GRADO EN INGENIERÍA INFORMÁTICA

DESARROLLO DE SOFTWARE

Un saludito para Juanmi

UNIVERSIDAD
DE GRANADA

P2 : Adaptación de software de python a dart (flutter)

Juan Miguel Acosta Ortega (*acostaojuanmi@correo.ugr.es*)

Jesús Pereira Sánchez (*jesuspereira@correo.ugr.es*)

David Serrano Domínguez (*davidserrano07@correo.ugr.es*)

Raúl Florentino Serra (*raulfloren@correo.ugr.es*)

21 de abril de 2024

Índice

Objetivos

La práctica consiste en realizar una adaptación del ejercicio 3 de la práctica primera, en nuestro caso, combinación del patrón de diseño conductual Observer con el patrón creacional Builder para la creación de hamburguesas. Se ha de trasladar en primera instancia el código de Python a Dart con los cambios convenientes, y tras esto realizar una interfaz gráfica para poder hacer uso del sistema sin la terminal.

ENLACE AL REPOSITORIO DE GITHUB : Repositorio de las prácticas

1. Transición de lenguaje

En esta segunda práctica se ha de cumplir el objetivo de adaptar el software del ejercicio tres de la primera práctica a un nuevo lenguaje de programación (dart) y framework (flutter) usados para aplicaciones multiplataforma.

El primer paso será el de adaptar todo el código python que se corresponde casi enteramente con el "modelo"a dart, para más adelante crear una interfaz gráfica (vista) que trabaje con este.

En este paso han habido pocos cambios con respecto al código original, sin embargo se han decidido aplicar ciertas medidas que cambian un poco el diseño primario.

En primer lugar se ha decidido cambiar la interfaz "HamburguesaBuilder" a clase abstracta, ya que hemos implementado en ella ciertos métodos de simulación de tiempo que tiene sentido que hereden los demás builder concretos.

```
1
2     import 'dart:async';
3     import 'Hamburguesa.dart';
4
5     abstract class HamburguesaBuilder {
6
7       late Hamburguesa hamburguesa;
8
9       void aniadePan();
10
11      void aniadeLechuga();
12
13      void aniadeTomate();
14
15      void aniadeCebolla();
16
17      void aniadePepinillos();
18
19      void aniadeBacon();
20
21      void aniadeCarne();
22
23      void aniadePrecio();
24
25      void sleep() {
26
27        Future.delayed(const Duration(milliseconds: 100));
28      }
29
30      void sleepLong() {
31
32        Future.delayed(const Duration(milliseconds: 800));
33      }
34    }
```

Listing 1: Esperas en las construcción de las hamburguesas

Además hemos decidido prescindir de la parte del código que generaba un archivo de texto con la comanda ya que más adelante esta se imprimirá por pantalla simulando la típica pantalla de cadena de comida rápida.

Por lo demás, el código es bastante similar al desarrollado inicialmente, aplicando, claro está, los conceptos adquiridos con el desarrollo de la parte individual.

2. Interfaz gráfica

Continuando con el desarrollo del código, iremos explicando los distintos widgets utilizados a lo largo de la realización de la práctica, junto con una breve descripción de su funcionamiento y para qué fue elegido. Antes que nada, para diferenciarlos, tenemos que saber que en flutter existen dos tipos de widgets:

- ***StatelessWidget***: estos son aquellos que no mantienen ningún estado interno. Esto significa que su apariencia es determinada por los parámetros y no cambia durante la ejecución de la aplicación.
- ***StatefulWidget***: tienen un estado mutable, lo que significa que su apariencia puede cambiar en función de su estado interno

Una vez vistos los tipos de widgets vamos a ver cuáles han sido utilizados en esta práctica, todos ellos son de tipo Stateless, salvo Scaffold, que proporciona la estructura básica de la página:

- **Scaffold:** Se utiliza como el marco visual principal de la página. Consiste en un contenedor con una imagen de fondo y una columna central de botones de hamburguesas.
- **AppBar:** Es una barra de aplicación que normalmente se encuentra en la parte superior de la pantalla. Se utiliza para mostrar un título "Bienvenido" y acciones, como un icono de carrito de compras y la cantidad de hamburguesas en el pedido.
- **IconButton:** Es un botón que muestra un icono en lugar de texto. Se utiliza para acciones específicas, como abrir el carrito de compras en este caso.
- **Container:** Se utiliza como un contenedor principal para el cuerpo de la página. Contiene una imagen de fondo y una columna de botones de hamburguesas.
- **Column y Row:** widgets que organizan a sus hijos de forma vertical u horizontal. Utilizado para colocar los botones de hamburguesas.
- **Padding:** Se utiliza alrededor de cada botón de hamburguesa para agregar un margen entre ellos y los bordes del contenedor.
- **Stack:** Se utiliza para apilar la imagen de la hamburguesa y el botón de la hamburguesa uno encima del otro. Esto permite que el botón de la hamburguesa aparezca sobre la imagen de la hamburguesa.
- **ElevatedButton:** Se utiliza como botón para cada tipo de hamburguesa. Al hacer clic en estos botones, se agrega la hamburguesa correspondiente al carrito de compras.
- **AlertDialog:** Se utiliza para mostrar los detalles del pedido cuando se hace clic en el icono de carrito de compras en la AppBar. Muestra una lista de hamburguesas en el carrito y el precio total del pedido.
- **SingleChildScrollView:** Se utiliza dentro del AlertDialog para permitir desplazarse por la lista de hamburguesas en caso de que la lista sea más larga que la altura del diálogo.
- **Text:** Se utiliza para mostrar el nombre de cada hamburguesa y su precio dentro del AlertDialog.
- **TextButton:** Se utiliza como el botón "Finalizar Pedido"dentro del AlertDialog. Al hacer clic en este botón, se finaliza el pedido y se cierra el diálogo.
- **MaterialPageRoute:** Este widget se utiliza para definir la ruta a una nueva pantalla cuando se navega. Aquí se utiliza para navegar al menú cuando se hace clic en los botones "Para llevar" y "Para tomar aquí".
- **DecorationImage:** Es una clase utilizada dentro del BoxDecoration del Container para mostrar una imagen como fondo. Aquí se usa para mostrar la imagen de fondo de la aplicación.

- **Center:** Este widget se utiliza para centrar su único hijo en la pantalla. En este caso, centra una columna de botones.
- **Drawer:** Proporciona un panel deslizable que generalmente contiene enlaces de navegación o acciones. En nuestro caso contiene el display de pedidos

Todos estos widgets han sido utilizados a lo largo de la práctica para diversos propósitos. Al igual que los widgets en Flutter, podemos utilizar alguna semántica para facilitar la inicialización e instanciación de distintas variables. Por ejemplo:

1. "String? pan; " para permitir que sean null ciertos parámetros.
2. "late String idPedido; " para indicar que más adelante se instanciará el valor.
3. "String nombre = nombreUsuario ?? 'Usuario desconocido'; " para inicializar con un valor predeterminado

Vistos los widgets y un poco la semantica vamos a analizar el nuevo los nuevos archivos que hemos añadido al proyecto anterior para adaptarlo al framework de flutter, como primer archivo tenemos ***main.dart***:

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

void main() {
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'PR2 DS',
      theme: ThemeData(
        primarySwatch: Colors.red,
      ),
      home: Inicio(),
    );
  }
}

class Inicio extends StatelessWidget  {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Bienvenido'),
      ),
      body: Container(
        decoration: BoxDecoration(
          image: DecorationImage(
            image: AssetImage('assets/fondo.jpg'),
            fit: BoxFit.cover,
          ),
        ),
        child: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Padding(
                padding: EdgeInsets.symmetric(vertical: 25.0, horizontal: 32.0),
                child: ElevatedButton(
```

```
                    style: ButtonStyle(
                      minimumSize:
                          MaterialStateProperty.all(Size(double.infinity, 80)),
                    ),
                    child: Text("Para llevar"),
                    onPressed: () {
                      Navigator.push(
                        context,
                        MaterialPageRoute(builder: (context) => Menu()),
                      );
                    },
                  ),
                ),
                SizedBox(height: 20),
                Padding(
                  padding: EdgeInsets.symmetric(vertical: 25.0, horizontal: 32.0),
                  child: ElevatedButton(
                    style: ButtonStyle(
                      minimumSize:
                          MaterialStateProperty.all(Size(double.infinity, 80)),
                    ),
                    child: Text("Para tomar aqui"),
                    onPressed: () {
                      Navigator.push(
                        context,
                        MaterialPageRoute(builder: (context) => Menu()),
                      );
                    },
                  ),
                ),
                SizedBox(height: 20),
                Padding(
                  padding: EdgeInsets.symmetric(vertical: 25.0, horizontal: 32.0),
                  child: ElevatedButton(
                    style: ButtonStyle(
                      minimumSize:
                          MaterialStateProperty.all(Size(double.infinity, 80)),
                    ),
                    child: Text("Cambiar idioma"),
                    onPressed: () {
                      mostrarSnackBar(
                          context, "En este momento no es posible ...");
                    },
                  ),
                ),
              ],
            ),
          ),
        ),
      );
    }

