

Programación Orientada a Objetos

Material de lectura

INTRODUCCIÓN A OBJETOS

## 

## 

## 

## 

## 

## OBJETIVOS DE LA GUÍA

En esta guía aprenderemos a:

* Comprender el paradigma orientado a objetos
* Crear clases
* Definir atributos de clases

## 

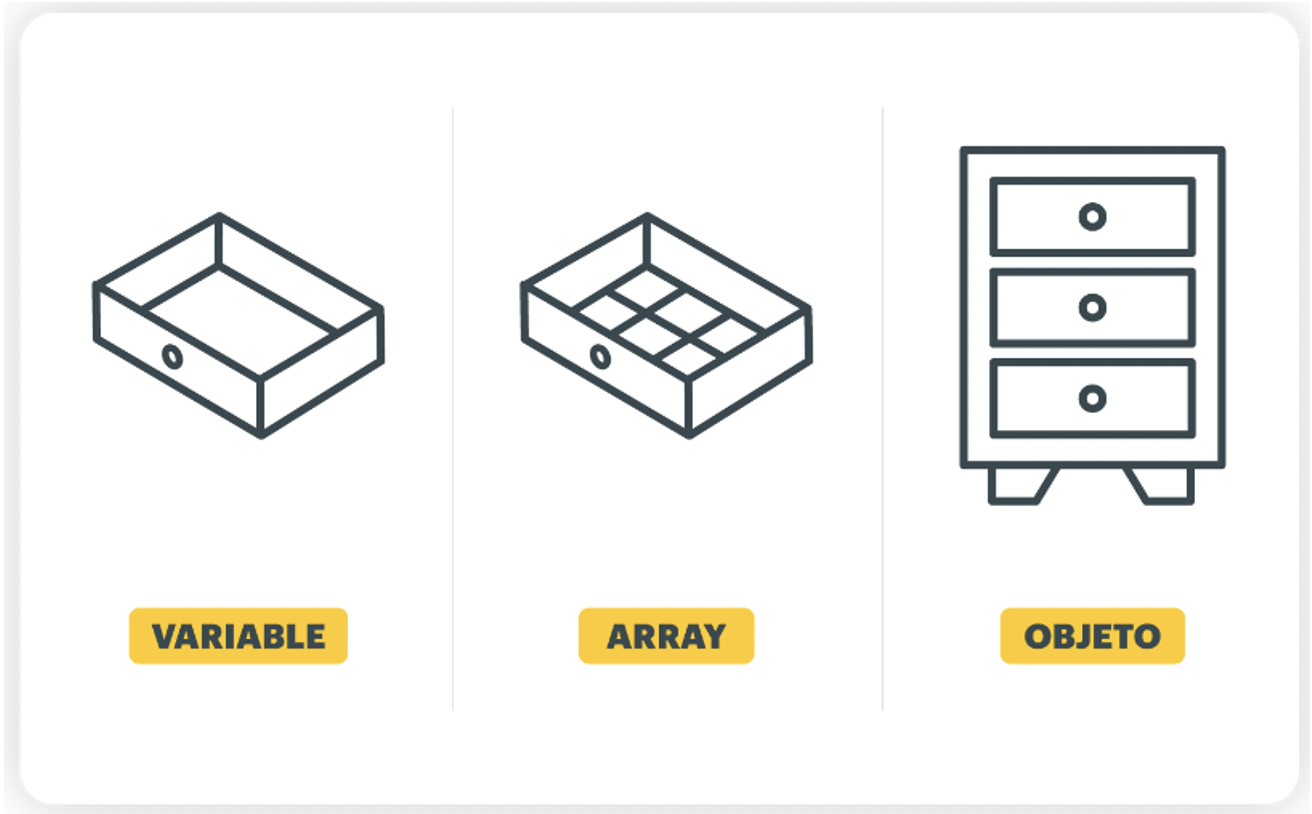
# 

# ¿Que es un paradigma de programación?

Un paradigma de programación es una manera o estilo de programación. Existen diferentes formas de diseñar un programa y varios modos de trabajar para obtener los resultados que necesitan los programadores. Por lo que un paradigma de programación se trata de un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para resolver problemas.

# ¿Qué es la programación orientada a objetos?

Antes de sumergirnos en el concepto hagamos una revisión de cómo evolucionó nuestra manera de pensar y alojar información en la programación. En un primer momento, aprendimos las variables. Dijimos que las variables eran como cajones donde podíamos guardar sólo un dato de cierto tipo. Luego estudiamos los arrays, que nos permitían alojar varios datos a la vez, como si fuera un cajón con compartimentos; siempre y cuando estos datos fueran del mismo tipo. La programación orientada a objetos nos trae una nueva forma de almacenar información: OBJETOS. Pensaremos los objetos como un fichero o cajonera. Iremos profundizando este ejemplo a medida que avancemos en esta guía.



La **Programación Orientada a Objetos (POO)** es un paradigma de programación, es decir, un modelo o un estilo de programación que se basa en el concepto de **clases y objetos**. Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de código (clases) para crear instancias individuales de objetos.

Con el paradigma de Programación Orientado a Objetos lo que buscamos es dejar de centrarnos en la lógica pura de los programas, para empezar a pensar en objetos, lo que constituye la base de este paradigma.

La programación orientada a objetos se enfoca en los **objetos, sus atributos y las interacciones** que se producen entre ellos para diseñar programas.

Un programa orientado a objetos es, esencialmente, un **conjunto de objetos** que se crean para que **interaccionen entre sí** y dejan de existir cuando ya no son útiles durante la ejecución de un programa. Un programa puede llegar a ser muy complejo. La complejidad es más manejable cuando se descompone y se organiza en partes pequeñas y simples, los objetos.

## ¿Por qué POO?

La Programación Orientada a objetos permite que el código sea reutilizable, organizado y fácil de mantener. Sigue el principio de desarrollo de *software* utilizado por muchos programadores **DRY *(Don’t Repeat Yourself)****,* para evitar duplicar el código y crear de esta manera programas eficientes. Además, evita el acceso no deseado a los datos o la exposición de código propietario mediante la encapsulación y la abstracción, de la que hablaremos en detalle más adelante.



## Pero entonces, ¿Qué es un objeto?

Un objeto en la programación es la forma que tenemos de llevar a código, la representación más fidedigna posible de un objeto de la vida real. Un objeto tendrá características (atributos) que son comunes a todos ellos, pero pueden variar entre sí.

Volvamos al ejemplo del fichero o cajonera. El objeto es la cajonera, cada cajón representará un atributo. Una vez creado el molde para crear este objeto cajonera, cada uno de sus cajones alojará un tipo de dato que será el mismo tipo pero la información que se aloje dentro será diferente. En el primer cajón debemos guardar remeras, pero estas pueden ser rojas en una cajonera o verdes en otra.

