Contents

[**Dart** 1](#_Toc50214377)

[**DartPad y qué editores utilizar** 2](#_Toc50214378)

[**Variables: Números (int, double), String y booleanos** 3](#_Toc50214379)

[**Colecciones** 3](#_Toc50214380)

[**Diccionarios** 6](#_Toc50214381)

[**Constantes (final y const)** 6](#_Toc50214382)

[**Operadores aritméticos, de asignación y lógicos** 7](#_Toc50214383)

[**Declaración de funciones y la función Main()** 13](#_Toc50214384)

[**Funciones de sintaxis corta (Arrow) y Anónimas (Lambda)** 14](#_Toc50214385)

[**Parámetros requeridos, opcionales, posicionados, nombrados y por defecto** 15](#_Toc50214386)

[**If-Else y Expresiones Condicionales** 17](#_Toc50214387)

[**Switch y Case** 19](#_Toc50214388)

[**For** 21](#_Toc50214389)

[**While y do-while** 22](#_Toc50214390)

[**Break y Continue** 23](#_Toc50214391)

[**Definición de clases y miembros** 25](#_Toc50214392)

[**Constructores por defecto, por parámetro o nombrados** 27](#_Toc50214393)

[**Constructores por defecto** 27](#_Toc50214394)

[**Constructores con parámetros** 27](#_Toc50214395)

[**Constructores nombrados** 28](#_Toc50214396)

[**Métodos Getter y Setter y variables privadas** 28](#_Toc50214397)

[**Herencia** 30](#_Toc50214398)

[**Clases Abstractas** 31](#_Toc50214399)

[**Interfaces implícitas** 31](#_Toc50214400)

[**Contenedores** 32](#_Toc50214401)

# **Dart**

Fue creado por Google en 2011 y liberado en su versión 1.0 en diciembre 2018, es un lenguaje de propósito general.

* Es un lenguaje de POO, el cual utiliza los cuatro pilares:
  + Abstracción.
  + Encapsulamiento.
  + Herencia.
  + Polimorfismo.
* Se traba con definición de clases.
* Cuenta con un recolector de basura.
* Su sintaxis es en base a C.
* Tiene un transpilador a JavaScript, esto significa que podemos convertir código Dart en código JavaScript el cual se puede ejecutar en un navegador.
* Es **JIT** (Justo en el momento) & **AOT** (Antes del momento), esto permite la potencia a flutter para desarrollar apps móviles.
  + **JIT**: Compila la aplicación en el dispositivo en momento de ejecución (interpretado).
  + **AOT**: Compila la aplicación en el momento de construcción (compilación).

**Puede ser utilizado**

* Servidor.
* Web.
* Móvil.

**La documentación**

En Dart la [documentación](https://dart.dev/guides) es muy importante lo cual nos ayudará a resolver muchas dudas ya que está es excelente, y no olvidar la característica principal de este lenguaje, **todo es un objeto**.

El lenguaje Dart sirve para trabajar en varios entornos, entre ellos:

* **Back End**: Aqueduct.
* **Front End**: AngularDart.
* **Móvil**: Flutter.

Es importante destacar que se puede transpilar a código Javascript utilizando **Dart2JS**, utilizando las herramientas **webdev** y **stagehand**.

¿Por qué Dart para Flutter?

* AOT (Ahead Of Time): Compilado a un rápido y predecible código nativo. (Totalmente personalizable)
* Puede ser JIT (Just In Time): Compilado para una velocidad excepcional de desarrollo. (Esto incluye el popular Hot Reload).
* Hace fácil la creación de animaciones y transiciones que corren a 60fps (frames per second)
* Al ser compilado a código nativo, no hay puentes innecesarios para correr el código.
* Dart le permite a Flutter evitar el desarrollo de diseños en archivos independientes como JSX, XML o bien interfaces separadas.
* Dart es relativamente fácil de aprender.

# **DartPad y qué editores utilizar**

[DartPad](https://dartpad.dev/) es una herramienta web en la que puedes editar código Dart, nos sirve para empezar bastante rápido, esto porque no necesita que instales ningún complemento, para este curso DartPad es el mínimo requerido.

Pero si eres de los que quieren utilizar un programa de Escritorio, los editores de código e IDEs modernos te permiten programar en Dart, sólo necesitas instalar el plugin correspondiente a cada uno.

