1. MÉTODOS EN JAVA

En la POO el conjunto de métodos de un objeto es el que determina el comportamiento del mismo. Los métodos son lo que en otros paradigmas se conoce como funciones, y que pueden ser llamadas dentro de la clase o desde otras clases (según la visibilidad). Un método consta de dos partes: una declaración (cabecera o **firma**) y un cuerpo (implementación o código). La declaración en Java se puede expresar básicamente como:

```
[modificadores] tipoRetorno nombreMetodo( [lista_de_parámetros] ) {
    cuerpoDelMetodo
}
```

Los modificadores son opcionales. Se tratarán más adelante.
La lista de parámetros se expresa declarando el tipo y nombre de los mismos (como en las declaraciones de variables). Si hay más de uno se separan por comas. La lista es opcional. Si no hubiera parámetros, se colocan dos paréntesis vacíos luego del nombre. Los parámetros se utilizan para pasar información al cuerpo del método.

```
int sumaEnteros ( int a, int b ) {
        int c = a + b;
        return c;
}
```

- Nombre del método: sumaEnteros.
- Recibe dos parámetros enteros. Sus nombres son a y b.
- Devuelve un entero (tipoRetorno indicado por el **int** delante del nombre del método)

La clausula **return** se usa para finalizar el método devolviendo el valor de la variable **c.** El tipo de la variable devuelta debe coincidir con el tipo de retorno declarado para el método (tipoRetorno). En el ejemplo, ambos son **int**. Cuando se llama a un método, este se ejecuta hasta encontrarse con una sentencia return. Esta pasa el valor especificado al código que llamó a dicho método y se devuelve el control al código invocador. Su misión tiene que ver con el control de flujo: se deja de ejecutar código secuencialmente y se pasa al código que invocó al método.

1.1. Valor de retorno de un método en java

En Java es obligatorio indicar el tipo de dato que devolverá el método (tipoRetorno). Si no devuelve ningún valor, se debe indicar el tipo **void** como retorno.

```
void imprimirAlgo() {
    System.out.println("Es una prueba");
}
```

Cuando no se devuelve ningún valor, la cláusula return no es necesaria. Obsérvese que en el ejemplo el método imprimirAlgo tampoco recibe ningún parámetro. No obstante los paréntesis, son obligatorios.
Los métodos pueden devolver una variable o un objeto (la referencia al mismo). En este último caso, lo que se devuelve es la dirección de la posición en memoria donde se encuentra almacenado el objeto.

1.2. Modificadores de un método en java

Los modificadores son palabras clave que afectan al comportamiento del método.
La sintaxis de la declaración completa de un método es la que se muestra a continuación con los items opcionales en negrita:

```
especificadorAcceso static abstract final native synchronized tipoRetorno nombreMetodo(
        lista_de_argumentos ) throws listaEscepciones {
                    CuerpoDelMetodo
}
```

especificadorAcceso: determina si otros objetos pueden acceder al método y cómo pueden hacerlo.

static: se usa para definir datos miembros o métodos como pertenecientes a una clase, en lugar de pertenecer a una instancia.

abstract: indica que el método no está definido en la clase, sino que se encuentra declarado ahí para ser definido en una subclase (sobreescrito).

final: evita que un método pueda ser sobreescrito.

native: son métodos escritos en otro lenguaje. Java soporta C y C++.

synchronized: se usa para procesos concurrentes (multithreading).

throws listaExcepciones: indica las excepciones que puede generar y manipular el método.

1.3. El método main en Java

Un programa Java inicia su ejecución al proporcionar al intérprete Java un nombre de clase. La clase es cargada en memoria y se inicia su ejecución siempre por un método estático que debe estar codificado en esa clase. El nombre de este método es **main** y debe declararse de la siguiente forma:

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

          CuerpoDelMetodo
}
```

- Es un método estático. Se aplica por tanto a la clase y no a una instancia en particular, lo que es conveniente puesto que en el momento de iniciar la ejecución todavía no se ha creado ninguna instancia de ninguna clase.
- Recibe un argumento de tipo String [ ]. String es una clase que representa una cadena de caracteres
- Los corchetes [ ] indican que se trata de un array.

No es obligatorio que todas las clases declaren un método main. Sólo aquellas clases que serán invocadas directamente desde la línea de comandos. En la práctica la mayor parte de las clases no lo tienen.

1.4. Sobrecarga de métodos

Una misma clase puede tener varios métodos con el mismo nombre, siempre que se diferencien en el tipo o número de los parámetros, de forma que el compilador pueda diferenciar claramente cuándo se invoca a uno o a otro, en función de los argumentos que se utilicen en la llamada al método. Cuando esto sucede se dice que el método está sobrecargado. Por ejemplo, una misma clase podría tener los métodos:

```
int metodoSobrecargado() { . . .}
int metodoSobrecargado(int x) { . . .}
```

Sin embargo no se puede sobrecargar cambiando **sólo el tipo del valor devuelto**. Por ejemplo:

```
int metodoSobrecargado() { . . .}
void metodoSobrecargado() { . . .}  // error en compilación
```

En Java, los métodos sobrecargados siempre deben devolver el mismo tipo. Con esta última declaración, al invocar al método con la expresión `objeto.metodoSobrecargado()` el compilador no sabría cual de los métodos invocar.
Se puede sobrecargar cualquier método miembro de una clase, incluido el constructor (ver más adelante).
El siguiente fragmento de código muestra una clase Java con cuatro métodos sobrecargados, el último no es legal porque tiene el mismo nombre y lista de parámetros que el declarado previamente:

```
class MiClase {
        . . .
        void miMetodo( int x, int y ) { . . . }
        void miMetodo( int x ) { . . . }
        void miMetodo( int x, float y ) { . . . }
        void miMetodo( int a, float b ) { . . . } // no válido
}
```

2. CLASES EN JAVA

2.1. Atributos y métodos miembro. Modo de acceso

En la POO, una clase es una agrupación de datos y de métodos que actúan sobre esos datos, a la que se le da un nombre.
Una clase contiene:

- Atributos (se denominan Atributos Miembro). Estos pueden ser de tipos primitivos o referencias a objetos.
- Métodos (se denominan Métodos Miembro).

Atributos Miembro y Métodos Miembro configuran el estado y el comportamiento que puede realizar un objeto de la clase.

En java la sintaxis general es:

```
modificadores class nombre_clase {
       declaraciones_de_atributos_miembro ;
       declaraciones_de_metodos_miembro ;
}
```

```
public class Punto {
  // atributos miembro
  private int x;
  private int y;

  // métodos miembro
  public int getX(){
    return x;
  }
  public void setX(int p_x){
    x = p_x;
  }
}
```

La acción de crear objetos de una clase se llama instanciación. Un objeto es una instancia de una clase. Ocupa un lugar en la memoria del ordenador. Los objetos se manipulan con referencias. **Una referencia es una variable que apunta a un objeto** (tiene la dirección de memoria donde se aloja el objeto). Las referencias se declaran igual que las variables de tipos primitivos (*tipo nombre*). Los objetos se crean (se instancian) con el operador de instanciación **new**, que le asigna memoria.

```
public class CreaPuntos{
    public static void main (String [] args){
       Punto p;         // declaración de una referencia (p) a un objeto de tipo Punto. "p" no apunta a ningún sitio aún
      p=new Punto();  // instanciación. La variable "p" recibe la dirección de memoria del objeto de tipo Punto creado
    }
}
```

Nota: Se pueden hacer ambas operaciones en la misma expresión (declaración y definición).

```
Punto p = new Punto();
```

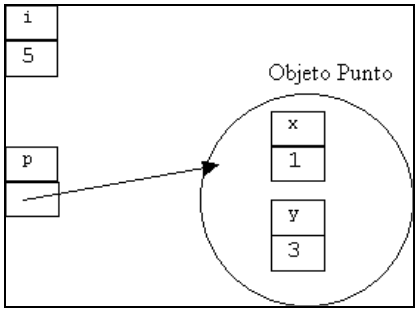
Los atributos de un objeto también se denominan *variables de instancia* (v.i.). Se puede acceder a ellos a través de su referencia (p):

