

## Índice de contenido

### 1. Tipos de Clase en Java

### 2. Estructura de una clase POJO

#### 2.1. Declaración de paquetes

#### 2.2. Declaración de imports

- Análisis del import y la librerías del JDK desde nuestro computador

#### 2.3. Declaración de la clase

##### 2.3.1. Declaración de atributos

- Encapsulamiento y atributos private
- Atributos con valor inicial

##### 2.3.2. Declaración de métodos

###### 2.3.2.1. Método Constructor

- Constructor vacío o por defecto
- Constructor sin parámetros
- Constructor con parámetros

###### 2.3.2.2. Métodos Getter y Setter

- Método Getter
- Método Setter

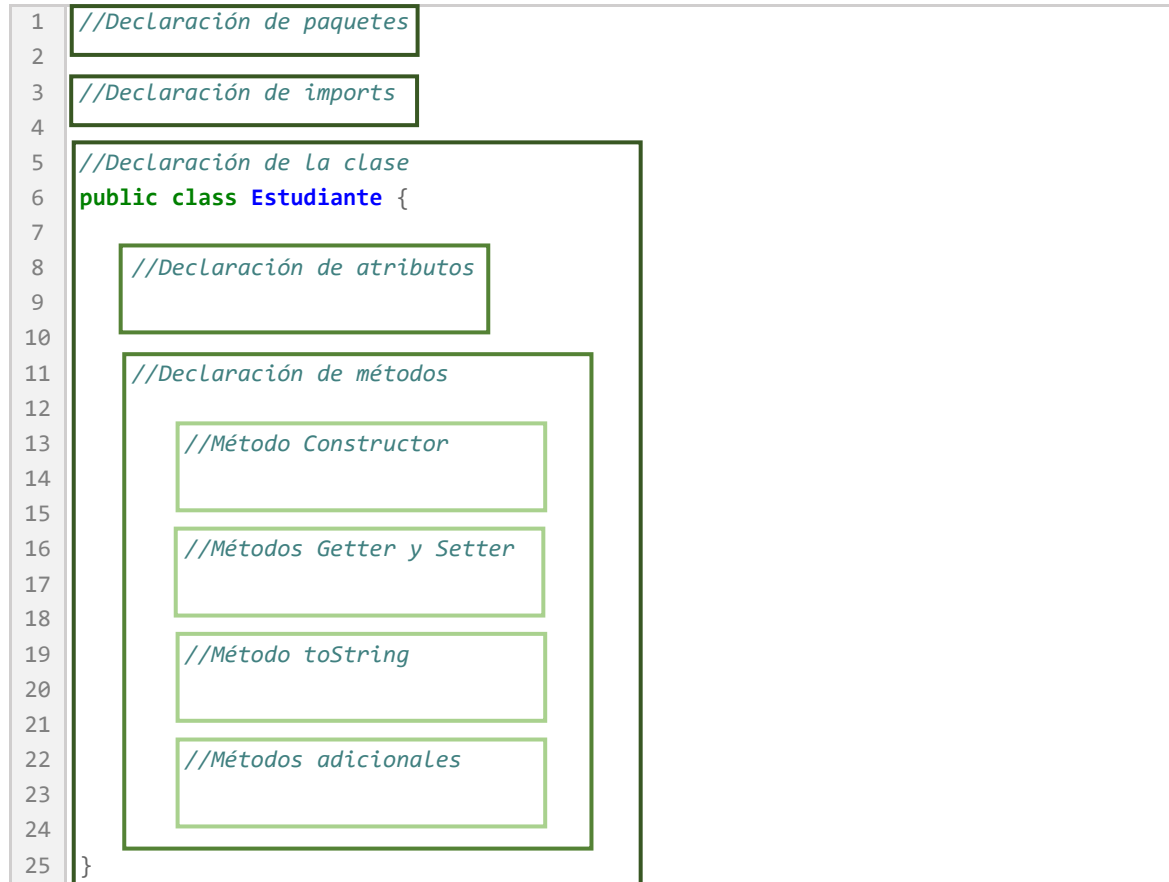
###### 2.3.2.3. Método toString

## 1. Tipos de clases en Java

En la programación básica de java podemos distinguir dos “tipos” de clases: Clases POJO y clases de servicio o lógicas.

Las clases POJO se utilizan para representar datos en nuestro programa de manera simple y sin lógica adicional del programa o negocio. Por otro lado, las clases de servicio o lógicas se utilizan para escribir procesos del programa y contienen la lógica del negocio.

## 2. Estructura de una clase POJO



Una clase POJO está estructurada en la siguiente manera:

1. Declaración de paquetes
2. Declaración de import
3. Declaración de la clase
  - a. Declaración de atributos
  - b. Declaración de métodos
    - i. Declaración de método constructor
    - ii. Declaración de método Getter y Setter
    - iii. Declaración de método toString
    - iv. Declaración de métodos adicionales

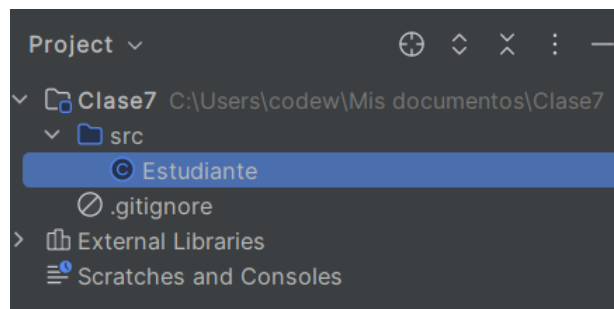
## 2.1. Declaración de paquetes

Si una clase se encuentra declarada directamente en el paquete “src” la declaración de paquete no es necesaria ya que está en el directorio raíz de nuestro proyecto. De lo contrario, si la clase se encuentra dentro de un paquete del directorio raíz se debe indicar la dirección del paquete en el que se encuentra.

Si utilizamos IntelliJ Idea u otro IDE como Eclipse o NetBeans al momento en que creamos el paquete y metemos la clase dentro de ella, la declaración de paquetes se hará automáticamente.

Por ejemplo:

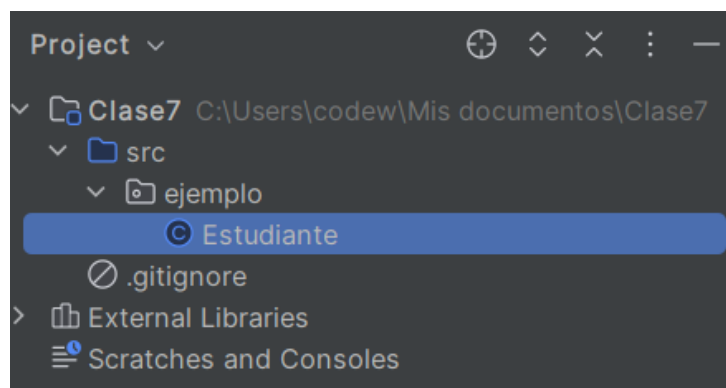
Creamos un proyecto “Clase7” en IntelliJ. Luego creamos una clase “Estudiante” en el paquete raíz “src”:



Si vemos el código fuente de nuestra clase “Estudiante” no se encuentra declarado ningún paquete en el código de la clase:

```
1  
2 public class Estudiante {  
3  
4 }  
5
```

Ahora, si creamos un paquete en “src” y le llamamos “ejemplo” y movemos la clase “Estudiante” dentro del paquete “ejemplo” veremos el siguiente cambio en el código de la clase “Estudiante”.



