Jesús Uriel Sánchez Rodríguez

ID 147822

**Variables estáticas.**

En la informática y/o programación variable estática o de clase, es una [variable](https://es.wikipedia.org/wiki/Variable_(programación)) que ha sido ubicada estáticamente y cuyo tiempo de vida se extiende durante toda la [ejecución](https://es.wikipedia.org/w/index.php?title=Ejecución_(informática)&action=edit&redlink=1) del [programa](https://es.wikipedia.org/wiki/Programa_informático), los valores que puede tomar una variable estática se establecen una vez o se pueden cambiar en múltiples ocasiones durante la ejecución del programa.

Son propias únicamente de la clase y no de los objetos que pueden crearse de la misma, por lo tanto, sus valores son compartidos por todos los objetos de la clase. U**na variable de clase es aquella que puede ser invocada sin existir una instancia.**

Para invocar a una variable estática no se necesita crear un objeto de la clase en la que se define:

* Si se invoca desde la clase en la que se encuentra definido, basta con escribir su nombre.
* Si se le invoca desde una clase distinta, debe anteponerse a su nombre, el de la clase en la que se encuentra seguido del operador punto (.) <NombreClase>.variableEstatica

**Ciclo de las variables.**

Automática. Este es el caso de las variables que se declaran dentro de una función, conocidas también como variables locales. Las variables locales automáticas mantienen su valor mientras se esté ejecutando el código de la función en que hayan sido declaradas. Cuando concluye la función, el espacio reservado en la pila se vuelve a poner a disposición del programa, y se pierde el contenido de las variables locales automáticas. Las variables locales (automáticas o estáticas) sólo se pueden utilizar en el código contenido en la función en que están declaradas.

Estática. Si se antepone la palabra reservada static a la declaración de una variable local, ésa variable pasa a crearse en la zona destinada a datos estáticos; por tanto, el espacio reservado para ella se mantendrá operativo durante toda la ejecución del programa. Por tanto, las variables estáticas se caracterizan porque su valor se mantiene entre ejecuciones de la función, esto es, su valor no se pierde cuando finaliza la ejecución de la función en que hayan sido declaradas. Las variables locales (automáticas o estáticas) sólo se pueden utilizar en el código contenido en la función en que están declaradas

Global. Si se declara una variable fuera de toda función, la variable es automáticamente estática en el sentido del párrafo anterior, y el espacio reservado para ella se mantiene operativo durante todo el programa, sí que se pierda su valor. Además, las variables globales se pueden utilizar en el código de cualquier función del programa, por tanto, su valor se puede leer y modificar desde cualquier función del programa.

Dinámica. El lenguaje C, entre otros, aporta la posibilidad de reservar espacio para almacenar variables mientras se está ejecutando el programa. Las variables que se crean de esta manera, denominadas dinámicas, siguen existiendo mientras el programa no decida lo contrario, esto es, las variables reservadas dinámicamente se pueden crear y destruir al arbitrio del programador. La memoria utilizada para una variable dinámica, cuando ésta es destruida, puede reutilizarse para crear otras variables. El acceso a las variables dinámicas se realiza a través de otras variables de un tipo especial denominado [puntero](http://maxus.fis.usal.es/fichas_c.web/07xx_PAGS/0701.html).

**Memoria dinámica.**

Es la memoria que se reserva en un determinado de ejecución tiempo. Su tamaño puede variar durante la ejecución del programa. El uso de memoria dinámica es necesario cuando no conocemos el número de datos a tratar; sin embargo es algo más lento, ya que el tiempo ejecución depende del espacio que se va a usar.

Su tamaño y forma es variable a lo largo de un programa, por lo que se crean y destruyen en tiempo de ejecución. Esto permite dimensionar la estructura de datos de una forma precisa: se va asignando memoria en tiempo de ejecución según se va necesitando.

**Objeto.**

Entidad existente en la memoria del ordenador que tiene unas propiedades llamadas atributos que son valores o características de los objetos y permiten definir el estado del objeto u otras cualidades y unas operaciones disponibles específicas llamadas métodos que en pocas palabras son acciones que puede realizar el objeto.

Se trata de un abstracto usado en programación que permite separar los diferentes componentes de un programa, simplificando así su elaboración, depuración y posteriores mejoras.  
Los objetos integran, a diferencia de los métodos, tanto los procedimientos como las variables y datos referentes al objeto.  
A los objetos se les otorga ciertas características en la vida real. Cada parte del programa que se desea realizar es tratado como objeto, siendo así estas partes independientes las unas de las otras. Los objetos se componen de 3 partes fundamentales: métodos, eventos y atributos.

