3.D. Métodos.

Sitio: Aula Virtual
Curso: Programación_DAM
Libro: 3.D. Métodos.
Imprimido por: LUIS PUJOL
Día: martes, 3 de diciembre de 2019, 13:12

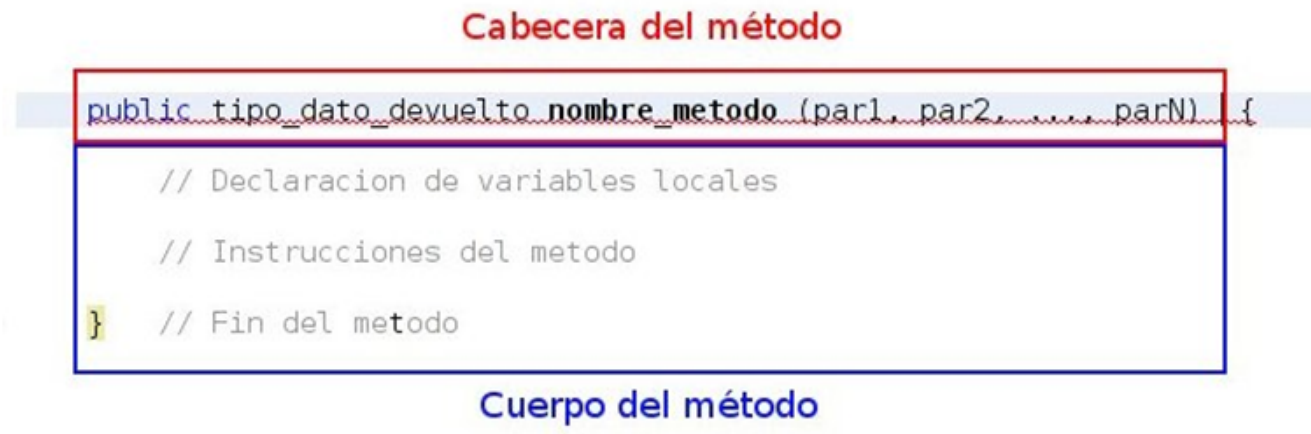
Tabla de contenidos

1 Utilización de métodos.

Los métodos, junto con los atributos, forman parte de la estructura interna de un objeto. Los métodos contienen la declaración de variables locales y las operaciones que se pueden realizar para el objeto, y que son ejecutadas cuando el método es invocado. Se definen en el cuerpo de la clase y posteriormente son instanciados para convertirse en **métodos instancia** de un objeto.

Para utilizar los métodos adecuadamente es conveniente conocer la estructura básica de que disponen.

Al igual que las clases, los métodos están compuestos por una **cabecera** y un **cuerpo**. La cabecera también tiene modificadores, en este caso hemos utilizado `public` para indicar que el método es público, lo cual quiere decir que le pueden enviar mensajes no sólo los métodos del objeto sino los métodos de cualquier otro objeto externo.



Dentro de un método nos encontramos el cuerpo del método que contiene el código de la acción a realizar. Las acciones que un método puede realizar son:

- **Inicializar** los atributos del objeto
- **Consultar** los valores de los atributos
- **Modificar** los valores de los atributos
- **Llamar** a otros métodos, del mismo objeto o de objetos externos

