

## ¿Qué es un objeto y cuál es la diferencia entre un objeto y una clase?

Un objeto en java es lo mismo que un objeto e la vida real. No tiene existencia física (en el sentido de que no puedes tocarlo como puedes tocar una lámpara), pero sí existe como "algo" que contiene información sobre sí mismo y sobre su estado, y con el que se puede interactuar.

Si queremos crear un objeto en Java, primero debemos crear un molde o receta, donde vamos a especificar el comportamiento del, así como sus características. Utilizando este molde podremos producir tantos objetos del mismo tipo como lo permita la memoria del dispositivo. A este molde se le denomina clase en Java.

Un ejemplo podría ser una lámpara, su comportamiento podría ser encender/apagar, mientras que las características podrían ser el color.

Para poder entender esto, utilicemos otro ejemplo.

Pensemos en 3 perros diferentes, como los de la imagen:



Pensemos en las diferencias entre cada uno.

Algunos ejemplos pueden ser **raza, tamaño, color, edad**.

Estas diferencias también son características comunes compartidas. A esto le llamaremos *campos o atributos*. También se les denomina variables miembros. Estos datos pueden ser de tipo primitivo (boolean, int, double, char...) o, a su vez, de otro tipo de objeto (lo que se denomina agregación o composición de objetos). La idea es que un atributo representa una propiedad determinada de un objeto.

Ahora bien, pensemos en comportamientos comunes que tengan.

Algunas podrían ser **comer, dormir, sentar, correr**.

A esto le llamamos métodos o rutinas. Son componentes de un objeto que lleva a cabo una determinada acción o tarea con los atributos.

De esta manera podemos definir:

**Clase: *Perros***

**Campos Atributos: *Raza, Tamaño, Color, Años***

**Métodos: *Comer, Dormir, Jugar, Sentarse.***

Nuestra clase Perro quedaría:

```
class Perro {  
  
    String raza;  
    String tamano;  
    String color;  
    int edad;  
  
    public Perro(String raza, String tamano, String color, int edad){  
        this.raza = raza;  
        this.tamano = tamano;  
        this.color = color;  
        this.edad = edad;  
    }  
  
    public String getInfo() {  
        return("La raza del perro es "+raza+", es un perro "+tamano+" de color "+color+" y tiene "+edad+" años");  
    }  
  
    public String getDormir() {  
        return("El "+raza+" esta durmiendo");  
    }  
  
    Run | Debug  
    public static void main(String[] args) {  
        Perro dog = new Perro(raza: "Maltés", tamano: "chico", color: "blanco", edad: 15);  
        System.out.println(dog.getInfo());  
    }  
}
```

Y en este caso dog sería nuestro objeto

Por lo tanto, podemos decir que la diferencia entre objeto y clase es:

- Una clase es un plan o prototipo que define las variables y los métodos (funciones) comunes a todos los objetos de un cierto tipo.
- Un objeto es un espécimen de una clase. Los objetos de software a menudo se utilizan para modelar objetos del mundo real que se encuentran en la vida cotidiana.

## ¿Qué es un constructor?

Un constructor es un elemento de una clase cuyo identificador coincide con el de la clase correspondiente y que tiene por objetivo obligar a controlar cómo se inicializa una instancia de una determinada clase, ya que el lenguaje Java no permite que las variables miembros de una nueva instancia queden sin inicializar. Además, a

diferencia de los métodos, los constructores sólo se emplean cuando se quiere crear una nueva instancia. En nuestro ejemplo anterior esta fue nuestro constructor:

```
public Perro(String raza, String tamano, String color, int edad){
    this.raza = raza;
    this.tamano = tamano;
    this.color = color;
    this.edad = edad;
}
```

En esta le asignamos a las variables creadas los parámetros de nuestro constructor

```
public static void main(String[] args) {
    Perro dog = new Perro(raza: "Maltés", tamano: "chico", color: "blanco", edad: 15);
    System.out.println(dog.getInfo());
}
```

La sobrecarga permite que puedan declararse varios constructores (con el mismo identificador que el de la clase), siempre y cuando tengan un tipo y/o número de parámetros distinto.

