**APUNTES PROGRAMACIÓN UD3: Programación Orientada a Objetos (POO)**

**Clase POJO:** clase en la que se guardan los objetos principales del programa.

**OBJETOS POJO:** Objetos principales de la aplicación.

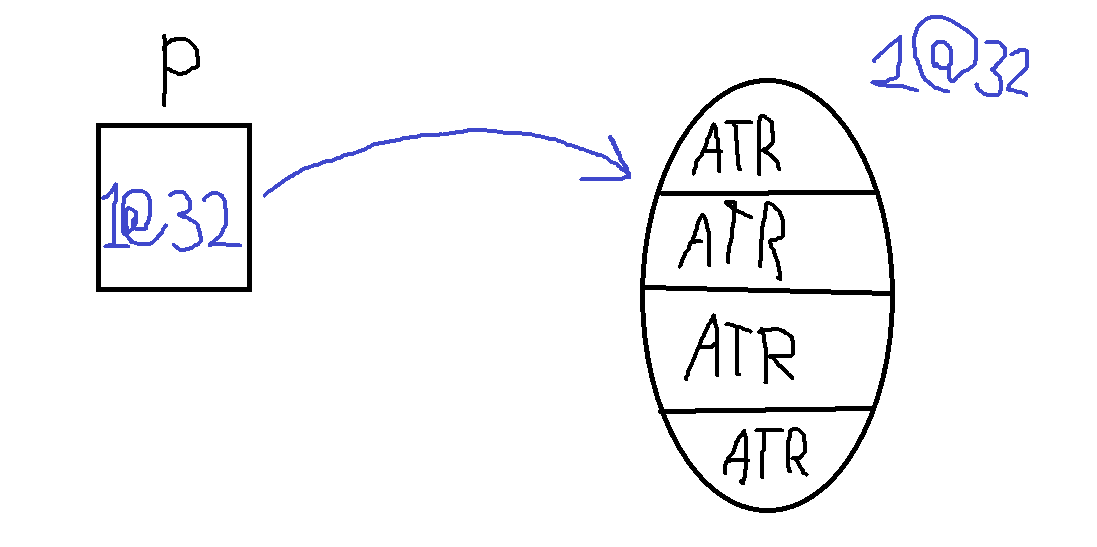
**INSTANCIAR UNA CLASE**

**NombreClase Identificador = new NombreConstructor ();**

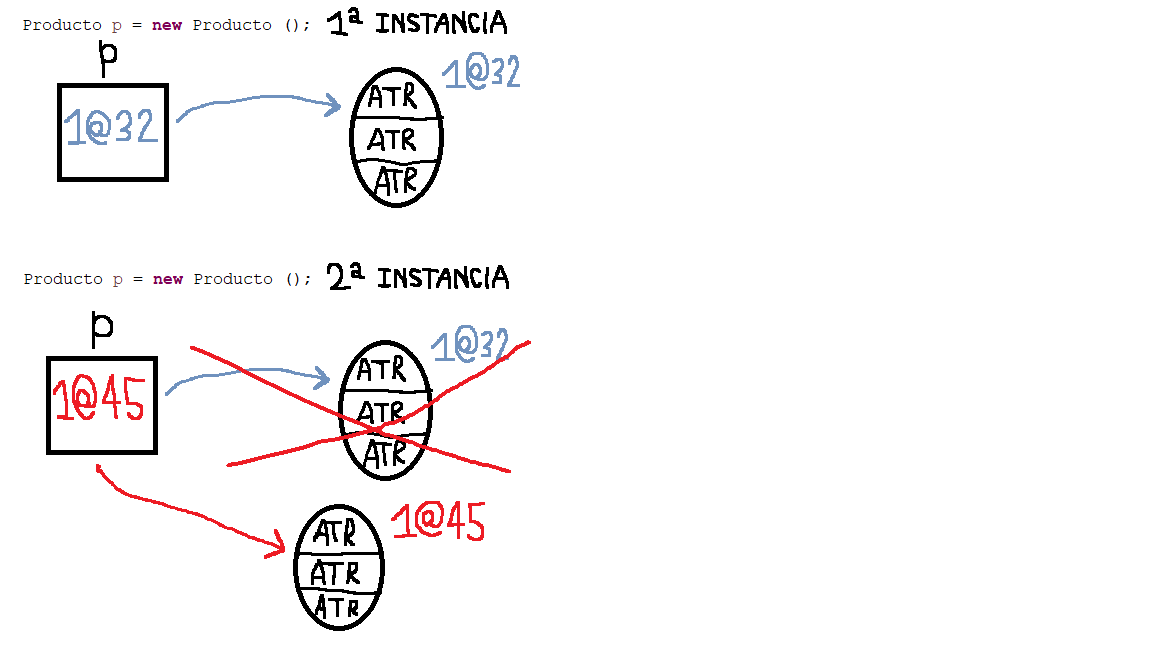
**Producto p = new Producto ();**

Una clase siempre se puede siempre se puede instanciar (aunque esta esté vacía), PERO NUNCA SE DEBE HACER, SE DEBE INSTANCIAR UNA CLASE LLENA.

Lo que ocurre al instanciar una clase es que se reserva un espacio en la memoria del ordenador al que se le llama como el identificador. Y en ese espacio se guarda la dirección de memoria de esa clase.



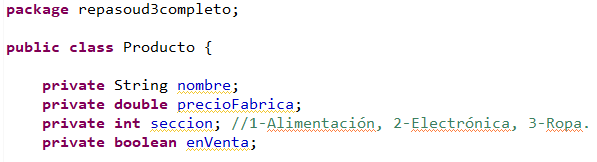