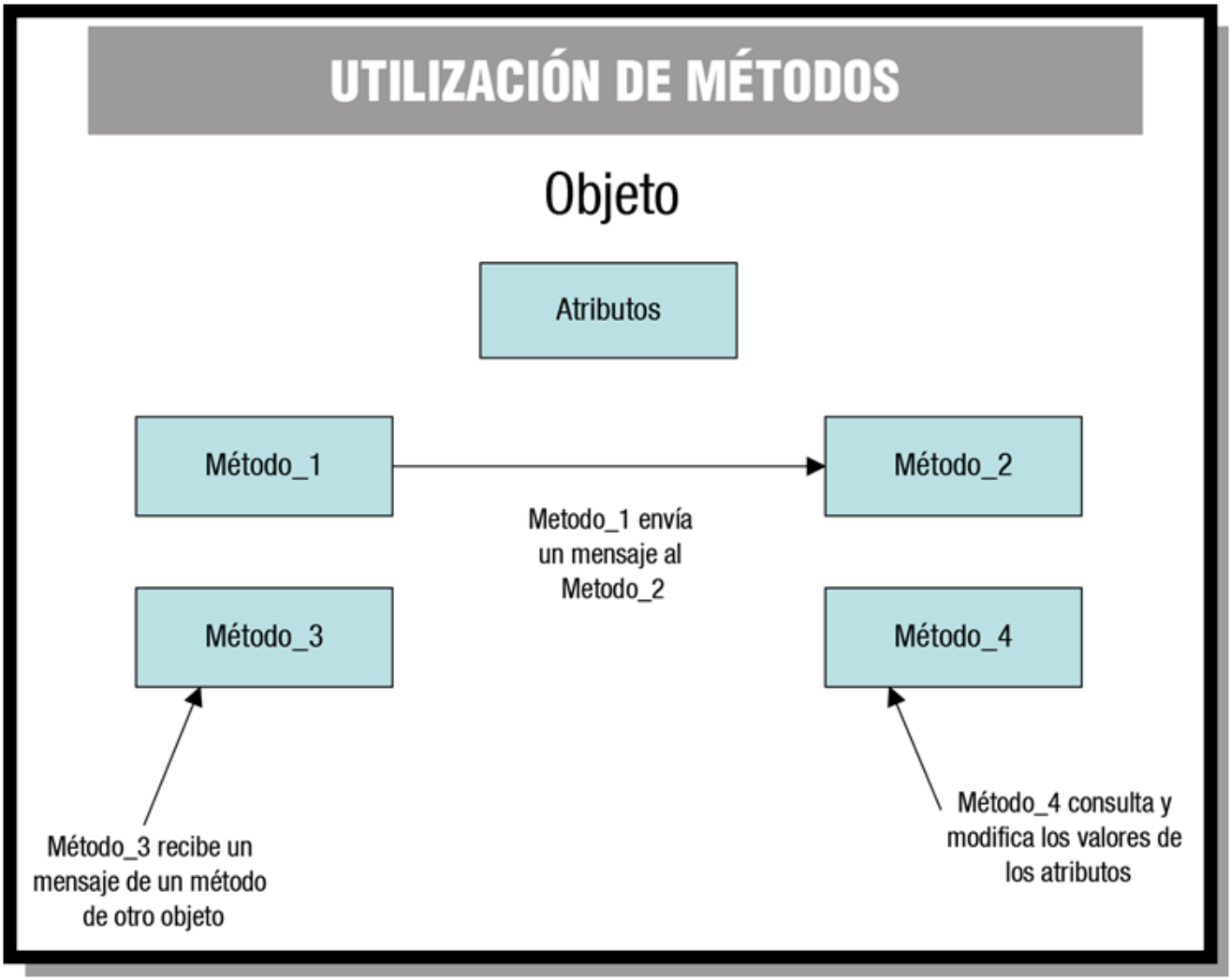


Imagen extraída de curso Programación del MECD.

1.1 Parámetros y valores devueltos.

Los métodos se pueden utilizar tanto para consultar información sobre el objeto como para modificar su estado. La información consultada del objeto se devuelve a través de lo que se conoce como **valor de retorno**, y la modificación del estado del objeto, o sea, de sus atributos, se hace mediante la **lista de parámetros**.

En general, la lista de parámetros de un método se puede declarar de dos formas diferentes:

- **Por valor.** El valor de los parámetros no se devuelve al finalizar el método, es decir, cualquier modificación que se haga en los parámetros no tendrá efecto una vez se salga del método. Esto es así porque cuando se llama al método desde cualquier parte del programa, dicho método recibe una copia de los argumentos, por tanto cualquier modificación que haga será sobre la copia, no sobre las variables originales.