Por ejemplo, para la clase Fecha se declaran dos constructores, el primero sin parámetros (por lo tanto, se redefine el constructor por defecto) y el segundo con tres parámetros:

```
/**
 * Declaracion de la clase Fecha
 */
public class Fecha {
    // Atributos o variables miembro
    private int dia;
    private int mes;
    private int anho;

    /**
     * Constructor 1
     * Asigna los valores 1, 1 y 2000 a los atributos
     * dia, mes y anho respectivamente
     */
    public Fecha() {
        this.dia = 1;
        this.mes = 1;
        this.anho = 2000;
    }
}
```

```

/**
 * Constructor 2[1]
 * @param ndia el día del mes a almacenar
 * @param nmes el mes del año a almacenar
 * @param nanho el año a almacenar[1]
 */
public Fecha(int dia, int mes, int año) {
    this.dia = dia;
    this.mes = mes;
    this.año = año;
}

public String toString() [1]{
    return this.dia + "/" + this.mes + "/" + this.año;
}
}

```

## ¿Cuáles son las diferencias entre String y StringBuilder?

Los objetos String son inmutables. Un objeto inmutable es aquel cuyo estado no se puede cambiar una vez construido.

La clase StringBuilder es similar a la clase String en el sentido de que sirve para almacenar cadenas de caracteres. Es mutable, a diferencia de String, es decir, su tamaño y contenido pueden modificarse.

No se permite instanciar directamente a una cadena como sí permiten los String. Debe crearse con alguno de sus constructores asociados.

Un StringBuilder está indexado. Cada uno de sus caracteres tiene un índice: 0 para el primero, 1 para el segundo, etc

Al utilizar un método constructor sobrecargado para inicializar variables, puede crear una nueva instancia de la clase StringBuilder, como en el siguiente ejemplo:

Por ejemplo: `StringBuilder s = new StringBuilder ();`

## ¿En términos de eficiencia, es mejor utilizar String o StringBuilder y por qué?

Cada vez que se utiliza uno de los métodos de la clase System.String, se debe crear un nuevo objeto de cadena en la memoria, lo que requiere asignar un nuevo espacio

para el nuevo objeto. En el caso de modificaciones repetidas a la cadena, la sobrecarga del sistema asociada con la creación de un nuevo objeto Cadena puede ser muy costosa. En estos casos se recomienda utilizar `System.Text.StringBuilder`.

Por ejemplo, cuando se conectan muchas cadenas en un bucle, el uso de la clase `StringBuilder` puede mejorar el rendimiento.

## ¿Cuáles son las diferencias entre un método no estático y un método estático?

Un método **estático** es un método que pertenece a una clase, pero no pertenece a una instancia de esa clase, es decir no pertenece al objeto y este método se puede llamar sin la instancia u objeto de esa clase.

En el método estático, el método solo puede acceder a atributos de datos estáticos y métodos estáticos de otra clase o la misma clase, pero no puede acceder a métodos y variables no estáticos. Además, un método estático puede reescribir los valores de cualquier atributo de datos estáticos.

En método estático, la memoria de un método estático está fijada en la RAM, por esta razón no necesitamos el objeto de una clase en la que el método estático está definido para llamar al método estático. Para llamar al método, necesitamos escribir el nombre del método seguido del nombre de la clase.

Un método `static` no puede hacer referencia a «`this`» o «`super`».

Todos los métodos en java tienen como valor predeterminado un método **no estático** sin una palabra clave estática que lo precede. Los métodos no estáticos pueden acceder a cualquier método estático y variable estática, sin crear una instancia del objeto.

En el método no estático, el método puede acceder a atributos de datos estáticos y métodos estáticos, así como a atributos y métodos no estáticos de otra clase o la misma clase, y también puede cambiar los valores de cualquier miembro de datos estáticos.

En el método no estático, la memoria del método no estático no está fija en la memoria RAM, por lo que necesitamos un objeto de clase para llamar a un método no estático. Para llamar al método, necesitamos escribir el nombre del método seguido del nombre del objeto de la clase.

## ¿Qué es el bloque `static` y para que nos sirve?

También conocido como inicializador estático es aquel bloque de código que pertenece a una clase o interface y se define entre llaves de código precedidas por la palabra reservada `static` como se presenta el siguiente fragmento de código:

```

public class Ejemplo{
    static int numero;

    static{
        System.out.println("Esto se imprime dentro de un bloque estático o inicializador estático");
        numero = 10;
    }

    public static void main(String [] args){

    }
}

```

Su tarea principal es la de inicializar las variables estáticas, por ello son conocidos como inicializadores estáticos, en el ejemplo se inicializa la variable "numero".

