Análisis y diseño de una solución orientada a objetos (OO)

Tarjeta de responsabilidades de los métodos

En este vídeo verás con detalle lo que corresponde a los métodos en las tarjetas de responsabilidades .

De la descripción de los métodos sabes también qué es lo que requiere el método de entrada (parámetros) para trabajar y de qué tipo es cada uno de ellos; también sabes qué tipo de valor entrega como resultado cada método. Lo que no sabes todavía es cómo va a hacer su trabajo, pero eso lo dejaremos para el final de esta lección. A continuación te mostramos una tabla con la información de entradas y salida de cada método, junto con la descripción de lo que debe hacer.

Los primeros métodos que verás son aquellos que construyen un objeto. Como tipo (y nombre) tienen el nombre de la clase que el objeto quiere implementar, ya que regresan un objeto construido de acuerdo a las especificaciones de esa clase.

Es en los constructores donde se *deben* asignar valores a los atributos declarados como constantes y que quieres que tengan valor distinto en cada objeto. Es decir, aquellos atributos declarados con el modificador **final** pero sin el modificador **static**. Si en tus constructores no aparecen las asignaciones del valor de este conjunto de atributos, el compilador de Java protestará, pues supone, y con razón, que las constantes no deben tener valores cero o **null**.

Java te da, por omisión y si tú no defines ningún otro constructor, un constructor que no recibe ninguna entrada, sin parámetros, y que coloca un cero binario en los atributos de tipo primitivo y una referencia nula en los atributos que corresponden a referencias, como son, por ejemplo, los atributos de tipo String. Por lo que mencionamos en el párrafo anterior, en este proyecto deberás declarar tus propios constructores, que serán tres y se definen en la siguiente tarjeta de responsabilidades.

Clase: **Disco**			
Responsabilidades (métodos): (constructores públicos)			
Nombre	**Regresa**	**Recibe**	**Descripción**
Disco	Disco	(nada)	Construye un Disco, pidiéndole al usuario los campos fijos
Disco	Disco	short tipo, String nombre, int anho	Construye un disco con los valores que permanecerán fijos, pero con cero en transmisiones permitidas y activas
Disco	Disco	short tipo, String nombre, int anho, int permisos	Construye un disco con los valores fijos y el número de transmisiones permitidas, pero con cero en transmisiones activas

El primer constructor, aunque no recibe parámetros, se espera que se comunique con el usuario a través de la consola, para que el usuario decida los valores correspondientes a las constantes y al número de transmisiones permitidas.

El segundo constructor deberá ser invocado con los argumentos que especifica como parámetros, lo mismo que el tercer constructor.

Ahora toca agregar a la tarjeta de responsabilidades los servicios que deberá dar un objeto que sea un ejemplar de la clase Disco y que revisaste al plantear el problema a resolver. Como el acceso de los atributos es privado, la clase debe tener métodos que den a conocer el valor de los atributos y, más importante aun, que puedan modificar un valor (que no se constante) siguiendo las reglas que ese valor debe tener. Como es común esta situación, Java tiene como convención que los identificadores de los primeros, los que proporcionan el valor de un atributo, empiecen con el verbo get y sigan con el identificador del atributo al que se refieren, pero empezando con mayúscula. En el caso de constantes simplemente se escribirá todo el identificador del atributo tal cual.

Para los métodos que van a actualizar un valor, su identificador debe empezar con el verbo set y seguir con el identificador del atributo al que deben modificar, también empezando con mayúscula. No puede haber métodos set para atributos declarados como **final**.

En la tabla que sigue, para ahorrar espacio, identificamos a estos métodos como setXXX y getXXX, donde XXX representa al identificador del atributo.

Clase: **Disco**

Responsabilidades (métodos): (métodos públicos)

Nombre	**regresa**	**recibe**	**Descripción**
daTransmision	String	(nada)	Verifica si hay transmisiones disponibles y regresa la fecha y hora en la que se otorga o un mensaje de que no hay transmisiones disponibles
terminaTransmision	boolean	(nada)	Decrementa el contador de transmisio–nes activas para este disco, si puede
muestraDisco	String	String	Muestra el contenido del disco anotan-do a qué corresponde cada dato y enca-bezado con la cadena que recibe
copiaDisco	ServiciosDisco	(nada)	El objeto se copia a sí mismo, construyendo un objeto idéntico, excepto por las transmisiones activas
toString	String	(nada)	Escribe el contenido del objeto con cada atributo ocupando un lugar fijo en la cadena
getXXX	valor del tipo del atributo	(nada)	Regresa el valor del atributo "XXX"
setXXX	void	valor del tipo del atributo	Actualiza el valor del atributo "XXX" al valor que recibe