# **Variables: Números (int, double), String y booleanos**

Las variables son espacios reservados en la memoria que pueden cambiar de contenido a lo largo de la ejecución de un programa.

Para declarar una variable podemos utilizar la palabra reservada **var**, seguido del nombre de la variable o también podemos escribir el tipo de variable (**int**, **double**, **String** y **bool**) seguido del nombre de la variable.

En Dart tenemos los tipos básicos de variables que encontramos en casi todos los lenguajes tales como **int**, **double**, **String** y **bool** y funcionan de la siguiente manera:

* **int** son valores numéricos enteros que no aceptan punto decimal.
* **double** son valores numéricos decimales que tienen punto decimal.
* **String** son cadenas de caracteres que aceptan textos.
* **bool** son constantes en tiempo de compilación que permiten valores verdaderos (true) o falsos (false).

Las variables de tipo **dynamic** aceptarán cualquier tipo de variable y cambiarán según la necesidad.

# **Colecciones**

Las colecciones son objetos de tipo estructuras de datos que almacenan diversos elementos, existen 2 tipos de colecciones, las listas y los sets.

En Dart los arreglos son Objetos ordenados de tipo List los cuales almacenan con un índice a cada uno de los elementos, mientras que los sets son colecciones que contiene valores únicos (no se pueden repetir) no ordenados (no se pueden recuperar mediante índices).

Una lista permitirá agregar los elementos que se desees, mientras que un set no permitirá volver a agregar un valor ya existente, pero tampoco mandará un error.

**Colecciones**

**List**

* Una de las colecciones más comunes en casi todos los lenguajes es el Array, en Dart los arreglos son objetos tipo List.
* Trabaja con index 0
* Tiene muchas funciones por defecto.

**Declaración**

List list; //forma 1

List list<tipo>; //forma 2

**Inicialización**

list = ["negro","blanco"];

**Agregar elemento**

list.add("Rosa");

**Borrar elemento**

list.removeAt(indice);

**Borrar último elemento**

list.removeLast();

**Añadir elementos de otra lista**

list.addAll(lista\_auxiliar);

**Set**

* En este tipo de colección un elemento puede existir una sola vez. Si intentamos enviar uno repetido no lo permitirá (pero tampoco mandará un error)
* Esta no regresará en el mismo orden y por este mismo motivo no se puede pedir por index

**Declaración**

Set set;

**Inicialización**

set = set.from(['Argel', 'Juan']);

**Agregar elemento**

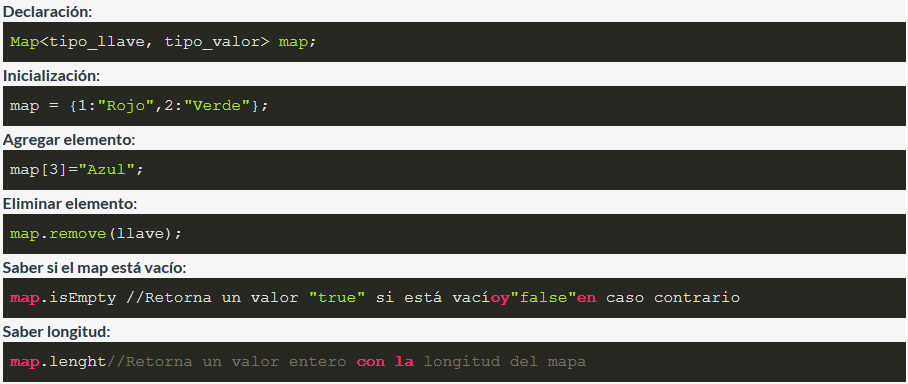
set.add("Erika");

**Eliminar elemento**

set.remove("Argel");

# **Diccionarios**

Los **Maps**, también llamadas **Hash** o **Diccionarios** son objetos que tienen una asociación entre **llaves** y **valores**.



# **Constantes (final y const)**

Las variables de tipo **const** y **final** sólo pueden ser declaradas al mismo tiempo de ser creadas, su diferencia mayor es que **final** es **inicializada** al momento de **usarse**, mientras **const** al momento de **escribirse**.

**final**: solo se pueden declarar una vez y es inicializada cuando se accede a ella.



**const**: Son implícitamente tipo **final**, pero son constantes de tiempo de compilación. Si se requiere una constante a nivel de clase, debe anteponerse el prefijo “**static**”.