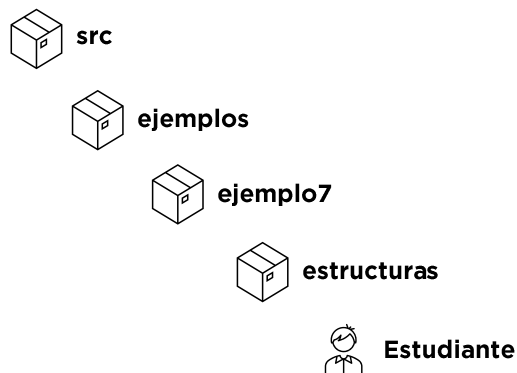
```
1  
2 package ejemplo;  
3  
4 public class Estudiante {  
5  
6 }  
7
```

Vemos que se agregó en la línea 2, la declaración del paquete en el que se encuentra ubicado nuestra clase.

La declaración siempre inicia con la palabra reservada “package” seguida de la ruta del paquete de la clase.

En nuestro ejemplo solo tenemos un paquete; sin embargo, en proyectos mas complejos veremos que se puede tener una jerarquía compleja de paquetes. Para estos casos se emplea el uso del “punto” “.” Para indicar la jerarquía de los paquetes:

Jerarquía de paquetes de ejemplo:



Declaración del paquete en la clase “Estudiante”:

```
package ejemplos.ejemplo7.estructura;
```

La mayoría de las veces el IDE se encargará de hacer las declaraciones de los paquetes por más complicada que sea.

## 2.2. Declaración de imports

Como ya hemos hablado anteriormente, en Java existen paquetes con clases predefinidas para facilitar la creación de software. Muchas de estas librerías y clases se encuentran implementadas desde un inicio por defecto ya sea porque su uso es muy común o son esenciales para el funcionamiento básico de un programa en Java.

Por ejemplo: algunas clases que son importadas por defecto en cualquier proyecto java y que se pueden utilizar sin necesidad de ser importadas explícitamente tenemos:

- **String**: La clase String se encuentra en el paquete java.lang y se utiliza para manipular cadenas de caracteres en Java.
- **System**: La clase System contiene métodos y propiedades relacionadas con el sistema, como la entrada y salida estándar, propiedades del sistema, entre otros.
- **Object**: La clase Object es la superclase de todas las clases en Java y proporciona métodos comunes a todas las clases, como equals, hashCode y toString.
- **Math**: La clase Math proporciona métodos estáticos para realizar operaciones matemáticas comunes, como sin, cos, sqrt, entre otros.
- **RuntimeException**: La clase RuntimeException y sus subclases son excepciones que pueden ser lanzadas durante la ejecución de un programa sin necesidad de ser declaradas en la firma del método.

Por otra parte, la gran mayoría de clases dentro del JDK de Java necesitan ser importadas explícitamente en la clase en la que será utilizada.

Por ejemplo: Para poder pedirle información por consola al usuario, leer esa información y almacenarla en una variable; se puede utilizar la clase "Scanner". Esta clase es bastante común en programas java básicos. Si queremos utilizarla será necesario declarar este import:

```
import java.util.Scanner;
```

Si no declaramos este import en nuestra clase, no podremos hacer el uso del Scanner ya que debemos indicarle a Java que importe esa librería a nuestra clase ya que necesitamos utilizar el código que se encuentra ahí dentro.

La utilización del import ya que el desarrollador no necesita escribir la definición completa de la clase. Por lo tanto, mejora y simplifica el código del programa.

## Análisis del import y la librerías del JDK desde nuestro computador

Si bien el concepto detrás del import es bastante sencillo me gustaría especificar un poco más de cómo funciona realmente.

Tomemos de ejemplo el import de la clase “Scanner”.

```
import java.util.Scanner;
```

Al igual que la declaración del paquete, nos está indicando que la clase Scanner se encuentra en el paquete “util” del paquete “java”. Como la clase Scanner pertenece al JDK el paquete “java” se encuentra dentro del JDK.

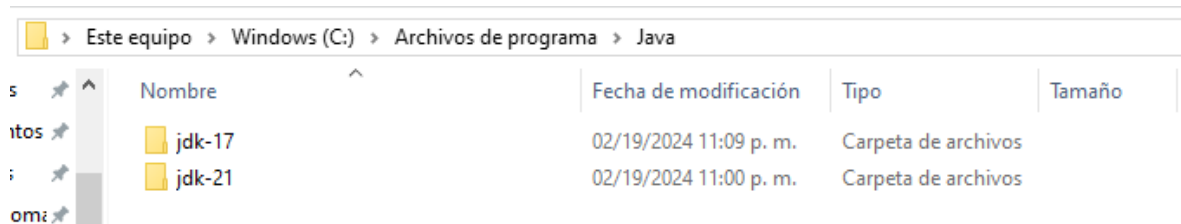
```
import java.util.Scanner;
```

Diagram illustrating the structure of the import statement:

- Paquete** (Package): java
- Subpaquetes** (Subpackages): util
- Clase** (Class): Scanner

Para ver la clase Scanner en nuestro sistema podemos ir al directorio donde tengamos descargado el JDK. En mi caso yo tengo el SDK 17 y 21 instalado en mi computador en la siguiente ruta:

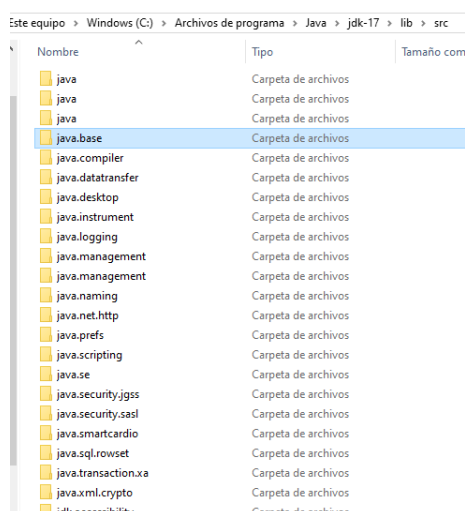
C:\Program Files\Java



Este equipo > Windows (C:) > Archivos de programa > Java				
	Nombre	Fecha de modificación	Tipo	Tamaño
	jdk-17	02/19/2024 11:09 p. m.	Carpeta de archivos	
	jdk-21	02/19/2024 11:00 p. m.	Carpeta de archivos	

Si abrimos el jdk-17 podremos ver su contenido. Vamos al directorio “lib”. Encontraremos un .zip llamado “src”. En el encontraremos todas las librerías del JDK:

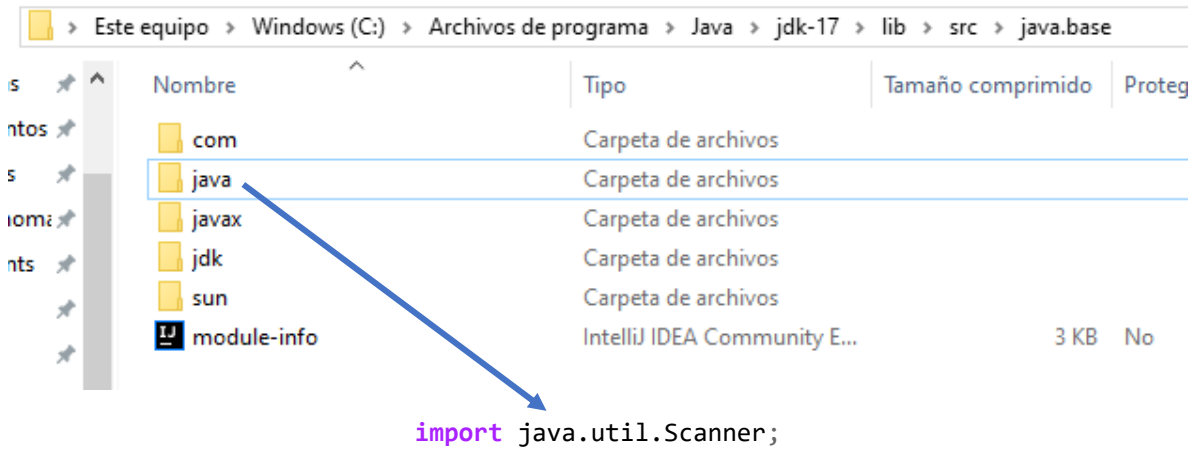
C:\Program Files\Java\jdk-17\lib\src.zip



Este equipo > Windows (C:) > Archivos de programa > Java > jdk-17 > lib > src		
Nombre	Tipo	Tamaño
java	Carpeta de archivos	
java	Carpeta de archivos	
java	Carpeta de archivos	
java.base	Carpeta de archivos	
java.compiler	Carpeta de archivos	
java.datatransfer	Carpeta de archivos	
java.desktop	Carpeta de archivos	
java.instrument	Carpeta de archivos	
java.logging	Carpeta de archivos	
java.management	Carpeta de archivos	
java.management	Carpeta de archivos	
java.naming	Carpeta de archivos	
java.net.http	Carpeta de archivos	
java.prefs	Carpeta de archivos	
java.scripting	Carpeta de archivos	
java.se	Carpeta de archivos	
java.security.jgss	Carpeta de archivos	
java.security.sasl	Carpeta de archivos	
java.smartcardio	Carpeta de archivos	
java.sql.rowset	Carpeta de archivos	
java.transaction.xa	Carpeta de archivos	
java.xml.crypto	Carpeta de archivos	

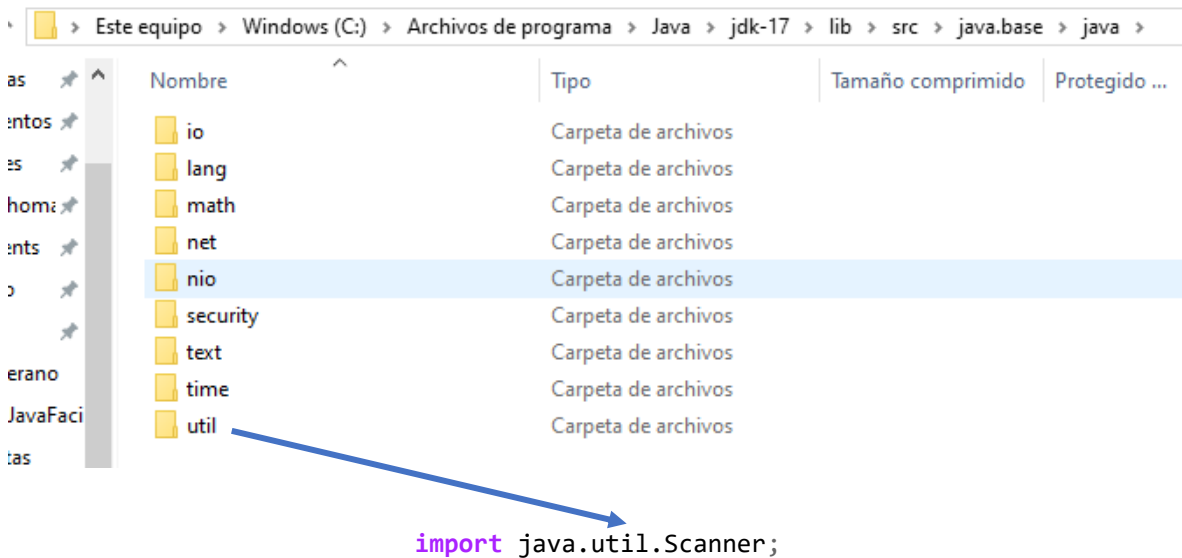
Dentro de la librería “java.base” encontraremos la librería “java”. Esta es la librería que hace referencia la declaración del import:

C:\Program Files\Java\jdk-17\lib\src.zip\java.base



Si abrimos esta carpeta veremos las librerías principales del JDK. Una de esas librerías es la “util” que es el subpaquete al que hacer referencia la declaración del import:

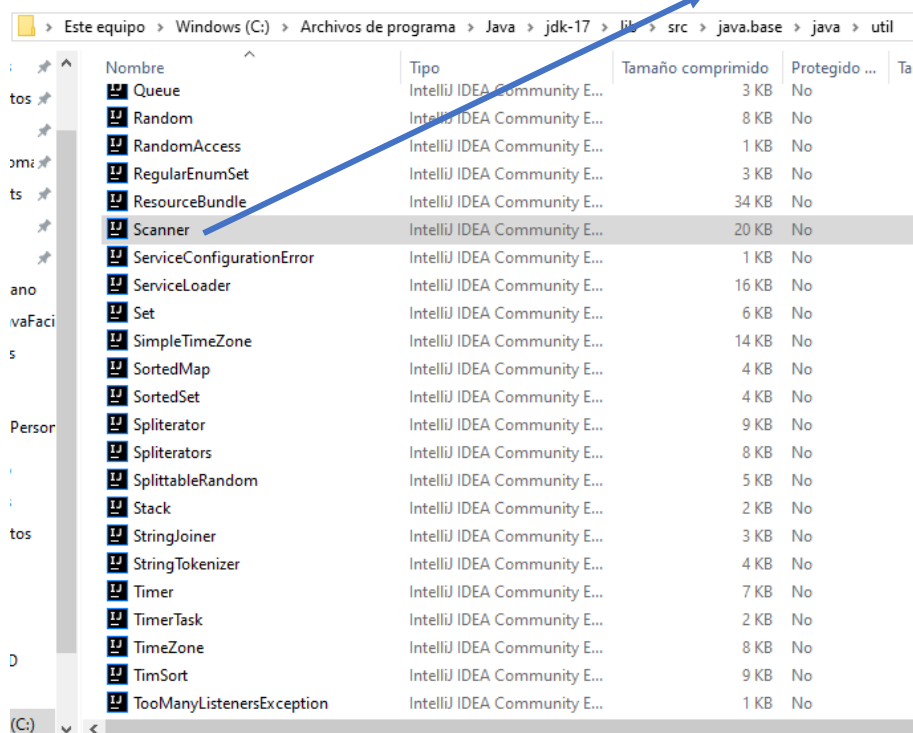
C:\Program Files\Java\jdk-17\lib\src.zip\java.base\java



Por último, cuando abrimos el paquete “util” encontraremos todas las clases que pertenecen a este paquete; entre ellas, la clase “Scanner”:

C:\Program Files\Java\jdk-17\lib\src.zip\java.base\java\util

```
import java.util.Scanner;
```



Si abrimos esta clase con el bloc de notas o con otro editor de texto podremos ver todo el código fuente que está incorporado en la clase “Scanner”.



