Programación II

Curso 24-25

Práctica 1

0 Cuestiones previas

Todos los programas que escribas deben respetar un correcto estilo de programación, sangrando adecuadamente el código fuente, eligiendo los identificadores según las recomendaciones ya conocidas, y añadiendo comentarios apropiados para explicar lo que haces.

Crea el directorio `p1` dentro del directorio `Practicas`.

1 Actividad 1

Antes de realizar la siguiente actividad deberías haber leído el documento previo asociado a esta práctica ("**String y Scanner**"), prestando especial atención a:

- La descripción de la clase `Scanner` para leer líneas del teclado (sección 2 del citado documento).
- La descripción de la API de la clase `String` (sección 1.2 del citado documento).

1.1 Tarea 1a

Escribe el programa `EncriptaCadena.java`, que debe hacer lo siguiente:

1. Lee una línea del teclado y después la imprime.
2. Informa de su longitud, e imprime su versión en mayúsculas, y luego en minúsculas.
3. Imprime por pantalla la línea encriptada según el siguiente procedimiento: cambiará cada carácter sumándole su posición (cadena[i] = cadena[i] + i). Por ejemplo, si la cadena es "ahora", la 'a' (posición 0) se cambiará por 'a'+0 = 'a', la 'h' (posición 1) se cambiará por 'h'+1 = 'i'….

   Pista: Emplea `charAt` para averiguar el carácter de cada posición. Recuerda que las operaciones matemáticas se hacen entre números. Y, si quieres imprimir un carácter en lugar de un número, tendrás que convertir su tipo de número a carácter (recuerda el *cast* de C, en Java funciona igual).

Guarda el programa en `p1`. Sitúate en el terminal en ese mismo directorio p1 y compila el programa creando el fichero **.class** en el propio directorio `p1`. Ejecuta el programa y comprueba que hace lo que se pide.

1.2 Tarea 1b

Escribe el programa `ModificaCadena.java`, que debe hacer lo siguiente:

1. Lee una línea del teclado y después la imprime.
2. Informa de si la línea contiene o no la cadena "nueve", y de si es menor o mayor alfabéticamente que "nueve". Luego cambia cualquier aparición de "nueve" por "9" e imprime el resultado.

   Pista: Revisa los métodos `contains, compareTo` y `replaceAll` de la clase `String.`
3. Cambia la primera y última aparición del carácter '0' (si lo hay) por la cadena "cero", e imprime el resultado.

   Pista: Averigua la posición del '0' (`indexOf` y `lastIndexOf`) y crea las subcadenas anterior y posterior (`substring`) para construir el resultado.

Guarda el programa en `p1`. Sitúate en el terminal en ese mismo directorio p1 y compila el programa creando el fichero **.class** en el propio directorio `p1`. Ejecuta el programa y comprueba que hace lo que se pide.

2 Actividad 2

Antes de realizar la siguiente actividad deberías haber leído el documento previo asociado a esta práctica ("**Arrays**"), prestando especial atención a:

- La declaración, creación y uso de un array unidimensional (sección 1).
- El uso de la clase `Array` para diversos procesamientos sobre arrays, como ordenar un array (sección 2).
- La declaración, creación y uso de un array multidimensional (sección 3).

2.1 Tarea 2a

Escribe el programa `LeeArray.java`, que debe hacer lo siguiente:

1. Lee del teclado 5 números enteros, cada uno en una línea, y los va almacenando, conforme los lee, en un array de enteros. Al terminar, recorre el array e imprime los números, en una línea.

   Pista: recuerda el método `Integer.parseInt()`

2. Ordena el array, y vuelve a imprimirlo mediante una única instrucción.

   Pista: revisa la clase `Arrays` (sección 2.1.2 del documento previo).

Guarda el programa en `p1`. Ejecuta el programa y comprueba que hace lo que se pide.

2.2 Tarea 2b

Escribe el programa `ProcesaClasspath.java`, que debe hacer lo siguiente:

1. Lee una línea del teclado que sigue el formato del CLASSPATH en Linux (una serie de rutas -compuestas por uno o más directorios- o ficheros **.jar** separados por el carácter ':').

   `/ruta1:/ruta2:/ruta3/lib.jar:/ruta4:…:/ruta`

2. Divide la cadena en sus distintos componentes (los elementos separados por ':'), muestra su número, y los imprime cada uno en una línea.

   Pista: ver el método `split` de la clase `String.`

3. Informa de cuántos componentes son directorios, cuántos ficheros **.jar**, y cuántos componentes pertenecen al directorio `/usr` (es decir, cuántos son descendientes de `/usr`).

   Pista: ver los métodos `startsWith` y `endsWith` de la clase `String.`

Guarda el programa en `p1`. Ejecuta el programa y comprueba que hace lo que se pide.

2.3 Tarea 2c

Escribe el programa `ProcesaClasspath2.java`, que debe hacer lo siguiente:

1. Lee una línea del teclado que sigue el formato del CLASSPATH, y divide la cadena en sus distintos componentes (el mismo formato de la tarea anterior).
2. Crea un array bidimensional, donde el número de filas se corresponde con el número de componentes.
3. Procesará cada componente uno a uno, dividiéndolo en sus distintos subcomponentes.

   Pista: será otro `split`, siendo esta vez el separador el carácter '/'.

   a. Para cada fila del array, creará la segunda dimensión de acuerdo al tamaño de la lista de subcomponentes del componente que va en esa fila.

   b. Guardará cada subcomponente en la fila y columna que le corresponda.

   Importante: Fíjate que el método `split` devuelve ya un array.

