5.B. Atributos.

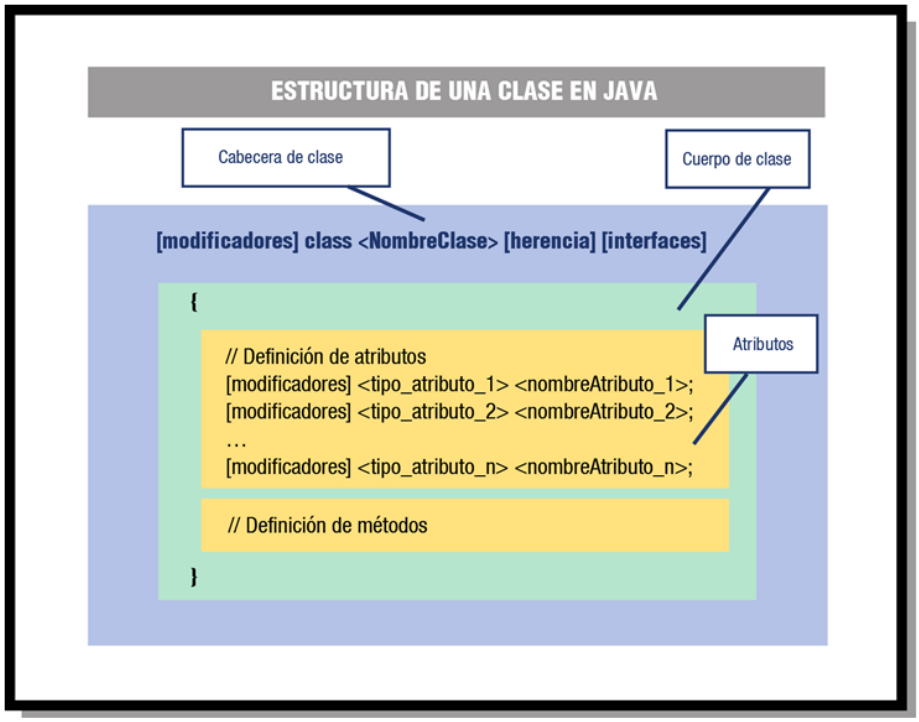
Sitio: Formación Profesional a Distancia
Curso: Programación
Libro: 5.B. Atributos.
Imprimido por: Iván Jiménez Utiel
Día: martes, 7 de enero de 2020, 22:17

Tabla de contenidos

1. Atributos.

Los **atributos** constituyen la estructura interna de los objetos de una clase. Se trata del conjunto de datos que los objetos de una determinada clase almacenan cuando son creados. Es decir es como si fueran variables cuyo ámbito de existencia es el objeto dentro del cual han sido creadas. Fuera del objeto esas variables no tienen sentido y si el objeto deja de existir, esas variables también deberían hacerlo (proceso de destrucción del objeto). Los atributos a veces también son conocidos con el nombre de **variables miembro** o **variables de objeto**.

Los atributos pueden ser de cualquier tipo de los que pueda ser cualquier otra variable en un programa en Java: desde tipos elementales como `int`, `boolean` o `float` hasta tipos referenciados como `arrays`, `Strings` u objetos.



Además del tipo y del nombre, la declaración de un atributo puede contener también algunos modificadores (como por ejemplo `public`, `private`, `protected` o `static`). Por ejemplo, en el caso de la clase `Punto` que habíamos definido en el aparado anterior podrías haber declarado sus atributos como:

```
public int x;
```

```
public int y;
```

De esta manera estarías indicando que ambos atributos son públicos, es decir, accesibles por cualquier parte del código programa que tenga acceso a un objeto de esa clase.

Como ya verás más adelante al estudiar el concepto de **encapsulación**, lo normal es declarar todos los atributos (o al menos la mayoría) como privados (`private`) de manera que si se desea acceder o manipular algún atributo se tenga que hacer a través de los métodos proporcionados por la clase.

