Métodos de consulta o acceso

Pasas ahora a los métodos getXXX, que devuelven el valor del atributo. Donde tenías **return** en cada uno de estos métodos, en lugar del valor que tenías colocas el nombre del atributo. Los desarrollas en orden alfabético de acuerdo al nombre del atributo que deseas consultar. En este listado omitimos renglones en blanco en Javadoc y entre métodos para ahorrar espacio.

```
/**
 * Proporciona el numero de transmisiones activas.
 * @return numero de transmisiones activas.
 */
public int getActivas()   {
  return activas;
}
/**
 * Proporciona el anho en que fue grabado el disco.
 * @return fecha de grabacion.
 */
public int getANHO()   {
  return ANHO;
}
/**
 * Proporciona el nombre que tenga asociado el disco. Puede
 * ser el musico si se trata de un CD o la pelicula si se
 * trata de un DVD o Bluray.
 * @return nombre del musico si se trata de un CD o de la
 * pelicula, si se trata de un DVD y de la serie si se trata de
 * un Bluray.
 */
public String getNOMBRE()   {
  return NOMBRE;
}
/**
 * Proporciona el numero de transmisiones permitidas.
 * @return numero de transmisiones permitidas.
 */
public int getPermitidas()   {
  return permitidas;
}
/**
 * Proporciona el tipo del disco.
 * @return el tipo de disco entre 1 y 3.
 */
public short getTIPO_DISCO()   {
  return TIPO_DISCO;
}
```

Métodos mutantes o de actualización

Toca ahora el turno, según los encabezados de la interfaz, a los métodos ***mutantes*** de la forma setXXX(par), que simplemente obligan (mutan) al atributo a tomar el valor del argumento que les pasen, verificando cuando sea necesario que el valor está dentro de rangos.

```
/**
 * Modifica el valor de transmisiones permitidas.
 * @param permisos valor que se desea establecer
 */
public void setPermitidas(int permisos) {
  permitidas = checaEntero(0, permisos, MAX_PERMITIDAS);
}
/**
 * Modifica el valor de transmisiones activas.
 * @param newActivas valor que se desea establecer
 */
public void setActivas(int newActivas)  {
  activas = checaEntero(0, newActivas, permitidas);
}
```

Como para los atributos constantes no se puede modificar su valor una vez construido el objeto, no tenemos más métodos mutantes en esta clase.

Métodos de implementación

Los métodos de ***implementación*** son aquellos que llevan a cabo diversas tareas como imprimir, mostrar, mover, y que si bien trabajan con los valores de los atributos no se refieren exclusivamente a uno en particular. También estos métodos pueden requerir de métodos auxiliares.

El primer método de implementación que vas codificar es copiaDisco. Este método simplemente construye un disco nuevo con los valores del disco que lo está llamando y el nuevo disco lo entrega como resultado.

```
/**
 * Duplica a este disco, construyendo otro objeto con los mismos
 * valores, pero con identidad distinta.
 * @return un nuevo disco identico al que se le pide, pero
 *         con cero en transmisiones activas.
 */
public ServiciosDisco copiaDisco( )   {
return new Disco (this.TIPO_DISCO, this.NOMBRE,
                  this.ANHO, this.permitidas);
}
```

Pasas ahora a implementar el método muestraDisco(String titulo) que debe mostrar al usuario el estado de un disco dado en forma agradable para el usuario. Este es, claramente, un método de implementación, pues combina varios de los atributos. Empiezas con una cadena vacía a la que le

vas pegando renglones que vayan mostrando cada uno de los valores precedidos por sus respectivas descripciones:

```
/**
 * Muestra de forma estetica el contenido de este disco.
 * @return una cadena con la informacion y que contiene saltos de
 *         linea.
 */
public String muestraDisco(String encabezado)  {
  String salida = ""; // Una cadena vacia
  salida += "Tipo de disco: "
    + (TIPO_DISCO == CD
        ? "CD"
        : (TIPO_DISCO == DVD
            ? "DVD"
            : "Bluray")
      ) + "\n";
  salida += "Nombre de"
    + (TIPO_DISCO == CD
        ? "l artista"
        : (TIPO_DISCO == DVD
            ? " la pelicula"
            : " la serie")
      ) + ": " + NOMBRE + "\n";
  salida += "Anho de grabacion: " + ANHO + "\n";
  salida += "Numero de transmisiones permitidas: "
    + permitidas +"\n";
  salida += "Numero de transmisiones activas: "
    + activas + "\n";
  return salida;
}
```

