Programación

Tema 4

Clases y Objetos en JAVA

Índice

Tema 4

Definición de una clase

La POO se inspira en una abstracción del mundo real, en la que los objetos se clasifican en grupos. Por ejemplo, todos los mamíferos de cuatro patas que dicen ¡guau! se engloban dentro del grupo de los perros. Si observamos a Pepa, Paco y Miguel, vemos que los tres pertenecen al mismo grupo: los tres son personas. Pepa es una persona, Paco es una persona y Miguel también es una persona. Todos pertenecen al grupo de las personas. En el argot de la POO, a cada uno de estos grupos se le denomina *clase*.

Definición de una clase

Podemos definir cada grupo o clase mediante las propiedades y comportamientos que presentan todos sus miembros. Una propiedad es un dato que conocemos de cada miembro del grupo, mientras que un comportamiento es algo que puede hacer.

Vamos a definir la clase persona mediante dos elementos:

- **Propiedades:** un nombre, una edad y una estatura. Tanto Pepa como Paco como Miguel tienen un nombre, una edad y una estatura; son datos que en cada uno tendrá un valor distinto.
- **Comportamientos:** Pepa, Paco y Miguel —en realidad todas las personas— pueden saludar, crecer o cumplir años, por ejemplo.

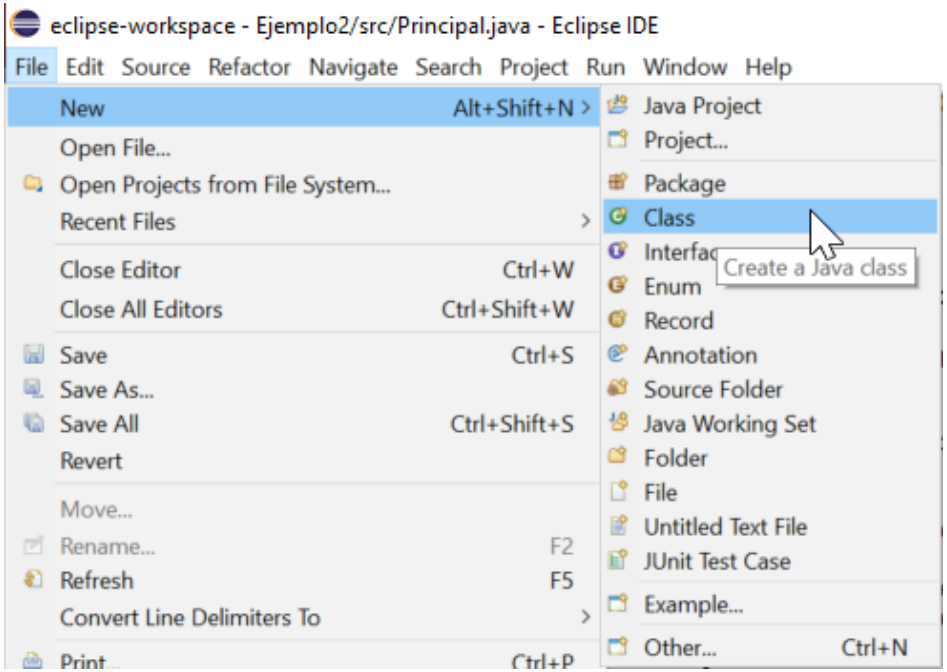
Como vemos, es posible definir una clase mediante un conjunto de propiedades y comportamientos. La sintaxis para definir una clase usa la palabra reservada `class`.

```
class NombreClase {
   //definición de la clase
}
```

Índice

Crear una clase en eclipse

Para definir una clase en Eclipse



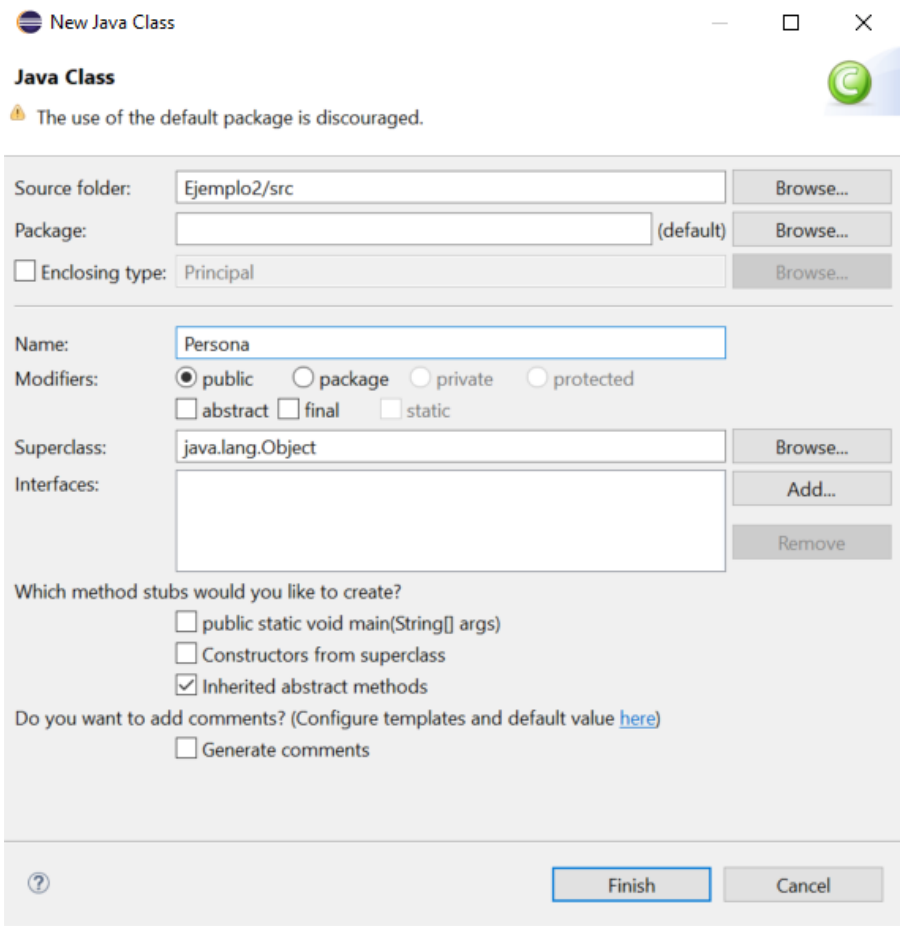
Crear una clase en eclipse

Escribimos el nombre

No seleccionamos el public static main

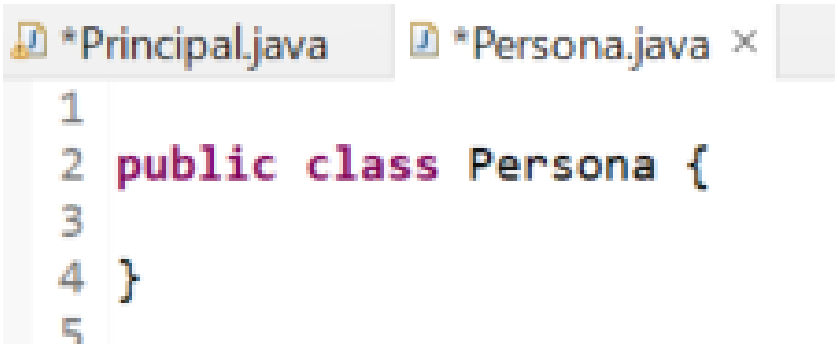
Dejamos todo como está de momento

Pulsamos Finish



Crear una clase en eclipse

Se ha creado una clase Persona, de momento sin ningún atributo ni método en su interior



Crear una clase en eclipse

Por ejemplo, la clase `Persona` se define:

```
class Persona {
   //definición de Persona
}
```

Crear una clase en eclipse

La declaración de una clase en Java tiene la siguiente estructura general:

[modificadores] **class** <NombreClase> [herencia] [interfaces] { // Cabecera de la clase
    // Cuerpo de la clase
    Declaración de los atributos
    Declaración de los métodos
}

Crear una clase en eclipse

En el ejemplo anterior hemos visto lo mínimo que se tiene que indicar en la cabecera de una clase (el nombre de la clase y la palabra reservada class).

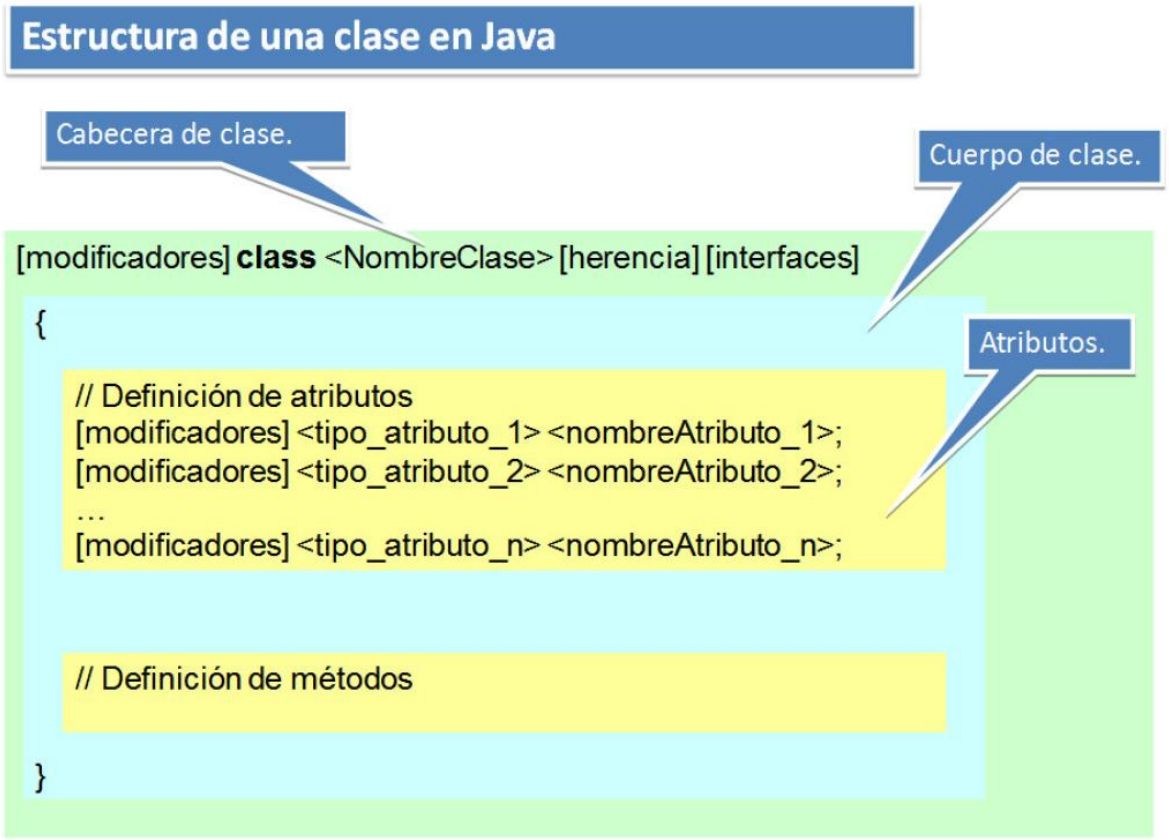
Se puede proporcionar bastante más información mediante modificadores y otros indicadores como por ejemplo el nombre de su superclase (si es que esa clase hereda de otra), si implementa algún interfaz y algunas cosas más que irás aprendiendo poco a poco.

Crear una clase en eclipse

A la hora de implementar una clase Java debes tener en cuenta:

- Por convenio, se ha decidido que en lenguaje Java los nombres de las clases deben de **empezar por una letra mayúscula**. Así, cada vez que observes en el código una palabra con la primera letra en mayúscula sabrás que se trata de una clase sin necesidad de tener que buscar su declaración. Además, si el nombre de la clase está formado por varias palabras, cada una de ellas también tendrá su primera letra en mayúscula. Siguiendo esta recomendación, algunos ejemplos de nombres de clases podrían ser: Recta, Circulo, Coche, CocheDeportivo, Jugador, JugadorFutbol, AnimalMarino, AnimalAcuatico, etc.
- El archivo en el que se encuentra una clase Java debe tener el mismo nombre que esa clase si queremos poder utilizarla desde otras clases que se encuentren fuera de ese archivo (**clase principal del archivo**).