**Principal diferencia**

* **final** únicamente se alojará en memoria al momento de utilizarla, es decir, que el código haga uso **realmente** de ella.
* **const** es declara y alojada en memoria al momento de ponerla en nuestro código.

# **Operadores aritméticos, de asignación y lógicos**

**Operadores**

Los operadores utilizados en Dart son los siguientes:

**Unario PostExpr**

****

**Unario PreExpr**

****

**Multiplicativo**

****

**Adición**

****

**Shift**

****

**Bitwise**

****

**Relacional y tipo prueba**

****

**Igualdad**

****

**Lógicos**

****

**Si Nulo**

****

**Condicional**

****

**Cascada**

****

**Asignación**

****

Con los operadores puedes crear expresiones, algunos ejemplos son:

**a++**

**a + b**

**a = b**

**a == b**

**c ? a : b**

**a is T**

En la tabla de operadores, cada operador tiene mayor precedencia que los operadores de las filas siguientes. Por ejemplo, el operador multiplicativo % tiene mayor precedencia que (y por lo tanto ejecuta antes) el operador de igualdad ==, que tiene mayor precedencia que el operador lógico AND (&&). Esa precedencia significa que las siguientes dos líneas de código se ejecutan de la misma manera:

// Los paréntesis mejoran la lectura



// Mas difícil de leer, pero equivalente.



**Operadores Aritméticos**

Dart soporta los operadores aritméticos habituales:

* Suma ( +)
* Resta ( - )
* -expr Unario menos también conocido como negación (invierte el signo de la expresión)
* \* Multiplicación
* / División
* ~/ Divide, regresando el valor entero
* % regresa el restante del entero en una división (modulo)

Dart también soporta operadores de incrementos y decrementos de prefijos y postfijos.

* **++var** El cual se leería; var = var + 1 (expression value is var + 1)
* **var++** El cual se leería; var = var + 1 (expression value is var)
* **--var** El cual se leería; var = var – 1 (expression value is var – 1)
* **var--** El cual se leería; var = var – 1 (expression value is var)

**Operadores de Igualdad y relacionales**

Enumeraremos el significado de la igualdad y de los operadores relacionales.

* == Igual.
* != No igual (diferente)
* > Mayor que.
* < Menor que.
* >= Mayor que o igual a.
* <= Menor que o igual a.

**Operadores de prueba de tipo**

Los operadores **as**, **is**, **and is!** son útiles para verificar los tipos en tiempo de ejecución.

**as** Typecast (También utilizado para especificar prefijos de biblioteca)

**is** Verdadero si el objeto es el tipo especificado.

**is!** Falso si el objeto tiene el tipo especificado.

**Operadores de Asignación**

Como ya has visto, puedes asignar valores usando el operador =. Para asignar sólo si la variable asignada es **nula**, utilice el operador **??=**

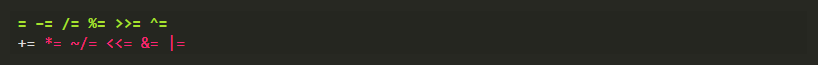
**Asigna valor a variable a**

a = value;

**Asigna valor a b, si b es nulo; de lo contrario b se mantiene igual**

b ??= value;

**Operadores que existen**



//Así es como trabajan

Donde **op** es un operador a **op**= b a = a **op** b

Ejemplo a += b a = a + b

**Operadores Lógicos**

Puedes invertir o combinar expresiones booleanas utilizando los operadores lógicos.

* **!expr** invierte la expresión que le sigue (cambia falso a verdadero y viceversa)
* **||** OR lógico
* **&&** AND lógico

**Operadores Bitwise y Shift**

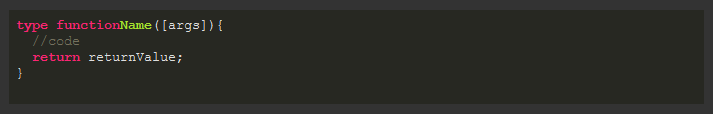
Puedes manipular los bits individuales de los números en Dart. Por lo general, se utilizan estos operadores de bits y de desplazamiento con enteros.

* **&** AND
* **|** OR
* **^** XOR
* **~expr** Complemento unario bitwise (0s a 1s; 1s a 0s)
* **<<** Desplazamiento hacia la izquierda.
* **>>** Desplazamiento hacia la derecha.

