

Applicació exemple: RecyclerView, bases de dades i patró de disseny

Projecte Integrat de Software (PIS)

Universitat de Barcelona

victor.campello@ub.edu

carlos.martinisla@ub.edu

1-3 de Març de 2022

Overview

- 1 Patrons de disseny
 - Model Vista Controlador (MVC)
 - Model-Vista-Model de Vista (MVVM)
- 2 Exemple: calculadora IMC amb MVVM
- 3 Exemple amb base de dades
- 4 RecyclerView
- 5 Base de dades local

Model Vista Controlador (MVC)

- 1 **Model:** conté la informació amb la qual el sistema treballa, proporcionant-la a la vista per a que aquesta la mostri, i permetent realitzar canvis en la vista des del controlador.
- 2 **Controlador:** respon a accions de l'usuari, modificant el model quan és necessari. A més a més, es comunica amb la vista per a que s'actualitzi amb els últims canvis al model.
- 3 **Vista:** presenta a l'usuari la informació del model.

Model Vista Controlador (MVC)

A Androd hi ha un component específic per fer de controlador. L'*Activity* o el *Fragment* són els responsables de comunicar-se amb la **vista** i d'implementar el **model**.

- ▶ Aquests components són específics d'Android.
- ▶ Per tant, implementar la lògica a l'*Activity* fa que la nostra aplicació sigui dependent de l'ecosistema Android.
- ▶ A més a més, es viola el principi de responsabilitat única.
- ▶ Això provoca que els test unitaris siguin complicats (s'ha de executar tota l'activitat per testejar la lògica).

Model-Vista-Model de Vista (MVVM)

- ① **Model:** conté la informació amb la qual el sistema treballa, similar al MVC.
- ② **Model de Vista:** és una abstracció de la vista i conté tota la lògica de presentació.
- ③ **Vista:** presenta a l'usuari la informació del model. **Respon a canvis de configuració.**

Model-Vista-Model de Vista (MVVM)

Per què MVVM?

- 1 **Compatibilitat:** L'equip d'Android va introduir diferents classes per a poder implementar MVVM fàcilment.
- 2 **Resposta:** La lògica de presentació sobreviu a canvis de configuració i s'integra en el cicle de vida d'Android. Per exemple, el ViewModel farà que les dades de l'Activity en pantalla sobrevisquin al fer una rotació del dispositiu. L'activitat és destruïda i creada de nou, però el ViewModel la proveirà de nou amb la informació necessària.
- 3 **Testing, manteniment i escalabilitat:** A l'encapsular els diferents components, el testeig unitari se simplifica extremadament.

Model-Vista-Model de Vista (MVVM)

Android disposa dels següents **models** per proveir dades:

- 1 **Bases de dades:** Com Firebase o SQL.
- 2 **Shared Preferences:** Funcionalitat simple d'Android que ens permet serialitzar dades de maner local.
- 3 **Serveis Web:** Obtenció d'informació mitjançant APIs disponibles amb una connexió a internet. Per exemple, un video de YouTube o informació climatològica.

Model-Vista-Model de Vista (MVVM)

Mentre les **vistes** són els elements que presenten els ViewGroups o layouts i els seus Views o widgets. Aquestes són

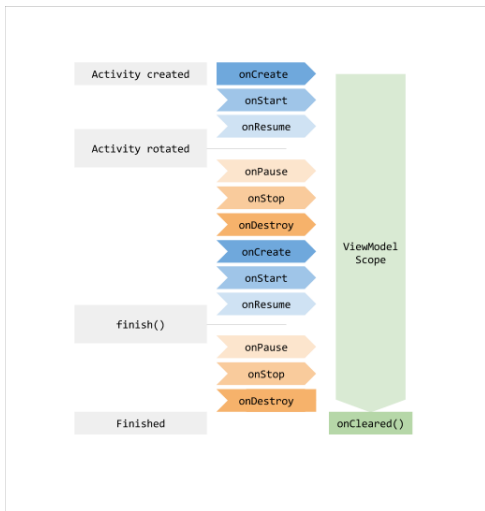
- ① **Activity**
- ② **Fragment**

Als dos tipus de vista hem d'aconseguir uns elements el més senzills possible i que aquests estiguin totalment desacoblats de la lògica. És a dir, aquests només s'han d'encarregar d'instanciar els Views amb *getViewByld* i configurar la seva resposta amb events com el *OnClick* mitjançant observacions al ViewModel.