    void mostrarSnackBar(BuildContext context, String mensaje) {
      final snackBar = SnackBar(
        content: Text(mensaje),
        duration: Duration(seconds: 2),
      );
      ScaffoldMessenger.of(context).showSnackBar(snackBar);
    }
}
```

Este main dart esta dividido en 4 partes

1. Fondo, donde se declara la imagen de fondo
2. Primer botón *Para llevar*

3. Segundo botón *Para tomar aquí*
4. Tercer botón *Cambiar idioma*, que no hace nada

Los dos primeros botones al hacer click sobre ellos nos lleva a la página y segundo archivo *menu.dart*. En este archivo se especifica el funucionamiento de la interfaz para realizar un pedido o más de uno.

```dart
import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';
import 'package:pantalla_pedidos_hamburgueseria/model/Cocinero.dart';
import 'package:pantalla_pedidos_hamburgueseria/model/HamburguesaNormalBuilder.dart';
import 'package:pantalla_pedidos_hamburgueseria/model/HamburguesaSinGlutenBuilder.dart';
import 'package:pantalla_pedidos_hamburgueseria/model/HamburguesaVeganaBuilder.dart';
import 'package:pantalla_pedidos_hamburgueseria/model/ObservadorPedido.dart';
import 'model/Pedido.dart';
import 'model/Hamburguesa.dart';
import 'model/DisplayPedidos.dart';

class Menu extends StatefulWidget {

  @override
  _MenuState createState() => _MenuState();
}

class _MenuState extends State<Menu> {
  List<Pedido> _historialPedidos = [];
  late DisplayPedidos _display;
  int _cantidad = 0;
  Cocinero _cocinero = Cocinero();
  List<String> _pedidoActual = [];

  @override
  void initState() {
    super.initState();
    _display = DisplayPedidos(_historialPedidos, _actualizarHistorial);

    WidgetsBinding.instance?.addPostFrameCallback((_) {
      _display.init(context);
      _cocinero.attach(_display);
    });
  }
```

En esta primera parte conocemos las variables que se van a utilizar:

- **_display**: instancia de la clase DisplayPedidos que se utiliza para mostrar y gestionar la visualización de los pedidos en la interfaz de usuario. Se inicializa más adelante en el código.
- **_cantidad**: Lleva la cantidad de elementos que se han agregado al carrito de compras, para mostrarlo en la AppBar.
- **_cocinero**: Director del builder para llevar a cabo la construcción de las hamburguesas.
- **_pedidoActual**: Una lista para almacenar temporalmente las hamburguesas que el usuario ha agregado al carrito antes de finalizar el pedido.

En *initState()* se inicializa el display con el historial de pedidos y la funcion para actualizar el historial, luego utilizamos WidgetsBindig.instance?, esto nos devuelve el arból de widgets una vez vinculado y addPostFromCallBack() nos permite ejecutar la función después de que se complete el proceso de construcción

Luego en el método build definimos la interfaz del menú:

```
@override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Row(
         children: [
           Text('Bienvenido'),
           SizedBox(width: 8),
           IconButton(
             icon: Icon(Icons.shopping_cart),
             onPressed: () {
               _mostrarPedido(context);
             },
           ),
           SizedBox(width: 8),
           Text('$_cantidad'),
         ],
       ),
     ),
     endDrawer: Drawer(

       child: ListView(
         padding: EdgeInsets.zero,
         children: [
           SizedBox(
             height: 60,
             child: DrawerHeader(
               decoration: BoxDecoration(
                 color: Colors.red,
               ),
               child: Row(
                 mainAxisAlignment: MainAxisAlignment.spaceBetween,

                 children: [
                   Text('Historial de Pedidos'),
                   IconButton(
                     icon: Icon(Icons.close_sharp),
                     onPressed: () {
                       Navigator.of(context).pop();
                     },
                   ),
                 ],
               ),
             ),
           ),

           if (_display.historial.isEmpty)
             ListTile(
               title: Text('No hay pedidos en el historial'),
             )
           else

