

UNIVERSIDAD DE GRANADA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA
Y DE TELECOMUNICACIÓN



UNIVERSIDAD
DE GRANADA

DESARROLLO DE SOFTWARE

Practica 2

Ejercicio Grupal en Flutter

Profesor: Alberto Jesús Durán López

Alumnos:

Victor CASALINI GONZALEZ

Joaquin SALAS CASTILLO

Antonio FERNANDEZ SANTIAGO

Kaito LORENZO OKOCHI

21 de abril de 2024

Índice

- 1 Ejercicio Grupal
 - Adaptacion y Perfeccion de la Actividad 3 de la Practica 1 en Flutter/Dart **2**
 - 1.1 Patrón Builder 4
 - 1.2 Patrón Decorador 4
 - 1.3 Patrón Fachada 5
 - 1.4 Patrón Factory Method 5
 - 1.5 Patrón Singleton 6

Los resultados se pueden encontrar en github a traves del siguiente enlace https://github.com/vicmaviclu/DS/tree/master/P2/ejercicio_grupal

1. Ejercicio Grupal

Adaptacion y Perfeccion de la Actividad 3 de la Practica 1 en Flutter/Dart

En esta práctica el objetivo es adaptar la aplicación realizada en el ejercicio 3 de la práctica anterior a un nuevo lenguaje de programación, en este caso utilizando el framework flutter que nos permite realizar interfaces de usuario personalizadas y atractivas.

Para ello, hemos llevado a cabo un mantenimiento adaptativo sobre el patrón Builder, asegurándonos de que se ajuste a las especificaciones del lenguaje Dart. Este proceso implica adaptar las estructuras y convenciones del patrón Builder para que sean coherentes con las características y sintaxis de Dart, garantizando así su correcto funcionamiento dentro del contexto de Flutter.

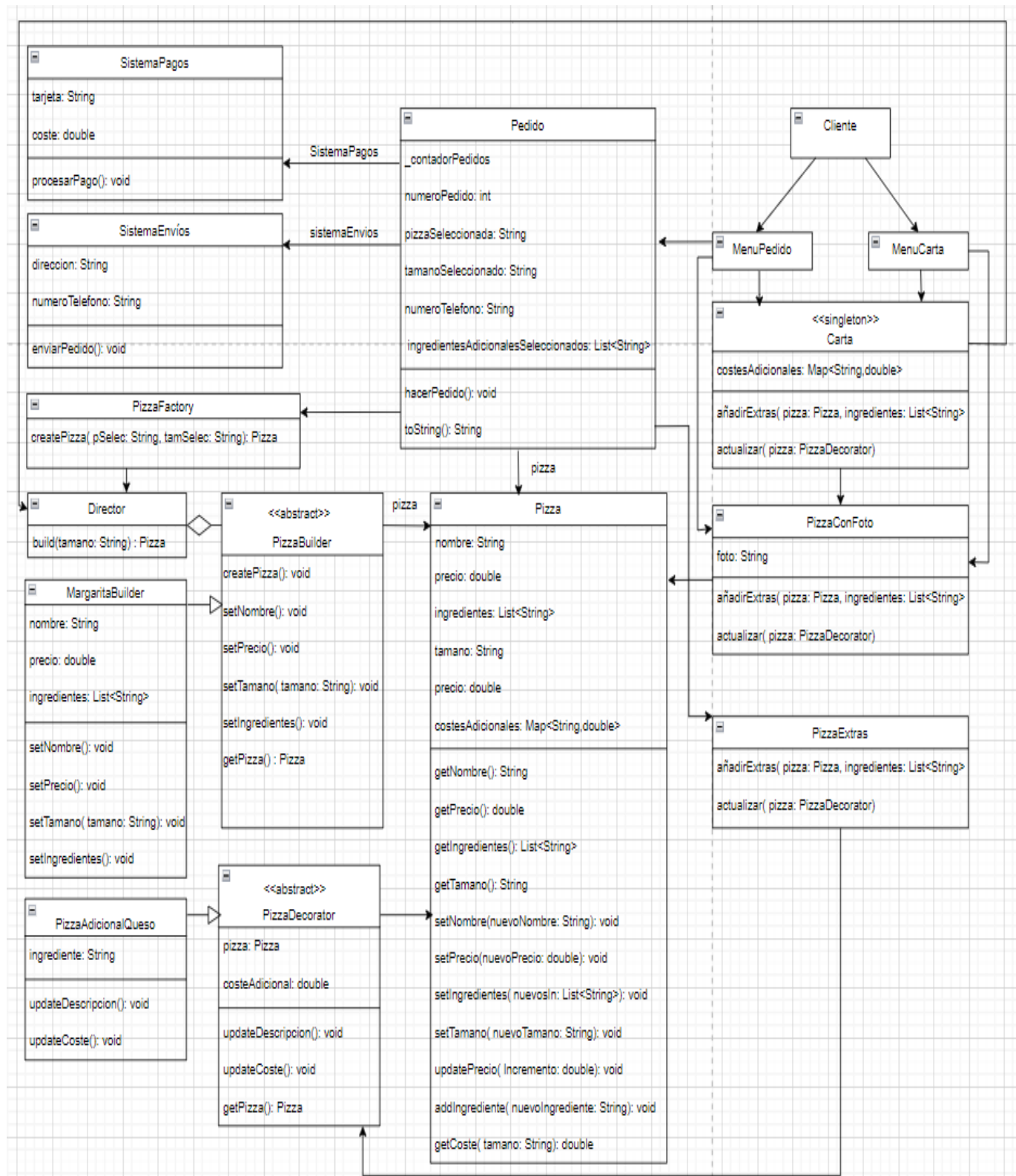
Además, como parte del mantenimiento perfectivo, hemos introducido nuevos patrones de diseño que se comunican entre sí de manera efectiva, mejorando así la funcionalidad de la aplicación.

