

## Clases

### 1. Introducción

Ahora que hemos revisado algunas de las sentencias fundamentales presentes en el lenguaje Java, profundizaremos en la correlación entre las clases, detallando su sintaxis en el contexto del lenguaje Java.

Por ejemplo, si tenemos un sistema de almacenamiento y gestión de cursos, docentes y estudiantes, tendremos en nuestro sistema la representación de la clase Curso, la clase Docente y la clase Estudiante. Debemos, además, conocer algunos datos de interés de estos en función de la realidad a modelar.

Las clases, son la definición formal de una entidad de la realidad. Son las que nos definen el “template” de las entidades, nos definen “la plantilla” de su estructura. La misma, se compone por: nombre, atributos, métodos y relaciones con otras clases.

Cada clase java, va estar en un archivo “.java” independiente y debemos hacer coincidir el nombre de nuestra clase con el nombre de nuestro archivo. Es decir, la clase Curso debe estar definida en el archivo Curso.java, la clase Estudiante se debe encontrar en el archivo Estudiante.java.

Además, tenemos siempre una clase que auspicia de “clase inicial” del programa, que debe ser el punto de entrada de nuestro programa. Esta, es la que siempre tiene el método “public static void main(String[] args)” el cual hemos visto en ejemplos anteriores.

Al escribir un programa en un lenguaje orientado a objetos tratamos de modelar un problema del mundo real pensando en objetos que forman parte del problema y que se relacionan entre sí.

Clase: Plantilla o modelo para la creación de objetos. El nombre de una clase, por convención comienza en mayúscula.

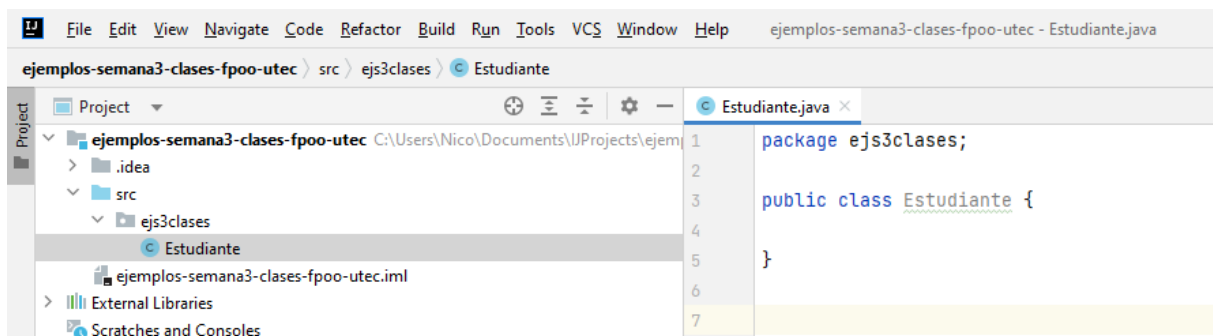
Una clase tiene:

- Atributos.
- Relaciones con otras clases.
- Métodos que puede realizar.

Objeto o instancia de una clase: Entidad existente en la memoria del ordenador que representa a un individuo en particular que pertenece a una clase. Por ejemplo, un objeto representa a Juan que es un ejemplo en particular de la clase Estudiante.

Todo objeto es instancia de alguna clase. Es importante destacar que cada objeto es una instancia única de alguna clase, reflejando así la singularidad y las propiedades particulares asociadas con la entidad que representa.

Veamos un ejemplo, donde definimos la clase llamada “Estudiante”.



```
1 package ejs3clases;
2
3 public class Estudiante {
4
5 }
6
7
```

- Como se puede ver en imagen, dentro del paquete “programa”, ahora tenemos dos archivos, “Programa.java” donde se encuentra el método main y luego el archivo nuevo que se llama “Estudiante.java”.
- En él, por el momento tenemos solamente la definición de la clase que se llama “Estudiante”.

Centrémonos ahora, únicamente en el archivo “Estudiante.java” para poder analizar los diferentes componentes que tiene una Clase y luego, veremos cómo desde el método main que se encuentra en Programa utilizaremos la clase Estudiante.

## 2. Estructura de Clase

Las clases tienen un nombre, que por convención comienza con mayúscula: Estudiante. Si es un nombre compuesto por más de una palabra, las palabras sucesivas comienzan también con mayúscula: NombreDeClaseCorrecto.

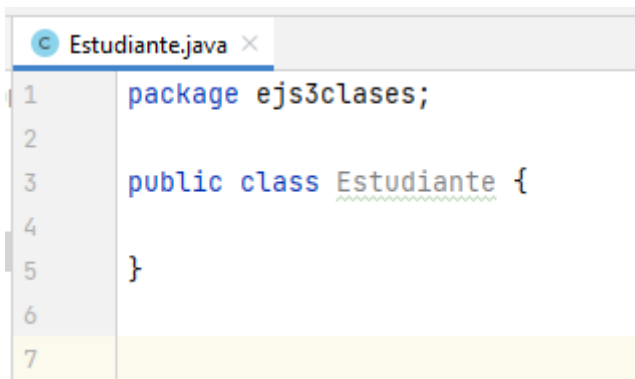
Tienen también tres secciones: La primera donde se definen los atributos, la segunda donde definimos los constructores y luego, los métodos que esta puede realizar.

