Implementación de la clase Disco

Codificación del constructor sin parámetros

Ahora pasas a codificar el constructor que interacciona con el usuario. Se menciona en la documentación del constructor sin parámetros, que debe a comunicarse con el usuario y pedirle por consola los datos para inicializar al objeto. Como viste en la lección anterior, Java te proporciona al objeto System.in, conocido como la *entrada estándar*. Este objeto requiere para funcionar que "vigiles" su interacción con el usuario usando excepciones, algo que no conoces todavía; adicionalmente, usa una entrada muy primitiva (carácter por carácter, de muy bajo nivel) por lo que no lo usarás.

Para poder leer números o cadenas directamente usarás otra clase que te proporciona Java y que se llama Scanner (pertenece al paquete util). Te va a servir como un filtro o intérprete de lo que proporcione System.in para utilizar acciones de lectura de más alto nivel que el que proporciona System.in. Puedes ver los atributos y métodos públicos que puedes usar en la documentación de Java para esta clase. Irás conociendo nada más los que vayas a utilizar.

Como Scanner no está en el paquete java.lang, tienes que "importarlo" (avisarle al compilador que vas a utilizar métodos de esa clase). Este aviso aparece inmediatamente después del enunciado de paquete (**package**) y antes que cualquier otro enunciado de Java; tiene el formato:

```
import java.util.Scanner;
```

También podrías pedir todas las clases e interfaces del paquete java.util de la siguiente manera:

```
import java.util.*;
```

Lo anterior no es conveniente, a menos que vayas a utilizar muchísimas de las clases e interfaces del paquete, pues estarías metiendo al código ejecutable de tu proyecto o paquete una cantidad exagerada de material que no vas a usar. Como por lo pronto únicamente vas a usar Scanner, usas la primera versión.

En la implementación del constructor (y de cualquier método) puedes tener lo que se conoce como *declaraciones locales*. Las declaraciones locales toman *casi* la misma forma que los atributos, excepto que no llevan acceso, pues las declaraciones locales sólo se conocen entre las llaves del bloque en el que están hechas; tampoco pueden tener el modificador **static**, pues las declaraciones locales no sobreviven a la **ejecución** del método y, por lo tanto, no pueden estar en la clase compilada. Los parámetros de un método juegan el papel de variables locales, excepto que el valor inicial se da en la llamada del método. El modificador **final** sí lo puedes usar en declaraciones locales y, si bien desaparece al terminar la ejecución del método, tienes que inicializarlo en la declaración y no te permite hacer una asignación posterior dentro del método. Puedes inicializar a una variable local en el momento de declararla usando la siguiente sintaxis:

$$\langle \text{declaración}_{tipo} \rangle = \langle \text{expresión}_{tipo} \rangle$$

La ⟨expresión⟩ puede ser, a su vez, una expresión que asigne una variable. La asignación se evalúa de derecha a izquierda, por lo que si hay más de una asignación, sólo a la última se le permite ser una operación o llamada a método. En los otros casos, deberá ser el nombre de una variable.

```
int valor1, valor2;
int ran = 1725;
// Asignacion multiple
valor1 = valor2 = ran = (int)Math.round(Math.random() * 111 * 113);
System.out.println("valor1:" + valor1 + "\tvalor2="
                    + valor2 + "\tran=" + ran);
```

El orden de evaluación en el cuarto renglón es el siguiente:

1. Se calcula la expresión del final de adentro hacia afuera:
   - Primero Math.random(), que está entre paréntesis y que regresa un **double** entre cero y uno.
   - Se llevan a cabo las dos multiplicaciones de este resultado, de izquierda a derecha, primero por 111 y después por 113.
   - Se redondea este valor (que tiene que ser positivo), pues está como argumento de Math.round(**double**).
   - Se le aplica un *casting* para convertir el valor a un entero.
2. Se asigna el resultado a la variable ran.
3. El valor de la variable ran se copia a la variable valor2.
4. El valor de la variable valor2 se copia a la variable valor1.