Descripcion de la aplicación

Hemos decidido desarrollar una aplicación móvil dedicada a una pizzeria. Esta aplicación tiene como objetivo principal ofrecer a los usuarios la posibilidad de consultar la carta, que incluye una amplia variedad de pizzas, así como realizar pedidos.

- **Consulta de la Carta:** Los usuarios pueden acceder a la carta mediante la aplicación. Esta sección incluye el nombre de la pizza, el precio, una foto y una lista de ingredientes .
- **Realización de Pedidos:** En esta sección el usuario podrá realizar un pedido especificando la pizza deseada, el tamaño, la dirección de entrega, la tarjeta de crédito para el pago y un número de telefono de contacto. Además, tendrá la posibilidad de añadir ingredientes adicionales al pedido.

Diagrama



1.1. Patrón Builder

Este patrón lo hemos utilizado al igual que en la práctica anterior para que el director construya el tipo de pizza deseado. En el diagrama solo aparece MargaritaBuilder ya que las demás se implementan igual y no cabían todas. Se implementa en PizzaBuilder.

```
abstract class PizzaBuilder {  
    late Pizza pizza;  
  
    void createPizza() {  
        pizza = Pizza();  
    }  
  
    void setNombre();  
    void setPrecio();  
    void setTamano(String tamano);  
    void setIngredientes();  
  
    Pizza getPizza() {  
        return pizza;  
    }  
}
```

1.2. Patrón Decorador

Este patrón se implementa para poder añadir ingredientes extra a la pizza base elegida. Al igual que en el caso anterior, en el diagrama solo aparece PizzaConQueso para simplificar.

```

abstract class PizzaDecorator extends Pizza {
    Pizza pizza;
    final double costeAdicional = 0.5;

    PizzaDecorator(this.pizza);

    void updateDescription();

    void updateCoste();

    Pizza get getPizza => pizza;
}

```

1.3. Patrón Fachada

Este patrón lo hemos utilizado para unificar la gestión de envío/pago y Pizza-Factory haciendo que así sea mucho más sencillo. En el diagrama, este patrón se implementa en la clase Pedido.

```

void hacerPedido() {
    sistemaPagos.procesarPago();
    sistemaEnvios.enviarPedido();
    pizza = PizzaFactory.createPizza(pizzaSeleccionada, tamanoSeleccionado);
    if (ingredientesAdicionalesSeleccionados.isNotEmpty) {
        PizzaExtras.anadirExtras(pizza!, ingredientesAdicionalesSeleccionados);
    }
    sistemaPagos.coste = pizza!.getCoste(tamanoSeleccionado);
    print('Pedido realizado');
}

```

1.4. Patrón Factory Method

Este patrón lo hemos utilizado para poder elegir el tipo de pizza mediante un switch y construir el director directamente en cada opción con el builder correspondiente para que sea más sencillo. Se implementa en la clase PizzaFactory.

```

class PizzaFactory {
    static Pizza createPizza(String pizzaSeleccionada, String tamanoSeleccionado) {
        switch (pizzaSeleccionada) {
            case 'Pizza Margarita':
                return Director(MargaritaBuilder()).build(tamanoSeleccionado);
            case 'Pizza Pepperoni':
                return Director(PepperoniBuilder()).build(tamanoSeleccionado);
            case 'Pizza Vegetariana':
                return Director(Vegetal()).build(tamanoSeleccionado);
            case 'Pizza Burrata Pesto':
                return Director(BurrataPesto()).build(tamanoSeleccionado);
            case 'Pizza Pistacho':
                return Director(Pistacho()).build(tamanoSeleccionado);
            case 'Pizza Tartufa':
                return Director(Tartufa()).build(tamanoSeleccionado);
            default:
                throw Exception('Tipo de pizza no reconocido: $pizzaSeleccionada');
        }
    }
}

```

1.5. Patrón Singleton

Este patrón lo hemos utilizado en la clase Carta para que haya una única instancia de esta.

```

class Carta {
    List<PizzaConFoto> pizzas;
    Map<String, double> costesAdicionales = {
        'Mediana': 1.0,
        'Grande': 2.0,
        'Gigante': 3.0,
    };

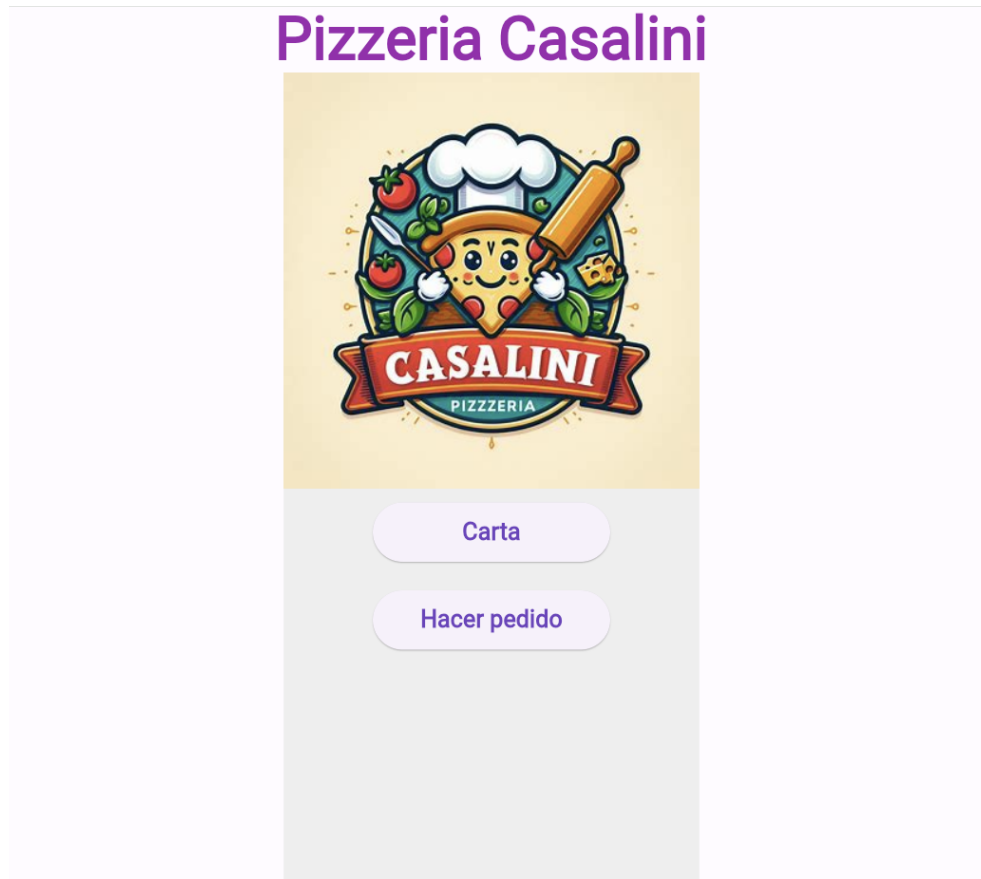
    static final Carta _singleton = Carta._internal();

    factory Carta() {
        return _singleton;
    }
}