- ▶ El sistema Android proporciona una classe, el *ViewModel*, que és el responsable d'emmagatzemar i administrar les dades relacionades amb la UI.
- ▶ Durant els canvis de configuració, els objectes *ViewModel* es mantenen de forma automàtica, de manera que les dades que contenen estan disponibles de forma immediata per a la següent instància d'una activitat o fragment.
 - Per exemple, si vols mostrar una llista d'usuaris a la teva app, hauràs d'assignar la responsabilitat d'adquirir i mantenir una llista d'usuaris a un *ViewModel*, en comptes de a una activitat o fragment.

ViewModel

Aquest és el cicle de vida del ViewModel durant un canvi de rotació i una finalització d'una app.



Com hem vist abans, *LiveData*, és una classe de dades que es poden “observar”. Està optimitzat segons els cicles de vida de la app, el que vol dir que respecta el cicle de vida d'altres components com activitats, fragments o serveis. Això garanteix que LiveData només actualitzarà els observadors dels diferents components quan aquests estan actius, fent que l'aplicació sigui més eficient.

- 1 Garanteix que la UI coincideixi amb l'estat de les dades.
- 2 Evita fugues de memòria.
- 3 No s'ha de fer un control manual del cicle de vida.
- 4 Canvis de configuració apropiats: Una activitat o fragment que es crea de nou per un canvi de configuració, ,com la rotació del dispositiu, rep immediatament les dades disponibles més recents.

Exemple de LiveData

Instància de LiveData a un ViewModel.

```
public class NameViewModel extends ViewModel {  
  
    // Create a LiveData with a String  
    private MutableLiveData<String> currentName;  
  
    public MutableLiveData<String> getCurrentName() {  
        if (currentName == null) {  
            currentName = new MutableLiveData<String>();  
        }  
        return currentName;  
    }  
  
    // Rest of the ViewModel...  
}
```

Exemple de LiveData

Creació d'un observer a l'activitat de la nostra aplicació.

```
public class NameActivity extends AppCompatActivity {  
  
    private NameViewModel model;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Other code to setup the activity...  
  
        // Get the ViewModel.  
        model = new ViewModelProvider(this).get(NameViewModel.class);  
  
        // Create the observer which updates the UI.  
        final Observer<String> nameObserver = new Observer<String>() {  
            @Override  
            public void onChanged(@Nullable final String newName) {  
                // Update the UI, in this case, a TextView.  
                nameTextView.setText(newName);  
            }  
        };  
  
        // Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.  
        model.getCurrentName().observe(this, nameObserver);  
    }  
}
```

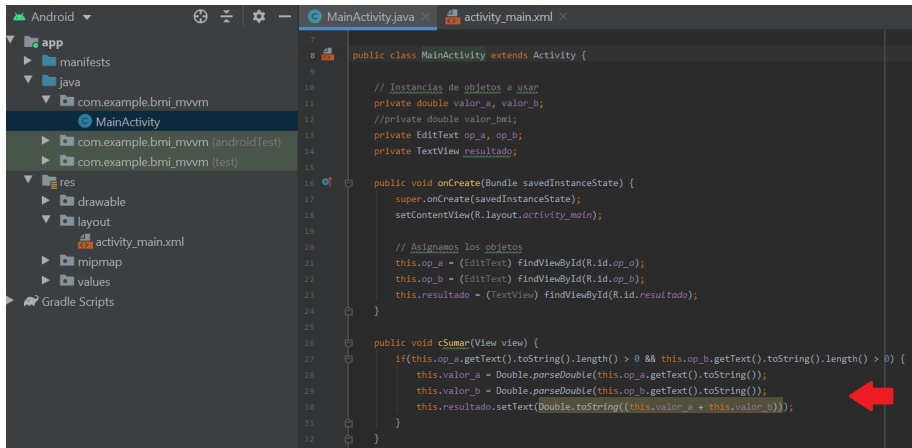
Exemple de LiveData

Finalment, al canviar la variable *currentName* del ViewModel, l'observer comunica el canvi i s'actualitza la vista associada.

```
button.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        String anotherName = "John Doe";  
        model.getCurrentName().setValue(anotherName);  
    }  
});
```