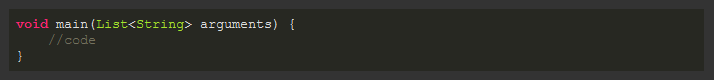
# **Declaración de funciones y la función Main()**

Las funciones son una colección de declaraciones agrupadas para desarrollar una operación, éstas pueden ser llamadas y reutilizadas.

Una función tendrá un tipo (mismo que regresará al utilizarse), un nombre, argumentos e instrucciones y seguirá la siguiente estructura.



**main()** es la función de alto nivel que define el inicio de la aplicación y sigue la siguiente estructura, recuerda que en web no se requieren los argumentos.



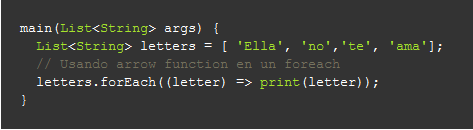
**Nota**: En Dart las funciones también son objetos, así que las podemos asignar a una variable.

# **Funciones de sintaxis corta (Arrow) y Anónimas (Lambda)**

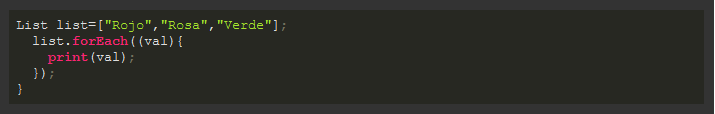
Las **funciones de** **sintaxis corta** (Arrow) son declaradas en la misma línea y sólo pueden tener una expresión, misma que se declara después de una flecha (**=>**), por ejemplo:



Utilizando **foreach** en Arrow.

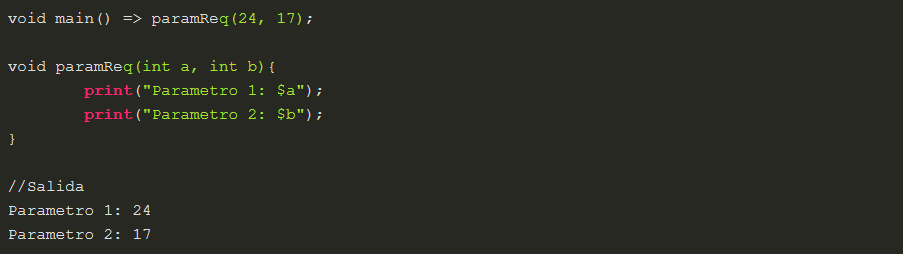


Las **funciones anónimas**, también conocidas como **lambda** o closures son similares a las nombradas, pero con la diferencia de que no llevan nombre y suelen utilizarse para realizar una acción dentro de otro proceso, por ejemplo:



# **Parámetros requeridos, opcionales, posicionados, nombrados y por defecto**

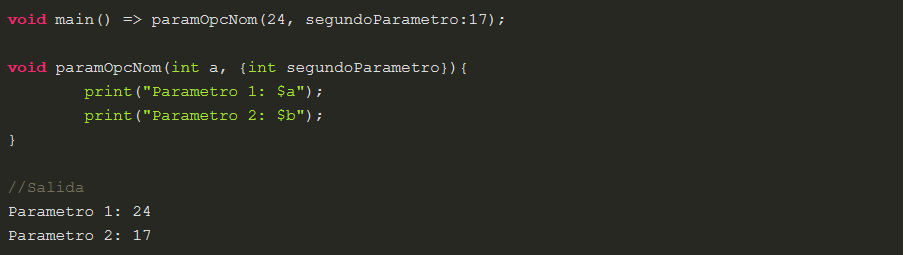
**Parámetros requeridos**: Al momento de invocar la función deben enviarse sí o sí, o el compilador indicará un error.



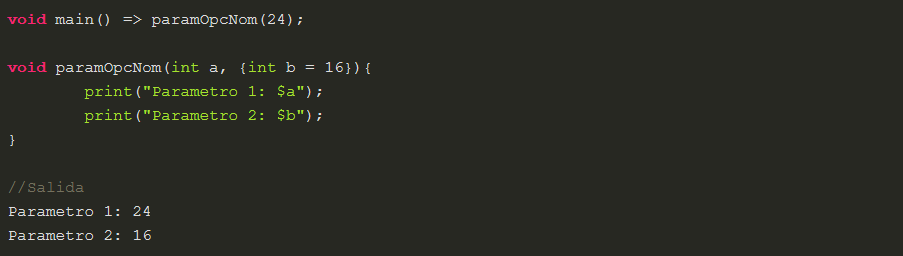
**Parámetros opcionales posicionados**: Los corchetes indican que el parámetro es opcional posicionado. Por lo tanto, no es obligatorio invocar la función con ese parámetro.



