Práctica 2. Ejercicio Grupal
Aplicación para Gestión de Empresa

Integrantes del grupo

- Carmen Chunyin Fernández Núñez
- Pablo García Guijosa
- Marta Xiaoyang Moraga Hernández
- Jesús Navarrete Caparrós

Índice

Descripción

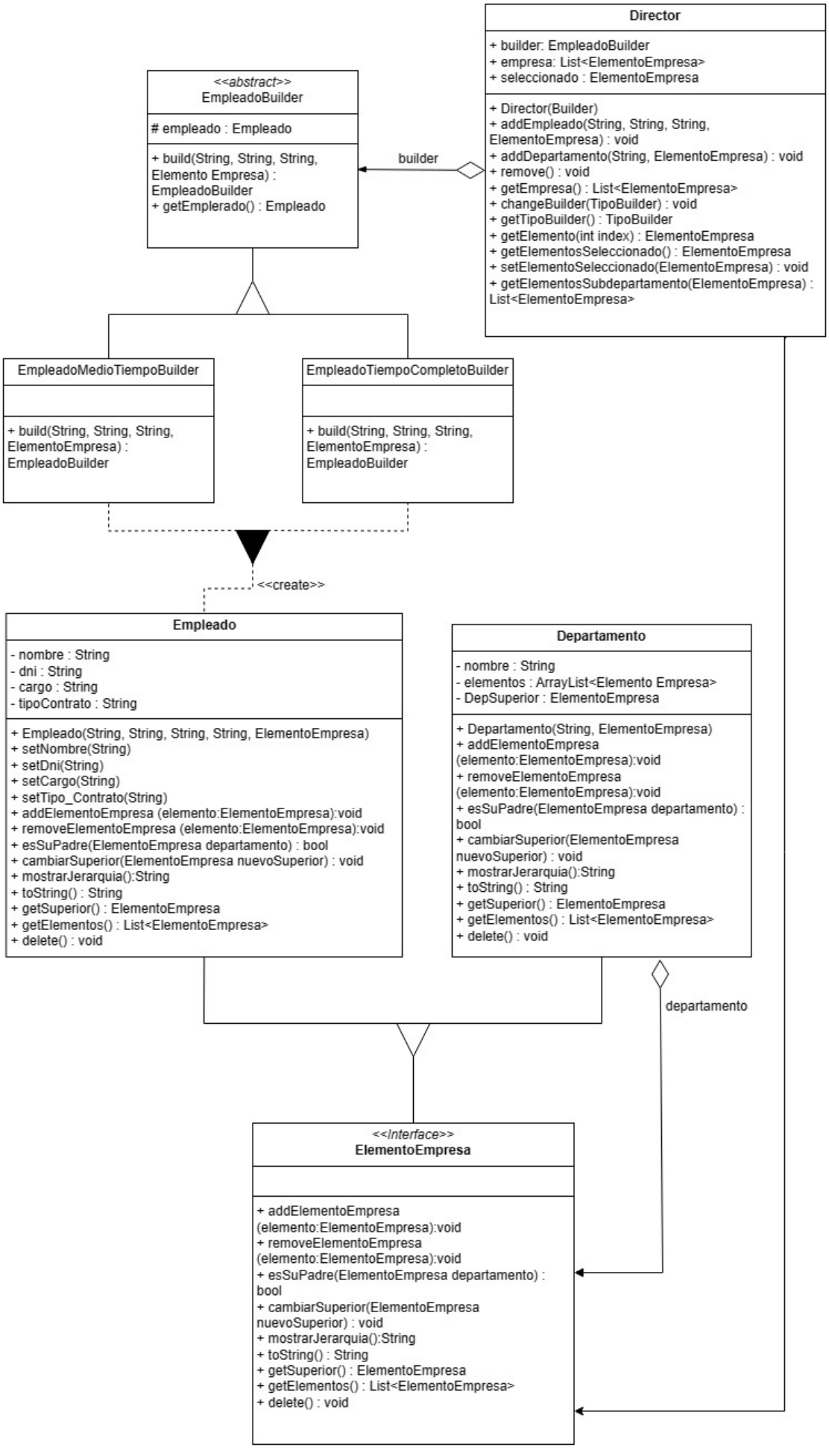
En esta práctica hemos revisado y mejorado el ejercicio planteado anteriormente. Nuestro objetivo es simular la organización de una empresa, utilizando el patrón Builder para construir diferentes tipos de empleados y el Composite para manejar la jerarquía de la empresa. Hemos realizado modificaciones principalmente en el Patrón Composite, ya que el enfoque utilizado en la práctica anterior no era completamente correcto.

Ahora, en lugar de tener múltiples arrays en la clase Departamento para almacenar diferentes tipos de elementos de la empresa, utilizamos un único array que guarda objetos del tipo Elemento Empresa. Además, hemos incluido en la clase Director un array de Elemento Empresa para que sea el Director quien maneje todo lo relacionado con la empresa.

El objetivo de esta práctica, además de mejorar el código anterior, es crear una interfaz para nuestro proyecto utilizando Flutter y Dart. En esta interfaz se mostrará la jerarquía de la empresa, la cual será controlada únicamente por el director. Además, se podrán añadir y eliminar los empleados y departamentos desde la interfaz.

Diagrama

[Imagen UML en la siguiente página]


Director
+ builder: EmpleadoBuilder
+ empresa: List<ElementoEmpresa>
+ seleccionado : ElementoEmpresa
+ Director(Builder)
+ addEmpleado(String, String, String, ElementoEmpresa) : void
+ addDepartamento(String, ElementoEmpresa) : void
+ remove() : void
+ getEmpresa() : List<ElementoEmpresa>
+ changeBuilder(TipoBuilder) : void
+ getTipoBuilder() : TipoBuilder
+ getElemento(int index) : ElementoEmpresa
+ getElementosSeleccionado() : ElementoEmpresa
+ setElementoSeleccionado(ElementoEmpresa) : void
+ getElementosSubdepartamento(ElementoEmpresa) : List<ElementoEmpresa>
<<abstract>>
EmpleadoBuilder
empleado : Empleado
+ build(String, String, String, Elemento Empresa) : EmpleadoBuilder
+ getEmplerado() : Empleado
builder
EmpleadoMedioTiempoBuilder
+ build(String, String, String, ElementoEmpresa) : EmpleadoBuilder
EmpleadoTiempoCompletoBuilder
+ build(String, String, String, ElementoEmpresa) : EmpleadoBuilder
<<create>>
Empleado
- nombre : String
- dni : String
- cargo : String
- tipoContrato : String
+ Empleado(String, String, String, String, ElementoEmpresa)
+ setNombre(String)
+ setDni(String)
+ setCargo(String)
+ setTipo_Contrato(String)
+ addElementoEmpresa (elemento:ElementoEmpresa):void
+ removeElementoEmpresa (elemento:ElementoEmpresa):void
+ esSuPadre(ElementoEmpresa departamento) : bool
+ cambiarSuperior(ElementoEmpresa nuevoSuperior) : void
+ mostrarJerarquia():String
+ toString() : String
+ getSuperior() : ElementoEmpresa
+ getElementos() : List<ElementoEmpresa>
+ delete() : void
Departamento
- nombre : String
- elementos : ArrayList<Elemento Empresa>
- DepSuperior : ElementoEmpresa
+ Departamento(String, ElementoEmpresa)
+ addElementoEmpresa (elemento:ElementoEmpresa):void
+ removeElementoEmpresa (elemento:ElementoEmpresa):void
+ esSuPadre(ElementoEmpresa departamento) : bool
+ cambiarSuperior(ElementoEmpresa nuevoSuperior) : void
+ mostrarJerarquia():String
+ toString() : String
+ getSuperior() : ElementoEmpresa
+ getElementos() : List<ElementoEmpresa>
+ delete() : void
departamento
<<Interface>>
ElementoEmpresa
+ addElementoEmpresa (elemento:ElementoEmpresa):void
+ removeElementoEmpresa (elemento:ElementoEmpresa):void
+ esSuPadre(ElementoEmpresa departamento) : bool
+ cambiarSuperior(ElementoEmpresa nuevoSuperior) : void
+ mostrarJerarquia():String
+ toString() : String
+ getSuperior() : ElementoEmpresa
+ getElementos() : List<ElementoEmpresa>
+ delete() : void