Autoevaluación

Dado que normalmente se pretende encapsular el contenido de un objeto en su interior y permitir el

acceso a sus atributos únicamente a través de los métodos, los atributos de una clase suelen declararse con el modificador public**. ¿Verdadero o falso?**

- Verdadero.
- Falso.

1.1. Declaración de atributos.

La sintaxis general para la declaración de un atributo en el interior de una clase es:

```
[modificadores] <tipo> <nombreAtributo>;
```

Ejemplos:

```
int x;
```

```
public int elementoX, elementoY;
```

```
private int x, y, z;
```

```
static double descuentoGeneral;
```

```
final bool casado;
```

Te suena bastante, ¿verdad? La declaración de los atributos en una clase es exactamente igual a la declaración de cualquier variable tal y como has estudiado en las unidades anteriores y similar a como se hace en cualquier lenguaje de programación. Es decir mediante la indicación del tipo y a continuación el nombre del atributo, pudiéndose declarar varios atributos del mismo tipo mediante una lista de nombres de atributos separada por comas (exactamente como ya has estudiado al declarar variables).

La declaración de un **atributo** (o **variable miembro** o **variable de objeto**) consiste en la declaración de una variable que únicamente existe en el interior del objeto y por tanto su vida comenzará cuando el objeto comience a existir (el objeto sea creado). Esto significa que cada vez que se cree un objeto se crearán tantas variables como atributos contenga ese objeto en su interior (definidas en la clase, que es la plantilla o "molde" del objeto). Todas esas variables estarán encapsuladas dentro del objeto y sólo tendrán sentido dentro de él.

En el ejemplo que estamos utilizando de objetos de tipo `Punto` (instancias de la clase `Punto`), cada vez que se cree un nuevo `Punto p1`, se crearán sendos atributos `x`, `y` de tipo `int` que estarán en el interior de ese punto `p1`. Si a continuación se crea un nuevo objeto `Punto p2`, se crearán otros dos nuevos atributos `x`, `y` de tipo `int` que estarán esta vez alojados en el interior de `p2`. Y así sucesivamente...

Dentro de la declaración de un atributo puedes encontrar tres partes:

- **Modificadores**. Son palabras reservadas que permiten modificar la utilización del atributo (indicar el control de acceso, si el atributo es constante, si se trata de un atributo de clase, etc.). Los iremos viendo uno a uno.

- **Tipo**. Indica el tipo del atributo. Puede tratarse de un tipo primitivo (`int`, `char`, `bool`, `double`, etc) o bien de uno referenciado (objeto, array, etc.).

- **Nombre**. Identificador único para el nombre del atributo. Por convenio se **suelen utilizar las minúsculas**. En caso de que se trate de un identificador que contenga varias palabras, **a partir de la segunda palabra se suele poner la letra de cada palabra en mayúsculas**. Por ejemplo: primerValor, valor, puertaIzquierda, cuartoTrasero, equipoVecendor, sumaTotal, nombreCandidatoFinal, etc. Cualquier identificador válido de Java será admitido como nombre de atributo válido, pero es importante seguir este convenio para facilitar la legibilidad del código (todos los programadores de Java lo utilizan).

Como puedes observar, los atributos de una clase también pueden contener modificadores en su declaración (como sucedía al declarar la propia clase). Estos modificadores permiten indicar cierto comportamiento de un atributo a la hora de utilizarlo. Entre los modificadores de un atributo podemos distinguir:

- **Modificadores de acceso**. Indican la forma de acceso al atributo desde otra clase. Son modificadores excluyentes entre sí. Sólo se puede poner uno.

- **Modificadores de contenido**. No son excluyentes. Pueden aparecer varios a la vez.

- **Otros modificadores**: `transient` y `volatile`. El primero se utiliza para indicar que un atributo es transitorio (no persistente) y el segundo es para indicar al compilador que no debe realizar optimizaciones sobre esa variable. Es más que probable que no necesites utilizarlos en este módulo.