Veamos el siguiente esquema:

```
/* Ejemplo Clase Estudiante */  
Clase Estudiante {  
  Propiedades:  
    Identificador del estudiante  
    Nombre completo  
    Cédula de identidad  
    Año de nacimiento  
  
  Constructor:  
    Crear Estudiante  
  
  Operaciones disponibles:  
    Asignar un identificador  
    Asignar un nombre  
    Asignar cédula de identidad  
    Asignar año de nacimiento  
    Calcular edad  
}
```

## 2.1 Atributos y Constructores

Partiremos de una clase “Estudiante” vacía:

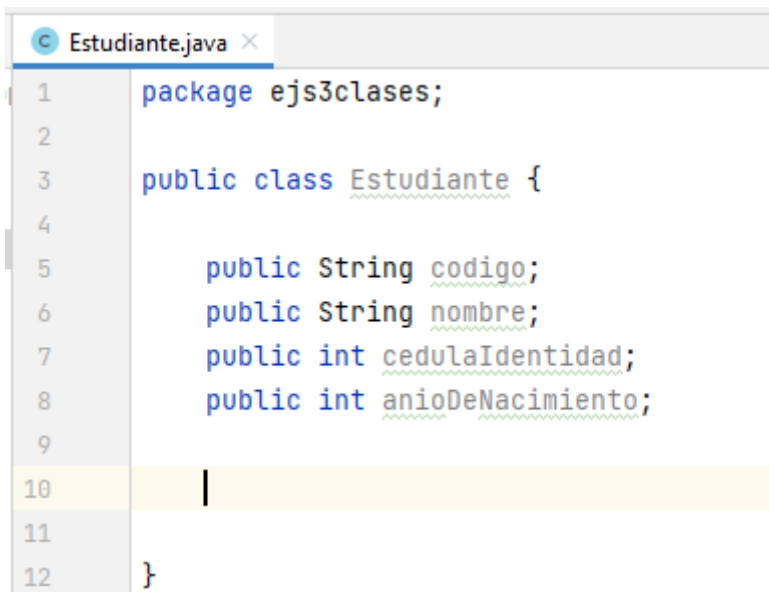


```
Estudiante.java x  
1 package ejs3clases;  
2  
3 public class Estudiante {  
4  
5 }  
6  
7
```

Todo comportamiento o características de la clase, deberá ir dentro de las llaves que la definen (su bloque de código). Si colocamos algo fuera, este código no va a pertenecer a la clase.

Ahora agregaremos atributos a Estudiante: Nombre, Código, Año de nacimiento y cédula de identidad.

Nos quedaría lo siguiente:



```
1 package ejs3clases;
2
3 public class Estudiante {
4
5     public String codigo;
6     public String nombre;
7     public int cedulaIdentidad;
8     public int anioDeNacimiento;
9
10
11
12 }
```

Como se puede ver, dentro de las llaves, definimos 4 atributos de esta clase Estudiante, su definición es muy similar a la que vimos con las variables: tienen un nombre y un tipo de datos. Por convención arrancan con minúscula.

Los atributos serían variables propias de los objetos de la clase, y dentro del bloque de la clase se puede utilizar/modificar cuantas veces se quiera.

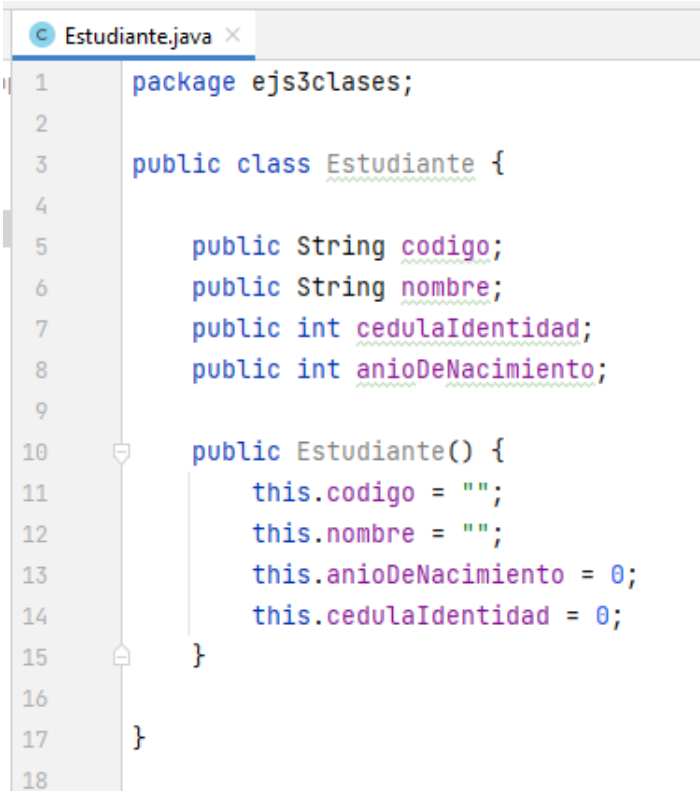
En el ejemplo, está la palabra `public` delante, esto lo veremos más adelante.

Hasta ahora tenemos la declaración de una clase, con sus atributos, pero falta tener un constructor de objetos de Estudiante.