Modelo

A la hora de modificar el código, hemos tenido ciertos problemas pues el cambio de java a flutter, además de mejorar el código, ha implicado tener que cambiar bastantes cosas. En muchas variables y métodos hemos tenido que poner ? ya que flutter no deja que una variable sea nula a menos que pongamos la interrogación. También hemos tenido que añadir una clase TipoBuilder, que es un enumerador, para utilizar un menú desplegable en la interfaz y seleccionar el tipo de Builder que deberá utilizar el director.

Departamento

La clase Departamento hereda de la clase ElementoEmpresa y tiene un array para almacenar diferentes tipos de ElementoEmpresa. También tiene un ElementoEmpresa DepSuperior que indica cual es el Departamento Superior al que pertenece, si lo hubiese.

En Dart, cuando se quiere llamar al método de un objeto que podría ser nulo se debe llamar con una interrogación. Por ello, al adaptarlo de Java, aunque se compruebe antes que es nulo y gestionemos que hacer en tal caso, muchos de los métodos deben ser llamados de la forma departamento?.metodo(). En tal caso de que el Departamento sea nulo, el método no se ejecuta.

Esto es una cosa que afecta bastante cuando es necesario usar el Departamento Superior (DepSuperior). Ya que podría ser nulo (el ElementoEmpresa no pertenece a un departamento) o no.

```dart
import 'ElementoEmpresa.dart';

class Departamento extends ElementoEmpresa {
  String nombre = '';
  late List<ElementoEmpresa> elementos;
  ElementoEmpresa? DepSuperior;

  Departamento(String nombre, ElementoEmpresa? superior) {
    this.nombre = nombre;
    elementos = <ElementoEmpresa>[];

    if (superior != null) {
      DepSuperior = superior;
      DepSuperior?.addElementoEmpresa(this);
       }
    }

  @override
  void addElementoEmpresa(ElementoEmpresa elemento) {
    // TODO: implement addElementoEmpresa
    elementos.add(elemento);
    }

  @override
  bool? esSuPadre(ElementoEmpresa departamento) {
    if (this.DepSuperior == null) {
      return false;
    } else if (departamento == this.DepSuperior) {
      return true;
    } else {
      bool? r = this.DepSuperior?.esSuPadre(departamento);
      if (r == null || r == false)
        return false;
      else
        return true;
       }
    }

  @override
  void cambiarSuperior(ElementoEmpresa? nuevoSuperior) {
    this.DepSuperior = nuevoSuperior;
    }
```

```
44    @override
45    void removeElementoEmpresa(ElementoEmpresa? elemento) {
46      elementos.remove(elemento);
47      elemento?.delete();
48      }
49
50    @override
51    ElementoEmpresa getElementoEmpresa(int index) {
52      return elementos[index];
53      }
54
55    @override
56    String mostrarJerarquia() {
57      String string = "Departamento: " + this.nombre + "\n";
58      for (int i = 0; i < elementos.length; i++) {
59        string += "\t" + elementos[i].toString();
60         }
61      return string;
62      }
63
64    @override
65    List<ElementoEmpresa> getElementos() {
66      return elementos;
67      }
68
69    @override
70    String toString() {
71      String s = this.nombre;
72      return s;
73      }
74
75    @override
76    ElementoEmpresa? getElemento(ElementoEmpresa elemento) {
77      for (int i = 0; i < elementos.length; i++) {
78        if (elementos[i] == elemento) return elemento.getElementoEmpresa(i);
79         }
80      return null;
81      }
82
83    @override
84    ElementoEmpresa? getSuperior(){
85      return DepSuperior;
86      }
87
88    @override
89    void delete(){
90      for(int i = 0; i< elementos.length; i++){
91        elementos[i].delete();
92         }
93      DepSuperior = null;
94      elementos.clear();
95      }
96  }
```

ElementoEmpresa

Esta interfaz, definida como clase abstracta en dart (puesto que los interface de Dart obligan a incluir implementación {}, aunque esta esté vacía, y por tanto esta es una adaptación más fiel a la intención del código original de Java), es la que implementan el Departamento y Empleado. En ella, se definen los métodos a utilizar por los objetos tipo ElementoEmpresa. Gracias a que existe ElementoEmpresa, hay cohesión entre Empleado y Departamento, y Departamento puede gestionar tener ambos en su interior. Esta es la razón de que se eligiera utilizar el patrón Composite para el problema diseñado.

```dart
abstract class ElementoEmpresa {

  void addElementoEmpresa(ElementoEmpresa elemento);

  void removeElementoEmpresa(ElementoEmpresa? elemento);

  bool? esSuPadre(ElementoEmpresa departamento);

  void cambiarSuperior(ElementoEmpresa? nuevoSuperior);

  ElementoEmpresa getElementoEmpresa(int index);

  ElementoEmpresa? getElemento(ElementoEmpresa elemento);

  String mostrarJerarquia();

  @override
  String toString();

  ElementoEmpresa? getSuperior();

  List<ElementoEmpresa> getElementos();

  void delete();
 }
```

