

ULP
Virtual

Tecnicatura Universitaria en Desarrollo de Software

Programación II

Guía 1.1

Introducción a Java



Universidad de
LA PUNTA



GOBIERNO DE
SAN LUIS

PARADIGMAS 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.

PROGRAMACIÓN ORIENTADA A OBJETOS:

Los Objetos:

Si miramos a nuestro alrededor, vemos que estamos rodeados de objetos. Y cuando debemos desarrollar un nuevo software para resolver un determinado problema, verificamos que los elementos que componen el problema son objetos. Pensemos en un sistema de gestión de negocios. Al analizar las características del problema, nos encontramos con que podríamos tener objetos como: productos, facturas, remitos, depósitos, clientes, entre otros. Y no sólo objetos concretos, sino también abstractos, como ser: vencimientos e impuestos.

La programación orientada a objetos es un paradigma basado en la identificación de los objetos inherentes al problema y en la escritura del código que modele esos objetos, sus propiedades, sus comportamientos y la forma en que se relacionan para resolver el problema. Visto de esta manera, la programación orientada a objetos resulta opuesta a la programación imperativa clásica, donde los programas son secuencias de instrucciones.

Clases y Objetos:

Cuando se analiza un problema para resolverlo mediante las técnicas de programación estructurada, nos enfocamos en identificar los datos que debemos manipular y las transformaciones que sufren como parte de la solución. Con el paradigma orientado a objetos, la tarea de análisis se centrará en la identificación de los objetos, de sus características y de cómo se relacionan entre sí para resolver el problema.

Ya dentro del contexto formal de la teoría de orientación a objetos, podemos definir un objeto como un concepto, abstracción o elemento con significado claro dentro del problema en cuestión (por ejemplo, un Vehículo). Como tal, un objeto se caracteriza por tener un estado, que es el conjunto de valores de sus propiedades en un momento del tiempo (por ejemplo: la marca del Vehículo, la cantidad de combustible que tiene el tanque). Además, todo objeto tiene un comportamiento, es decir, las acciones que puede realizar y que modificarán sus estados (Por ejemplo: un vehículo puede avanzar o retroceder). Por último, un objeto se caracteriza por tener identidad propia, esto es, por más que dos objetos tengan el mismo comportamiento y el mismo estado, resultan totalmente diferentes e identificables.

Según el diccionario, una clase es un grupo de elementos de un conjunto que tiene características comunes. Las técnicas de orientación a objetos se centran en la identificación de esos elementos comunes entre los objetos para poder agruparlos en clases y, así manipularlos fácilmente. Podemos decir que una clase es una abstracción de un grupo de objetos, porque no existe por sí sola.

Por ejemplo, pensemos en una mesa. Hay mesas rectangulares, mesas redondas, de madera, de metal, incluso de diferentes colores y tamaños. Todas son mesas distintas e identificables, es decir: objetos. Sin embargo, podemos verificar que todas tienen los mismos atributos, aunque con distintos valores. Todas tienen un color, un material, una forma, una dimensión; por lo tanto podemos agruparlas bajo un mismo concepto: la mesa. Desde el punto de vista de la orientación a objetos, podemos decir que **mesa** es una clase.



Ilustración 1 Clase mesa y sus instancias

Propiedades y Métodos:

Como decíamos, los objetos poseen atributos y comportamientos. Ya desde el punto de vista de la programación, los atributos de los objetos se traducen en propiedades de las clases que modelan.

El comportamiento en cambio, está representado por procedimientos y funciones que creamos dentro de la clase. En la jerga de la programación orientada a objetos, los procedimientos y funciones que creamos dentro de una clase se denominan **métodos**, y representan un conjunto de actividades que un objeto puede realizar, es decir, su comportamiento. Los libros más puristas dicen que los objetos se relacionan intercambiando mensajes, por lo que el comportamiento de un objeto está determinado por los mensajes que puede enviar y por los que puede aceptar desde otros objetos. Volviendo a la definición de método, justamente podemos decir, que los métodos son los encargados de interceptar y enviar mensajes desde y hacia otros objetos y, al mismo tiempo, alterar el estado actual del objeto.

Pensar en objetos:

Identificación de los objetos que modelan un problema.

El primer paso para modelar un problema usando la POO es la identificación de los objetos que son relevantes para representar el problema. No todos los posibles objetos que podemos hallar son necesarios en nuestro modelo. Los primeros candidatos a ser objetos son los sustantivos (nombres comunes) en el enunciado de un problema. Por ejemplo considere el siguiente problema:

Imaginemos que necesitamos implementar una aplicación de alquiler de autos, y nos presentan el siguiente requerimiento (resumido): La empresa dispone de autos para alquilar a sus clientes. Cuando se alquila un auto, el cliente firma una póliza de seguro. En un formulario de alquiler se registra la fecha, el nombre del cliente, el número de su registro de conductor y el número de su tarjeta de crédito.

Si aplicamos la técnica de los sustantivos, podremos identificar rápidamente tres objetos fundamentales para la aplicación: *auto*, *cliente* y *formulario de alquiler*. Dependiendo del tipo de análisis y diseño que queramos hacer, podríamos considerar como objetos también a la *tarjeta de crédito* e incluso, *el alquiler*.

Como decíamos al principio, los objetos poseen propiedades (atributos) y comportamientos (métodos).

Para identificar a las propiedades aplicamos una técnica similar a los sustantivos, pero buscando cualidades o adjetivos que ya hemos identificado. Siguiendo con el ejemplo de alquiler de autos, podemos ver que el formulario de alquiler tiene propiedades como: la fecha, el nombre del cliente, etc.

Una vez que hemos identificado los objetos y sus propiedades estamos en condiciones de definir clases para agruparlos y abstraer todas las posibles instancias de ellos.

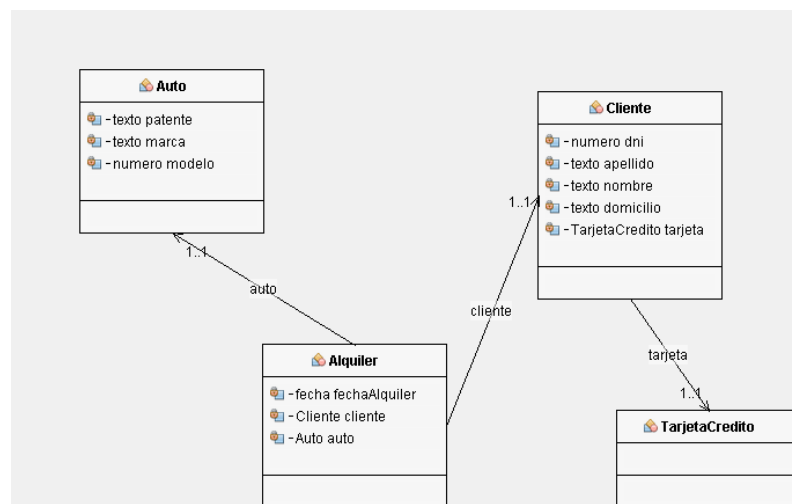


Ilustración 2 Diagrama de clases UML

Como lenguaje que soporta al paradigma orientado a objetos utilizaremos Java.