El constructor, es el que nos permite generar instancias u objetos de la clase. Instanciar es tomar la plantilla de la clase y darles valores a sus atributos.

Es un método particular de la clase, ya que el nombre del método constructor debe coincidir con el nombre de la clase, puede tener sí parámetros de entrada, pero no tiene tipo de retorno (ni void). Debe ser de acceso público.

Dentro del constructor se inicializan los valores de los atributos de la clase, esto puede ser con valores por defecto, u con valores que se pasan como parámetro.



```
1 package ejs3clases;
2
3 public class Estudiante {
4
5     public String codigo;
6     public String nombre;
7     public int cedulaIdentidad;
8     public int anioDeNacimiento;
9
10    public Estudiante() {
11        this.codigo = "";
12        this.nombre = "";
13        this.anioDeNacimiento = 0;
14        this.cedulaIdentidad = 0;
15    }
16
17 }
18
```

Debemos colocar un método en la clase, que sea de la forma “public NombreDeLaClase()”. Luego en el bloque de este método, inicializar los atributos con los valores que se desea, de

esta forma, cuando se cree una instancia de una clase del tipo Estudiante, sus atributos tienen dicho valor.

Se puede ver dentro del constructor que para hacer referencia a un atributo de la clase se utiliza la palabra `this`. Esto se utiliza para especificar que se están usando elementos que se encuentran dentro de la clase.

Por ejemplo, si quiero acceder al atributo código de mi instancia se puede escribir:  
`this.codigo`.

El uso de `this`, no es obligatorio, utilizando únicamente el nombre del atributo se puede acceder de igual manera, es recomendable utilizarlo para dejar en claro que dicho atributo es el de la clase.

En este caso, dentro del constructor vamos accediendo uno a uno los atributos y les asignamos un valor por defecto.

Una clase puede tener más de un constructor, siempre y cuando tengan distintos parámetros de entrada:

```
public Estudiante() {  
  
    public Estudiante(String codigo, String nombre, int cedulaIdentidad, int anioDeNacimiento) {
```

```
1 package ejs3clases;
2
3 public class Estudiante {
4
5     public String codigo;
6     public String nombre;
7     public int cedulaIdentidad;
8     public int anioDeNacimiento;
9
10    public Estudiante() {
11        this.codigo = "";
12        this.nombre = "";
13        this.anioDeNacimiento = 0;
14        this.cedulaIdentidad = 0;
15    }
16
17    public Estudiante(String codigo, String nombre, int cedulaIdentidad, int anioDeNacimiento) {
18        this.codigo = codigo;
19        this.nombre = nombre;
20        this.cedulaIdentidad = cedulaIdentidad;
21        this.anioDeNacimiento = anioDeNacimiento;
22    }
23
24 }
25
```

En este ejemplo, nuestra clase tiene dos constructores:

- uno sin parámetros de entrada, que le pone los valores iniciales por defecto a los atributos.
- otro, con 4 parámetros de entrada que corresponden a los valores que queremos que tengan inicialmente los atributos. Y dentro de este constructor se asignan uno a uno estos valores. En este caso el nombre de atributo coincide con el nombre de parámetro, pero esto no es obligatorio. ¿Al utilizar el nombre del parámetro igual al nombre del atributo, como se hace para diferenciar uno de otro al llamarse igual? Si queremos utilizar el atributo utilizamos this.

### ¿Cuál es el fin de una clase?

Utilizarla como un tipo de dato complejo, y poder utilizar distintas variables con este tipo de dato.



Vimos que el programa en el que estamos tiene un main en el archivo Programa.java. Entonces, es desde ese método que se quiere crear instancias de la clase Estudiante.

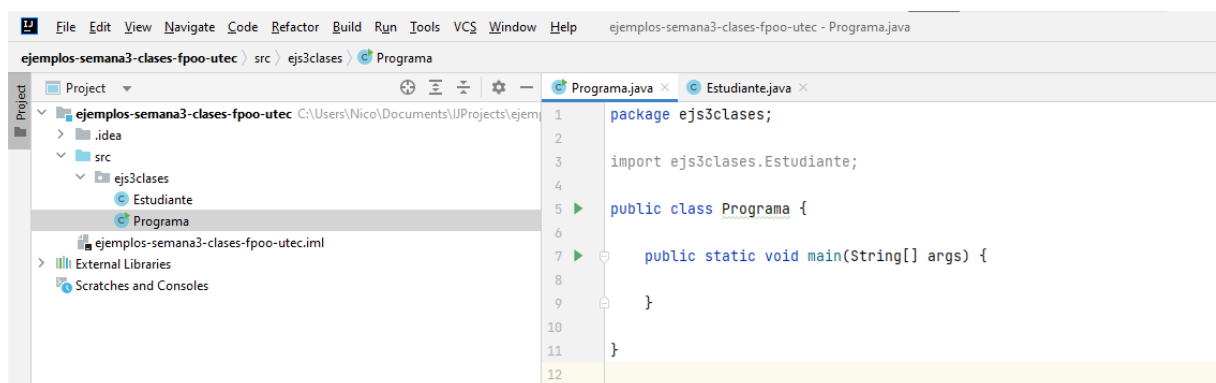
Lo primero que se debe hacer, es agregar una referencia en Programa que se desea utilizar la clase Estudiante.

