Construcción de objetos de la clase **Disco**

Recuerda los encabezados de los constructores de la clase Disco:

Disco()

Constructor que le pide al usuario que dé los datos de las constantes que debe inicializar en el constructor

Disco(short tipo, java.lang.String nombre, int fecha)

Constructor a partir del tipo de disco, nombre y fecha de grabación.

Disco(short tipo, java.lang.String nombre, int fecha, int permitidas)

Constructor a partir del tipo de disco, nombre, fecha de grabación y numero de transmisiones permitidas.

Si quisieras usar el tercer constructor cuya firma es Disco(**short**, **String**, **int**, **int**), la invocación sería

```
Disco elTuyo = new Disco(2, "Descifrando Enigma", 2014, 50);
```

lo que querría decir que el tipo de disco es DVD, el nombre es "Descifrando Enigma", el año de grabación 2014 y el número de transmisiones simultaneas permitidas es 30.

Si tratas de compilar la clase Usuario con esta sintaxis, el compilador te va a dar un error de *tipos incompatibles*, debido a que las constantes de números enteros son, por omisión, de tipo int. Necesitas indicarle al compilador de Java que el 2 se refiere a un entero de tipo short. Para eso tienes lo que se conoce como *casting*; esta palabra tiene dos significados en inglés. El primero de ellos se refiere al elenco cuando se presenta una obra de teatro o una película, que de cierta manera es lo que estás haciendo cuando invocas un método, en este caso un constructor, con los argumentos que pide la firma del método. Si el personaje es un tipo gordo y alto no puedes poner en ese papel a una persona flaca y chaparra porque el vestuario y el papel no le quedan. El otro significado de *casting* es el de enyesar, que consiste en obligar a un hueso que se mantenga en cierta posición, que tome una forma. Cuando el compilador te dice que hay tipos incompatibles, te está diciendo que el tipo flaco y chaparro no puede hacer el papel del gordo alto; para corregir esta situación "enyesas" al argumento (aplicas un *casting*) para que pase con la forma que espera el método.

El *casting* consiste del nombre de un tipo (primitivo o de referencia) encerrado entre paréntesis y se aplica al siguiente operando. Tiene una precedencia muy alta (se dará después la tabla de precedencia y asociatividad de los operadores de Java).

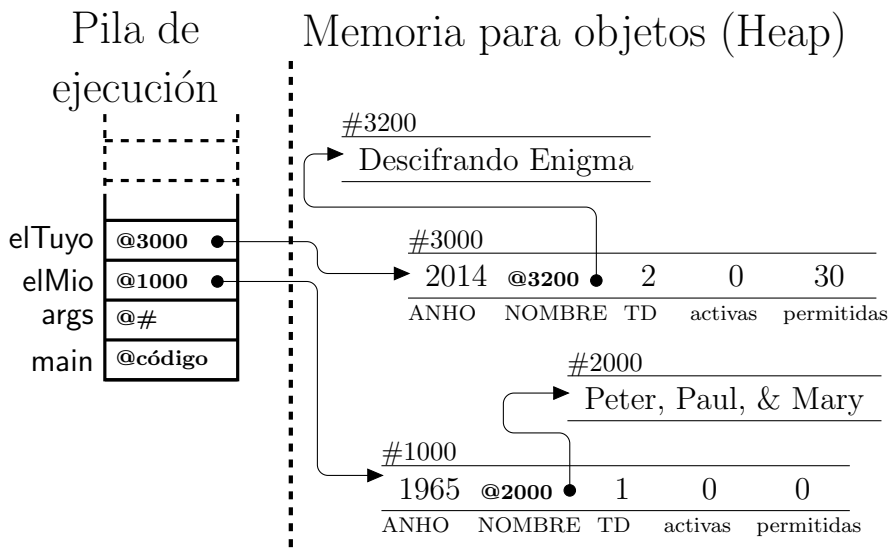
En tu caso, como quieres que el valor 2 sea interpretado como un short, la invocación al constructor tiene que hacerse de la siguiente manera:

```
Disco elTuyo = new Disco((short)2, "Descifrando Enigma", 2014, 30);
```

y la compilación ya no da ningún error. Cuando tienes tipos como los números enteros, donde los tipos están ordenados por tamaño y cada uno de ellos "cabe" en los de tamaño mayor, no es necesario hacer *casting* de un tamaño mayor declarado a uno menor usado, pero sí al revés. Lo mismo sucede con clases y subclases, donde las subclases extienden a las clases.