CARACTERISTICAS DEL LENGUAJE JAVA:

Java es una **plataforma de desarrollo** diseñada por Sun Microsystems.

A la mayoría de las plataformas de desarrollo se las pueden describir como una combinación de hardware y software mediante la cual se ejecuta un sistema. La diferencia de la plataforma Java con respecto a otras plataformas es que solamente se compone por **software** que corre sobre otras plataformas.

La plataforma Java está constituida por los siguientes componentes:

- Máquina Virtual Java (JVM).
- Interfaz de Programación de Aplicaciones Java (Java API).
- Lenguaje de Programación Java.

La Máquina Virtual Java:

Junto al lenguaje de programación, también se desarrolló un **procesador del lenguaje** encargado de **interpretar** instrucciones en un código binario especial de Java. Este procesador es la **Máquina Virtual de Java** o **JVM** por sus siglas en inglés (Java Virtual Machine).

La JVM representa un concepto bastante importante en la ejecución de programas basados en la plataforma Java. Es el componente encargado de:

- Proveer la independencia en cuanto al hardware y sistema operativo sobre los cuales pueden correr las aplicaciones Java.
- El pequeño **tamaño del código** compilado.
- **Proteger** a los usuarios de posibles aplicaciones maliciosas.

La JVM es una **máquina abstracta de computación**. Al igual que toda máquina de computación, la JVM posee un conjunto de instrucciones que puede interpretar y manipula sectores de memoria en tiempo de ejecución.

En el lenguaje de programación Java, todas las instrucciones que hacen a la constitución de un programa, son escritas en archivos de texto plano con formato “**.java**”. Estos archivos contienen el **código fuente** de un programa. La JVM no conoce nada sobre el lenguaje Java, solamente conoce cómo **interpretar** archivos en un formato binario en particular (“**.class**”).

Para generar estos **archivos binarios**, los **archivos fuentes** son **compilados** utilizando una aplicación específica para este fin (**compilador javac**). Se deben compilar todos los archivos fuentes y por cada uno de ellos tendremos el archivo binario correspondiente con formato “**.class**”.

Los archivos binarios contienen instrucciones de la máquina virtual en **código de bytes** (**bytecode**). Por razones de seguridad, la máquina virtual java impone un formato estricto y estructurado sobre el contenido de un archivo con formato “**.class**”.

La Compilación tiene los siguientes objetivos:

- Revisar los errores en la sintaxis del código fuente.
- Convertir el código fuente en código binario.

Cuando se ejecuta un programa, se ejecuta el comando “**java**” que se encarga de instanciar a la máquina virtual Java para **interpretar** las instrucciones de los archivos binarios y de esta forma **ejecutar** la aplicación programada.

Archivo fuente (.java)

La codificación de un programa se compone de archivos fuentes con formato .java

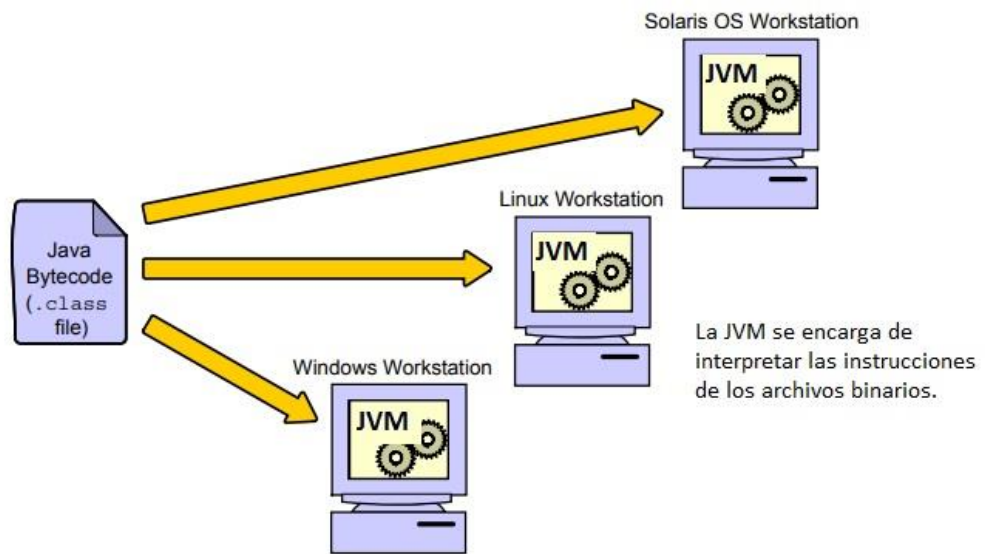
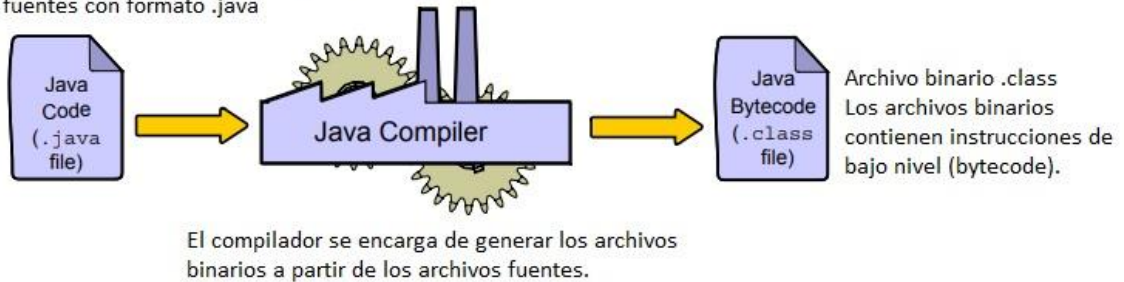


Ilustración 3 Proceso de ejecución de app Java

Se han desarrollado **distintas máquinas virtuales** para **distintas arquitecturas**, para de esta forma, las instrucciones de un archivo binario puedan ser interpretadas y ejecutadas por la Máquina Virtual **independientemente de la plataforma** sobre la que está corriendo. Es decir, la posibilidad de contar con intérpretes Java para cada plataforma permite que un mismo archivo “.class” sea posible ejecutarlo sobre distintos sistemas operativos (Microsoft Windows, Linux, Mac OS, Solaris OS, etc.).

