Clases y Objetos

¿Q[ué es clase y objeto?](https://guru99.es/java-oops-class-objects/)

Las clases y los objetos son los componentes fundamentales de los POO (programación orientada a objetos). A menudo hay una confusión entre clases y objetos.

¿Qué es una clase?

¿Qué es un objeto?

¿Cuál es la diferencia entre objeto y clase?

Concepto de clases y objetos

Código de ejemplo: clase y objeto

Ejemplo de objeto y clase: principal fuera de clase

**¿Qué es una clase?**

Una clase es una entidad que determina **cómo se comportará un objeto (métodos) y qué contendrá el objeto (atributos/miembros)**. En otras palabras, es un modelo o conjunto de instrucciones para construir un tipo específico de objeto.

**Sintaxis**

|  |  |
| --- | --- |
| 1  2  3  4 | class nombre{      atributos;      constructor;  [métodos](https://eva.educamos.com/sasr/eva/mod/page/view.php?id=69349);    } |

**¿Qué es un objeto?**

Un objeto no es más que un componente autónomo que consiste en métodos y propiedades para hacer útil un tipo particular de datos. El objeto determina el comportamiento de la clase. Cuando envía un mensaje a un objeto, le pide al objeto que invoque o ejecute uno de sus métodos.

Desde el punto de vista de la programación, un objeto puede ser una estructura de datos, una variable o una función. Tiene una ubicación de memoria asignada. El objeto está diseñado como jerarquías de clase.

**Sintaxis**

|  |  |
| --- | --- |
| 1 | ClassName nombre = new ClassName(); |

¿Cuál es la diferencia entre objeto y clase?

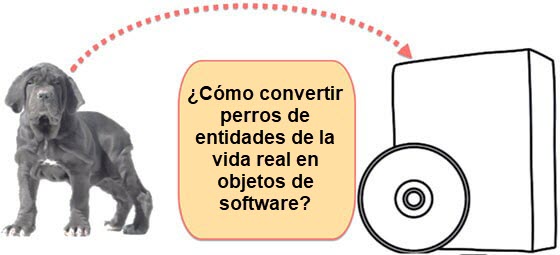
Una **clase** es un **plano o prototipo** de una entidad, que define los miembros/atributos (características), y los métodos (acciones), comunes a todos los objetos de esa clase.

Un **objeto** es un espécimen de una clase, es decir, cada uno de los entes que se crean de una clase. Los objetos de software a menudo se utilizan para modelar objetos del mundo real que se encuentran en la vida cotidiana.

Ejemplo:

Tomemos un ejemplo de cómo desarrollar un sistema de administración de mascotas, especialmente para perros. Necesitaremos información diversa sobre perros como diferentes razas de perros, edad, tamaño, etc.

Tendremos que pasar seres de la vida real, es decir, perros en entidades de software (objetos).

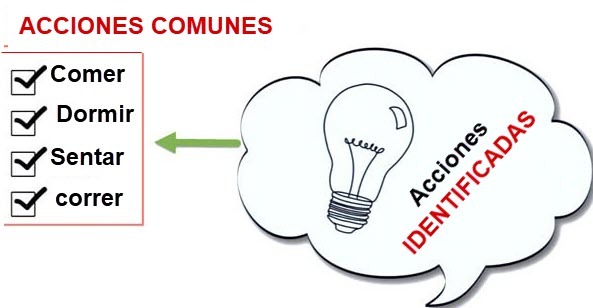


Tenemos tres razas diferentes de perros:



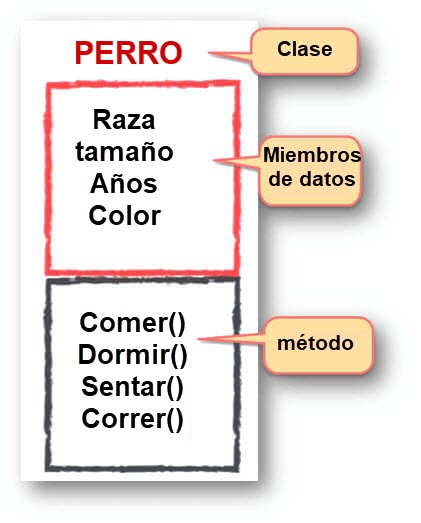
Por ejemplo, algunas de las diferencias son, la raza, la edad, el tamaño, el color, etc. Estas diferencias también son algunas de las características comunes compartidas por todos los perros. Estas características (raza, edad, tamaño, color) pueden definir un perro concreto, y por tanto un objeto.