En resumen, el valor de la última expresión se copia a todas las variables que aparezcan en la asignación.

No puedes usar una asignación en medio de una expresión, como en el siguiente código, ya que el significado sería asignar un valor a otro valor. Tendrás que hacer una asignación por enunciado.

```
valor1 = ( valor2 = ran * 10) = ran++;
```

Usarás la declaración y definición simultáneas de un objeto de la clase Scanner con una inicialización. Para ello emplearás el constructor de Scanner con la siguiente firma:

```
Scanner(InputStream source)
   Construye un nuevo Scanner que produce valores tomados del flujo
   de entrada especificado.
```

Vale la pena comentar que también se puede aplicar un Scanner a una cadena, con el siguiente constructor:

```
Scanner(String source)
   Construye un nuevo Scanner que produce valores tomados de la
   cadena especificada.
```

Ya sea que el Scanner esté montado sobre un InputStream o sobre una cadena (**String**), los caracteres que aparecen en el flujo de entrada se van tomando de uno en uno, en el orden en que se presentan. Los métodos de Scanner juntan tantos caracteres como necesitan, dependiendo

de si se pide un entero, un real o una cadena, y ya juntos se interpretan y se convierten a la representación interna solicitada.

Declaras e inicializas el Scanner:

```
Scanner sc = new Scanner(System.in);
```

El operador new ya lo visitaste en la lección anterior.

Usando System.out, que ya conoces, empiezas el diálogo con el usuario para pedirle los valores que quieres. Conforme el usuario te responde a las solicitudes, vas guardando los valores verificados en cada uno de los atributos.

Cuando estás definiendo los métodos o atributos en una clase, éstos se van a copiar o crear para cada objeto que se construya. Dentro de los métodos distingues al objeto que los está invocando mediante la palabra reservada **this**, que se refiere al objeto en el que se encuentra en ese momento la ejecución, el destino del mensaje.

Por el momento los métodos interesantes de Scanner los puedes encontrar en la página de documentación de Java 8 y en esta página. Utilizarás únicamente el constructor Scanner(InputStream in), nextLine(), nextInt(). nextShort () y close(), que te iremos describiendo conforme los tengas que usar.

Empiezas el constructor de **Disco** con la construcción de un Scanner. Siempre que interactúas con un usuario tienes que irle diciendo, paso a paso, lo que esperas de él. Hay pocas cosas más desesperantes para un usuario que enfrentarlo a una pantalla en blanco, porque el usuario va a empezar a oprimir teclas arbitrarias y hará que tu proyecto muy probablemente aborte.

Por la razón anterior, una vez construido el Scanner, avisas al usuario de qué se trata y lo invitas a que te dé los datos, usando System.out para salida y sc, el Scanner que acabas de construir, para la entrada.

La interacción con el usuario es complicada porque tienes que verificar siempre que te esté dando los datos correctos. Por ello, una vez que recibas cada dato por parte del usuario, usarás los métodos privados encargados de verificar que los valores sean correctos.

```
/**
 * Constructor sin parametros; interacciona con el usuario
 * para pedirle los datos de inicializacion del disco.
 */
public Disco() {
  Scanner sc = new Scanner(System.in);
  System.out.println("Bienvenido a la fabrica de Discos");
  System.out.println("Que deseas grabar: "
                     + "(1) CD, (2) DVD, (3) Bluray");
  System.out.print("Elige tipo de disco (1,2,3) "
                   + "(termina con [Enter]):--> ");
```

Nota que el último uso de System.out es con el método print y no println, para que no emita el cambio de línea y lea inmediatamente a continuación de la flecha.

