Expresiones

Antes de entrar de lleno a la codificación de los constructores, es conveniente que des una mirada al concepto de expresiones en general y cómo usarlas para la codificación de métodos.

Una expresión es aquello que como resultado arroja (calcula y entrega) un valor. Estás acostumbrados a las expresiones aritméticas, pero también, en Java, puedes tener expresiones que te regresen un valor booleano o una referencia (como el uso del operador new que viste en la lección pasada).

Como tienes que escribir las expresiones en forma lineal:

$$a/b \qquad \text{en lugar de} \qquad \frac{a}{b},$$

para saber el orden de evaluación Java asigna a cada operador una *precedencia* (el orden en que se tiene que aplicar el operador dentro de una misma expresión) y una *asociatividad*, que indica en que orden evaluar una expresión que repite operadores consecutivamente (si la aplicación del operador debe ser de izquierda a derecha o de derecha a izquierda). También cuentas con operadores unarios como **new**, binarios como la suma, resta y otros operadores aritméticos. Asimismo puedes usar la clase Math del paquete java.lang y que tiene muchos métodos estáticos que pueden ser invocados como Math.$\langle metodo \rangle$. En la tabla 1, que se encuentra en la siguiente página, puedes ver una lista de los operadores de Java, clasificados por su precedencia, de mayor –los que van antes– a menor. No usarás por lo pronto la mayoría de ellos, por lo que los iremos introduciendo conforme los requieras.

La asociatividad de todos los operadores es izquierda, excepto para los de asignación que es derecha. Para estos últimos hay que cuidar, como son los que menor precedencia tienen, que a su izquierda tengan una variable y no un valor, pues en este último caso habrá un error de sintaxis. Si quieres alterar la precedencia de los operadores –por ejemplo, que en $a + b * c$ se evalúe antes $a + b$, tendrás que usar paréntesis: $(a + b) * c$–.Los paréntesis sirven para alterar la precedencia y asociatividad *natural* de un operador.

El operador de condicional aritmética con precedencia 14 lo vas a usar mucho. Además de ser muy útil es muy elegante. La sintaxis se encuentra a continuación:

$\langle\, expresion\ booleana \,\rangle$
? $\langle\, valor_T \,\rangle$
: $\langle\, valor_F \,\rangle$

donde una $\langle\, expresion\ booleana \,\rangle$ es una variable declarada **boolean**, una expresión con operador relacional o lógico, o bien la invocación de un método que entregue un valor booleano. La necesidad o no de paréntesis estará dada por las precedencias, aunque muchas veces los usarás para que el editor acomode más claramente a los distintos componentes de este operador.

Podrías pensar en este operador como una expresión, pero viene en paquete: se tiene que usar la sintaxis dada y por eso le llamamos operador y no expresión. Entrega uno de los dos valores, $valor_T$ o $valor_F$ dependiendo si $\langle\, expresion\ booleana \,\rangle$ se evalúa a verdadero o falso, respectivamente. En esta tabla se dice que un operador es *relacional* si entrega un valor de verdadero o falso.

Tabla 1 Operadores de Java que utilizarás

Operandos	Símbolo	Descripción	Prec
posfijo unario	·	selector de clase	1
prefijo n-ario	(⟨*parámetros*⟩)	lista parámetros	
posfijo unario	⟨*var*⟩++ ⟨*var*⟩−−	auto post-incremento/decremento	
unario prefijo	++⟨*var*⟩ −−⟨*var*⟩	auto pre-incremento/decremento	2
unario prefijo	+⟨*expr*⟩ −⟨*expr*⟩	signo positivo/negativo	
unario prefijo	! ⟨expresión⟩	negación booleana	
unario prefijo	**new** ⟨*constructor*⟩	instanciador	3
unario prefijo	(⟨*tipo*⟩) ⟨*expr*⟩	*casting*	
binario infijo	* / %	multiplicación, división, módulo	4
binario infijo	+ −	suma, resta	5
binario infijo	< <=	relacional menor, menor o igual,	7
	> >=	mayor, mayor o igual	
binario infijo	**instanceof**	relacional "ejemplar de"	
binario infijo	==	relacional, igual a	8
binario infijo	!=	relacional, distinto de	
infijo	&&	AND lógico	12
binario infijo	||	OR lógico	13
ternario infijo	⟨*expr bool*⟩ ? ⟨$expr_T$⟩ : ⟨$expr_F$⟩	condicional aritmética	14
binario infijo	=	asignación	15
binario infijo	+=	auto suma y asignación	
binario infijo	−=	auto resta y asignación	
binario infijo	*=	auto producto y asignación	
binario infijo	/=	auto división y asignación	
binario infijo	% =	auto módulo y asignación	