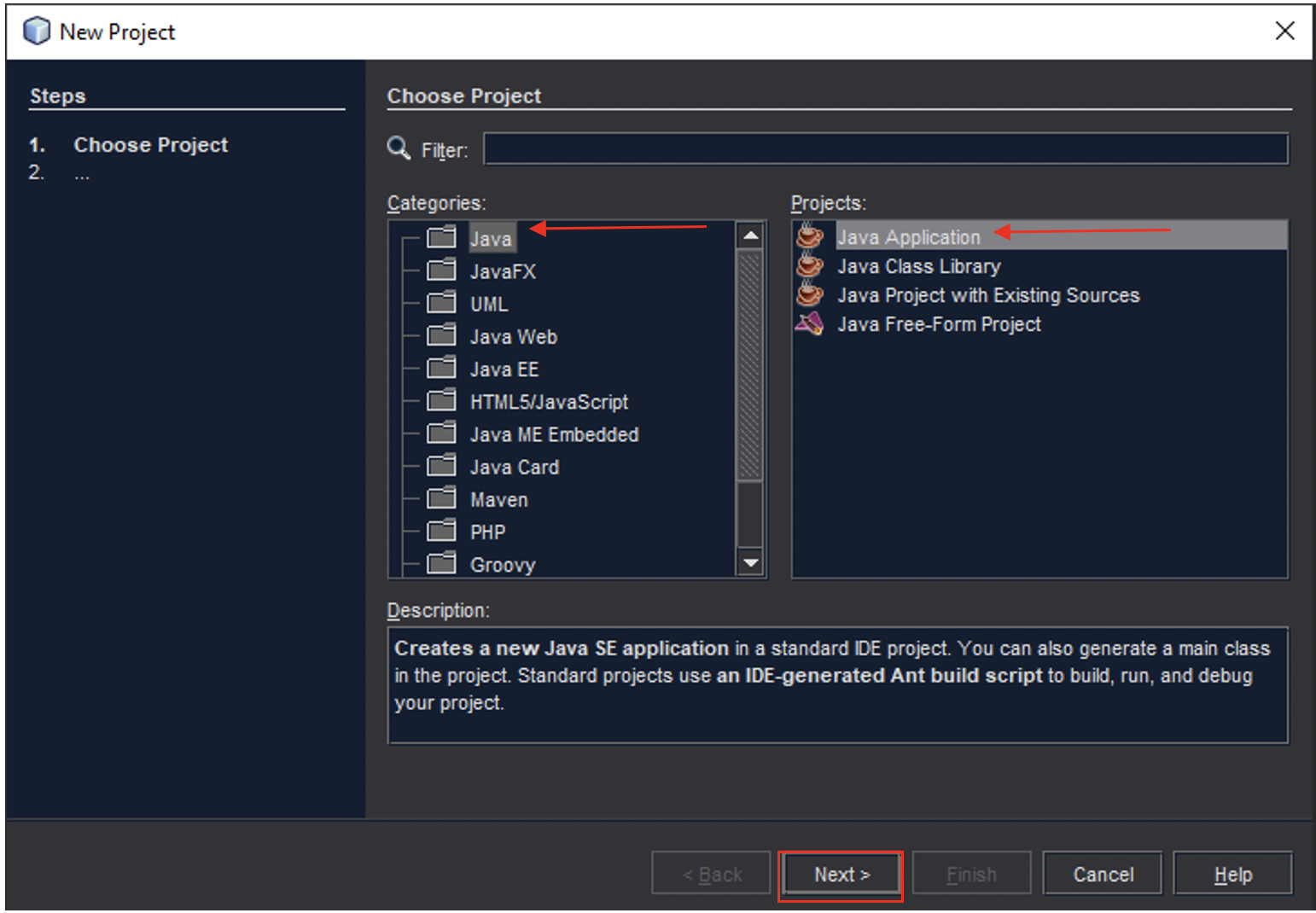
Pensemos también a las personas como objetos. Tenemos ciertas características que son uniformes a todos. Todos tenemos un nombre, un número de identificación, una fecha de nacimiento. Estas características variarían de una persona a otra, pero al compartir esta información podemos elaborar código que represente y nos permita identificar a una o varias personas.

