Más servicios del catálogo de discos

Continuarás con algunos de los métodos que aparecen en la tarjeta de responsabilidades y que no has programado todavía.

Método que muestra el histórico de transmisiones de un disco

En la tarjeta de responsabilidades el método que muestra el contenido del arreglo histórico correspondiente a un disco particular es como sigue:

Nombre	Salida	Entradas	Descripción
mstraHist	Cadena con el histórico del disco	posición del disco	Muestra la lista de transmisiones iniciadas y finalizadas para ese disco.

A partir de esta descripción puedes hacer la documentación de Javadoc y el encabezado de este método:

```
466  /**
467   * Muestra el historico de transmisiones que solo incluye
468   * las iniciadas que ya fueron terminadas de un disco dado.
469   * @param cual Elige al disco del que va a mostrar el historico.
470   * @return cadena con el historico bien presentado.
471   */
472  public String mstraHist( int cual ) {
```

Como siempre primero verificas que el argumento sea correcto: que elija uno de los discos en el catálogo. Si no es válido debe regresar un mensaje que avise que el disco no existe:

```
473     if (cual < 0 || cual >= numDiscos // si la posicion no es valida
474           || catalogo[cual] == null) // la posicion no contiene nada
475       return "Este␣disco␣no␣existe";
```

Si la ejecución llega a la línea 476 es porque la posición del disco es válida. Se inicia una cadena con el nombre del disco y un salto de renglón a continuación del nombre:

```
476     String cadena = "Historico␣del␣disco\t"
477       + catalogo[cual].getNOMBRE ( ) + "\n";
```

Incluyes en el listado tabuladores y cambios de línea para hacer más clara la impresión.

Verificas que el disco tenga fechas registradas y si no es así, avisar que ese disco no tiene nada en su histórico:

```
478     if (numHist[ cual ] == 0) { // No hay terminaciones registradas
479       cadena += "Este␣disco␣no␣tiene␣historico\n";
480       return cadena;
481     }
```

Si es que hay registros históricos de este disco procedes a recorrerlos con un `for`, tomando cada pareja de fecha de inicio y fecha de fin que haya registradas, respectivamente en el renglón 0 (cero) y en el renglón 1 (uno), usando para ello el método de clase `daCalndrio`:

```
482    for (int i = 0; i < numHist[cual]; i++) {
483      cadena += "[" + i + "]\tTransmision␣iniciada:␣\t"
484        + daCalndrio( historico[cual][0][i]) // inicio
485        + "\n␣␣␣␣\tTransmision␣terminada:␣\t"
486        + daCalndrio( historico[cual][1][i]) + "\n\n"; // fin
487    }
```

Una vez terminado el recorrido del histórico del disco y agregadas las fechas a la cadena, procedes a regresar la cadena.

```
490    return cadena;
491  }
```

Método que muestra el histórico de transmisiones de todos los discos

En la tarjeta de responsabilidades tienes la descripción de este método como sigue:

Nombre	Salida	Entradas	Descripción
mstraHistrs	Cadena con el histórico de todos los discos	(nada)	Muestra la lista de transmisiones iniciadas y finalizadas para todos los discos

De esta descripción puedes hacer la documentación de Javadoc y el encabezado del método:

```
491  /**
492   * Muestra el historico de transmisiones que solo incluye
493   * las iniciadas que fueron terminadas de aquellos discos
494   * que tienen historico.
495   * @return cadena con el historico bien presentado.
496   */
497  public String mstraHistrs ( ) {
```

Lo único que tienes que hacer en este método es recorrer el arreglo históricos y, para los discos que tengan un número mayor que cero en su posición en `numHist` procede a agregarlo a la cadena, usando el método que acabas de programar. Inicias una cadena con el encabezado adecuado:

```
498    String cadena = "Historico␣en␣los␣discos␣que␣lo␣tienen:\n"
499                + "=====================================\n";
```

Después procedes a recorrer el arreglo `historico` para aquellos discos que tienen registros:

```
500    for ( int cual = 0; cual < numDiscos; cual++) { // Recorrer cada disco
501      if (catalogo[cual] != null) // Si el disco existe
502        cadena += mstraHist ( cual ) + "\n"; // procesa, si hay, su historico
503    }
```

Una vez terminado el recorrido, regresas la cadena que armaste:

```
506     return cadena;
507   }
```

Método auxiliar **pideNum**

Este es un método que ya usaste y que vas a usar frecuentemente para pedirle al usuario, mostrándole un mensaje de texto, enteros que tienen que estar en un cierto rango, aprovechando el canal de comunicación (Scanner) que ya tenga la clase con el usuario. No tiene una descripción en la tarjeta de responsabilidades, porque es un método auxiliar, pero como ya lo usaste y se acaba de explicar cómo funciona, puedes dar la documentación de Javadoc:

```
507   /**
508    * Se comunica con un usuario y le solicita un entero que
509    * este en ciertos rangos. Le da un mensaje al usuario
510    * indicando lo que debe proporcionar y los limites que
511    * debe observar.
512    * @param cons un Scanner a traves del que se comunica la
513    *             clase con el usuario.
514    * @param msg el mensaje con el que pide el dato este metodo.
515    * @param minimo menor valor aceptado.
516    * @param maximo mayor valor aceptado.
517    */
```

Como el método recibe como parámetro todo lo que necesita para funcionar y no usa o modifica ningún atributo de objeto, el método va a ser público de la clase. De lo anterior, el encabezado es como sigue:

```
518   public static int pideNum(Scanner cons, String msg,
519                             int minimo, int maximo) {
```

Declaras un entero con valor inicial -1, ya que este es un valor inválido, por si el usuario no da algo correcto:

```
520     int num = -1; // Si no lee nada, tiene valor incorrecto
```

El método escribe el mensaje al usuario, pidiéndole lo que diga el mensaje:

```
521     System.out.print(msg + "\nterminando␣con␣un␣[enter]:␣-->");
```

Como el enunciado es un `print`, el cursor en la pantalla se queda donde termina la línea que esté escribiéndose durante la ejecución. Procedes a capturar el entero que te den, usando el método `readInt` de la clase `Scanner`:

```
522     num = cons.nextInt();
```

Como pediste un [Enter] y al leer un entero el [Enter] no se descarta, das lectura a una línea para procesar el [Enter], pero no lo usas:

```
523     cons.nextLine();
```

Una vez leído un dato, procedes a verificar que ese dato esté dentro de los límites. Si no es así, asignas un -1 para denotar que el dato dado por el usuario fue incorrecto. Si está dentro de los límites no le haces nada al dato dado por el usuario.

```
524     if (num < minimo || num > maximo) // fuera de rangos
525       num = -1;
```

En este punto el método tiene ya un valor en `num` que es el va a regresar; ya sea el dado por el usuario o -1:

```
526     return num;
527   }
```

Con esto terminamos la tercera lección. En la siguiente lección revisaremos la condicional enumerativa (`switch`) y otros tipos de iteraciones (`while` y `do ... while`).