En la condicional aritmética tanto $expr_T$ como $expr_F$ tienen que ser expresiones que entreguen un valor, y este valor podría ser de distinto tipo como en:

```
int r;
System.out.println(((r = (int)(Math.random() * 111 * 133 % 100))
                        > 49
                    ? "que mala onda"  // valor si true
                    : r));             // valor si false

System.out.println( "r=" + r);
```

El método Math.random() es un método estático de la clase Math –que viene con las demás clases en Java sin necesidad de importarla– que entrega un valor entre 0 y 1 de tipo **double** y supuestamente aleatorio (*pseudoaleatorio*). En varias corridas de este código los resultados son

como se muestra a continuación. Sin embargo, tú puedes codificar esto en una clase cualquiera y los resultados pueden ser totalmente distintos.

```
6
r=6

15
r=15

que mala onda
r=62
```

El código anterior lo toma como bueno Java porque System.out. println va a convertir a cadena el resultado de esta expresión. Cada uno de $expr_T$ o $expr_F$ pueden ser expresiones tan complicadas como quieras. En el ejemplo anterior haces una asignación al mismo tiempo que calculas un valor booleano. En general, tanto $expr_T$ como $expr_F$ tienen que ser del mismo tipo si quieres asignar el resultado a una variable, que tiene que ser del mismo tipo que $expr_T$ y $expr_F$.

Otro ejemplo, que obtiene el valor absoluto de un valor de tipo **double** "a pie" (tenemos la función Math.abs(**double**) para ello) se encuentra a continuación. Los paréntesis alrededor de la condicional aritmética no son necesarios pero hacen más claro el acomodo en el texto.

```
        double val = -256.078;
        double absVal =
          (val >= 0      // expresion booleana
             ? val       // valor si true
             : -val);    // valor si false
        System.out.println("Valor original:" + val
                           + "\tValor absoluto: " + absVal);
```

Si codificas esto en tu clase declarada para practicar verás que este segmento de código escribe lo siguiente en la consola:

```
Valor original:-256.078 Valor absoluto: 256.078
```

Entre las expresiones que puedes tener en $expr_T$ o $expr_F$ se encuentra, a su vez, otra expresión con un operador aritmético condicional si así se desea, cuidando únicamente la precedencia, usando para ello paréntesis cuando sea necesario, aunque las expresiones asocian de adentro hacia afuera como los paréntesis.

Implementación de los constructores

La implementación de un método consiste en enunciados que llevan a cabo las tareas necesarias para que el método haga lo que tenga que hacer. En este caso primero implementarás los constructores con parámetros. Dejarás para el final el constructor sin parámetros ya que, como se comunica con el usuario, es el que involucra más trabajo y cuidado.

Los encabezados del segundo y tercer constructores simplemente copian al atributo los valores dados, pero deben cuidar que los argumentos (los valores pasados en los parámetros) estén dentro de los límites especificados, que son:

valor mínimo	$\leq$	**argumento**	$\leq$	**valor máximo**
CD=1	$\leq$	tipo	$\leq$	3=BR
PRIMER_ANHO=1900	$\leq$	fecha	$\leq$	2017=ULT_ANHO
0	$\leq$	permitidas	$\leq$	50=MAX_PERMITIDAS
0	$\leq$	activas	$\leq$	permitidas

Por otro lado, deseas que el atributo NOMBRE no sea una referencia nula ni una cadena vacía.

Estas especificaciones las puedes partir en dos. Las que especifican un valor numérico (en este caso **int** o **short**) y las que vigilan a las cadenas.