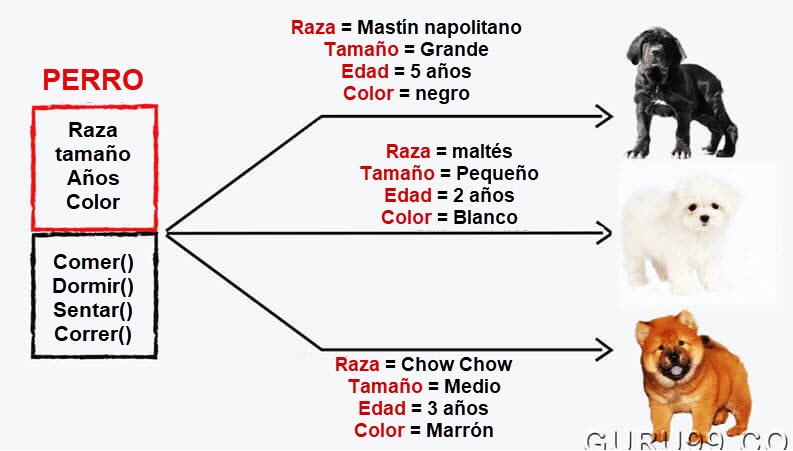
A continuación, tenemos los comportamientos comunes de estos perros, como dormir, sentarse, comer, etc. Por lo tanto, estas serán las acciones de nuestros objetos de software.

Hasta ahora hemos definido las siguientes cosas,

* **Clase** – Perros
* **Miembros/Atributos de los objetos**: tamaño, edad, color, raza.
* **Métodos (acciones)** : comer, dormir, sentarse y correr.



Ahora, para diferentes valores de miembros/atributos (tamaño de cría, edad y color) y usando esta clase Java, podremos crear diferentes objetos de perro.



Se puede diseñar cualquier programa usando este enfoque POO, y también con clases que no se asemejan a objetos del mundo real.

Código de ejemplo: clase y objeto

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | // clase Perro  public class Perro {      // Miembros o Atributos      String breed;      String size;      int age;      String color;        // método para sacar caracteristicas de un perro      public String getInfo() {          String info = "Raza: "+breed+" Tamaño:"+size+" Edad:"+age+" Color: "+color;          return info;      }          public static void main(String[] args) {          Perro maltes = new Perro();          maltes.breed="Maltes";          maltes.size="Small";          maltes.age=2;          maltes.color="white";          System.out.println(maltes.getInfo());      }  } |

**Salida:**

Raza:Maltes Tamaño:Small Edad:2 Color:blanco

**Resumen:**

* Una clase es una entidad que determina cómo se comportará un objeto y qué contendrá el objeto
* Un objeto Java es un componente autónomo que consiste en métodos y propiedades para hacer que cierto tipo de datos sean útiles.

## Métodos

Métodos:

Los **métodos son las acciones que pueden realizarse sobre los objetos**.

Cada método puede recibir valores de entrada, devolver un valor de salida, ambos, o ninguno de ellos.

Si recibe datos de entrada, estarán dentro de los paréntesis junto al nombre del método, y puede haber tantos parámetros de entrada como queramos, siempre separados por comas. En el caso de que no tenga parámetros de entrada, los paréntesis estarán vacíos ().