```
p.x = 1;
p.y = 3;
```

Si bien java permite esta forma directa de acceder a los datos, esto atenta contra el principio de encapsulamiento, por lo que más adelante se verá el modo seguro de acceder a los datos (geter's y seter's).

En memoria, los datos primitivos, referencias y objetos se almacenan de la siguiente forma:

Dato primitivo: **int** i = 5;
Referencia: **Punto** p;
Objeto: el punto creado con el operador new



Área de memoria que contiene:

- el dato primitivo "i" (de tipo entero) que almacena el valor 5
- la referencia "p" (de tipo Punto) que almacena la dirección de memoria en la que se encuentra el objeto Punto
- el objeto Punto, cuyas variables miembro tienen los valores 1 y 3 respectivamente

Es importante destacar que en el ejemplo, "p" no es el objeto. Es una **referencia** al objeto (apunta hacia el objeto).

Los métodos miembro se declaran dentro de la declaración de la clase:

```
public class Circulo {
  Punto centro;      // atributo miembro. Referencia a un objeto de tipo Punto
  int radio;         // atributo miembro. Valor primitivo

  double superficie() {            // método miembro
    return 3.14 * radio * radio;
  }                                // fin del método superficie
}                                  // fin de la clase Circulo
```

El acceso a métodos miembro, al igual que para atributos miembro, es a través de la referencia al objeto:

```
public class CreaCirculo{
    public static void main (String [] args){
      Circulo c = new Circulo();
      c.centro.x = 2;
      c.centro.y = 3;
      c.radio = 5;
      double s = c.superficie();   // acceso al método miembro "superficie"
    }
}
```

Observaciones:

- Los atributos miembro pueden ser tanto primitivos como referencias. La clase Circulo contiene un atributo miembro de tipo Punto (que es el centro del círculo).
- El acceso a los atributos miembros del Punto *centro* se hace encadenando el operador **.** en la expresión `c.centro.x` que se podría leer como "el miembro ***x*** del objeto ***centro*** (de tipo Punto) del objeto ***c*** (de tipo Circulo)".
- Aunque el método *superficie* no recibe argumentos, los paréntesis son obligatorios (distinguen los datos de los métodos).

2.2. Modificadores de clase en Java

Los modificadores son palabras clave que afectan al comportamiento de la clase. Existen modificadores con distintas funciones. Una de ellas es el tipo de acceso a la clase. Las clases pueden declararse:

- **public:** Cualquiera puede usar la clase. Se pueden crear instancias de esa clase, siempre y cuando alguno de sus constructores sea accesible.
- **sin modificador:** La clase puede ser usada e instanciada sólo por clases que pertenezcan al package donde fueron definidas

Observaciones:

- Las clases no pueden declararse ni **protected**, ni **private**.
- Se puede definir más de una clase en un archivo fuente, pero sólo una de ellas podrá ser declarada **public** (es decir podrá ser utilizada fuera del package donde se define). Todas las demás declaradas en el fuente serán internas al package.
- Si hay una clase **public** entonces, obligatoriamente, el nombre del archivo fuente tiene que coincidir con el de la clase declarada como public

3. CONSTRUCTORES EN JAVA

Son métodos especiales que, al ser invocados, permiten inicializar objetos. La invocación es implícita y se realiza automáticamente cuando se utiliza el operador **new**. Los constructores tienen algunas características especiales:

- El nombre del constructor tiene que ser igual al de la clase.
- Puede recibir cualquier número de argumentos de cualquier tipo, como cualquier otro método.
- No devuelve ningún valor (en su declaración no se declara ni siquiera **void**).
- Puede tener modificadores de acceso (public, private, etc) que actúan de igual modo que en los otros métodos.

```
public class Punto {
    int x;
    int y;

    Punto ( int a , int b ) {  // método constructor
       x = a ;
       y = b ;
    }
}
```

Con este constructor se puede crear un objeto de la clase Punto de la siguiente forma:

```
Punto p = new Punto (1, 3);
```

3.1. Sobrecarga de constructores en Java

Una clase puede definir más de un constructor, es decir, un objeto puede inicializarse de varias formas (distinta cantidad o tipo de parámetros). En el momento de crear un objeto, el operador **new** llama al constructor que coincida en número y tipo de argumentos. Si no hay ninguno coincidente se produce un error en tiempo de compilación.