Esto en Java se hace con la palabra import y seguido el paquete y la clase (“nombre\_de\_paquete.nombre\_de\_clase”) deseada:

```
import ejs3clases.Estudiante;
```

Los import se colocan luego del package y antes de la declaración de la clase (esto no es necesario en el caso de que ambas clases se encuentren en el mismo paquete):

Veamos ahora un ejemplo de cómo utilizar la clase estudiante para crear instancias de la misma dentro del método main.



- Para esto debemos declarar una variable de tipo Estudiante:

```
Estudiante e1;
```

- Inicializarla, llamando a su constructor:

```
e1 = new Estudiante();
```

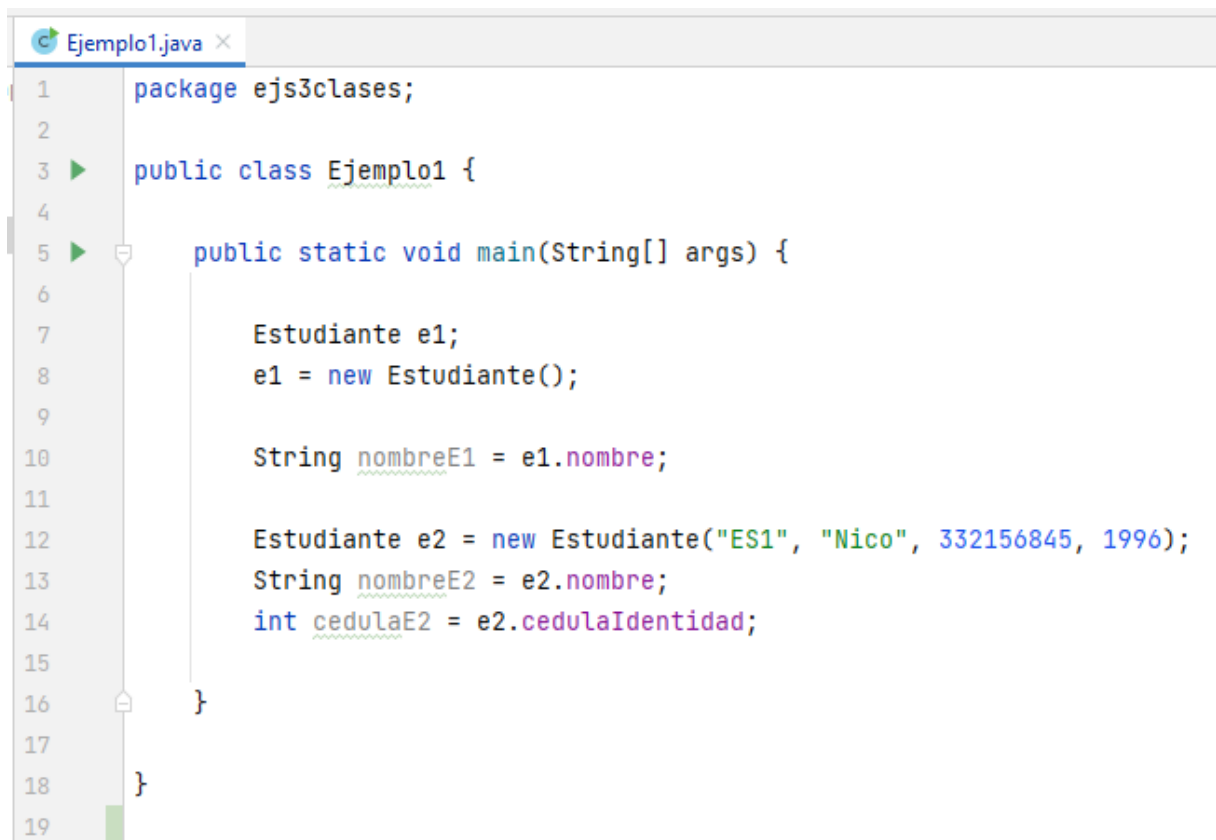
Notar que al ser de tipo complejo debemos utilizar new.

Ahora ya tenemos una variable de tipo de dato Estudiante, pero cómo podemos acceder/modificar los valores de sus atributos?

nombreVariable.nombreAtributo

### Ejemplo 1

➤ En el ejemplo, hemos creado dos instancias distintas de estudiantes. Para ello, hemos definido dos variables de tipo Estudiante, llamadas e1 y e2.



```
1 package ejs3clases;
2
3 public class Ejemplo1 {
4
5     public static void main(String[] args) {
6
7         Estudiante e1;
8         e1 = new Estudiante();
9
10        String nombreE1 = e1.nombre;
11
12        Estudiante e2 = new Estudiante("ES1", "Nico", 332156845, 1996);
13        String nombreE2 = e2.nombre;
14        int cedulaE2 = e2.cedulaIdentidad;
15
16    }
17
18 }
19
```

➤ En el caso de e1 hemos usado su constructor por defecto (sin parámetros) y en el caso de e2 hemos usado el constructor con parámetros.