Crear una clase en eclipse



Crear una clase en eclipse

Cabecera de una clase

En general, la declaración de una clase puede incluir los siguientes elementos y en el siguiente orden:

- Modificadores tales como public, abstract o final.
- El nombre de la clase (con la primera letra de cada palabra en mayúsculas, por convenio).
- El nombre de su clase padre (superclase), si es que se especifica, precedido por la palabra reservada extends ("extiende" o "hereda de").
- Una lista separada por comas de interfaces que son implementadas por la clase, precedida por la palabra reservada implements ("implementa").
- El cuerpo de la clase, encerrado entre llaves {}.

Crear una clase en eclipse

[modificadores] class<NombreClase>[extends<NombreSuperClase>][implements<NombreInterface1>]
[[implements <NombreInterface2>] ...] {

Por ejemplo:

class Persona {

….

….

}

En este caso no hay **modificadores**, ni indicadores de **herencia**, ni implementación de **interfaces**. Tan solo la palabra reservada **class** y el nombre de la clase. Es lo mínimo que puede haber en la cabecera de una clase.

Crear una clase en eclipse

La herencia y las interfaces las verás más adelante.

Vamos a ver ahora cuáles son los modificadores que se pueden indicar al crear la clase y qué efectos tienen.

Los modificadores de clase son:

[public] [final | abstract]

Crear una clase en eclipse

Veamos qué significado tiene cada uno de ellos:

- Modificador **public**. Indica que la clase es visible (se pueden crear objetos de esa clase) desde cualquier otra clase. Es decir, desde cualquier otra parte del programa. Si no se especifica este modificador, la clase sólo podrá ser utilizada desde clases que estén en el mismo paquete. El concepto de paquete lo veremos más adelante. Sólo puede haber una clase public (clase principal) en un archivo .java. El resto de clases que se definan en ese archivo no serán públicas.
- Modificador **abstract**. Indica que la clase es abstracta. Una clase abstracta no es instanciable. Es decir, no es posible crear objetos de esa clase (habrá que utilizar clases que hereden de ella). En este momento es posible que te parezca que no tenga sentido que esto pueda suceder (si no puedes crear objetos de esa clase, ¿para qué la quieres?), pero puede resultar útil a la hora de crear una jerarquía de clases. Esto lo verás también más adelante al estudiar el concepto de herencia.
- Modificador **final**. Indica que no podrás crear clases que hereden de ella. También volverás a este modificador cuando estudies el concepto de herencia. Los modificadores final y abstract son excluyentes (sólo se puede utilizar uno de ellos).

Todos estos modificadores y palabras reservadas las iremos viendo poco a poco, así que no te preocupes demasiado por intentar entender todas ellas en este momento.

Crear una clase en eclipse

En el ejemplo anterior de la clase Persona tendríamos una clase que sería sólo visible (utilizable) desde el mismo paquete en el que se encuentra la clase (modificador de acceso por omisión o de paquete, o package).

Desde fuera de ese paquete no sería visible o accesible.

Para poder utilizarla desde cualquier parte del código del programa bastaría con añadir el atributo public:

```
public class Punto {
```

Ejercicio

Crear una clase, llamada Principal, con el método *main* como hemos hecho hasta ahora y comprobar la cabecera de la clase.

¿Qué diferencia hay en las cabeceras de Persona y de la clase llamada Principal?

Índice

Crear una clase en eclipse

Como ya has visto anteriormente, el cuerpo de una clase se encuentra encerrado entre llaves y contiene la declaración e implementación de sus miembros. Los miembros de una clase pueden ser:

- **Atributos**, que especifican los datos que podrá contener un objeto de la clase.
- **Métodos**, que implementan las acciones que se podrán realizar con un objeto de la clase.

Una clase puede no contener en su declaración atributos o métodos, pero debe de contener al menos uno de los dos (la clase no puede estar vacía).

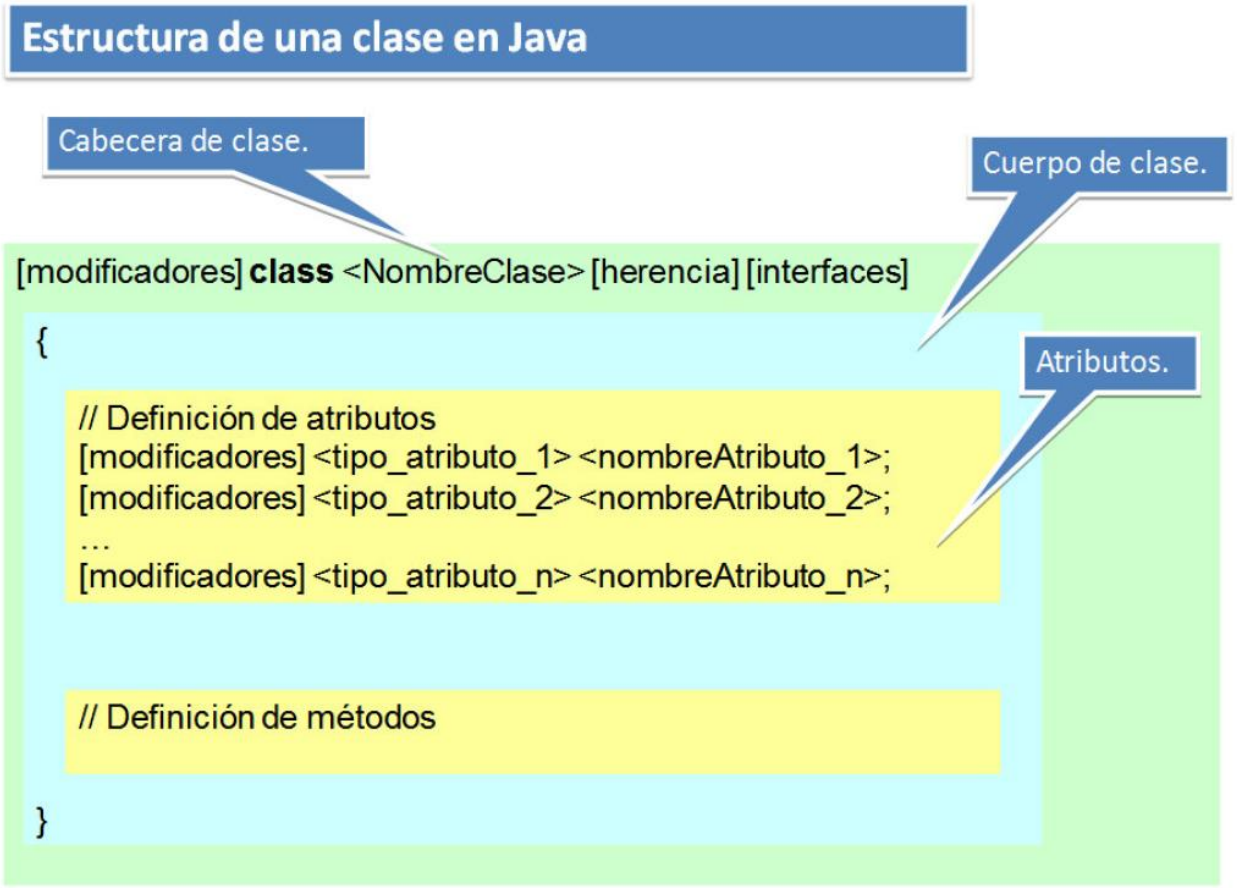
Crear una clase en eclipse

Los atributos constituyen la estructura interna de los objetos de una clase.

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.

Atributos


Estructura de una clase en Java
Cabecera de clase.
Cuerpo de clase.
[modificadores] class <NombreClase> [herencia] [interfaces]
{
Atributos.
// Definición de atributos
[modificadores] <tipo_atributo_1> <nombreAtributo_1>;
[modificadores] <tipo_atributo_2> <nombreAtributo_2>;
…
[modificadores] <tipo_atributo_n> <nombreAtributo_n>;
// Definición de métodos
}

Crear una clase en eclipse

Declaración de atributos

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

[modificadores] <tipo> <nombreAtributo>;

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.

Crear una clase en eclipse

Declaración de atributos

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).

Atributos

Además del tipo y del nombre, la declaración de un atributo puede contener también algunos modificadores (como por ejemplo public, private, protected o static).

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.

Atributos

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 que 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.

Atributos

Modificadores de acceso (Resumen)

	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

Dentro de la clase Persona, declarar 2 atributos públicos y uno private y con su tipo correspondiente:

- Nombre.
- Edad.
- Casado.

Ejercicio

Crear la clase Punto y dentro de la clase, declarar 1 atributo protegido y otro static y con su tipo correspondiente:

- PosicionX
- PosicionY.

Ejercicio

Crear la clase Cliente y dentro de la clase, declarar los atributos que creas necesarios, pensando que serán clientes de una sucursal bancaria.

Ejercicio

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.

Índice

Tema 4

Métodos

Los **métodos** son las herramientas que nos sirven para definir el comportamiento de un objeto en sus interacciones con otros objetos. Forman parte de la estructura interna del objeto junto con los atributos.

En el proceso de declaración de una clase que estás estudiando ya has visto cómo escribir la cabecera de la clase y cómo especificar sus atributos dentro del cuerpo de la clase. Tan solo **falta ya declarar los métodos**, que estarán también en el interior del cuerpo de la clase junto con los atributos.

Métodos

Los **métodos suelen declararse después de los atributos**. Aunque atributos y métodos pueden aparecer mezclados por todo el interior del cuerpo de la clase es aconsejable no hacerlo para mejorar la claridad y la legibilidad del código.

De ese modo, cuando echemos un vistazo rápido al contenido de una clase, podremos ver rápidamente los atributos al principio (normalmente ocuparán menos líneas de código y serán fáciles de reconocer) y cada uno de los métodos inmediatamente después. Cada método puede ocupar un número de líneas de código más o menos grande en función de la complejidad del proceso que pretenda implementar.

Los métodos representan la interfaz de una clase. Son la **forma que tienen otros objetos de comunicarse con un objeto** determinado solicitándole cierta información o pidiéndole que lleve a cabo una determinada acción.

Este modo de programar, facilita mucho la tarea al desarrollador de aplicaciones, pues le permite abstraerse del contenido de las clases haciendo uso únicamente del interfaz (métodos).

Métodos

Declaración de un método

La definición de un método se compone de dos partes:

- **Cabecera del método**, que contiene el nombre del método junto con el tipo devuelto, un conjunto de posibles modificadores y una lista de parámetros.
- **Cuerpo del método**, que contiene las sentencias que implementan el comportamiento del método (incluidas posibles sentencias de declaración de variables locales).

Los **elementos mínimos** que deben aparecer en la declaración de un método son:

- El tipo devuelto por el método.
- El nombre del método.
- Los paréntesis.
- El cuerpo del método entre llaves: { }.

Métodos

Por ejemplo, en la clase Punto que se ha estado utilizando en los apartados anteriores podrías encontrar

el siguiente método:

```
int obtenerX ()
{
// Cuerpo del método
...
}
```

Donde:

- El tipo devuelto por el método es int.
- El nombre del método es obtenerX.
- No recibe ningún parámetro: aparece una lista vacía entre paréntesis: ().
- El cuerpo del método es todo el código que habría encerado entre llaves: { }.

Dentro del cuerpo del método podrás encontrar declaraciones de variables, sentencias y todo tipo de estructuras de control (bucles, condiciones, etc.) **que has estudiado en los temas anteriores**.