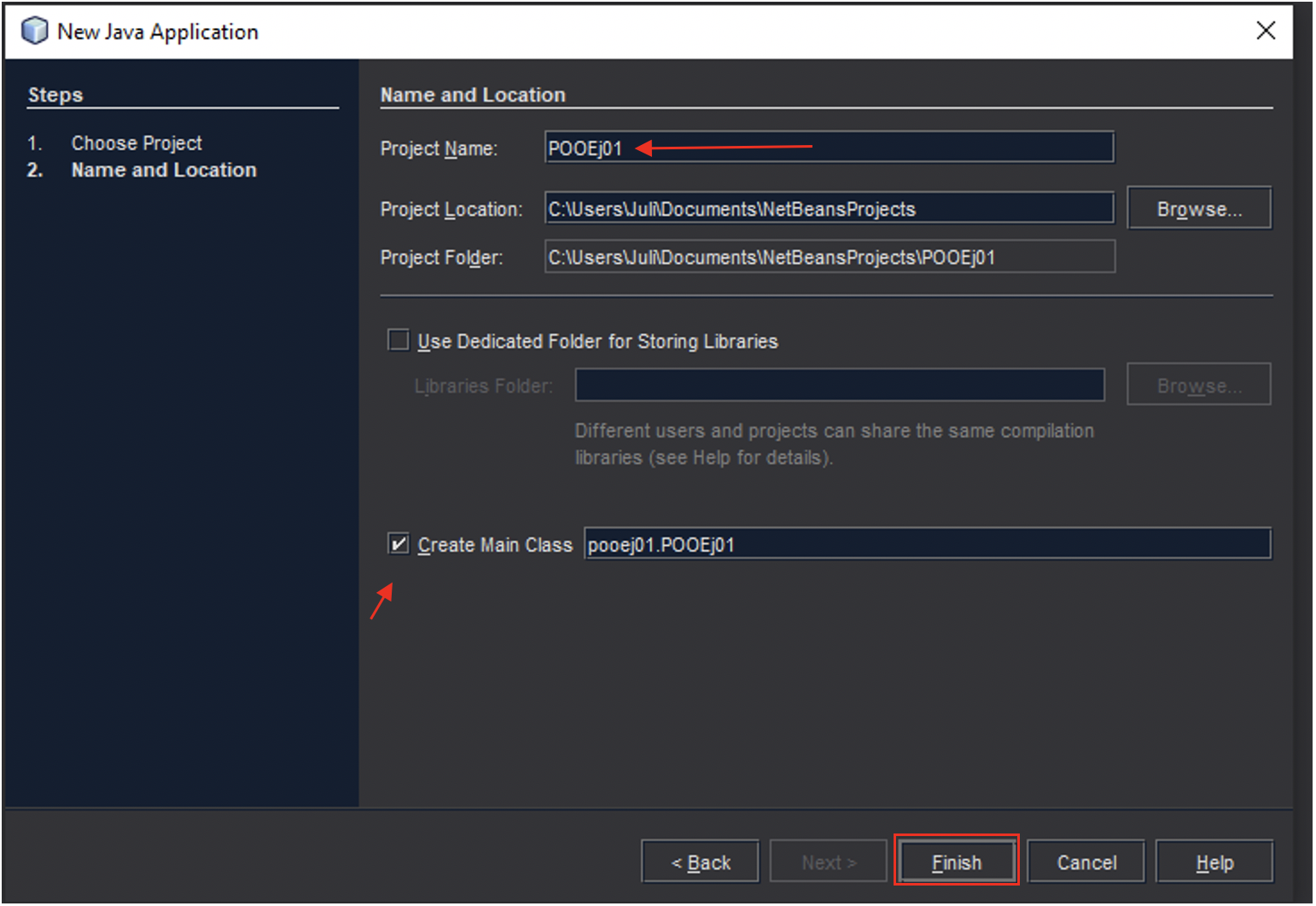


## ¿Cómo creamos objetos?

Para llevar cierto orden, por cada uno de los ejercicios de esta guía **crearemos un nuevo proyecto** en NetBeans.

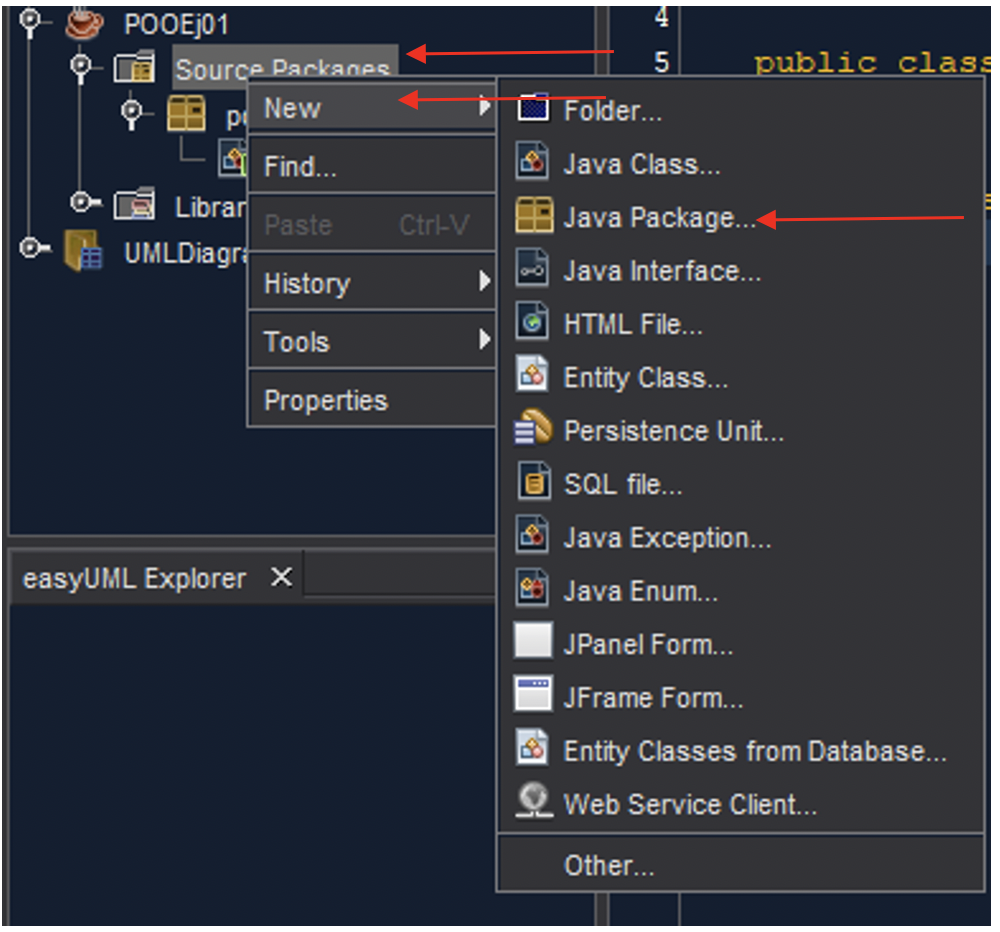
Primero crearemos el proyecto:

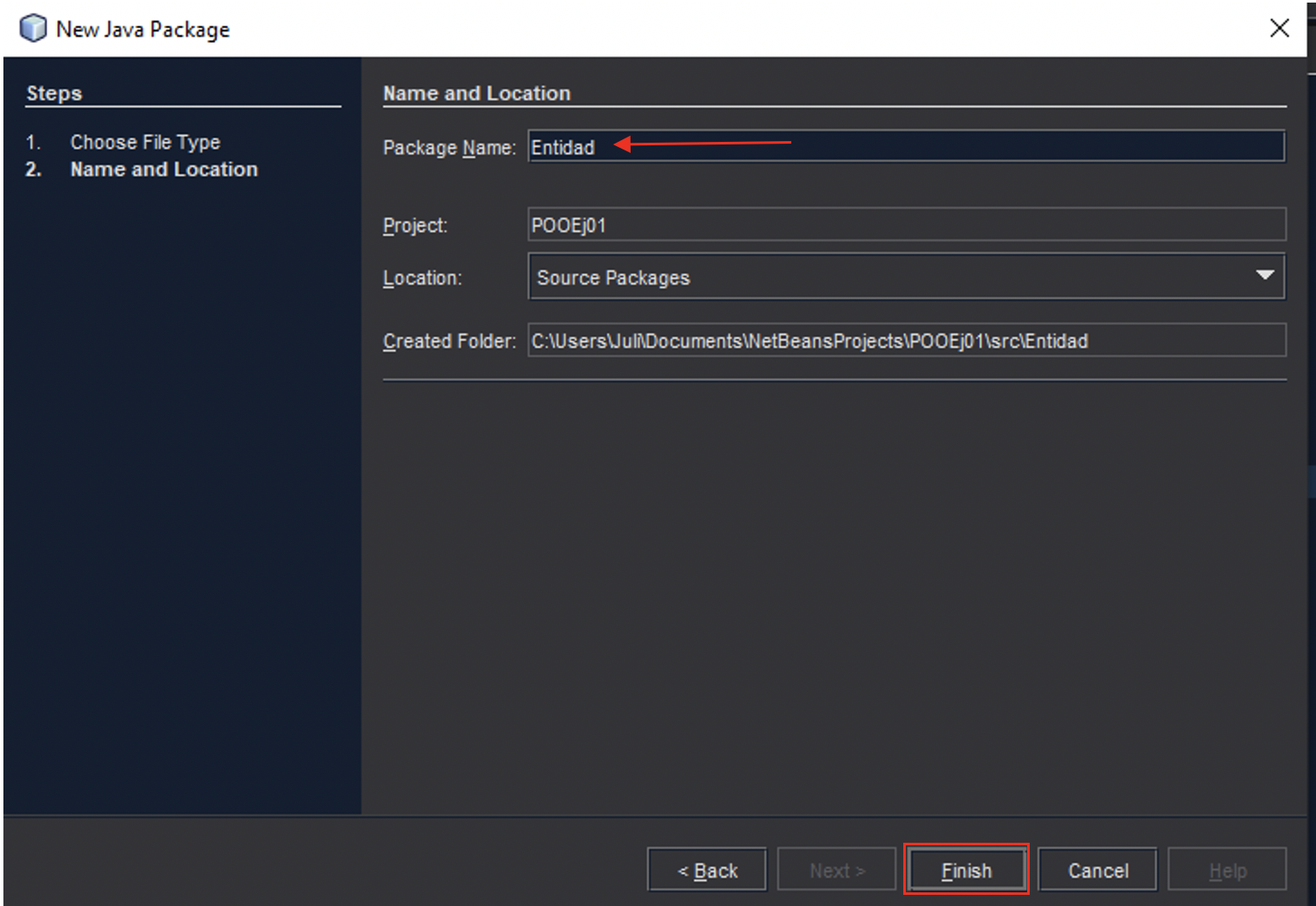


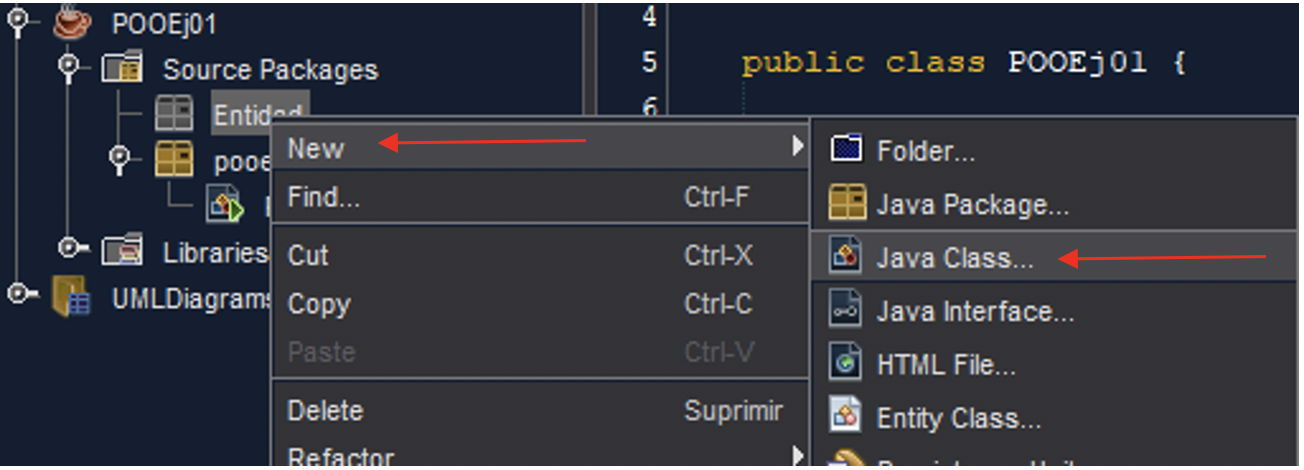


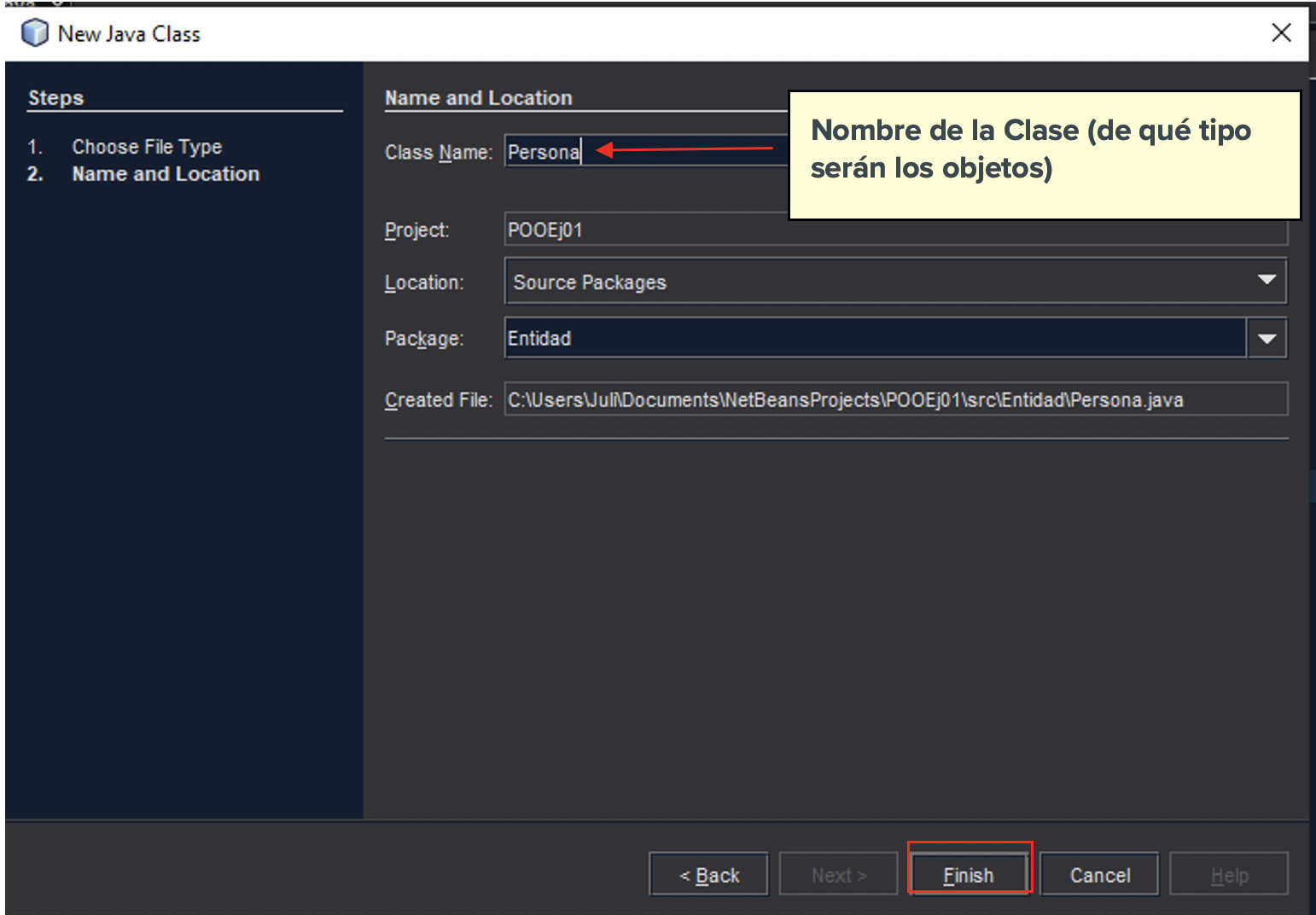
Luego, necesitamos crear una clase. Una **clase es un molde** para crear múltiples objetos que encapsula datos y comportamiento. Una clase es una combinación específica de atributos y métodos y puede considerarse un tipo de dato de cualquier tipo no primitivo.