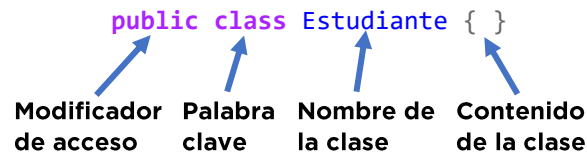
```
Scanner.java x
306  * <p>Whitespace is not significant in the above regular expressions.
307  *
308  * @since 1.5
309  */
310  public final class Scanner implements Iterator<String>, Closeable {
311
312      // Internal buffer used to hold input
313      private CharBuffer buf;
314
315      // Size of internal character buffer
316      private static final int BUFFER_SIZE = 1024; // change to 1024;
317
318      // The index into the buffer currently held by the Scanner
319      private int position;
320
321      // Internal matcher used for finding delimiters
322      private Matcher matcher;
323
324      // Pattern used to delimit tokens
325      private Pattern delimPattern;
326
327      // Pattern found in last hasNext operation
328      private Pattern hasNextPattern;
329
330      // Position after last hasNext operation
331      private int hasNextPosition;
332
333      // Result after last hasNext operation
334      private String hasNextResult;
335  }
```

Si ven es una clase ENORME. Tiene más de 3,000 líneas de código. Por esta razón el JDK es tan útil ya que el desarrollador si necesita usar alguna función de la clase “Scanner” con una simple línea de código con el import ya tiene acceso a toda la información contenida en ella.

Cuando importas una clase en Java, el import no "guarda" la clase en tu proyecto de ninguna manera. En cambio, el import simplemente le dice al compilador dónde encontrar la definición de la clase que estás utilizando.

## 2.3. Declaración de la clase

Como ya habíamos visto antes la sintaxis de la declaración de una clase en java es la siguiente:



- **public**: La clase se declara con el modificador de acceso public para que las otras clases del programa puedan tener acceso a ella. Lo más común es que las clases sean públicas, pero hay algunos casos en las que pueden ser privadas.
- **class**: palabra clave para declarar una clase.
- **Estudiante**: Nombre de la clase. Se debe seguir las reglas de nombramiento de identificadores y Upper Camel Case.

Ver *Clase 1: Elementos básicos de Java*.

### 2.3.1. Declaración de atributos

Un atributo se refiere a una variable que está asociada a una clase o a una instancia de esa clase. Los atributos describen las características o propiedades de un objeto y pueden ser de diferentes tipos de datos, como enteros, cadenas, booleanos, etc.

Sintaxis básica de declaración de un atributo de una clase:

```
[modificador_de_acceso] [tipo_de_dato] [nombre_del_atributo] ;
```

- **[modificador\_de\_acceso]**: modificador que controla la visibilidad del atributo. Puede ser:
  - public
  - protected
  - private
  - default (si no se indica ninguno)

Ver “Clase 6 Modificadores de acceso” para la explicación detallada.

- **[tipo\_de\_dato]**: Debido a que Java es un lenguaje de programación fuertemente tipado siempre que declaremos un atributo es necesario declarar explícitamente el tipo de dato de nuestro atributo. Puede ser:
  - Primitivo
    - byte, short, int, long, float, double, boolean y char.
  - Clase
    - Wrapper
      - Byte, Short, Int, Long, Float, Double, Boolean y Char
    - String, Array, Collections, otros.
    - Clase definida por el desarrollador.

Ver “Clase 2 Tipos de datos” para la explicación detallada.

- **[nombre\_del\_atributo]**: Siempre debemos asignarles un nombre a nuestros atributos. Este nombre servirá como identificador cuando lo utilicemos en métodos o clases. Se debe seguir las reglas de nombramiento de identificadores y Lower Camel Case.

*Ver Clase 1: Elementos básicos de Java.*

### Ejemplos:

- **public int numero ;**  
modificador de acceso: public  
tipo de dato: entero  
nombre del atributo: numero
- **protected String nombre ;**  
modificador de acceso: protected  
tipo de dato: Cadena  
nombre del atributo: nombre
- **Char numero ;**  
Modificador de acceso: default  
Tipo de dato: Carácter (Wrapper)  
Nombre del atributo: genero
- **private Long sueldo ;**  
Modificador de acceso: private  
Tipo de dato: Decimal Largo (Wrapper)  
Nombre del atributo: sueldo

### Encapsulamiento y atributos private

Uno de los pilares de la programación orientada a objetos es el encapsulamiento. A nivel de atributos se logra utilizando modificadores de acceso private o protected (mayor y comúnmente private) dependiendo de la herencia que se encuentre aplicada a la clase.

Se utilizan private o protected para controlar el acceso a los datos de los atributos de nuestra clase. Esto evita que otras clases puedan acceder directamente a los atributos de nuestras clases.

Para poder acceder a los valor de los atributos, se crean métodos públicos con la lógica del acceso a los datos del atributo. Estos métodos son luego implementados en la lógica del negocio/programa solo en los lugares en donde deba ser permitido acceder a esos datos.

Por ejemplo:

```

1
2 package ejemplo;
3
4 public class Estudiante {
5     //Declaración de atributos
6     private String nombre;
7     private String apellido;
8     private char genero;
9     private int edad;
10 }
11

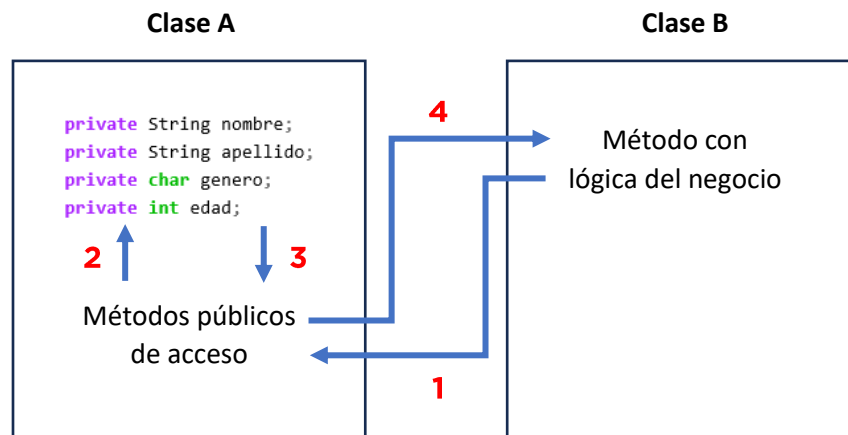
```

En la clase “Estudiante” estamos declarando cuatro atributos: “nombre”, “apellido”, “genero” y “edad”. Cada atributo tiene su respectivo tipo de dato. Todos los atributos están declarados como private.