ARCHIVO FUENTE EN JAVA

Un archivo fuente de la tecnología Java tiene la siguiente forma:

```
[<declaración_paquete>]
<declaración_importacion>*
<modificador>* <declaración_clase>{
    <atributos>*
    <constructores>*
    <métodos>*
}
```

El orden de estos puntos es importante. Cualquier sentencia de importación debe preceder todas las declaraciones de clases. Si usa una declaración de paquetes, el mismo debe preceder tanto a las clases como a las importaciones. El nombre del archivo fuente tiene que ser el mismo que el nombre de la declaración de la clase pública en ese archivo. Un archivo fuente puede incluir más de una declaración de clase, pero sólo una puede ser declarada pública. Si un archivo fuente no contiene clases públicas declaradas, el nombre del archivo fuente no está restringido. Sin embargo, es una buena práctica tener un archivo fuente para cada clase declarada y que el nombre del archivo sea idéntico al nombre de la clase y con extensión “.java”.

ESTRUCTURA DE UN PROGRAMA JAVA

Un programa describe cómo un ordenador debe interpretar las órdenes del programador para que ejecute y realice las instrucciones dadas tal como están escritas. Un programador utiliza los elementos que ofrece un lenguaje de programación para diseñar programas que resuelvan problemas concretos o realicen acciones bien definidas.

El siguiente programa Java muestra un mensaje en la consola con el texto “Hola Mundo”.

```
/** Este programa escribe el texto "Hola Mundo" en la consola, utilizando el
método System.out.println()
*/

package primerprograma;

public class HolaMundo {

    public static void main (String[] args) {

        System.out.println("Hola Mundo");

    }

}
```

Ejemplo 1

En este programa (Ejemplo 1), se pueden identificar los siguientes elementos del lenguaje Java: Comentarios, paquete, definiciones de clase, definiciones de método y sentencias.

COMENTARIO

Una aplicación Java puede ser documentada mediante la inserción de **comentarios** en algunos sectores relevantes del código fuente. El propósito de estos comentarios es el de mantener **documentada** la aplicación para facilitar el entendimiento de la misma. Estos comentarios son ignorados por el compilador.

Los tipos de comentarios que existen son los siguientes:

- 1) **Comentario de única línea:** se indican mediante los caracteres //
- 2) **Comentario de líneas múltiples:** Se indican utilizando los caracteres /* y */ en dicho orden. /* son los caracteres de inicio del comentario y */ son los caracteres de fin de comentario.
- 3) **Comentario de javadoc:** este tipo de comentario se indica utilizando los caracteres /** y */ en dicho orden. Los comentarios javadoc son utilizados para generar documentación HTML del programa en forma automatizada. Este tipo de comentario suele colocarse inmediatamente antes de la declaración de una clase, interface o método.

PAQUETES

Los paquetes son contenedores de clases y su función es la de organizar la distribución de las clases. Los paquetes y las clases son análogos a las carpetas y archivos utilizados por el sistema operativo, respectivamente.

El lenguaje de programación de la tecnología Java le provee la sentencia **package** como la forma de agrupar clases relacionadas. La sentencia package tiene la siguiente forma:

```
package <nombre_paq_sup>[.<nombre_sub_paq>]*;
```

La declaración package, en caso de existir, debe estar al principio del archivo fuente y sólo la declaración de un paquete está permitida. Los nombres de los paquetes los pondrá el programador al crear el programa y son jerárquicos (al igual que una organización de directorios en disco) además, están separados por puntos. Es usual que sean escritos completamente en minúscula.

CLASES

En el mundo de orientación a objetos, todos los programas se definen en término de objetos y sus relaciones. Las clases sirven para modelar los objetos que serán utilizados por nuestros programas. Una clase está formada por una parte correspondiente a la declaración de la clase, y otra correspondiente al cuerpo de la misma:

```
Declaración de clase {  
Cuerpo de clase  
}
```

Una **clase es un molde** para crear múltiples objetos que encapsulan 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.

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  
}
```

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

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.

ATRIBUTOS

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

```
<modificador>* <tipo> <nombre> [ =<valor inicial> ];
```

- **<nombre>** : puede ser cualquier identificador válido y denomina el atributo que está siendo declarado.
- **<modificador>** : si bien hay varios valores posibles para el , por el momento solo usaremos modificadores de visibilidad: public, protected, private.
- **<tipo>** : indica el tipo de dato que va a guardar la variable atributo definida.
- **<valor inicial>**: esta sentencia es opcional y se usa para inicializar el atributo del objeto con un valor particular.

Es una buena práctica colocar los atributos al principio de la clase.

CONSTRUCTORES

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 la clase> (<argumentos> * ) {  
    <sentencias> *  
}
```

- **<nombre de la 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. 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 creamos un nuevo constructor vacío.

MÉTODOS

Un método es una secuencia de sentencias ejecutables. Las sentencias de un método quedan delimitadas por los caracteres `{` y `}` que indican el inicio y el fin del método, respectivamente. Podemos decir que el término método en Java es equivalente al de subprograma, rutina, subrutina, procedimiento o función en otros lenguajes de programación.