- Una vez, que tenemos la instancia de nuestros objetos, podemos utilizar `nombreVariable.nombreAtributo` para poder obtener/asignar el valor que tiene dicha instancia un atributo dado.
- ¿Qué valor se va a almacenar en la variable `nombreE1`?  
El String vacío (`""`). Esto, es porque utilizamos el constructor por defecto, que si vemos como estaba implementado, le asignaba al atributo `nombre` el string vacío.
- ¿Qué valor se almacenará en la variable `cedulaE2`?  
La respuesta es 332156845, puesto que es el valor pasado como parámetro en el constructor para el atributo `cedulaIdentidad` del objeto `e2`.
- Esto se debe a que se crean dos instancias de “Estudiante” distintas, que, si bien se comportan iguales, los valores que almacenan son independientes de cada una.

## 2.2 Métodos

Una clase puede tener métodos propios.

En el caso de la clase `Estudiante` podríamos tener un método que haga el cálculo de la edad. Al cual se le pase por parámetro el año actual y dentro del método realizar la resta `año actual - año de nacimiento` y devolver este número. Por convención, los métodos arrancan con minúscula.

## Ejemplo 2

```
Estudiante.java x
1 package ejs3clases;
2
3 public class Estudiante {
4
5     public String codigo;
6     public String nombre;
7     public int cedulaIdentidad;
8     public int anioDeNacimiento;
9
10    public Estudiante() {
11        this.codigo = "";
12        this.nombre = "";
13        this.anioDeNacimiento = 0;
14        this.cedulaIdentidad = 0;
15    }
16
17    public Estudiante(String codigo, String nombre, int cedulaIdentidad, int anioDeNacimiento) {
18        this.codigo = codigo;
19        this.nombre = nombre;
20        this.cedulaIdentidad = cedulaIdentidad;
21        this.anioDeNacimiento = anioDeNacimiento;
22    }
23
24    //Ejemplo 2.
25
26    public int calcularEdad(int anioActual){
27        return anioActual - this.anioDeNacimiento;
28    }
29
30 }
```

➤ El algoritmo para calcular la edad de un estudiante sería, dado el año actual, pasado por parámetro, restarle el año de nacimiento del estudiante, y esto nos dará la edad.

➤ Por lo tanto, definimos en la clase “Estudiante” un método “calcularEdad” que retorne un int. El resultado será restar la variable añoActual menos el valor que tenga asignado mi instancia en el atributo añoDeNacimiento.

Para utilizar dicha función, debo tener una instancia de un estudiante creado y pedirle que ejecute la función. La llamada a este método se haría de la siguiente manera:

```
Programa.java x
1 package ejs3clases;
2
3 import ejs3clases.Estudiante;
4
5 public class Programa {
6
7     public static void main(String[] args) {
8
9         //Creamos un objeto del tipo Estudiante.
10        Estudiante e1 = new Estudiante("COD1", "Jorge", 4547166, 1987);
11
12        // Le pedimos que calcule la edad para el objeto creado de tipo Estudiante.
13        int edad = e1.calcularEdad(2023);
14
15        // Imprimimos la variable edad, donde se almaceno el cálculo.
16        System.out.println(edad);
17    }
18
19 }
20
```

➤ ¿Qué se imprimirá en la pantalla? Se imprimirá la edad del estudiante e1.

Si vemos, hemos indicado en el constructor que el valor que tiene el atributo añoNacimiento para e1 es 1987. Si vemos el algoritmo de calcular edad, vemos que siempre hacemos añoActual – (atributo añoNacimiento). A “calcularEdad” le estamos pasando por parámetro el entero 2023. Por lo tanto para la instancia e1 la cuenta es 2023 – 1987, lo que nos da 36.

➤ Como podemos ver, toda clase tiene siempre atributos y métodos, que trabajan con estos atributos para una finalidad dada.

## Overloading - Sobrecarga

Java soporta el overloading (sobrecarga) de métodos, esto es: métodos de igual nombre, pero con distintos parámetros.

```
//Ejemplo Overloading - Sobrecarga de métodos.

public int calcularEdad(int anioActual){
    return anioActual - this.anioDeNacimiento;
}

public int calcularEdad(){
    //Se utiliza Calendar para trabajar con fechas.
    Calendar cal = Calendar.getInstance();

    //Obtengo el año actual a través de Calendar.
    int anioActual = cal.get(Calendar.YEAR);

    return anioActual - this.anioDeNacimiento;
}

public int calcularEdad (Date date){
    Calendar cal = Calendar.getInstance();

    // Setea en calendar que la fecha es la pasada por parámetro.
    cal.setTime(date);

    //En anio queda almacenado el año de la fecha pasada por parámetro (date).
    int anio = cal.get(Calendar.YEAR);

    return anio - this.anioDeNacimiento;
}
```

Los métodos a pesar de tener igual nombre, se diferencian en la cantidad y el tipo de los parámetros pasados.

No se puede definir dos métodos de igual nombre, que tengan igual cantidad de parámetros que sean del mismo tipo de dato, ya que Java no los podría distinguir:

```
public int calcularEdad(int anioActual){
    return anioActual - this.anioDeNacimiento;
}

public int calcularEdad(int anioPorParametro){
    return anioPorParametro - this.anioDeNacimiento;
}
```

Como se puede ver, esto da error de compilación, ya que, a pesar de tener distinto nombre, los parámetros de entrada coinciden en cantidad y tipo, entonces el compilador Java no los puede diferenciar.