```

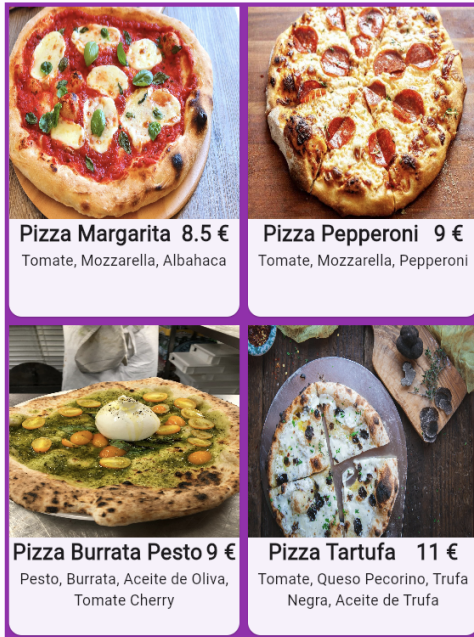
Widgets Utilizados

Para la UI del programa hemos utilizado los siguientes widgets:



En la página inicial hemos usado Text para poner el nombre de la pizzeria, Container para definir la seccion del cuerpo con un tamaño de móvil y Column y Center para colocar los distintos elementos centrados en vertical. Dentro del cuerpo tenemos image.asset para mostrar la imagen del logotipo y Elevated-Button para los botones de la carta y de hacer pedido, de color morado ya que hemos utilizado ThemeData y ColorScheme para cambiarlo.

Carta de pizzas



En la página de la carta tenemos además GridView.count para mostrar las pizzas en forma de matriz.