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;  
}
```

- **<nombre>**: puede ser cualquier identificador válido, con algunas restricciones basadas en los nombres que ya están en uso.
- **<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.

- `<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.

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. 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.

Por último, un método de tipo operación es aquel que realiza un cálculo o modifica el estado de un objeto. Este tipo de métodos pueden incluir una lista de parámetros y puede devolver un valor o no. Si el método no devuelve un valor, se declara *void*.

Invocación de 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.

Parámetros y argumentos

Los parámetros de un método definen la cantidad y el tipo de dato de los valores que recibe un método para su ejecución. Los argumentos son los valores que se pasan a un método durante su invocación. El método recibe los argumentos correspondientes a los parámetros con los que ha

sido declarado. Un método puede tener tantos parámetros como sea necesario. La lista de parámetros de la cabecera de un método se define con la siguiente sintaxis:

tipo nombre [tipo nombre,]

Durante la invocación de un método es necesario que el número y el tipo de argumentos coincidan con el número y el tipo de parámetros declarados en la cabecera del método. Durante el proceso de compilación se comprueba que durante la invocación de un método se pasan tantos argumentos como parámetros tiene declarados y que además coinciden los tipos. Esta es una característica de los lenguajes que se denominan “strongly typed” o “fuertemente tipado”

Paso de parámetros

Cuando se invoca un método se hace una copia de los valores de los argumentos en los parámetros. Esto quiere decir que, si el método modifica el valor de un parámetro, nunca se modifica el valor original del argumento. Cuando se pasa una referencia a un objeto se crea un nuevo alias sobre el objeto, de manera que esta nueva referencia utiliza el mismo espacio de memoria del objeto original y esto permite acceder al objeto original.

El valor de retorno

Un método puede devolver un valor. Los métodos que no devuelven un valor se declaran void, mientras que los métodos que devuelven un valor indican el tipo que devuelven: int, double, char, String o un tipo de objeto.

Sobrecarga de métodos

La sobrecarga de métodos es útil para que el mismo método opere con parámetros de distinto tipo o que un mismo método reciba una lista de parámetros diferente. Esto quiere decir que puede haber dos métodos con el mismo nombre que realicen dos funciones distintas. La diferencia entre los métodos sobrecargados está en su declaración, y más específicamente, en la cantidad y tipos de datos que reciben

MODIFICADORES DE ACCESO

Para lograr el uso correcto del encapsulamiento vamos 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 a 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 |

ATRIBUTOS Y MÉTODOS ESTÁTICOS

Un atributo o un método de una clase se puede modificar con la palabra reservada static para indicar que este atributo o método no pertenece a las instancias de la clase si no a la propia clase. Se dice que son atributos de clase si se usa la palabra clave static: en ese caso la variable es única para todas las instancias (objetos) de la clase (ocupa un único lugar en memoria), es decir que, si se poseen múltiples instancias de una clase, cada una de ellas no tendrán una copia propia de este atributo, si no que todas estas instancias compartirán una misma copia del atributo. A veces a las variables de clase se les llama variables estáticas. Si no se usa static, el sistema crea un lugar nuevo para esa variable con cada instancia (la variable es diferente para cada objeto).

En el caso de una constante no tiene sentido crear un nuevo lugar de memoria por cada objeto de una clase que se cree. Por ello es adecuado el uso de la palabra clave static. Cuando usamos "static final" se dice que creamos una constante de clase, un atributo común a todos los objetos de esa clase.

```
public class Auto {
    private String patente;
    private int modelo;
    private static int cantidad;
}
```

Ejemplo 2

ATRIBUTOS FINALES

En este contexto indica que una variable es de tipo constante: no admitirá cambios después de su declaración y asignación de valor. La palabra reservada final determina que un atributo no puede ser sobrescrito o redefinido, es decir, no funcionará como una variable “tradicional”, sino como una constante. Toda constante declarada con final ha de ser inicializada en el mismo momento de declararla. El modificador final también se usa como palabra clave en otro contexto: una clase final es aquella que no puede tener clases que la hereden. Lo veremos más adelante cuando hablemos sobre herencia.

Cuando se declaran constantes es muy frecuente que los programadores usen letras mayúsculas (como práctica habitual que permite una mayor claridad en el código), aunque no es obligatorio.

```
public class Matematico
{   private static final double PI=3.14;
}
```

Ejemplo 3

MÉTODO MAIN()

Ahora sabemos lo que es un método, pero en el *ejemplo 1* podemos ver el método main(). El main() sirve para que un programa se pueda ejecutar, este método, vendría a representar el Algoritmo / FinAlgoritmo de pseint y tiene la siguiente declaración:

```
public static void main(String[] args){
```

El método main() debe aceptar siempre, como parámetro, un vector de String, que contendrá los posibles argumentos que se le pasen al programa en la línea de comandos, aunque en nuestro caso, no se utilice.

Luego, al indicarle a la máquina virtual que ejecute una aplicación el primer método que ejecutará es el método main(). Si indicamos a la máquina virtual que corra una clase que no contiene este método, se lanzará un mensaje advirtiéndolo que la clase que se quiere ejecutar no contiene un método main(), es decir que dicha clase no es ejecutable.

Si no se han comprendido hasta el momento muy bien todos estos conceptos, los mismos se irán comprendiendo a lo largo del curso.

IDENTIFICADOR

Los identificadores son los nombres que se usan para identificar cada uno de los elementos del lenguaje, como ser, los nombres de las variables, nombres de las clases, interfaces, atributos y métodos de un programa.

Técnicamente, un **identificador debe obedecer las siguientes reglas:**

- Los identificadores deben comenzar con una letra, el símbolo de moneda (\$), o un carácter de subrayado “_”.
- Los identificadores no pueden comenzar con un número.
- Después del primer carácter, los identificadores pueden contener cualquier combinación de letras, símbolos de moneda (\$), caracteres de subrayado o números.
- Los identificadores no pueden contener espacios en blanco.
- Los identificadores pueden tener cualquier longitud.

VARIABLES Y CONSTANTES

Recordemos que en Pseint dijimos que los programas de computadora necesitan información para la resolución de problemas. Está información puede ser un número, un nombre, etc. Para utilizar la información, vamos a guardarla en algo llamado, variables y constantes. Las variables y constantes vendrían a ser como pequeñas cajas, que guardan algo en su interior, en este caso información. Estas, van a contar como previamente habíamos mencionado, con un identificador, un nombre que facilitara distinguir unas de otras y nos ayudara a saber que variable o constante es la que contiene la información que necesitamos.

Dentro de toda la información que vamos a manejar, a veces, necesitaremos información que no cambie. Tales valores son las constantes. De igual forma, existen otros valores que necesitaremos que cambien durante la ejecución del programa; esas van a ser nuestras variables.

DECLARACIÓN DE VARIABLES EN JAVA

Normalmente los identificadores de las variables y de las constantes con nombre deben ser declarados en los programas antes de ser utilizadas. La sintaxis de la declaración de una variable en java suele ser:

```
<tipo_de_dato> <nombre_variable>;
```

TIPOS DE DATO EN JAVA

Por **tipo de dato** vamos a entender al **conjunto de valores** que una variable puede tener. Además de definir el tipo de valor de una variable, el tipo de dato también **define las operaciones** que se pueden realizar con dicha variable.

El tipo de dato de una variable, se establece en la **declaración de la variable**.

En **Java** existen dos tipos de datos:

- **Tipos primitivos:** se utilizan para declarar variables que se almacenan en forma directa en memoria. No son objetos.
- **Tipos referencia:** se utilizan para manipular objetos. Su concepto es similar a lo que en otros lenguajes se denominan “punteros”.

TIPOS DE DATOS PRIMITIVOS

Primitivos: Son predefinidos por el lenguaje y se pueden dividir en cuatro categorías:

- **Numéricos integrales:** byte, short, int, long.
- **Numéricos de punto flotante:** float, double.
- **Valores lógicos:** boolean.
- **Caracteres simples:** char.

Cada tipo define un rango de valores que una variable puede almacenar. Este rango de valores está definido por la cantidad de bits que ocupa su almacenamiento en memoria.

| | |
|--------|--|
| byte | Es un entero con signo de 8 bits, el mínimo valor que se puede almacenar es 128 y el máximo valor es de 127 (inclusive). |
| short | Es un entero con signo de 16 bits. El valor mínimo es -32,768 y el valor máximo 32,767 (inclusive). |
| int | Es un entero con signo de 32 bits. El valor mínimo es -2,147,483,648 y el valor máximo es 2,147,483,64 (inclusive). Generalmente es la opción por defecto. |
| long | Es un entero con signo de 64 bits, el valor mínimo que puede almacenar este tipo de dato es -9,223,372,036,854,775,808 y el máximo valor es 9,223,372,036,854,775,807 (inclusive). |
| float | Es un número decimal de precisión simple de 32 bits (IEEE 754 Punto Flotante). |
| double | Es un número decimal de precisión doble de 64 bits (IEEE 754 Punto Flotante). |

| | |
|---------|--|
| boolean | Este tipo de dato sólo soporta dos posibles valores: verdadero o falso y el dato es representado con tan solo un bit de información. |
| char | El tipo de dato carácter es un simple carácter unicode de 16 bits. Su valor mínimo es de '\u0000' (En entero es 0) y su valor máximo es de '\uffff' (En entero es 65,535). Nota: un dato de tipo carácter se puede escribir entre comillas simples, por ejemplo 'a', o también indicando su valor Unicode, por ejemplo '\u0061'. |

CADENA DE CARACTERES:

Un caso especial de tipo de datos es el de las cadenas de caracteres. Una cadena de caracteres no representa un tipo de dato primitivo, son **objetos de la clase String**.

Encerrando la cadena de caracteres con comillas dobles se creará de manera automática una nueva instancia de un objeto tipo String.

