📚 Guía Completa de Flutter - PARTE 3: EJERCICIOS RESUELTOS

Ejercicios Explicados Línea por Línea

✅ EJERCICIO 1: Carrito de Cafetería con Provider

🎯 Objetivo del Ejercicio

Crear una app de cafetería donde:

- Puedas añadir productos al carrito
- El total y el número de items se actualicen automáticamente
- Puedas vaciar el carrito
- **TODO funcione con Provider** (gestión de estado global)

📚 Conceptos que Aprenderás

- Cómo usar Provider para compartir datos entre widgets
- Cuándo usar `context.watch` vs `context.read`
- Por qué y cuándo llamar a `notifyListeners()`

PASO 1: Crear el Modelo (El Cerebro)

```dart
import 'package:flutter/material.dart';

// Esta clase extiende de ChangeNotifier
// ChangeNotifier es como un "notificador" que avisa cuando algo cambia
class CarritoModel extends ChangeNotifier {
```

```dart
  // Variables PRIVADAS (el guion bajo _ las hace privadas)
  // Nadie puede modificarlas directamente desde fuera
  int _items = 0;
  double _total = 0.0;

  // Getters PÚBLICOS (solo lectura)
  // Permiten leer los valores pero no modificarlos directamente
  int get items => _items;
  double get total => _total;

  // Método para AÑADIR un producto
  void add(double precio) {
    _items++;          // Incrementamos el contador
    _total += precio;  // Sumamos el precio al total

    // 🔔 CRÍTICO: notifyListeners() le dice a Flutter:
    // "He cambiado algo, redibuja todos los widgets que me están escuchando"
    notifyListeners();
  }

  // Método para VACIAR el carrito
  void clear() {
    _items = 0;
    _total = 0.0;
    notifyListeners(); // 🔔 Avisamos del cambio
  }
}
```

¿Por qué variables privadas?
Para que nadie pueda hacer `carrito._items = 999` desde fuera. Solo se pueden modificar a través de los métodos `add()` y `clear()`.

¿Qué pasa si olvido `notifyListeners()`?
Los datos cambiarían internamente, pero la pantalla NO se actualizaría. Verías siempre “0 i tems” aunque hayas añadido productos.

PASO 2: Inyectar el Provider en main.dart

```dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

void main() {
  runApp(
    // ChangeNotifierProvider hace que CarritoModel esté disponible
    // para TODA la app (todas las pantallas)
    ChangeNotifierProvider(
      // create: función que crea el modelo
      // El guion bajo _ significa "no uso este parámetro"
      create: (_) => CarritoModel(),

      // child: la app principal
      child: const CafeteriaApp(),
    ),
  );
}

class CafeteriaApp extends StatelessWidget {
  const CafeteriaApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      debugShowCheckedModeBanner: false, // Quita el banner de "DEBUG"
      home: CafeteriaPage(),
    );
  }
}
```

¿Qué hace `ChangeNotifierProvider`?

Es como poner el modelo en un “lugar central” donde todas las pantallas pueden acceder a

él. Sin esto, tendrías que pasar el carrito de pantalla en pantalla manualmente.

PASO 3: La Pantalla Principal

```dart
class CafeteriaPage extends StatelessWidget {
  const CafeteriaPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Cafetería Provider'),
        centerTitle: true,
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          children: [
            // Widget separado que muestra el resumen
            const _ResumenCarrito(),

            const SizedBox(height: 16),

            // Lista de productos
            const _Producto(nombre: 'Café', precio: 1.20),
            const _Producto(nombre: 'Tostada', precio: 1.80),
            const _Producto(nombre: 'Zumo', precio: 2.10),

            // Spacer empuja todo lo demás hacia arriba
            const Spacer(),

            // Botón para vaciar el carrito
            SizedBox(
```

```
            width: double.infinity, // Ocupa todo el ancho
            child: ElevatedButton(
              // context.read: EJECUTAR una acción
              // No necesitamos redibujar este botón, solo llamar a clear()
              onPressed: () => context.read<CarritoModel>().clear(),
              child: const Text('Vaciar carrito'),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

¿Por qué `const`?
`const` le dice a Flutter: “Este widget NUNCA cambiará”. Flutter lo optimiza y no lo reconstruye innecesariamente.

¿Por qué widgets separados (_ResumenCarrito, _Producto)?
Para que el código sea más limpio y organizado. Cada widget tiene una responsabilidad clara.

PASO 4: El Widget del Resumen

```
class _ResumenCarrito extends StatelessWidget {
 const _ResumenCarrito();

 @override
 Widget build(BuildContext context) {
  // context.watch: ESCUCHAR cambios
  // Este widget se redibujará cada vez que el carrito cambie
  final carrito = context.watch<CarritoModel>();
```

```
    return Container(
      padding: const EdgeInsets.all(12),
      decoration: BoxDecoration(
        border: Border.all(color: Colors.black12),
        borderRadius: BorderRadius.circular(10),
      ),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          // Mostramos el número de items
          Text(
            'Items: ${carrito.items}',
            style: const TextStyle(fontSize: 18),
          ),