**Clase.**

Abstracción que define un tipo de objeto especificando qué propiedades (atributos) y operaciones disponibles va a tener. Representan un tipo particular de objetos, cada clase tiene asociado un código, que determina los atributos que tienen los objetos de la clase, los métodos que pueden ejecutar los objetos de la clase y cómo lo hacen.

**INSTANCIA**

Se llama instancia a todo objeto que derive de algún otro. De esta forma, todos los objetos son instancias de algún otro, si bbien, decíamos que una clase es como la definición de un objeto, pero no es el objeto en sí, del modo como una idea no es una cosa física. Así que para sentarnos necesitaremos convertir esa idea en algo, en un objeto real; a ese objeto lo llamamos instancia**.**

En un mismo proyecto puedo tener una o más instancias de una misma clase sin problemas.

Cada vez que creamos una nueva instancia, ésta adquiere las propiedades, métodos y eventos de la clase a la que pertenece, sin embargo, cada instancia es independiente de las otras; esto nos da dos ventajas:

1. Si hago algún cambio en la clase, todas las instancias de esta clase se actualizarán automáticamente; esto nos permite hacer cambios sin tener que ir a cada una de las instancias.
2. Al ser independientes de las otras instancias, puedo darles valores diferentes sin que afecten a las demás (como tener una silla negra, una roja, una más alta, etc.). Aunque comparten la misma estructura, pueden programarse individualmente, dando versatilidad y flexibilidad al código.

**Herencia.**

La herencia es la transmission del código entre unas clases y otras. Para soportar un mecanismo de herencia tenemos dos clases: la clase padre y la/s clase/s hija/s. La clase padre es la que transmite su código a las clases hijas. En muchos lenguajes de programación se declara la herencia con la palabra "extends".

Eso quiere decir que todo el código de la clase padre se transmite, tal cual, a la clase hija. Si lo quieres ver así, es como si tuvieras escrito, línea a línea, todo el código de la class "Padre" dentro de las llaves de la class "Hija". Por eso, la herencia es fundamental para reutilizar código, porque no necesitas volver a incorporar el código de Padre en Hija, sino que realmente al hacer el "extends" es como si ya estuviera ahí.

**Sobre carga.**

Es un mecanismo que permite asignar el mismo nombre a [funciones](http://www.ecured.cu/index.php/Función_(programación)) distintas. Para el compilador estas funciones no tienen nada en común a excepción del [identificador](http://www.ecured.cu/index.php/Identificadores_(Informática)), por lo que se trata en realidad de un recurso semántico del lenguaje que solo tiene sentido cuando se asigna el mismo nombre a funciones que realizan tareas similares en objetos diferentes.

Por ejemplo, si tuviésemos objetos que fuesen diversos tipos de polígono (triángulo, cuadrado, pentágono, círculo, etc), tendría sentido denominar getArea a todas las funciones que calculasen el área de las diversas figuras, aunque **naturalmente** serían funciones distintas en cada caso. También tendría sentido denominar open a las funciones que abrieran un fichero o una línea de comunicación.

**Shadowing.**

En la programación de computadoras, sombras variables se produce cuando una variable declarada dentro de un determinado ámbito (bloque de decisión, el método o clase interna) tiene el mismo nombre que una variable declarada en un ámbito exterior.A nivel de los identificadores (nombres, en lugar de variables), esto se conoce como nombre de enmascaramiento. Se dice Esta variable externa a la sombra de la variable interna, mientras que el identificador interno se dice para enmascarar el identificador exterior. Esto puede llevar a confusión, ya que puede no estar claro que los usos posteriores variables del nombre de la variable en sombras se refieren a, que depende de la resolución de nombres reglas del lenguaje.

Uno de los primeros lenguajes de introducir sombras variables fue ALGOL,que introdujo por primera vez bloques de establecer ámbitos. También se le permitió por muchos de los lenguajes de programación derivados, incluyendo C ++ y Java.

El C # lenguaje rompe esta tradición, lo que permite el remedo variable entre un interior y una clase externa, y entre un método y su clase que contiene, pero no entre un si-bloque y su método contiene, o entre las declaraciones de caso en un interruptor de bloque.