```
String cadena = "Sebastián";
```

Cuando asignamos una cadena de caracteres a una variable, en realidad Java está creando un objeto de la clase String. En el módulo siguiente veremos que para declarar e inicializar un objeto es necesario usar el operador **new**. Sin embargo, para facilitar el uso de este tipo de dato ya que es de uso muy común en la programación, Java permite hacer uso de las cadenas como si se tratara de un tipo de dato primitivo.

EJEMPLO

Después de haber leído estos primeros conceptos, vamos a aplicarlos en un ejemplo:

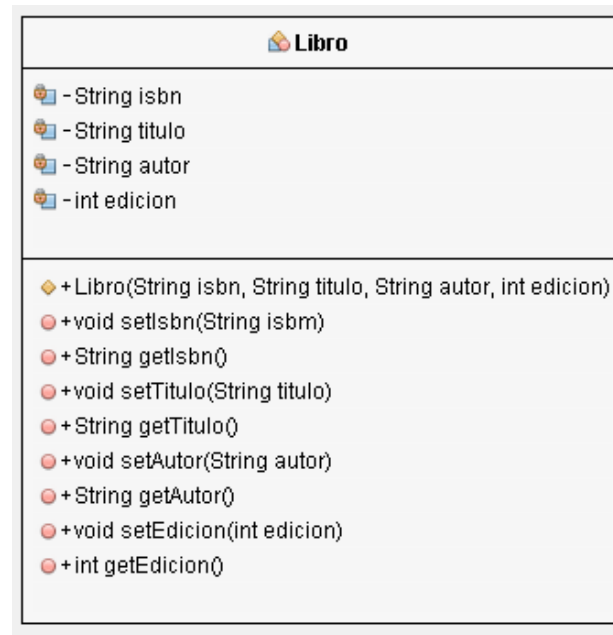
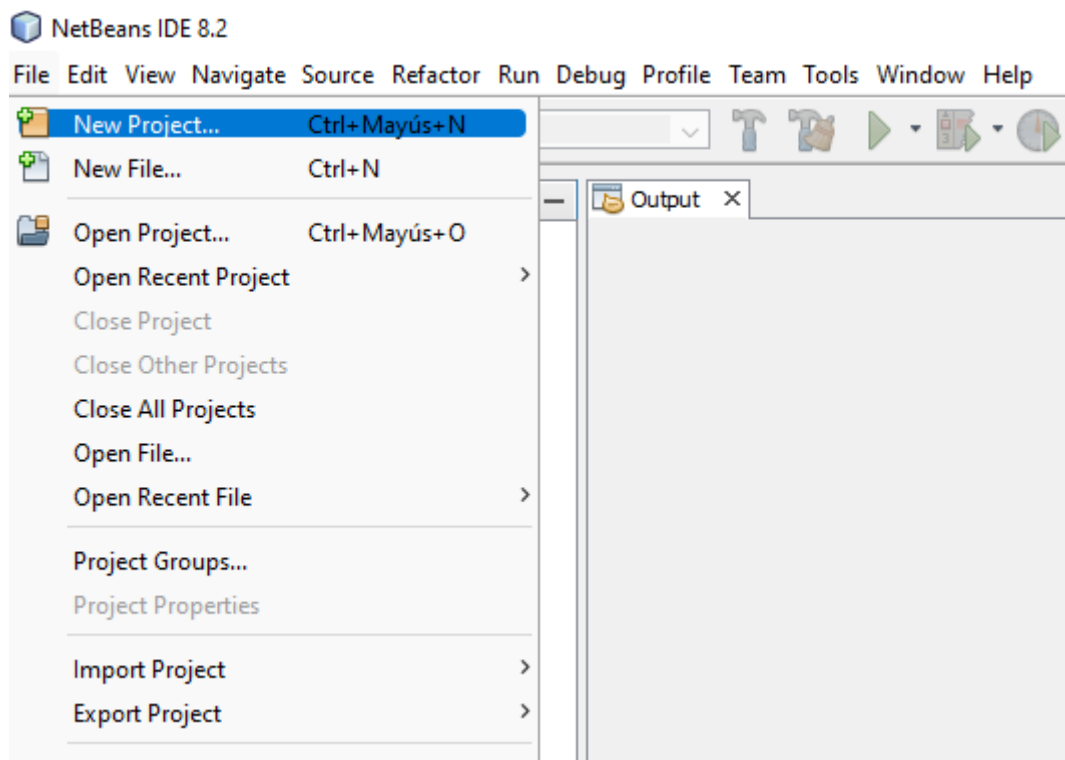
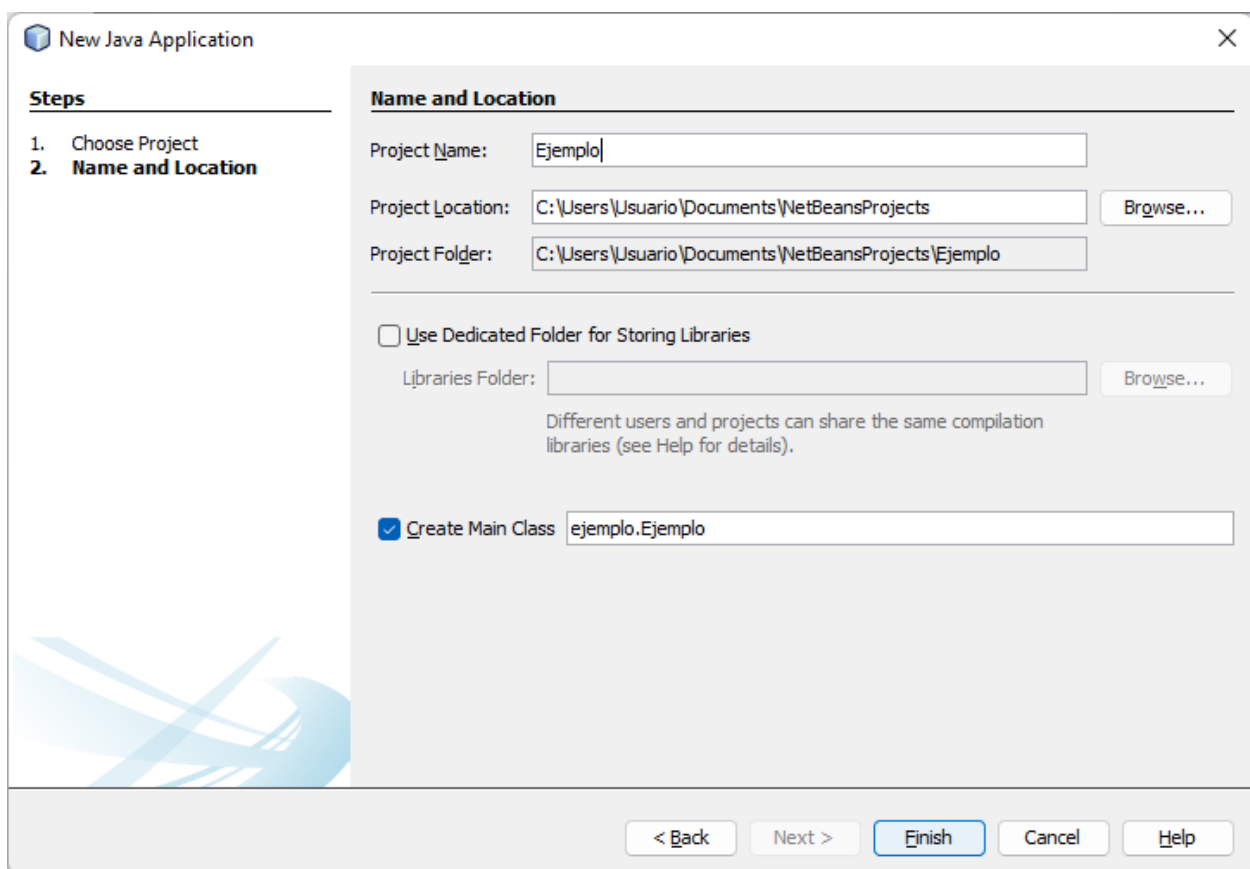
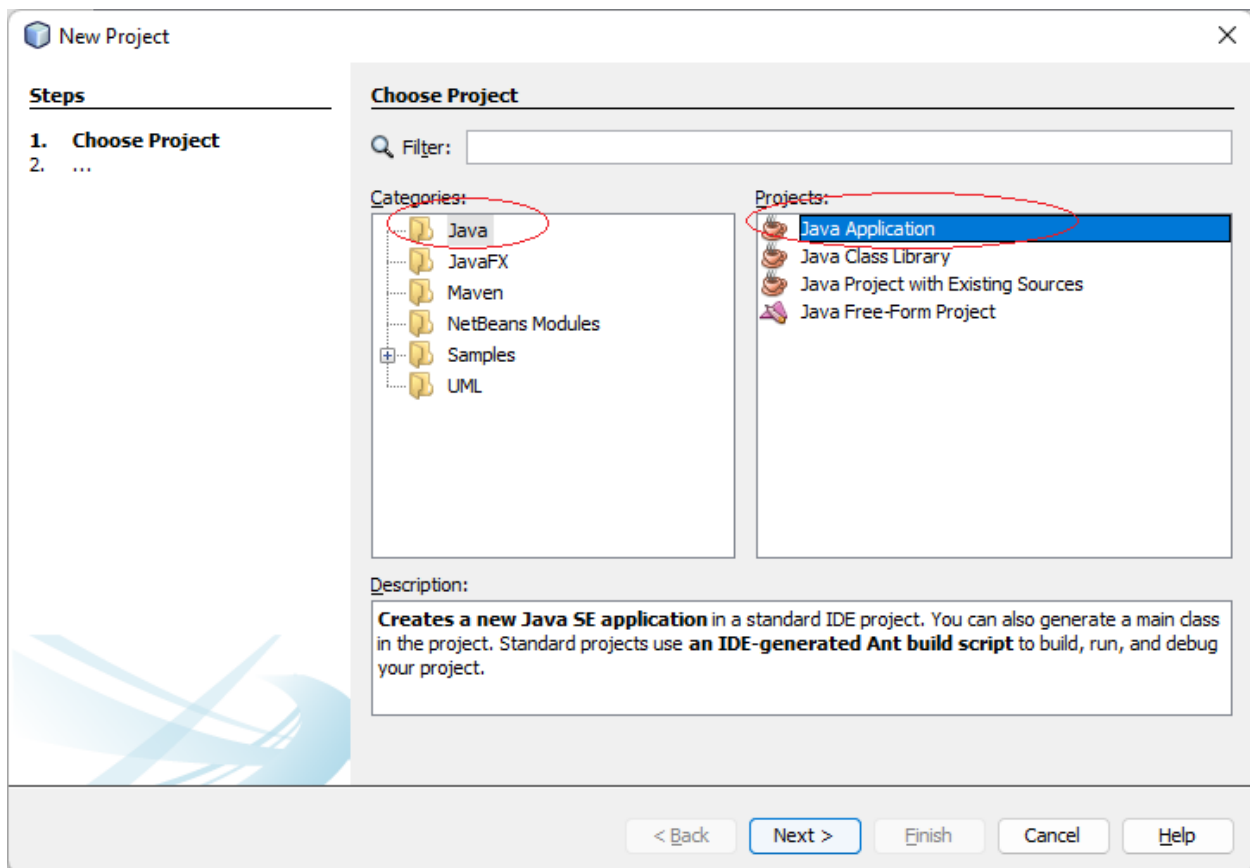


Ilustración 4 Diagrama UML- Clase Libro

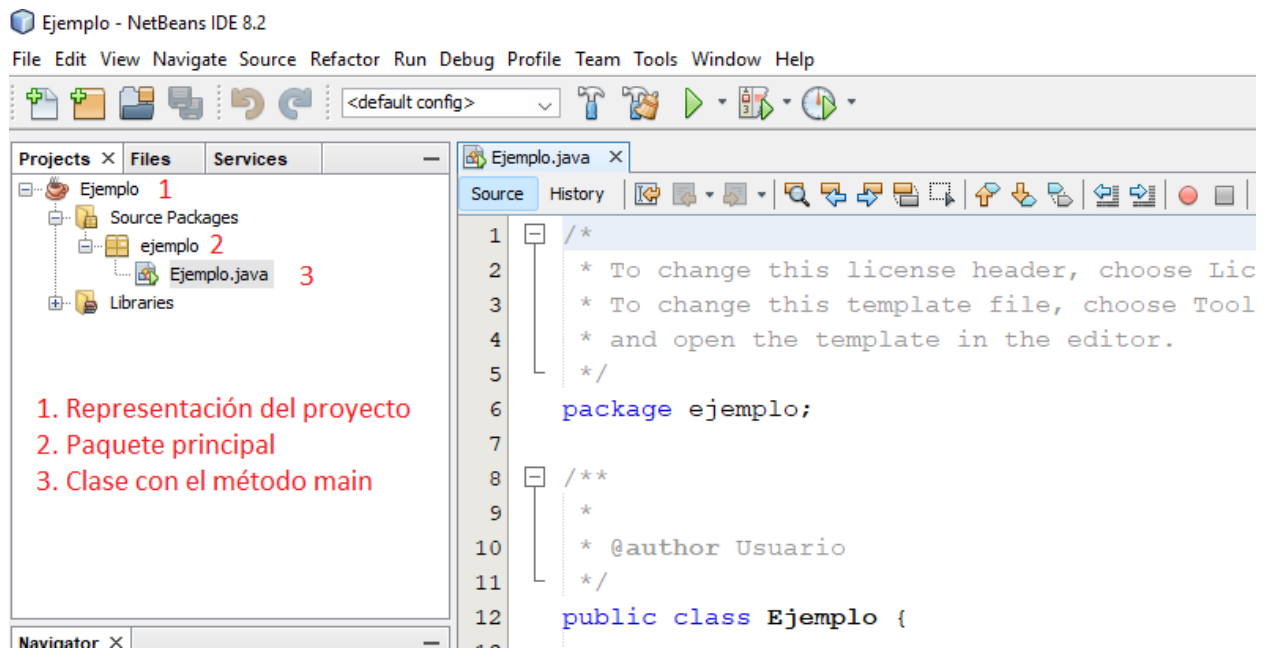
En la ilustración 4, tenemos representado a través de un diagrama de clases UML la plantilla o clase que representa a un Libro; en ella están representados los atributos: ISBN, TITULO, AUTOR y EDICION. Además, como podemos observar