# Contenido

# 1. MIEMBROS ESTÁTICOS

Durante la instanciación de un objeto, es decir, al llamar al constructor de una clase para construir un objeto, todos los miembros regulares (todos los miembros no-estáticos) que se hayan definido en la clase son construidos en el objeto recién creado.

Sin embargo, a menudo interesa definir un miembro que no se construye en los objetos creados a partir de la clase, sino que permanece definido en la propia clase. A este tipo de miembros que existen en la clase, pero no están presentes en los objetos se les denomina miembros **estáticos**.

Cualquier miembro de una clase puede ser declarado con el modificador `static`. Al aplicar este modificador, se está indicando al compilador que dicho miembro está presente **en la clase que lo contiene**, pero no estará presente en las instancias que se creen de dicha clase.

Por tanto, existirá un único miembro estático, que no estará presente en las instancias, sino que estará presente en la propia clase, al que se accederá a través del identificador de la clase.

- Un **miembro estático** representa un campo o método de una clase que **NO se construye en cada objeto** creado a partir de la clase, como ocurre con los miembros regulares, sino que se mantiene definido en la propia clase, sin crearse en los objetos y, por tanto, sólo existe un único campo/método.
- Como el miembro no pertenece a ningún objeto en particular de la clase, **se accede a él a través del identificador de la clase.**
- No tiene sentido (y se genera un error sintáctico en caso de hacerlo) acceder a un miembro estático a través de una instancia de la clase, puesto que el miembro estático no está presente en la instancia sino en la propia definición de la clase.
- Una clase que contiene algún miembro estático puede ser derivada por herencia, pero los miembros estáticos seguirán siendo accesibles únicamente a través del identificador de la clase base.

*[1]

[1] Abordaremos los mecanismos de herencia y el modo en que se comportan los miembros estáticos en relación con la herencia en un documento posterior.

## 1.1. CAMPOS ESTÁTICOS

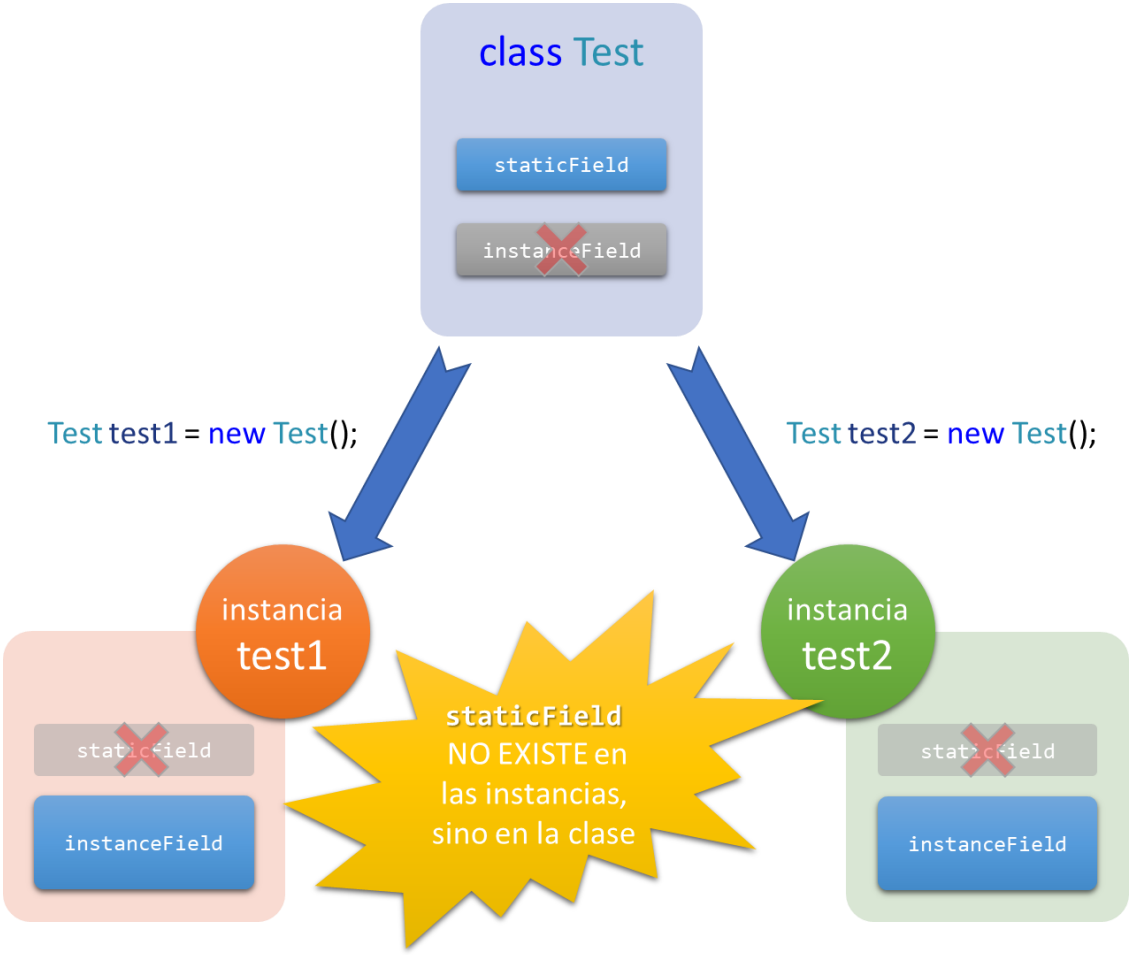
Un campo puede declararse con el modificador `static` para definirlo **a nivel de clase** en vez de hacerlo a nivel de instancia. Al declarar el campo estático, existirá **un único valor de ese campo definido en la clase**, que no estará presente en los objetos creados a partir de esa clase.