## Modificadores

A los atributos y métodos de una clase se le define su acceso o visibilidad, esto lo hacemos a través de los modificadores de acceso. Con estos podemos restringir el acceso a métodos y atributos de una clase, de forma de proveer mayor seguridad a un programa.

Para indicar acceso a los miembros (atributos y métodos) de una clase se puede utilizar uno de los siguientes:

- public
- protected
- no utilizar modificador - no se indica visibilidad explícitamente
- private

**Access Levels**

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

La primera columna (modifier) indica el nombre del modificador.

La segunda (class) indica para un modificador dado, si el miembro de la clase es posible accederlo dentro de la clase donde fue definido. Esto siempre es posible no importa cuál sea el modificador utilizado.

La tercera (package) indica para un modificador dado, si el miembro de la clase es posible accederlo dentro de otras clases que se encuentren en el mismo paquete.

La cuarta (sub clase) indica para un modificador dado, si el miembro de la clase es posible accederlo dentro de las clases hijas. La quinta (world) indica para un modificador dado, si el miembro de la clase es posible accederlo dentro de cualquier otra clase.

Hasta ahora hemos visto que a todo atributo o método le indicamos que son público, es decir, pueden ser accedidos desde cualquier bloque de código, inclusive por fuera de la clase donde fueron definidos (a través de `nombreClase.nombreMiembroClase` ).

En la clase Estudiante, por ejemplo, todos sus miembros eran public, es decir son visibles desde otras clases. Esto significa que, se puede hacer `e1.nombre` o `e1.codigo` en cualquier otra clase, como es Programa.



Podemos en cambio hacer algo como lo siguiente:

```
public class Estudiante {  
  
    private String codigo;  
    private String nombre;  
    private int cedulaIdentidad;  
    private int anioDeNacimiento;  
  
    public Estudiante() {  
        this.codigo = "";  
        this.nombre = "";  
        this.anioDeNacimiento = 0;  
        this.cedulaIdentidad = 0;  
    }  
  
    public Estudiante(String codigo, String nombre, int cedulaIdentidad, int anioDeNacimiento) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
        this.cedulaIdentidad = cedulaIdentidad;  
        this.anioDeNacimiento = anioDeNacimiento;  
    }  
}
```

```
public class Programa {  
  
    public static void main(String[] args) {  
  
        Estudiante ee1;  
  
        ee1 = new Estudiante();  
  
        String nombreDeEE1 = ee1.nombre;  
  
        Estudiante ee2 = new Estudiante("ES1", "Nico", 4547166, 1996);  
  
        String nombreDeEE2 = ee2.nombre;  
  
        int cedulaDeEE2 = ee2.cedulaIdentidad;  
  
    }  
}
```

➤ Para este caso, se ve que el compilador, en la clase “Programa” señala error de compilación cuando se accede a los atributos. Esto se debe a que no es posible accederlos desde otra clase, ya que se especificó que son privados (modificador private).

¿Cómo se puede resolver esto, manteniendo el nivel de acceso de los atributos? Se puede definir métodos que sí sean públicas y que dentro de estas se modifiquen los valores de los atributos. Cómo la implementación de estas funciones estaría dentro de la clase Estudiante, tendríamos acceso a los atributos privados.

```
public class Estudiante {  
  
    private String codigo;  
    private String nombre;  
    private int cedulaIdentidad;  
    private int anioDeNacimiento;  
  
    public Estudiante() {  
        this.codigo = "";  
        this.nombre = "";  
        this.anioDeNacimiento = 0;  
        this.cedulaIdentidad = 0;  
    }  
  
    public Estudiante(String codigo, String nombre, int cedulaIdentidad, int anioDeNacimiento) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
        this.cedulaIdentidad = cedulaIdentidad;  
        this.anioDeNacimiento = anioDeNacimiento;  
    }  
  
    public int calcularEdad(int anioActual){  
        return anioActual - this.anioDeNacimiento;  
    }  
  
    public String obtenerCodigo(){  
        return this.codigo;  
    }  
}
```

Como se puede ver, existe un método de la clase Estudiante, que se llama “obtenerCodigo”, cuya función es devolver el valor del código. De esta forma, como este método es público, sí podemos llamarlo desde otra clase. Por lo tanto, cada vez que se quiera acceder al código de

un objeto estudiante, en vez de referenciar directo al atributo (que no es posible ya que es privado), utilizamos la función obtenerCodigo:

```
public static void main(String[] args) {  
  
    // Creamos un objeto del tipo Estudiante.  
    Estudiante e1 = new Estudiante("COD1", "Jorge", 4547166, 1987);  
  
    // Almacenamos el código en una variable de tipo String (mismo tipo que el método obtenerCodigo()).  
    String codigo = e1.obtenerCodigo();  
  
    // Imprimimos por consola.  
    System.out.println("El estudiante tiene el código: " + codigo);  
  
}
```