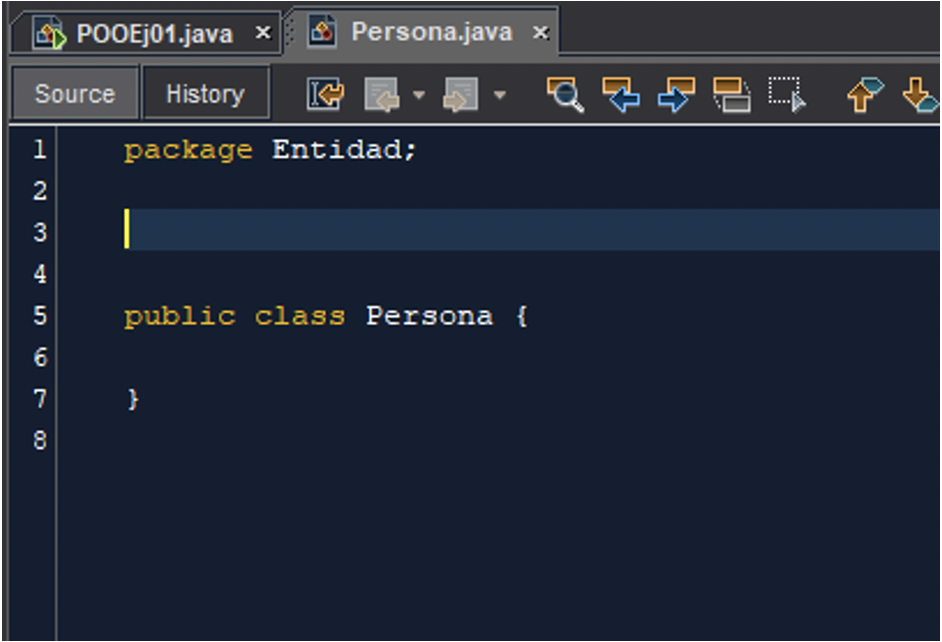
Las clases deberán guardarse en un paquete llamado Entidad:











Así, una clase es una especie de plantilla o prototipo de objetos: define los atributos que componen ese tipo de objetos y los métodos que pueden emplearse para trabajar con esos objetos. En su forma más simple, una clase se define por la palabra reservada class seguida del nombre de la clase. El nombre de la clase debe empezar por mayúscula.

Si el nombre es compuesto, entonces cada palabra debe empezar por mayúscula. La definición de la clase se pone entre las llaves de apertura y cierre.

public class NombreClase {

// atributos

// constructores

// métodos propios

}



## Ejercicio persona

Es tu turno, crea tu propio proyecto con la clase Persona.

# Estado y Comportamiento

En términos más generales, **un objeto es una abstracción conceptual del mundo real** que se puede **traducir a un lenguaje de programación** orientado a objetos. Los objetos del mundo real comparten dos características: Todos poseen **estado y comportamiento**. Por ejemplo, el perro tiene estado (color, nombre, raza, edad) y el comportamiento (ladrar, caminar, comer, acostarse, mover la cola). Por lo tanto, un estado permite informar cuáles son las características del objeto y lo que este representa, y el comportamiento, consiste en decir lo que sabe hacer.

El estado de un objeto es una **lista de variables conocidas como sus atributos**, cuyos valores representan el estado que caracteriza al objeto.

El comportamiento es una **lista de métodos, procedimientos, funciones u operaciones** que un objeto puede ejecutar a solicitud de otros objetos. Los objetos también se conocen como instancias.

# Elementos de una Clase

Una clase describe un tipo de objetos con características comunes. Es necesario definir la información que almacena el objeto y su comportamiento.

## ¿Qué son los atributos?

Los atributos son **características comunes** a todos los objetos. Son los “espacios” donde **alojaremos información** que cambiará en cada objeto pero que corresponda a una descripción del mismo. Son los “cajones” de la cajonera.

El estado o información de un objeto se almacena en atributos. Los atributos pueden ser de tipos primitivos de Java (descriptos en la guía Intro Java) o del tipo de otros objetos. La declaración de un atributo de un objeto tiene la siguiente forma:

<modificador>\* <tipo> <nombre>

· **<modificador>:** si bien hay varios valores posibles para el <modificador>, por el momento solo usaremos modificadores de visibilidad: public, protected, private.

· **<tipo>:** indica la clase a la que pertenece el atributo definido.

· **<nombre>:** puede ser cualquier identificador válido y denomina el atributo que está siendo declarado.



| public class Persona{  public String nombre; } |
| --- |

Estos atributos irán al principio de la clase.



## Ejercicio persona – atributos

Seguiremos trabajando sobre la clase Persona que creamos y ahora deberás sumarle 3 atributos que creas pertinentes.

## ¿Qué son los constructores?

Los constructores con métodos propios del objeto que nos permiten CREARLO. A la creación de un objeto se le denomina INSTANCIACIÓN.

Además de definir los atributos de un objeto, es necesario definir los métodos que determinan su comportamiento. Toda clase debe definir un método especial denominado constructor para instanciar los objetos de la clase. Este método tiene el mismo nombre de la clase. La declaración básica toma la siguiente forma:

[<modificador>] <nombre de clase> ( <argumento>\* ) {

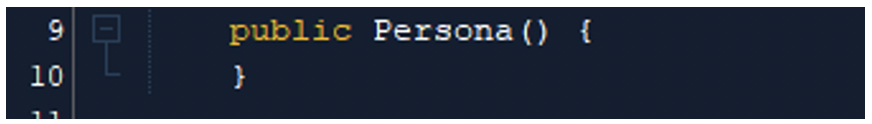
<sentencia>\*

}

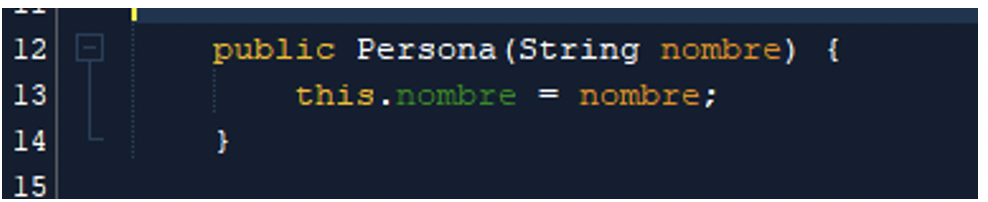
* **<nombre de clase>:** El nombre del constructor debe ser siempre el mismo que el de la clase.
* **<modificador>:** Actualmente, los únicos modificadores válidos para los constructores son public, protected y private.
* <**argumentos>:** es una lista de parámetros que tiene la misma función que en los métodos.