Aquí tienes la sintaxis completa de la declaración de un atributo teniendo en cuenta la lista de todos los modificadores e indicando cuáles son incompatibles unos con otros:

```
[private | protected | public] [static] [final] [transient] [volatile] <tipo>
<nombreAtributo>;
```

Vamos a estudiar con detalle cada uno de ellos.

1.2. Modificadores de acceso.

Los modificadores de acceso disponibles en Java para un atributo son:

- Modificador de acceso **por omisión** (o **de paquete**). Si no se indica ningún modificador de acceso en la declaración del atributo, se utilizará este tipo de acceso. Se permitirá el acceso a este atributo desde todas las clases que estén dentro del **mismo paquete** (**package**) que esta clase (la que contiene el atributo que se está declarando). No es necesario escribir ninguna palabra reservada. Si no se pone nada se supone se desea indicar este modo de acceso.

- Modificador de acceso `public`. Indica que **cualquier clase** (por muy ajena o lejana que sea) tiene acceso a ese atributo. No es muy habitual declarar atributos públicos (`public`).

- Modificador de acceso `private`. Indica que sólo se puede acceder al atributo desde **dentro de la propia clase**. El atributo estará "oculto" para cualquier otra zona de código fuera de la clase en la que está declarado el atributo. Es lo opuesto a lo que permite `public`.

- Modificador de acceso `protected`. En este caso se permitirá acceder al atributo desde cualquier **subclase** (lo verás más adelante al estudiar la **herencia**) de la clase en la que se encuentre declarado el atributo, y también desde las clases del **mismo paquete**.

A continuación puedes observar un resumen de los distintos niveles accesibilidad que permite cada modificador:

Cuadro de niveles accesibilidad a los atributos de una clase.

	Misma clase	Subclase	Mismo paquete	Otro paquete
Sin modificador (paquete)	Sí		Sí	
`public`	Sí	Sí	Sí	Sí
`private`	Sí			
`protected`	Sí	Sí	Sí	

¡Recuerda que los modificadores de acceso son excluyentes! Sólo se puede utilizar uno de ellos en la declaración de un atributo.

Ejercicio resuelto

Imagina que quieres escribir una clase que represente un **rectángulo** en el plano. Para ello has pensado en los

siguientes atributos:

- Atributos **x1**, **y1**, que representan la coordenadas del vértice inferior izquierdo del rectángulo. Ambos de tipo double (números reales).
- Atributos **x2**, **y2**, que representan las coordenadas del vértice superior derecho del rectángulo. También de tipo double (números reales).

Con estos dos puntos (x1, y1) y (x2, y2) se puede definir perfectamente la ubicación de un rectángulo en el plano.

Escribe una clase que contenga todos esos atributos teniendo en cuenta que queremos que sea una clase visible desde cualquier parte del programa y que sus atributos sean también accesibles desde cualquier parte del código.

Solución:

Dado que se trata de una clase que podrá usarse desde cualquier parte del programa, utilizaremos el modificador de acceso `public` para la clase:

```
public class Rectangulo
```

Los cuatro atributos que necesitamos también han de ser visibles desde cualquier parte, así que también se utilizará el modificador de acceso `public` para los atributos:

```
public double x1, y1; // Vértice inferior izquierdo
```

```
public double x2, y2; // Vértice superior derecho
```

De esta manera la clase completa quedaría:

```
public class Rectangulo {
```

```
public double x1, y1; // Vértice inferior izquierdo
```

```
public double x2, y2; // Vértice superior derecho
```

```
}
```

1.3. Modificadores de contenido.

Los modificadores de contenido **no son excluyentes** (pueden aparecer varios para un mismo atributo). Son los siguientes:

- Modificador `static`. Hace que el atributo sea común para todos los objetos de una misma clase. Es decir, todos los objetos de la clase compartirán ese mismo atributo con el mismo valor. Es un caso de miembro estático o miembro de clase: un **atributo estático** o **atributo de clase** o **variable de clase**.