Exemple: calculadora IMC

A una aplicació sense patró de disseny, la lògica està implementada a la MainActivity.



```
7
8 public class MainActivity extends Activity {
9
10     // Instancias de objetos a usar
11     private double valor_a, valor_b;
12     //private double valor_bmi;
13     private EditText op_a, op_b;
14     private TextView resultado;
15
16     public void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_main);
19
20         // Asignamos los objetos
21         this.op_a = (EditText) findViewById(R.id.op_a);
22         this.op_b = (EditText) findViewById(R.id.op_b);
23         this.resultado = (TextView) findViewById(R.id.resultado);
24     }
25
26     public void cSumar(View view) {
27         if(this.op_a.getText().toString().length() > 0 && this.op_b.getText().toString().length() > 0) {
28             this.valor_a = Double.parseDouble(this.op_a.getText().toString());
29             this.valor_b = Double.parseDouble(this.op_b.getText().toString());
30             this.resultado.setText(Double.toString((this.valor_a + this.valor_b)));
31         }
32     }
```

Exemple: calculadora IMC

Per a implementar MVVM, el model serà la classe proveidora. Recordeu que els models poden proveir informació de diferents fonts de dades. Si volem migrar el use case del càlcul de l'índex de massa corporal de la Activity a un model, ho farem de la següent manera (a dalt, implementació en Activity, a sota, en un model proveïdor).

```
public void BMI(View view) {
    if (this.op_a.getText().toString().length() > 0 && this.op_b.getText().toString().length()
        this.valor_a = Double.parseDouble(this.op_a.getText().toString());
        this.valor_b = Double.parseDouble(this.op_b.getText().toString());
        this.valor_bmi = ((this.valor_a * this.valor_a) / this.valor_b);
        if (this.valor_bmi < 25) {
            this.resultado.setText("Peso Ideal");
        }
        if (this.valor_bmi >= 25) {
            this.resultado.setText("Sobrepeso");
        }
    }
}
```

```
package com.example.bmi_mvvm;

public class BMIDummyModel {

    public static String BMI(String op1, String op2) {
        double valor_a = Double.parseDouble(op1);
        double valor_b = Double.parseDouble(op2);
        double valor_bmi = ((valor_a * valor_a) / valor_b);
        if (valor_bmi < 25) {
            return "Peso Ideal";
        }
        else return "Sobrepeso";
    }
}
```


Exemple: calculadora IMC

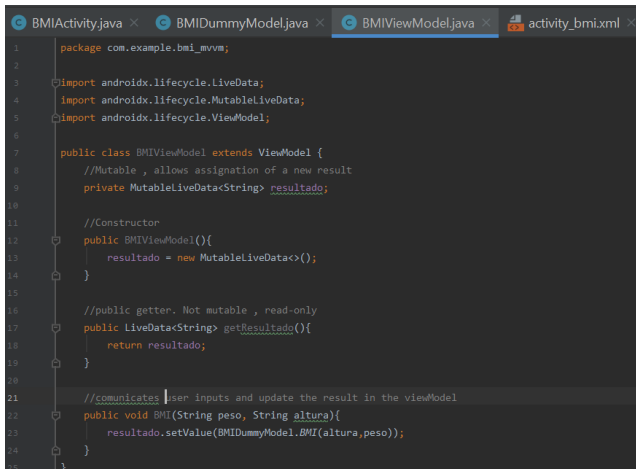
Observeu els primers beneficis:

- ❶ Lògica separada, fàcil per fer tests unitaris.
- ❷ **Arguments Java** en comptes de Views, que depenen d'Android.
- ❸ Activity simplificada al màxim. Només s'encarregarà dels components relatius a la vista i d'observar el ViewModel.

De totes maneres, hauríem d'instanciar el mdoel BMIDummyModel a l'Activity per poder accedir al proveïdor de dades, que en aquest cas, és el mètode BMI de BMIDummyModel. Això violaria el **principi de responsabilitat única**.

