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

Identificadores

Para poder detallar mejor las tarjetas de responsabilidades debes asignarle a los campos y funciones nombres válidos en Java (adelantándonos un poco a la programación), lo que se conoce como *identificador*.

Los identificadores pueden consistir de letras mayúsculas o minúsculas, guiones bajos (_), dígitos (0,...,9) o símbolos de pesos ($), pero tienen que empezar con guion bajo o letra. Otro aspecto importante es que para Java las mayúsculas son distintas que las minúsculas, por lo que el identificador miCasa es distinto de MiCasa, micasa o cualquier combinación distinta que hagas de mayúsculas y minúsculas.

Las convenciones (reglas de educación, pero sin ser forzosas) para los identificadores de Java son las siguientes:

Generales: Se usan una o más palabras que describan a lo que estás identificando. A partir de la segunda, si es que hay, las palabras empiezan con mayúscula para distinguirlas. También se pueden separar las palabras entre sí con guiones bajos.

Clases e interfaces: Se usan sustantivos e incluso el primer sustantivo empieza con mayúscula. Siempre que lo consideres conveniente puedes separar la segunda (y tercera) palabra con guiones bajos, pero la mayoría de los casos no es necesario y luego, al programar, no recuerdas los guiones bajos.

Disco	representa a los datos de un disco
EquipoFutbol	se refiere a la representación y responsabilidades de un equipo de futbol
Equipo_Futbol	se espera que represente lo mismo que el anterior pero el identificador es distinto
GrupoCurso	descripción de lo que contiene el grupo de un curso y sus responsabilidades (métodos) asociados
MujerBonita	descripción de los atributos y métodos de una mujer bonita

Variables o atributos: Se usan sustantivos que empiezan con minúscula, aunque si hay más de un sustantivo, los siguientes empiezan con mayúscula o se separan con guiones bajos.

permisos	nombre
cuantos	i
j	count
lmpza	cntdr_hojas

Aunque para Java podemos escribir identificadores de hasta 65,000 símbolos válidos, la mayoría de las veces vas a escribirlos tan cortos como puedas. No debes sacrificar la claridad por hacerlo demasiado corto, como sería escribir c1 y c2 como contadores pero sin decir qué cuentan.

Métodos o funciones: Empiezan con un verbo imperativo con minúscula y después sustantivos que describen a quién se le hace la acción.

modifica_fecha	pide_nombre
muestraCodigo	mstraCodgo
armaRegstro	mueveFigura
mueve_Figura	cuentaRegstrs

Constantes: Para las constantes se usan sustantivos exclusivamente con mayúsculas. Si hay más de uno se separan entre sí por guion bajo o la división de palabras es clara, como en los siguientes ejemplos.

MAX_NUM_DISCOS	máximo número de discos
MAXDISCOS	máximo número de discos, pero con identificador distinto al anterior
NOMBRE_EMPLEADO	nombre del empleado
TAMANHO	tamaño de algo
FECHA	podría ser de nacimiento o algo así

La costumbre es que los identificadores tanto de las clases como de las interfaces empiecen con mayúscula, como los ejemplos anteriores. Además, para que Java las pueda encontrar, el archivo en que se escriben tienen que tener el mismo nombre que le dimos a la interfaz o clase, pero con la extensión .java. Por ejemplo una clase que se llama Estudiante debe estar en un archivo que se llame Estudiante.java.

Recordemos que siempre que un identificador contiene más de dos palabras, a partir de la segunda se empieza con mayúscula. Un identificador debe *describir*, lo más breve posible, a quien nombra.

Tipos en Java

Estamos viendo que los campos pueden contener distinto *tipo* de valores, esto es, dependiendo de para qué lo vamos a usar o cómo, puede ser un número, otro disco, una cadena de caracteres, un valor lógico (falso o verdadero). Decimos que Java tiene valores *primitivos* (que no se pueden subdividir y que son ellos mismos) y valores de *referencia*, que representan a otros objetos. Los tipos *primitivos* de Java de encuentran en la tabla 1.

Tabla 1 Tipos *primitivos* de Java

tipo	en Java	tamaño	Descripción
Números			
Enteros	long	64 bits	Enteros "grandes" con signo que van desde -2^{63} hasta $2^{63}-1$
	int	32 bits	Enteros (el tipo por omisión para enteros) con signo Rango -2^{31} hasta $2^{31}-1$
	short	16 bits	Enteros pequeños. Rango: -32,768 hasta 32,767
	byte	8 bits	Enteros muy pequeños. Rango -128 hasta 127
Reales	double	64 bits	Reales de doble precisión. Tipo por omisión de un número con decimales
	float	32 bits	Reales de precisión sencilla
Caracteres			
Carácter	char	16 bits	Letras, dígitos, caracteres especiales (+ * / , ; : etc.) en UNICODE
Valores lógicos			
Valores lógicos	boolean	16 bits	Valores de *false* o *true*

Irás viendo, conforme desarrolles más la solución al problema, cuál de los tipos es el adecuado, dependiendo de lo que quieras hacer o representar.