Si bien, no podemos acceder directamente al atributo haciendo e1.codigo (porque este es privado) sí podemos hacerlo a través del método que hemos definido, llamado “obtenerCodigo”. Podríamos hacer lo mismo, para poder asignarle un valor a un atributo:

```
public class Estudiante {  
  
    private String codigo;  
    private String nombre;  
    private int cedulaIdentidad;  
    private int anioDeNacimiento;  
  
    public Estudiante() {  
        this.codigo = "";  
        this.nombre = "";  
        this.anioDeNacimiento = 0;  
        this.cedulaIdentidad = 0;  
    }  
  
    public Estudiante(String codigo, String nombre, int cedulaIdentidad, int anioDeNacimiento) {  
        this.codigo = codigo;  
        this.nombre = nombre;  
        this.cedulaIdentidad = cedulaIdentidad;  
        this.anioDeNacimiento = anioDeNacimiento;  
    }  
  
    public String obtenerCodigo(){  
        return this.codigo;  
    }  
  
    public void asignarCodigo(String codigo){  
        this.codigo = codigo;  
    }  
}
```

Ahora agregamos un nuevo método llamado `asignarCodigo` que recibe como parámetro un `String` y no devuelve ningún resultado. Dentro del código del método, simplemente se le asigna al atributo código, el valor pasado por parámetro. De esta manera podríamos modificar el valor de código por fuera de la clase, utilizando dicho método:

```
public static void main(String[] args) {  
    // Creamos un objeto del tipo Estudiante.  
    Estudiante e1 = new Estudiante("COD1", "Jorge", 4547166, 1987);  
  
    // Asignamos un nuevo código.  
    e1.asignarCodigo("NUEVOCODIGO");  
  
    // Almacenamos el código en una variable de tipo String (mismo tipo que el método obtenerCodigo()).  
    String codigo = e1.obtenerCodigo();  
  
    // Imprimimos por consola.  
    System.out.println("El estudiante tiene el código: " + codigo);  
}
```

Esto nos es útil, por si queremos en cierto momento cambiar el estado de nuestro objeto. En este caso, el valor inicial del atributo código es COD1 (por el constructor) pero luego modificamos el valor del atributo código a NUEVOCODIGO (por el método `asignarCodigo`).

A los métodos que devuelven atributos, se los denomina “getters”, de la palabra “get” de inglés, que es obtener. Y a los métodos que asignan valores a los atributos, se les denomina “setters”, por la palabra “set” en inglés

Es una buena práctica y se recomienda mantener los atributos de una clase como privados y accederlos mediante sus métodos getter y setter. Por convención a estos métodos se los llama: *getNombreDeAtributo*, *setNombreDeAtributo*.

¿Cómo quedaría la clase Estudiante luego de indicar que todos sus atributos son private e implementar los getters y setters?

```
package ejs3clases;

import java.util.Calendar;
import java.util.Date;

public class Estudiante {

    private String codigo;
    private String nombre;
    private int cedulaIdentidad;
    private int anioDeNacimiento;

    public Estudiante() {
        this.codigo = "";
        this.nombre = "";
        this.anioDeNacimiento = 0;
        this.cedulaIdentidad = 0;
    }

    public Estudiante(String codigo, String nombre, int cedulaIdentidad, int anioDeNacimiento) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.cedulaIdentidad = cedulaIdentidad;
        this.anioDeNacimiento = anioDeNacimiento;
    }

    //Getters & Setters
    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getCedulaIdentidad() {
        return cedulaIdentidad;
    }

    public void setCedulaIdentidad(int cedulaIdentidad) {
        this.cedulaIdentidad = cedulaIdentidad;
    }

    public int getAnioDeNacimiento() {
        return anioDeNacimiento;
    }

    public void setAnioDeNacimiento(int anioDeNacimiento) {
        this.anioDeNacimiento = anioDeNacimiento;
    }
}
```

## Atributos y Métodos de Clase

Para definir atributos o métodos de clase y no de instancia (como los anteriores), se utiliza *static*.