**Parámetros opcionales nombrados**: Poseen la misma lógica que el anterior tipo, pero aquí podemos nombrarlos al momento de la invocación.



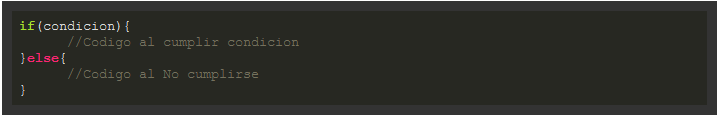
**Parámetros opcionales por defecto**: Poseen la misma lógica que el anterior tipo nombrado, pero podemos asignar un valor por defecto si no se invoca a la función con otro diferente.



# **If-Else y Expresiones Condicionales**

El **if**/**else** ejecuta una sentencia si una condición especificada es evaluada como **verdadera**. Si la condición es evaluada como **falsa**, otra sentencia puede ser ejecutada.

La sintaxis básica es:



Los operadores de las condicionales son:

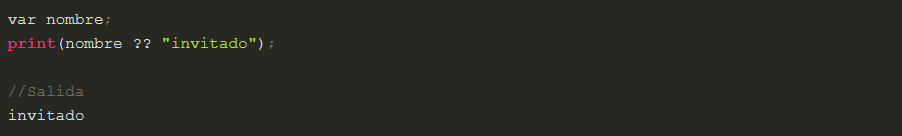
* **==** ¿El primero es Igual que el segundo?
* **!=** ¿El primero es Diferente que el segundo?
* **>** ¿El primero es Mayor que el segundo?
* **<** ¿El primero es Menor que el segundo?
* **>=** ¿El primero es Mayor o igual que el segundo?
* **<=** ¿El primero es Menor o igual que el segundo?
* **!expr** Niega la expresión
* **||** ¿El primero O el segundo?
* **&&** ¿El primero Y el segundo?
* **??** Si la primera es null evalúa la segunda

Puede utilizarse también un if-else anidados, con varias condiciones, de esta manera:



Expresiones condicionales:

Si queremos evaluar una expresión y si esta es null utilizar otra, usaremos el operador **??** de la siguiente manera:



Podemos transformar un if-else a una línea de código, de esta manera:



Lo que está antes de "**?**"es la condición, lo que está después es lo que pasa si se cumple, y lo que está después de "**:**" es lo que pasa si no se cumple.

# **Switch y Case**

**Switch**/**Case** es una **estructura de control** utilizada para la toma de decisiones múltiples en las que se **evalúa un valor** y realiza un **set de acciones** dependiendo de **cuál es el valor**.

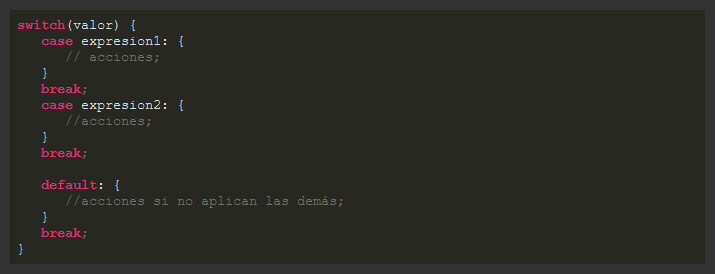
**Reglas**

-Se puede utilizar **int**, **String** y **Enum** como tipo para evaluar la condición.

-La cláusula case debe terminar con **break**, **continue**, **throw** o **return**.

-Si ningún **case** coincide con el valor evaluado, se utiliza la cláusula **default**.