En la clase 6 Modificadores de acceso aprendimos que al declarar private un atributo solo podremos tener acceso a sus datos desde esa misma clase. Siguiendo este principio, declararemos métodos públicos que otras clases si puedan invocar y vamos a acceder a estos cuatro atributos únicamente mediante estos métodos.

A estos métodos públicos de acceso se les denomina métodos getter y métodos setter. Ambos métodos serán explicados a profundidad más adelante en esta clase.

La relación entre los atributos private y los métodos de acceso public funciona de la siguiente manera:



1. Digamos que la clase “B” tiene un método con lógica del negocio/programa y necesita utilizar los datos de algún atributo de la clase “A”. Por encapsulamiento, la clase “B” no puede acceder directamente al contenido de los atributos de la clase “A”. Para poder acceder a esos valores necesita hacerlo mediante el uso de los métodos públicos de acceso declarados en la clase “A”.

Por lo tanto, el método en la clase “B” invoca al método público de acceso en la clase “A”.

2. El método público de acceso declarado en la clase “A” accede al valor del atributo.
3. El valor del atributo es retornado al método público de acceso.
4. El método público de acceso, ahora teniendo el valor del atributo requerido, retorna el contenido del atributo al método con lógica del negocio en la clase “B”.

### Atributos con valor inicial

Cuando declaramos un atributo también es posible inicializarlo con un valor.

Le daremos valores iniciales a nuestros atributos en los siguientes casos:

- **Valores predeterminados:** Si tienes un valor predeterminado que quieres asignar al atributo cada vez que se crea una instancia de la clase, inicializarlo al declararlo es una buena práctica. Esto evita que tengas que repetir el mismo valor en varios lugares de tu código.
- **Valores constantes:** Si el valor del atributo es constante y no va a cambiar durante la vida útil del objeto, inicializarlo al declararlo ayuda a dejar claro que es un valor fijo.
- **Valores por defecto:** Cuando un atributo tiene un valor que es comúnmente utilizado en la mayoría de los casos, inicializarlo al declararlo puede simplificar la creación de instancias de la clase.
- **Evitar valores nulos:** Si un atributo no puede ser nulo y quieres asignarle un valor por defecto en caso de que no se especifique uno, la inicialización en la declaración puede ser útil.

### Sintaxis para declarar atributos con valor inicial

```
[modificador_de_acceso] [tipo_de_dato] [nombre_del_atributo] = [valor_inicial];
```

Ejemplos:

- `private double pi = 3,1416;`
- `private String pais = "Panama";`
- `private int año = 2024;`

En general, inicializar un atributo con un valor al declararlo puede mejorar la legibilidad del código y hacer más explícito el comportamiento esperado del atributo. Sin embargo, es importante tener en cuenta que esto puede aumentar la complejidad si el valor inicial depende de la lógica más complicada o de otros elementos del programa.

El desarrollador es el que tomará la decisión sobre si el atributo tendrá valor inicial o no. Lo más común es que los atributos solo se declaren mas no se inicialicen en la misma línea de código de la clase.

### 2.3.2. Declaración de métodos

Lo métodos son todas las funciones que se declararán dentro de nuestra clase.

La estructura básica de una clase POJO consiste en los siguientes métodos:

- Métodos constructores
- Métodos Getter y Setter de cada atributo
- Método toString
- Métodos adicionales

#### 2.3.2.1. Método Constructor

El método constructor es el que tendrá la lógica de como se crearán los objetos de esa clase.

Es un método especial que se llama automáticamente cuando se crea una instancia (objeto) de esa clase. Su propósito principal es inicializar los atributos de la clase y preparar el objeto para su uso.

#### Sintaxis del método Constructor

```
[modificador_de_acceso] [nombre_de_la_clase] ( [parámetros] ) {  
    [contenido del constructor]  
};
```

- **[modificador\_de\_acceso]**: Lo más común es que los métodos constructores sean declarados como “public”. Son declarados como “public” para que se puedan crear objetos desde cualquier clase de nuestro proyecto donde sea necesario. También puede ser declarado como “protected” o “private” pero éstos se utilizan en patrones de diseño específico como “singleton”.
- **[nombre\_de\_la\_clase]**: El método constructor es un método especial de cada clase. Por lo que para declarar un método constructor, el nombre del método debe ser idéntico al nombre de la clase (incluyendo la mayúscula).
- **[parámetros]**: los parámetros son valores que se pasan a un método cuando se llama. Estos parámetros se utilizan para que el método pueda realizar su tarea de manera específica, ya que pueden contener información necesaria para que el método funcione correctamente. Los parámetros se definen en la declaración del método entre paréntesis, y pueden ser de cualquier tipo de datos válido en Java, incluidos tipos primitivos, objetos y arrays.
- **[contenido del constructor]**: Aquí escribiremos la lógica sobre cómo se inicializarán los valores de los atributos del objeto.

## Tipos de métodos constructores

Existen tres tipos de métodos constructores: Constructor vacío o por defecto, Constructor sin parámetros y Constructor con parámetros.

### Constructor vacío o por defecto

Todas las clases NECESITAN tener un método constructor. Es estrictamente necesario porque el método constructor se ejecutará siempre que se quiera crear un objeto de una clase.

Por ejemplo: Si quisiéramos crear un objeto de la clase Estudiante tendríamos que utilizar la siguiente línea de código obligatoriamente:

`Estudiante objEstudiante = new Estudiante( );`

Diagrama de anotación del código anterior:

- `Estudiante`: Nombre de la clase
- `objEstudiante`: Nombre del objeto
- `=`: Operador
- `new`: new
- `Estudiante( )`: Método constructor

Sin el método constructor no es posible crear objetos. Si tratamos de crear un objeto y ningún constructor es invocado entonces ocurrirá un error de compilación y el programa no se ejecutará.

Para evitar este problema, Java siempre tendrá un constructor vacío por defecto para todas las clases que se creen.

Este Constructor vacío es **implícito**; es decir, aunque nosotros como desarrolladores no declaremos ningún método constructor, java provee un método constructor vacío a la hora de compilar el programa.

Java crea el constructor por defecto durante la compilación, no durante la ejecución del programa. Cuando se compila una clase y no se define ningún constructor, el compilador de Java automáticamente agrega un constructor por defecto público y sin argumentos a la clase compilada. Este proceso es parte del proceso de compilación de Java y se realiza antes de que el programa se ejecute.

### Ejemplo

```
1
2 package ejemplo;
3
4 public class Estudiante {
5     //Declaración de atributos
6     private String nombre;
7     private String apellido;
8     private char genero;
9     private int edad;
10
```

```
11 //Declaración de constructores
12 //Constructor por defecto
13 public Estudiante() { }
14
15 }
16
```

En la línea 13, estamos declarando explícitamente el constructor por defecto de la clase “Estudiante”. No tiene parámetros ni ninguna implementación. No es obligatorio declararlo explícitamente ya que como explicamos el compilador lo utilizará aunque no esté declarado; sin embargo, es parte de buenas practicas siempre declararlo para tener código más limpio y fácil de leer.

Cuando utilizamos el constructor por defecto en la instanciación de un objeto de una clase todos sus atributos serán inicializados con valores predeterminados.

