Constructores del Catálogo

En la lección anterior viste cómo construir arreglos; en particular viste las declaraciones de todos los arreglos que vas a utilizar para el Catálogo de discos. Sin embargo, no es hasta la construcción del catálogo que puedes construir los arreglos, aunque ya los hayas declarado.

En esta lección programarás los constructores del catálogo, que son los que *definen* los arreglos relacionados con el mismo. Por la tarjeta de responsabilidades tienes tres constructores, cuyos encabezados se obtienen directamente de la tarjeta. Los colocas, junto con la documentación de Javadoc, a continuación de las declaraciones que ya hiciste para esta clase.

```
68  /**
69    * Construye un catalogo. Da el maximo numero posible de
70    * registros para discos y anota que no tiene ningun disco
71    * registrado. Se construiran los arreglos correspondientes
72    * a historico y fechas, pero unicamente con su numero
73    * de renglones (o tablas). Al registrar a cada disco mediante el menu, se
74    * construiran los arreglos correspondientes a sus fechas
75    * y su historico.
76    */
77   public Catalogo () {
93   } // Termina constructor sin parametros
94
95   /**
96    * Construye un catalogo con espacio para un numero
97    * definido de discos Los espacios para discos contienen
98    * una referencia nula. Se construiran los arreglos correspondientes
99    * a historico y fechas, pero unicamente con su numero
100   * de renglones (o tablas). Al registrar a cada disco, se
101   * completaran los arreglos correspondientes a sus fechas
102   * y su historico.
103   * @param numDscs Maximo numero de discos que va a tener.
104   */
105  public Catalogo( int numDscs ) {
113  } // Termina constructor con un parametro
114
115  /**
116   * Construye una Catalogo con espacio para un numero definido de discos.
117   * El catalogo lo inicializas con el contenido del arreglo que le pasas.
118   * Inicializa el contador relativo al numero de discos. Para aquellos discos
119   * que ya estan construidos, construye las columnas relacionadas en fechas
120   * e historico. Si el numero de lugares solicitados es menor que el de discos
121   * para inicializacion, se usa este ultimo para los tamanhos de arreglos.
122   * @param numDscs Maximo numero de discos que va a tener.
123   * @param nuevos  Arreglo de discos con los que inicia
124   *                el catalogo.
125   */
126  public Catalogo ( int numDscs, Disco[ ] nuevos ) {
155  } // Termina constructor con dos parametros
```

Nos falta dar el bloque correspondiente a la implementación de los constructores.

Todos los constructores tienen que realizar las siguientes tareas:

1. Construir el "armario" para el catálogo de discos (dando únicamente el tamaño), pero con todos los lugares vacíos (`null`).
2. Construir el total de casilleros, según el número de discos posibles, pero sin dar el tamaño para cada casillero (cuántas fechas de inicio va a tener cada disco, que depende del número de transmisiones permitidas del disco particular).
3. Construir el número de casilleros para los registros históricos. El tamaño de cada casillero (número total de registros en cada posición) está dado por el número de transmisiones permitidas en cada disco particular.
4. Iniciar los contadores de históricos, con un contador por disco.

Estas acciones aplican tanto para el primer como el segundo constructor. Para el tercer constructor tienes, además, que copiar los elementos del arreglo que llega como entrada al catálogo que se construya. Asimismo anotas, en este caso, el número de discos que hayas copiado, y para disco das el número de registros en cada uno de los casilleros.

Siguiendo este libreto codificas el primer constructor.

```
77  public Catalogo ( ) {
78    // Das el maximo de casilleros posibles al construir el catalogo
79    catalogo = new Disco[MAX_DISCOS];
80    numDiscos = 0; // No llevas ningun disco registrado
81    /* El numero de posiciones disponibles para registrar transmisiones e
82     * historica estara dado por el numero de transmisiones permitidas
83     * del disco en esa posicion, que no se sabra hasta que se construya
84     * el disco correspondiente. El numero de lugares ocupados esta dado
85     * por transmisiones activas del disco en esa posicion. */
86    fechas   = new GregorianCalendar[ catalogo.length ][ ];
87    historico = new GregorianCalendar[catalogo.length][2][ ];
88    // el numero de lugares ocupados en historico esta dado por
89    // numHist[disco] para cada disco. Para cada disco se requiere
90    // la fecha de inicio y fin de la transmision, por eso el 2.
91    numHist  = new int[catalogo.length];
92    // Como los elementos del arreglo son enteros, cada elemento se
93    // inicia en ceros
94  }
```