Ejemplo:

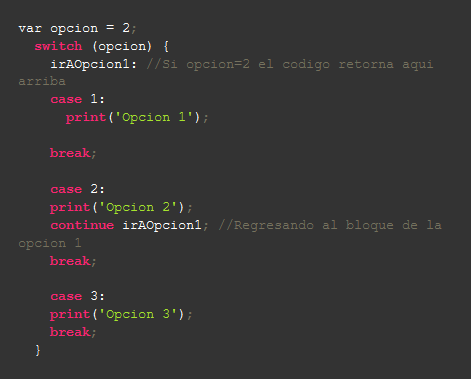


Información adicional

Con la cláusula **continue** se puede mandar al programa a una etiqueta definida. Por ejemplo:



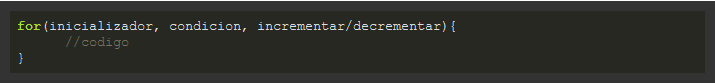
No sólo nos permite saltar a código que esta **debajo**, sino que también nos permite saltar a código que está **arriba**.



# **For**

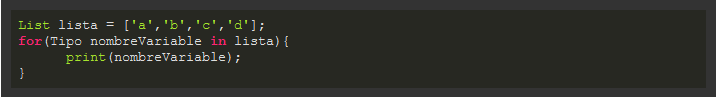
El **for** es una **estructura de control** en la que se indica el número de iteraciones.

La estructura de un ciclo **for** es:



El ciclo **for in** recorre el contenido de una lista.

La estructura de un ciclo **for in** es:



**Ciclo forEach**: Es un **método interno** del objeto **lista**, y la recorre al igual que el ciclo **for**…**in**.

Sintaxis



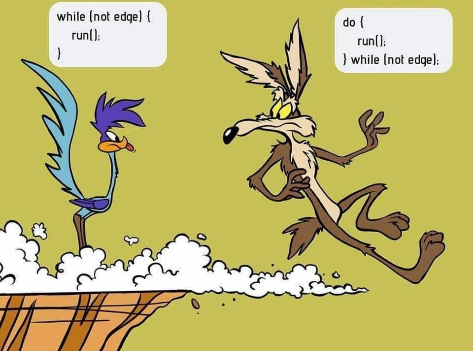
# **While y do-while**

La estructura de control while evalúa una sentencia al inicio y luego ejecuta el código y por otro lado do/while primero se ejecuta y luego evalúa si continuar.





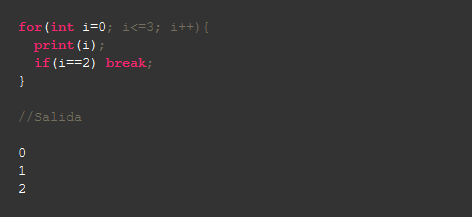
Una manera más grafica de entender la diferencia entre los dos es.



# **Break y Continue**

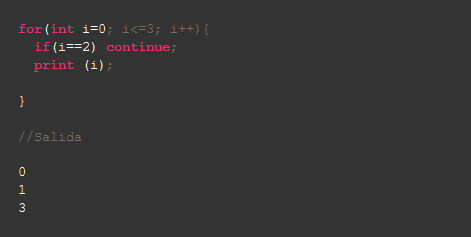
**Break**

Se utiliza **break** para **detener** el trabajo del **ciclo actual**.



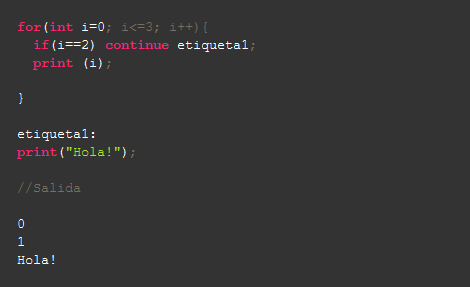
**Continue**

Con este se **detiene el trabajo**, pero únicamente del **bloque** de código que **continúe inmediatamente** después, pero el **ciclo continuará** trabajando.



**Etiquetas**

Lo que hacen estas es **continuar** la **ejecución** de tu código desde un **punto definido** utilizado por una **etiqueta**.