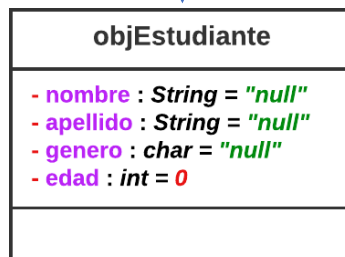
Los valores predeterminados dependen del tipo de dato:

Tipo de Dato	Valor predeterminado
N Numérico	0
Boolean	False
Clases/Referencia	null

Podríamos ver cómo funciona mediante el siguiente diagrama:

```
Estudiante objEstudiante = new Estudiante( );
```

```
public Estudiante() { }
```



Cuando se invoca el método constructor “Estudiante()” desde la instanciación, se ejecuta el método constructor por defecto. Debido a que estamos usando el constructor por defecto (que está vacío) se creará un objeto donde todos sus atributos tendrán valores por defecto.



### Constructor sin parámetros

Podemos implementar otro tipo de método constructor en el que tenemos la libertad de inicializar todas las variables con algún valor o solo inicializar las variables que necesitamos.

```
1
2 package ejemplo;
3
4 public class Estudiante {
5     //Declaración de atributos
6     private String nombre;
7     private String apellido;
8     private char genero;
9     private int edad;
10
11     //Declaración de constructores
12     //Constructor sin parámetros
13     public Estudiante() {
14         this.nombre = "Omar";
15         this.apellido = "Montoya";
16         this.genero = 'M';
17         this.edad = 27;
18     }
19
20 }
21
```

En este ejemplo estamos utilizando el constructor “Estudiante()” sin parámetros. Si comparamos este constructor con el constructor por defecto son exactamente iguales. La diferencia radica en que en este constructor nosotros sí estamos inicializando los valores de los atributos de la clase.

También podemos notar algo distinto que no habíamos utilizado antes: **this**

“**this**” es una palabra clave que hace referencia al objeto actual en el que se está trabajando dentro de un método o constructor de una clase. Se utiliza principalmente para hacer referencia a variables de instancia (atributos) y métodos de la clase actual.

En nuestro código “**this**” hace referencia a los atributos de nuestra clase. El uso de “**this**” es una buena practica ya que veremos más adelante que hay ocasiones en la que podríamos tener dos variables con el mismo nombre; sin embargo, se refieren a variables distintas (Lo veremos en el constructor con parámetros).

El uso de `this.nombre` hace más claro que nos estamos refiriendo al atributo "nombre" de nuestra clase. En el caso de que exista otra variable con el mismo nombre, el compilador sabrá a cuál de las dos nos estamos refiriendo

```
//Declaración de atributos
private String nombre;
private String apellido;
private char genero;
private int edad;

//Constructor sin parámetros
public Estudiante() {
    this.nombre = "Omar";
    this.apellido = "Montoya";
    this.genero = 'M';
    this.edad = 27;
}
```

Utilizando este método constructor tendremos:

```
Estudiante objEstudiante = new Estudiante( );
```

```
public Estudiante() {
    this.nombre = "Omar";
    this.apellido = "Montoya";
    this.genero = 'M';
    this.edad = 27;
}
```

objEstudiante
<ul style="list-style-type: none"> <li>- nombre : String = "Omar"</li> <li>- apellido : String = "Montoya"</li> <li>- genero : char = 'M'</li> <li>- edad : int = 27</li> </ul>

### Constructor con parámetros

Es el tipo de constructor más común.

Declaramos el método constructor, pero le vamos a pasar los valores iniciales que queremos asignarle a cada parámetro al momento de crear el objeto.

```

1  .
2  package ejemplo;
3
4  public class Estudiante {
5      //Declaración de atributos
6      private String nombre;
7      private String apellido;
8      private char genero;
9      private int edad;
10
11     //Declaración de constructores
12     //Constructor con parámetros
13     public Estudiante(String nombre, String apellido, char genero, int edad) {
14         this.nombre = nombre;
15         this.apellido = apellido;
16         this.genero = genero;
17         this.edad = edad;
18     }
19
20 }
21 .

```

En los constructores con parámetros tenemos que declarar variables locales dentro de los paréntesis del método separados por comas. Debemos declarar el tipo de dato de la variable local y un nombre para la variable local. Por buenas prácticas se nombra la variable local con el mismo nombre que el atributo al que será asignado:

```

public Estudiante(String nombre, String apellido, char genero, int edad)

```

↑
↑
↑
↑

**Parámetro**    **Parámetro**    **Parámetro**    **Parámetro**  
**#1**            **#2**            **#3**            **#4**

Estas variables locales o parámetros serán asignados a su respectivo atributo:

```

public Estudiante(String nombre, String apellido, char genero, int edad) {
    this.nombre = nombre;
    this.apellido = apellido;
    this.genero = genero;
    this.edad = edad;
}

```

viéndolo como sería aplicado en un programa java:

**1. Los valores iniciales son definidos directamente en el método constructor en la sentencia de instanciación del objeto.**

```
Estudiante objEstudiante = new Estudiante("Omar", "Montoya", 'M', 27);
```

```
public Estudiante(String nombre, String apellido, char genero, int edad)
```

**2. Cada valor es mapeado a su respectiva variable local. El mapeo se hace según la posición de cada argumento**

```
public Estudiante(String nombre = "Omar", String apellido = "Montoya",  
                  char genero = 'M', int edad = 27)
```

**3. De esta manera se le asigna los valores de las variables locales a los atributos de la clase**

```
public Estudiante(String nombre, String apellido, char genero, int edad) {  
    this.nombre = "Omar";  
    this.apellido = "Montoya";  
    this.genero = 'M';  
    this.edad = 27;  
}
```

**4. Con estos valores iniciales asignados a los atributos entonces el objeto es creado**

objEstudiante
<ul style="list-style-type: none"> <li>- nombre : String = "Omar"</li> <li>- apellido : String = "Montoya"</li> <li>- genero : char = 'M'</li> <li>- edad : int = 27</li> </ul>

### 2.3.2.2. Métodos Getter y Setter

En Java, los métodos getter y setter son métodos especiales utilizados para acceder y modificar los valores de los campos de una clase, respectivamente.

Estos métodos complementan el principio de encapsulamiento que discutimos en esta clase cuando vimos los atributos declarados como private.

#### Método Getter

Un método getter, también conocido como método de acceso, es un método que devuelve el valor de un campo de una clase. Su nombre generalmente comienza con "get" seguido del nombre del campo al que accede.

#### Sintaxis de un método Getter

```
public [valor_de_retorno] [nombre_del_método] ( ) {  
    return nombre_del_atributo;  
}
```

- **public**: declararemos los métodos de acceso Getter como public para poder acceder a los valores de los atributos en otras clases.
- **[valor\_de\_retorno]**: Siempre que declaramos un método hay que especificar el valor de retorno. El valor de retorno consiste en que tipo de dato será retornado por el método. El método Getter se utiliza para retornar el valor de cada atributo por lo que el valor de retorno será igual al tipo de dato del atributo del método getter.
- **[nombre\_del\_método]**: En teoría el nombre del método puede ser cualquiera que el desarrollador desee siempre y cuando cumpla con la función esperada. Sin embargo, por convención y buenas practicas el nombre del método getter de un atributo empieza con la palabra "get" (de "obtener" en inglés) junto con el nombre del atributo que se desea retornar (Por ejemplo: getNombre, getApellido, getEdad, etc).
- **return**: es una palabra clave que se utiliza en métodos para declarar la sentencia de retorno de un método. Sirve para devolver un valor y salir del método. Cuando se ejecuta una instrucción return, el flujo de control de la ejecución del programa sale inmediatamente del método en el que se encuentra y devuelve el valor especificado, si es necesario.
- **nombre\_del\_atributo**: Como es un método Get, el valor de retorno siempre será el atributo que estemos declarando su método getter.