- **Por referencia.** La modificación en los valores de los parámetros sí tienen efecto tras la finalización del método. Cuando pasamos una variable a un método por referencia lo que estamos haciendo es pasar la dirección del dato en memoria, por tanto cualquier cambio en el dato seguirá modificado una vez que salgamos del método.

En el lenguaje Java, todas las variables se pasan por valor, excepto los objetos que se pasan por referencia. En Java, la declaración de un método tiene dos restricciones:

- **Un método siempre tiene que devolver un valor** (no hay valor por defecto). Este **valor de retorno** es el valor que devuelve el método cuando termina de ejecutarse, al método o programa que lo llamó. Puede ser un tipo primitivo, un tipo referenciado o bien el tipo void, que indica que el método no devuelve ningún valor.

- **Un método tiene un número fijo de argumentos.** Los argumentos son variables a través de las cuales se pasa información al método desde el lugar del que se llame, para que éste pueda utilizar dichos valores durante su ejecución. Los argumentos reciben el nombre de **parámetros** cuando aparecen en la declaración del método.

El **valor de retorno** es la información que devuelve un método tras su ejecución.

Según hemos visto en el apartado anterior, la cabecera de un método se declara como sigue:

```
public tipo_de_dato_devuelto nombre_metodo (lista_de_parametros);
```

Como vemos, el tipo de dato devuelto aparece después del modificador public y se corresponde con el valor de retorno.

La lista de parámetros aparece al final de la cabecera del método, justo después del nombre, encerrados entre signos de paréntesis y separados por comas. Se debe indicar el tipo de dato de cada parámetro así:

```
(tipo_parámetro1 nombre_parámetro1, ..., tipo_parámetroN nombre_parámetroN)
```

Cuando se llame al método, se deberá utilizar el nombre del método, seguido de los argumentos que deben coincidir con la lista de parámetros.

La **lista de argumentos** en la llamada a un método debe coincidir en número, tipo y orden con los **parámetros** del método, ya que de lo contrario se produciría un error de sintaxis.

1.2 Constructores.

¿Recuerdas cuando hablábamos de la creación e instanciación de un objeto? Decíamos que utilizábamos el operador **new** seguido del nombre de la clase y una pareja de abrir-cerrar paréntesis. Además, el nombre de la clase era realmente el constructor de la misma, y lo definíamos como un método especial que sirve para inicializar valores. En este apartado vamos a ver un poco más sobre los constructores.