Este es un constructor por defecto



Éste es un constructor con parámetros



El método constructor se ejecuta cada vez que se instancia un objeto de la clase. Este método se utiliza para **inicializar** los atributos del objeto que se instancia.

Para diferenciar entre los atributos del objeto y los identificadores de los parámetros del método constructor, se utiliza la palabra this. De esta forma, los parámetros del método pueden tener el mismo nombre que los atributos de la clase.

La instanciación de un objeto consiste en asignar un espacio de memoria al que se hace referencia con el nombre del objeto. Los identificadores de los objetos permiten acceder a los valores almacenados en cada objeto.



## **El constructor por defecto**

Cada clase tiene al menos un constructor. Si no se escribe un constructor, el lenguaje de programación Java le provee uno por defecto. Este constructor no posee argumentos y tiene un cuerpo vacío. Si se define un constructor que no sea vacío, el constructor por defecto se pierde, salvo que creemos un nuevo constructor vacío.

Una vez que se ha declarado una clase, se pueden crear objetos a partir de ella. A la creación de un objeto se le denomina **instanciación**. Por esta razón que se dice que un objeto es una instancia de una clase y el término instancia y objeto se utilizan indistintamente. Para crear objetos, basta con declarar una variable de alguno de los tipos de las clases definidas.

NombreClase nombreObjeto;

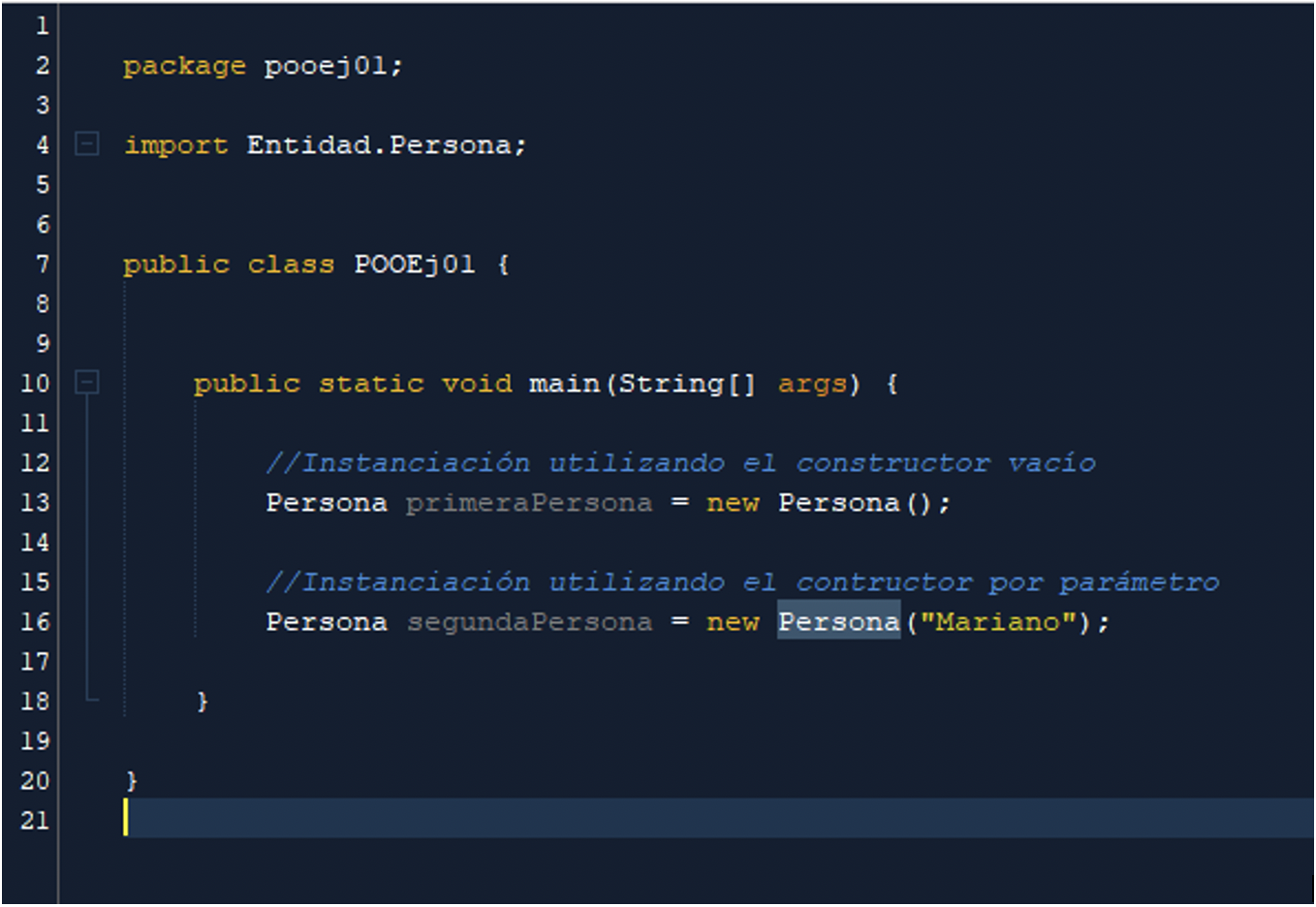
Para crear el objeto y asignar un espacio de memoria es necesario realizar la instanciación con el operador new. El operador new instancia el objeto y reserva espacio en memoria para los atributos y devuelve una referencia que se guarda en la variable.

nombreObjeto = new nombreClase();