En lugar que cada constructor (y los métodos set) estén verificando cada uno que los argumentos que les pasan están en rango, mejor harás dos funciones propias (métodos), uno para enteros y otro para cadenas, que regresen el valor dado, si el argumento está en rango, el valor mínimo si le dan un valor por abajo de éste o el valor máximo si le pasan un valor por encima de éste. En cuanto a las cadenas deberán regresar alguna cadena alusiva si el argumento es nulo o las cadenas está vacía. Estos dos métodos son *auxiliares*, ya que el usuario no los requiere, y serán declarados **private**. También, como reciben todos sus argumentos explícitamente y no usan ningún atributo de objeto, pueden ser estáticos y que residan en la clase compilada.

Métodos auxiliares

Clasificamos como métodos auxiliares a aquellos en los que delegan los métodos definidos en la interfaz para hacer más sencillo o rápido su trabajo. También se usan para aislar algunos pedazos de código y hacer más clara la lectura del mismo. Asimismo, se trata de códigos que se ejecutan frecuentemente y el aislar ese código en métodos auxiliares facilita el proceso de codificación o implementación. En este caso sirven para verificar rangos o valores permitidos, lo que tienes que hacer para garantizar que los valores almacenados en los objetos son siempre válidos. Los métodos auxiliares, en general, se declaran privados, por lo que la documentación de Javadoc no se usa. Harás una documentación más breve y sencilla.

Método que verifica que un entero esté en rango:

Este método auxiliar debe tener tres parámetros: el mínimo valor permitido, el valor que se desea verificar y el máximo valor permitido. Su encabezado se encuentra a continuación:

```
private static int checaEntero (int limInf, int arg, int limSup) {

}
```

- Se llama checaEntero.
- Recibe tres parámetros, todos de tipo entero. Usas el nombre del parámetro para identificar cómo se va a usar:
  - limInf que informa el valor mínimo deseado.
  - arg que indica quién es el entero a verificar.
  - LimSup que indica el máximo valor que puede tener.

Lo vas a implementar con la condicional aritmética. Tienes el seudo código dado por la especificación:

1. **Si** el argumento es menor que el mínimo, regresa el mínimo;
2. **Si no** es menor que el mínimo:
   2.1. Si el argumento es mayor que el máximo, regresa el máximo
   2.2. Si no: regresa el argumento.

El inciso 1 es el que va a continuación del primer "?". El inciso 2 va a continuación del primer ":" y es, a su vez, una condicional aritmética "anidada" (dentro de la primera). El inciso 2.1 se ejecuta si no se cumplió la primera condición. El inciso 2.2 se ejecuta si no se cumplieron ni la primera ni la segunda condición, lo que significa que el argumento está en rangos.

```
/* Verifica que el argumento esta en los limites
 * establecidos.
 * Recibe como parametros el limite inferior permitido,
 * el valor dado y el limite superior permitido y
 * entrega un numero valido.
 */
private static int checaEntero (int limInf, int arg, int limSup) {
  return (arg < limInf        // expresion booleana 1
            ? limInf          // valor si True
            : (arg > limSup   // si False, expresion booleana 2
                ? limSup      // valor si eb1=false y eb2=true
                : arg         // valor si eb1=false y eb2=false
              )
          );
}
```

Ninguno de los paréntesis es necesario, pero se añadieron para hacer explícito el anidamiento de las condicionales aritméticas.

Método que verifica que una cadena no sea nula o vacía

De manera similar puedes codificar el método que verifica si una cadena está en rango y no es nula. Recibe únicamente el argumento, pues sólo va a verificar que no sea una referencia nula. El encabezado de este método se encuentra a continuación:

```
  private static String checaCadena( String cadena )  {

  }
```

Queda codificada de la siguiente manera:

- Se llama checaCadena
- Recibe como entrada una cadena.
- Entrega como resultado una cadena.
- Es privada y estática por las mismas razones que checaEntero y pertenece a la clase.

Implementas el método, como ya se mencionó, con una condicional aritmética, de la siguiente forma:

1. **Si** la cadena es nula o tiene tamaño 0, regresa "No definido".
2. **si no** se cumple ninguna de estas condiciones, regresa la cadena que te dieron.