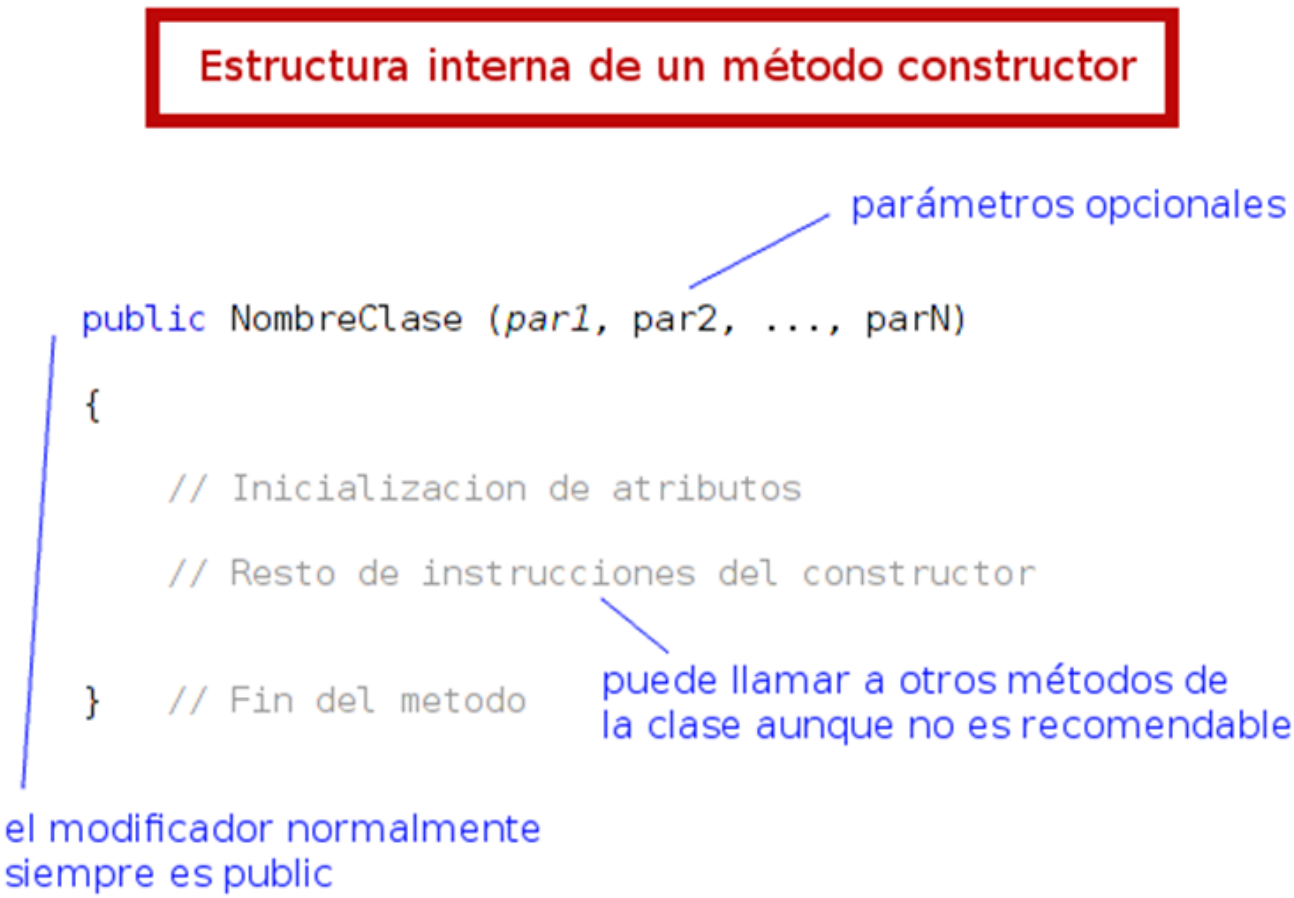
Un constructor es un método especial con el mismo nombre de la clase y que no devuelve ningún valor tras su ejecución.

Cuando creamos un objeto debemos instanciarlo utilizando el constructor de la clase. Veamos la clase `Date` proporcionada por la Biblioteca de Clases de Java. Si queremos instanciar un objeto a partir de la clase `Date` tan sólo tendremos que utilizar el constructor seguido de una pareja de abrir-cerrar paréntesis:

```
Date fecha = new Date();
```

Con la anterior instrucción estamos creando un objeto `fecha` de tipo `Date`, que contendrá la fecha y hora actual del sistema.

La estructura de los constructores es similar a la de cualquier método, salvo que no tiene tipo de dato devuelto porque no devuelve ningún valor. Está formada por una cabecera y un cuerpo, que contiene la inicialización de atributos y resto de instrucciones del constructor.



El método constructor tiene las siguientes particularidades:

- **El constructor es invocado automáticamente en la creación de un objeto,** y sólo esa vez.

- **Los constructores no empiezan con minúscula**, como el resto de los métodos, ya que se llaman igual que la clase y las clases empiezan con letra mayúscula.

- **Puede haber varios constructores** para una clase.

- Como cualquier método, el constructor puede tener **parámetros** para definir qué valores dar a los atributos del objeto.

- El **constructor por defecto** es aquél que no tiene argumentos o parámetros. Cuando creamos un objeto llamando al nombre de la clase sin argumentos, estamos utilizando el constructor por defecto.