Si devuelve un dato de salida, antes del nombre del método deberemos poner el tipo de dato que contiene, por ejemplo:

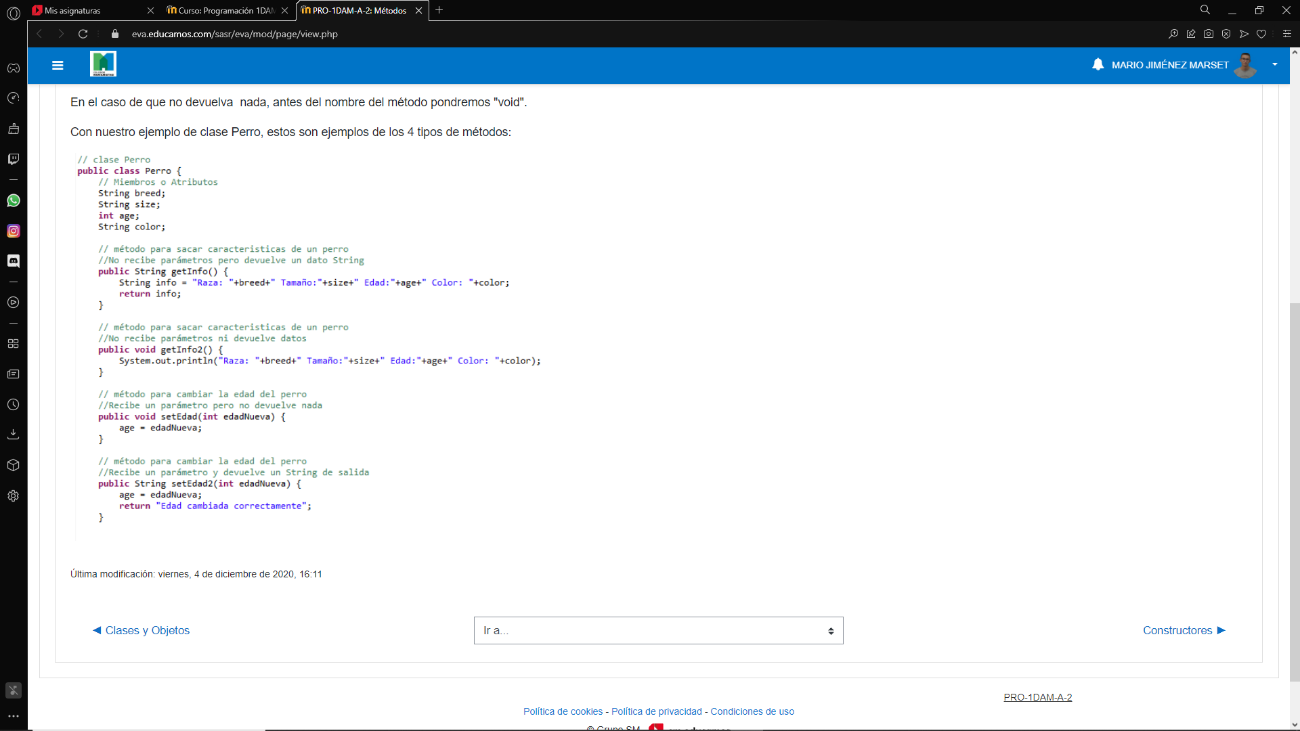
public boolean pregunta(){

      return true;

}

En el caso de que no devuelva  nada, antes del nombre del método pondremos "void".

Con nuestro ejemplo de clase Perro, estos son ejemplos de los 4 tipos de métodos:



## Constructores

## Constructores:

Siguiendo con el ejemplo anterior, veremos qué son los constructores dentro de una clase.

Un constructor sirve para "construir" el objeto, y literalmente es así porque se crea físicamente en la memoria en este paso. Se declaran como cualquier otro método pero con una peculiaridad, que **todos los constructores se llaman igual que la clase**. De esta manera los podemos distinguir de cualquier otro método de la clase.

Cuando creamos un objeto:

Perro maltes = **new**Perro();

**new**es el comando que ejecuta el controlador de la clase, y el que creará físicamente el objeto. En este caso, está ejecutando el constructor Perro(), que es el constructor por defecto.