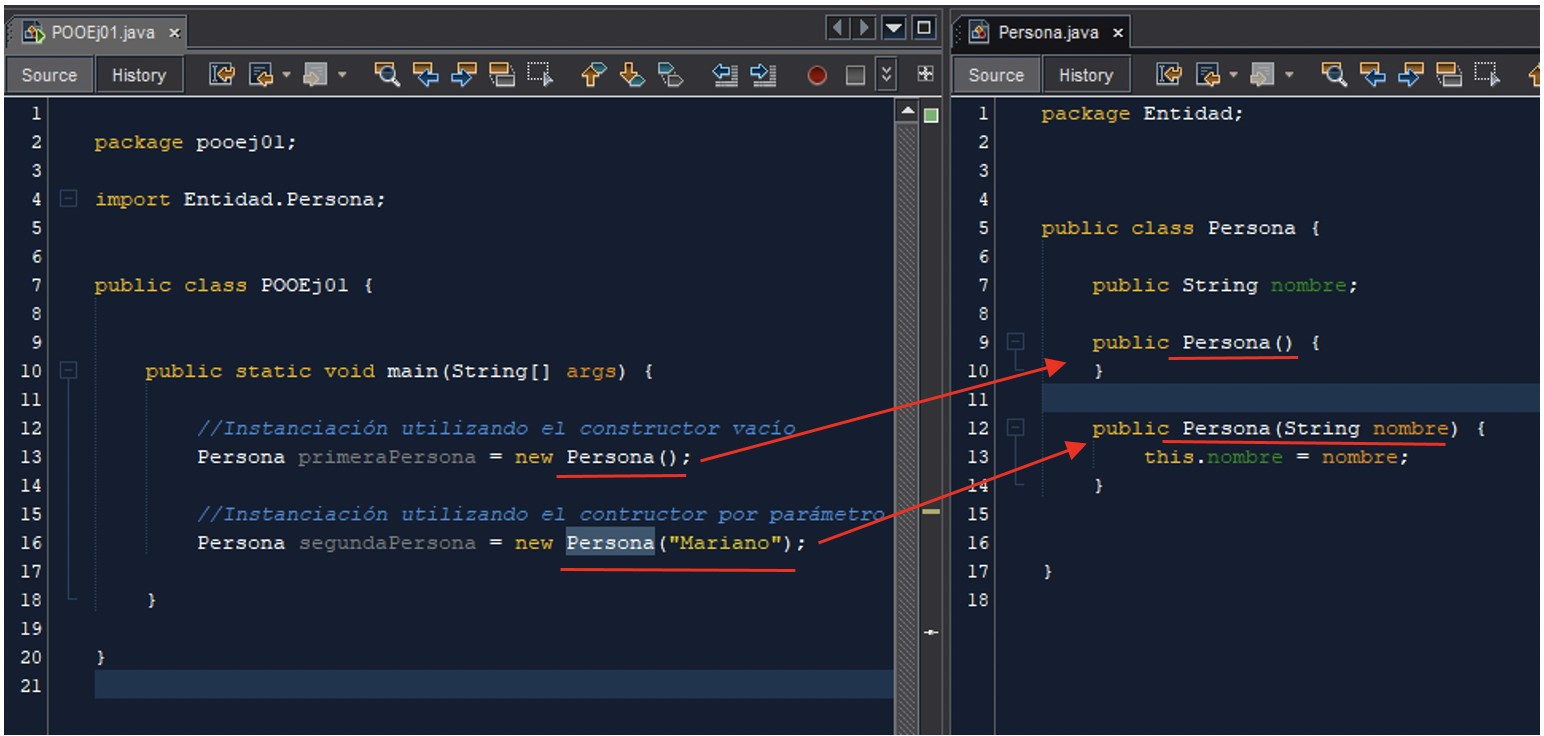
Tanto la declaración de un objeto como la asignación del espacio de memoria se pueden realizar en un solo paso:NombreClase nombreObjeto = new NombreClase();

A partir de este momento los objetos ya pueden ser referenciados por su nombre.





Instanciamos dos objetos en el main del tipo Persona



## **¿Cómo se crean los constructores?**

Dentro de la clase, para agregar automáticamente los constructores podemos presionar el botón derecho > Insert Code > Constructor y podemos no seleccionar ningún atributo o seleccionar todos los atributos.



## **Ejercicio persona – constructores**

Volveremos a usar la clase Persona y vamos a crear objetos en tu Main utilizando los constructores.



# **Abstracción y Encapsulamiento**

La abstracción es la habilidad de ignorar los detalles de las partes para enfocar la atención en un nivel más alto de un problema. El encapsulamiento sucede cuando algo es envuelto en una capa protectora. Cuando el encapsulamiento se aplica a los objetos, significa que los datos del objeto están protegidos, “ocultos” dentro del objeto. Con los datos ocultos, ¿cómo puede el resto del programa acceder a ellos? (El acceso a los datos de un objeto se refiere a leerlos o modificarlos.) El resto del programa no puede acceder de manera directa a los datos de un objeto; lo tiene que hacer con ayuda de los métodos del objeto. Al hecho de proteger los datos o atributos con los métodos se denomina encapsulamiento.

## **Abstracción**

La abstracción es la propiedad que considera los aspectos más significativos o notables de un problema y expresa una solución en esos términos. La abstracción posee diversos grados o niveles de abstracción, los cuales ayudan a estructurar la complejidad intrínseca que poseen los sistemas del mundo real. La abstracción encarada desde el punto de vista de la programación orientada a objetos es el mecanismo por el cual se proveen los límites conceptuales de los objetos y se expresan sus características esenciales, dejando de lado sus características no esenciales. Si un objeto tiene más características de las necesarias los mismos resultan difíciles de usar, modificar, construir y comprender. En el análisis hay que concentrarse en ¿Qué hace? y no en ¿Cómo lo hace?

## **Encapsulamiento**

La encapsulación o encapsulamiento significa reunir en una cierta estructura a todos los elementos que a un cierto nivel de abstracción se pueden considerar pertenecientes a una misma entidad, y es el proceso de agrupamiento de datos y operaciones relacionadas bajo una misma unidad de programación, lo que permite aumentar la cohesión de los componentes del sistema.

El encapsulamiento **oculta lo que hace un objeto** de lo que hacen otros objetos y del mundo exterior por lo que se denomina también ocultación de datos. Un objeto tiene que presentar “una cara” al mundo exterior de modo que se puedan iniciar sus operaciones.

Los métodos operan sobre el estado interno de un objeto y sirven como el mecanismo primario de comunicación entre objetos. Ocultar el estado interno y hacer que toda interacción sea a través de los métodos del objeto es un mecanismo conocido como encapsulación de datos.

## **Modificadores de Acceso**

Para lograr el uso correcto del encapsulamiento vamos a utilizar los modificadores de acceso, estos, van a dejarnos elegir como se accede a los datos y a través de que se accede a dichos datos. Todas las clases poseen diferentes niveles de acceso en función del modificador de acceso (visibilidad). A continuación, se detallan los niveles de acceso con sus símbolos correspondientes:

· **Public:** Este modificador permite acceder **a los elementos desde cualquier clase**, independientemente de que esta pertenezca o no al paquete en que se encuentra el elemento.

· **Private:** Es el modificador más restrictivo y especifica que los elementos que lo utilizan **sólo pueden ser accedidos desde la clase en la que se encuentran**. Este modificador sólo puede utilizarse sobre los atributos de una clase y sobre interfaces y clases internas, no sobre clases o interfaces de primer nivel, dado que esto no tendría sentido. Es importante destacar también que el modificador private convierte los elementos en privados para otras clases, no para otras instancias de la clase. Es decir, un objeto de una determinada clase puede acceder a los atributos privados de otro objeto de la misma clase.