Como ya mencionamos, el otro tipo de valor que maneja Java es el tipo *referencia*. Como su nombre lo indica, estos valores se refieren a una dirección en memoria, un apuntador a dónde se encuentra, la referencia o dirección donde se construyó un objeto de algún tipo (el tipo lo da la clase). Por lo anterior debemos tener al menos una referencia por cada objeto que construyamos[1]. Como cada objeto de una clase puede ocupar una cantidad de espacio distinto que el de otra clase, sería complicado colocar los objetos en la pila de ejecución, que es algo así como la memoria principal. Por ello, por cada variable de tipo referencia que se declare en una aplicación se coloca siempre la misma cantidad de espacio en la pila. La Máquina Virtual de Java, que es donde se ejecuta tu proyecto, tiene un espacio reservado para objetos, llamado *heap*, que es donde se acomodan todos los objetos. Allí, cada objeto ocupará el espacio demandado por la clase que lo describe. La referencia es la dirección en el *heap* en el que se encuentra colocado el objeto.

Sin embargo un objeto puede estar declarado pero no definido. Esto quiere decir que ya tienes el identificador al que lo vas a asociar, pero no has construido el objeto. En este caso se dice que la referencia que contiene la variable asociada al objeto es una referencia nula o no válida, o que la referencia está vacía y lo vas a ver representado con el símbolo del conjunto vacío ($\varnothing$).

Tabla 2 Tipos dados por clases e interfaces

tipo	en Java	tamaño	Descripción
Referencia	Nombre de clase o interfaz	64 bits	Se guarda la dirección del heap donde se construye el objeto, o bien el valor null si no está definido todavía.

Ya has visto, además del disco y el usuario, una clase que vas a usar muchísimo: se trata de la clase String, donde cada cadena de caracteres que construyas, como se trata de un objeto, se va a alojar en el heap. Como es una clase tan común tiene formas sencillas de construirse "al vuelo", por lo que en lo que sigue las usarás frecuentemente.

Modificadores

Es importante en las tarjetas de responsabilidades determinar si un dato o función se encuentran en la clase compilada (static) o hay uno por cada objeto que se crea. También hay que especificar, aunque son independientes de a quién pertenece un dato o función, si su valor puede ser modificado después de ser definido o no. Si no puede ser modificado decimos que es una constante y le corresponde el modificador final.

Podemos resumir lo que acabamos de mencionar de la siguiente forma:

Objetivo de los modificadores para atributo o método:

- Dónde se encuentra el atributo o método:
  - Si se encuentra en la clase compilada entonces se usa **static**
  - Si se encuentra en cada objeto que se construya entonces no se usa nada.
- En el caso de atributos, si es modificable después de la construcción del objeto:
  - Se trata de una constante, no se puede modificar: **final**

[1] En cuanto hay objetos sin variables que contengan su dirección, Java se encarga de recuperar automáticamente esa memoria.

- Si se puede modificar en cualquier momento, no se usa nada.

- En el caso de métodos:
  - Si no puede ser redefinido en las subclases, se usa **final**
  - Si puede ser redefinido en las subclases no se usa nada.

Te insistimos en que lo que identifica a constantes (ya sean atributos o métodos) es el modificador **final**, mientras que el modificador **static** identifica en qué memoria va a residir lo que se está declarando.

Como ya tienes identificados en Java los tipos primitivos y el de cadenas, puedes hacer una tarjeta de responsabilidades más detallada que la anterior, que te servirá como base casi directa para la programación en Java. La imagen de esta tarjeta se encuentra en la tabla 3.

Tabla 3 Tarjeta de responsabilidades con tipos y modificadores anotados

Clase: **Disco**				
Atributos (datos, campos):				
Acceso	campo	**Modificador(es)**	**Tipo**	**Identificador**
private	tipo de disco	final	short	TIPO_DISCO
	nombre	final	String	NOMBRE
	año	final	int	ANHO
	transm. actvas		int	activas
	transm. prmtdas		int	permitidas

En lo que sigue verás con más detalle la parte que corresponde a las anotaciones en la tarjeta de responsabilidades de los métodos.