Empleado

La clase Empleado hereda de la clase ElementoEmpresa, además para crear este tipo de objetos, se utilizan los Builders. Estos Builders los utilizará el direcctior según sea necesario. La mayoría de los métodos de ElementoEmpresa, están implementdos como errores, pues hay muchas funcionalidades que los Empleados no deberían poder hacer.

```dart
import 'ElementoEmpresa.dart';

class Empleado extends ElementoEmpresa {
  String nombre = '';
  String dni = '';
  String cargo = '';
  String tipoContrato = '';
  ElementoEmpresa? DepSuperior;

  Empleado.vacio(){

    }

  Empleado(nombre, String dni, String cargo, String tipoContrato,
      ElementoEmpresa? superior) {
    this.nombre = nombre;
    this.dni = dni;
    this.cargo = cargo;
    this.tipoContrato = tipoContrato;
    if (superior != null) {
      DepSuperior = superior;
      DepSuperior?.addElementoEmpresa(this);
       }
    }

  @override
  String mostrarJerarquia() {
    String string = "Empleado:\n";
    string += "\tNombre: " + this.nombre + " \n";
    string += "\tDNI: " + this.dni + " \n";
    string += "\tCargo: " + this.cargo + " \n";
    string += "\tContrato actual: " + this.tipoContrato + " \n";
    return string;
    }
```

```dart
@override
void addElementoEmpresa(ElementoEmpresa elemento) {
  throw UnimplementedError();
 }

@override
void removeElementoEmpresa(ElementoEmpresa? elemento) {
  throw UnimplementedError();
 }

@override
bool? esSuPadre(ElementoEmpresa departamento) {
  throw UnimplementedError();
 }

@override
void cambiarSuperior(ElementoEmpresa? nuevoSuperior) {
  throw UnimplementedError();
 }

@override
ElementoEmpresa getElementoEmpresa(int index) {
  return this;
 }

@override
String toString() {
  String s = this.nombre;
  return s;
 }
```

```
83    ElementoEmpresa? getElemento(ElementoEmpresa elemento) {
84      return null;
85      }
86
87    String getDni(){
88      return dni;
89      }
90    String getTipoContrato(){
91      return tipoContrato;
92      }
93    String getCargo(){
94      return cargo;
95      }
96    @override
97    ElementoEmpresa? getSuperior(){
98      return DepSuperior;
99      }
100   @override
101   List<ElementoEmpresa> getElementos() { throw UnimplementedError();}
102
103   @override
104   void delete() {
105     DepSuperior = null;
106     }
107 }
```

EmpleadoBuilder

Este clase abstracta es la que permite construir el objeto Empleado, en el director se elige que tipo de Builder se quiere utilizar y se llama a la función build del Builder especificado

```dart
import 'package:ejercicio_grupal/Model/ElementoEmpresa.dart';
import 'Empleado.dart';

abstract class EmpleadoBuilder {
    late Empleado empleado;

    EmpleadoBuilder(ElementoEmpresa? DepSuperior)
       {
        empleado = Empleado.vacio();
       }

    EmpleadoBuilder build(String nombre, String dni, String cargo, ElementoEmpresa? DepSuperior);

    Empleado getEmpleado(){
        return Empleado(empleado.toString(), empleado.getDni(), empleado.getCargo(), empleado.getTipoContrato(), empleado.getSuperior());
       }

}
```

EmpleadoMedioTiempoBuilder

Hereda de EmpleadoBuilder y crea Empleados con un contrato de Medio Tiempo

```dart
import 'package:ejercicio_grupal/Model/ElementoEmpresa.dart';
import 'package:ejercicio_grupal/Model/Empleado.dart';
import 'package:ejercicio_grupal/Model/EmpleadoBuilder.dart';

class EmpleadoMedioTiempoBuilder extends EmpleadoBuilder{

  EmpleadoMedioTiempoBuilder(ElementoEmpresa? DepSuperior) : super(DepSuperior);

  @override
  EmpleadoBuilder build(String nombre, String dni, String cargo,ElementoEmpresa? DepSuperior) {
    empleado = Empleado.vacio();
    empleado.setNombre(nombre);
    empleado.setDni(dni);
    empleado.setCargo(cargo);
    empleado.setTipoContrato("Medio Tiempo");
    if (DepSuperior != null) {
      empleado.DepSuperior = DepSuperior;
      empleado.DepSuperior?.addElementoEmpresa(empleado);
       }
    return this;
    }

 }
```

EmpleadoTiempoCompletoBuilder

Hereda de EmpleadoBuilder y crea Empleados con un contrato de Tiempo Completo

```dart
import 'package:ejercicio_grupal/Model/ElementoEmpresa.dart';
import 'package:ejercicio_grupal/Model/Empleado.dart';
import 'package:ejercicio_grupal/Model/EmpleadoBuilder.dart';

class EmpleadoTiempoCompletoBuilder extends EmpleadoBuilder{

  EmpleadoTiempoCompletoBuilder(ElementoEmpresa? DepSuperior) : super(DepSuperior);

  @override
  EmpleadoBuilder build(String nombre, String dni, String cargo, ElementoEmpresa? DepSuperior) {
    empleado = Empleado.vacio();
    empleado.setNombre(nombre);
    empleado.setDni(dni);
    empleado.setCargo(cargo);
    empleado.setTipoContrato("Tiempo Completo");
    if (DepSuperior != null) {
      empleado.DepSuperior = DepSuperior;
      empleado.DepSuperior?.addElementoEmpresa(empleado);
       }
    return this;
    }
 }
```

TipoBuider

Esta clase está creada para facilitar en la interfaz la selección del builder a utilizar.

```dart
enum TipoBuilder {
  completo('Tiempo Completo'),
  parcial('Medio Tiempo');

  const TipoBuilder(this.label);
  final String label;
 }
```