Exemple: calculadora IMC

Per tant, implementarem un intermediari observable ViewModel que s'encarregui de la comunicació entre els dos objectes, i a més a més, sigui conscient del cicle de vida de l'ecosistema Android.



```
1 package com.example.bmi_mvvm;
2
3 import androidx.lifecycle.LiveData;
4 import androidx.lifecycle.MutableLiveData;
5 import androidx.lifecycle.ViewModel;
6
7 public class BMIViewModel extends ViewModel {
8     //Mutable , allows assignation of a new result
9     private MutableLiveData<String> resultado;
10
11     //Constructor
12     public BMIViewModel(){
13         resultado = new MutableLiveData<>();
14     }
15
16     //public getter. Not mutable , read-only
17     public LiveData<String> getResultado(){
18         return resultado;
19     }
20
21     //communicates user inputs and update the result in the viewModel
22     public void BMI(String peso, String altura){
23         resultado.setValue(BMIDummyModel.BMI(altura,peso));
24     }
25 }
```

Exemple: calculadora IMC

Anem a afegir el viewModel a la nostra Activity.

```
public class BMIActivity extends AppCompatActivity {

    private EditText op_a, op_b;
    private TextView resultado;
    private Button bmiButton;
    //declare our viewModel
    private BMIViewModel viewModel;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_bmi);
        initView();
    }

    private void initView(){
        // Instance the viewModel (update gradle dependencies if required)
        viewModel = new ViewModelProvider(    owner: this).get(BMIViewModel.class);

        this.op_a = (EditText) findViewById(R.id.op_a);
        this.op_b = (EditText) findViewById(R.id.op_b);
        this.bmiButton = (Button) findViewById(R.id.bmi_button);
        this.resultado = (TextView) findViewById(R.id.resultado);

        bmiButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                viewModel.BMI(op_a.getText().toString(),op_b.getText().toString());
            }
        });
    }
    //Observer related stuff...
}
```

Exemple: calculadora IMC

Per acabar, a la nostra Activity crearem un observador per al nostre MutableLiveData i el subscriu als seus canvis.

```
//Observer related stuff...
//Observe changes in LiveData
final Observer<String> observer = new Observer<String>() {
    @Override
    public void onChanged(String s) {
        resultado.setText(s);
    }
};

//Subscribe the activity to the observable
viewModel.getResultado().observe(owner: this,observer);
}
```

Exemple: calculadora IMC

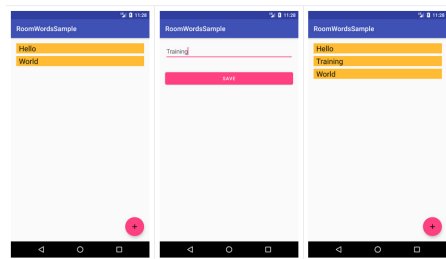
Implementació MVVM a la calculadora de IMC

- 1 La lògica està totalment desacoblada de la vista.
- 2 Aquesta lògica és extremadament fàcil de testejar.
- 3 El ViewModel sobreviu a tots els canvis de configuració i proveeix a la vista amb les dades de manera autònoma.
- 4 La vista (Activity) només s'encarrega de rebre la informació de l'intermediari ViewModel i d'actualitzar els seus elements, reduint així la seva complexitat.

Aplicació de partida

En aquest laboratori utilitzarem l'aplicació de la documentació oficial que podeu trobar al següent tutorial:

developer.android.com/codelabs/android-room-with-a-view



Podeu descarregar-la a la vostra carpeta amb la comanda:

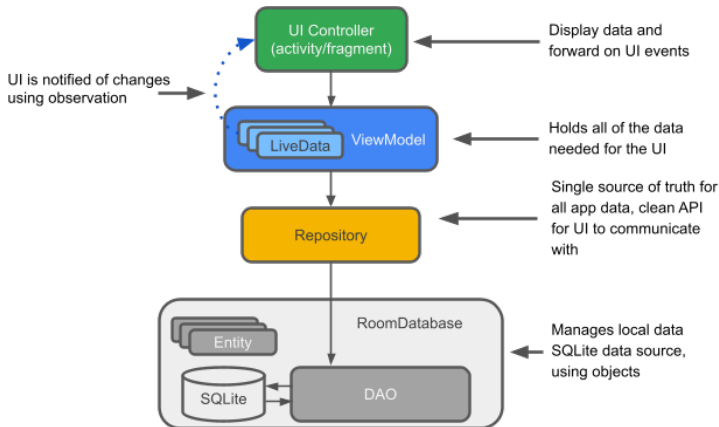
`git clone https://github.com/googlecodelabs/android-room-with-a-view.git`

Amb aquest exemple treballarem els següents continguts:

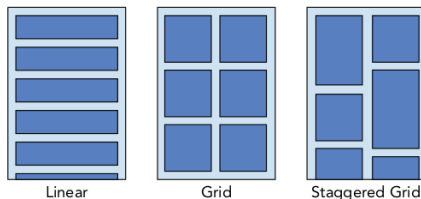
- ▶ L'element RecyclerView per mostrar llistes de Views.
- ▶ Utilització d'un botó flotant de la llibreria Material.
- ▶ Un exemple de navegació entre activitats.
- ▶ Interacció amb una base de dades local i una remota.
- ▶ Implementació del patró de disseny Model-View-ViewModel (MVVM).