El segundo constructor recibe el número máximo de casilleros en el catálogo, por lo que hay poca diferencia con el primero, excepto por este número inicial. Lo único adicional que debes hacer es verificar que el número de discos esté en rango y a continuación utilizar ese número verificado en las distintas construcciones. Puedes copiar el bloque del primer constructor pero utilizar el número verificado en lugar del máximo.

```
95  public Catalogo ( int numDscs ) {
96    // Verificas que el numero de discos sea valido
97    int maxDscs = Disco.checaEntero(1, numDscs, MAX_DISCOS);
```

A partir de ahí, simplemente sustituyes `MAX_DISCOS` por el valor certificado `maxDscs`:

```
95    catalogo = new Disco[maxDscs];
96    numDiscos = 0; // No llevas ningun disco registrado
```

Como en el primer constructor estás usando el tamaño de `catalogo`, puedes copiar el código exactamente igual.

```
97      fechas    = new GregorianCalendar[catalogo.length][];
98      historico = new GregorianCalendar[catalogo.length][2][];
99      numHist   = new int[catalogo.length];
100   }
```

Para el tercer constructor se recibe el número total de casilleros para discos y una colección inicial con la cual llenar el catálogo, y que da el número inicial de registros activos. El tamaño final del catálogo es el máximo entre el número de registros en el arreglo que están pasando y el tamaño que te dicen debe tener. Hay que verificar, antes que nada, si la referencia `nuevos` es nula, para no tener un error en ejecución. Si es así, dices que el tamaño del arreglo que te pasan es cero.

```
126   public Catalogo ( int numDscs, Disco[ ] nuevos ) {
127     int numNvos = nuevos == null ? 0 : nuevos.length;
128     numDscs     = numDscs < nuevos.length ? numNvos : numDscs;
129     numDscs     = Disco.checaEntero(1, numDscs, MAX_DISCOS);
130     /* El numero de discos a copiar es el minimo entre el tamanho final
131      * del arreglo y el numero de discos nuevos */
132     numNvos     = numNvos > numDscs ? numDscs : numNvos;
```

Una vez definido el tamaño final del arreglo, procedes a construir lo necesario de la misma forma que en los otros dos constructores, excepto que no inicializas todavía el número de discos registrados en el catálogo.

```
133     catalogo  = new Disco[numDscs];
134     numDiscos = 0;
135     fechas    = new GregorianCalendar[catalogo.length][];
136     historico = new GregorianCalendar[catalogo.length][2][];
137     numHist   = new int[catalogo.length];
```

Repetición de enunciados en ejecución

Lo siguiente que tienes que hacer es "poblar" el catálogo con los discos que vienen en el arreglo `nuevos`. No sabes cuántos son, pero con cada uno, excepto por el lugar que ocupan, vas a hacer exactamente lo mismo, teniendo como variable el lugar que deben ocupar. Necesitas un enunciado que repita ciertas acciones. En general, para un enunciado de repetición necesitas lo siguiente:

1. La inicialización o primer valor o valores con los que el enunciado de la repetición (en adelante, *iteración*) va a trabajar.
2. Una condición que, mientras se evalúe a verdadero, el bloque de la iteración se repita.
3. El bloque a repetir.
4. La actualización de los valores usados en la repetición, de tal forma que se consiga que la condición mencionada en (2) deje de cumplirse.

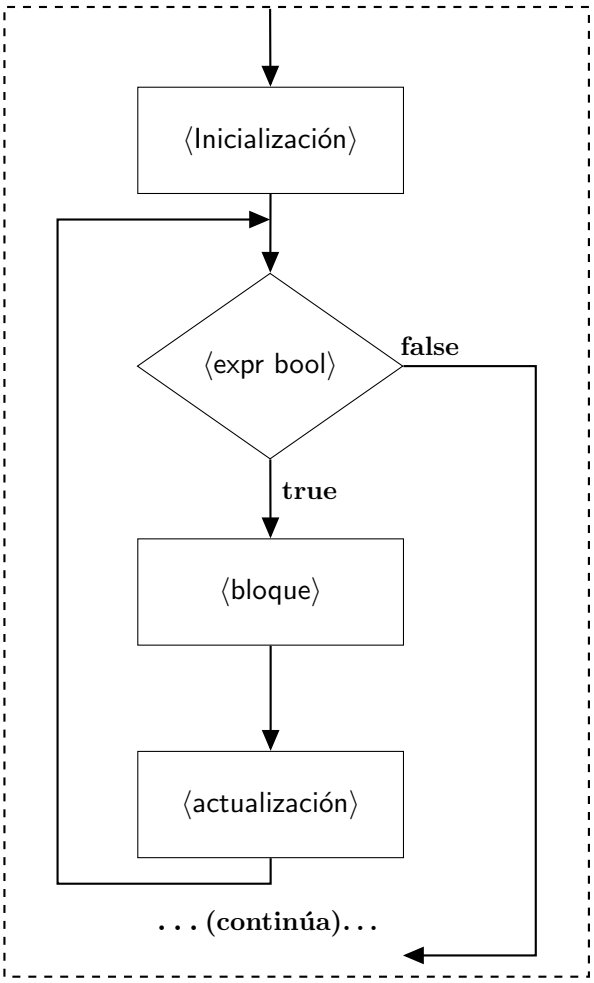
La inicialización, en general, no forma parte del bloque a repetir, ya que se ejecuta una única vez