- **Es necesario que toda clase tenga al menos un constructor**. Si no definimos constructores para una clase, y sólo en ese caso, el compilador crea un constructor por defecto vacío, que inicializa los atributos a sus valores por defecto, según del tipo que sean: 0 para los tipos numéricos, false para los boolean y null para los tipo carácter y las referencias. Dicho constructor lo que hace es llamar al constructor sin argumentos de la superclase (clase de la cual hereda); si la superclase no tiene constructor sin argumentos se produce un error de compilación.

Cuando definimos constructores personalizados, el constructor por defecto deja de existir, y si no definimos nosotros un constructor sin argumentos cuando intentemos utilizar el constructor por defecto nos dará un error de compilación.

1.3 El operador this.

Los constructores y métodos de un objeto suelen utilizar el operador `this`. Este operador sirve para referirse a los atributos de un objeto cuando estamos dentro de él. Sobre todo se utiliza cuando existe ambigüedad entre el nombre de un parámetro y el nombre de un atributo, entonces en lugar del nombre del atributo solamente escribiremos `this.nombre_atributo`, y así no habrá duda de a qué elemento nos estamos refiriendo.

Vamos a ilustrar mediante un ejemplo la utilización de objetos y métodos, así como el uso de parámetros y el operador `this`. Aunque la creación de clases la veremos en las siguientes unidades, en este ejercicio creamos una pequeña clase para que podamos instanciar el objeto con el que vamos a trabajar.

Las clases se suelen representar como un rectángulo, y dentro de él se sitúan los atributos y los métodos de dicha clase.

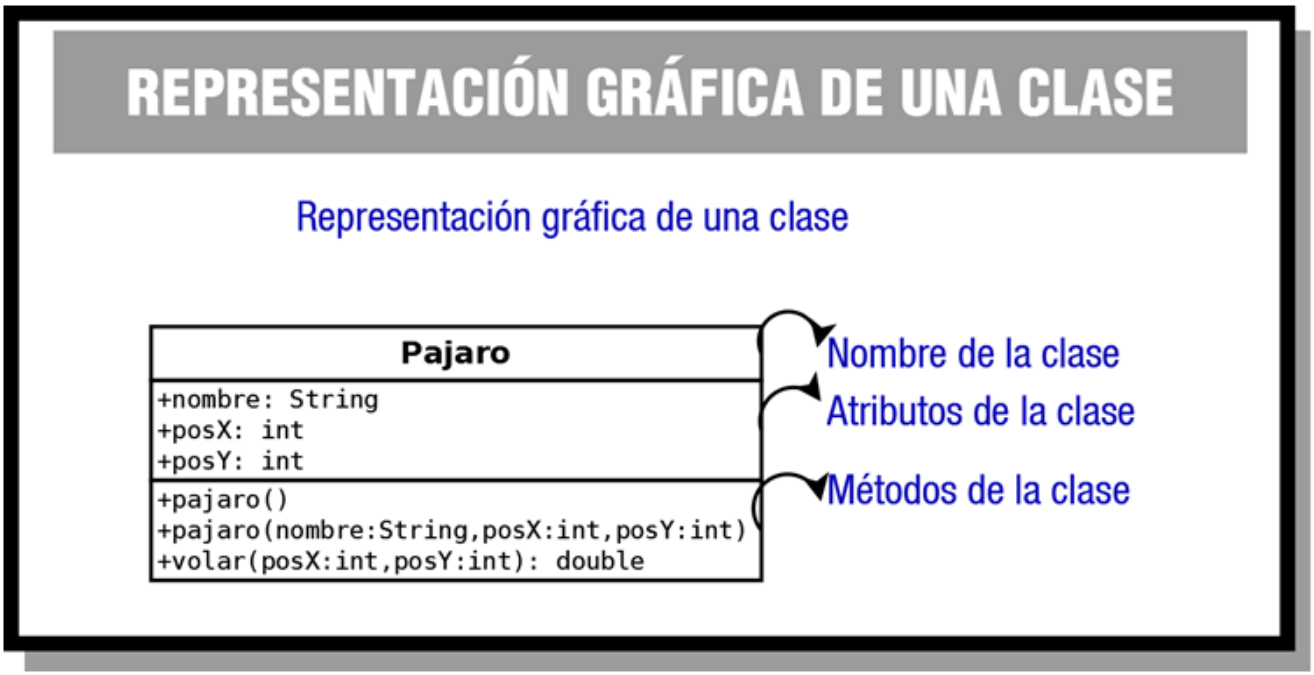


Imagen extraída de curso Programación del MECD.