- Modificador `final`. Indica que el atributo es una **constante**. Su valor no podrá ser modificado a lo largo de la vida del objeto. Por convenio, el nombre de los **atributos constantes** (`final`) se escribe con **todas las letras en mayúsculas**.

En el siguiente apartado sobre atributos estáticos verás un ejemplo completo de un atributo estático (`static`). Veamos ahora un ejemplo de atributo constante (`final`).

Imagina que estás diseñando un conjunto de clases para trabajar con expresiones geométricas (figuras, superficies, volúmenes, etc.) y necesitas utilizar muy a menudo la constante pi con abundantes cifras significativas, por ejemplo, 3.14159265. Utilizar esa constante literal muy a menudo puede resultar tedioso además de poco operativo (imagina que el futuro hubiera que cambiar la cantidad de cifras significativas). La idea es declararla una sola vez, asociarle un nombre simbólico (un identificador) y utilizar ese identificador cada vez que se necesite la constante. En tal caso puede resultar muy útil declarar un atributo `final` con el valor 3.14159265 dentro de la clase en la que se considere oportuno utilizarla. El mejor identificador que podrías utilizar para ella será probablemente el propio nombre de la constante (y en mayúsculas, para seguir el convenio de nombres), es decir, `PI`.

Así podría quedar la declaración del atributo:

```
class claseGeometria {
```

```
// Declaración de constantes
```

```
public final float PI= 3.14159265;
```

```
...
```

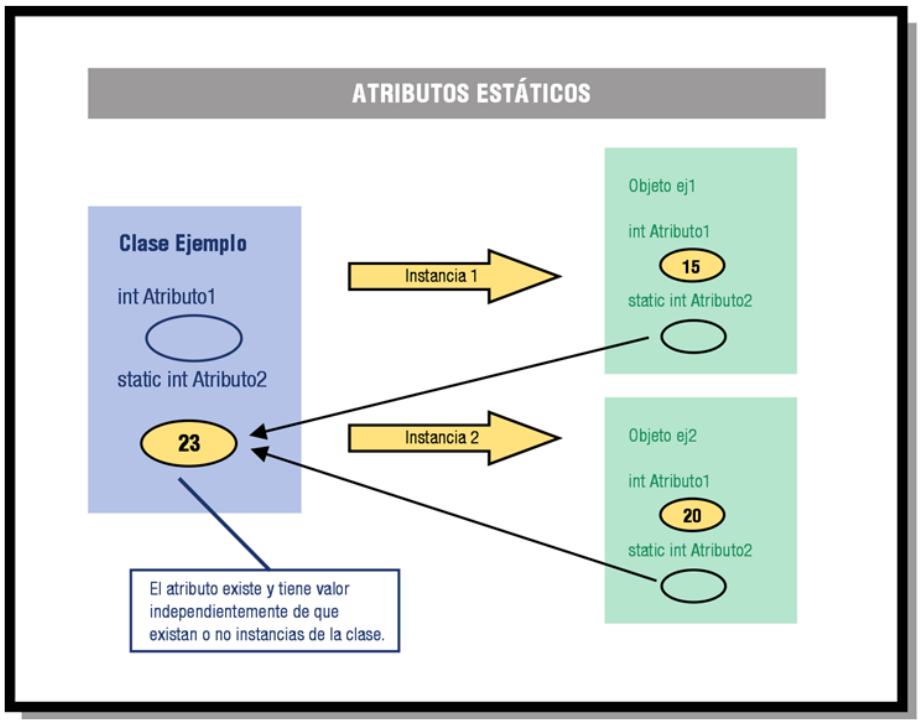
Autoevaluación

¿Con qué modificador puede indicarse en Java que un atributo es constante?

- Con el modificador constant.
- Con el modificador starter.
- Con el modificador final.
- Con el modificador static.