En ejecución, con la declaración hecha anteriormente para elMio, la memoria queda como se muestra en la figura 1.

Figura 1 Alojamiento en memoria durante ejecución



Uso de elementos de acceso público (o de paquete), estáticos o de objeto

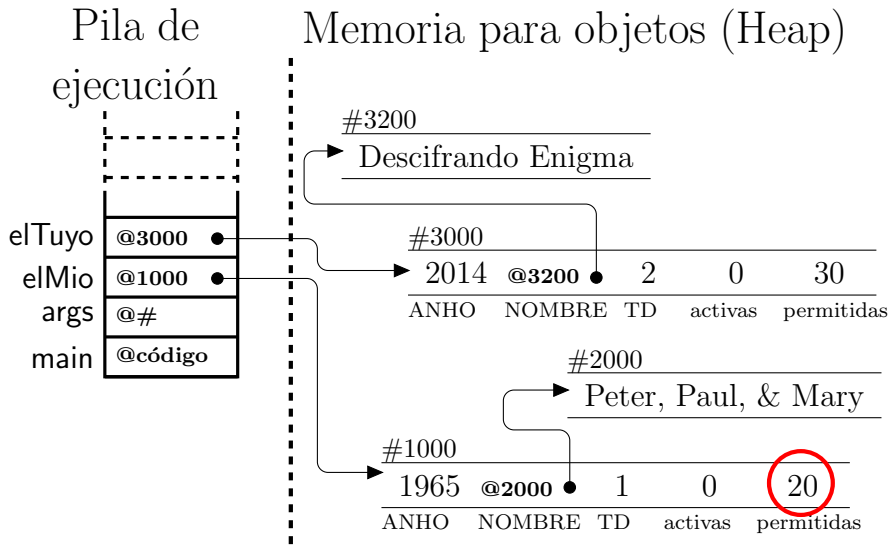
Ya tienes dos objetos de la clase (de tipo) Disco construidos. Ahora quieres usarlos.

Entre los operadores de Java tenemos el llamado *de selección* que se representa con un punto (.). Este es un operador binario que tiene a la izquierda el identificador de un objeto (o clase, para el caso de elementos estáticos) y a la derecha el identificador de un atributo o el identificador de un método seguido de sus argumentos. Por ejemplo, si quieres asignar un valor distinto de cero al número de transmisiones permitidas en el objeto elMio desde el método main, usas

```
elMio.setPermitidas(20);
```

lo que invocaría al método setPermitidas en el objeto elMio con el argumento 20. En ejecución la memoria quedaría como se muestra en la figura 2.

Figura 2 Alojamiento en memoria durante ejecución



Si quieres invocar al método muestraDisco(**String** encabezado) una vez que construiste un objeto de la clase Disco, como elMio o elTuyo, simplemente invocas a este método con alguno de los dos objetos:

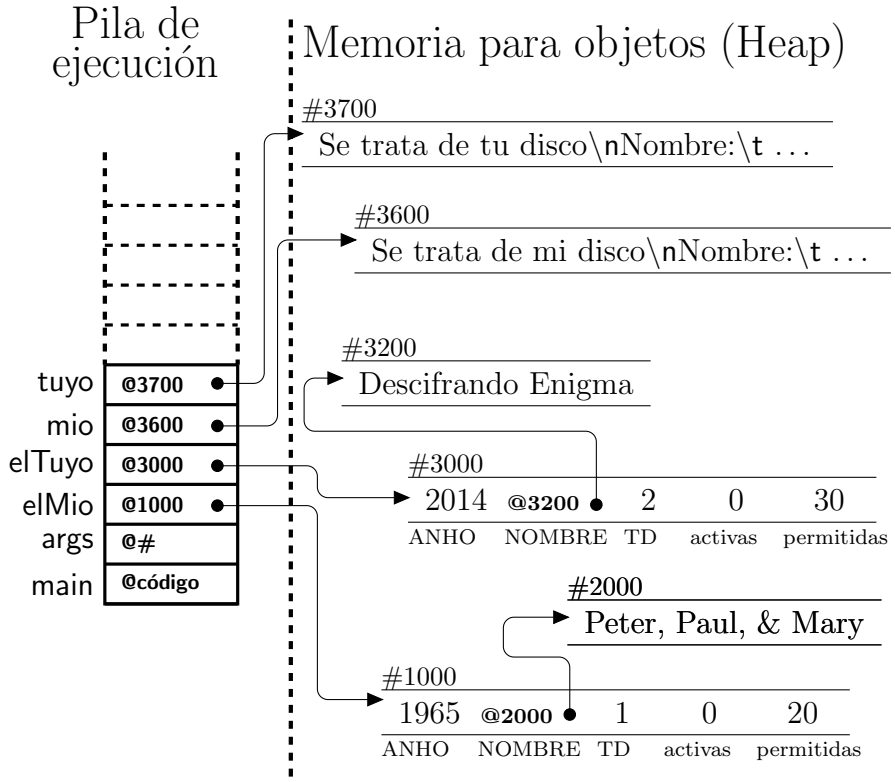
```
elMio.muestraDisco ( "Se trata de mi disco");
elTuyo.muestraDisco ( "Se trata de tu disco");
```