· **Protected:** Este modificador indica que los elementos **sólo pueden ser accedidos desde su mismo paquete y desde cualquier clase que extienda la clase** en que se encuentra, independientemente de si esta se encuentra en el mismo paquete o no. Este modificador, como private, no tiene sentido a nivel de clases o interfaces no internas.

Si no especificamos ningún modificador de acceso se utiliza el nivel de acceso por defecto (Default), que consiste en que el elemento puede ser accedido sólo desde las clases que pertenezcan al mismo paquete. Los distintos modificadores de acceso quedan resumidos en la siguiente tabla:

| **Visibilidad** | **Public** | **Private** | **Protected** | **Default** |
| --- | --- | --- | --- | --- |
| Desde la misma Clase | SI | SI | SI | SI |
| Desde cualquier Clase del mismo Paquete | SI | NO | SI | SI |
| Desde una Subclase del mismo Paquete | SI | NO | SI | SI |
| Desde una Subclase fuera del mismo Paquete | SI | NO | SI, a través de la herencia | NO |
| Desde cualquier Clase fuera del Paquete | SI | NO | NO | NO |

## 

## 

## **Métodos propios**

Los métodos son funciones que determinan el comportamiento de los objetos. Un objeto se comporta de una u otra forma dependiendo de los métodos de la clase a la que pertenece. Todos los objetos de una misma clase tienen los mismos métodos y el mismo comportamiento. Para definir los métodos, el lenguaje de programación Java toma la siguiente forma básica:

<modificador>\* <tipo de retorno> <nombre> ( <argumento>>\* ) {

<sentencias>\*

return valorRetorno;

}

* **<modificador>:** el segmento es opcional y puede contener varios modificadores diferentes incluyendo a public, protected y private. Aunque no está limitado a estos.
* **<tipo de retorno>:** el tipo de retorno indica el tipo de valor devuelto por el método. Si el método no devuelve un valor, debe ser declarado void. La tecnología Java es rigurosa acerca de los valores de retorno. Si el tipo de retorno en la declaración del método es un int, por ejemplo, el método debe devolver un valor int desde todos los posibles caminos de retorno (y puede ser invocado solamente en contextos que esperan un int para ser devuelto). Se usa la sentencia return dentro de un método para devolver un valor.
* **<nombre>:** puede ser cualquier identificador válido, con algunas restricciones basadas en los nombres que ya están en uso.
* **<argumento>:** permite que los valores de los argumentos sean pasados hacia el método. Los elementos de la lista están separados por comas y cada elemento consiste en un tipo y un identificador.

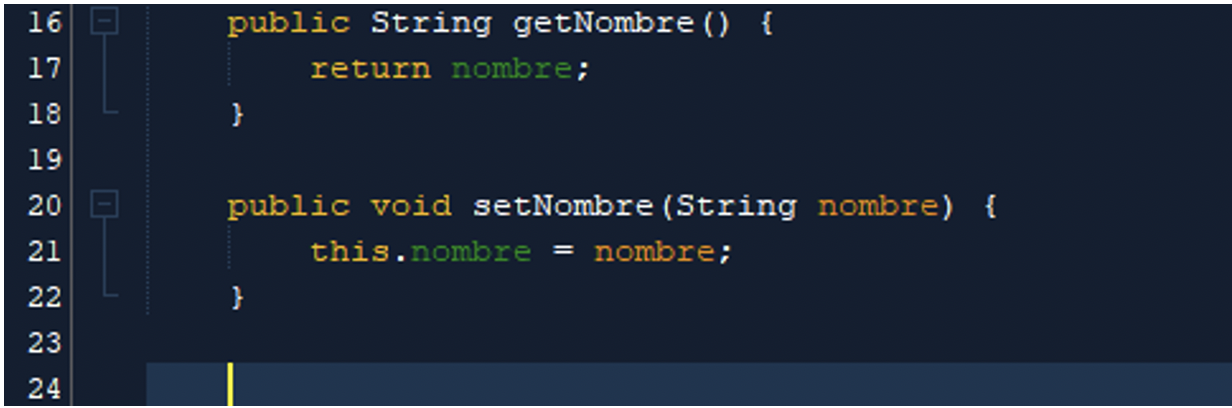
Existen tres tipos de métodos: métodos de consulta, métodos modificadores y operaciones. Los métodos de consulta sirven para extraer información de los objetos, los métodos modificadores sirven para modificar el valor de los atributos del objeto y las operaciones definen el comportamiento de un objeto.

# Getter & Setter

Para acceder a los atributos de un objeto se definen los **métodos get y set**. Los métodos get se utilizan para **consultar el estado de un objeto** y los métodos set para **modificar su estado**. Un método get se declara public y a continuación se indica el tipo de dato que devuelve. Es un método de consulta. La lista de parámetros de un método get queda vacía. En el cuerpo del método se utiliza return para devolver el valor correspondiente al atributo que se quiere devolver, y al cual se hace referencia como **this.nombreAtributo.**

Por otra parte, un método set se declara public y devuelve void. La lista de parámetros de un método set incluye el tipo y el valor a modificar. Es un método modificador. El cuerpo de un método set asigna al atributo del objeto el parámetro de la declaración.





### ¿Cómo se crean los getter y setter?

Dentro de la clase, para agregar automáticamente los getter & setter podemos presionar el botón derecho > Insert Code > Getter & Setter y selecciono todos los atributos.

## **¿Cómo utilizamos estos métodos?**

Un método se puede invocar dentro o fuera de la clase donde se ha declarado. Si el método se invoca dentro de la clase, basta con indicar su nombre**. Si el método se invoca fuera de la clase** entonces se debe indicar el nombre del objeto y el nombre del método. Cuando se invoca a un método ocurre lo siguiente:

* En la línea de código del programa donde se invoca al método se calculan los valores de los argumentos.
* Los parámetros se inicializan con los valores de los argumentos.
* Se ejecuta el bloque código del método hasta que se alcanza return o se llega al final del bloque.
* Si el método devuelve un valor, se sustituye la invocación por el valor devuelto.
* La ejecución del programa continúa en la siguiente instrucción donde se invocó el método.





## **Ejercicio persona – Get y set**

Continuaremos con nuestra clase Persona y vamos a primero crear los getter y setter y después invocarlos desde el Main con alguno de los objetos que instanciaste.



# En resumen

Antes de POO, la técnica estándar de programación era la programación procedural. Se denomina programación procedural porque en ella se destacan los procedimientos o tareas que resuelven un problema. Se piensa primero en lo que se quiere hacer: los procedimientos.

En contraste, el paradigma POO invita a pensar en lo que se desea que represente el programa. Normalmente se responde esta invitación identificando algunas cosas en el mundo que se desea que el programa modele.

Estas cosas podrían ser entidades físicas o conceptuales, por ejemplo, un libro. Una vez identificadas las cosas que se quiere modelar, se identifican sus propiedades/atributos básicos. Estos se pueden agrupar todos juntos en una estructura coherente llamada objeto que creamos a través de las clases.