Controlador

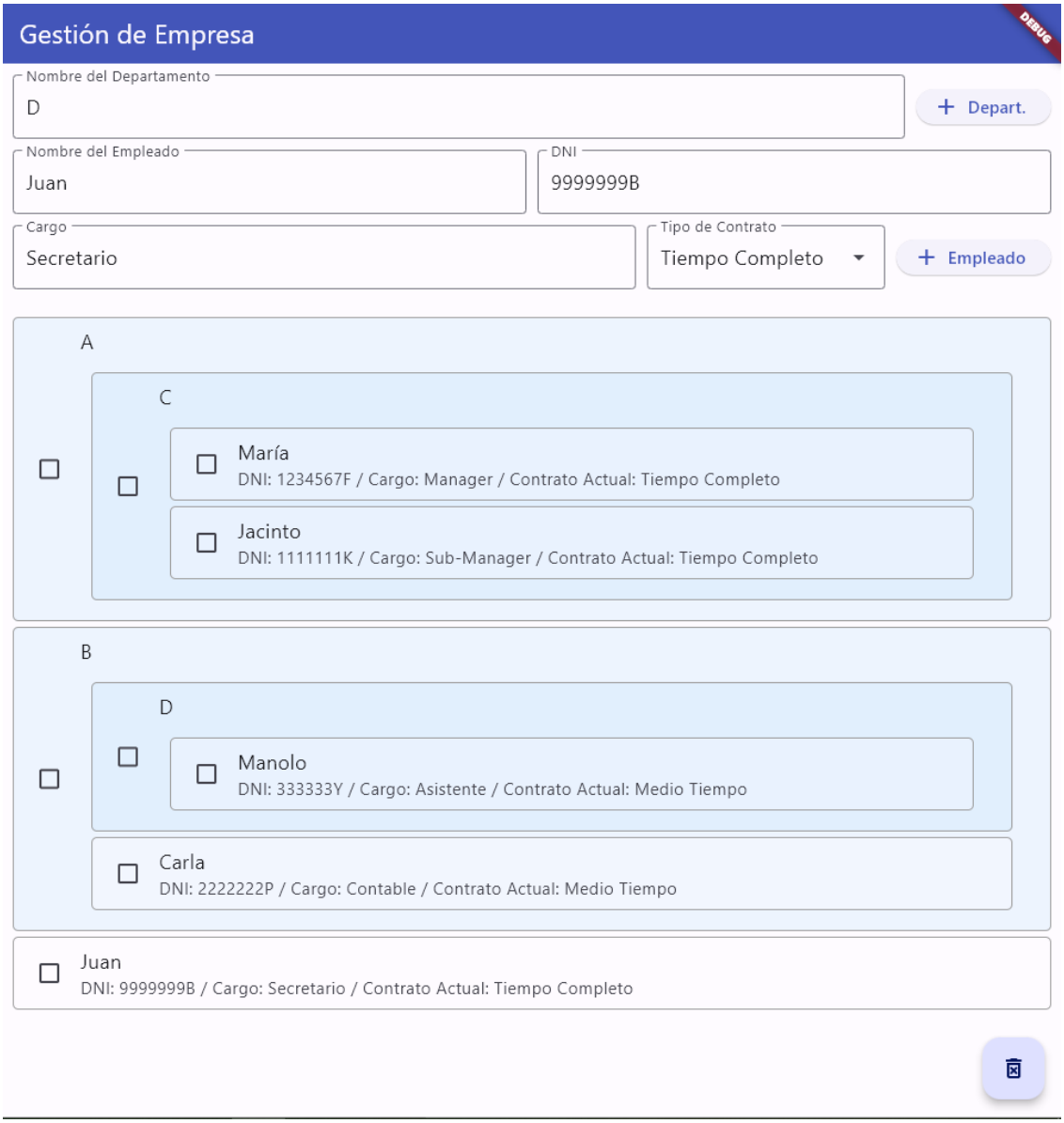
Director

El director es el que se encarga de controlar toda la empresa. Esta clase tiene un objeto builder, para construir los empleados, un array de ElementoEmpresa para manejar la jerarquía de la empresa y un objeto ElementoEmpresa que sería el objeto sobre el cual se está trabajando actualmente. El objeto seleccionado lo hemos creado porque al ser el atributo empresa un array, era más difícil trabajar sobre él, y para poder guardar sobre qué objeto se desea trabajar.

```dart
import 'package:ejercicio_grupal/Model/EmpleadoMedioTiempoBuilder.dart';
import 'package:ejercicio_grupal/Model/EmpleadoTiempoCompletoBuilder.dart';
import 'package:ejercicio_grupal/Model/TipoBuilder.dart';

import 'EmpleadoBuilder.dart';
import 'ElementoEmpresa.dart';
import 'Departamento.dart';

class Director {
  late EmpleadoBuilder builder;
  late List<ElementoEmpresa> empresa;
  ElementoEmpresa? seleccionado;

  Director(EmpleadoBuilder builder)
    {
    this.builder = builder;
    empresa = <ElementoEmpresa>[];
    }

  void addEmpleado(String nombre, String dni, String cargo, ElementoEmpresa? superior) {
    builder.build(nombre, dni, cargo, superior);
    if(seleccionado == null){
      empresa.add(builder.getEmpleado());
       }


    }

  void addDepartamento(String nombre, ElementoEmpresa? superior){
    if(seleccionado == null) {
      empresa.add(Departamento(nombre, null));
    } else{
      Departamento(nombre, superior);
       }
    }

  void remove(){
    if(seleccionado != null){
      if(seleccionado?.getSuperior() == null){
        empresa.remove(seleccionado);
        seleccionado?.delete();
      }else{
        seleccionado?.getSuperior()?.removeElementoEmpresa(seleccionado);
          }
      seleccionado = null;
       }
    }

  List<ElementoEmpresa> getEmpresa(){
    return empresa;
    }
```

```
53    void changeBuilder(TipoBuilder tipoBuilder){
54      ElementoEmpresa? sup = seleccionado is Departamento ? seleccionado : null;
55      if( tipoBuilder == TipoBuilder.completo){
56        builder = EmpleadoTiempoCompletoBuilder(sup);
57      } else {
58        builder = EmpleadoMedioTiempoBuilder(sup);
59         }
60      }
61
62    TipoBuilder getTipoBuilder(){
63      return builder is EmpleadoTiempoCompletoBuilder ? TipoBuilder.completo : TipoBuilder.parcial;
64      }
65
66    ElementoEmpresa getElemento(int index){
67      return empresa[index];
68      }
69
70    ElementoEmpresa getElementoSeleccionado(){
71      return seleccionado!;
72      }
73
74    void setElementoSeleccionado(ElementoEmpresa e){
75      if(seleccionado == e){
76        seleccionado = null;
77      } else{
78        seleccionado = e;
79         }
80
81      }
82
83    List<ElementoEmpresa>? getElementosSubdepartamento(ElementoEmpresa e){
84      if(e.getSuperior() != null){
85        return e.getElementos();
86      }else{
87        return null;
88         }
89      }
90
91  }
```

Vista



Para la vista, como se indica en la práctica, se utilizan los Widgets de Flutter.

EmpleadoWidget

Este es un StatelessWidget que crea un Text del empleado dado. Lo utiliza el Widget que está encargado de generar la lista de ElementoEmpresa para añadirlo cuando hace falta. Necesita que sea pasado un empleado, puesto que los departamentos carecen de todos esos datos, pero una alternativa

sería que se pasará un ElementoEmpresa en el que se utilizará casting para crear una variable local Empleado que utilizamos. Esta alternativa es mejor puesto que así sólo acepta el objeto que puede utilizar, permitiendo detectar errores, y deberá ser quién lo use quién se encargue de pasar el objeto correcto.

```dart
import 'package:ejercicio_grupal/Model/Empleado.dart';
import 'package:flutter/material.dart';

class EmpleadoWidget extends StatelessWidget {
  const EmpleadoWidget({super.key, required this.empleado});
  final Empleado empleado;

  @override
  Widget build(BuildContext context) {
    return (Container(
        child: Text("DNI: " +
            empleado.getDni() +
            " / Cargo: " +
            empleado.getCargo() +
            " / Contrato Actual: " +
            empleado.getTipoContrato())));  // Text, Container
  }
}
```

ListaElementosWidget

Es un Widget que se encarga de mostrar una lista de ElementoEmpresa. Requiere de dicha lista, para mostrar los elementos, y de un Director, que actúa como controlador. Algunas explicaciones:

- ListView.builder es lo que permite generar de forma automática, definiendo un objeto que se usa de base, los elementos de la lista de longitud variable. El valor de itemCount es cuántos elementos generará, por lo que se establece a la longitud de la lista. El itemBuilder utiliza un index para identificar qué elemento de la lista se está creando.
- Padding, como su nombre indica nos permite darle a su hijo espacio de relleno, y así espaciar los elementos al gusto.
- ListTile representa un elemento de la ListView. Su color de fondo variará dependiendo de si puede tener hijos o no. Esta consulta se la hacemos al director.
- Checkbox permite al usuario seleccionar un elemento y a la vez mostrar una retroalimentación de ello. Su valor se obtiene consultando al director.estaSeleccionado(ElementoEmpresa).