Menú de Pedido

Selecciona una pizza ▼

Selecciona un tamaño ▼

Dirección

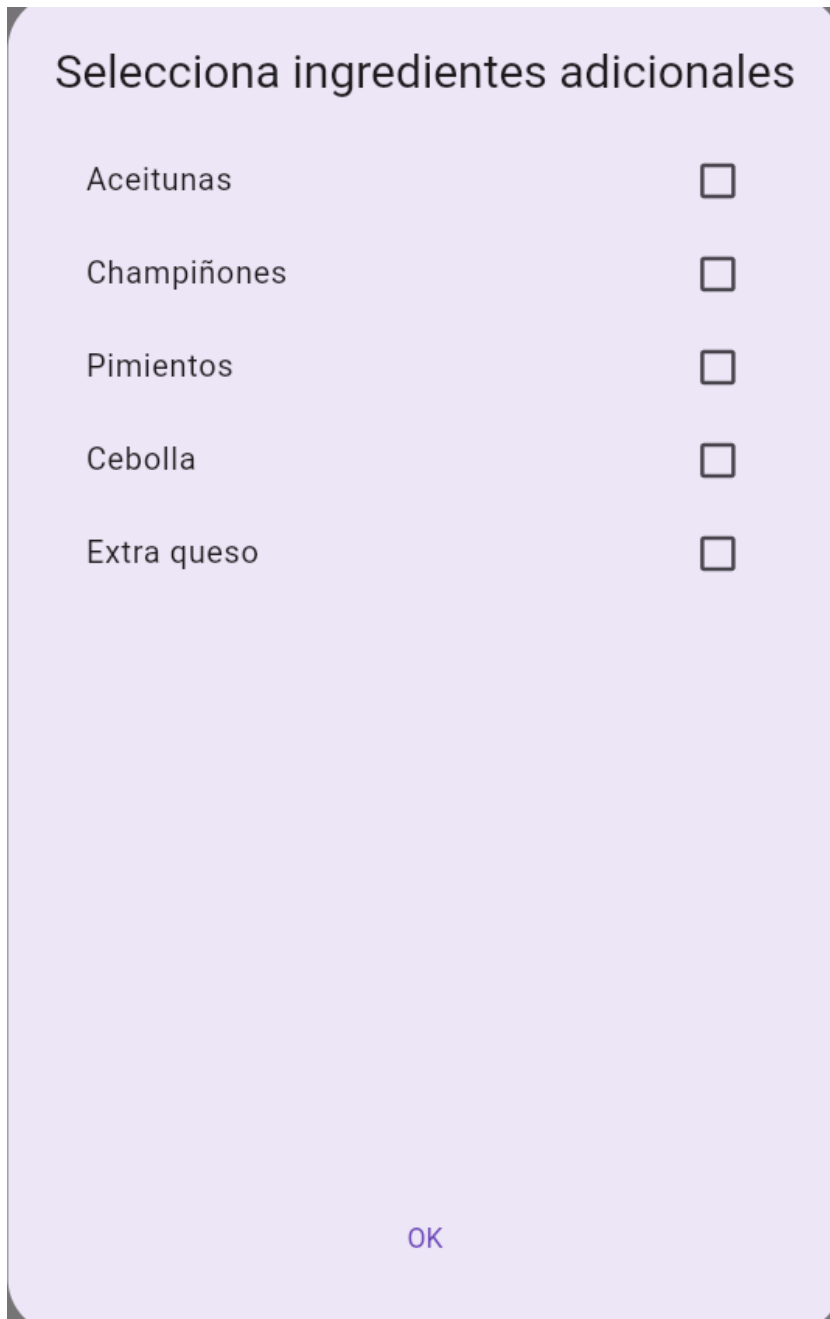
Tarjeta de Crédito

Número de Teléfono

Añadir ingredientes adicionales

Realizar Pedido

En la página para hacer el pedido hemos usado `DropDownButton` y `DropDownMenuItem` para los botones desplegables y sus posibles opciones, `TextField` para los campos de texto y `ElevatedButton` para los botones normales. Además también hemos añadido `AlertDialog` para mostrar la ventana emergente de los ingredientes y para notificar errores al querer realizar el pedido.



The image shows a screenshot of a mobile application interface. It features a light purple background with rounded corners. At the top, there is a title "Selecciona ingredientes adicionales" in a bold, dark font. Below the title, there is a list of five ingredients, each followed by a square checkbox: "Aceitunas", "Champiñones", "Pimientos", "Cebolla", and "Extra queso". At the bottom center of the dialog, there is a blue button labeled "OK".

Ingrediente	Seleccionado
Aceitunas	<input type="checkbox"/>
Champiñones	<input type="checkbox"/>
Pimientos	<input type="checkbox"/>
Cebolla	<input type="checkbox"/>
Extra queso	<input type="checkbox"/>

OK

Para la ventana de los ingredientes hemos usando CheckboxListTile para la lista y TextButton para el boton de confirmación.

Mejoras Perfectivas

Una de las mejoras perfectivas que hemos implementado para mejorar el programa ha sido la siguiente:

```
void hacerPedido() {
    sistemaPagos.procesarPago();
    sistemaEnvios.enviarPedido();
    pizza = PizzaFactory.createPizza(pizzaSeleccionada, tamanoSeleccionado);
    if (ingredientesAdicionalesSeleccionados.isNotEmpty) {
        PizzaExtras.anadirExtras(pizza!, ingredientesAdicionalesSeleccionados);
    }
    sistemaPagos.coste = pizza!.getCoste(tamanoSeleccionado);
    print('Pedido realizado');
}
```

Añadiendo un if para comprobar antes de añadir ingredientes que la lista no esté vacía, para así no llamar al método cuando no haya nada y mejorar la eficiencia.

Otra mejora que hemos implementado ha sido añadir métodos getters/setters en la clase Pizza para gestionar mejor sus atributos

```
String get getNombre => nombre;
double get getPrecio => precio;
List<String> get getIngredientes => ingredientes;
String get getTamano => tamano;

set setNombre(String nuevoNombre) {
    nombre += nuevoNombre;
}

set setPrecio(double nuevoPrecio) {
    precio = nuevoPrecio;
}
```

Problemas Encontrados

Uno de los problemas con los que nos hemos encontrado mientras realizábamos la práctica ha sido el de mostrar imágenes, ya que además de añadirlas a una carpeta assets, había que modificar el archivo pubspec.yaml para definirlas y luego utilizar las funciones correspondientes de dart para mostrarlas en el programa, lo cual ha resultado muy poco intuitivo.

Para solucionarlo tuvimos que investigar en internet hasta que al final entendimos cómo se hacía.