En la imagen, la clase `Pajaro` está compuesta por tres atributos, uno de ellos el nombre y otros dos que indican la posición del ave, posX y posY. Tiene dos métodos constructores y un método `volar()`. Como sabemos, los métodos constructores reciben el mismo nombre de la clase, y puede haber varios para una misma clase, dentro de ella se diferencian unos de otros por los parámetros que utilizan.

Ejercicio resuelto

Dada una clase principal llamada `Pajaro`, se definen los atributos y métodos que aparecen en la imagen. Los métodos realizan las siguientes acciones:

- `pajaro()`. Constructor por defecto. En este caso, el constructor por defecto no contiene ninguna instrucción, ya que Java inicializa de forma automática las variables miembro, si no le damos ningún valor.
- `pajaro(String nombre, int posX, int posY)`. Constructor que recibe como argumentos una cadena de texto y dos enteros para inicializar el valor de los atributos.
- `volar(int posX, int posY)`. Método que recibe como argumentos dos enteros: posX y posY, y devuelve un valor de tipo double como resultado, usando la palabra clave return. El valor devuelto

es el resultado de aplicar un desplazamiento de acuerdo con la siguiente fórmula:

$$\text{desplazamiento} = \sqrt{\text{posX} \cdot \text{posX} + \text{posY} \cdot \text{posY}}$$

Diseña un programa que utilice la clase `Pajaro`, cree una instancia de dicha clase y ejecute sus métodos.

Solución:

Lo primero que debemos hacer es crear la clase Pajaro, con sus métodos y atributos. De acuerdo con los datos que tenemos, el código de la clase sería el siguiente:

```
public class Pajaro {

    String nombre;
    int posX, posY;

    public Pajaro() {

    }

    public Pajaro(String nombre, int posX, int posY) {
        this.nombre=nombre;
        this.posX=posX;
        this.posY=posY;
    }
    double volar (int posX, int posY) {

        double desplazamiento = Math.sqrt( posX*posX + posY*posY );
        this.posX = posX;
        this.posY = posY;

        return desplazamiento;
    }
}
```

Debemos tener en cuenta que se trata de una clase principal, lo cual quiere decir que debe contener un método main() dentro de ella. En el método main() vamos a situar el código de nuestro programa. El ejercicio dice que tenemos que crear una instancia de la clase y ejecutar sus métodos, entre los que están el constructor y el método volar(). También es conveniente imprimir el resultado de ejecutar el método volar(). Por tanto, lo que haría el programa sería:

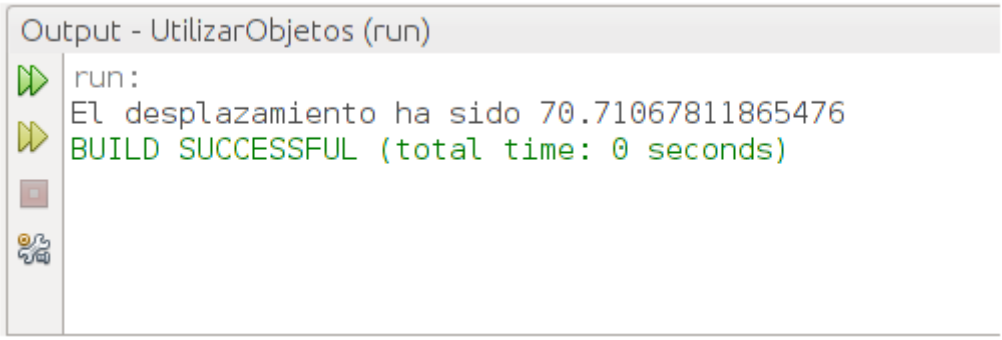
- **Crear** un objeto de la clase e inicializarlo.
- **Invocar** al método volar.
- **Imprimir** por pantalla la distancia recorrida.