Quieres leer un valor de tipo **int** con sc. Llamas al método nextInt() desde sc y este método te regresa el valor de tipo **int** que tecleó el usuario o aborta el proyecto si el usuario teclea algo distinto a un entero. Si la ejecución no aborta, garantizas que es un valor válido paraTIPO_DISCO de la misma manera que en los otros dos constructores.

```
TIPO_DISCO = (short)checaEntero(CD, sc.nextShort(), BR);
sc.nextLine();
```

El programa no detecta que tiene algo para leer hasta que el usuario no oprima la tecla [Enter], por lo que la segunda línea es para comérselo y que no lo use, equivocadamente, en la siguiente lectura. El método nextLine() de un Scanner lee desde donde acaba lo que le pediste leer hasta e inclusive el siguiente [Enter], devolviendo la cadena leída sin incluir al [Enter]. Los otros métodos no usan el [Enter]; si lo siguiente a leer fuera una cadena, usaría al [Enter] como dato de entrada. Para evitarlo, cada vez que leas algo que no sea una línea agregarás un nextLine() para que el [Enter] no se use como dato. Como el resultado de este tipo de lecturas no se asigna a ninguna variable, el resultado de la lectura simplemente "se tira".

El método que verifica el nombre sólo se asegura que el usuario no le dé una referencia nula o una cadena vacía –representada por ""–. No es lo mismo una referencia nula que una cadena vacía, pues la primera no existe, por lo que no puede invocar a ningún método de la clase **String** pero la segunda sí existe pero tiene tamaño 0.

Nuevamente usas la condicional aritmética. Para obtener el valor de NOMBRE te comunicas con el usuario y se lo pides.

```
System.out.print("Ahora␣dame␣"
                 + ( TIPO_DISCO == CD
                     ? "el␣nombre␣del␣cantante"
                     : (TIPO_DISCO == DVD
                         ? "el␣nombre␣de␣la␣pelicula"
                         : "el␣nombre␣de␣la␣serie"
                       )
                   )
                 + "␣(␣terminando␣con␣[Enter])-->");
```

El mensaje que le escribes al usuario depende del tipo de disco que tengas. La condicional aritmética te regresa una de tres cadenas. Está encerrada entre paréntesis porque estás usando la concatenación de cadenas dentro de la print; si se encuentra el operador de igualdad no lo reconoce como operador de una condicional aritmética sino como parte de una cadena, ya que == tiene menor precedencia que +; el compilador trataría de pegar primero a la cadena "Ahora␣dame" con TIPO_DISCO; una vez hecho esto (que sí lo puede hacer, como ya mencionamos arriba) trata de aplicar el operador == a una cadena y un entero pequeño, la constante simbólica CD, por lo que reporta tipos incomparables y da un error de sintaxis.

```
NOMBRE = checaCadena(sc.nextLine());
```

Al método checaCadena le pasas como argumento el resultado de la lectura. A continuación pides el año de la grabación:

```
System.out.print("Dame␣ahora␣el␣numero␣de␣"
                    + "anho␣en␣que␣se␣grabo␣entre␣"
                    +  PRIMER_ANHO + "␣y␣" + ULT_ANHO
                    + "␣(terminando␣con␣[Enter])␣-->␣");
ANHO = checaEntero(PRIMER_ANHO, sc.nextInt(), ULT_ANHO);
```

Por último pides el número de transmisiones simultáneas permitidas, que debe ser un entero entre 0 y MAX_PERMITIDAS:

```
System.out.print("Dame␣ahora␣el␣numero␣de␣"
                    + "transmisiones␣permitidas␣entre␣0␣y"
                    +  MAX_PERMITIDAS
                    + "␣(terminando␣con␣[Enter])␣-->␣");
permitidas = checaEntero(0,sc.nextInt(),MAX_PERMITIDAS);
sc.nextLine();
```

Das las gracias y te despides, dejando a System.in como estaba, esto es, desconectado del Scanner, por lo cual cierras el Scanner utilizando el método close() de la clase Scanner. Con esto terminas de implementar el constructor sin parámetros.

```
System.out.println("Gracias");
sc.close();
```

¡Espero verte en la siguiente lección!