Cada vez que se instancia una clase 2 veces, la primera se de ellas se borra, ya que ese espacio de memoria reservado pasa a guardar la dirección de memoria de esa otra clase creada en la segunda instancia:



**ATRIBUTOS**

Los atributos son aquellas características que definen a un objeto.

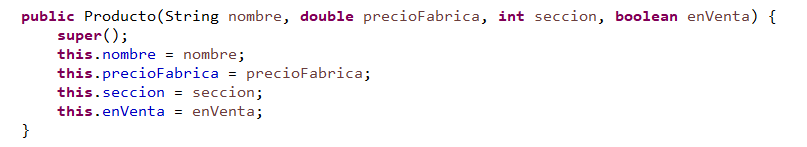
Se definen con: **private Tipo NombreAtributo;**

****

**CONSTRUCTORES**

Un constructor sirve para inicializar el objeto y establecer sus propiedades y valores predeterminados.

Tal que así:



**CREAR UN OBJETO**

A la hora de crear un objeto debemos tener en cuenta los atributos con los que se ha armado el constructor, ya que establecen las propiedades del objeto.

Los atributos puestos en un constructor se llaman **parámetros**.

Por ejemplo, vamos a crear un objeto “a cascaporra”. Las características que le vayamos a dar a dicho objeto tienen que corresponder con los parámetros impuestos en el constructor.

Tal que así:

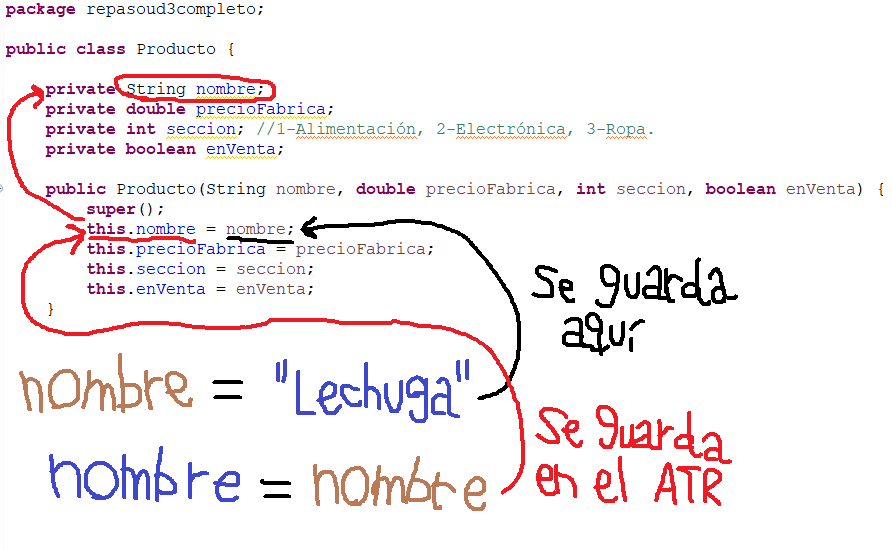


El constructor tiene impuestos como parámetros: **String nombre, double precioFabrica, int seccion** y **boolean enVenta**.

Y el objeto que acabamos de crear tiene características que corresponden con esos parámetros.

Esas “características” que le hemos impuesto al objeto se corresponden con los parámetros del constructor (las variables marrones) y estas se guardan en los atributos (las variables azules).

**“Lechuga”** se guardaría en la variable **nombre** y esta se guardaría como el atributo **nombre**.



Por lo tanto, ya tendríamos nuestro objeto creado con sus respectivos atributos.

**GETTERS Y SETTERS**

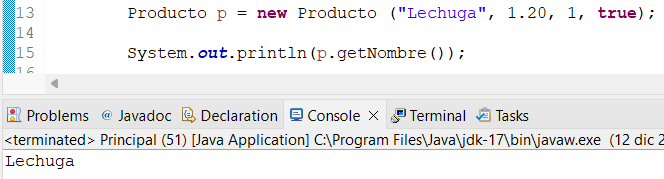
Los Getters y Setters nos permiten leer y escribir (respectivamente) los valores de nuestras variables privadas desde fuera de la clase donde fueron creadas.

**Con los Getters obtenemos los datos de los parámetros y con los Setters asignamos o cambiamos su valor.**

Los getters y setters son métodos y se nombran como uno.

En cuanto a la llamada, para llamar a un getter desde la clase principal se hace de esta forma: **Identificador.getNombreParametro();**

Tal que así:



Para llamar a los setters se hace así: **Identificador.setNombreParametro();**

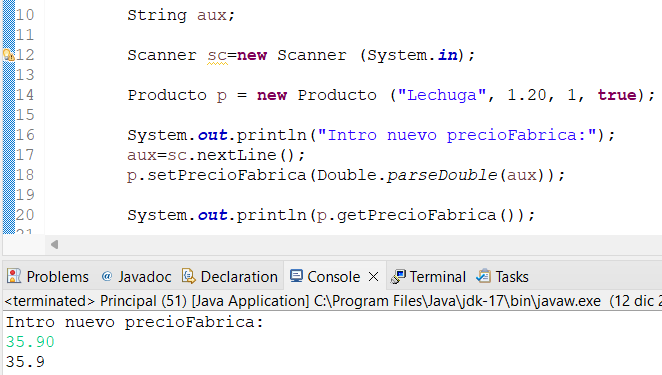
Como los setters cambian los valores de los parámetros solamente hay que darle a este un valor distinto y llamar al método setter.

Por ejemplo, el parámetro **precioFabrica** vale **1.20**, por tanto, si queremos cambiar su valor a 35.90 sólo tenemos que asignarle ese valor al método entre paréntesis de esta forma:

**p.setPrecioFabrica(35.90);**

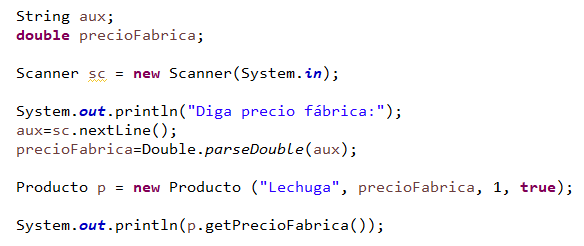
Ahora el atributo **precioFabrica** ha pasado de valer 1.20 a valer 35.90.