## Aplicación

```
1
2 package ejemplo;
3
4 public class Estudiante {
5     //Atributos
6     private String nombre;
7     private String apellido;
8     private char genero;
9     private int edad;
10
11     //Constructor
12     public Estudiante(String nombre, String apellido, char genero, int edad) {
13         this.nombre = nombre;
14         this.apellido = apellido;
15         this.genero = genero;
16         this.edad = edad;
17     }
18
19     //Getter atributo nombre
20     public String getNombre(){
21         return nombre;
22     }
23
24     //Getter atributo apellido
25     public String getApellido() {
26         return apellido;
27     }
28
29     //Getter atributo genero
30     public char getGenero() {
31         return genero;
32     }
33
34     //Getter atributo edad
35     public int getEdad() {
36         return edad;
37     }
38 }
39
```

Según el principio de encapsulamiento, cada atributo private declarado debe tener un método público getter declarado.

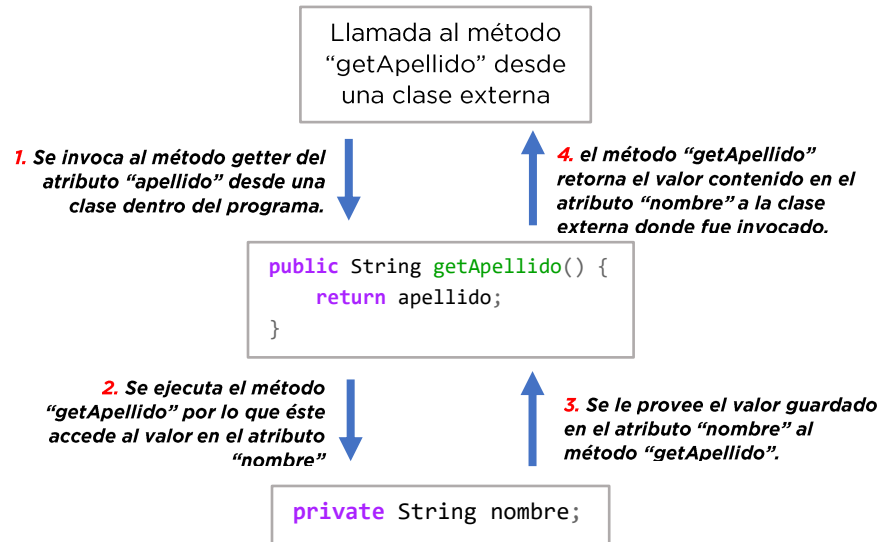
En nuestra clase “Estudiante” tenemos cuatro atributos: “nombre”, “apellido”, “genero” y “edad”; cada uno declarado como “private”. Por lo que de esta misma manera, debemos tener declarado cuatro métodos getter, uno para cada atributo.



Cuando una clase externa necesite acceder al contenido de uno de estos cuatro atributos, se hará el llamado al método getter de ese atributo.

Por ejemplo:

Si necesito acceder el valor contenido en el atributo “apellido” en vez de hacer un llamado directo a ese atributo, se hace un llamado al método “getApellido” y éste es el que retornará el valor contenido en “apellido”.



## Método Setter

Un método setter (del inglés “set” que significa “establecer”), también conocido como método de modificación, es un método que establece el valor de un campo de una clase.

De la misma manera en que una clase no puede acceder a los valores de un atributo, **tampoco puede modificar sus valores**.

Si el atributo “genero” tiene el valor ‘M’ cuando se creó el objeto, una clase externa no pueda llamarlo directamente y asignarle un nuevo valor ‘F’ debido a que el atributo es privado.

En el caso de que sea necesario y esté dentro de la lógica del negocio/programa cambiar el valor inicial de un atributo, se utilizará el método Setter de este atributo para cambiar correctamente el valor del atributo.

## Sintaxis de un método Setter

```
public void [nombre_del_método] ( parámetro  
                                ([tipo_de_dato] [nombre_del_parámetro]) ) {  
    this.[nombre_del_atributo] = [nombre_del_parámetro];  
}
```

- **public:** Se debe declarar los métodos Setter como “public” para que las otras clases puedan modificar los valores de los atributos, según la lógica del negocio/programa.
- **void:** Se utiliza para indicar que el método no retornará ningún valor. Todos los métodos deben indicar el valor de retorno del método. En algunos métodos, no se requiere retornar ningún valor, sino ejecutar algún proceso. Los métodos setter realizan el cambio del valor del atributo por uno nuevo, por lo que no deben retornar ningún valor.
- **[nombre\_del\_método]:** En los métodos setter se utiliza el prefijo “set” seguido del nombre del atributo a que corresponde el método. Por ejemplo: para el método “edad” su método setter debe llamarse “setEdad”.
- **[tipo\_de\_dato]:** Para declarar el parámetro debemos de indicar el tipo de dato de la variable que declararemos dentro del parámetro. Este tipo de dato debe coincidir con el tipo de dato del atributo que se retornará.
- **[nombre\_del\_parámetro]:** Para declarar el parámetro también se debe declarar el nombre de la variable local que funcionará como parámetro. Esta variable debe ser nombrada por convención con el mismo nombre del atributo.
- **this.[nombre\_del\_atributo] = [nombre\_del\_parámetro]:** se hará el uso de la palabra clave “this” para hacer referencia al atributo declarado en la clase. Utilizando el signo de igual “=” se define la sentencia que asignará el valor nuevo al atributo.