Si definimos la siguiente clase `Test`, que contiene un campo de instancia y otro estático, e instanciamos la clase desde el programa principal, tendríamos lo siguiente:

```
public class Test
   {
   private static String staticField;

   private String instanceField;

   public void testMethod ()
      {
      // Los miembros de instancia están presentes en la instancia actual, por lo
      // que se puede acceder a ellos mediante la auto-referencia this (recuerda
      // que this. se lee como "en el objeto actual, accede al miembro…")
      this.instanceField = "Este es un campo de instancia";

      // Los miembros estáticos no pueden ser accedidos a través de una instancia.
      // La siguiente sentencia es errónea y no compila:
      // this.staticField = "este campo es estático";

      // El acceso a un miembro estático se realiza a través del identificador de
      // la clase que lo define:
      Test.staticField = "Este campo es estático";
      }
   }
```

Como puedes ver, el campo estático está definido en la clase y, por tanto, **sólo existe un campo**, **presente en la clase** y **NO en las instancias**. Por el contrario, los campos de instancia se transfieren a los objetos construidos durante la instanciación y, por tanto, cada objeto posee su campo de instancia, independiente de los demás.

De manera gráfica, ocurre lo siguiente:




### 1.1.1. ¿CUÁNDO DEBE HACERSE ESTÁTICO UN CAMPO?

Normalmente, se aplica el modificador `static` sobre un campo cuando representa un dato **común a todas las instancias, pero que no es específico de ninguna de ellas en particular**.

## 1.2. MÉTODOS ESTÁTICOS

Al igual que los campos, un método puede ser declarado estático para indicar que está presente en la clase y no en las instancias. Del mismo modo, los métodos estáticos son accesibles a través del identificador de la clase que los contiene, no pudiendo ser invocados a través de una instancia.

Al implementar métodos estáticos debes tener en cuenta que, al estar el método presente en la clase y no en las instancias, en el cuerpo del método estático **no podrás acceder a los miembros de instancia** definidos en la clase (ya que el motor de ejecución no sabe sobre *qué instancia* hay que actuar):

```
public class Test
   {
   private static String staticField;

   private String instanceField;

   public static void staticMethod()
      {
      // Desde un método estático no se puede acceder a los miembros de instancia
      // de la clase. Esto se debe a que el método estático está presente en la
      // clase, pero los miembros de instancia están presentes en las instancias.
      // Al tratar de acceder a un miembro de instancia desde un método estático,
      // el motor de ejecución no sabe cuál es la instancia a la que debe acceder
      // para leer o escribir en el miembro de instancia.
      // Por esta razón, la siguiente sentencia es errónea y no compila, ya que
      // carece de sentido:

      // this.instanceField = "Hola";

      // Los miembros estáticos son siempre accesibles (si su modificador de
      // visibilidad lo permite) a través del identificador de la clase:
      Test.staticField = "este campo es estático";
      }

   public void instanceMethod()
      {
      // Desde un método de instancia, se puede acceder a los miembros de
      // instancia:
      this.instanceField = "Hola";

      // Los miembros estáticos son siempre accesibles (si su modificador de
      // visibilidad lo permite) a través del identificador de la clase:
      Test.staticField = "este campo es estático";
      Test.staticMethod();
      }
   }
```