             ..._display.historial.map((pedido) {
               return ListTile(
                 title: Text(
                   'Pedido: ${pedido.idPedido} listo',
                   style: TextStyle(color: Colors.green),
                 ),
               );
             }).toList(),
         ],
       ),
     ),
     body: Container(
       decoration: BoxDecoration(
         image: DecorationImage(
```

```
            image: AssetImage('assets/fondo.jpg'),
            fit: BoxFit.cover,
          ),
        ),
        child: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              SizedBox(height: 20),
              _botonHamburguesa(
                  'assets/normal.jpg', "Hamburguesa normal", 5.00),
              SizedBox(height: 20),
              _botonHamburguesa(
                  "assets/vegana.jpg", "Hamburguesa vegana", 5.50),
              SizedBox(height: 20),
              _botonHamburguesa(
                  "assets/singluten.jpg", "Hamburguesa sin gluten", 6.00),
            ],
          ),
        ),
      ),
    );
  }
```

Este método lo podemos dividir en 3 partes:

1. **AppBar:** Título de "Bienvenidoçon un icono de carrito y el número de elementos en el carrito. Al hacer clic en el icono de carrito, se llama al método _mostrarPedido(context) para mostrar los detalles del pedido actual.
2. **endDrawer:** Es un cajón desplegable que contiene el historial de pedidos. Si no hay pedidos en el historial, se muestra un mensaje indicando que no hay pedidos. En este caso al utilizar endDrawer lo ponemos a la derecha de la interfaz.
3. **body:** Una imagen de fondo. que contiene un Column de botones para diferentes tipos de hamburguesas. Cada botón está creado con _botonHamburguesa, que crea un botón con una imagen y un texto.

```

  Widget _botonHamburguesa(String hamburguesaImagePath, String nombre, double precio) {
    return Padding(
      padding: EdgeInsets.symmetric(vertical: 25.0, horizontal: 32.0),
      child: Stack(
        alignment: Alignment.bottomCenter,
        children: [
          Container(
            decoration: BoxDecoration(
              image: DecorationImage(
                image: AssetImage(hamburguesaImagePath),
                fit: BoxFit.fitHeight,
              ),
            ),
            height: 150,
          ),
          Align(
            alignment: Alignment.bottomCenter,
            child: ElevatedButton(
              onPressed: () {
                _agregarAlCarrito(nombre);
              },
              child: Text(
                nombre,
                textAlign: TextAlign.center,
              ),
            ),
          ),
```

```
        ],
      ),
    );
  }


  void mostrarSnackBar(BuildContext context, String mensaje) {
    final snackBar = SnackBar(
      content: Text(mensaje),
      duration: Duration(milliseconds: 500),
    );
    ScaffoldMessenger.of(context).showSnackBar(snackBar);
  }

  void _agregarAlCarrito(String hamburguesa) {
    setState(() {
      this._pedidoActual.add(hamburguesa);
      _cantidad++;
    });
    mostrarSnackBar(context, "Anadiendo al pedido una $hamburguesa ...");
  }

  void _mostrarPedido(BuildContext context) {
    double total = _calculaTotalPedido(_pedidoActual);
    showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: Text('Hamburguesas en el carrito'),
          content: SingleChildScrollView(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                ..._pedidoActual.map((hamburguesa) {
                  return Row(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    children: [
                      Text("${hamburguesa}"),
                      Text("${_getPrecioHamburguesa(hamburguesa)}"),
                    ],
                  );
                }).toList(),
                SizedBox(height: 20),
                Row(
                  mainAxisAlignment: MainAxisAlignment.end,
                  children: [
                    TextButton(
                      onPressed: () {
                        _clearPedido();
                        Navigator.of(context).pop();
                      },
                      child: Icon(Icons.delete),
                    ),
                    SizedBox(width: 10),
                    TextButton(
                      style: ButtonStyle(
                        backgroundColor:
                        MaterialStateProperty.all(Colors.redAccent),
                        foregroundColor:
                        MaterialStateProperty.all(Colors.black),
                      ),
                      child: Text('Finalizar Pedido'),
                      onPressed: () {
                        _finalizarPedido(context);
                        Navigator.of(context).pop();
                      },
                    ),
                    SizedBox(width: 20),
                    Text('Total:  '),
```

