Primeros pasos

DART 101

Objetivos

- Hola Mundo
- Comentarios
- Variables
- Tipos de datos – Primitivos
- Tipos de datos – Compuestos
- Null Safety

Glosario

- Consola o Terminal: interfaz en donde vamos a colocar comandos. Ej: “flutter doctor”
- Script: conjunto de instrucciones de código dentro del entorno de desarrollo.
- Entorno de desarrollo: se refiere al software que se usa para codificar. Ej: Visual Studio Code o VSCode. También se lo puede llamar como “Editor de código”
- Extensión: nombre de la extensión de nuestros scripts. “.dart”
- Path: ruta o dirección en donde se ubica un archivo.
- snake_case: nomenclatura para declarar el nombre de archivos y carpetas dentro de proyectos de Dart. // hola_mundo -> richar_cangui.dart -> luna.dar
- camelCase: nomenclatura para declarar el nombre de variables. // variables -> lunaLlena -> luna
- PascalCase: nomenclatura para declarar el nombre de clases. // Clases -> PerroJugueton -> Luna
- Wizard o setup de instalación de un programa
- Framework -> Flutter -> conjunto de herramientas que permiten agilizar el desarrollo.

Consideraciones iniciales

- Todo es un objeto “Object”
- Fuertemente tipado
- Gestión automática de memoria

Hola Mundo

```dart
void main() {
  print('Hello, World!');
}
```

- Void -> vacío
- Main -> principal
- Print -> “Función - método” -> Mostrar algo en la consola
- “Hello, World!” -> Mensaje -> Texto -> comillas simples ‘’ , comillas dobles “”.

Comentarios en Dart

Dart como cualquier otro lenguaje de programación permiten agregar comentarios dentro del código, un comentario ayuda al desarrollador a entender de mejor manera el programa ya sea suyo o de algún otro desarrollador.

- • Comentario simple //
- • Comentario de bloque /* */
- • Comentario de documentación ///

Palabras reservadas

abstract	continue	FALSE	new	this
as	default	final	null	throw
assert	deferred	finally	operator	TRUE
async	do	for	part	try
async*	dynamic	get	rethrow	typedef
await	else	if	return	var
break	enum	implements	set	void
case	export	import	static	while
catch	external	in	super	with
class	extends	is	switch	yield
const	factory	library	sync*	yield*

- Desde dart 2.12 se agregaron:
- required
- late

Tipos de datos primitivos

String

- Define cadenas de texto o caracteres.
- Se crean con comillas dobles o simples.
- No existe límite de almacenamiento.

num

- Clase abstracta principal para representar números.
- Subclases int y double.
- int: representa números enteros positivos o negativos.
- double: representa números decimales positivos o negativos.

bool

- Define valores boleanos (true o false).
- Utilizados para toma de decisiones o control de flujos.

Variables

- Se utiliza para almacenar datos, que posteriormente se pueden utilizar.
- Deben ser declarados e inicializados.
  - late (declarado pero inicializado más tarde).
  - var
  - Tipo de dato
- Pueden ser constantes o inmutables.
  - const
  - final
- Pueden ser locales (dentro de funciones)
- Pueden ser estáticas (definidas en clases)
- Pueden ser globales (nivel superior)
- Pueden ser variables.

Buenas prácticas al declarar variables

- Usar nomenclatura camelCase y si es posible en inglés. (timezone, lastHour, opimiceQuantityTime, precioTotal)
- No hagas que el editor infiera el tipo de dato, siempre especifica uno cuando se trate de variables.
- Usa const la mayor cantidad de veces. (más optimo)
- Siempre que se pueda utilizar final en lugar de valores variables para conservar el principio de inmutabilidad.
- **No utilices num, usa double o int.**
- **Usar nombres descriptivos al uso de la variable, si vas a utilizar una variable para guardar un nombre llamala “nombre” no la llames “n”.**
- **Llama las variables en inglés (modulo 1 voy a usar español).**