El método checaCadena queda como sigue:

```
/* Verifica que el nombre del artista o pelicula no sea una cadena
 * vacia o una referencia nula.
 * Recibe como parametro la cadena a verificar y regresa una
 * cadena correcta.
 */
private static String checaCadena(String cadena) {
  return (cadena == null || cadena.length() == 0)
    ? "No definido"
    : cadena;
}
```

En el caso de las operaciones lógicas, los operadores no son simétricos. Java usa para evaluarlos lo que se conoce como *corto circuito*: Evalúa tantos operandos como necesite para obtener el resultado. Por ejemplo, si tienes un OR, como es este caso, si el primer operando es verdadero ya no necesita evaluar el segundo, porque el resultado de la expresión es verdadero y ya no trata de operar con una referencia nula. Pero si el primer operando es falso entonces se ve obligado a evaluar el segundo operando. En el caso de un AND si el primer operando es falso ya no tiene por qué evaluar el segundo, porque el resultado va a ser falso –ver las tablas de verdad de los operadores lógicos–. Si la pregunta se hiciera en otro orden simétrica,

```
    (cadena.length() == 0   || cadena == null)
```

corres el riesgo que cadena sea **null** y el programa aborte porque estás pidiendo el tamaño de una cadena que no existe.

Agregas estos métodos al final del archivo que corresponde a la clase Disco y pueden ser invocados desde cualquier otro método, incluso desde métodos estáticos, declarados en la clase.

Las tablas de verdad (con los resultados) de los operadores lógicos pueden ser consultadas en cualquier texto de álgebra de bachillerato o en Wikipedia, por lo que no lo presentamos en esta sección para ahorrar un poco de espacio.

Codificación de los constructores

Puedes codificar ya los dos constructores que reciben parámetros. Ambos, después de verificar que sus argumentos están en rangos, los asignan a los atributos correspondientes.

```
/**
 * Constructor a partir del tipo de disco, nombre y fecha.
 *
 * @param tipo si es CD, DVD o BR.
 * @param nombre del artista o pelicula.
 * @param fecha de grabacion.
 */
public Disco ( short tipo, String nombre, int fecha)  {
  TIPO_DISCO = checaEntero(CD, tipo, BR);
  NOMBRE = checaCadena( nombre );
  ANHO = checaEntero (PRIMER_ANHO, fecha, ULT_ANHO);
}
```

Tratas de compilar y da un error de sintaxis en la línea en la que se asigna valor a TIPO_DISCO. Te dice que estás tratando de poner un **int**, que es lo que regresa checaEntero, en un campo declarado como **short**, que es más pequeño que un **int**. Sabes que el valor que regresa checaEntero es pequeño y cabe en un **short**, por lo que lo obligas a que "se presente" como un **short** haciendo un *casting* –(**short**)– que es también un operador prefijo– al valor que regresa checaEntero:

```
  TIPO_DISCO = (short)checaEntero(CD, tipo, BR);
```

Compilas y ahora sí está todo bien. Procedes a codificar el otro constructor con parámetros, copiando la asignación de los los primeros tres campos y agregando la asignación al atributo permitidas.

```
/**
 * Constructor a partir del tipo de disco, nombre, fecha
 * y numero de transmisiones permitidas.
 *
 * @param tipo si es CD, DVD o BR.
 * @param nombre del artista o pelicula.
 * @param fecha de grabacion.
 * @param permitidas el numero de transmisiones.
 */
public Disco ( short tipo, String nombre, int fecha,
               int permitidas)                          {
  TIPO_DISCO = (short)checaEntero(CD, tipo, BR);
  NOMBRE = checaCadena( nombre );
  ANHO = checaEntero (PRIMER_ANHO, fecha, ULT_ANHO);
  this.permitidas = checaEntero(0, permitidas, MAX_PERMITIDAS);
}
```

En la última línea de la implementación de este constructor, que tiene cuatro parámetros (entradas), para guardar el valor en el atributo permitidas, como el parámetro se llama igual que el atributo, este nombre identifica al más cercano de esa línea, que es el parámetro. Tienes que aclarar que quieres guardar en el atributo, por lo que haces uso de "**this**.", que indica,

precisamente, que se trata del objeto que se está construyendo. No lo tuviste que hacer antes, porque a los parámetros les diste nombres distintos que los de los atributos. Verificas, asimismo. que el valor esté en rangos.

Nos vemos en el siguiente video donde vas a aprender cómo leer información del usuario desde la consola.