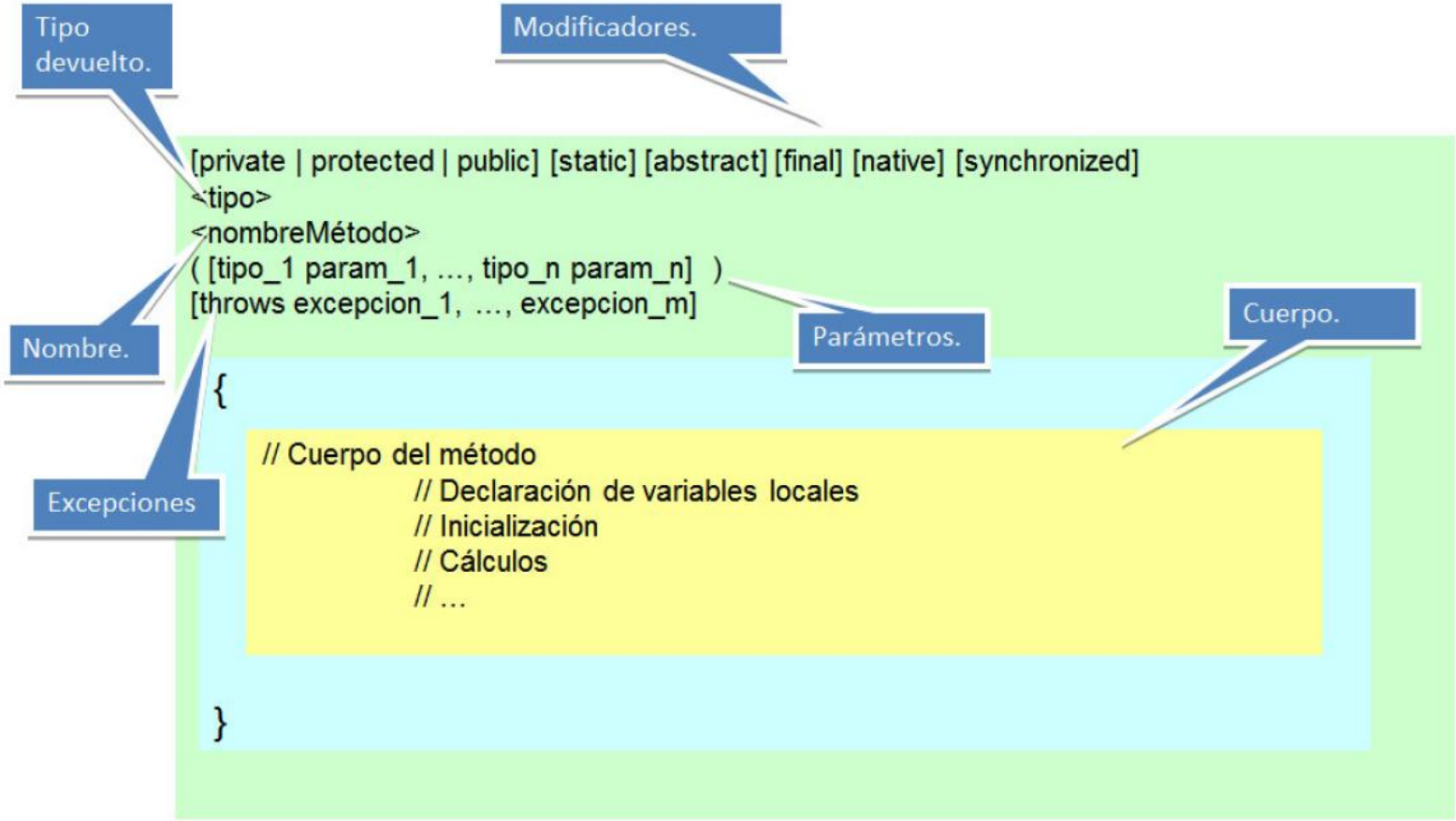
Métodos

Cabecera de método

La declaración de un método puede incluir los siguientes elementos:

1. **Modificadores** (como por ejemplo los ya vistos public o private, más algunos otros que irás conociendo poco a poco). No es obligatorio incluir modificadores en la declaración.
2. El **tipo devuelto** (o tipo de retorno), que consiste en el tipo de dato (primitivo o referencia) que el método devuelve tras ser ejecutado. Si eliges void como tipo devuelto, el método no devolverá ningún valor.
3. El **nombre del método**, aplicándose para los nombres el mismo convenio que 3. para los atributos. Una lista de parámetros separados por comas y entre paréntesis donde cada parámetro debe ir precedido por su tipo. Si el método no tiene parámetros la lista estará vacía y únicamente aparecerán los paréntesis.
4. Una **lista de excepciones** que el método puede lanzar. Se utiliza la palabra reservada throws seguida de una lista de nombres de excepciones separadas por comas. No es obligatorio que un método incluya una lista de excepciones, aunque muchas veces será conveniente. Más adelante haremos uso de ellas.
5. El **cuerpo del método**, encerrado entre llaves. El cuerpo contendrá el código del método (una lista sentencias y estructuras de control en lenguaje Java) así como la posible declaración de variables locales.

Métodos


Tipo devuelto.
Modificadores.
[private | protected | public] [static] [abstract] [final] [native] [synchronized]
<tipo>
<nombreMétodo>
([tipo_1 param_1, …, tipo_n param_n])
[throws excepcion_1, …, excepcion_m]
Nombre.
Parámetros.
Cuerpo.
Excepciones
{
// Cuerpo del método
// Declaración de variables locales
// Inicialización
// Cálculos
// …
}

Métodos

La **sintaxis general de la cabecera** de un método podría entonces quedar así:

[private | protected | public] [static] [abstract] [final] [native] [synchronized]

<tipo> <nombreMétodo> ([<lista_parametros>])

[throws <lista_excepciones>]

Modificadores en la declaración de un método

En la declaración de un método también pueden aparecer modificadores (como en la declaración de la clase o de los atributos). Un método puede tener los siguientes tipos de modificadores:

- Modificadores **de acceso**. Son los mismos que en el caso de los atributos (por omisión o de paquete, public, private y protected) y tienen el mismo cometido (acceso al método sólo por parte de clases del mismo paquete, o por cualquier parte del programa, o sólo para la propia clase, o también para las subclases).
- Modificadores **de contenido**. Son también los mismos que en el caso de los atributos (static y final) aunque su significado no es el mismo.
- **Otros** modificadores (no son aplicables a los atributos, sólo a los métodos): abstract, native, synchronized.

Métodos

Un método **static** es un método cuya implementación es igual para todos los objetos de la clase y sólo tendrá acceso a los atributos estáticos de la clase (dado que se trata de un método de clase y no de objeto, sólo podrá acceder a la información de clase y no la de un objeto en particular). Este tipo de métodos pueden ser llamados sin necesidad de tener un objeto de la clase instanciado.

En Java un ejemplo típico de métodos estáticos se encuentra en la clase Math, cuyos métodos son todos estáticos (Math.abs, Math.sin, Math.cos, etc.). Como habrás podido comprobar en este ejemplo, la llamada a métodos estáticos se hace normalmente usando el nombre de la propia clase y no el de una instancia (objeto), pues se trata realmente de un método de clase. En cualquier caso, los objetos ambién admiten la invocación de los métodos estáticos de su clase y funcionaría correctamente.

Un método **final** es un método que no permite ser sobrescrito por las clases descendientes de la clase a la que pertenece el método. Volverás a ver este modificador cuando estudies en detalle la **herencia**.

El modificador **native** es utilizado para señalar que un método ha sido implementado en código nativo (en un lenguaje que ha sido compilado a lenguaje máquina, como por ejemplo C o C++). En estos casos simplemente se indica la cabecera del método, pues no tiene cuerpo escrito en Java.

Métodos

Un método **abstract** (método abstracto) es un método que no tiene implementación (el cuerpo está vacío).

La implementación será realizada en las clases descendientes. Un método sólo puede ser declarado como abstract si se encuentra dentro de una clase abstract. También volverás a este modificador en unidades posteriores cuando trabajes con la **herencia**.

Por último, si un método ha sido declarado como **synchronized**, el entorno de ejecución obligará a que cuando un proceso esté ejecutando ese método, el resto de procesos que tengan que llamar a ese mismo método deberán esperar a que el otro proceso termine. Puede resultar útil si sabes que un determinado método va a poder ser llamado concurrentemente por varios procesos a la vez. **Por ahora no lo vas a necesitar**.

Métodos

Dada la cantidad de modificadores que has visto hasta el momento y su posible aplicación en la declaración de clases, atributos o métodos, veamos un resumen de todos los que has visto y en qué casos pueden aplicarse:

	Clase	Atributo	Método
Sin modificador (paquete)	Sí	Sí	Sí
public	Sí	Sí	Sí
private		Sí	Sí
protected	Sí	Sí	Sí
static		Sí	Sí
final	Sí	Sí	Sí
synchronized			Sí
native			Sí
abstract	Sí		Sí

Métodos

Parámetros en un método

La lista de parámetros de un método se coloca tras el nombre del método. Esta lista estará constituida por pares de la forma "<tipoParametro> <nombreParametro>". Cada uno de esos pares estará separado por comas y la lista completa estará encerrada entre paréntesis:

<tipo> nombreMetodo (<tipo_1> <nombreParametro_1>, <tipo_2> <nombreParametro_2>, ..., <tipo_n> <nombreParametro_n>)

Si la lista de parámetros es vacía, tan solo aparecerán los paréntesis:

<tipo> nombreMetodo ( )

Métodos

A la hora de declarar un método, debes tener en cuenta:

- Puedes incluir cualquier cantidad de parámetros. Se trata de una decisión del programador, pudiendo ser incluso una lista vacía.
- Los parámetros podrán ser de cualquier tipo (tipos primitivos, referencias, objetos, arrays, etc.).
- No está permitido que el nombre de una variable local del método coincida con el nombre de un parámetro.
- No puede haber dos parámetros con el mismo nombre. Se produciría ambigüedad.
- Si el nombre de algún parámetro coincide con el nombre de un atributo de la clase, éste será ocultado por el parámetro. Es decir, al indicar ese nombre en el código del método estarás haciendo referencia al parámetro y no al atributo. Para poder acceder al atributo tendrás que hacer uso del operador de autorreferencia this, que verás un poco más adelante.
- En Java el paso de parámetros es siempre por valor, excepto en el caso de los tipos referenciados (por ejemplo los objetos) en cuyo caso se está pasando efectivamente una referencia. La referencia (el objeto en sí mismo) no podrá ser cambiada pero sí elementos de su interior (atributos) a través de sus métodos o por acceso directo si se trata de un miembro público.

Cuerpo de un método

El interior de un método (cuerpo) está compuesto por una serie de sentencias en lenguaje Java:

- Sentencias de declaración de variables locales al método.
- Sentencias que implementan la lógica del método (estructuras de control como bucles o condiciones; utilización de métodos de otros objetos; cálculo de expresiones matemáticas, lógicas o de cadenas; creación de nuevos objetos, etc.). Es decir, todo lo que has visto en las unidades anteriores.
- Sentencia de devolución del valor de retorno (return). Aparecerá al final del método y es la que permite devolver la información que se le ha pedido al método. Es la última parte del proceso y la forma de comunicarse con la parte de código que llamó al método (paso de mensaje de vuelta). Esta sentencia de devolución siempre tiene que aparecer al final del método. Tan solo si el tipo devuelto por el método es void (vacío) no debe aparecer (pues no hay que devolver nada al código llamante).

Métodos

```
[modificadores] <tipo> <nombreMétodo> ( [tipo_1 param_1, …, tipo_n param_n]  )
[throws excepcion_1, …, excepcion_m]
{
// Cuerpo del método
        // Declaración de variables locales
        tipo1 var1;
        tipo2 var2:
        …
        tipo_resultado resultado;

        // Inicializaciones.
        // Procesamiento de información.
        // Cálculos
        …

        // Valor devuelto
        return resultado;
}
```

Variables locales.

Realización de tareas diversas.

Devolución de un valor final.

Ejercicio

Crear 3 métodos para devolver el valor de cada uno de los 3 atributos creados en la clase Persona

Ejercicio

Crear 2 métodos para devolver el valor de cada uno de los 2 atributos creados en la clase Punto

Ejercicio

Vamos a realizar nuestra propia clase Math, llamada MiMath y llamar a las funciones estáticas desde el método Main, al igual que hacíamos con la clase Math propia de Java.

Métodos estáticos a crear:

- Suma.
- Resta.
- Multiplicación.
- División.