Podías haber hecho una sola asignación a salida , cuidando simplemente de no empezar una cadena en una línea de código y seguirla en otra, porque lo entrecomillado tienen que empezar y terminar en la misma línea de código (aunque se inserten "\n", que ordenan un cambio de línea, pero esto sucede en la ejecución) y hubiese quedado así:

```
String  salida = "Tipo de disco: "
  + (TIPO_DISCO == CD ? "CD"
      : (TIPO_DISCO == DVD ? "DVD"
          : "Bluray")) + "\n"
  + "Nombre de" + (TIPO_DISCO == CD ? "l artista"
      : (TIPO_DISCO == DVD ? " la pelicula"
          : " la serie")) + ": " + NOMBRE + "\n"
  + "Anho de grabacion: " + ANHO + "\n"
  + "Numero de transmisiones permitidas: "
  + permitidas +"\n"
  + "Numero de transmisiones activas: "
  + activas + "\n";
```

O podías no haber declarado la cadena salida y pegárselo todo al **return** (un poco menos claro):

```
    return "Tipo de disco: " + (TIPO_DISCO == CD ? "CD"
            : (TIPO_DISCO == DVD ? "DVD" : "Bluray")) + "\n"
        + "Nombre de" + (TIPO_DISCO == CD ? "l artista"
                        : (TIPO_DISCO == DVD ? " la pelicula"
                          : " la serie"))
        + ": " + NOMBRE + "\n" + "Anho de grabacion: " + ANHO
        + "\n" + "Numero de transmisiones permitidas: "
         + permitidas +"\n" + "Numero de transmisiones activas: "
         + activas + "\n";
```

El método terminaTransmision() verifica que, en efecto, haya alguna transmisión activa. Si la hay, decrementa el número de transmisiones activas en 1 y regresa verdadero. So no hay transmisiones activas regresa falso. Antes de regresar verdadero o falso debe, en el caso de que haya transmisiones por terminar, decrementar en 1 el número de transmisiones activas. Para ello, primero guarda en una variable booleana el valor que va a regresar, asignando el valor de verdadero si es que hay alguna transmisión activa. Dependiendo de este resultado, en una condicional aritmética, resta 1 o 0 para verdadero o falso respectivamente. Finalmente regresa el valor lógico calculado. Toma en cuenta que una vez que aparece un **return** en un método, nada de lo que se encuentre después va a ejecutarse, pero no puedes, sin saber si hay transmisiones activas o no, decrementar este valor.

```
/**
 * Decrementa el numero de transmisiones activas si es que
 * las hay.
 * @return si pudo o no terminar la transmision.
 *         Podria no haberla terminado si no hay transmision que
 *         terminar.
 */
public boolean terminaTransmision()    {
  boolean hayActivas = activas > 0;
  activas -= hayActivas ? 1 : 0;
  return hayActivas;
}
```

Para el método toString(), en donde quieres que cada atributo esté en una posición fija, la codificación se vuelve un poco más complicad, porque necesitas que todos los atributos ocupen cada uno un número de espacios determinados. Para ello tienes que conocer el número de posiciones que ocupa el máximo número de grabaciones permitidas; el número de transmisiones activas ocupará el mismo número de espacios. Por lo tanto, debes establecer un máximo para el número de transmisiones permitidas, que podemos pensar es, arbitrariamente, 9999, por lo que ocupa cuatro posiciones. Agregas estos límites a las constantes estáticas de esta clase (junto a las otras declaraciones de constantes de clase) y las usas para acomodar los valores. Agregas, a continuación de las constantes de clase que ya tienes, las siguientes líneas:

```
public static final int LUG_TD = 1,
  LUG_NOMBRE = 40,
  LUG_ANHO = 4,
  LUG_PERMITIDAS = 4,
  LUG_ACTIVAS = 4;
```