Una clase puede tener un constructor, varios constructores, o no tener ninguno, en cuyo caso Java utiliza un constructor por defecto de esa clase.

El constructor por defecto inicializa todos los miembros/atributos con el valor por defecto, y correspondería al constructor sin parámetros de entrada:

Clase nombre = **new Clase()**;

El resto de constructores tendrán mínimo un parámetro de entrada, y **se diferenciarán entre sí, tanto por el número de parámetros de entrada, como por el tipo de cada uno de ellos, no pudiendo haber dos con el mismo número de parámetros y del mismo tipo**.

Si no ponemos constructor en una clase, Java toma el constructor por defecto, si ponemos un constructor, o varios con parámetros, obligatoriamente tenemos que implementar también el constructor por defecto.

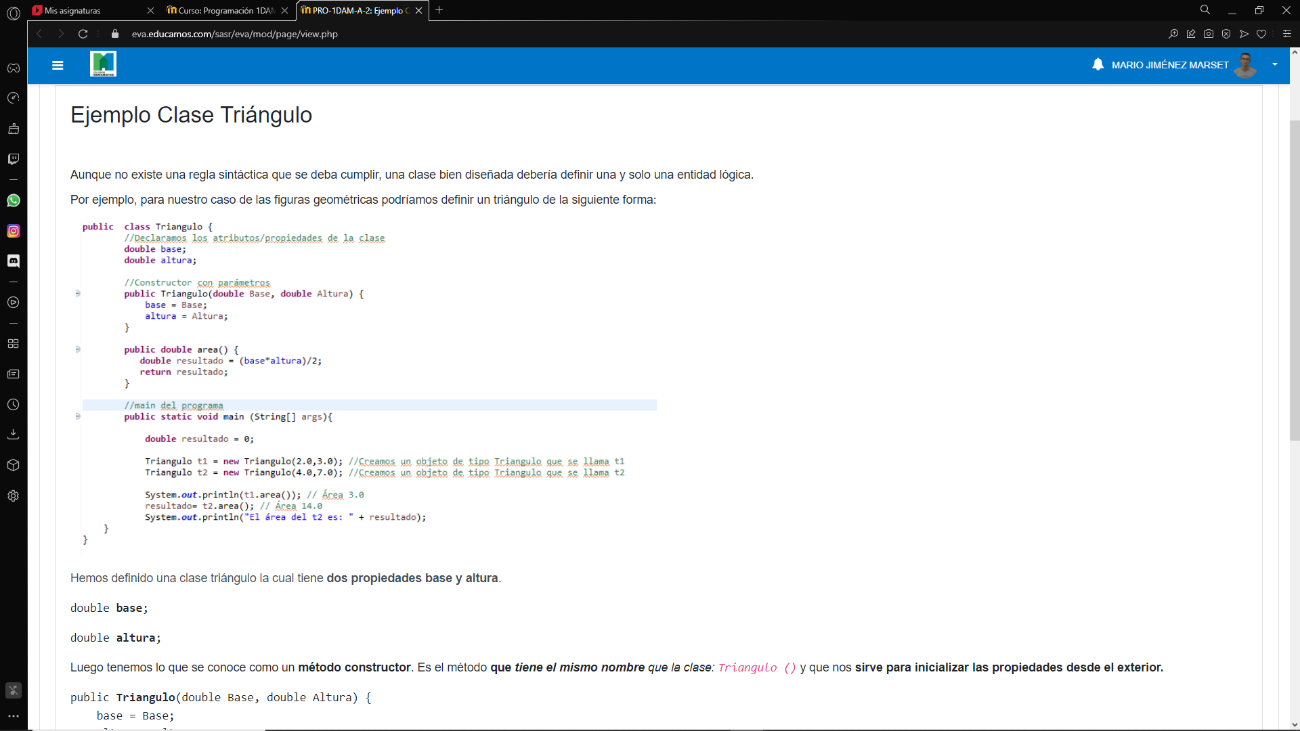
Ejemplo:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | // clase Perro  public class Perro {      // Miembros o Atributos      String breed;      String size;      int age;      String color;         //Constructor por defecto       Perro (){           breed="";           size="";           age=0;           color="";       }      //Constructor con parámetros de entrada       Perro (String raza, String talla, int edad, String colour){           breed=raza;           size=talla;           age=edad;           color=colour;       }      // método para sacar caracteristicas de un perro      public String getInfo() {          return ("Raza: "+breed+" Tamaño:"+size+" Edad:"+age+" Color: "+color);      }          public static void main(String[] args) {          Perro maltes1 = new Perro();          Perro maltes2 = new Perro("Maltes","Small";2,"white");          System.out.println(maltes1.getInfo());          System.out.println(maltes2.getInfo());      }  } |