posee un constructor que permitirá inicializar todos los atributos de la clase y los respectivos métodos getter y setter que tendrán por tarea permitir cambiar o ver desde el exterior el estado de sus atributos.

Lo primero que haremos será desde Netbeans crear un nuevo proyecto de nombre Ejemplo.

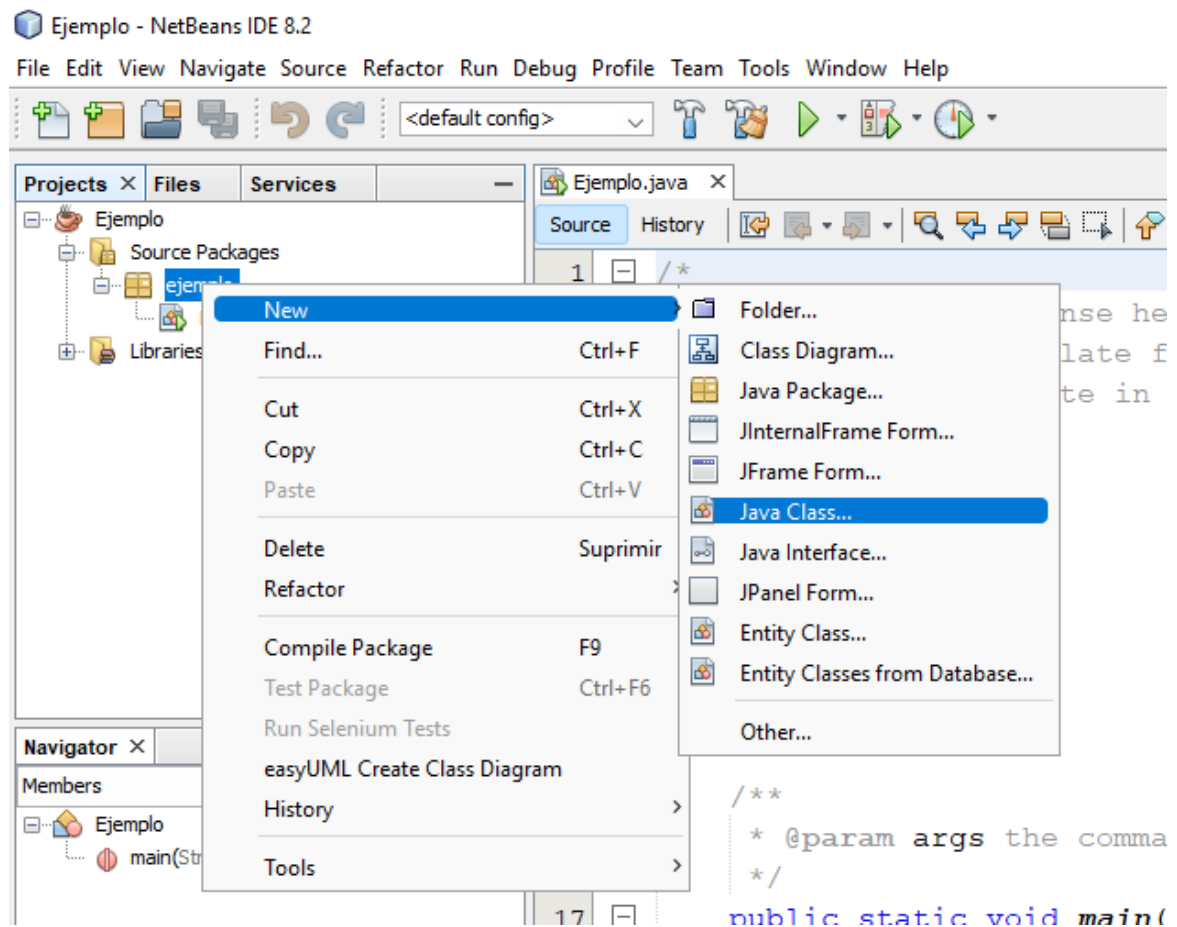




Cuando hacemos clic en el botón “Finish”, nos creará un nuevo proyecto con la estructura que se muestra a continuación.



Para crear la nueva clase Libro, dentro del paquete principal del proyecto, haremos clic secundario sobre el mismo y elegimos **New->Java Class**.



En la siguiente vista, colocaremos el nombre de la clase y Netbeans creará automáticamente un nuevo archivo con extensión .java que contendrá la definición de nuestra clase.

New Java Class

Steps

1. Choose File Type
- 2. Name and Location**

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

< Back Next > **Finish** Cancel Help

A continuación se muestran las líneas de código que implementan a la clase Libro