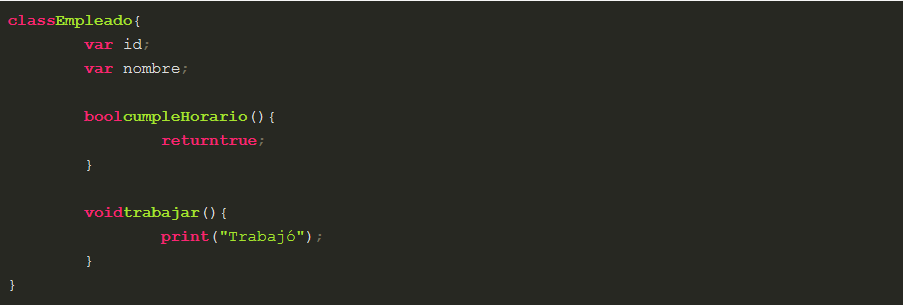


# **Definición de clases y miembros**

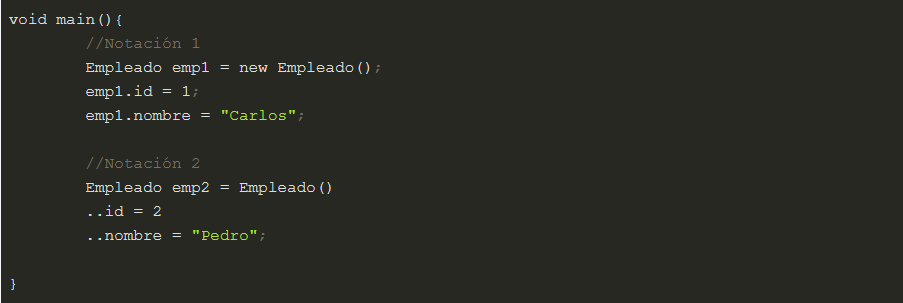
Una clase **es el plano de un objeto**, es la descripción del objeto, pero no el objeto, más bien es una plantilla para crear objetos.

**Todo** puede ser descrito como un **objeto**, con ciertas variables de instancia y métodos, por ejemplo, casas, carros, personas, animales, etc.

Se le llama **miembros de una clase** a todas **las variables de instancia** y **métodos** que se existen dentro de ella y podemos acceder a ellos **llamándolos** desde un objeto.



Instanciación



**Nota**: A partir de la versión 2.1 de Dart la palabra reservada **new** ya no es necesaria para instanciar una clase.

# **Constructores por defecto, por parámetro o nombrados**

Los constructores se utilizan para inicializar una clase, son el primer método que se visualiza al instanciar un objeto.

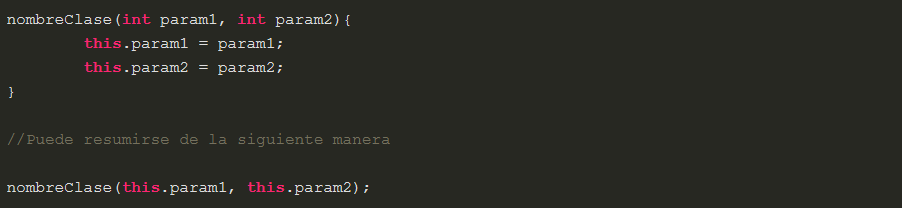
## **Constructores por defecto**

* El constructor por defecto ya existe cuando se crea una clase y se define creando un método con el mismo nombre de la clase.



## **Constructores con parámetros**

* Los constructores con parámetros son aquellos que pueden definir los miembros mediante el constructor.
* No puede existir en la misma clase un constructor por defecto y uno con parámetros.



## **Constructores nombrados**

* También podemos definir constructores con un nombre definido por nosotros y esto hace que puedan existir múltiples constructores.



**Nota**: La palabra reservada **this** hace referencia al **objeto actual**. Por lo tanto, cuando se hace, por ejemplo, la sentencia **this.id = id**, debe diferenciarse el primer “**id**”, ya que este es un **atributo del objeto**, y el segundo un **parámetro del constructor**. Y es mediante ese parámetro del constructor que inicializamos el atributo del objeto, **no son las mismas variables** (aunque se llamen igual).

# **Métodos Getter y Setter y variables privadas**

Los constructores son métodos especiales que proveen acceso de lectura y escritura a las propiedades de un objeto.

Existen dos tipos

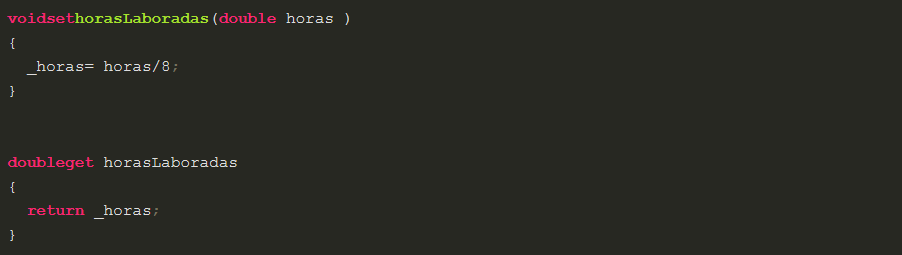
* Por defecto.
* Personalizados.