Métodos GETTER (get – obtener)

Se trata de uno de los métodos más sencillos que se pueden implementar: un método que devuelve el valor de un atributo. En inglés se les suele llamar métodos de tipo get, que en inglés significa obtener.

Ejercicio

Crear 3 métodos para asignar el valor de cada uno de los 3 atributos creados en la clase Persona

Ejercicio

Crear 2 métodos para asignar el valor de cada uno de los 2 atributos creados en la clase Punto

Métodos SETTER (set – establecer)

Además de esos métodos get, la clase también debe disponer de otros dos que sirven para la función opuesta (establecerX y establecerX).

Ahora no se devuelve ningún valor (el tipo devuelto es void y no hay sentencia return). En inglés se suele hablar de métodos de tipo set, que en inglés significa poner o fijar (establecer un valor

Métodos

Normalmente el código en el interior de un método será algo más complejo y estará formado un conjunto de sentencias en las que se realizarán cálculos, se tomarán decisiones, se repetirán acciones, etc.

Ejercicio

Añadir a la clase Persona, estos atributos:

- Apellidos.
- Dni.
- Peso.
- Sexo

Añadir a la clase Persona,

- Sus métodos getter y setter para los atributos creados.
- Método público que devuelva la descripción de la persona.

Ejercicio

Crear un clase Cuenta que tendrá los siguientes atributos: titular y cantidad (pueden tener decimales).

Crea sus métodos get, set y toString.

Tendrá dos métodos públicos:

- ingresar(double cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.
- retirar(double cantidad): se retira una cantidad a la cuenta, si restando la cantidad actual a la que nos pasan es negativa, la cantidad de la cuenta pasa a ser 0.

Índice

Tema 4

Encapsulación, control de acceso y visibilidad

Dentro de la Programación Orientada a Objetos ya has visto que es muy importante el concepto de **ocultación**, la cual ha sido lograda gracias a la **encapsulación** de la información dentro de las clases. De esta manera una clase puede ocultar parte de su contenido o restringir el acceso a él para evitar que sea manipulado de manera inadecuada.

Los **modificadores de acceso** en Java permiten especificar el ámbito de visibilidad de los miembros de una clase, proporcionando así un mecanismo de accesibilidad a varios niveles.

Encapsulación, control de acceso y visibilidad

Acabas de estudiar que cuando se definen los miembros de una clase (atributos o métodos), e incluso la propia clase, se indica (aunque sea por omisión) un modificador de acceso.

En función de la visibilidad que se desee que tengan los objetos o los miembros de esos objetos se elegirá alguno de los modificadores de acceso que has estudiado. Ahora que ya sabes cómo escribir una clase completa (declaración de la clase, declaración de sus atributos y declaración de sus métodos), vamos a hacer un repaso general de las opciones de visibilidad (control de acceso) que has estudiado.

Encapsulación, control de acceso y visibilidad

Los modificadores de acceso determinan si una clase puede utilizar determinados miembros (acceder a atributos o invocar miembros) de otra clase.

Existen dos niveles de control de acceso:

- A nivel **general** (nivel de clase): visibilidad de la propia clase.
- A nivel de **miembros**: especificación, miembro por miembro, de su nivel de visibilidad.

Encapsulación, control de acceso y visibilidad

En el caso de la clase, ya estudiaste que los niveles de visibilidad podían ser:

- **Público** (modificador public), en cuyo caso la clase era visible a cualquier otra clase (cualquier otro fragmento de código del programa).
- **Privada** al paquete (sin modificador o modificador "por omisión"). En este caso, la clase sólo será visible a las demás clases del mismo paquete, pero no al resto del código del programa (otros paquetes).

Encapsulación, control de acceso y visibilidad

En el caso de los miembros, disponías de otras dos posibilidades más de niveles de accesibilidad, teniendo un total de cuatro opciones a la hora de definir el control de acceso al miembro:

- **Público** (modificador public), igual que en el caso global de la clase y con el mismo significado (miembro visible desde cualquier parte del código).
- **Privado al paquete** (sin modificador), también con el mismo significado que en el caso de la clase (miembro visible sólo desde clases del mismo paquete, ni siquiera será visible desde una subclase salvo si ésta está en el mismo paquete).
- **Privado** (modificador private), donde sólo la propia clase tiene acceso al miembro.
- **Protegido** (modificador protected)

Ocultación de atributos. Métodos de acceso.

Los atributos de una clase suelen ser declarados como **privados** a la clase o, como mucho, **protected** (accesibles también por clases heredadas), pero **no como public**. De esta manera puedes evitar que sean manipulados inadecuadamente desde el exterior del objeto.

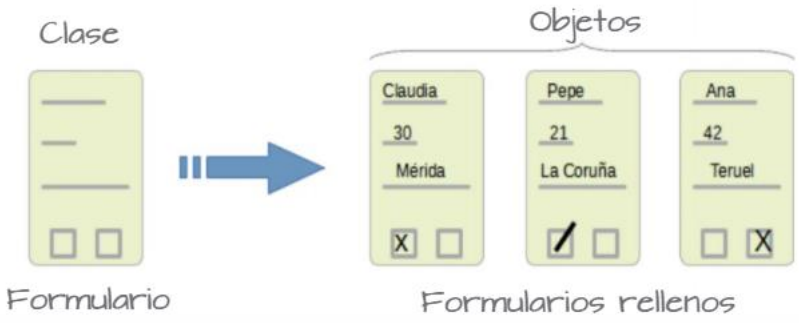
Se crearán métodos públicos que permitan acceder a esos atributos. Si se trata de un atributo cuyo contenido puede ser observado pero no modificado directamente, puede implementarse un método de "obtención" del atributo (en inglés se les suele llamar método de tipo get) y si el atributo puede ser modificado, puedes también implementar otro método para la modificación o "establecimiento" del valor del atributo (en inglés se le suele llamar método de tipo set).

Índice

Tema 4

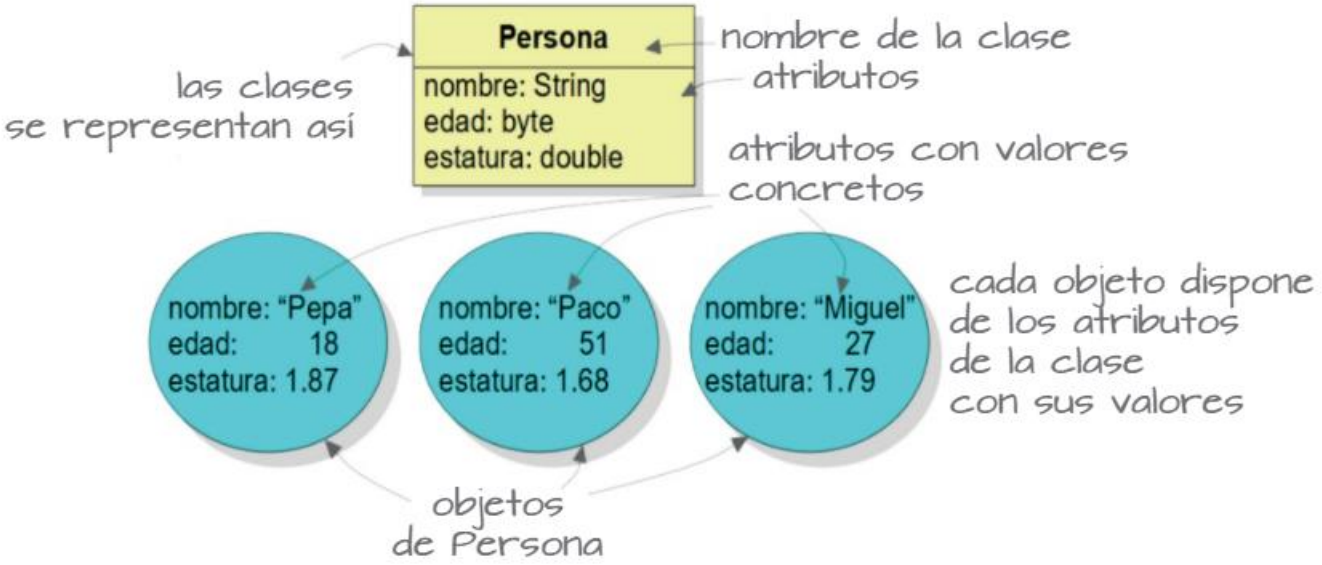
Objetos

Los elementos que pertenecen a una clase se denominan *instancias* u *objetos*. Cada uno tiene sus propios valores de los atributos definidos en la clase. Explicaremos este concepto con un símil: supongamos que una clase es un formulario donde se solicitan una serie de datos. Cada formulario relleno recoge distintos valores para los datos que se solicitan, siendo cada uno de los formularios rellenos, en nuestro símil, un objeto concreto. Todos los formularios cumplimentados —los objetos— tendrán la misma estructura. Esto es lógico, ya que utilizamos como plantilla el mismo formulario —la clase—. Sin embargo, cada formulario relleno —objetos— tendrá distintos valores: distintos nombres, direcciones, etcétera.



Objetos

Si nos fijamos de nuevo en Pepa, Paco y Miguel, nos damos cuenta de que cada uno de ellos es un objeto de la clase `Persona`.



Referencias

El comportamiento de los objetos en la memoria del ordenador y sus operaciones elementales (creación, asignación y destrucción) son idénticos al de las tablas. Esto es debido a que ambos, objetos y tablas, utilizan las referencias

Recordemos brevemente el concepto de referencia: la memoria de un ordenador está formada por pequeños bloques consecutivos identificados por un número único que se denomina *dirección de memoria.* Es habitual utilizar números hexadecimales; por este motivo, las direcciones de memoria tienen un aspecto similar a: 2a139f55.

Cualquier dato almacenado en la memoria ocupará, dependiendo de su tamaño, una serie de bloques consecutivos, y puede ser identificado mediante la dirección del primero de ellos. A esta primera dirección de memoria que identifica un objeto se le denomina en Java *referencia.*

Objetos

Declaración de un objeto

La declaración de un objeto se realiza exactamente igual que la declaración de una variable de cualquier tipo:

<tipo> nombreVariable;

En este caso el tipo será alguna clase que ya hayas implementado o bien alguna de las proporcionadas por la biblioteca de Java o por alguna otra biblioteca escrita por terceros.

Por ejemplo:

```
Punto p1;
Rectangulo r1, r2;
Coche cocheAntonio;
String palabra;
```

Declaración de un objeto

Esas variables (p1, r1, r2, cocheAntonio, palabra) en realidad son referencias (también conocidas como punteros o direcciones de memoria) que apuntan (hacen "referencia") a un objeto (una zona de memoria) de la clase indicada en la declaración.

Como ya estudiaste en la unidad anterior, un objeto recién declarado (referencia recién creada) no apunta a nada. Se dice que la referencia está vacía o que es una referencia nula (la variable objeto contiene el valor null). Es decir, la variable existe y está preparada para guardar una dirección de memoria que será la zona donde se encuentre el objeto al que hará referencia, pero el objeto aún no existe (no ha sido creado o instanciado). Por tanto se dice que apunta a un objeto nulo o inexistente.

Para que esa variable (referencia) apunte realmente a un objeto (contenga una referencia o dirección de memoria que apunte a una zona de memoria en la que se ha reservado espacio para un objeto) es necesario crear o instanciar el objeto. Para ello se utiliza el operador new.

Ejercicio

Declarar una variable de tipo entero llamada *edad*.

Declarar una variable de tipo entero llamada *entregado*.

Declarar una variable de tipo objeto de la clase Persona.

Declarar dos variables de tipo objeto de la clase Punto en la misma línea.