### 1.2.1. ¿CUÁNDO DEBE HACERSE ESTÁTICO UN MÉTODO?

En general, un método que **no accede al *estado de la instancia***, es decir, que no accede a los atributos de la instancia, es un buen candidato para ser marcado como estático

## 1.3. CONSTRUCTOR ESTÁTICO

Si una clase posee algún campo estático, puede declarar un ***constructor estático*** para inicializar esos miembros estáticos. Este constructor se ocupa de dar un valor inicial a los miembros estáticos (y sólo a los miembros estáticos de la clase), antes de que la clase pueda ser invocada.

El constructor estático se ejecuta **sólo una vez**, ***antes*** de que la clase sea invocada de alguna forma por primera vez en el programa (se ejecuta ***antes*** que cualquier constructor de instancias de la clase en la que se encuentra). Esta invocación automática, transparente para el programador, la realiza el motor de ejecución.

Por tanto, tenemos la seguridad de que los miembros estáticos estarán inicializados con el valor que hayamos indicado en constructor estático antes de cualquier uso que se pueda hacer de la clase.

Dado que este constructor es invocado automáticamente por el motor de ejecución antes que cualquier otra invocación a la clase, el constructor estático **no puede recibir parámetros** (ya que no podemos llamarlo desde nuestro código, y, por tanto, no podemos pasarle parámetros). Por la misma razón, **tampoco puede verse afectado por modificadores de visibilidad**.

La sintaxis básica para definir un constructor estático es la siguiente:

```
[modificadores] class <identificador_clase>
   {
   [...]
   // El constructor estático se define con la palabra clave static seguida del
   // identificador de la clase:
   static <identificador_clase> ()
      {
      [inicialización_de_miembros_estáticos]
      }

   [...]
   }
```

De manera alternativa al uso del constructor estático, es posible inicializar los miembros estáticos ***inline***. Esta forma de hacerlo, es completamente equivalente al uso del constructor estático e impone una ligera **mejora de rendimiento** (imperceptible) respecto a la inicialización a través del constructor estático.

En el ejemplo siguiente se muestra una inicialización estática mediante constructor estático y también inline:

```
public class Example
   {
   // Inicialización estática inline
   private static String staticField1;

   private static String staticField2 = "inicialización estática inline";

   public String instanceField;

   // constructor estático: inicializa los miembros estáticos
   static Example()
      {
      saticField1 = "esta campo es estático";
      }

   // constructor de instancias: inicializa los miembros de instancia
   public Example()
      {
      this.instanceField = "este campo es de instancia";
      }
   }
```

Recuerda:

- La **inicialización** de un campo estático puede realizarse desde un **constructor estático** o bien mediante una ***inicialización inline***.
- Siempre que sea posible, es conveniente inicializar el valor de un campo estático mediante una **inicialización *inline***, es decir, se debe declarar e inicializar el campo en la misma sentencia. Hacerlo desde el constructor estático impone una ligera (imperceptible) penalización al rendimiento.

Observa que podrías realizar la inicialización estática en el constructor de instancias, pero no tendría sentido: se estaría realizando cada vez que se instanciase un nuevo objeto y, probablemente, ese comportamiento carece de sentido, ya que el miembro estático no se crea en cada objeto, sino que sólo hay un miembro estático definido en la clase y se estaría redefiniendo con cada instanciación.

En ausencia de una inicialización estática, los miembros estáticos, como cualquier otro miembro, toman el valor por defecto correspondiente a su tipo de dato.

## 1.4. CONSTANTES

Como hemos visto en documentos previos, las constantes definidas a nivel de clase se ven afectadas por los modificadores `final` y `static`. Por tanto, todas estas constantes son **estáticas**. Esto quiere decir que el compilador las define a nivel de clase (y no de instancia), y, por tanto, son accesibles a través del identificador de la clase que las define (y no a través de las instancias).