Estructura de l'aplicació

L'aplicació segueix el següent esquema:



La propietat més interessant d'aquest *View* és la seva capacitat per gestionar gran quantitat d'elements de manera òptima, ja que recicla un número finit d'elements i els instància només quan s'han de visualitzar per pantalla.



Source: Professional Android. Meier & Lake.

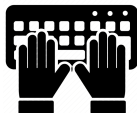
El **RecyclerView** consta de diversos components:

- ▶ El *RecyclerView* és el *ViewGroup*, el pare que conté tota la resta d'elements.
- ▶ *RecyclerView.ViewHolder* és el contenidor de Views. Gestiona els elements de la UI.
- ▶ *RecyclerView.Adapter* és el vincle entre els elements de la UI i la funcionalitat.
- ▶ *LayoutManager* és el que administra la disposició dels elements al *RecyclerView*.

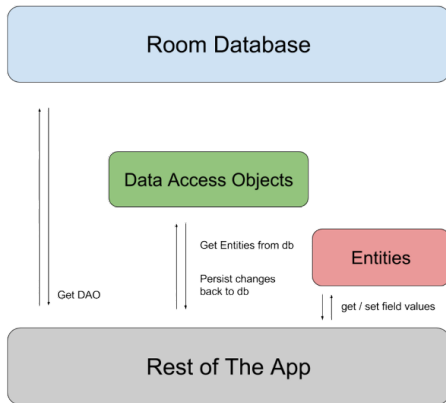
A la pràctica, amb components per defecte i extenent un adaptador (*RecyclerView.Adapter*) podrem fer la majoria d'exemples.

Exercici 1

Canvieu el LayoutManager del RecyclerView i el disseny dels Views que contenen paraules. Com a idea, podeu canviar el color de fons del text de forma aleatòria.



Aquest exemple fa servir una base de dades local SQLite amb la que es comunica mitjançant la llibreria *Room*.



A l'exemple, tenim un DAO *WordDao* i una entitat *Word* definits per a poder treballar amb la llibreria *Room*.

```
@Dao
public interface WordDao {

    // allowing the insert of the same word multiple times by passing a
    // conflict resolution strategy
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    void insert(Word word);

    @Query("DELETE FROM word_table")
    void deleteAll();

    @Query("SELECT * FROM word_table ORDER BY word ASC")
    List<Word> getAlphabetizedWords();
}
```

```
@Entity(tableName = "word_table")
public class Word {

    @PrimaryKey
    @NonNull
    @ColumnInfo(name = "word")
    private String mWord;

    public Word(@NonNull String word) {this.mWord = word;}

    public String getWord(){return this.mWord;}
}
```

Fixeu-vos que són classes usuals de Java amb annotations especials per a *Room*.

Per a visualitzar la base de dades podem fer servir **View > Tool Windows > App Inspection** mentre l'aplicació està executant-se al nostre dispositiu.

Teniu més detalls a developer.android.com/studio/inspect/database

- ▶ Com veiem a l'estructura del projecte, la classe repositori s'encarregarà de comunicar-se directament amb el DAO per operar amb les dades.
- ▶ El *ViewModel* demanarà les dades al repositori i farà servir un tipus de dades, **LiveData**, especialment adient per a aquest patró de disseny de l'aplicació.
 - El valor afegit d'aquesta classe és que permet fer accions predefinides quan les dades s'actualitzin mitjançant el que s'anomenen *Observers*.

```
// Add an observer on the LiveData returned by getAlphabetizedWords.  
// The onChanged() method fires when the observed data changes and the activity is  
// in the foreground.  
mWordViewModel.getAllWords().observe(owner: this, words -> {  
    // Update the cached copy of the words in the adapter.  
    adapter.submitList(words);  
});
```

Exercici 2

Guardau a la base de dades la data de creació del text i un id únic. Afegiu una icona per eliminar un text de la llista i de la base de dades. Haureu d'implementar el mètode al DAO per eliminar un únic element.