```
public class Punto {
    int x;
    int y;

    Punto(int a, int b) {       // constructor con parámetros
        x = a;
        y = b;
    }

    Punto () {                 // constructor sin parámetros
        x = 0;
        y = 0;
    }
}
```

Cuando se declaran varios constructores para una misma clase estos deben distinguirse en la lista de parámetros, ya sea en el número, o en el tipo. Este mecanismo de **definir métodos con el mismo nombre se denomina "sobrecarga"** y es aplicable a cualquier método miembro de una clase.

4. MODIFICADORES.

Los modificadores son elementos del lenguaje que se colocan delante de la declaración de atributos (datos miembro), métodos o clases, y que alteran o condicionan el significado del elemento. Algunos de ellos, ya mencionados, son: static, abstract o synchronized. Los más comunes son los modificadores de acceso, que son aquellos que permiten limitar o generalizar el acceso a los componentes de una clase o a la clase en si misma.

4.1. Modificadores de acceso

Los modificadores de acceso permiten al diseñador de una clase determinar de qué modo otros objetos pueden acceder a los datos y métodos miembros de dicha clase. Preceden a la declaración de un elemento de la clase (atributo/método), de la siguiente forma:

```
[modificadores] tipo_variable nombreVariable;
```

```
[modificadores]  tipoRetorno nombreMetodo ( lista_Parámetros );
```

Existen cuatro modificadores de acceso:

- **public** – Cualquier clase desde cualquier lugar puede acceder al elemento. Si es un atributo miembro, cualquiera puede ver el elemento, es decir, usarlo y asignarlo. Si es un método cualquiera puede invocarlo.
- **private** - Sólo se puede acceder al atributo desde métodos de la clase. Si es un método, sólo puede invocarse el método desde otro método de la misma clase. No son accesibles desde las subclases.
- **protected** – Está relacionado con la herencia. Sólo la propia clase y las subclases, pueden acceder a las variables y métodos de instancia protegidos.
- **sin modificador** - Se puede acceder al elemento desde cualquier clase del package donde se define la clase. Habitualmente se conoce como acceso friendly (amistoso), lo que significa que son accesibles por todos los objetos dentro del mismo paquete, pero no por los externos al paquete.

Pueden utilizarse estos modificadores para cualquier tipo de miembros de la clase, incluidos los constructores (con lo que se puede limitar quién puede crear instancias de la clase).

4.2. Métodos de acceso a los atributos

Si bien Java (que es un lenguaje híbrido) permite declarar los atributos sin ocultarlos (sin modificador ***private***), **para respetar los principios del paradigma de objetos, y aprovechar sus beneficios, se deben ocultar (encapsular) todos sus atributos, y proveer los métodos de acceso apropiados**. El encapsulamiento debe aplicarse también dentro del propio constructor, utilizando los set para las asignaciones (doble encapsulamiento). Este mecanismo se implementa mediante los *accessors*, que permiten acceder a los atributos de modo consistente:

- observadores (getters), que muestran (retornan) los atributos, sin modificarlos
- mutadores (setters), que asignan valores a los atributos

Estos no son nombres obligatorios. Solo es una convención de la comunidad java, y son utilizados por su significado: get=obtener, set=establecer/asignar.

Habitualmente se utiliza get/set agregado al nombre del atributo correspondiente: setX, getX, getNombre, setDomicilio.

```
public class Punto {
    private int x;
    private int y;

    public Punto ( int a , int b ) {
       setX(a);
       setY(b);
    }

    int getX() {              // sin modificador. Acceso de paquete
        return x;
    }

    public int getY() {       // público. Acceso irrestricto
        return y;
    }

    public void setX(int a) {    // público. Acceso irrestricto
        x = a;
    }

    private void setY(int b) {    // privado. Acceso restringido al propio objeto
        y = b;
    }
}
```

En el ejemplo los datos miembros de la clase Punto se declaran como private, y se incluyen métodos que devuelven las coordenadas del punto y asignan un valor a las mismas (get/set). De esta forma el diseñador de la clase controla el contenido de los datos que la representan, e independiza la implementación de la interface.

Si desde una clase externa a Punto se intenta:

```
public class CreaPunto{
    public static void main (String [] args){
      Punto p = new Punto(0,0);
      p.x = 5;
    }
}
```

Se obtendrá un error del compilador. Debe realizarse a través de su interfaz pública. Tampoco es posible asignar valor al atributo y, dado que el método setY() es privado, por lo cual no forma parte de la interfaz pública.