Creación de un objeto

Para poder crear un objeto (instancia de una clase) es necesario utilizar el operador new, el cual tiene la siguiente sintaxis:

nombreObjeto= new <ConstructorClase> ([listaParametros]);

Operador New

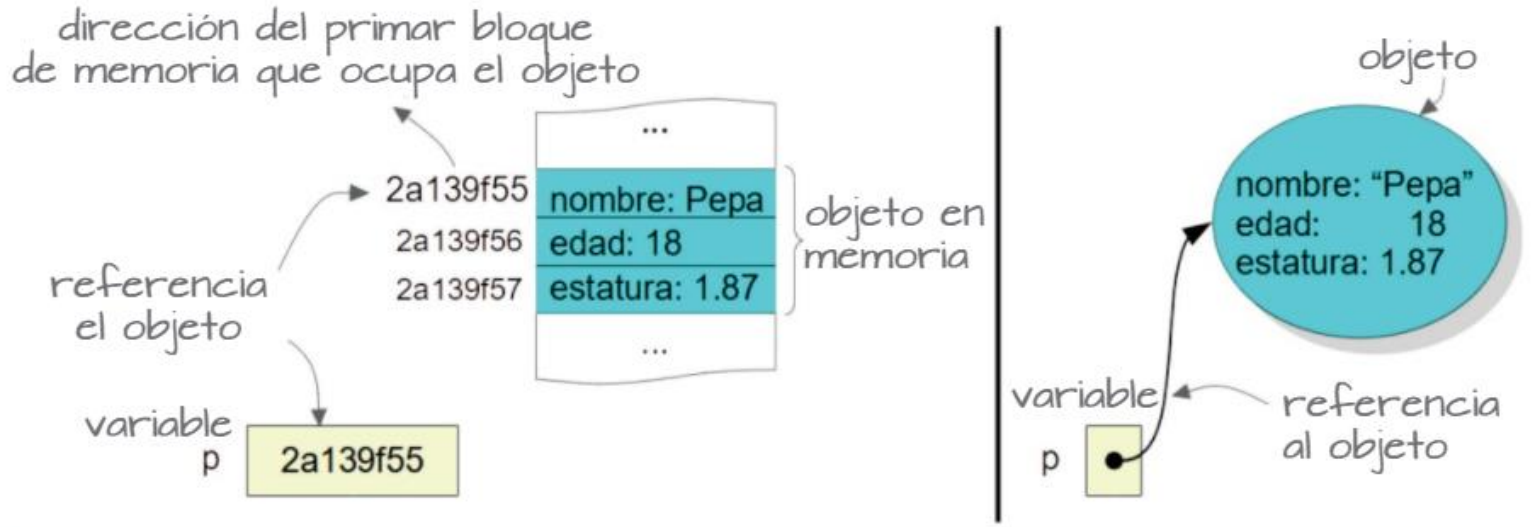
La forma de crear objetos, como en las tablas, es mediante el operador `new`.

```
p = new Persona();
```

En este caso, crea un objeto de tipo `Persona` y asigna su referencia a la variable `p`.

El operador `new` primero busca en memoria un hueco disponible donde construir el objeto. Este, dependiendo de su tamaño, ocupará cierto número consecutivo de bloques de memoria. Por último, devuelve la referencia del objeto recién creado, que se asigna a la variable `p`.

Objetos



Dos formas de representar una misma referencia. En la izquierda, tal y como se construye el objeto en memoria. En la derecha, la representación es más intuitiva, sustituyendo la referencia por una flecha que indica a qué objeto se accede desde la variable.

Objetos

El constructor de una clase (**ConstructorClase**) es un método especial que tiene toda clase y cuyo nombre coincide con el de la clase.

Es quien se encarga de **crear o construir el objeto**, solicitando la reserva de memoria necesaria para los atributos e inicializándolos a algún valor si fuera necesario. Dado que el constructor **es un método más de la clase**, podrá tener también su lista de parámetros como tienen todos los métodos.

De la tarea de reservar memoria para la estructura del objeto (sus atributos más alguna otra información de carácter interno para el entorno de ejecución) se encarga el propio entorno de ejecución de Java. Es decir, que por el hecho de ejecutar un método constructor, el entorno sabrá que tiene que realizar una serie de tareas (solicitud de una zona de memoria disponible, reserva de memoria para los atributos, enlace de la variable objeto a esa zona, etc.) y se pondrá rápidamente a desempeñarlas.

Cuando escribas el código de una clase no es necesario que implementes el método constructor si no quieres hacerlo.

Java se encarga de dotar de un **constructor por omisión** (también conocido como **constructor por defecto**) a toda clase.

Ese constructor por omisión se ocupará exclusivamente de las tareas de reserva de memoria. Si deseas que el constructor realice otras tareas adicionales, tendrás que escribirlo tú. **El constructor por omisión no tiene parámetros**.

El constructor por defecto no se ve en el código de una clase. Lo incluirá el compilador de Java al compilar la clase si descubre que no se ha creado ningún método constructor para esa clase.

Objetos

Algunos ejemplos de instanciación o creación de objetos podrían ser:

```
p1= new Punto ();
r1= new Rectangulo ();
r2= new Rectángulo();
cocheAntonio= new Coche();
palabra= String;
```

Objetos

Un objeto puede ser declarado e instanciado en la misma línea. Por ejemplo:

```
Punto p1= new Punto ();
```

Ejercicio

Crear dos objetos de tipo Persona.

Declarar un objeto de tipo Punto.

Manipulación de un objeto: utilización de métodos y atributos.

Una vez que un objeto ha sido declarado y creado (clase instanciada) ya se puede decir que el objeto existe en el entorno de ejecución, y por tanto que puede ser manipulado como un objeto más en el programa, haciendo uso de sus atributos y sus métodos.

Para acceder a un miembro de un objeto se utiliza el operador punto (.) del siguiente modo:

<nombreObjeto>.<nombreMiembro>

Donde <nombreMiembro> será el nombre de algún miembro del objeto (atributo o método) al cual se tenga acceso.

Objetos

Por ejemplo, en el caso de los objetos de tipo Punto que has declarado e instanciado en los apartados anteriores, podrías acceder a sus miembros de la siguiente manera:

```
Punto p1, p2, p3;
p1= new Punto();
p1.x= 5;
p1.y= 6;
System.out.printf ("p1.x: %d\np1.y: %d\n", p1.x, p1.y);
System.out.printf ("p1.x: %d\np1.y: %d\n", p1.obtenerX(), p1.obtenerY());
p1.establecerX(25);
p1.establecerX(30);
System.out.printf ("p1.x: %d\np1.y: %d\n", p1.obtenerX(), p1.obtenerY());
```

Es decir, colocando el operador punto (.) a continuación del nombre del objeto y seguido del nombre del miembro al que se desea acceder.

Ejercicio

Declarar todos los atributos de la clase Persona públicos.

Haciendo uso de la clase Persona, crear un método main y:

- Crear un objeto de tipo persona.
- Mostrar por consola su nombre.

Recolector de basura

Existen tres formas de conseguir que un objeto no esté referenciado:

- Es posible, aunque no tenga mucho sentido, crear un objeto y no asignarlo a ninguna variable.

```
new Persona();
```

- Otra posibilidad es asignar `null` a todas las variables que contenían una referencia a un objeto.
- También podemos asignar un objeto distinto a la variable.

```
Persona p = new Persona(); //objeto 1
p = new Persona(); //objeto 2. Ahora el objeto 1 queda sin referencia
```

En todos los casos, el objeto se queda perdido en memoria, es decir, no existe forma de acceder a él. Sin embargo está ocupando memoria . Si este comportamiento se repite demasiado, fortuita o malintencionadamente, es posible que se agote toda la memoria libre disponible, lo que impediría el normal funcionamiento del ordenador.

Objetos

En el momento en que disponemos de un objeto, podemos acceder a sus atributos mediante el nombre de la variable seguido de un punto (.) Por ejemplo, para asignar valores a los atributos del objeto referenciado por `p` escribimos:

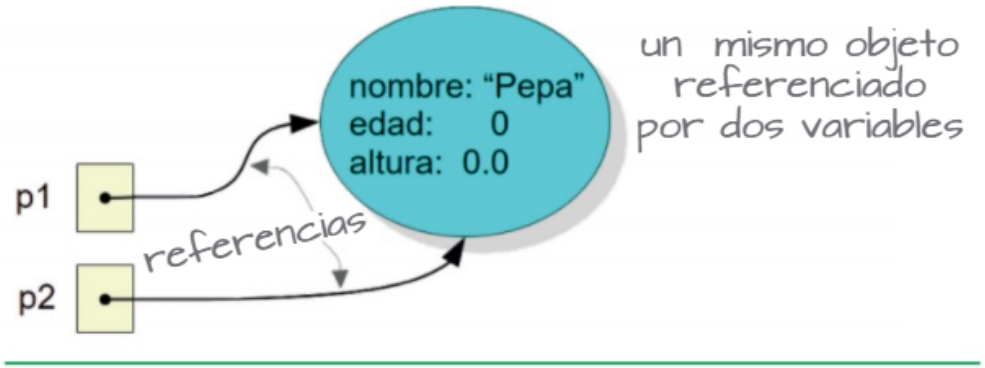
```
p = new Persona();
p.nombre = "Pepa";
p.edad = 18;
p.estatura = 1.87;
```

Es importante comprender que podemos acceder al mismo objeto mediante distintas variables que almacenen la misma referencia. La Figura 7.6 representa el siguiente código, donde un objeto está referenciado por dos variables:

```
Persona p1, p2;
p1 = new Persona(); //p1 referencia al objeto creado
p2 = p1; //asignamos a p2 la referencia contenida en p1
p2.nombre = "Pepa" //es equivalente a utilizar p1.nombre
```

Objetos

Ahora podemos acceder al objeto de dos maneras: mediante `p1` o mediante `p2`. En ambos casos estamos referenciando el mismo objeto.



Un mismo objeto referenciado por dos variables.

El mecanismo en el que varias variables comparten la misma referencia es aprovechado por la clase `String` para ahorrar espacio en textos usados frecuentemente, ya que Java se encarga por su cuenta de que todas las variables a las que se les han asignado idéntico literal cadena compartan su referencia.

Referencia NULL

El valor literal `null` es una referencia nula. Dicho de otra forma, una referencia a ningún bloque de memoria. Cuando declaramos una variable referencia se inicializa por defecto a `null`.

Hay que tener mucho cuidado de no intentar acceder a los miembros de una referencia nula, ya que se produce un error que termina la ejecución del programa de forma inesperada.

```
Persona p; //se inicializa por defecto a null
p.nombre //¡error!
```

La última instrucción genera un error del tipo: Null pointer exception, que significa que estamos intentando acceder a los atributos de un objeto que no existe.

El literal `null` se puede asignar a cualquier variable referencia.

```
Persona p = new Persona(); //p referencia un objeto
...
p = null; //p no referencia nada
```

Objetos

Para evitar este problema, Java dispone de un mecanismo llamado **recolector de basura** —*garbage collector*—, que se ejecuta de vez en cuando de forma transparente al usuario, y se encarga de comprobar, uno a uno, todos los objetos de la memoria. Si alguno de ellos no estuviera referenciado por ninguna variable, se destruye, liberando la memoria que ocupa.

Ejercicio

Realizar en el método Main un menú, con las siguientes opciones:

- 1. Ver cuenta
- 2. Ingresar dinero
- 3. Retirar dinero
- 4. Salir

La opción 1, saca por pantalla el estado de la cuenta.

La opción 2, pide al usuario el importe a ingresar por teclado. Cuando se finaliza esta acción, se debe mostrar al usuario el importe total en la cuenta.

La opción 3, pide al usuario el importe a retirar por teclado. Cuando se finaliza esta acción, se debe mostrar al usuario el importe total en la cuenta.

El programa finalizará cuando se pulse la opción 4 salir.

Ejercicio

Crear una clase DNI con los siguientes **atributos privados**:

- Dni: int
- Letra: char
- Valido: boolean

Métodos públicos :

- Validar(dni: String): valida que el dni coincida con la letra introducida.
- Getter y setter.
- toString()

Desde el método main crear un objeto de tipo DNI y pedir al usuario que introduzca su dni por teclado. Llamar al método validar mediante el objeto creado y después mostrar la información del objeto, es decir, dni, letra y si es válido o no.