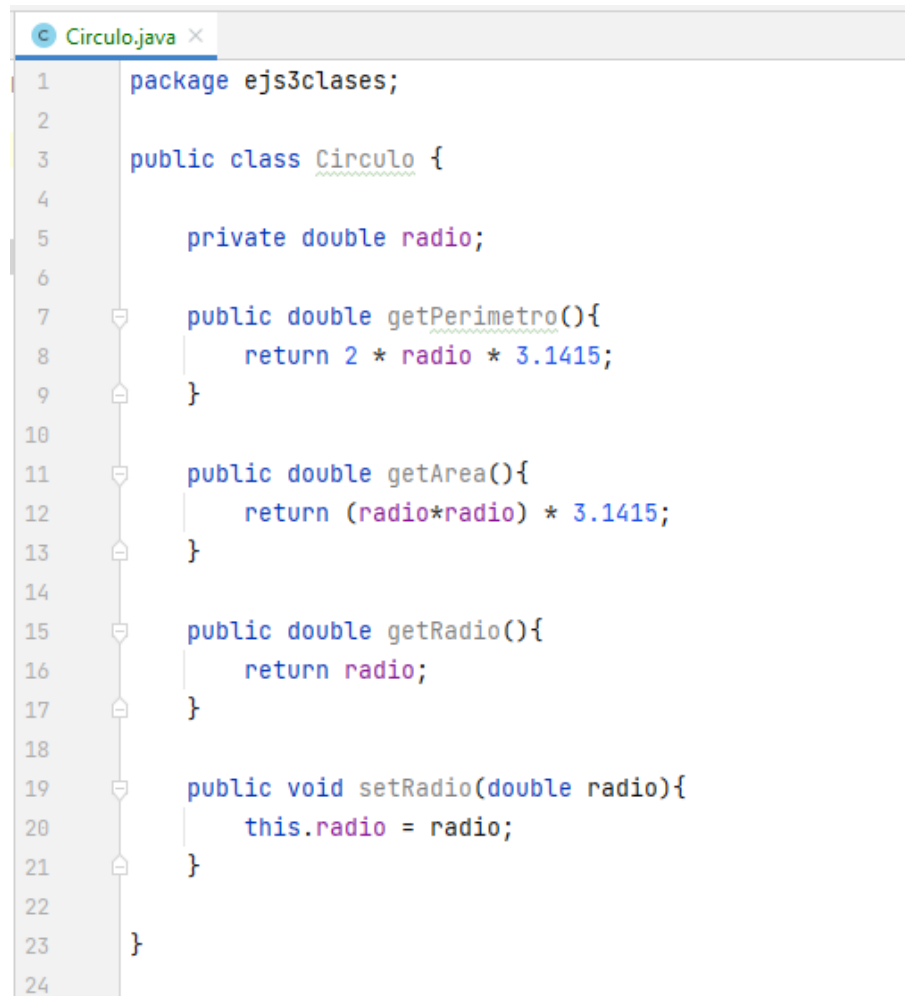
Cuando se instancia un objeto de una clase, cada objeto particular les da valor a sus atributos. Por ejemplo, al instanciar dos objetos de la clase Estudiante, cada objeto mantiene sus propios valores (almacenados en espacio de memoria distintos) y estos valores no son compartidos entre objetos (son independientes). Estos son atributos de instancia.

Hay casos en que es necesario compartir atributos y que su valor sea común a todas las instancias creadas de una clase. Es decir, en vez de asociar valores a objetos asociarlos a la clase.

Para esto se utilizan los atributos de clase a través de la palabra *static*.

### Ejemplo 3

Creo una clase Círculo, donde voy a encapsular el comportamiento de los círculos: como atributos va a tener el radio, y como métodos el cálculo del perímetro y área.



```
1 package ejs3clases;
2
3 public class Circulo {
4
5     private double radio;
6
7     public double getPerimetro(){
8         return 2 * radio * 3.1415;
9     }
10
11     public double getArea(){
12         return (radio*radio) * 3.1415;
13     }
14
15     public double getRadio(){
16         return radio;
17     }
18
19     public void setRadio(double radio){
20         this.radio = radio;
21     }
22
23 }
24
```

El cálculo del perímetro y área dependen de cuánto vale el radio del objeto y del valor de pi, que es un número fijo (3,1415). Por lo tanto, el valor del radio depende del objeto (es una variable de instancia), pero el valor de pi es común a todos los objetos (es una variable de clase).

Para esto se podría definir el atributo de clase pi:

```
Circulo.java x
1  package ejs3clases;
2
3  public class Circulo {
4
5      private static double pi = 3.1415;
6
7      private double radio;
8
9      public double getPerimetro(){
10         return 2 * radio * pi;
11     }
12
13     public double getArea(){
14         return (radio*radio) * pi;
15     }
16
17     public double getRadio(){
18         return radio;
19     }
20
21     public void setRadio(double radio){
22         this.radio = radio;
23     }
24
25 }
26
```

De esta forma decimos que el atributo pi tiene un valor que es compartido por todos los objetos de la clase.

En este caso, ¿el valor de pi puede cambiar? No. Pi es una constante matemática.



Por lo que, al dejarlo como variable se está dejando abierta la posibilidad a que pi pueda ser modificado (y se sabe que, al ser una constante matemática, esto no podría suceder).

¿Cómo en Java especificamos que un valor es constante?

Utilizando *static final*.

La combinación de static con final define una constante. El modificador *final* indica que el valor del campo no puede ser modificado.

A las constantes se las declara y asigna un valor inicial, y este no puede ser modificado luego, es más, si se encuentra otra asignación, el programa no compila.

Por convención las constantes se escriben todo en mayúscula, con las diferentes palabras separadas por guión bajo (\_).

```
private static double PI = 3.1415;
```

También es posible implementar **métodos de clase**, al igual que las variables de clase, utilizando static.

Estos métodos, al ser de clase, son métodos que no necesitan de un objeto creado para ser invocados, esos se llaman:

*nombreClase.metodoStatic(parámetros)*

Es posible también llamar estos métodos en una instancia de clase:

*nombreObjeto.metodoStatic(parámetros)* Pero este tipo de acceso no es recomendado.

Estos métodos se utilizan comúnmente para acceder a las variables static.

Veamos las posibles combinaciones de métodos de clase e instancia a que variables pueden acceder:

- Dentro de un método de instancia se puede acceder a:
  - Variables y otros métodos de instancia.
  - Variables y métodos de clase.
  
- Dentro de un método de clase se puede acceder a:
  - Variables y métodos de clase.
  - No puede acceder a variables y métodos de instancia a menos que se haya llamado desde un objeto (*nombreObjeto.metodoStatic(parámetros)*), cuyo uso no es recomendado.
  - No se puede utilizar la palabra *this*, ya que *this* hace referencia a la instancia.