Para inicializar el objeto utilizaremos el constructor con parámetros, después ejecutaremos el método volar() del objeto creado y finalmente imprimiremos el valor que nos devuelve el método. El código de la clase main() quedaría como sigue:

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

        Pajaro loro = new Pajaro("Lucy",50,50) ;
        double d = loro.volar(50,50);
        System.out.println("El desplazamiento ha sido " + d);
    }
```

Si ejecutamos nuestro programa el resultado sería el siguiente:



El contenido completo del fichero será:

```
/*
 * Programa que simula el comportamiento de un Pajaro
 */

/**
 *
 * @author FMA
 */
public class Pajaro {

    String nombre;
    int posX, posY;

    public Pajaro() {

    }

```

```
    public Pajaro(String nombre, int posX, int posY) {
        this.nombre=nombre;
        this.posX=posX;
        this.posY=posY;
    }
    double volar (int posX, int posY) {

        double desplazamiento = Math.sqrt( posX*posX + posY*posY );
        this.posX = posX;
        this.posY = posY;

        return desplazamiento;
    }

    public static void main(String[] args) {

       Pajaro loro = new Pajaro("Lucy",50,50) ;
       double d = loro.volar(50,50);
       System.out.println("El desplazamiento ha sido " + d);
    }
```

```
}
```

1.4 Métodos estáticos.

Cuando trabajábamos con cadenas de caracteres utilizando la clase `String`, veíamos las operaciones que podíamos hacer con ellas: obtener longitud, comparar dos cadenas de caracteres, cambiar a mayúsculas o minúsculas, etc. Pues bien, sin saberlo estábamos utilizando métodos estáticos definidos por Java para la clase `String`. Pero ¿qué son los métodos estáticos? Veámoslo.

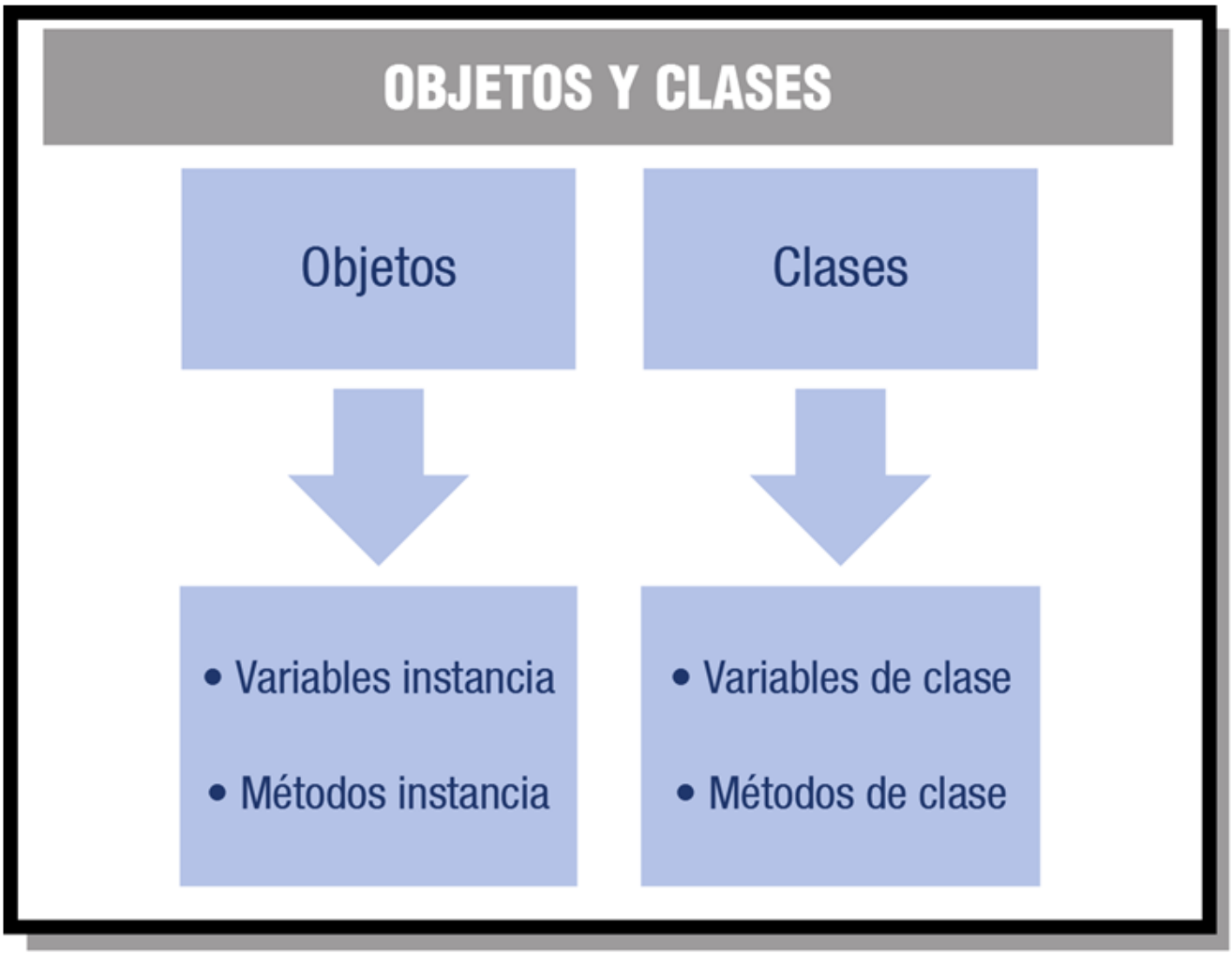


Imagen extraída de curso Programación del MECD.

Los **métodos estáticos** son aquellos métodos definidos para una clase que se pueden usar directamente, sin necesidad de crear un objeto de dicha clase. También se llaman **métodos de clase.**