## Aplicación

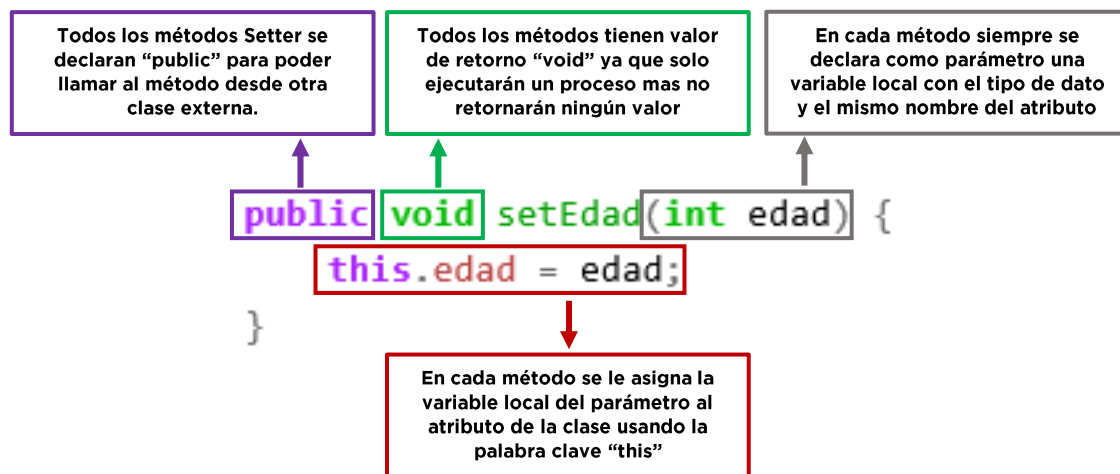
```
1
2 package ejemplo;
3
4 public class Estudiante {
5     //Atributos
6     private String nombre;
7     private String apellido;
8     private char genero;
9     private int edad;
10
11     //Constructor
12     public Estudiante(String nombre, String apellido, char genero, int edad) {
13         this.nombre = nombre;
14         this.apellido = apellido;
15         this.genero = genero;
16         this.edad = edad;
17     }
18
19     //Getter atributo nombre
20     public String getNombre(){
21         return nombre;
22     }
23
24     //Getter atributo apellido
25     public String getApellido() {
26         return apellido;
27     }
28
29     //Getter atributo genero
30     public char getGenero() {
31         return genero;
32     }
33
34     //Getter atributo edad
35     public int getEdad() {
36         return edad;
37     }
38
39     //Setter atributo nombre
40     public void setNombre(String nombre) {
41         this.nombre = nombre;
42     }
43
44     //Setter atributo apellido
45     public void setApellido(String apellido) {
46         this.apellido = apellido;
47     }
48 }
```

```

48
49 //Setter atributo genero
50 public void setGenero(char genero) {
51     this.genero = genero;
52 }
53
54 //Setter atributo edad
55 public void setEdad(int edad) {
56     this.edad = edad;
57 }
58
59 }
60

```

Como vemos para cada atributo se tiene declarado un método Setter.



Por ejemplo:

Tenemos un objeto de la clase "Estudiante" llamado "objEstudiante". Este objeto cuando fue creado utilizando el método constructor, su atributo "edad" se le dio un valor inicial de 27. Ahora, según la lógica del negocio/aplicación, una clase "B" necesita modificar ese valor y cambiarlo de 27 a 30.

Como "edad" es "private" debido al encapsulamiento, la clase "B" no puede llamar directamente a la variable y modificar su valor, si no que debe llamar al método "Setter" de ese atributo llamado "setEdad" y proveer la nueva edad que se desea asignar al atributo de la siguiente manera:

Instanciación de la clase Estudiante utilizando constructor con parámetros:

```
Estudiante objEstudiante = new Estudiante("Omar", "Montoya", 'M', 27);
```

1. Se crea el objeto "objEstudiante" con los valores iniciales indicados en el método constructor

objEstudiante
- nombre : String = "Omar" - apellido : String = "Montoya" - genero : char = 'M' - edad : int = 27

2. Una vez creado se hace el llamado del método setter del atributo "edad" para modificar su valor.

Llamada al método Setter del atributo "edad" desde una clase externa:

```
objEstudiante.setEdad(30);
```

3. Se ejecuta el método "setEdad" utilizando como parámetro el número entero 30 que fue indicado en la llamada al método.

```
public void setEdad(30) {  
    this.edad = 30;  
}
```

4. El método al finalizar su ejecución, cambia el valor de 27 a 30 para el objeto "objEstudiante"

objEstudiante
- nombre : String = "Omar" - apellido : String = "Montoya" - genero : char = 'M' - edad : int = 30

### 2.3.2.3. Método toString

El método toString() en Java se utiliza para devolver una representación de cadena de un objeto.

#### Sintaxis

##### @Override

```
public String toString() {  
    return "NombreDeLaClase{" +  
        "NombreAtributo1=" + atributo1 + '\n' +  
        "NombreAtributo2=" + atributo2 + '\n' +  
        [...]  
        '}' ;  
}
```

- **@Override**: Es una anotación que se utiliza para indicar que el método declarado está sobre escribiendo un método que originalmente está declarado en la superclase o clase padre. En este caso la clase padre es la clase "Object". La clase "Object" será explicada en clases siguientes.
- **public**: Al igual que los métodos getter y setter, el método toString necesita ser público para ser accedido por clases externas.
- **String**: Es el valor de retorno del método. Como el método toString se utiliza para devolver una representación de cadena de un objeto, se debe especificar que será una cadena su valor de retorno.
- **toString**: A diferencia de los métodos Getter o Setter, que dependiendo del atributo su nombre cambia, el método "toString" siempre tendrá ese mismo nombre. Esta razón es debido a que para cada clase se definirá solo un método toString normalmente.
- **return**: El método retornará una concatenación de distintas cadenas con el nombre y valores de cada atributo.

#### Aplicación

```
1  
2 package ejemplo;  
3  
4 public class Estudiante {  
5     //Atributos  
6     private String nombre;  
7     private String apellido;  
8     private char genero;  
9     private int edad;  
10
```

```
11 //Constructor
12 public Estudiante(String nombre, String apellido, char genero, int edad) {
13     this.nombre = nombre;
14     this.apellido = apellido;
15     this.genero = genero;
16     this.edad = edad;
17 }
18
19 //Getter atributo nombre
20 public String getNombre(){
21     return nombre;
22 }
23
24 //Getter atributo apellido
25 public String getApellido() {
26     return apellido;
27 }
28
29 //Getter atributo genero
30 public char getGenero() {
31     return genero;
32 }
33
34 //Getter atributo edad
35 public int getEdad() {
36     return edad;
37 }
38
39 //Setter atributo nombre
40 public void setNombre(String nombre) {
41     this.nombre = nombre;
42 }
43
44 //Setter atributo apellido
45 public void setApellido(String apellido) {
46     this.apellido = apellido;
47 }
48
49 //Setter atributo genero
50 public void setGenero(char genero) {
51     this.genero = genero;
52 }
53
54 //Setter atributo edad
55 public void setEdad(int edad) {
56     this.edad = edad;
57 }
58
59 //ToString
```

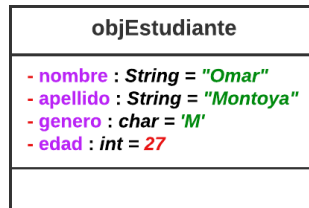
```

60  @Override
61  public String toString() {
62      return "Estudiante{" +
63          "nombre='" + nombre + '\'' +
64          ", apellido='" + apellido + '\'' +
65          ", genero=" + genero +
66          ", edad=" + edad +
67          "'}";
68  }
69  }
70

```

Instanciación de la clase Estudiante utilizando constructor con parámetros:  
 Estudiante objEstudiante = new Estudiante("Omar", "Montoya", 'M', 27);

1. Se crea el objeto "objEstudiante" con los valores iniciales indicados en el método constructor



2. Una vez creado se hace el llamado al método "toString" de la clase "Estudiante"

Invocación del método toString e imprimiéndolo mediante el método print:  
 System.out.print(objEstudiante.toString());

3. Se imprime por consola el método toString con los datos del objeto estudiante

```

"C:\Program Files\Java\jdk-17\bin\java.exe"...
Estudiante{nombre='Omar', apellido='Montoya', genero=M, edad=27}

Process finished with exit code 0

```