Resultado:  
Raza: Tamaño: Edad: Color:  
Raza:Maltes Tamaño:Small Edad:2 Color:blanco

## Ejemplo Clase Triángulo

## Aunque no existe una regla sintáctica que se deba cumplir, ****una clase bien diseñada debería definir una y solo una entidad lógica****.

Por ejemplo, para nuestro caso de las figuras geométricas podríamos definir un triángulo de la siguiente forma:

Hemos definido una clase triángulo la cual tiene **dos propiedades base y altura**.

double **base**;

double **altura**;

Luego tenemos lo que se conoce como un **método constructor**. Es el método **que tiene el mismo nombre** que la clase: *Triangulo ()* y que nos **sirve para inicializar las propiedades desde el exterior.**

public **Triangulo**(double Base, double Altura) {

base = Base;

altura = Altura;

}

Además hemos creado un **método que nos calcula el área de un triángulo (base x altura / 2)**. Este método ya es público y podrá ser invocado de forma externa.

public double **area**() {

return (base\*altura)/2;

}

Vemos cómo creamos diferentes objetos del tipo Triángulo. A estos objetos los pasamos diferentes valores.

Triangulo **t1** = new Triangulo(2.0,3.0);

Triangulo **t2** = new Triangulo(4.0,7.0);

Y por último hemos invocado al método que nos devuelve el área del triángulo del objeto en concreto.

**t1**.area(); // Área 3.0

**t2**.area(); // Área 14.0

## Manejo de Excepciones

Java es un lenguaje compilado, por tanto durante el desarrollo pueden darse dos tipos de errores: los de tiempo de compilación y los de tiempo de ejecución. En general es preferible que los lenguajes de compilación estén diseñados de tal manera que la compilación pueda detectar el máximo número posible de errores. Es preferible que los errores de tiempo de tiempo de ejecución se deban a situaciones inesperadas y no a descuidos del programador. Errores de tiempo de ejecución siempre habrá, y su gestión a través de excepciones es fundamental en cualquier lenguaje de programación actual.

## Errores en tiempo de ejecución: Excepciones

Los errores en tiempo de ejecución son aquellos que ocurren de manera inesperada: disco duro lleno, error de red, división por cero, cast inválido, etc. Todos estos errores pueden ser manejados a través de excepciones. También hay errores debidos a tareas multihilo que ocurren en tiempo de ejecución y no todos se pueden controlar. Por ejemplo, un bloqueo entre hilos sería muy difícil de controlar y habría que añadir algún mecanismo que detecte esta situación y mate los hilos que corresponda.

Las excepciones son eventos que ocurren durante la ejecución de un programa y hacen que éste salga de su flujo normal de instrucciones. Este mecanismo permite tratar los errores de una forma elegante, ya que separa el código para el tratamiento de errores del código normal del programa. Se dice que una excepción es lanzada cuando se produce un error, y esta excepción puede ser capturada para tratar dicho error.