Tipos de datos compuestos

List

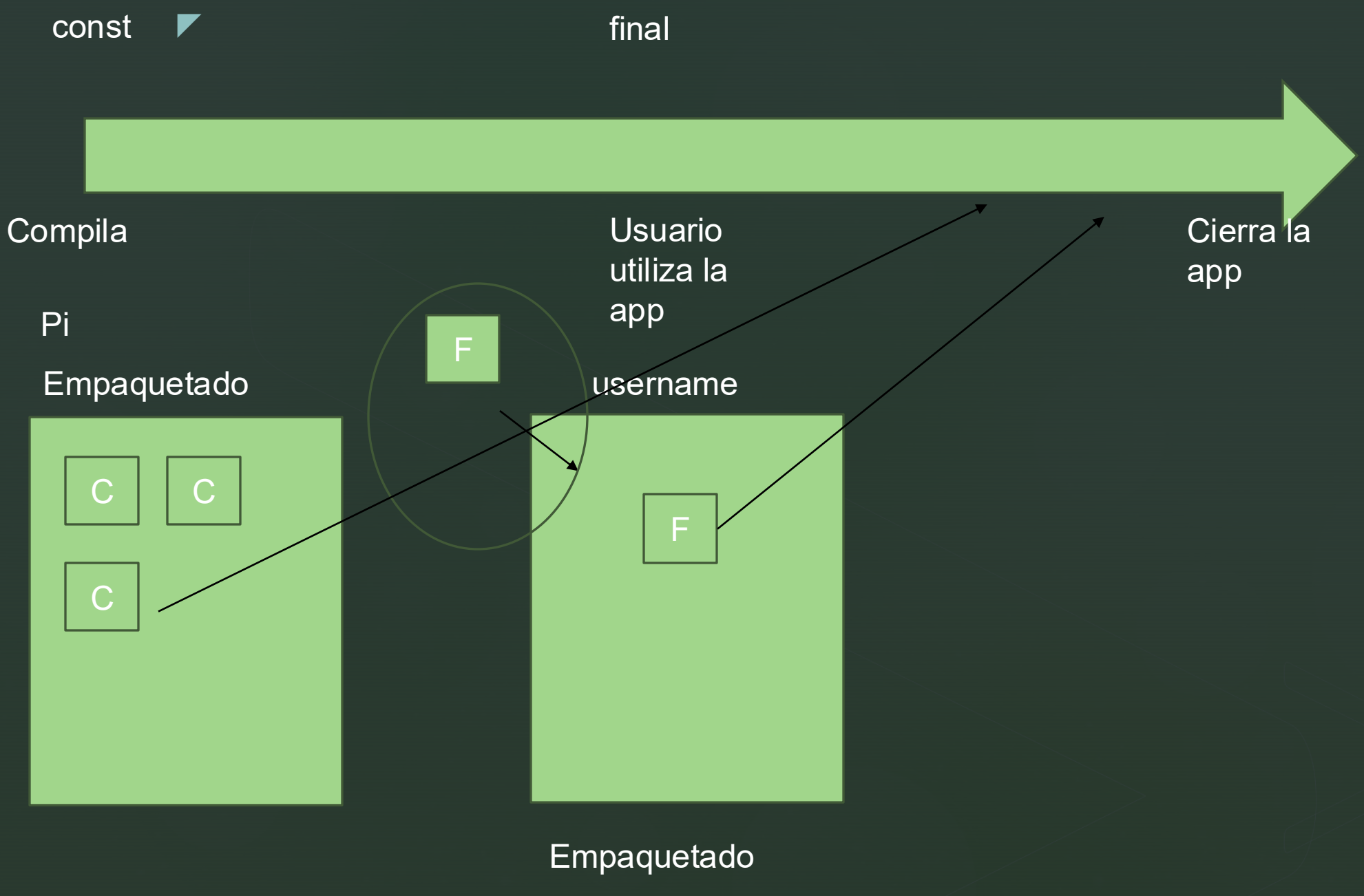
- Representa una lista de elementos.
- La lista puede ser fija o extensible.
- Nomenclatura: palabra reservada <Tipo de dato> (List<int>)

Map

- Representa una colección, par de valores “clave”:”valor”
- Nomenclatura: palabra reservada<tipo de dato, tipo de dato> (Map<String, int>)

Null Safety o Seguridad Nula

- Dart incorporó el null safety desde la versión 2.12 como opcional y actualmente para la versión 3 se usa obligtoriamente.
- El null safety previene errores debido a parametros no declarados o nulos, esto es de gran ayuda en las aplicaciones de flutter ya que asegura un valor y se previenen errores de valores nulos.


const
final
Compila
Usuario utiliza la app
Cierra la app
Pi
Empaquetado
F
username
C
C
C
F
Empaquetado