Las **variables privadas** que se definen empiezan con un **\_**



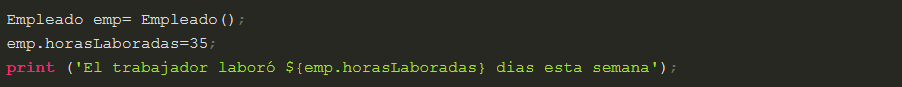
Al querer intentar acceder a esta variable no se podrá a excepción que la clase este en el mismo archivo

Establecemos los **sets** y **getters** así:



Nota: Los métodos getters no llevan paréntesis.

Y luego para su instancia es algo como:



También podemos optimizar el código para que quede así y su funcionamiento será el mismo.

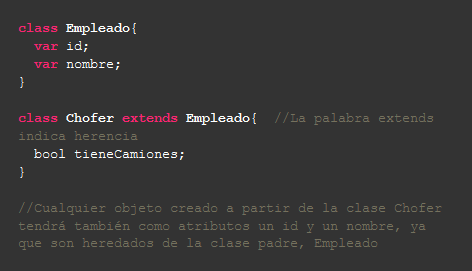


# **Herencia**

La herencia nos permite **adquirir** las **propiedades** de una clase ‘**Padre’**.

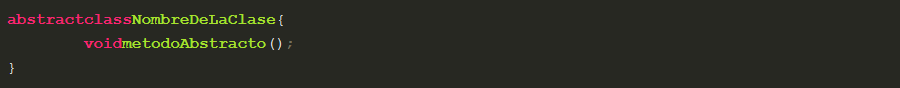
La clase hijo entonces **tiene las propiedades** de la clase ‘**Padre’** y además tiene sus **propiedades propias**, mismas que heredará a sus hijos.

Por ejemplo, todas las clases heredan por defecto de la clase ‘**Object’** ya que en Dart **todo** es un objeto, por lo tanto, todas las clases de Dart comparten algunos miembros.



# **Clases Abstractas**

* Una clase abstracta **no puede ser instanciada**, es decir, **no se puede crear objetos**, aunque puede ser extendida.
* Un método abstracto es **estructurado**, pero no definido, deberá ser sobrescrito (override) en el futuro.



Las clases que extiendan de una clase abstracta deberán sobrescribir todos sus métodos. Esto se hace colocando la etiqueta.

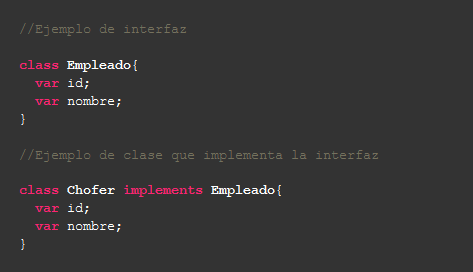


# **Interfaces implícitas**

Una **clase normal** puede funcionar como una **interfaz,** a esto de le llama **interfaces implícitas**, éstas se harán **implementando múltiples clases** utilizando las clases como una interfaz.

Para que la interfaz implícita sea **descripta correctamente**, la clase deberá de definir **todos los miembros** (atributos y métodos) de las clases de las que va a implementar.

En Dart a esto se le conoce como **mixings**.



# **Contenedores**

Los contenedores o containers son la estructura visual básica de una aplicación en Flutter, nos permiten agrupar u organizar internamente otros widgets, además de que tienen propiedades visuales

**Propiedades**

* **height**: alto (double.infinity ocupa todo el espacio posible)
* **width**: ancho
* **color**: color del container
* **padding**: bordes internos del container
* **margin**: bordes externos del container
* **child**: hijo
* **alignment**: alineación
* **decoration**: decoración
  + **.color**: color de decoración
  + **.border**: borde del container
* **transform**: rota el container.
* **constraints**: restricciones del container.

**debugShowCheckedModeBanner**: Propiedad que nos permite activar o desactivar el banner de “DEBUG”