También se puede cambiar por teclado:



Y también cabe mencionar que se puede, en lugar de darle valores “a cascaporra” o por teclado a los parámetros, guardar el valor que queramos darle a ese parámetro en una variable y esa variable imponerla como parámetro del constructor.

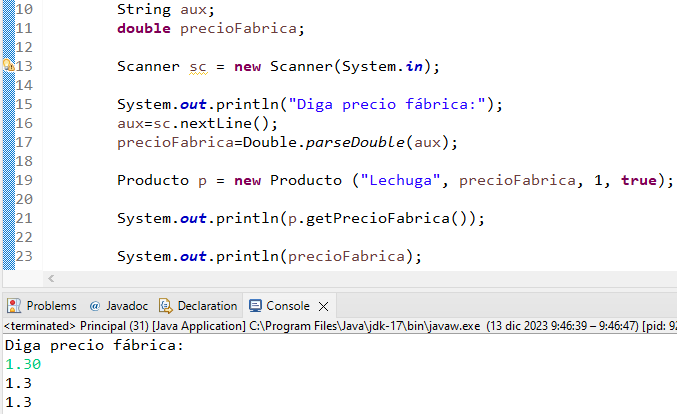
Tal que así:



En la imagen se puede ver que hemos creado en el main una variable llamada **double precioFabrica** y posteriormente le hemos asignado un valor por teclado. Seguido, hemos instanciado una clase y hemos creado un objeto. EL VALOR QUE LE HEMOS DADO AL PARÁMETRO **precioFabrica** es el de la variable **double precioFabrica** creada en el main.

**Ahora bien, hay que tener claro que la variable double precioFabrica y el parámetro precioFabrica NO SON LO MISMO.** Ya que la primera es una variable “chusquera” del main con el mismo nombre que el del parámetro y la segunda contiene el valor del atributo correspondiente del objeto.

Por ejemplo, si tenemos este caso:



Podemos observar que al imprimir el valor del parámetro **precioFabrica** y el valor de la variable **double precioFabrica** ambos tienen el mismo: **1.30**

Pero una vez que ya se le ha dado un valor a un parámetro SEA DE LA FORMA QUE SEA se debe utilizar un getter o un setter para operar con él, YA QUE TIENE EL PARÁMETRO CONTIENE EL VALOR DEL ATRIBUTO CORRESPONDIENTE DE ESE OBJETO. Si después de haberle dado un valor por variable al parámetro queremos volver a modificarlo y, en este caso, lo hacemos con la variable **double precioFabrica** de esta forma:

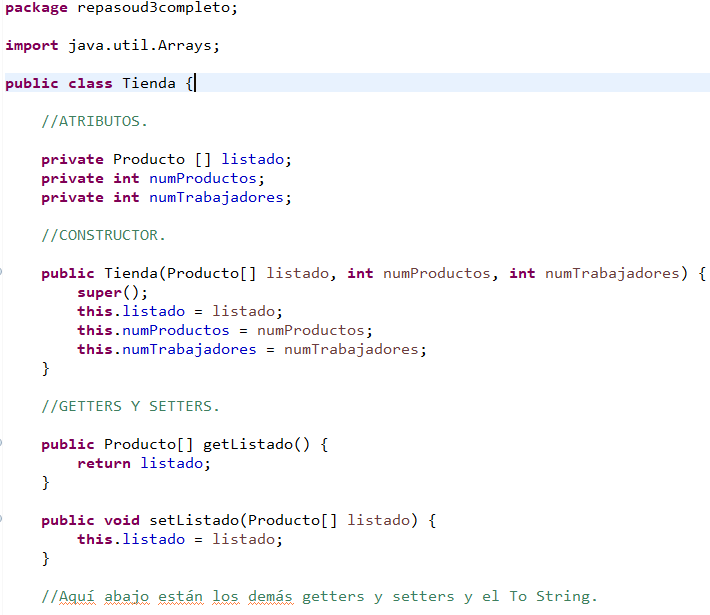
**precioFabrica = 3.40;**

**EL VALOR QUE SE MODIFICA ES EL DE LA VARIABLE double precioFabrica, NO EL DEL PARÁMETRO precioFabrica DEL OBJETO.**

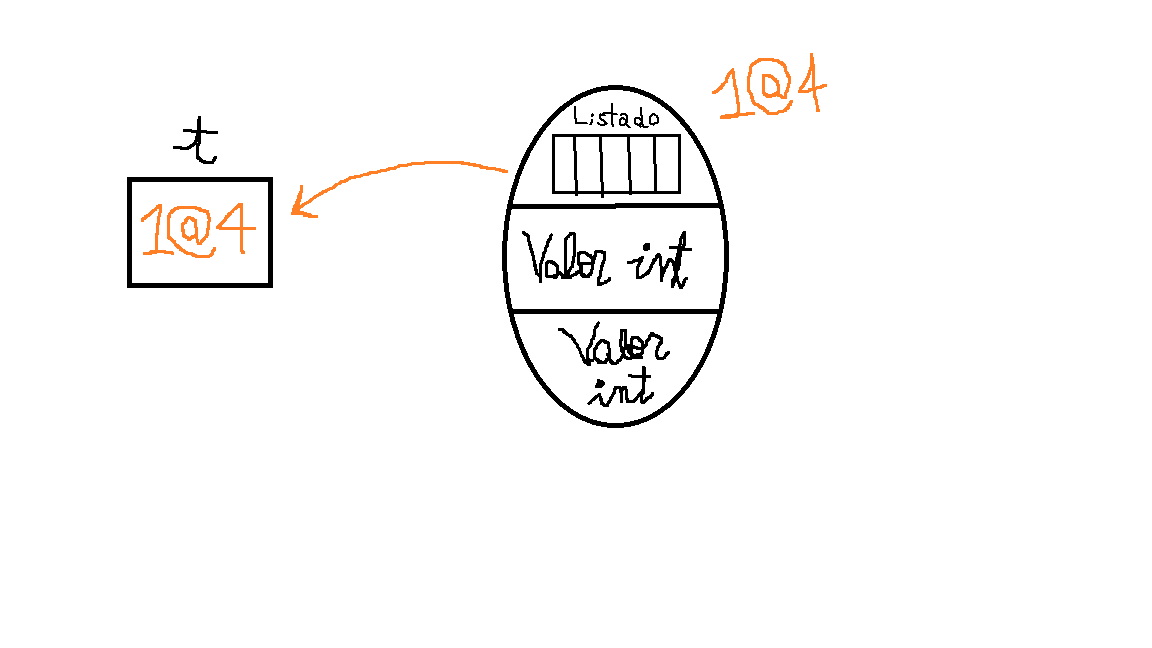
JFIFUIF

**ARRAY DE OBJETOS**

Hemos creado esta nueva clase llamada “Tienda” con los siguientes atributos:



Lo que hemos creado es esto:



La forma de crear un array como atributo es esta:

**private NombreClaseObjeto [] NombreAtributo;**

En este caso queremos crear un listado de productos en la clase tienda, por lo que el array sería el siguiente:

**private Producto [] listado;**

**CREAR UN MÉTODO**

**LOS NOMBRES DE LOS MÉTODOS DEBEN LLEVAR UN VERBO EN INFINITIVO SIN Ñ NI TILDES, Y QUE INDIQUE QUÉ DEBE HACER ESE MÉTODO. ADEMÁS, NO SE PUEDEN AÑADIR ESPACIOS EN EL NOMBRE Y LA PRIMERA PALABRA DEBE EMPEZAR POR MINÚSCULA Y LAS DEMÁS POR MAYÚSCULA. EN SERIO SIEMPRE.**

La forma de agregar un método es la siguiente:

**public TipoDeDatoADevolver NombreMetodo () {**

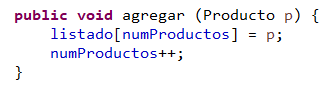
**}**

Ahora que tenemos la clase “Tienda” y el array de productos vamos a crear un método que agregue productos a ese array.