Cuando se pulsa la CheckBox, se llama director.setElementoSeleccionado(ElementoEmpresa) y este método es el encargado de que director recuerde el seleccionado (o lo olvide si la casilla ya estaba marcada y se quiere deseleccionar). Después de eso, utiliza un callback y esto notifica al padre del cambio. Utilizar el callback es necesario, porque si no, un elemento marcado con anterioridad no se actualizará y seguirá apareciendo como el seleccionado al usuario aunque internamente no lo sea.

- El subtítulo será otro ListaElementosWidgets si el elemento puede tener hijos (pasamos el mismo director y la lista de ElementoEmpresa que tiene el elemtento) o será EmpleadoWidget, en caso contrario.
- La lista de ElementoEmpresa de un Departamento se comporta igual que la lista original.
- El callback que se pasa por el constructor de ListaElementosWidget es necesario por la creación reiterada de ListaElementosWidget, permite avisar al padre hasta llegar a la raíz de la primera invocación, y pedir que se actualice el estado de este. Como se explicó anteriormente, sin esto no se mostraría la información de forma correcta al usuario.

```dart
import 'package:ejercicio_grupal/Model/Director.dart';
import 'package:ejercicio_grupal/Model/Empleado.dart';
import 'package:ejercicio_grupal/Widgets/EmpleadoWidget.dart';
import 'package:ejercicio_grupal/Model/ElementoEmpresa.dart';

import 'package:flutter/material.dart';

class ListaElementosWidget extends StatefulWidget {
  const ListaElementosWidget(
      {super.key,
      required this.director,
      required this.listElems,
      required this.callback});
  final Director director;
  final List<ElementoEmpresa> listElems;
  final Function callback;

  @override
  State<ListaElementosWidget> createState() => _ListaElementosWidget();
}

class _ListaElementosWidget extends State<ListaElementosWidget> {
  _ListaElementosWidget();

  callback() {
    widget.callback();
  }
```

```dart
Widget build(BuildContext context) {
  return ListView.builder(
      itemCount: widget.listElems.length,
      shrinkWrap: true,
      padding: const EdgeInsets.all(10),
      itemBuilder: (BuildContext context, int index) {
        return Padding(
          padding: const EdgeInsets.only(
            top: 5,
          ),  // EdgeInsets.only
          child: ListTile(
            tileColor:
                widget.director.puedeTenerHijos(widget.listElems[index])
                    ? (Theme.of(context).primaryColorLight).withOpacity(0.25)
                    : Theme.of(context).canvasColor.withOpacity(0.25),
            shape: RoundedRectangleBorder(
              side: const BorderSide(color: Colors.grey, width: 1),
              borderRadius: BorderRadius.circular(5),
            ),  // RoundedRectangleBorder
            leading: Checkbox(
              onChanged: (bool? value) {
                widget.director
                    .setElementoSeleccionado(widget.listElems[index]);
                widget.callback();
              },
              value: widget.director.estaSeleccionado(widget.listElems[index])
                  ? true
                  : false,
            ),  // Checkbox
            title: Text(widget.listElems[index].toString()),
            subtitle: widget.director.puedeTenerHijos(widget.listElems[index])
                ? ListaElementosWidget(
                    director: widget.director,
                    listElems: widget.listElems[index].getElementos(),
                    callback: callback,
                  )  // ListaElementosWidget
                : EmpleadoWidget(
                    empleado: widget.listElems[index] as Empleado),  // EmpleadoWidget
          ),  // ListTile
        );  // Padding
      });  // ListView.builder
}
```

main

El main llama a ListaElementosWidget y define el callback original que actualiza el estado con setState. Los hijos del main simplemente llaman al callback del padre hasta llegar aquí. Las funciones que se hacen por pulsar los botones son básicamente pedirle al director que añada un elemento, obteniendo los datos de los controladores, o que elimine un elemento. El director es el que internamente intentará eliminar el que tenga guardado como seleccionado o comprobará si debe añadir el ElementoEmpresa a su lista o a la de algún elemento de esta. Sobre los Widgets:

- Flexible permite que los TextField adapten su tamaño dependiendo de los elementos con los que compartan Row o Column.
- Extended es necesario (junto a shrinkWrap: true, en el ListView) para poder mostrar una ListView en el interior de una columna. Esto es porque el comportamiento predeterminado de ListView no se adapta correctamente a Row o Column, ya que es Scrollable.
- Para espaciar columnas y filas hay varias formas, como envolver cada elemento con Padding. En este caso utilizamos un SizezBox cuando sea necesario, ya que es más simple encontrar las separaciones a simple vista.

```dart
import 'package:ejercicio_grupal/Model/EmpleadoTiempoCompletoBuilder.dart';
import 'package:ejercicio_grupal/Model/TipoBuilder.dart';
import 'package:ejercicio_grupal/Widgets/ListaElementosWidget.dart';
import 'package:ejercicio_grupal/Model/Director.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';

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

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.indigo),
        useMaterial3: true,
      ),  // ThemeData
      home: const MyHomePage(title: 'Gestión de Empresa'),
    );  // MaterialApp
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  TextEditingController nombre = TextEditingController();
  TextEditingController dni = TextEditingController();
```

```
class _MyHomePageState extends State<MyHomePage> {
  TextEditingController nombre = TextEditingController();
  TextEditingController dni = TextEditingController();
  TextEditingController cargo = TextEditingController();
  TextEditingController tipo_contrato = TextEditingController();
  TextEditingController nombre_dep = new TextEditingController();
  final Director director = Director(EmpleadoTiempoCompletoBuilder(null));

  @override
  void initState() {
    super.initState();
  }

  @override
  void dispose() {
    nombre.dispose();
    dni.dispose();
    cargo.dispose();
    tipo_contrato.dispose();
    nombre_dep.dispose();
    super.dispose();
  }

  callback() {
    setState(() {});
  }

  void removeElement() {
    setState(() {
      director.remove();
    });
  }

  void addEmpleado() {
    setState(() {
      director.addEmpleado(
          nombre.text, dni.text, cargo.text, director.seleccionado);
    });
  }
```

```
void addDepartamento() {
  setState(() {
    director.addDepartamento(nombre_dep.text, director.seleccionado);
  });
}

@override
Widget build(BuildContext context) {
```