Ejercicio

Realizar un programa con su método *main* correspondiente y realizar las siguientes acciones:

- Crear la clase Animal y poner sus atributos: tipo y nombre privados.
- Declarar 2 variables de tipo Animal en el main de la aplicación e instanciar animales en las dos variables.
- Crear sus métodos getter y setter en la clase Animal.
- Desde el main, al primer animal asignarle el nombre ‘Gato López’ y el tipo ‘Felino’ en el main.
- Desde el main, al segundo animal asignarle el nombre ‘Perro Ricki’ y el tipo ‘Canino’ en el main.
- Utilizando los métodos getter, sacar por consola cada uno de los valores de los atributos de los dos objetos creados.
- Llamar al método toString de ambos objetos para mostrar su estado.

- Añadir un atributo más a la clase Animal, llamado amigo y que sea de tipo Animal.
- Desde el método main, asignar al Gato, el amigo Perro, mediante su método correspondiente.
- Modificar el toString de la clase para que además del estado, diga también el nombre del su amigo. Si no tiene amigo (atributo nulo), debe decir “No tiene amigos”.

Ejercicio

Crear un array de tipo entero, que guarde 5 enteros.

Crear un array de tipo String, que guarde 5 String.

Crear un array de tipo Animal que guarde 5 animales.

Agregar a cada una posición del array un Animal, ya sea perro o gato con nombres y tipos que el usuario introduzca por teclado.

Realizar una función en el main que le pasemos un array de animales y nos muestre la descripción de todos los animales por consola.

Realizar una función que me devuelva un array solo los animales de tipo gato.

Realizar una función que me devuelva un array solo los animales de tipo perro.

Índice

Tema 4

Constructores

Como ya has estudiado en unidades anteriores, en el ciclo de vida de un objeto se pueden distinguir las fases de:

- Construcción del objeto.
- Manipulación y utilización del objeto accediendo a sus miembros.
- Destrucción del objeto.

Constructores

Como has visto en el apartado anterior, durante la fase de construcción o instanciación de un objeto es cuando se reserva espacio en memoria para sus atributos y se inicializan algunos de ellos.

Un constructor es un método especial con el mismo nombre de la clase y que se encarga de realizar este proceso.

El proceso de declaración y creación de un objeto mediante el operador new ya ha sido estudiado en apartados anteriores. Sin embargo las clases que hasta ahora has creado no tenían constructor.

Has estado utilizando los constructores por defecto que proporciona Java al compilar la clase. Ha llegado el momento de que empieces a implementar tus propios constructores

Constructores

Los métodos constructores se encargan de llevar a cabo el proceso de creación o construcción de un objeto.

Concepto de constructor

Un constructor es un método que tiene el mismo nombre que la clase a la que pertenece y que no devuelve ningún valor tras su ejecución. Su función es la de proporcionar el mecanismo de creación de instancias (objetos) de la clase.

Constructores

Cuando un objeto es declarado, en realidad aún no existe. Tan solo se trata de un nombre simbólico (una variable) que en el futuro hará referencia a una zona de memoria que contendrá la información que representa realmente a un objeto. Para que esa variable de objeto aún "vacía" (se suele decir que es una referencia nula o vacía) apunte, o haga referencia a una zona de memoria que represente a una instancia de clase (objeto) existente, es necesario "construir" el objeto. Ese proceso se realizará a través del método constructor de la clase.

Constructores

Por tanto para crear un nuevo objeto es necesario realizar una llamada a un método constructor de la clase a la que pertenece ese objeto. Ese proceso se realiza mediante la utilización del operador **new**.

Constructores

Hasta el momento ya has utilizado en numerosas ocasiones el operador new para instanciar o crear objetos. En realidad lo que estabas haciendo era una llamada al constructor de la clase para que reservara memoria para ese objeto y por tanto "crear" físicamente el objeto en la memoria (dotarlo de existencia física dentro de la memoria del ordenador). Dado que en esta unidad estás ya definiendo tus propias clases, parece que ha llegado el momento de que empieces a escribir también los constructores de tus clases.

Por otro lado, si un constructor es al fin y al cabo una especie de método (aunque algo especial) y Java soporta la **sobrecarga de métodos**, podrías plantearte la siguiente pregunta: ¿podrá una clase disponer de más de constructor? En otras palabras, ¿será posible la sobrecarga de constructores? La respuesta es afirmativa.

Constructores

Una misma clase puede disponer de varios constructores. Los constructores soportan la sobrecarga.

Constructores

Es necesario que toda clase tenga al menos un constructor. Si no se define ningún constructor en una clase, el compilador creará por nosotros un constructor por defecto vacío que se encarga de inicializar todos los atributos a sus valores por defecto (0 para los numéricos, null para las referencias, false para los boolean, etc.).

Constructores

Creación de constructores

Cuando se escribe el código de una clase normalmente se pretende que los objetos de esa clase se creen de una determinada manera. Para ello se definen uno o más constructores en la clase. En la definición de un constructor se indican:

- El tipo de acceso.
- El nombre de la clase (el nombre de un método constructor es siempre el nombre de la propia
- clase).
- La lista de parámetros que puede aceptar.
- Si lanza o no excepciones.
- El cuerpo del constructor (un bloque de código como el de cualquier método).

Creación de constructores

Como puedes observar, la estructura de los constructores es similar a la de cualquier método, con las excepciones de que no tiene tipo de dato devuelto (no devuelve ningún valor) y que el nombre del método constructor debe ser obligatoriamente el nombre de la clase.

Constructores

Si defines un constructor personalizado para una clase, el constructor por defecto deja de existir, es decir el compilador ya no lo crea, de manera que si quieres hacer uso de él, has de implementarlo tú mismo

Constructores

Si se ha creado un constructor con parámetros y no se ha implementado el constructor por defecto, el intento de utilización del constructor por defecto producirá un error de compilación (el compilador no lo hará por nosotros).

Un ejemplo de constructor para la clase Punto podría ser:

```
public Punto (int x, int y)
{
        this.x= x;
        this.y= y;
        cantidadPuntos++; // Suponiendo que tengamos un atributo estático cantidadPuntos
}
```

En este caso el constructor recibe dos parámetros. Además de reservar espacio para los atributos (de lo cual se encarga automáticamente Java), también asigna sendos valores iniciales a los atributos x e y. Por último incrementa un atributo (probablemente estático) llamado cantidadPuntos.

Utilización de constructores

Una vez que dispongas de tus propios constructores personalizados, la forma de utilizarlos es igual que con el constructor por defecto (mediante la utilización de la palabra reservada new) pero teniendo en cuenta que si has declarado parámetros en tu método constructor, tendrás que llamar al constructor con algún valor para esos parámetros.

Un ejemplo de utilización del constructor que has creado para la clase Punto en el apartado anterior podría ser:

```
Punto p1;
p1= new Punto (10, 7);
```

En este caso no se estaría utilizando el constructor por defecto sino el constructor que acabas de implementar en el cual además de reservar memoria se asigna un valor a algunos de los atributos.

Ejercicio

Modificar el ejercicio que valida el DNI para crear un constructor que reciba como parámetros el dni del usuario.

En el constructor hay que inicializar los atributos correspondientes.

Poner el método Validar(String dni) como privado ya que no se va a llamar desde la clase principal, solamente desde el constructor de la clase.

Probar desde el main de la clase Principal.

Constructores

Ejercicio

Crear la clase CuentaCorriente con los siguiente atributos privados:

- Nombre de la cuenta (ahorro, universitaria, préstamos, …).
- Titular de la cuenta.
- Saldo inicial.
- DNI.

En la clase CuentaCorriente sobrecargar los constructores para poder crear objetos.

- Con el DNI del titular de la cuenta y el saldo inicial.
- Con el DNI, nombre de la cuenta y saldo.

Escribir un programa que compruebe el funcionamiento de los métodos.

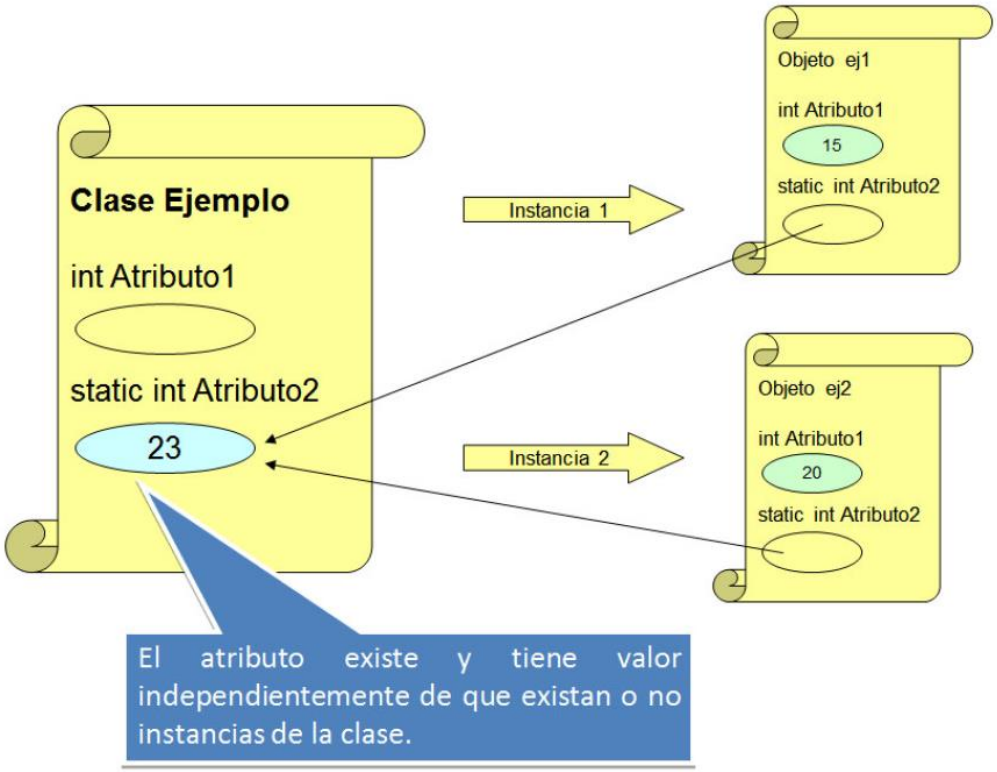
Índice

Tema 4

Atributos y métodos estáticos

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).

Atributos y métodos estáticos



Atributos y métodos estáticos

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.

Atributos y métodos estáticos

```
class Punto {
    // Coordenadas del punto
    private int x, y;
    // Atributos de clase: cantidad de puntos creados hasta el momento
    public static cantidadPuntos;
}
```

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.

Atributos y métodos estáticos

Un ejemplo de constructor para la clase Punto podría ser:

```
public Punto (int x, int y)
{
        this.x= x;
        this.y= y;
        cantidadPuntos++; // Suponiendo que tengamos un atributo estático cantidadPuntos
}
```

Atributos y métodos estáticos

Un **método estático** es un método que puede ser usado directamente desde la clase, **sin necesidad de tener que crear una instancia** para poder utilizar al método. También son conocidos como métodos de clase (como sucedía con los atributos de clase), frente a los métodos de objeto (es necesario un objeto para poder disponer de ellos).

Los métodos estáticos no pueden manipular atributos de instancias (objetos) sino atributos estáticos (de clase) y suelen ser utilizados para realizar operaciones comunes a todos los objetos de la clase, más que para una instancia concreta.

Atributos y métodos estáticos

Algunos ejemplos de operaciones que suelen realizarse desde métodos estáticos:

- Acceso a atributos específicos de clase: incremento o decremento de contadores internos de la clase (no de instancias), acceso a un posible atributo de nombre de la clase, etc.
- Operaciones genéricas relacionadas con la clase pero que no utilizan atributos de instancia. Por ejemplo una clase NIF (o DNI) que permite trabajar con el DNI y la letra del NIF y que proporciona funciones adicionales para calcular la letra NIF de un número de DNI que se le pase como parámetro. Ese método puede ser interesante para ser usado desde fuera de la clase de manera independiente a la existencia de objetos de tipo NIF.

Atributos y métodos estáticos

En la biblioteca de Java es muy habitual encontrarse con clases que proporcionan métodos estáticos que pueden resultar muy útiles para cálculos auxiliares, conversiones de tipos, etc.