Para llamar a un método estático utilizaremos:

- El nombre del método, si lo llamamos desde la misma clase en la que se encuentra definido.

- El nombre de la clase, seguido por el operador punto (.) más el nombre del método estático, si lo llamamos desde una clase distinta a la que se encuentra definido:

```
nombre_clase.nombre_metodo_estatico
```

```

```

- El nombre del objeto, seguido por el operador punto (.) más el nombre del método estático. Utilizaremos esta forma cuando tengamos un objeto instanciado de la clase en la que se encuentra definido el método estático, y no podamos utilizar la anterior:

```
nombre_objeto.nombre_metodo_estatico
```

Los métodos estáticos no afectan al estado de los objetos instanciados de la clase (variables instancia), y **suelen utilizarse para realizar operaciones comunes a todos los objetos de la clase**. Por ejemplo, si necesitamos contar el número de objetos instanciados de una clase, podríamos utilizar un método estático que fuera incrementando el valor de una variable entera de la clase conforme se van creando los objetos.

En la Biblioteca de Clases de Java existen muchas clases que contienen métodos estáticos. Pensemos en las clases que ya hemos utilizado en unidades anteriores, como hemos comentado la clase `String` con todas las operaciones que podíamos hacer con ella y con los objetos instanciados a partir de ella. O bien la clase `Math`. para la conversión de tipos de datos. Todos ellos son métodos estáticos que la API de Java define para esas clases. Lo importante es que tengamos en cuenta que al tratarse de métodos estáticos, para utilizarlos no necesitamos crear un objeto de dichas clases.

Autoevaluación

Los métodos estáticos, también llamados métodos instancia, suelen utilizarse para realizar operaciones comunes a todos los objetos de la clase.

- Verdadero
- Falso