          // Mostramos el total con 2 decimales
          Text(
            'Total: ${carrito.total.toStringAsFixed(2)} €',
            style: const TextStyle(fontSize: 18),
          ),
        ],
      ),
    );
  }
}
```

¿Qué hace `context.watch<CarritoModel>()`?

1. Busca el `CarritoModel` más cercano en el árbol de widgets
2. Se “suscribe” a él (lo escucha)
3. Cada vez que llames a `notifyListeners()` en el modelo, este widget se redibuja

¿Qué es `toStringAsFixed(2)`?
Convierte el número a texto con exactamente 2 decimales. Ejemplo: `3.5` → `"3.50"`

PASO 5: El Widget de Producto

```dart
class _Producto extends StatelessWidget {
  // Propiedades finales que recibe el widget
  final String nombre;
  final double precio;

  // Constructor con parámetros obligatorios
  const _Producto({
    required this.nombre,
    required this.precio,
  });

  @override
  Widget build(BuildContext context) {
    return Card(
      child: ListTile(
        // Título del producto
        title: Text(nombre),

        // Subtítulo con el precio formateado
        subtitle: Text('${precio.toStringAsFixed(2)} €'),

        // Botón a la derecha
        trailing: ElevatedButton(
          onPressed: () {
            // context.read: EJECUTAR una acción
            // No necesitamos redibujar este producto, solo añadir al carrito
            context.read<CarritoModel>().add(precio);
          },
          child: const Text('Añadir'),
        ),
      ),
    );
```

```
  }
}
```

¿Por qué `context.read` y no `context.watch`?
Porque NO necesitamos que este widget se redibuje cuando el carrito cambie. Solo queremos ejecutar la acción `add()`. Si usáramos `watch`, el producto se redibujaría cada vez que añadas algo al carrito (innecesario y menos eficiente).

🎓 Resumen del Ejercicio 1

Flujo completo:

1. Usuario pulsa "Añadir" en un producto
2. Se llama a `context.read<CarritoModel>().add(precio)`
3. El método `add()` modifica `_items` y `_total`
4. Se llama a `notifyListeners()`
5. Flutter redibuja SOLO los widgets que usan `context.watch<CarritoModel>()`
6. El resumen se actualiza automáticamente

Errores comunes:

- ❌ Olvidar `notifyListeners()` → La pantalla no se actualiza
- ❌ Usar `watch` en un botón → El botón se redibuja innecesariamente
- ❌ Usar `read` para mostrar datos → Los datos no se actualizan
- ❌ No envolver la app con `ChangeNotifierProvider` → Error: "Could not find Carrito Model"

✅ EJERCICIO 2: Navegación Básica

🎯 Objetivo del Ejercicio

Aprender a navegar entre pantallas y volver atrás.

📚 Conceptos que Aprenderás

- `Navigator.push`: Ir a una nueva pantalla

- `Navigator.pop`: Volver atrás
- Pasar datos entre pantallas

Código Completo Explicado

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(const MaterialApp(home: InicioPage()));
}

// ========== PANTALLA 1: INICIO ==========
class InicioPage extends StatelessWidget {
  const InicioPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Inicio')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            // Botón para ir a la pantalla de Info
            ElevatedButton(
              onPressed: () {
                // Navigator.push: Ir a una nueva pantalla
                Navigator.push(
                  context, // Desde dónde navegamos
                  MaterialPageRoute(
                    // builder: función que devuelve la nueva pantalla
                    builder: (_) => const InfoPage(),
                  ),
```

```
              );
            },
            child: const Text('Ir a Info'),
          ),

          const SizedBox(height: 20),

          // Botón para ir a la pantalla de Contacto
          ElevatedButton(
            onPressed: () {
              Navigator.push(
                context,
                MaterialPageRoute(builder: (_) => const ContactoPage()),
              );
            },
            child: const Text('Ir a Contacto'),
          ),
        ],
      ),
    ),
  );
 }
}

// ========== PANTALLA 2: INFO ==========
class InfoPage extends StatelessWidget {
 const InfoPage({super.key});

 @override
 Widget build(BuildContext context) {
  return Scaffold(
   // Flutter añade automáticamente un botón de "atrás" en el AppBar
   appBar: AppBar(title: const Text('Info')),
   body: Center(
```

```
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Text('Esta es la pantalla de información'),
            const SizedBox(height: 20),

            // Botón manual para volver
            ElevatedButton(
              onPressed: () {
                // Navigator.pop: Volver a la pantalla anterior
                Navigator.pop(context);
              },
              child: const Text('Volver'),
            ),
          ],
        ),
      ),
    );
  }
}

// ========== PANTALLA 3: CONTACTO ==========
class ContactoPage extends StatelessWidget {
  const ContactoPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Contacto')),
      body: Center(
        child: ElevatedButton(
          onPressed: () => Navigator.pop(context),
          child: const Text('Volver'),
        ),
```

```
      ),
    );
  }
}
```

🔍 Explicación Detallada

¿Qué es `MaterialPageRoute`?
Es la “ruta” que Flutter usa para ir a una pantalla con una animación bonita (deslizamiento de derecha a izquierda en Android, de abajo hacia arriba en iOS).