Por ejemplo, la mayoría de las clases del paquete java.lang que representan tipos (Integer, String, Float, Double, Boolean, etc.) ofrecen métodos estáticos para hacer conversiones. Aquí tienes algunos ejemplos:

- static String valueOf (int i). Devuelve la representación en formato String (cadena) de un valor int. Se trata de un método que no tiene que ver nada en absoluto con instancias de concretas de String, sino de un método auxiliar que puede servir como herramienta para ser usada desde otras clases. Se utilizaría directamente con el nombre de la clase.

Por ejemplo:

String enteroCadena= String.valueOf (23).

Atributos y métodos estáticos

- static String valueOf (float f). Algo similar para un valor de tipo float.

Ejemplo de uso:

String floatCadena= String.valueOf (24.341).

Atributos y métodos estáticos

- static int parseInt (String s). En este caso se trata de un método estático de la clase Integer. Analiza la cadena pasada como parámetro y la transforma en un int.

Ejemplo de uso:

int cadenaEntero= Integer.parseInt ("-12").

Atributos y métodos estáticos

Todos los ejemplos anteriores son casos en los que se utiliza directamente la clase como una especie de caja de herramientas que contiene métodos que pueden ser utilizados desde cualquier parte, por eso suelen ser métodos públicos.

Ejercicio

Crear en la clase Punto el atributo estático cantidad de puntos.

Incrementarlo en el constructor cada vez que se crea un punto.

Ejercicio

Implementar una clase Calculadora, cuyos métodos serán:

- Sumar.
- Restar.
- Multiplicar.
- Dividir.

Para usar sus métodos no será necesario instanciar la clase.

Además, crear un atributo que se llame numeroOperaciones, donde se guarde el total del número de operaciones realizadas.

Realizar en el método main, un menú con las 4 opciones y una 5ª que sea Salir. El menú debe mostrarse hasta que se pulse la opción 5.

Al finalizar el programa se ha de mostrar el número total de operaciones por consola.

Ejercicio

Se debe modelar un sistema de almacenamiento de Prestamos en una biblioteca.

Para dar de alta un préstamos es necesario que tanto el usuario como el libro existan, es decir, debemos crear las clases **Usuario** y **Libro**, además de la mencionada **Prestamo**.

El menú principal será el siguiente, además del submenú de la opción 4:

1. **Dar de alta un libro**
2. **Dar de alta un usuario**
3. **Prestar libro**
4. **Listar**
    1. **Listar Usuarios**
    2. **Listar Libros**
    3. **Listar Prestamos actuales.**

Ejercicio

Para dar de alta un libro, es necesario pedir al usuario su ISBN, su título y autor.

Para dar de alta un usuario, es necesario su dni, nombre, apellidos y edad.

Para dar de alta un prestamos es necesario pedir el dni del usuario y el isbn del libro. Si alguno de los dos no existe hay que mostrar al usuario un mensaje indicando que no existe bien el usuario o bien el libro.

Atributos y métodos estáticos

Ejercicio

Crear una clase llamada Password que siga las siguientes condiciones:

- Que tenga los atributos longitud y contraseña . Por defecto, la longitud será de 8.
- Los constructores serán los siguiente:
  - Un constructor por defecto.
  - Un constructor con la longitud que nosotros le pasemos. Generara una contraseña aleatoria con esa longitud.
- Los métodos que implementa serán:
  - esFuerte(): devuelve un booleano si es fuerte o no, para que sea fuerte debe tener mas de 2 mayúsculas, mas de 1 minúscula y mas de 5 números.
  - generarPassword(): genera la contraseña del objeto con la longitud que tenga.
  - Método get para contraseña y longitud.
  - Método set para longitud.
- Pruebas en el Main:
  - Crear una contraseña por defecto y mostrar por consola su longitud y contraseña.
  - Crear una longitud con una longitud pedida al usuario por teclado.
  - Con las dos contraseñas llamar generarPassword y al método esFuerte. Sacar por consola su resultado.

Para generar la password, generar valores aleatorios entre el carácter **!** y el **~**

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Índice

Tema 4

Modificación de acceso

Paquetes

En Java es importante controlar la accesibilidad de unas clases desde otras por razones de seguridad y eficiencia. Esto se consigue mediante paquetes, que son contenedores que permiten guardar clases en compartimentos separados, de modo que podamos decidir, por medio de la importación, qué clases son accesibles y cómo se accede a ellas desde una clase que estemos implementando.

Todas las clases están dentro de algún paquete que, a su vez, pueden estar anidados, unos dentro de otros. Se considera que una clase que pertenece a un paquete, que a su vez está dentro de otro, solo pertenece al primero, pero no al segundo.

Modificación de acceso

Paquetes

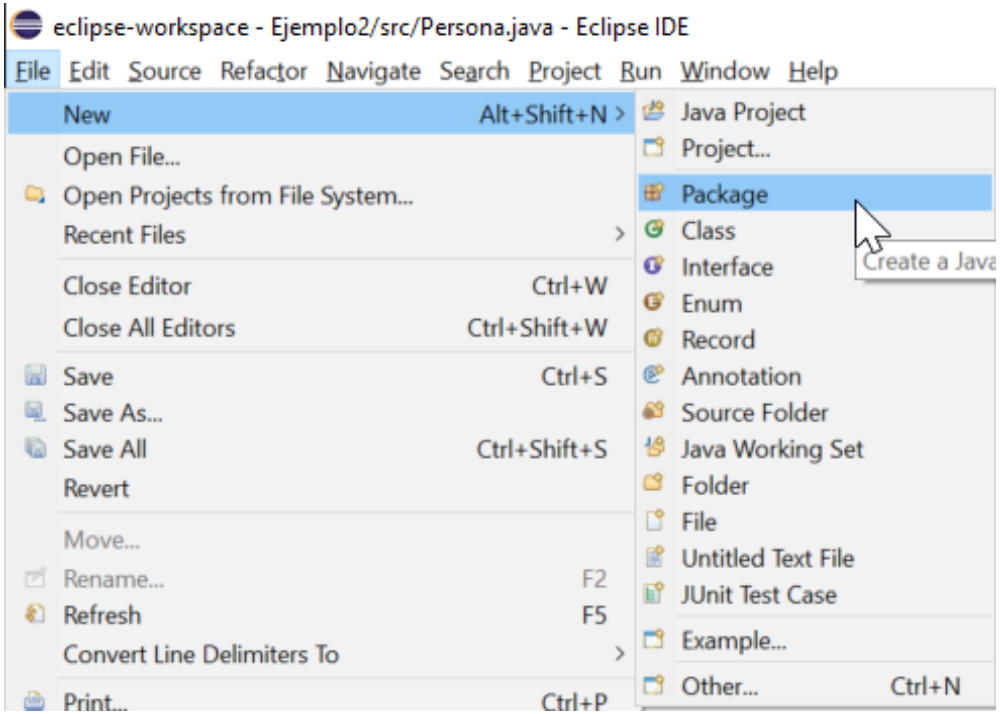
Un archivo fuente de Java es un archivo de texto con extensión .java, que se guarda en un paquete y que contiene los siguientes elementos:

- Una sentencia donde se especifica el paquete al que pertenece, que empieza con la palabra clave `package` seguida del nombre del paquete.
- Una serie opcional de sentencias de importación, con la palabra reservada `import`, que permite importar clases definidas en otros paquetes.
- La definición de una o más clases, de las cuales solo una puede ser declarada pública —por medio del modificador de acceso `public`—. De todas formas, es recomendable que en cada archivo fuente se defina una sola clase, que debe tener el mismo nombre que el archivo.

Modificación de acceso

Paquetes

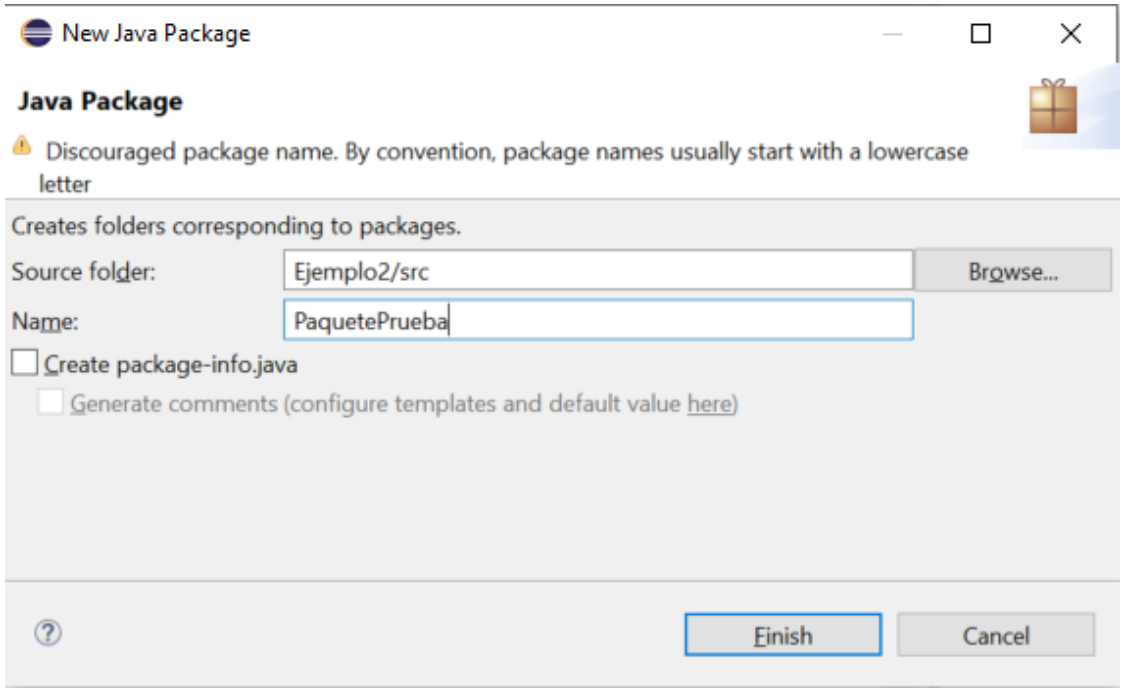
Para crear un paquete en Eclipse:



Modificación de acceso

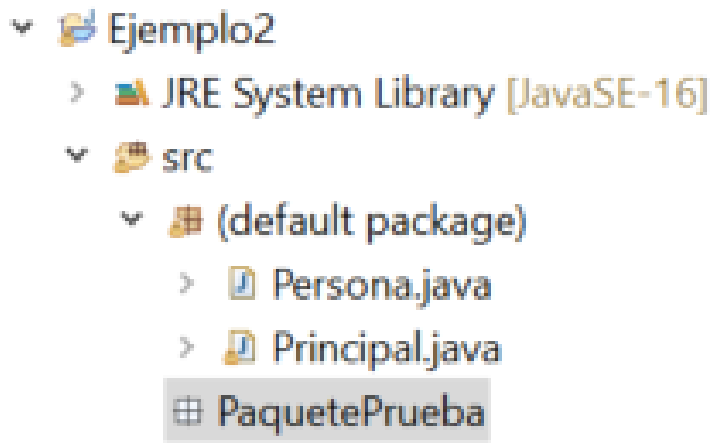
Paquetes

Le damos el nombre y aceptamos:



Paquetes

Se crea el paquete vacío, sin ninguna clase en su interior:



Modificación de acceso

Una clase será visible por otra, o no, dependiendo de si se ubican en el mismo paquete y de los modificadores de acceso que utilice. Estos modifican su visibilidad, permitiendo que se muestre u oculte.