El método muestraDisco regresa una cadena (String). Si lo invocas nada más así, lo haces trabajar pero no haces nada con el resultado que te entrega, es como si lo tiraras. Puedes "guardarlo" en un objeto de la clase String, haciendo la declaración y definición simultáneamente:

```
String mio  = elMio.muestraDisco ("Se trata de mi disco");
String tuyo = elTuyo.muestraDisco ("Se trata de tu disco");
```

con lo que, en ejecución, la memoria se va a ver como en la figura 3.

Figura 3 Alojamiento en memoria durante ejecución



Hemos hablado de invocar o llamar a un método con tal o cual *argumento*, aparentemente refiriéndonos a lo mismo que antes llamábamos *parámetros*. Los *parámetros* son los *identificadores* que aparecen en la declaración de un método, mientras que los *argumentos* son valores que van a ocupar el lugar de los parámetros. Nos interesa la firma del método para saber el orden y tipo de los argumentos. Esto se hace en la declaración. Cuando quieres usar un método tienes que pasarle valores o expresiones, en orden, que coincidan con el tipo del parámetro declarado. Algunos autores llaman a los parámetros con el nombre de *parámetros formales* (*formal parameters*) y a los argumentos les llaman *parámetros reales* (*actual parameters*), pero es más fácil hablar de parámetros y argumentos, por lo que eso es lo que haremos.

Tienes ya todo lo que necesitas para probar la clase que te programaron. Puedes invocar (llamar, usar) clases y objetos públicos o de paquete que se encuentren en el paquete discos o públicos en el paquete java.lang. Usarás los distintos métodos e incluirás inmediatamente después de los métodos que construyan o cambien al objeto, un enunciado de impresión para ir vigilando la ejecución.

Listado 1 Clase Usuario (método main)

```
1  package discos;
2  // Omitimos los comentarios de Javadoc para ahorrar espacio
3  public class Usuario
4
5    public static void main(String[] args)  {
6      // No comentamos este metodo ya que antes de cada
7      // invocacion hay un mensaje al usuario de que es
8      // exactamente lo que se esta haciendo.
9      String saludo = "Bienvenido␣al␣sistema␣de␣streaming␣";
10     System.out.println(saludo);
11     Disco elMio = new Disco( );
12     elMio.setPermitidas(15);
13     Disco elTuyo = new Disco((short)2,"Descifrando␣Enigma",2014,30);
14     String mio = elMio.muestraDisco("Se␣trata␣del␣objeto␣elMio:\n");
15     System.out.println(mio);
16     System.out.println("elMio␣usando␣toString():");
17     System.out.println("*******\n" + elMio +"\n********");
18     elTuyo.setActivas(elTuyo.getActivas() + 5);
19     String tuyo = "El␣contenido␣de␣'elTuyo'␣con␣toString():\n"
20       + elTuyo;
21     System.out.println(tuyo);
22
23     System.out.println();
24     elTuyo.setPermitidas(50);
25     elTuyo.setActivas(49);
26     System.out.println("objeto␣elTuyo:\n"
27                        + elTuyo.daTransmision());
28     System.out.println();
29     System.out.println("objeto␣elTuyo:\n" + elTuyo.daTransmision());
30     System.out.println();
31     System.out.println("objeto␣elTuyo:\n"
32                        + (elTuyo.terminaTransmision()
33                            ?"Termino␣en␣'elTuyo'"
34                            : "No␣habia␣nada␣que␣terminar"));
35     System.out.println();
36     System.out.println("objeto␣elTuyo:\n" + elTuyo.daTransmision());
37     System.out.println();
```

```
38      Disco elNuestro = (Disco)(elMio.copiaDisco());
39      System.out.println(elMio.daTransmision());
40      System.out.println("Disco 'elMio':\n" + elMio);
41      System.out.println();
42      System.out.println("Disco 'elNuestro':\n" + elNuestro);
43      System.out.println();
44      System.out.println("Hasta luego");
45    }
46 }
```