En comparación con la inicialización dentro de un método estático podemos mencionar que el bloque estático solo se ejecuta una sola vez cada que la clase a la que pertenece es cargada por un classloader, y el método estático se puede ejecutar cuantas veces se desee, es decir el número de veces que se invoque.

Por lo tanto, el uso de bloques estáticos se adecua más para la inicialización de constantes pues solo se ejecuta una sola ocasión, lo cual no impide que también se usen bloques estáticos para dar un valor inicial a una variable que pueda cambiar durante la ejecución del programa.

Bloque estático vs Metodo estático

```

public class Ejemplo2{
    static int numero;

    static
    {
        System.out.println("Esto solo se imprime cuando la clase es cargada por un classloader");
        numero = 10; // <-- Se inicializa una sola vez.
    }

    public static void inicializar(int numero) {
        Ejemplo2.numero = numero; // <-- El metodo inicializar puede cambiar su valor cuantas veces se invoque.
        System.out.println("Valor cambio al invocar inicializar: " + Ejemplo2.numero);
    }

    public static void main(String[] args) {
        System.out.println("Valor inicial: " + Ejemplo2.numero);
        Ejemplo2.inicializar(5);
        Ejemplo2.inicializar(3);
    }
    ...
}

```

Salida:

```

Esto solo se imprime cuando la clase es cargada por un classloader
Valor inicial: 10
Valor cambio al invocar inicializar: 5
Valor cambio al invocar inicializar: 3

```

## ¿Qué es encapsulamiento y como implementarlo en una clase?

Para garantizar la seguridad de los datos, la encapsulación protege a los miembros de los datos del acceso no deseado al especificar modificadores de acceso y también agrupa los datos en una sola unidad.

Usando la encapsulación también podemos ocultar los miembros de datos de la clase (variables) de las otras clases. Se puede acceder a estas variables miembro de datos indirectamente utilizando métodos de la clase en la que están declaradas. A su vez, se accede a los métodos utilizando el objeto de esa clase.

Gestionando la visibilidad de los métodos y atributos que se encuentran en una clase determinada es posible ponerlos en modo privado para que se utilicen en esa clase solamente, en modo protegido permite utilizar todos los métodos, clases y atributos que se encuentren en el mismo paquete y en modo público que pueden ser usados por cualquier clase o método.

Los beneficios del encapsulamiento de objetos radican en la modularidad, esto quiere decir que el código fuente del objeto puede ser escrito, independientemente del código fuente de otros objetos, esta modularidad también aplica cuando se habla del mantenimiento y en la transferencia del objeto alrededor del sistema sin alterar su estado y conducta. El ocultamiento de la información también es considerado una ventaja del encapsulamiento ya que un objeto puede elegir tener una interfaz pública no que otros objetos eligen para comunicarse con él, cuando la configuración del objeto se encuentra en privado esta puede ser cambiada en cualquier momento sin afectar a los otros objetos que dependan de ello. La modularidad hace del encapsulamiento Java un gran beneficio, así como el ocultamiento pero las clases también juegan un papel fundamental a la hora de reutilizar el código, con estas características los programadores de software pueden utilizar la misma clase, así como el mismo código para muchos objetos al mismo tiempo.

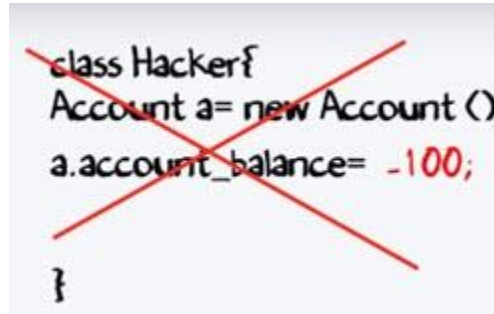
### Implementación:

```
Cuenta de clase {  
    private int account_number;  
    private int account_balance;  
  
    public void show Data () {  
        // código para mostrar datos  
    }  
    depósito público nulo (ina a) {  
        if (a <0) {  
            // muestra error  
        } else  
            account_balance = account_balance + a;  
    }  
}
```

Por lo general, una variable en una clase se establece como “privada”. Solo se puede acceder con los métodos definidos en la clase. Ninguna otra clase u objeto puede acceder a ellos.

Si un miembro de datos es privado, significa que solo se puede acceder dentro de la misma clase. Ninguna clase externa puede acceder a miembros de datos privados o variables de otra clase.

Por lo tanto, realizar algo como esto no funcionaría:



```
class Hacker{  
    Account a= new Account ();  
    a.account_balance= -100;  
}
```