Al hacer el análisis también tomaste en cuenta que necesitas una clase Usuario en el que haya un método main para probar a la clase Disco. La modificarás en el siguiente módulo para que

presente un menú y pueda elegir acciones a llevar a cabo, ya sea como el dueño de la empresa o como un usuario del streaming. Esa clase se describe en la siguiente tarjeta de responsabilidades:

Tabla 1 Descripción de la clase Usuario

Clase: **Usuario**			
Responsabilidades (métodos):			(servicios públicos)
Nombre	**Regresa**	**Recibe**	Descripción
main	nada	Sucesión de cadenas al invocar a la clase	Éste es el único método que puede iniciar la ejecución de un sistema o proyecto para construir objetos y verificar que trabajan adecuadamente

También debes agregar a la tarjeta de responsabilidades la comunicación con el usuario, esto es, quién puede llamar y usar a estos métodos y atributos (o campos), pero como no nos cabe en la tabla anterior, la copiaremos, excepto que el tercer campo va a ser quién lo puede invocar. La imagen de esta tarjeta se encuentra en las tablas 2 y 3.

Tabla 2 Tarjeta de comunicación para la clase Disco

Clase: Disco		
Elemento		**Comunicación** (quién lo puede usar)
Privado	tipo del disco	éste y todos los objetos de esta clase
	año de grabación	éste y todos los objetos de esta clase
	nombre	éste y todos los objetos de esta clase
	transmisiones permitidas	éste y todos los objetos de esta clase
	transmisiones activas	éste y todos los objetos de esta clase

Para el caso de los atributos, como son de acceso privado, únicamente tienen acceso a ellos los objetos construidos de esa clase. Es un poco contra intuitivo, pero todos los objetos de la clase se pueden ver entre sí. Es como si en tu familia todos los hermanos conocieran los secretos de todos sus hermanos y, en la vida real no es así. Pero en el caso de la programación en Java tenemos esta situación.

En el caso de los métodos, como en la tarjeta de responsabilidades únicamente se listan los servicios que va a dar un objeto de la clase, pero no cómo lo va a hacer, es lógico que sólo se muestren aquellos métodos que son públicos (o podrían ser de paquete, pero en este caso no es así). Como todo mundo tiene acceso a lo que es público, se puede acceder a todos los métodos donde se construya un objeto de la clase.

Tabla 3 Tarjeta de comunicación para la clase Disco

Clase: Disco		
Responsabilidades		**Comunicación** (quién lo puede usar)
Público	Constructores	cualquier usuario que conozca al disco
	obtén valor de atributo	cualquier usuario que conozca al disco
	pon valor a atributo	Para aquellos atributos que se pueden cambiar, cualquier usuario que conozca al objeto
	muestra el disco	cualquier usuario que conozca al disco
	transcribe disco	cualquier usuario que conozca al disco
	otorga transmisión	cualquier usuario que conozca al disco
	termina transmisión	cualquier usuario que conozca al disco
	Copia disco	cualquier usuario que conozca al disco

Cuando decimos "éste y todos los objetos de esta clase", queremos decir que mientras se identifique al objeto de que se trata, cualquier objeto puede ver todos los campos privados de cualquier otro objeto. Es como si vivieran "en la misma casa" y no hubiese puertas entre ellos. En cambio, los atributos públicos (o métodos) los puede usar cualquiera que sepa de cuál objeto se trata y lo hace pidiéndole al objeto (o a la clase, en el caso de los campos o métodos static) que los invoque consigo mismo.

0.0.1. Uso de la memoria durante la ejecución

Queremos insistir en cómo se va a repartir la memoria durante la ejecución de tu proyecto. Por un lado, tienes las clases compiladas donde residen todos los atributos o métodos que hayas declarado **static** (aunque no has declarado métodos **static** entre los servicios de tu clase, es posible que sea conveniente tener algunos de estos métodos). En este mismo espacio se encontrarán todas las clases que uses en el proyecto, algunas declaradas por ti y otras que usarás de la enorme biblioteca de Java.

En segundo término tenemos la pila de ejecución, donde se van a encontrar las variables primitivas y las de referencia de los objetos declaradas dentro de los métodos; se acomodan en la pila en el orden en que los los métodos sean invocados durante la ejecución que definas en el método main.

Por último tendremos un espacio en memoria, al que hemos denominado el *heap*, donde residirán todos los objetos que construyas durante la ejecución del proyecto.

Podemos resumir estos puntos en la tabla 4, en la siguiente página.

Tabla 4 Uso de memoria en ejecución

Tipo	**Uso**
Código	Clases compiladas y variables o métodos declarados **static** dentro de las clases.
Pila	Variables primitivas y variables de referencia que se declaran dentro de los métodos invocados durante la ejecución del proyecto.
Heap	Objetos construidos durante la ejecución del proyecto.