```
                    Text('${total.toStringAsFixed(2)}\    ',
                        style: TextStyle(fontWeight: FontWeight.bold)),
                  ],
                ),
              ],
            ),
          ),
        );
      },
    );
  }

  void _finalizarPedido(BuildContext context) async{

    if (_pedidoActual.isNotEmpty) {
      setState(() {
        mostrarSnackBar(context, "Cocinando pedido...");
        _actualizarHistorial();
        Future.delayed(Duration(seconds: 5), () {
          _cocinero.cocinaPedido(_pedidoActual, context);
        });
        _pedidoActual.clear();
        _cantidad = 0;
      });
    } else {
      mostrarSnackBar(context, "El pedido est  vac o");
    }
  }


  void _clearPedido() {

    setState(() {
      if (_pedidoActual.length > 0) {
        _pedidoActual.clear();
        _cantidad = 0;
        mostrarSnackBar(context, "Limpiando pedido...");
      }
      else {
        mostrarSnackBar(context, "El pedido est  vac o");
      }
    });
  }


  double _calculaTotalPedido(List<String> pedido) {
    double total = 0;
    for (String hamburguesa in pedido) {
      total += _getPrecioHamburguesa(hamburguesa);
    }
    return total;
  }


  double _getPrecioHamburguesa(String hamburguesa) {
    switch (hamburguesa) {
      case "Hamburguesa normal":
        {
          return 5;
        }
      case "Hamburguesa vegana":
        {
          return 6;
        }
      case "Hamburguesa sin gluten":
        {
          return 5.5;
```

```
165           }
166         default:
167           return 0;
168       }
169     }
170
171     void _actualizarHistorial() {
172       setState(() {});
173     }
174
175 }
```

Estos son el resto de métodos donde la mayoría encapsulan una funcionalidad pero el importante es _finalizarPedido,

```
1   void _finalizarPedido(BuildContext context) async{
2
3     if (_pedidoActual.isNotEmpty) {
4       setState(() {
5         mostrarSnackBar(context, "Cocinando pedido...");
6         _actualizarHistorial();
7         Future.delayed(Duration(seconds: 5), () {
8           _cocinero.cocinaPedido(_pedidoActual, context);
9         });
10        _pedidoActual.clear();
11        _cantidad = 0;
12      });
13    } else {
14      mostrarSnackBar(context, "El pedido est  vacio");
15    }
16  }
```

Este método se llama cuando pulsamos el botón de finalizar pedido, se usa para pasarle al cocinero las hamburguesas a preparar. La clave de este método es la palabra async indicando que es asíncrono, pudiendo realizar algunas líneas de código sin que se vea afectado el hilo de ejecución principal. En este caso se ejecuta la función _cocinero.cocinaPedido, 5 segundos más tarde después de haber llamado al método, una vez hecho esto limpiamos el pedido actual y reseteamos el carrito si el pedido estuviera vació lo mostramos por el snackBar.

Veamos ahora la clase cocinero para ver que sucede cuando llamamos al método para cocinar:

```
1  import 'package:pantalla_pedidos_hamburgueseria/model/HamburguesaSinGlutenBuilder.dart';
2  import 'package:pantalla_pedidos_hamburgueseria/model/HamburguesaVeganaBuilder.dart';
3
4  import 'package:flutter/material.dart';
5
6  import 'HamburguesaNormalBuilder.dart';
7  import 'ObservadorPedido.dart';
8  import 'Subject.dart';
9  import 'Pedido.dart';
10 import 'HamburguesaBuilder.dart';
11 import 'Hamburguesa.dart';
12
13 class Cocinero implements Subject {
14   late HamburguesaBuilder _builder;
15   String status = "Tomando pedido";
16   List<ObservadorPedido> observers = [];
17   Pedido pedidoActual = Pedido();
18
19   Cocinero(){}
20
21   Cocinero.Parametros(HamburguesaBuilder builder) {
22     _builder = builder;
23   }
24
25   Future<void> cocinaPedido(List<String> hamburguesas, BuildContext context) async{
```