Tal que así: **public void agregar () {}**

Bien, una vez creado el método necesitamos rellenarlo. Para ello, vamos a nombrar al atributo **listado**. Como vamos agregar productos al array, este necesita un índice y para ello tenemos el atributo **numProducto** que tendrá esa función. Además de ello, necesitamos que el índice aumente en 1 tras añadir un producto para que se puedan seguir agregando más. Y ya, finalmente, en los paréntesis del nombre del método escribimos las variables con las que está operando ese método. En este caso, como agregamos productos, hacemos referencia a la instancia de dicha clase.

Dicho todo, el método quedaría de esta forma:



Ahora que el método está completo vamos a ponerlo en práctica.

* Primero, nombramos la clase “Producto” en el main.
* Después, he creado variables que se llaman de igual forma que los parámetros y cuyo valor se pide por teclado.
* Una vez que las variables ya tienen el valor asignado, instanciamos el método con los parámetros del constructor.

Tras este paso, ya tenemos el objeto “Producto” creado, ahora solamente queda meterlo en la lista.

Como la lista es un atributo del objeto “Tienda”, necesitamos instanciar su clase.

Yo lo he hecho de esta forma:

**Tienda t = new Tienda (lista, tam, 3);**

Los valores del parámetro **listado** se asignan con un array inicializado en el main. En este caso el array **lista**:

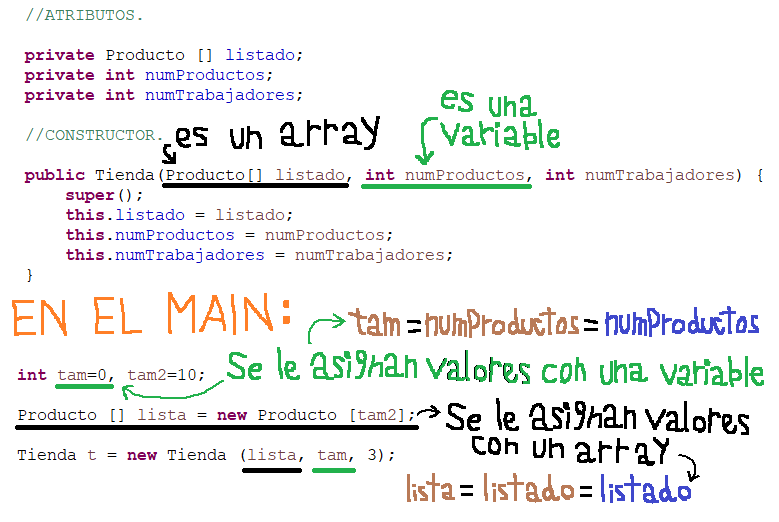
**Producto [] lista = new Producto [tam2];**

(tam2 es una variable que le da el tamaño al array **lista**).

El valor del parámetro **numProducto** es asignado por la variable **tam**. Como este parámetro actúa como índice del array, esta variable vale **0**.