```
1 //paquete en donde se encuentra la clase
2 package ejemplo;
3
4 //comienza la definición de la clase Libro
5 public class Libro {
6
7     //ocultamos el estado de los atributos utilizando el modificador private
8     private String isbn;
9     private String titulo;
10    private String autor;
11    private int edicion;
12
13    /*Constructor de la clase, es como un método, ya que puede recibir
14    datos por parámetro, pero no tiene tipo de retorno ni siquiera void,
15    además el nombre es el mismo que el de la clase.
16    */
17    public Libro(String isbn, String titulo, String autor, int edicion) {
18        this.isbn = isbn;
19        this.titulo = titulo;
20        this.autor = autor;
21        this.edicion = edicion;
22    }
23
24    //Métodos getter y setter
25
26    public String getIsbn() {
27        return isbn;
28    }
29
30    public void setIsbn(String isbn) {
31        this.isbn = isbn;
32    }
33
34    public String getTitulo() {
35        return titulo;
36    }
37
38    public void setTitulo(String titulo) {
39        this.titulo = titulo;
40    }
41
42    public String getAutor() {
43        return autor;
44    }
```

```
45
46     public void setAutor(String autor) {
47         this.autor = autor;
48     }
49
50     public int getEdicion() {
51         return edicion;
52     }
53
54     public void setEdicion(int edicion) {
55         this.edicion = edicion;
56     }
57
58
59
60 }
```


Ahora, desde la clase Ejemplo.java, que es la clase que contiene el método main, procederemos a instanciar un objeto de tipo Libro y mostraremos por consola sus datos.

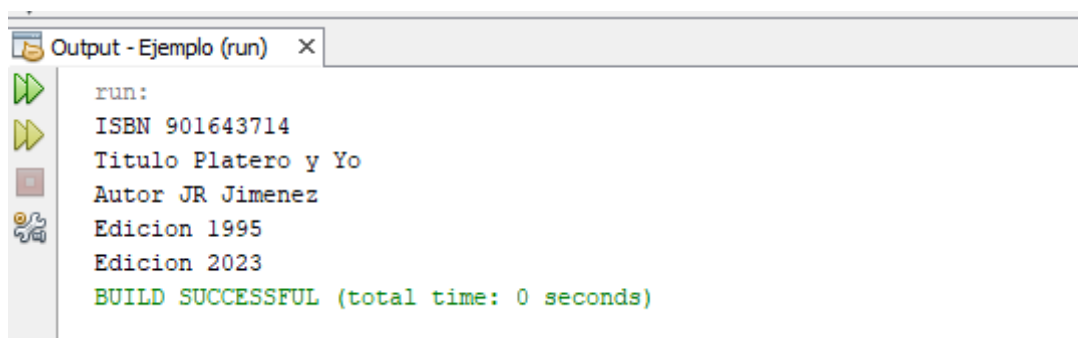
```
1
2 package ejemplo;
3
4 public class Ejemplo {
5
6     public static void main(String[] args) {
7
8         Libro platero =new Libro("901643714","Platero y Yo","JR Jimenez",1995);
9         System.out.println("ISBN "+platero.getIsbn());
10        System.out.println("Titulo "+platero.getTitulo());
11        System.out.println("Autor "+platero.getAutor());
12        System.out.println("Edicion "+platero.getEdicion());
13
14        platero.setEdicion(2023);
15        System.out.println("Edicion "+platero.getEdicion());
16
17    }
18 }
19
```

Como pueden observar, desde la línea 6 hasta la línea 17, está la definición del método main. En la línea 8, creamos un objeto de tipo Libro, pasando a través de los parámetros del constructor sus datos y guardamos en la variable local “platero”, la referencia a este objeto. De aquí en adelante, cada vez que quiera acceder a alguna función de este libro creado lo haremos a través de la variable de referencia y el operador punto (platero.)

Desde las líneas 9 hasta la 12, mostramos por consola, utilizando la función println (print line), los datos del libro instanciado en la línea 8 utilizando sus métodos get.

En la línea 14, cambiamos el estado del año de edición del libro creado en la línea 8, utilizando el método set para dicho atributo (setEdicion).

Por lo tanto si ejecutamos este código, la salida por consola, será la siguiente:



```
run:
ISBN 901643714
Titulo Platero y Yo
Autor JR Jimenez
Edicion 1995
Edicion 2023
BUILD SUCCESSFUL (total time: 0 seconds)
```

BIBLIOGRAFIA:

Christopher Alexander, “A Pattern Language”, 1978

Erich Gamma, “Design Patterns: Elements of Reusable OO Software”

Martin Fowler, “Analysis Patterns: Reusable Object Models”, Addison Wesley,
1997