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Theme.of(context).colorScheme.primary,
      title: Text(widget.title,
          style: TextStyle(
            color: Theme.of(context).canvasColor,
          )),  // TextStyle, Text
    ),  // AppBar
    body: Column(
      children: [
      Container(
        color: Theme.of(context).colorScheme.surface.withOpacity(1),
        padding: const EdgeInsets.all(10),
        child: Column(
          children: [
          Row(
            children: [
            Flexible(
              child: TextField(
                controller: nombre_dep,
                obscureText: false,
                decoration: const InputDecoration(
                  border: OutlineInputBorder(),
                  labelText: 'Nombre del Departamento',
                ),  // InputDecoration
              ),  // TextField
            ),  // Flexible
            const SizedBox(width: 10),
            ElevatedButton.icon(
              onPressed: addDepartamento,
              label: const Text('Depart.'),
              icon: const Icon(Icons.add),
            ),  // ElevatedButton.icon
          ],
        ),  // Row
        const SizedBox(height: 10),
        Row(
          children: [
```

```
const SizedBox(height: 10),
Row(
  children: [
Flexible(
  child: TextField(
    controller: nombre,
    obscureText: false,
    decoration: const InputDecoration(
      border: OutlineInputBorder(),
      labelText: 'Nombre del Empleado',
    ),  // InputDecoration
  ),  // TextField
),  // Flexible
const SizedBox(width: 10),
Flexible(
  child: TextField(
    controller: dni,
    obscureText: false,
    decoration: const InputDecoration(
      border: OutlineInputBorder(),
      labelText: 'DNI',
    ),  // InputDecoration
  ),  // TextField
),  // Flexible
  ],
),  // Row
const SizedBox(height: 10),
Row(
  children: [
Flexible(
  child: TextField(
    controller: cargo,
    obscureText: false,
    decoration: const InputDecoration(
      border: OutlineInputBorder(),
      labelText: 'Cargo',
    ),  // InputDecoration
  ),  // TextField
),  // Flexible
const SizedBox(width: 10),
DropdownMenu<TipoBuilder>(
  requestFocusOnTap: true,
```

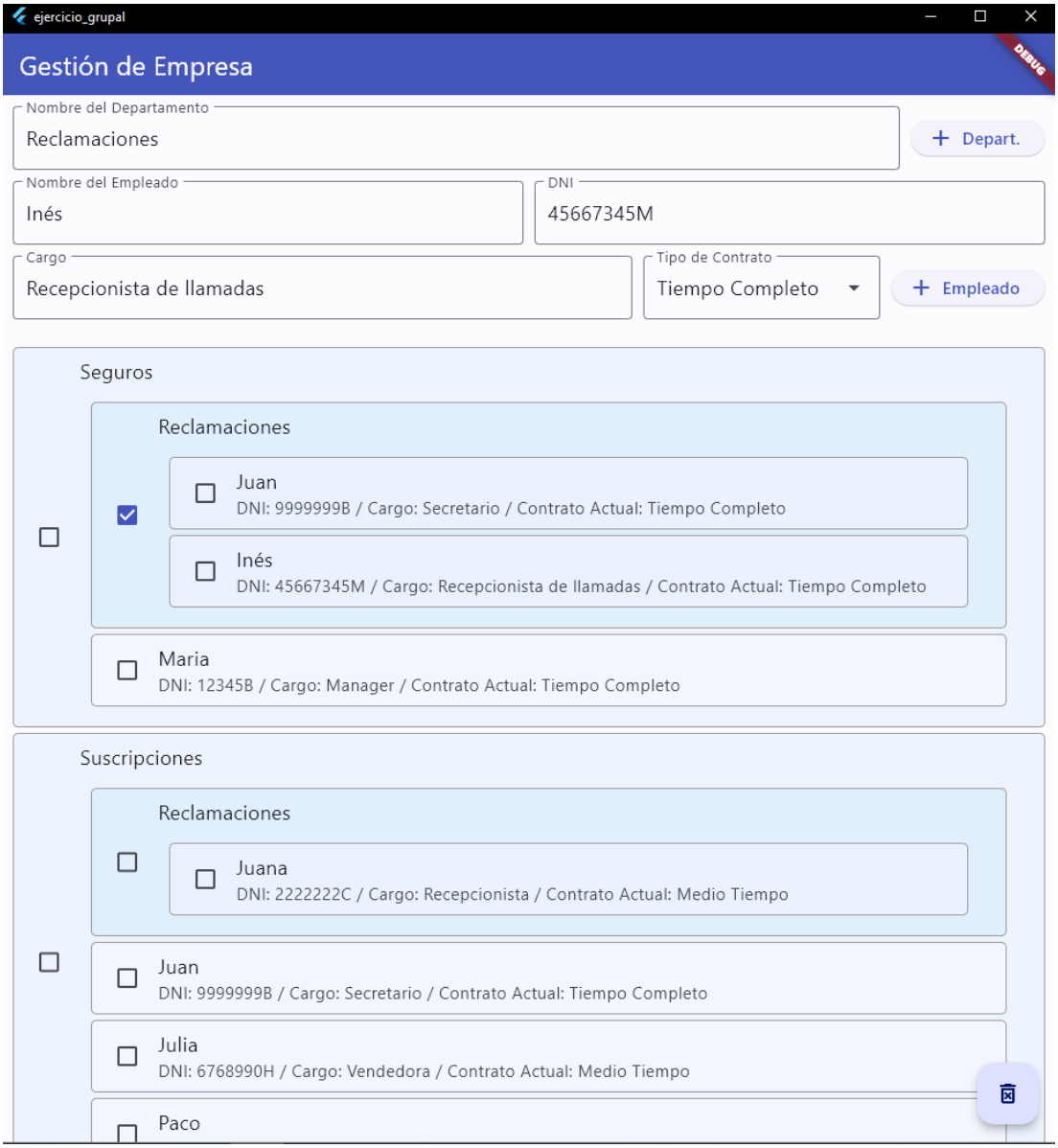
```
            DropdownMenu<TipoBuilder>(
              requestFocusOnTap: true,
              initialSelection: TipoBuilder.completo,
              label: const Text('Tipo de Contrato'),
              onSelected: (TipoBuilder? op) {
                setState(() {
                  if (op != null &&
                      op is TipoBuilder &&
                      op != director.getTipoBuilder()) {
                    director.changeBuilder(op);
                  }
                });
              },
              dropdownMenuEntries: TipoBuilder.values
                  .map<DropdownMenuEntry<TipoBuilder>>(
                      (TipoBuilder op) {
                return DropdownMenuEntry<TipoBuilder>(
                  value: op,
                  label: op.label,
                );  // DropdownMenuEntry
              }).toList(),
            ),  // DropdownMenu
            const SizedBox(width: 10),
            ElevatedButton.icon(
              onPressed: addEmpleado,
              label: const Text("Empleado"),
              icon: const Icon(Icons.add),
            ),  // ElevatedButton.icon
          ],
        ),  // Row
      ],
    ),  // Column
  ),  // Container
  Expanded(
    child: ListaElementosWidget(
      director: director,
      listElems: director.getEmpresa(),
      callback: callback,
    ))  // ListaElementosWidget, Expanded
  ],
),  // Column
floatingActionButton: FloatingActionButton(
```