Ahora que ya sabemos el número de posiciones que ocupan, como máximo, las transmisiones permitidas y activas (lo colocamos también en una constante estática, LUG_ACTIVAS y lo mismo hacemos para cada atributo) podemos construir la cadena para el disco, rellenando con blancos a la izquierda en los números y a la derecha en las cadenas. Para ello utilizaremos una constante de la clase que consista del número de blancos que tenemos como máximo para el nombre (que es mayor que el de los números), declarando una constante de clase BLANCOS:

```
private final static String ESPACIOS = "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"
  + "␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣";
```

Usas el operador + para pegar cadenas, ya que una cadena entre comillas tiene que empezar y terminar en la misma línea de código. Cada una de las cadenas que estás concatenando tiene 20 espacios (o caracteres blancos).

Vas a manipular cadenas, por lo que es conveniente presentar algunos de los métodos más comunes que tiene la clase **String**. Se encuentran a continuación. Aquellos métodos que no tienen el modificador **static** tienen que ser invocados desde un objeto de la clase **String**.

En la parte correspondiente a este video puedes ver una tabla con todas las funciones de cadenas listadas y ejemplificadas.

Como el proceso de editar todos los números es el mismo, pegarle espacios por la izquierda y luego tomar los últimos lugares válidos, puedes hacer un método que edite a un número:

```
/* Acomoda a un numero para que tenga cierto numero de
 * espacios a la izquierda.
 * Recibe como parametros el valor y el numero
 * de lugares a ocupar.
 */
public static String editaNum(int valor, int lugares)  {
  String conEspacios = ESPACIOS + valor; // Agregamos espacios
                                         // por la izq.
  return conEspacios.substring(sNum.length() - lugares);
         // Tomamos lo que incluye al numero original
         // que se encuentra al final de la cadena.
}
```

El método substring de una cadena tiene dos sintaxis válidas:

```
substring ( int desde, int hasta)
```

Extrae, de la cadena con la que se la invoca, (hasta - desde) posiciones a partir de la posición desde y hasta, pero no incluyendo, la posición hasta.

La segunda forma es:

```
substring(int desde)
```

Que toma la subcadena que empieza en la posición desde y hasta la última posición de la cadena. Recuerda que las posiciones en una cadena empiezan en 0.

El método length() devuelve el número de símbolos en la cadena. La última posición de una cadena str es str.length() − 1, porque, como se indicó arriba, empiezan en 0.

Para el caso de completar cadenas con espacios, al parámetro le pegas los espacios al final y tomas desde el principio de la cadena así formada y extraes el número de posiciones que deseas. El método es sencillo:

```
public static String editaCad (String sValor, int lugares) {
  String conEspacios = sValor + ESPACIOS;
  return conEspacios.substring(0,lugares);
```

Puedes usar estos dos métodos para hacer más fácil la codificación del método toString:

```
/** Proporciona una cadena con los distintos campos ocupando un
  * lugar definido.
  *
  * @return la informacion del disco linealizada en forma de
  *         cadena, todos los discos con la misma informacion en
  *         las mismas posiciones.
  */
public String toString()   {
  String cadena = ""
    + TIPO_DISCO // ocupa un unico lugar
    + editaNum(ANHO,LUG_ANHO)
    + editaCad(nombre,LUG_NOMBRE)
    + editaNum(permitidas,LUG_PERMITIDAS)
    + editaNum(activas,LUG_ACTIVAS);
  return cadena;
```

Podías también no guardar nada en variables locales y simplemente colocar todo en el enunciado de return:

```
  return ""
    + TIPO_DISCO // ocupa un unico lugar
    + editaNum(ANHO,LUG_ANHO)
    + editaCad(NOMBRE,LUG_NOMBRE)
    + editaNum(permitidas,LUG_PERMITIDAS)
    + editaNum(activas,LUG_ACTIVAS);
```

Es importante poner la cadena vacía "" inmediatamente después de la asignación para que interprete todo lo que sigue como cadenas.

¡Espero verte en el siguiente video!