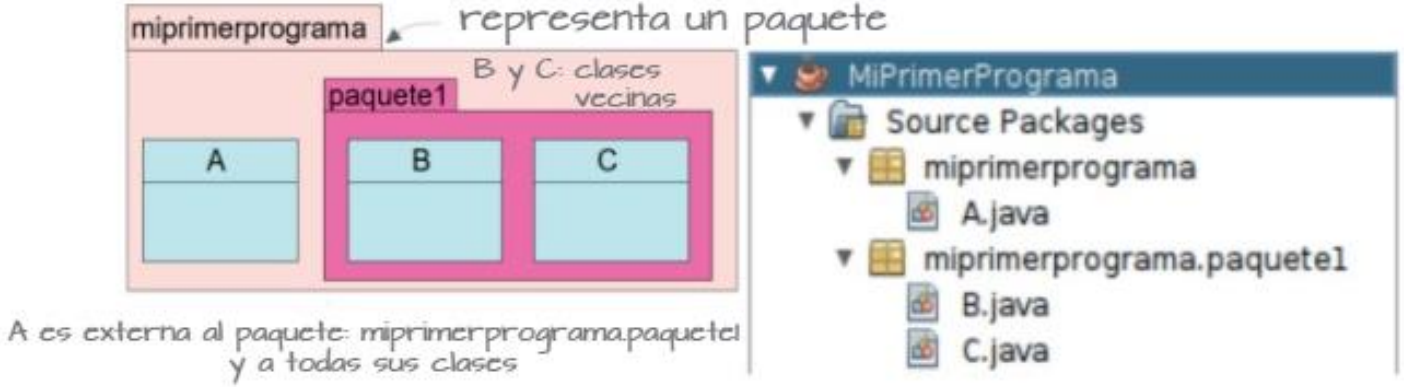
De igual manera que podemos modificar la visibilidad entre clases, es posible modificar la visibilidad entre los miembros de distintas clases, es decir, qué atributos y métodos son visibles para otras clases.

Modificación de acceso

Modificadores de acceso para las clases

Debido a la estructura de clases, organizadas en paquetes, que utiliza Java, dos clases cualesquiera pueden definirse de las siguientes formas

- **Clases vecinas:** cuando ambas pertenecen al mismo paquete.
- **Clases externas:** cuando se han definido en paquetes distintos.



Modificadores de acceso para las clases

Una aplicación puede entenderse como un conjunto de instrucciones que usan los servicios proporcionados por otras clases para resolver un problema. Para conocer qué servicios o herramientas están disponibles, las clases siguen el lema «si lo ves, puedes utilizarlo».

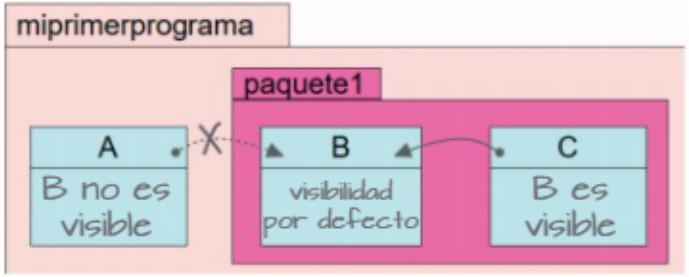
Modificación de acceso

Visibilidad por defecto

Cuando definimos una clase sin utilizar ningún modificador de acceso,

```
package miprimerprograma.paquete1;
class B { //sin modificador de acceso
   ...
}
```

se dice que usa visibilidad por defecto, que hace que solo sea visible por sus clases vecinas. En nuestro caso, B es visible por C, pero no será visible por A



Modificación de acceso

Ejercicio:

Implementar el anterior diagrama de clases con sus paquetes:

- Miprimerprograma.
  - Clase A.
- Paquete1:
  - Clase B: visibilidad por defecto.
  - Clase C: visibilidad pública.

```java
package miprimerprograma;

import paquete1.C;

public class A {

    public A() {
        B b = new B();
        C c = new C();
    }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Illegal modifier for the class Persona; only public, abstract & final are permitted

        at Persona.<init>(Persona.java:2)
        at Principal.main(Principal.java:11)
```

Modificación de acceso

Visibilidad total

En la Figura la clase B es invisible para A y para todas las clases externas. ¿Cómo podemos hacer que B sea visible desde A? Mediante el modificador de acceso `public`, la clase B, además de ser visible para sus vecinas, lo será desde cualquier clase externa usando una sentencia de importación. De esta forma, el modificador `public` proporciona visibilidad total a la clase.

Vamos a redefinir B para que tenga visibilidad total:

```
package miprimerprograma.paquete1;
public class B { //clase marcada como pública
   ...
}
```

A partir de ahora, cualquier clase, vecina o externa, puede crear objetos o acceder a los miembros públicos de B. Lo único que necesita una clase externa, como A, para acceder a B es importarla.

```
package miprimerprograma;
import miprimerprograma.paquete1.B; //ahora A puede usar la clase B

class A {
   ...
}
```

Se importan las clases, no los paquetes. Si queremos importar todas las clases públicas de un paquete en una sola sentencia de importación, se usa el asterisco.

Modificación de acceso

Ejercicio:

Declarar la clase B como pública y comprobar que podemos acceder a la clase igual que a la clase C

```
1  package miprimerprograma;
2
3  import paquete1.B;
4  import paquete1.C;
5
6  public class A {
7
8      public A() {
9          B b = new B();
10         C c = new C();
11     }
12 }
```

Modificación de acceso

Visibilidad total

```
package miprimerprograma;
import miprimerprograma.paquete1.*; //A puede usar cualquier clase pública
                                    //del paquete miprimerprograma.paquete1
class A {
   ...
}
```

La visibilidad entre clases puede resumirse como: una clase siempre será visible por sus clases vecinas. Que sea visible —previa importación— por clases externas dependerá de si está declarada como pública

Tabla Resumen de la visibilidad entre clases

	Visible desde…	
	clases vecinas	clases externas
sin modificador	✓	
public	✓	✓

Modificación de acceso

Ejercicio:

Declarar la clase A sin modificador de acceso.

Comprobar que no hay errores y puede usar los objetos del paquete1.

```
1  package miprimerprograma;
2
3  import paquete1.B;
4  import paquete1.C;
5
6  class A {
7
8      public A() {
9          B b = new B();
10         C c = new C();
11     }
12 }
```

Modificación de acceso

Modificadores de acceso para miembros

De igual manera que es posible modificar la visibilidad de una clase, podemos regular la visibilidad de sus miembros. Que un atributo sea visible significa que podemos acceder a él, tanto para leer como para modificarlo. Que un método sea visible significa que puede ser invocado.

Para que un miembro sea visible, es indispensable que su clase también lo sea. Es evidente que si no podemos acceder a una clase, no existe forma alguna de acceder a sus miembros.

Debemos destacar que cualquier miembro es siempre visible dentro de su propia clase, indistintamente del modificador de acceso que utilicemos. Es decir, desde dentro de la definición de una clase siempre tendremos acceso a todos sus atributos y podremos invocar cualquiera de sus métodos.

```
public class A { //clase pública
   int dato; //su ámbito es toda la clase:
   ... // el atributo dato es accesible desde cualquier lugar de A
}
```

Modificación de acceso

Visibilidad por defecto

Cuando queramos acceder a miembros de otra clase hay diversos grados de visibilidad. La visibilidad por defecto es aquella que se aplica a miembros declarados sin ningún modificador de acceso, como el atributo `dato` en el código anterior.

La visibilidad por defecto hace que un miembro sea visible desde las clases vecinas, pero invisible desde clases externas.

Modificación de acceso

Visibilidad por defecto

En nuestro ejemplo, el atributo `dato` será:

- Visible por clases vecinas, que pueden acceder tanto a la clase `A` como al atributo `dato`. Acceder a una clase significa utilizar sus miembros visibles, incluidos los constructores que permiten crear objetos.
- Invisible desde clases externas. Cualquier clase externa podrá acceder a la clase `A` —previa importación— por ser pública, pero no al atributo `dato`.

No olvidemos que la clase `A` se ha definido `public`, lo que permite que sea visible desde clases externas. Si `A` no fuera visible desde el exterior, tampoco lo serían sus miembros, sin importar el modificador utilizado en su declaración.

Modificación de acceso

Modificadores de acceso private y public

Con el modificador `private` obtenemos una visibilidad más restrictiva que por defecto, ya que impide el acceso incluso para las clases vecinas. Un miembro, ya sea un atributo o un método, declarado privado es invisible desde fuera de la clase.

En cambio, `public` hace que un miembro sea visible incluso desde clases externas previa importación. Otorga visibilidad total.

El uso de `private` está justificado cuando queremos controlar los cambios de un atributo o cuando deseamos que no se conozca directamente su valor, o bien cuando queremos que un método solo sea invocado desde otros métodos de la clase, pero no fuera de ella. El acceso a esos miembros privados deberá hacerse a través de algún método `public` de la misma clase.

Modificadores de acceso private y public

En el siguiente ejemplo se implementa la clase `Alumno` con los atributos nombre y nota media. Esta última será un atributo privado, ya que interesa controlar el rango de valores válidos, que estarán comprendidos entre 0 y 10, inclusive. El método público `asignaNota()` será el encargado de controlar el valor asignado a la nota.

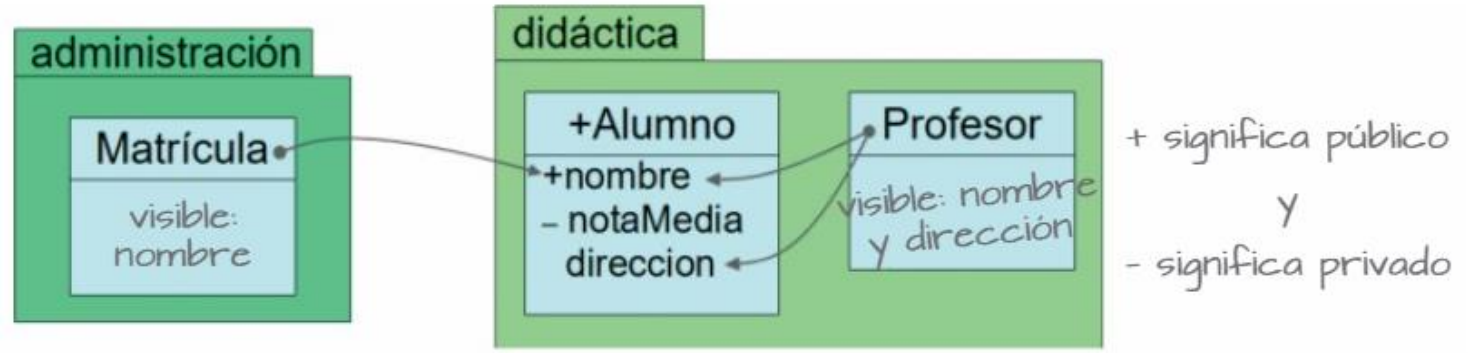
```
public class Alumno {
   public String nombre; //atributo público
   private double notaMedia; //atributo privado
   String direccion; //atributo con visibilidad por defecto

   public void asignaNota(double notaMedia) {
      //nos aseguramos de que esté en el rango 0..10
      if (notaMedia < 0 || notaMedia > 10) {
         System.out.println("Nota incorrecta");
      } else {
         this.notaMedia = notaMedia;
      }
   }
}
```

De este modo, solo es posible modificar la nota a través del método que la controla.

Modificación de acceso

Tabla Alcance de la visibilidad según el modificador de acceso

	Visible desde…		
	la propia clase	clases vecinas	clases externas
private	✓		
sin modificador	✓	✓	
public	✓	✓	✓



Ejercicio

Modificar la visibilidad de la clase CuentaCorriente para que sea visible desde clases externas y la visibilidad de sus atributos para que:

- Saldo no sea visible para otras clases.
- Nombre sea público desde cualquier clase.
- Dni solo sea visible desde clases vecinas.

Realizar un programa para comprobar la visibilidad de los atributos

Actividad

Ejercicio

Definir una clase que permita controlar un sintonizador digital de emisoras FM.

Concretamente se desea dotar al controlador de una interfaz que permita al usuario subir (up) y bajar (down) la frecuenciay mostrar la frecuencia sintonizada en un momento dado (display).

Suponemos que el rango de la frecuencia que debemos manejar oscila entre 80 Mhz y los 108 Mhz y que al inicio el se sintonice en 80 Mhz por defecto.

Si durante una operación de subida o bajada se sobrepasa uno de los dos límites, la frecuencia sintonizada debe pasar a ser el extremo contrario.

Escribir un método main para comprobar su funcionamiento.


Kahoot!

FIN