4. Recorrerá el array bidimensional fila a fila y columna a columna e imprimirá los distintos componentes, uno en cada línea (los distintos subcomponentes de cada línea se separarán por el carácter '-').

Por ejemplo, si el CLASSPATH es: "`/dir1/dir2:/dir3/dir4/lib.jar`"

- Se creará un array bidimensional de 2 filas.
- La primera fila será un array de 2 componentes (“dir1” y “dir2”)
- La segunda fila será un array de 3 componentes (“dir3”, “dir4” y “lib.jar”)

Guarda el programa en p1. Ejecuta el programa y comprueba que hace lo que se pide.

3 Actividad 3

Antes de realizar la siguiente actividad deberías haber leído el documento previo asociado a esta práctica (“**Ficheros**”): revisa la descripción de la clase `File` (sección 1 del documento previo) y cómo escribir en un fichero de texto (sección 2).

3.1 Formato de la información sobre cotrabajadores

En esta práctica trabajaremos con ficheros que almacenan información sobre personas que trabajan en un centro de *coworking* (cotrabajo). Para simplificar, sólo trabajaremos con particulares y asalariados (estos últimos pueden ser trabajadores de base o ejecutivos), y, de forma general, los llamaremos cotrabajadores o *coworkers*. La información que describe cada *coworker* es la siguiente:

- **T** (*Type*: su tipo). Será un carácter: ‘P’ (particular), ‘E’ (asalariado) o ‘X’ (ejecutivo).
- **I** (*Id*: su identificador único): Será una cadena con el DNI de la persona (DDDDDDDDL, 8 dígitos y una letra).
- **N** (*Name*: su nombre). Será una cadena de caracteres compuesta sólo por letras.
- **S** (*Seniority*: la antigüedad -meses- de la persona en el centro): un número real positivo (*float*) menor o igual que 30, que representa el número de meses que este cotrabajador ha tenido ocupada alguna oficina sumando sus diferentes contratos.
- **C** (*Country*: el país de origen del cotrabajador): Será una cadena de un máximo de 20 caracteres.
- **CC** (*CreditCard*: el número de la tarjeta de crédito, sólo para particulares). Una cadena de texto según el formato DDDD-DDDD-DDDD-DDDD, o sea, cuatro grupos de 4 dígitos, separados por guiones.
- **CO** (*Company*: nombre de la empresa, sólo para asalariados). Una cadena de 30 caracteres máximo.
- **BA** (*BankAccount*: el IBAN o cuenta bancaria de la empresa, sólo para asalariados). Una cadena según el formato de un IBAN, es decir 2 letras seguidas de 22 dígitos (sin guiones de separación).
- **PI** (*PassId*: Identificador de la tarjeta de acceso 24h al centro, solo para ejecutivos). Una cadena de 6 dígitos.
- Al final, opcionalmente, la lista de fechas en los que ha ido a trabajar al centro de *coworking* en el último año.

Cada *coworker* se describirá con una línea en el fichero, de acuerdo a los siguientes formatos según el tipo de *coworker* (el tamaño de la lista de fechas es desconocido, y puede ser cero):

- P;I;N;S;C;CC;F1;F2…
- E;I;N;S;C;CO;BA;F1;F2…
- X;I;N;S;C;CO;BA;PI;F1;F2…

Si una línea del fichero comienza por ‘#’ (el primer carácter distinto de un espacio, tabulador…), se trata de un comentario, que puede aparecer al principio del fichero, al final, o entre líneas. Por ejemplo, el contenido del fichero podría ser (**fíjate que el separador de decimales es una coma**):

```
  # Lista de coworkers
P;11118888X;David;1,1;Spain;1111-2222-3333-4444;02/04/2024;30/11/2024
E;22228888Y;Amelia;5,3;Spain;Vodafone;ES1234205555000011112222;09/08/2024
X;33338888Z;Oscar;2,6;France;Orange;ES1234201111222233334444;987123
# Fin de la lista
```

3.2 Tarea 3a

Escribe el programa `WriteCoworkersFile.java`, que hace lo siguiente:

1. Recibe un argumento que representa el nombre de un fichero (por ejemplo, '`coworkers.txt`').
2. Comprueba si el fichero ya existe. Si existe, avisará al usuario de que su contenido se va a perder, y le preguntará si quiere continuar. Si el usuario responde que no, el programa terminará (ejecutando el método `System.exit()`).
3. Crea un fichero de texto con ese nombre y guarda en él la información sobre varios trabajadores de distinto tipo, una línea por trabajador, de acuerdo al formato anterior (puedes inventarte los datos de los trabajadores o pedirlos al usuario por teclado, línea a línea o campo a campo, indicando la finalización como prefieras). Añade algún comentario al fichero, al principio, al final y donde creas oportuno.

Recuerda que, cuando uses algunas clases relacionadas con ficheros, tendrás que capturar o propagar las excepciones que se puedan producir, tal como se describe en el documento previo sobre ficheros.

Guarda el programa, compílalo, ejecútalo, y comprueba que hace lo que se pide.

3.3 Tarea 3b

Escribe el programa `FileInfo.java`, que hace lo siguiente:

1. Recibe un argumento que representa el nombre de un fichero.
2. Obtiene la información que pueda sobre ese fichero y la imprime en pantalla. Por ejemplo, si existe el fichero, su nombre completo (incluida su ruta), su tamaño, si se puede leer/escribir/ejecutar, y la última vez que fue modificado (emplear, por ejemplo, la clase `java.util.Date` para mostrar el valor devuelto por el método `lastModified()`).
3. Cambia su nombre por otro.

Guarda el programa, compílalo, ejecútalo, y comprueba que hace lo que se pide.