Por el contrario, las constantes definidas en el cuerpo de un método no se ven afectadas por el modificador `static`, sino únicamente por el modificador `final`, por lo que se comportan como variables automáticas de sólo-lectura

# 2. CLASES ESTÁTICAS

Una clase estática es aquella que carece de estado interno, siendo todos sus miembros estáticos. Java no tiene en cuenta este concepto de clase estática y no es posible calificar con un modificador a este tipo de construcciones, por lo que se pierde semántica y, además, el compilador no puede realizar ciertas optimizaciones.

## 2.1. STATIC CLING

***Static cling*** es un "*bad smell*" o **antipatrón** que hace referencia a un efecto de acoplamiento indeseable derivado del acceso a miembros estáticos **globales**.

Un ejemplo típico de static cling se da cuando, en el código de un programa, se crea una clase estática para almacenar un conjunto de constantes y miembros estáticos públicos, que pueden ser invocados desde distintos puntos del programa.

Este tipo de diseño dificulta la realización de pruebas automatizadas e introduce dependencias (acoplamientos) que ocasionan cambios en cascada cada vez que se tienen que introducir modificaciones en el código.

Una alternativa al uso de clases estáticas es el empleo de **inyección de dependencias**.

# 3. GESTION DE LOS MIEMBROS ESTÁTICOS EN MEMORIA

La carga de un tipo regular (una clase no-estática) en memoria **es responsabilidad del programador**. La carga tiene lugar cuando, en el código de un programa, se instancia un nuevo objeto del tipo regular.

Por ejemplo, si un programador declara una variable, pero olvida instanciar el objeto al que debe referenciar esa variable, el objeto no estará cargado en memoria, y cualquier intento de acceso al objeto provocará que el motor de ejecución emita una excepción `NullPointerException`.

```
// Declaración de una variable. NO hay instanciación y, por tanto, la
// variable builder referencia a null.
StringBuilder builder;

// Al intentar acceder al método Append, se produce una excepción, puesto
// que no hay un objeto sobre el que se pueda invocar al método (la
// referencia builder apunta a null, no a un objeto).
builder.append("Crash!");
```

Cuando en el código del programa se trata de acceder a un miembro de un tipo regular (un miembro no-estático), el motor de ejecución verifica, antes de acceder al miembro, que dicho miembro se encuentra cargado en memoria. En caso de no encontrarlo cargado, se emite una excepción `NullPointerException`.

A diferencia de lo que ocurre con los tipos regulares, la carga de un tipo estático (o un miembro estático) en memoria **es realizada por el motor de ejecución**. El programador no puede controlar el momento exacto en el que tiene lugar la carga en memoria del código estático, pero el motor de ejecución garantiza que el tipo o miembro estático estará cargado en memoria ***antes*** de que éste sea invocado.

```
// Acceso al miembro estático parseInt de la clase Integer.
// NO hay instanciación de un objeto Integer, porque parseInt es un miembro
// estático.
// El motor de ejecución ha cargado el miembro parseInt en algún momento
// antes de llegar a esta instrucción, por lo que no se emiten
// excepciones.
int x = Integer.parseInt(25);
```

En resumen:

- Cuando se invoca a un miembro regular (no-estático) el motor de ejecución **realiza comprobaciones para verificar que la referencia no es nula**, es decir, para verificar que el objeto se encuentra cargado en memoria.

- Cuando se invoca a un método estático, el motor de ejecución puede **omitir las comprobaciones de referencia nula**, puesto que el propio motor de ejecución se ocupa de cargar el tipo o el miembro en memoria de manera anticipada.

- Al omitir esas comprobaciones, la invocación de un miembro estático es *ligeramente* más rápida que la correspondiente invocación de un miembro regular.

Además, debes tener en cuenta lo siguiente:

- Una instancia (un objeto) de un tipo regular cargada en memoria **es liberada** (descargada de memoria) por el *recolector de basura* en **algún momento posterior a que no existan referencias apuntando a dicho objeto**.

- Un tipo estático o miembro estático **permanecerá en memoria** desde que el motor de ejecución efectúa su carga hasta que finaliza el ciclo de vida de la aplicación.

Teniendo en cuenta todo lo anterior, podemos concluir:

- Un tipo o miembro estático **se invoca de forma ligeramente más rápida** que un tipo regular.

- Un tipo o miembro estático **permanece en memoria durante más tiempo** que un tipo regular, por lo que tiene una huella de memoria mayor.