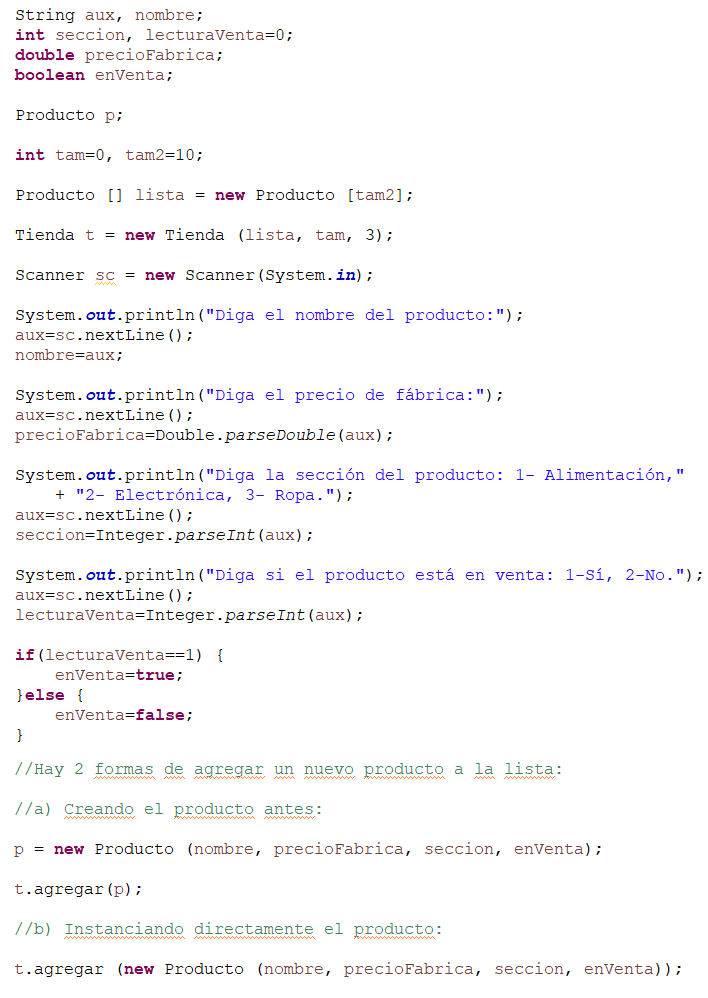
**SI EL PARÁMETRO DE UN CONSTRUCTOR ES UNA VARIABLE, SU VALOR SE PUEDE ASIGNAR CON OTRA VARIABLE EN EL MAIN.**

**SI EL PARÁMETRO DE UN CONSTRUCTOR ES UN ARRAY, SU VALOR SE DEBE ASIGNAR CON OTRO ARRAY EN EL MAIN.**

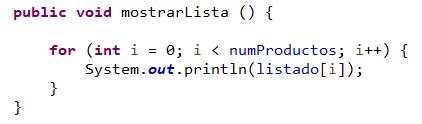
****

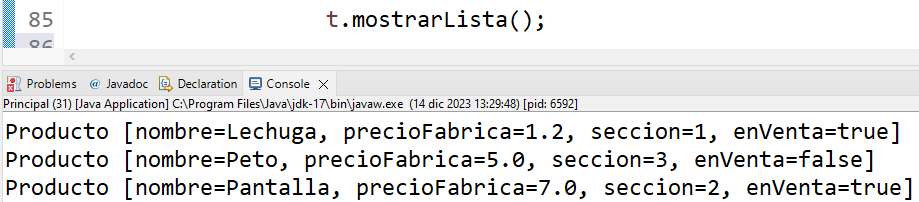
Y por último, el valor del parámetro **numTrabajadores** es **3**, ya que es un valor introducido “a cascaporra”.

Listo, pues ya estaría, ahora solo queda construir el programa. Tal y como yo lo he hecho, quedaría tal que así:



**MÉTODO MOSTRAR**





Los productos que aparecen en la consola del programa son productos que he agregado a la lista antes de mostrarla.

**MÉTODO BUSCAR 1 (Buscar y devolver un array)**

Imagínate que alguien al usar tu programa quiere buscar productos específicos ya introducidos en la lista. Para estos casos necesitamos un método “buscar” que recorra la lista y seleccione los productos que se hayan pedido.

En nuestro caso, alguien quiere buscar los productos por sección (1- Alimentación, 2- Electrónica, 3- Ropa).

Para ello vamos a crear un método que va devolver un array de todos los productos que tengamos agregados en nuestra lista.

Este método va a devolver un array de productos de una determinada sección y se va llamar “buscarPorSeccion)”:

**public Producto [] buscarPorSeccion (int seccion) {}**