```
26      for (String hamburguesa in hamburguesas) {
27        switch (hamburguesa) {
28          case "Hamburguesa normal":
29            {
30              cambiaReceta(HamburguesaNormalBuilder());
31            }
32            break;
33
34          case "Hamburguesa vegana":
35            {
36              cambiaReceta(HamburguesaVeganaBuilder());
37            }
38            break;
39
40          case "Hamburguesa sin gluten":
41            {
42              cambiaReceta(HamburguesaSinGlutenBuilder());
43            }
44            break;
45
46          default:
47            throw Exception("Tipo de hamburguesa no v lido");
48        }
49        await buildHamburguesa();
50      }
51      notify(context);
52    }
53
54    Future<void> buildHamburguesa() async {
55      status = "Cocinando";
56
57      _builder.aniadePan();
58      _builder.aniadeLechuga();
59      _builder.aniadeTomate();
60      if (_builder is HamburguesaNormalBuilder) {
61        (_builder as HamburguesaNormalBuilder).aniadeQuesoCabra();
62      }
63      _builder.aniadeCebolla();
64      _builder.aniadePepinillos();
65      _builder.aniadeBacon();
66      _builder.aniadeCarne();
67      _builder.aniadePrecio();
68
69      pedidoActual.aniadeHamburguesa(_builder.hamburguesa);
70      pedidoActual.listo = true;
71      status = "Tomando pedido";
72    }
73
74    void cambiaReceta(HamburguesaBuilder builder) {
75      _builder = builder;
76    }
77
78    Pedido getPedido(){
79      return this.pedidoActual;
80    }
81
82    @override
83    void attach(ObservadorPedido observer) {
84      observers.add(observer);
85    }
86
87    @override
88    void detach(ObservadorPedido observer) {
89      observers.remove(observer);
90    }
91
92    @override
93    void notify(BuildContext context) {
94      for (var notified in observers) {
```

```
95          notified.update(pedidoActual, context);
96        }
97        pedidoActual = Pedido();
98      }
99    }
```

Los dos métodos que tienen relevancia son cocinaPedido y buildHamburguesa:

- **cocinaPedido:** Itera sobre las hamburguesas para ir construyendolas de una en una, gracias a la línea await buildHarmguesa, no se hace la siguiente iteración hasta que la hamburgusa primera no esté hecha
- **buildHamburguesa:** Construye la hamburguesa, aunque en esté método no se utilice ninguna espera, tenemos que declararlo como async por que es un método al que tenemos que esperar.

El último cambio importante que tenemos en el código sería en la clase DisplayPedidos:

```
1  import 'ObservadorPedido.dart';
2  import 'Pedido.dart';
3  import 'package:flutter/material.dart';
4
5  class DisplayPedidos implements ObservadorPedido {
6    List<Pedido> historial = [];
7
8    late ScaffoldMessengerState _scaffoldMessengerState;
9    Function() _actualizarHistorial;
10
11   DisplayPedidos(this.historial, this._actualizarHistorial);
12
13   void init(BuildContext context) {
14     _scaffoldMessengerState = ScaffoldMessenger.of(context);
15   }
16
17   void mostrarSnackBar(BuildContext context, String mensaje) {
18     final snackBar = SnackBar(
19       content: Text(mensaje),
20       duration: Duration(seconds: 5),
21       backgroundColor: Colors.redAccent,
22     );
23     _scaffoldMessengerState.showSnackBar(snackBar);
24   }
25
26   @override
27   void update(Pedido pedido, BuildContext context) {
28     if (!_scaffoldMessengerState.mounted) {
29
30       return;
31     }
32     historial.add(pedido);
33     mostrarSnackBar(context, 'El pedido ${pedido.idPedido} esta listo');
34     _actualizarHistorial();
35   }
36 }
```

En este clase tenemos varios puntos nuevos, vamos a verlos para entender el funcionamiento del display:

- ScaffoldMessengerState _scaffoldMessengerState: es una referencia al estado de ScaffoldMessenger para mostrar el snackBar, lo declaramos como late para incializarlo más tarde cuando llamemos al constructor.
- Funcion() _actualizarHistorial: Una función que se utiliza para actualizar el historial de pedidos.

Luego tenemos la clase update que es la encarga de actualizar el estado de los observadores. En este caso el funcionamiento es sencillo:

1. Comprabamos si el _scaffoldMessengerState está montado, esto significa que está presente en el árbol de widgets, es decir, es visible por el usuario, e intentar mostrarlo podría generar un error.
2. Añadimos el pedido al historial
3. Mostramos en el snackBar el pedido
4. Actualizamos el historial

Como hemos visto antes el método _actualizarHistorial en la clase Menú no tiene implementación, esto es porque queremos actualizar el estado, la función setState() se utiliza para notificar a Flutter que el estado del widget ha cambiado y necesita volver a llamar al método build() del widget para reconstruir la interfaz de usuario con los nuevos cambios