```
        Expanded(
          child: ListaElementosWidget(
            director: director,
            listElems: director.getEmpresa(),
            callback: callback,
          ))  // ListaElementosWidget, Expanded
        ],
      ),  // Column
    floatingActionButton: FloatingActionButton(
      onPressed: removeElement,
      child: const Icon(Icons.delete_forever_outlined),
    ),  // FloatingActionButton
  );  // Scaffold
 }
}
```

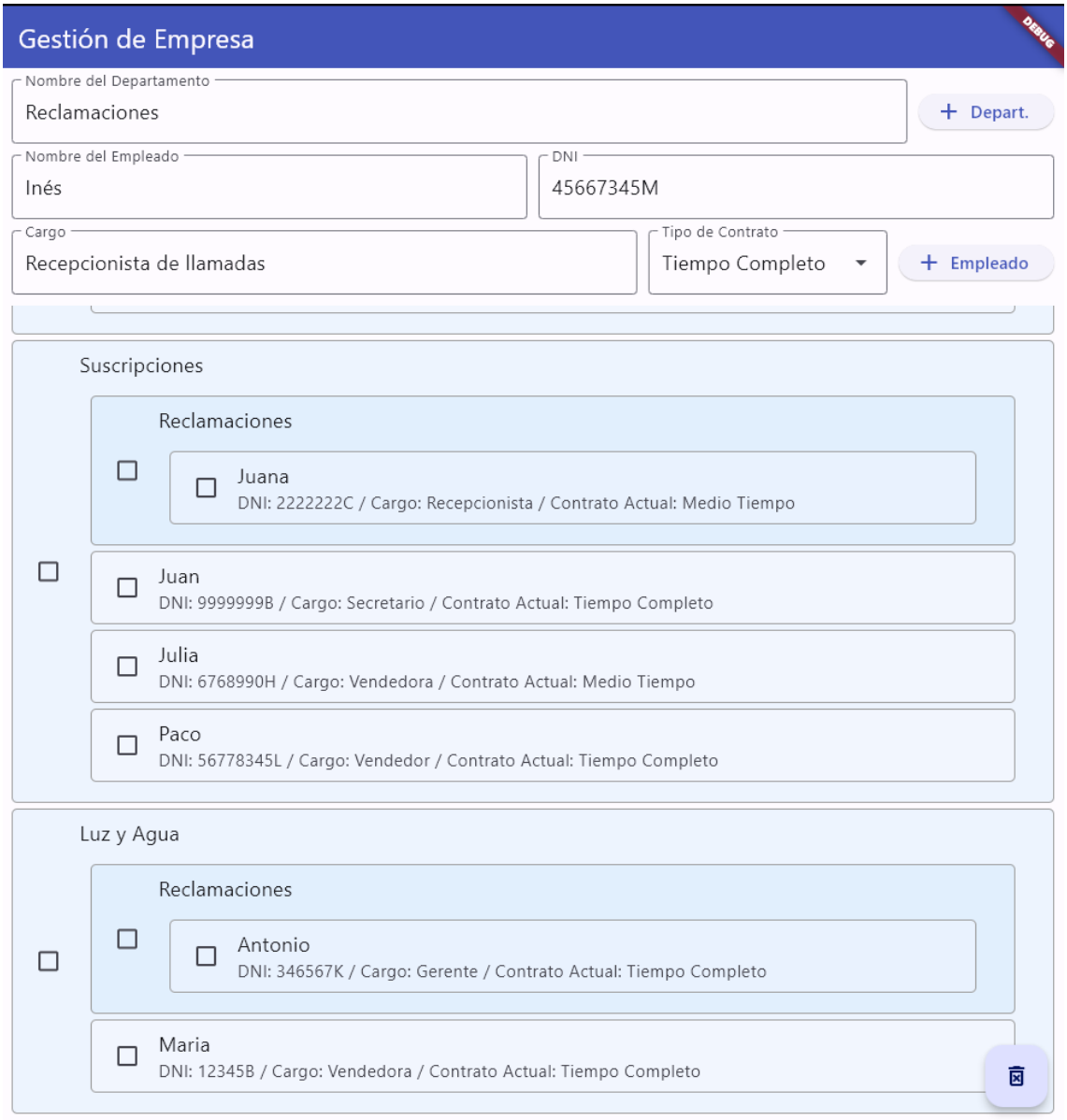
Funcionamiento

Ejemplo del funcionamiento.

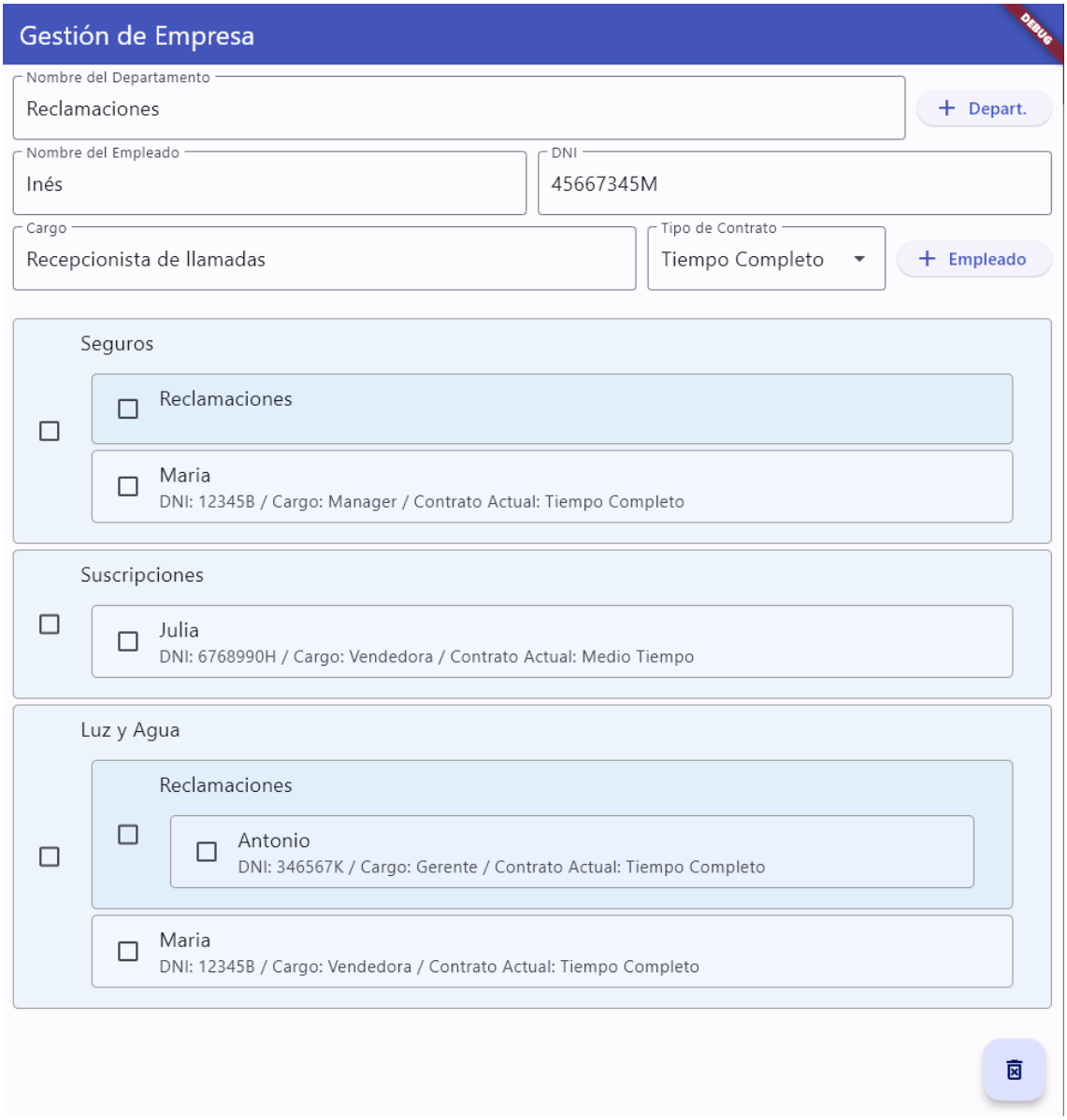
Situación: Empresa de telemarketing, cuyos empleados trabajan desde sus propias casas. Tienen un departamento para cada tipo de productos y una sección de reclamaciones en cada una. Un empleado puede pertenecer a varias divisiones.



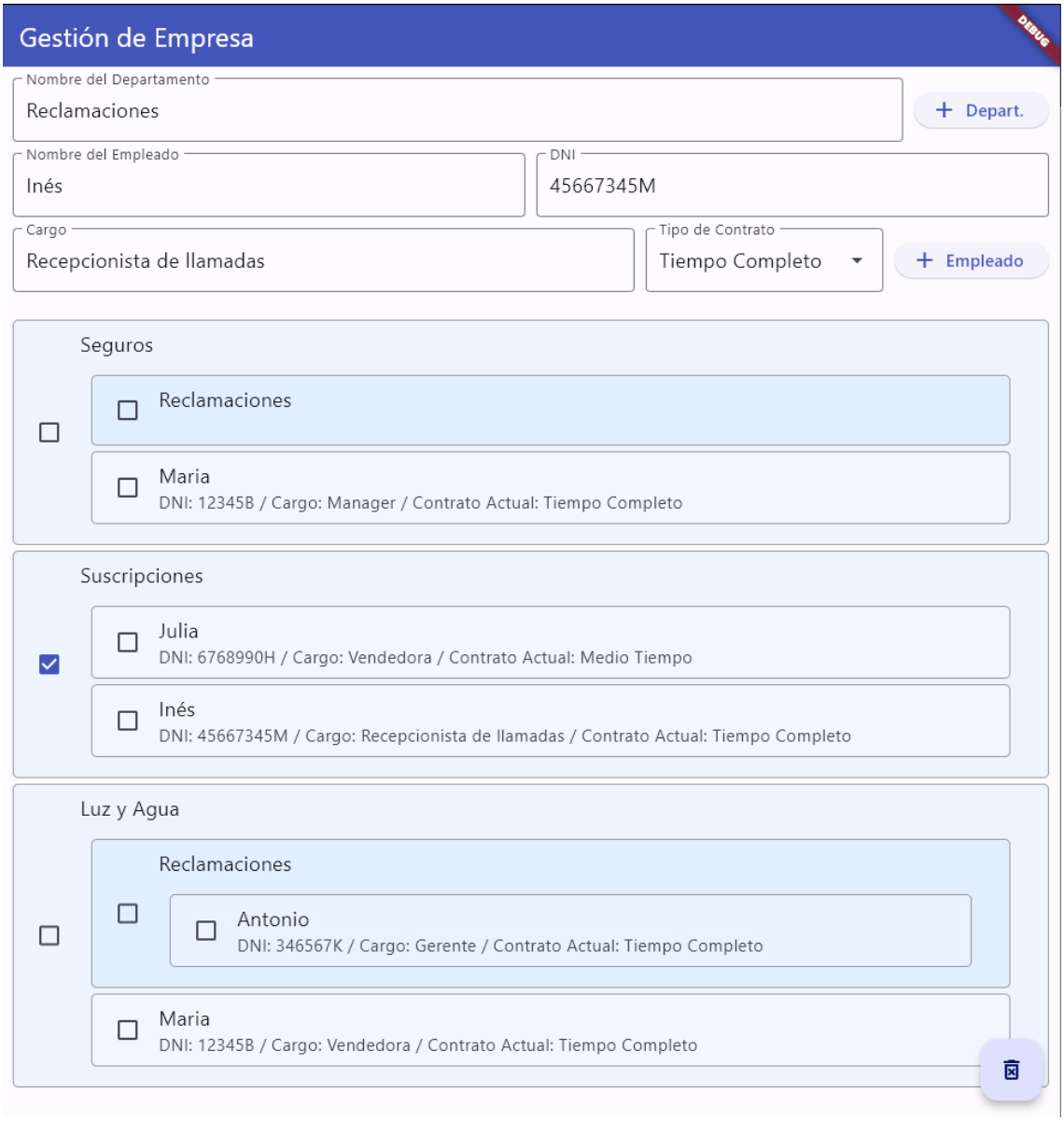
Se puede hacer Scroll



Eliminamos algunos elementos (se puede eliminar un padre y sus hijos son eliminados)


Gestión de Empresa
DEBUG
Nombre del Departamento
Reclamaciones
Depart.
Nombre del Empleado
Inés
DNI
45667345M
Cargo
Recepcionista de llamadas
Tipo de Contrato
Tiempo Completo
Empleado
Seguros
Reclamaciones
Maria
DNI: 12345B / Cargo: Manager / Contrato Actual: Tiempo Completo
Suscripciones
Julia
DNI: 6768990H / Cargo: Vendedora / Contrato Actual: Medio Tiempo
Luz y Agua
Reclamaciones
Antonio
DNI: 346567K / Cargo: Gerente / Contrato Actual: Tiempo Completo
Maria
DNI: 12345B / Cargo: Vendedora / Contrato Actual: Tiempo Completo

Añadimos



Conclusión

Para esta práctica se ha adaptado el código original y se ha rediseñado para una aplicación que permite al usuario modificar los datos sin tocar el código. El mayor cambio perfectivo que añade una nueva funcionalidad es el hecho de que el director recuerde que ha sido seleccionado. Esto es lo que permite los cambios desde la UI. Puesto que el modelo cuenta con funciones para editar los valores, esa sería la mejora principal que recibiría la aplicación en una supuesta expansión o

quizá poder elegir qué restricciones darle al director para la creación de elementos, puesto que ahora mismo es bastante libre para poder adaptarse a muchas estructuras de organización.