1.4. Atributos estáticos.

Como ya has visto, el modificador `static` hace que el atributo sea común (el mismo) para todos los objetos de una misma clase. En este caso sí podría decirse que la existencia del atributo no depende de la existencia del objeto, sino de la propia clase y por tanto sólo habrá uno, independientemente del número de objetos que se creen. El atributo será siempre el mismo para todos los objetos y tendrá un valor único independientemente de cada objeto. Es más, aunque no exista ningún objeto de esa clase, el atributo sí existirá y podrá contener un valor (pues se trata de un **atributo de la clase** más que del objeto).



Uno de los ejemplos más habituales (y sencillos) de atributos estáticos o de clase es el de un **contador** que indica el número de objetos de esa clase que se han ido creando. Por ejemplo, en la clase de ejemplo `Punto` podrías incluir un atributo que fuera ese contador para llevar un registro del número de objetos de la clase `Punto` que se van construyendo durante la ejecución del programa.

Otro ejemplo de atributo estático (y en este caso también constante) que también se ha mencionado anteriormente al hablar de miembros estáticos era disponer de un atributo **nombre**, que contuviera un `String` con el nombre de la clase. Nuevamente ese atributo sólo tiene sentido para la clase, pues habrá de ser compartido por todos los objetos que sean de esa clase (es el nombre de la clase a la que pertenecen los objetos y por tanto siempre será la misma e igual para todos, no tiene sentido que cada objeto de tipo `Punto` almacene en su interior el nombre de la clase, eso lo debería hacer la propia clase).

```
class Punto {
```

```
// Coordenadas del punto
```

```
private int x, y;
```

```
// Atributos de clase: cantidad de puntos creados hasta el momento
```

```
public static cantidadPuntos;
```

```
public static final nombre;
```

Obviamente, para que esto funcione como estás pensando, también habrá que escribir el código necesario para que cada vez que se cree un objeto de la clase `Punto` se incremente el valor del atributo `cantidadPuntos`. Volverás a este ejemplo para implementar esa otra parte cuando estudies los constructores.

Ejercicio resuelto

Ampliar el ejercicio anterior del rectángulo incluyendo los siguientes atributos:

- Atributo **numRectangulos**, que almacena el número de objetos de tipo rectángulo creados hasta el momento.
- Atributo **nombre**, que almacena el nombre que se le quiera dar a cada rectángulo.
- Atributo **nombreFigura**, que almacena el nombre de la clase, es decir, "Rectángulo".
- Atributo **PI**, que contiene el nombre de la constante PI con una precisión de cuatro cifras decimales.

No se desea que los atributos **nombre** y **numRectangulos** puedan ser visibles desde fuera de la clase. Y además se desea que la clase sea accesible solamente desde su propio paquete.

Solución:

Los atributos **numRectangulos**, **nombreFigura** y **PI** podrían ser **estáticos** pues se trata de valores más asociados a la propia clase que a cada uno de los objetos que se puedan ir creando. Además, en el caso de **PI** y **nombreFigura**, también podría ser un **atributo final**, pues se trata de valores únicos y constantes (3.1416 en el caso de **PI** y "Rectángulo" en el caso de **nombreFigura**).

Dado que no se desea que se tenga accesibilidad a los atributos **nombre** y **numRectangulos** desde fuera de la clase podría utilizarse el atributo `private` para cada uno de ellos.

Por último hay que tener en cuenta que se desea que la clase sólo sea accesible desde el interior del paquete al que pertenece, por tanto habrá que utilizar el modificador por omisión o de paquete. Esto es, no incluir ningún modificador de acceso en la cabecera de la clase.

Teniendo en cuenta todo lo anterior la clase podría quedar finalmente así:

```
class Rectangulo { // Sin modificador "public" para que sólo sea accesible desde el paquete

// Atributos de clase

private static int numRectangulos; // Número total de rectángulos creados

public static final String nombreFigura= "Rectángulo"; // Nombre de la clase

public static final double PI= 3.1416; // Constante PI

// Atributos de objeto

private String nombre; // Nombre del rectángulo

public double x1, y1; // Vértice inferior izquierdo

public double x2, y2; // Vértice superior derecho

}
```