```
public class CreaPunto{
    public static void main (String [] args){
      Punto p = new Punto(0,0);
      p.x = 5;    // error
      p.setX(5);
      p.setY(8);  // error. No pertenece a la interfaz píblica

    }
}
```

Los modificadores de acceso son muy importantes, dado que permiten al diseñador de clases delimitar la frontera entre lo que es accesible para los usuarios de la clase (public), lo que es estrictamente privado y no es de la incumbencia de nadie más que el diseñador de la clase (private), e incluso lo que podría llegar a importar a otros diseñadores de clases que quisieran redefinir, completar o especializar el comportamiento de la clase (protected).
Estos modificadores aseguran uno de los principios básicos de la POO, que es la encapsulación: Las clases tienen un comportamiento definido para quienes las usan, conformado por los elementos que tienen un acceso **público**, y una implementación oculta formada por los elementos **privados**, de la que no tienen que preocuparse los usuarios de la clase.
Los otros dos modificadores, **protected** y el acceso por defecto (**package**) complementan a los primeros. El primero puede utilizarse cuando se declara relaciones de herencia entre las clases, y el segundo establece relaciones de "confianza" entre clases afines dentro del mismo package. Así, la pertenencia de las clases a un mismo package es algo más que una clasificación de clases por cuestiones de orden.

5. MENSAJES

Los objetos colaboran entre sí enviándose mensajes. Un objeto (solicitante/emisor) le hace una petición a otro objeto (receptor/colaborador), a través de una ***referencia*** al objeto, mediante el envío de un mensaje, con el siguiente formato:

```
objeto_receptor.nombre_de_mensaje(argumentos);
```

Para responder, el objeto receptor debe saber cómo hacerlo, y para ello busca entre sus métodos aquel que coincida con la **firma** del mensaje recibido:

```
public class Circulo {
    private Punto centro;
    private int radio;

    Circulo (Punto p_centro, int p_radio){
       setCentro(p_centro);
       setRadio(p_radio);
    }

    public int getRadio() {
        return radio;
    }

    public Punto getCentro() {
        return centro;
    }

    private void setRadio(int p_radio) {
        radio = p_radio;
    }

    private void setCentro(Punto p_centro) {
        centro = p_centro;

    }

    public double superficie(){
       double sup = 3.14 * radio * radio;
       return sup;
    }
}
```

Para solicitar (hacer una petición) a un objeto de tipo Circulo, se debería enviar el correspondiente mensaje:

```
public class CreaCirculo{
    public static void main (String [] args){
         Punto unPunto = new Punto(2,5);
         Circulo unCirculo = new Circulo(unPunto, 4);
         double sup = unCirculo.superficie();   // envío de mensaje "superficie"
         System.out.println("La superficie es: " + sup);
    }
}
```

6. Referencia del objeto a sí mismo

Dado que el envío de mensaje se realiza a través de una ***referencia*** al objeto, para enviarse un mensaje así mismo un objeto necesita tener una forma de nombrarse a sí mismo. Este autoreferenciamiento se hace en Java por medio de la seudo-variable ***this***. Se dice que es una seudo-variable ya que funciona similar a una variable de instancia, excepto por dos motivos: no está definida en ninguna clase y no puede ser asignada (no se le puede asignar un valor). Todos los objetos tienen la referencia implícita ***this***, que apunta a sí mismo.

```
public class Punto {
    private int x;
    private int y;

    Punto(int a, int b){
       this.setX(a);
       this.setY(b);
    }

    public int getX(){
        return this.x;
    }

    public int getY() {
        return this.y;
    }

    private void setX(int a) {
        this.x = a;
    }

    private void setY(int b) {
        this.y = b;
    }
}
```

```
public class Circulo {
    private Punto centro;
    private int radio;

    Circulo(Punto p_centro, int p_radio){
       this.setCentro(p_centro);
       this.setRadio(p_radio);
    }

    public int getRadio() {
        return this.radio;
    }

    public Punto getCentro() {
        return this.centro;
    }

    private void setRadio(int p_radio) {
        this.radio = p_radio;
    }

    private void setCentro(Punto p_centro) {
        this.centro = p_centro;

    }

    public double superficie(){
       double sup = 3.14 * this.getRadio()  * this.getRadio();
       return sup;
   }
}
```

7. Atributos y Métodos de clase

Las clases pueden tener atributos exclusivos, que no se repiten en cada instancia de la misma, y sin embargo están disponibles para todas las instancias. Se llaman **atributos de clase**. En Java, a estos atributos se antepone el modificador ***static***. Para acceder a los atributos de clase se deben definir métodos que también se identifican con el modificador ***static.*** Los métodos que acceden a atributos de clase se denominan **métodos de clase** y sólo pueden ser invocados por mensajes enviados a la clase, no así a sus instancias. Los métodos de clase (static) sólo pueden acceder a atributos de clase y **NO** a los de objeto. Esto es debido a que pueden ser invocados ANTES de que el objeto sea creado.