¿Qué es `builder`?
Es una función que devuelve el widget de la nueva pantalla. Se ejecuta cuando Flutter nec esita construir la pantalla.

¿Por qué el guion bajo `_` en `builder: (_)`?
Significa “no voy a usar este parámetro”. Es una convención en Dart.

¿Qué pasa si hago `pop` en la primera pantalla?
La app se cierra (porque no hay nada “debajo” en la pila).

Pasar Datos a la Siguiente Pantalla

```
// Pantalla de destino que RECIBE datos
class DetallePage extends StatelessWidget {
  final String producto; // Dato que recibimos
  final double precio;

  // Constructor: obligamos a recibir estos datos
  const DetallePage({
    super.key,
    required this.producto,
    required this.precio,
  });
```

```dart
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text(producto)), // Usamos el dato
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(
              producto,
              style: const TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
            ),
            const SizedBox(height: 10),
            Text(
              'Precio: ${precio.toStringAsFixed(2)} €',
              style: const TextStyle(fontSize: 20),
            ),
          ],
        ),
      ),
    );
  }
}

// Al navegar, pasamos los datos:
ElevatedButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (_) => const DetallePage(
          producto: 'Café Latte',
          precio: 3.50,
        ),
```

```
      ),
    );
  },
  child: const Text('Ver Detalle'),
)
```

✅ EJERCICIO 3: Animaciones Implícitas (Bombilla)

🎯 Objetivo del Ejercicio

Crear una bombilla que se encienda y apague con animación suave.

📚 Conceptos que Aprenderás

- Widgets `Animated...` (AnimatedOpacity, AnimatedScale)
- Operador ternario `? :`
- Cómo Flutter anima automáticamente

Código Completo Explicado

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(const MaterialApp(home: BombillaPage()));
}

// StatefulWidget porque el estado (encendida/apagada) cambia
class BombillaPage extends StatefulWidget {
  const BombillaPage({super.key});

  @override
```

```dart
  State<BombillaPage> createState() => _BombillaPageState();
}

class _BombillaPageState extends State<BombillaPage> {
  // Variable de estado: ¿está encendida?
  bool encendida = false;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Bombilla Animada')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            // AnimatedOpacity: Anima la transparencia
            AnimatedOpacity(
              // Duración de la animación
              duration: const Duration(milliseconds: 500),

              // Operador ternario: condicion ? siVerdadero : siFalso
              // Si encendida es true, opacidad 1.0 (visible)
              // Si encendida es false, opacidad 0.2 (casi invisible)
              opacity: encendida ? 1.0 : 0.2,

              // AnimatedScale: Anima el tamaño (anidado dentro)
              child: AnimatedScale(
                duration: const Duration(milliseconds: 500),

                // Si encendida, escala 2.0 (doble de tamaño)
                // Si apagada, escala 1.0 (tamaño normal)
                scale: encendida ? 2.0 : 1.0,

                // El emoji de la bombilla
```

```
              child: const Text(
                '💡',
                style: TextStyle(fontSize: 50),
              ),
            ),
          ),

          const SizedBox(height: 60),

          // Texto que cambia según el estado
          Text(
            encendida ? "ENCENDIDA" : "APAGADA",
            style: const TextStyle(
              fontWeight: FontWeight.bold,
              fontSize: 20,
            ),
          ),
        ],
      ),
    ),

    // Botón flotante para cambiar el estado
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        // setState: Cambiamos el estado y redibujamos
        setState(() {
          // Operador !: invierte el valor (true → false, false → true)
          encendida = !encendida;
        });
      },
      child: const Icon(Icons.power_settings_new),
    ),
  );
}
```

```
}
```

🔍 Explicación Detallada

¿Qué es el operador ternario `? :`?
Es una forma corta de escribir un `if-else`:

```dart
// Forma larga:
double opacidad;
if (encendida) {
  opacidad = 1.0;
} else {
  opacidad = 0.2;
}

// Forma corta (ternario):
double opacidad = encendida ? 1.0 : 0.2;
```

¿Cómo funciona la animación?

1. Cambias el estado con `setState(() => encendida = !encendida)`
2. Flutter redibuja el widget
3. `AnimatedOpacity` ve que `opacity` cambió de 0.2 a 1.0
4. En lugar de cambiar instantáneamente, Flutter anima el cambio durante 500ms

¿Por qué anidar AnimatedOpacity y AnimatedScale?
Para combinar dos animaciones: la bombilla se hace más grande Y más visible al mismo ti empo.

Continuaré con los ejercicios 4 y 5 en la siguiente parte…