En las líneas 1 y 3 simplemente estamos definiendo una clase y colocándola en el paquete discos. En la línea 5 empezamos con la declaración y definición del método main de esta clase, que una vez compiladas todas las clases del proyecto, será invocado desde la línea de comandos, en el subdirectorio en el que se encuentra el subdirectorio discos (el superior de discos) con:

```
elisa$java discos/Usuario
```

A continuación describimos, casi línea por línea, qué es lo que va a hacer la ejecución del método main de la clase Usuario. Usaremos una tabla para abreviar el texto.

Línea	Descripción (semántica)
9	Declaras una cadena para el saludo de la clase.
10	Usas el método println del objeto estático out de la clase System para imprimir la cadena declarada.
11	Invocación al constructor sin parámetros, que se va a comunicar con el usuario a través de la consola para obtener los valores de las constantes.
12	Como el constructor sin parámetros únicamente inicializa las constantes, inicializamos el atributo permitidas. Lo tenemos que hacer con el método que provee la clase, pues main no tiene acceso al interior de un objeto, más que aquello que sea público o de paquete.
13	Se construye un nuevo objeto. El orden de los argumentos es: (TIPO_DISCO,NOMBRE,ANHO,permitidas). Regresa una referencia al objeto construido.
14	Como muestraDisco(**String**) regresa una cadena, la estamos guardando en la variable mio, que es de tipo referencia a una cadena.
15	Escribimos en la consola el contenido de la variable mio.
16 y	Se imprime una cadena en la consola.
19	
17	Se concatena la cadena que produce elMio.toString() con otras cadenas.
18	Se cambia el estado del objeto elTuyo, obteniendo primero el valor actual del atributo activas en este objeto, le suma 1 y ese es el nuevo valor.
19 y 20	Combina cadenas, una de las cuales, por aparecer en la concatenación de cadenas, es el resultado de elTuyo.toString(), y guarda esta cadena en la variable tuyo, de tipo cadena (referencia a una cadena).

Línea	Descripción (semántica)
23	Escribe la cadena referida por tuyo.
26 y 27	Como el método daTransmision() regresa una cadena, se puede pegar con otras e imprimirlas.
28, 30, 35, 37, 41 y 43	Escriben una línea en blanco.
24	Se modifica el atributo permitidas del objeto elTuyo.
25	Se modifica el atributo activas del objeto elTuyo.
26 y 27	Como el método daTransmision() regresa una cadena, se puede pegar con otras e imprimirlas.
29	Nuevamente se pide una transmisión, pero el objeto elTuyo ya no debe tener transmisiones disponibles.
31 a 34	Se intenta terminar una transmisión del objeto elTuyo. Como el método terminaTransmision devuelve un valor booleano, se usa la expresión ⟨*expresion booleana*⟩ **?** ⟨$expresion_T$⟩ **:** ⟨$expresion_F$⟩ conocida como *condicional aritmética*. Si la ⟨*expresión booleana*⟩ se evalúa a verdadera, lo que regresa la condicional es la $expresión_T$, mientras que si se evalúa a falso regresa $expresión_F$. Lo único que exige es que las dos expresiones para falso y verdadero sean del mismo tipo. La expresión está entre paréntesis para que primero la evalúe y después la concatene.
38	Construimos un nuevo objeto Disco usando el método copiaDisco con el objeto elMio. Pero este método regresa una referencia tipo **ServiciosDisco**, que no es un subtipo de Disco, por lo que no compila. Como Disco implementa a la interfaz **ServiciosDisco**, tenemos que aplicar un *casting* con Disco.
40 a 42	Se invocan métodos de los distintos objetos aprovechando que cuando aparece el nombre de un objeto en una cadena, automáticamente se invoca al método toString() de la clase.

Te recomendamos que bajes el archivo Usuario.java disponible en el curso. Compilas y ejecuta, para observar lo que sucede en la consola y ver si coincide con la descripción dada.