```
public class Punto {
    private int x;
    private int y;
    static private int numPuntos;

    // los métodos de objeto pueden acceder a atributos de clase y de objeto
    public Punto ( int a , int b ) {
       this.setX(a);
       this.setY(b);
            // lleva this (this es una referencia al objeto. El objeto puede acceder a atributos de su clase)
       this.setNumPuntos(this.numPuntos + 1);
    }

    public int getX() {
        return this.x;
    }

    public int getY() {
        return this.y;
    }

    private void setX(int a) {
        this.x = a;
    }

    private void setY(int b) {
        this.y = b;
    }

    public static void setNumPuntos(int puntosCreados) {
          // numPuntos NO lleva this (this es referencia a objeto, y numPuntos es de la clase)
            numPuntos = puntosCreados;
    }
```

```
    public static int cuantosPuntos() {
            // numPuntos NO lleva this (this es referencia a objeto, y numPuntos es de la clase)
        return numPuntos;
    }
}
```

Un ejemplo de los usos correctos e incorrectos de la clase Punto es el siguiente:

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

      // se puede acceder al valor de numPuntos ANTES de instanciar un objeto, enviando el mensaje a la clase

       Punto.setNumPuntos(0); // se inicializa el valor del atributo de clase
       System.out.println("Hay "+Punto.cuantosPuntos()+ " puntos creados");

       Punto p1 = new Punto(10,5);
       Punto p2 = new Punto(15,9);


      // el valor de numPuntos es el mismo si se lo muestra desde la clase o desde el objeto

       System.out.println("Puntos desde clase     "+Punto.cuantosPuntos());
       System.out.println("Puntos desde un objeto "+p1.cuantosPuntos());


       // se puede asignar valor desde la clase o desde el objeto al atributo de clase numPuntos

       Punto.setNumPuntos(4);
       System.out.println("Cantidad asignada desde clase     "+Punto.cuantosPuntos());
       p1.setNumPuntos(3);
       System.out.println("Cantidad asignada desde un objeto "+p1.cuantosPuntos());


      // sólo se puede acceder al valor de un atributo de objeto desde el objeto

       System.out.println("Valor de X: "+p1.getX());


       // ATENCION !!! la siguiente instrucción señala error al compilar

       System.out.println("Valor de X: "+Punto.getX());
   }
}
```

Resultado de la ejecución

```
Hay 0 puntos creados
Puntos desde clase     2
Puntos desde un objeto 2
Cantidad asignada desde clase     4
Cantidad asignada desde un objeto 3
Valor de X: 10
```

8. PASO DE PARAMETROS

En Java, todos los métodos deben estar declarados y definidos dentro de una clase, y se debe indicar el tipo y nombre de los parámetros que acepta (***parámetros formales***). Los parámetros recibidos en un método actúan como variables locales declaradas en el cuerpo del método, y están inicializadas al valor que se pasa como **argumento** en la invocación del método (***parámetros reales***).
En Java, todos los argumentos de **tipos primitivos** deben pasarse por valor, mientras que **los objetos** deben pasarse por referencia. Cuando se pasa un objeto por referencia, se está pasando **la dirección de memoria** en la que se encuentra almacenado el objeto.
Si en el cuerpo del método se modifica una variable que haya sido pasada por valor, no se modificará la variable original que se haya utilizado para invocar al método, mientras que si se modifica una variable pasada por referencia, la variable original del método de llamada se verá afectada por los cambios que se produzcan en el método al que se le ha pasado como parámetro.
Cuando se pasa un array como argumento, en realidad se pasa la referencia al array (recordar que es un objeto de una clase predefinida), por lo tanto, se debe tener en cuenta que los cambios realizados en el cuerpo del método, modificarán sus valores.