El diagrama de flujo de este tipo de iteración es como se muestra en la siguiente página. Los rectángulos se refieren a código ejecutable en Java, mientras que el diamante se refiere a la evaluación de una expresión booleana, cuyo resultado solo puede ser `falso` o `verdadero`. Dependiendo

de esta evaluación, la ejecución continúa por la rama de `falso` o `verdadero`. Una vez terminado el bloque de la iteración, se lleva a cabo una actualización que permite que la ⟨`expresion booleana`⟩ cambie su evaluación de verdadera a falsa, para poder salir del ciclo.

Cuando esta actualización no se hace de manera adecuada o la expresión booleana no está bien elaborada, la ejecución del programa puede caer en lo que se conoce en la jerga de programación como un *loop* o ciclo infinito.

Figura 1 Flujo en la ejecución de una iteración en Java



El ⟨`bloque`⟩ generalmente consiste de una lista de enunciados entre llaves –después verás cuándo se pueden omitir las llaves en cierto tipo de iteración–.

Lo que quieres hacer en el tercer constructor es recorrer, de uno en uno (pero sin saber a ciencia cierta cuántos hay) los registros que vienen en el arreglo `nuevos` para copiarlos a `catalogo` y dar los tamaños que dejaste sin definir en los arreglos correspondientes.

El algoritmo para recorrer un arreglo es como sigue:

1. Colocarse al principio del arreglo o lista
2. Si se desea construir o calcular algo con los elementos del arreglo, iniciar "acumuladores", que pueden ser una cadena, un número u otro arreglo, dependiendo de lo que queramos hacer.

3. Mientras haya elemento en el arreglo, repetir 3.1. hasta 3.2.:
   3.1. Ejecutar las acciones requeridas con el elemento actual.
   3.2. Pasar al siguiente elemento del arreglo.
4. Entregar el resultado, si es que hay alguno.

En este algoritmo, la *inicialización* son los puntos 1 y 2. Pueden ser parte del enunciado de iteración o llevarse a cabo antes de empezar la iteración. En el recorrido de un arreglo quiere decir poner el o los índices en cero.

La *expresión booleana* se refiere a verificar que todavía estamos frente a un elemento del arreglo (inciso 3), o sea que el índice es menor que el tamaño del arreglo.

El *bloque* consiste en ejecutar lo que se desea con ese elemento del arreglo (inciso 3.1), ya sea copiarlo, agregarlo a una cadena, contarlo, entre otras acciones posibles.

La *actualización* corresponde al inciso 3.2. En un arreglo esto quiere decir incrementar el índice que se usa con el arreglo.

En general, para el "recorrido" de arreglos usas la iteración conocida como `for`, cuya sintaxis es la siguiente:

```
for ( ⟨inicializacion⟩ ; ⟨expr bool⟩ ; ⟨actualizacion⟩ )
      ⟨bloque⟩
```

En el caso de esta iteración, como todo menos el el ⟨`bloque`⟩ está entre los paréntesis, separados entre sí por punto y coma (`;`), el ⟨`bloque`⟩ puede consistir de un único enunciado (terminado con punto y coma) o una lista de enunciados entre llaves. Las iteraciones pueden estar anidadas, es decir, que haya un `for` en el bloque de otro `for`, en cuyo caso decimos que el de adentro se repite completo por cada vez que se ejecuta el de afuera.

Aunque el `for` no está limitado a recorrer arreglos (es una iteración muy poderosa) se usa típicamente cuando quieres que una variable tome valores consecutivos o cuando el siguiente valor es una función del valor anterior.