### **Tipos de excepciones**

Tenemos diferentes tipos de excepciones dependiendo del tipo de error que representen. Todas ellas descienden de la clase Throwable, la cual tiene dos descendientes directos:

* **Error**: Se refiere a errores graves en la máquina virtual de Java, como por ejemplo fallos al enlazar con alguna librería. Normalmente en los programas Java no se tratarán este tipo de errores.
* **Exception**: Representa errores que no son críticos y por lo tanto pueden ser tratados y continuar la ejecución de la aplicación. La mayoría de los programas Java utilizan estas excepciones para el tratamiento de los errores que puedan ocurrir durante la ejecución del código.

Dentro de estos grupos principales de excepciones podremos encontrar tipos concretos de excepciones o bien otros grupos que a su vez pueden contener más subgrupos de excepciones, hasta llegar a tipos concretos de ellas. Cada tipo de excepción guardará información relativa al tipo de error al que se refiera, además de la información común a todas las excepciones. Por ejemplo, una ParseException se suele utilizar al procesar un fichero. Además de almacenar un mensaje de error, guardará la línea en la que el parser encontró el error.

### **Captura de excepciones**

Cuando un fragmento de código sea susceptible de lanzar una excepción y queramos tratar el error producido, podremos hacerlo mediante la estructura try-catch-finally, que consta de tres bloques de código:

* **Bloque try:** Contiene el código regular de nuestro programa que puede producir una excepción en caso de error.
* **Bloque catch:**Contiene el código con el que trataremos el error en caso de producirse.
* **Bloque finally:** Este bloque contiene el código que se ejecutará al final tanto si se ha producido una excepción como si no lo ha hecho. Este bloque se utiliza para, por ejemplo, cerrar algún fichero que haya podido ser abierto dentro del código regular del programa, de manera que nos aseguremos que tanto si se ha producido un error como si no este fichero se cierre. **El bloque finally no es obligatorio ponerlo.**

Para el bloque catch además deberemos especificar el tipo o grupo de excepciones que tratamos en dicho bloque, pudiendo incluir varios bloques catch, cada uno de ellos para un tipo/grupo de excepciones distinto. La forma de hacer esto será la siguiente:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | try {      // Código regular del programa      // Puede producir excepciones  } catch(TipoDeExcepcion1 e1) {      // Código que trata las excepciones de tipo      // TipoDeExcepcion1 o subclases de ella.      // Los datos sobre la excepción los encontraremos      // en el objeto e1.  } catch(TipoDeExcepcion2 e2) {      // Código que trata las excepciones de tipo      // TipoDeExcepcion2 o subclases de ella.      // Los datos sobre la excepción los encontraremos      // en el objeto e2.  } catch(Exception e) {      // Código que trata las excepciones de tipo      // general que no hayan sido tratatas anteriormente.      // Los datos sobre la excepción los encontraremos      // en el objeto e.  } finally {      // Código de finalización (opcional)  } |

Si como tipo de excepción especificamos un grupo de excepciones este bloque se encargará de la captura de todos los subtipos de excepciones de este grupo. Por lo tanto, si especificamos Exception capturaremos cualquier excepción, ya que esta es la que engloba a todas las excepciones.

En el bloque catch pueden ser útiles algunos métodos de la excepción (que podemos ver en la API de la clase padre Exception):

|  |  |
| --- | --- |
| 1  2 | String getMessage()  void printStackTrace() |

con getMessage obtenemos una cadena descriptiva del error (si la hay). Con printStackTrace se muestra por la salida estándar la traza de errores que se han producido (en ocasiones la traza es muy larga y no puede seguirse toda en pantalla con algunos sistemas operativos).

Un ejemplo de uso:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | try {      int a=0, b=7, c=0;      a=b/c; //provocará la excepción por dividir entre 0  } catch (Exception e) {      System.out.println ("El error es: " + e.getMessage());      System.out.println ("Error al dividir por cero");      e.printStackTrace();  } |

Nunca deberemos dejar vacío el cuerpo del catch, porque si se produce el error, nadie se va a dar cuenta de que se ha producido.

A continuación vemos todas de las excepciones, pudiendo inspeccionarlas en la API de Java:

