Constructores en Dart

Presentado por:

Edison Stiven Chamorro Melo  
Duvan Bedoya Bañol  
John Cesar Carrillo Guarnizo  
Fabio Andres Naranjo Morales

Presentado a:  
Edison Sandoval

Servicio Nacional de Aprendizaje

Centro de Atención al Sector Agropecuario  
ADSO

Pereira  
2024

1. **Concepto de constructor:**

Los constructores son funciones especiales *(Dart.Dev,* 2024) que crean instancias de clases.

1. **Inicialización de variables de instancia:**

La forma de inicializar una variable en la determinación de la clase es importante en algunos tipos de constructores *(Dart.Dev,* 2024), a groso modo, se esbozan los siguientes para tenerlos en cuenta:

**2.1 Inicialización Directa:**

Se Asigna valor directo a la variable.

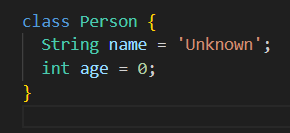


Figura 1: ilustración de asignación de inicialización directa. Fuente: elaboración propia.

**2.2 Inicialización en el Constructor:**

La forma más común de inicializar variables de instancia es hacerlo en el constructor de la clase.

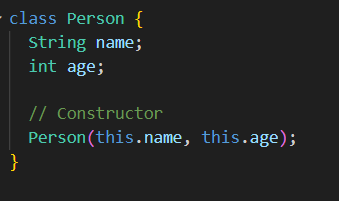


Figura 2: ilustración de asignación de Inicialización en el Constructor. Fuente: elaboración propia.

**2.3 Inicialización en la Lista de Inicialización del Constructor:**

Puedes inicializar variables de instancia en la lista de inicialización del constructor, antes del cuerpo del constructor.

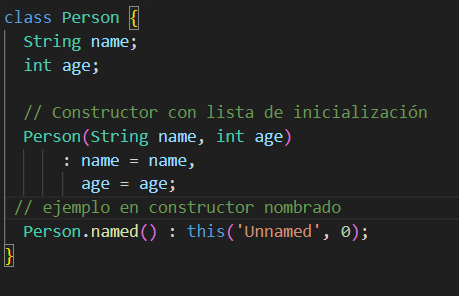


Figura 3: ilustración de asignación de Inicialización en la Lista de Inicialización del Constructor. Fuente: elaboración propia.

**2.4 Inicialización Tardía (Lazy Initialization):**

En algunas situaciones, es posible que no tengas un valor para inicializar una variable en el momento de la creación del objeto. En tales casos, puedes usar late para indicar que la variable se inicializará más tarde.

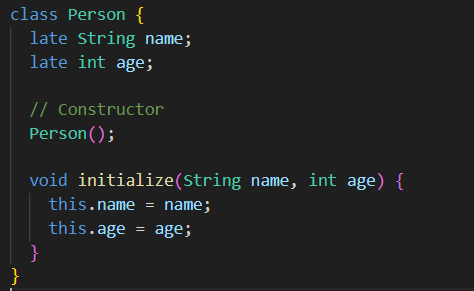


Figura 4: ilustración Inicialización Tardía. Fuente: elaboración propia.

**2.5 Inicialización de Variables Finales:**

Las variables finales deben ser inicializadas en su declaración o en el constructor. Una vez inicializadas, no pueden cambiar su valor.

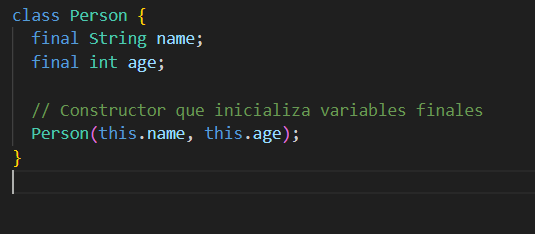


Figura 5: ilustración Inicialización de Variables Finales. Fuente: elaboración propia.

**2.6 Inicialización con Métodos de Fábrica:**

Puedes usar un constructor de fábrica para inicializar instancias de una clase, especialmente cuando necesitas controlar cómo se crean las instancias.

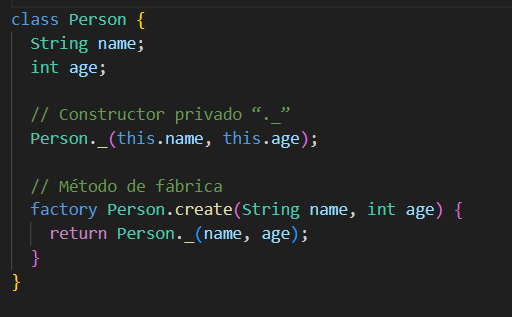


Figura 6: ilustración Inicialización con Métodos de Fábrica. Fuente: elaboración propia.

1. **Tipos de constructores:**

**3.1 Constructor por defecto:**

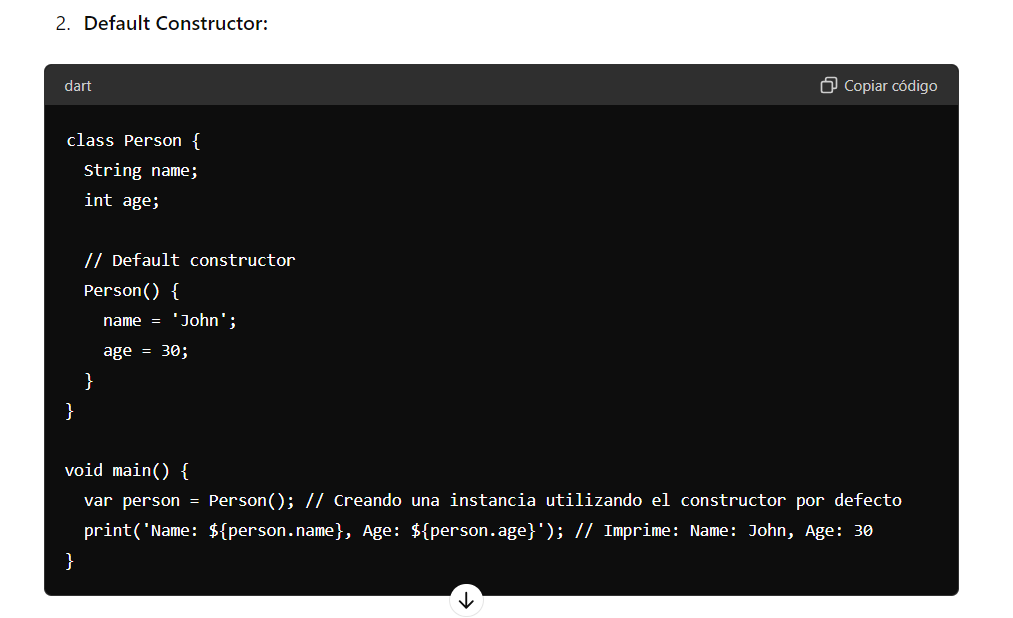
Este es el constructor que se proporciona automáticamente si no se declara ningún otro constructor en una clase (BlackeyeB, 2023), inicializa los campos de la clase con sus valores por defecto.   

Figura 7: Ilustración de constructor por defecto. Fuente: OpenAI. (2024).

**Referencia a herencia polimorfismo y encapsulamiento:**

**Encapsulamiento**: son parte de cómo se instancia una clase y pueden ser utilizados para inicializar los atributos (BlackeyeB, 2023).

**Polimorfismo y Herencia:** su relación es indirecta, ya que son el punto de partida para crear instancias de clases, que luego pueden ser tratadas polimórficamente y heredadas.

**3.2 Constructor Generativo:**

El "constructor generative" es el tipo más común de constructor, se utiliza para crear nuevas instancias de una clase y para inicializar los campos de esa instancia.

Cuando defines un constructor en una clase de Dart sin utilizar ninguna palabra clave especial (como factory o const), se está definiendo un constructor generativo *(Dart.Dev,* 2024).

Este constructor toma parámetros y puede realizar cualquier tipo de inicialización (BlackeyeB, 2023) que desees en una instancia de la clase.

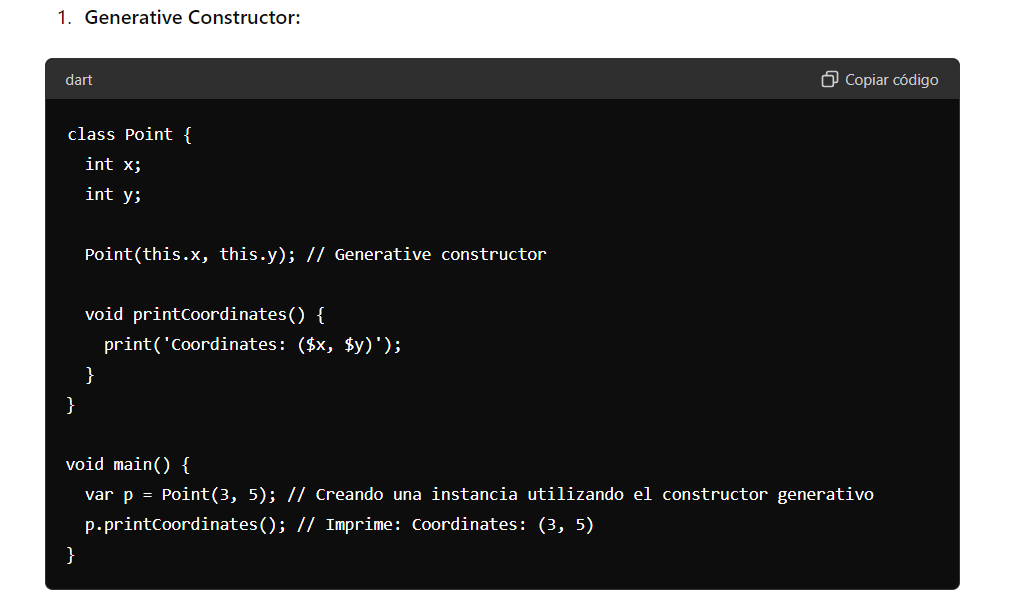


Figura 8: Ilustración de constructor generativo. Fuente: OpenAI. (2024).

**Referencia a herencia polimorfismo y encapsulamiento:**

**Encapsulamiento:** Su relación con el encapsulamiento radica en que pueden inicializar los atributos de la clase (BlackeyeB, 2023), controlando así su estado interno.

**Polimorfismo y Herencia:** Estos constructores permiten la creación de objetos de subclases que pueden ser tratados como objetos de la clase base, lo que es esencial para el polimorfismo y la herencia en Dart.

**3.3 Constructor con nombre:**

Este tipo de constructor permite definir múltiples constructores dentro de una clase con diferentes nombres *(Dart.Dev,* 2024), se utilizan para proporcionar diferentes formas de inicializar un objeto.



Figura 9: Ilustración de constructor con nombre. Fuente: OpenAI. (2024).

**Referencia a herencia polimorfismo y encapsulamiento:**

**Encapsulamiento**: pueden ayudar a organizar y definir distintas formas de inicializar una clase (BlackeyeB, 2023)**.**

**Polimorfismo y Herencia:** no tienen una relación directa, pero pueden ser utilizados para crear instancias de subclases como los demás.

**3.4   Constructor constant:**

Un "constant constructor" en Dart es un tipo especial de constructor que se utiliza para crear instancias que son constantes en tiempo de compilación, esto significa que las instancias creadas con un constant constructor son evaluadas en tiempo de compilación y se pueden utilizar en contextos donde se requieren constantes *(Dart.Dev,* 2024), como en declaraciones de variables constantes o en casos donde el rendimiento y la eficiencia son críticos; este tipo de constructor se declara utilizando la palabra clave “const” en la declaración del constructor.

Algunos otros lenguajes (BlackeyeB, 2023) de programación también se refieren a este tipo de constructor como un "constructor de tiempo de compilación" o "constructor constante".

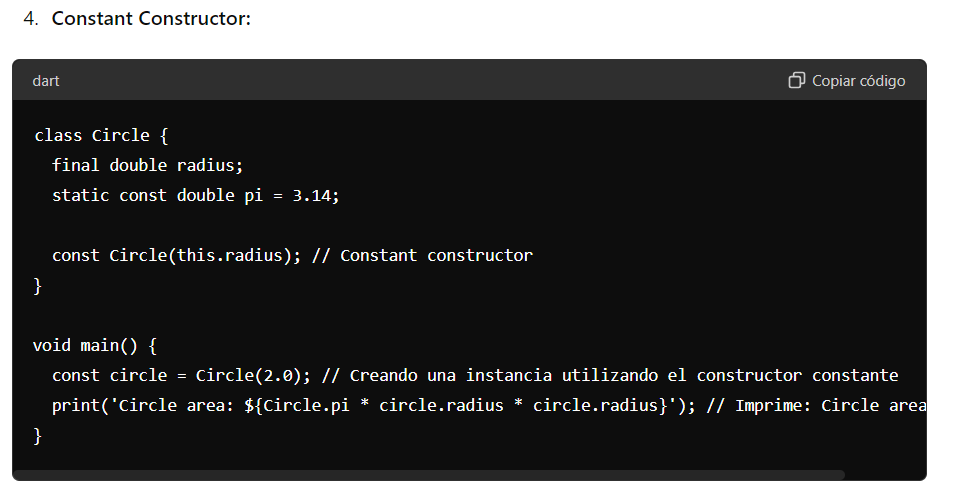


Figura 10: Ilustración de constructor constant. Fuente: OpenAI. (2024).

**Referencia a herencia polimorfismo y encapsulamiento:**

**Encapsulamiento:** no tienen una relación directa, ya que su función principal es crear instancias inmutables (BlackeyeB, 2023).

**Polimorfismo y Herencia:** no tienen una relación directa, pero pueden ser utilizados en clases que participan en jerarquías de herencia y polimorfismo como los demás.

**3.5 Constructor de redirección:**

Este constructor permite redirigir la inicialización de un objeto a otro constructor *(Dart.Dev,* 2024) en la misma clase o en una clase padre, utilizando la palabra clave  “this” ; esta define un boceto del constructor principal o sea el normal y despues se define el constructor de redirección, o sea donde van a llegar los datos (BlackeyeB, 2023), se necesita poner el nombre del constructor principal seguido de un “.” y después el nombre del constructor de redireccionamiento.

Si se debe mandar datos para ingresar, se define la cantidad de variables, que se decidan a enviar, luego se usa el símbolo “:” seguido de la palabra “this”, acto seguido, se ponen los nombres de las variables que se van a llamar o insertar (BlackeyeB, 2023) los valores directamente, después de eso, es recomendable llenarlo en una instancia, así se denota la manera de enviar los datos al constructor de redireccionamiento.

Siguiendo este procedimiento para crear constructores, se permite hacer varios constructores que deriven del normal (BlackeyeB, 2023) y así poder redireccionar diferente información a distintos lugares del código.

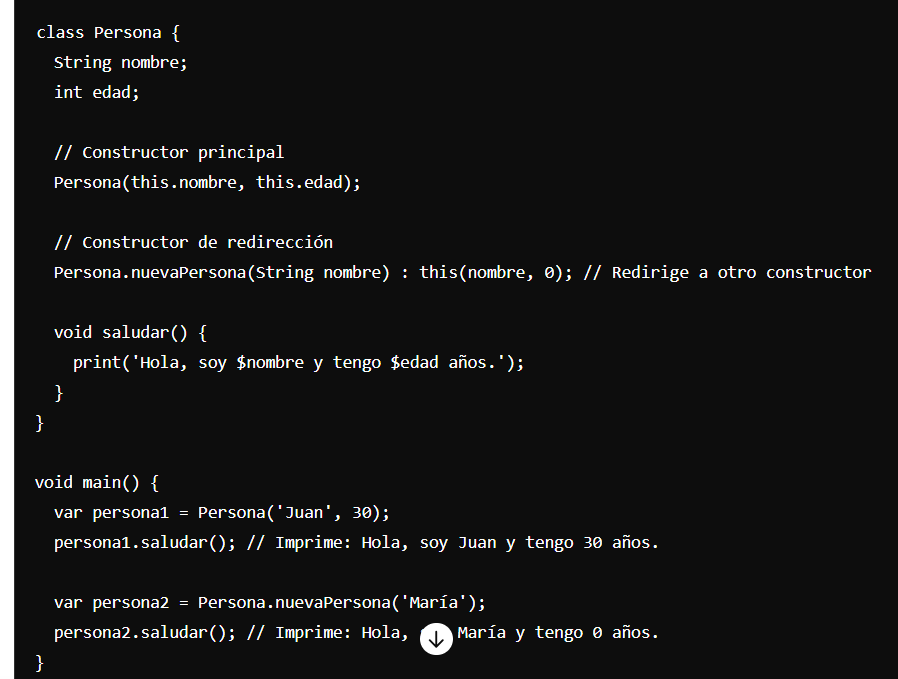


Figura 11: Ilustración de constructor de redirección. Fuente: OpenAI. (2024).

**Referencia a herencia polimorfismo y encapsulamiento:**

**Encapsulamiento:** pueden ayudar a reutilizar la lógica de inicialización en diferentes constructores de la misma clase (BlackeyeB, 2023).

**Polimorfismo y Herencia:** pueden ser utilizados en clases base y derivadas para simplificar la inicialización de objetos y facilitar la implementación del polimorfismo y la herencia.

**3.6 Constructores de fábrica:**

Estos constructores son responsables de crear instancias de una clase, ya sea creando una nueva instancia de una subclase o devolviendo una instancia existente de una caché *(Dart.Dev,* 2024), lo que significa que este constructor tiene la capacidad de decidir qué instancia devolver; se determinan al usar la palabra clave “Factory” antes del constructor.

Pueden ser utilizados para implementar patrones de diseño como el Singleton, donde solo se permite la creación de una única instancia de la clase, o para implementar la lógica de caché, donde en lugar de crear una nueva instancia cada vez, se devuelve una instancia existente desde una caché, en lugar de simplemente crear una nueva instancia cada vez que es invocado.

El constructor Factory no siempre crea una nueva instancia de su clase, no puede devolver null (BlackeyeB, 2023), podría devolver una instancia existente caché en lugar de crear una nueva instancia de un subtipo.

Necesitas realizar un trabajo no trivial antes de construir una instancia. Esto podría incluir la comprobación de argumentos o cualquier otro proceso que no pueda ser manejado en la lista de inicializadores.

**Se puede hacer redirección de los constructores de fábrica:**

Un constructor de fábrica de redirección especifica una llamada a un constructor de otra clase para usar siempre que alguien haga una llamada al constructor de redirección.

¿Dónde se podría usar una redirección de los constructores de fábrica?

Aunque los constructores de fábrica ordinarios podrían crear y devolver instancias de otras clases, hecho que haría innecesarias las fábricas de redirección, Las fábricas de redirección tienen varias ventajas:

-Una clase abstracta puede proporcionar un constructor constante que utilice el constructor constante de otra clase.

-Un constructor de fábrica redireccionable evita la necesidad de que los reenviadores repitan los parámetros formales y sus valores por defecto.

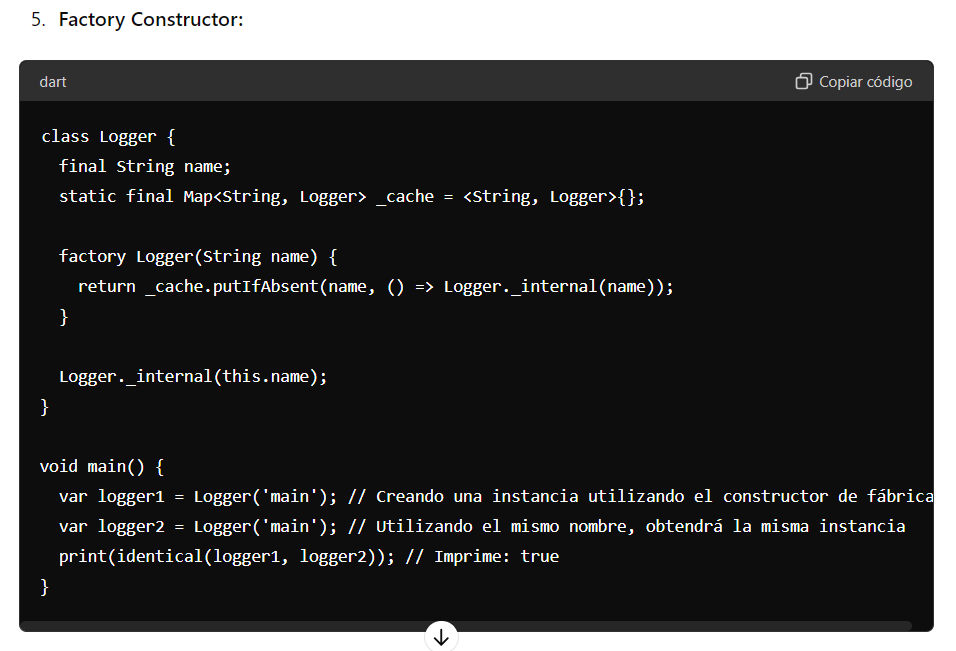


Figura 12: Ilustración de constructor Factory. Fuente: OpenAI. (2024).

**Referencia a herencia polimorfismo y encapsulamiento:**

**Encapsulamiento:** pueden controlar cómo se crean las instancias de una clase (BlackeyeB, 2023).

**Polimorfismo y Herencia:** los constructores factory pueden ser utilizados para crear instancias de clases derivadas, lo que contribuye al polimorfismo y la herencia en Dart.

**Bibliografía:**

**BlackeyeB. (2023, mayo 5). *Constructores en Dart: Casos de uso y ejemplos*. freecodecamp.org.** [**https://www.freecodecamp.org/espanol/news/constructores-en-dart-casos-de-uso-y-ejemplos/**](https://www.freecodecamp.org/espanol/news/constructores-en-dart-casos-de-uso-y-ejemplos/)

***Dart.Dev.* (2024) *Constructors*. (s/f). Dart.dev. Recuperado el 24 de mayo de 2024, de** [**https://dart.dev/language/constructors**](https://dart.dev/language/constructors)

**OpenAI. (2024). Captura de pantalla de ejemplo de constructor tomado de ChatGPT 4o. Recuperado de:**

[**https://chatgpt.com/c/bff9189d-836c-4d77-8c4e-559acf569e3b**](https://chatgpt.com/c/bff9189d-836c-4d77-8c4e-559acf569e3b) **(Fecha de acceso: 24 de mayo de 2024).**