mantendrá el tiempo empleado por el ciclista en cubrir la etapa correspondiente (el tiempo máximo empleado en una etapa es 180 minutos).

      Se deberá diseñar esta clase definiendo, además, los métodos constructores, consultores y modificadores que se consideren pertinentes.
    - Una clase `VueltaCiclista` que contendrá un array `ciclistas`, de elementos de la clase `Ciclista`. Se considerará que el array tiene los elementos estrictamente necesarios; esto es, no existen posiciones del array no ocupadas.

      Se deberá diseñar esta clase definiendo, además de los métodos que se consideren pertinentes (tales como constructores, etc.), métodos para almacenar y recuperar los datos de una `VueltaCiclista` en y desde un fichero de objetos (esto es, usando `ObjectInputStream` y `ObjectOutputStream`, así como los métodos necesarios para, al igual que en el problema anterior:

      - dado el nombre de un ciclista, mostrar por pantalla los tiempos empleados por este en cada una de las etapas si ha participado o el mensaje *"No ha participado en esta vuelta"* en caso contrario.
      - mostrar por pantalla el nombre del ciclista ganador de la vuelta y el tiempo que este empleó. Gana la vuelta el ciclista cuya suma de tiempos de las cinco etapas es menor.
      - mostrar por pantalla los ciclistas y sus tiempos ordenados según el tiempo empleado.

13. Si se han resuelto los dos problemas anteriores, se está en posición de discutir las mejoras (y tal vez inconvenientes) que haya podido introducir la *solución orientada a objetos*.

    Se pide señalar las diferencias más significativas entre las dos soluciones al problema de la vuelta ciclista, desde el punto de vista del almacenamiento y recuperación de la información en memoria externa. Tratar de responder a la cuestión planteada, determinando cómo la posible variación de elementos en las clases, altera la organización de los ficheros y/o de las operaciones encargadas de su lectura o escritura.

    ¿Cuál de las organizaciones de los datos parece más cómoda para trabajar si tiene que sufrir modificaciones posteriores?

14. Se desea gestionar la información sobre los visitantes a cierto parque de atracciones *Gran Aventura*:

    - de cada `visitante` se conoce sus apellidos, nombre, edad y un código como, por ejemplo, "`GA325`".
    - Existen `atracciones` que tienen cierto nombre, tales como: *DragonKhan*, *Furius*, *TutukiSplash*, *Stampida*.
    - Además, se conocen los visitantes que han participado en cada una de las atracciones ya que se mantienen los códigos de aquellos visitantes que hayan accedido a cada una de las mismas.

    Se pide escribir un programa para la gestión básica del parque de atracciones para lo que se deberá:

    a) diseñar una clase `Visitante`, para mantener los datos individuales de cada uno de ellos. La clase deberá incluir los métodos de gestión (constructores, consultores y modificadores que se consideren pertinentes).

    b) diseñar una clase `Atraccion` mediante la que se mantenga sus elementos propios, tales como su `nombre` y una lista de los visitantes que han accedido a los largo de un día a la misma (dicha lista se puede implementar mediante un array parcialmente completo de valores de tipo `Visitante`).

    Esta clase contendrá, por lo menos, dos métodos, uno para añadir un nuevo visitante a los que han accedido a la atracción y otro para determinar si dado un código de visitante, ha accedido o no a la misma,

    c) diseñar una clase `ParqueAtracciones`, mediante la que se mantenga una lista, implementada una vez más mediante un array, de las atracciones que están en funcionamiento a lo largo de un día. Esta clase deberá contener, por lo menos, los siguientes métodos:

    - Métodos para almacenar y recuperar en un fichero de objetos los datos correspondientes a todo el parque de atracciones en un momento dado. Estos métodos recibirán como parámetro el nombre del fichero que contendrá los datos.
    - Un constructor de la clase `ParqueAtracciones` que construirá uno de tales objetos a partir de los datos incluidos en un fichero de objetos (cuyo nombre recibirá el constructor como parámetro). Este constructor deberá utilizar el método diseñado en el punto anterior para recuperar la información de un parque de atracciones.
    - Y, además, métodos para:
        - mostrar el número de visitantes que han accedido a cada una de las atracciones.
        - comprobar si un determinado visitante del que se conocen sus apellidos ha accedido a una determinada atracción.
        - mostrar los datos del `visitante` más joven de una atracción determinada.
        - mostrar por pantalla la lista de atracciones a las que ha accedido un determinado `visitante` del que se conocen sus apellidos.

15. En la clase `Agenda`, que se muestra en la figura 16.16 del Capítulo 16, se ha definido el método de clase `leerAgenda(String)` que, a partir del nombre de un fichero, devuelve el objeto `Agenda` que se encuentra almacenado en el mismo.

    Definir, utilizando el método anterior, un constructor de la clase `Agenda` que recibiendo como argumento el nombre del fichero en el que se ha almacenado el objeto `Agenda`, construya un objeto de dicho tipo. Modificar la clase `GestorPrueba` (Fig. 16.17) para utilizar el nuevo constructor.