

Programação Para Dispositivos Móveis I

VIEW MODEL
LIVE DATA

2024/_25 CTeSP – Desenvolvimento para a Web e Dispositivos Móveis

Ricardo Barbosa , rmb@estg.ipp.pt

Carlos Aldeias, cfpa@estg.ipp.pt

Adaptação do conteúdo dos slides de João Ramos jrmr@estg.ipp.pt e Fábio Silva fas@estg.ipp.pt

Índice

- ViewModel;
- LiveData;
- LiveData in Android;
- Leitura Adicional.

ViewModel

Um componente da biblioteca de Architecture Components de Android responsável pela preparação de dados para a interface do utilizador:

- Interage bem com os ciclos de vida dos componentes de Android;
- Permite encapsular modelos de dados de forma a poderem ser usados em padrões de software como o Observable;
- Capaz de reagir a alterações do estado dos dados no modelos de dados e acionar callbacks.

ViewModel

Motivação

- O Android gere os ciclos de vida de componentes da UI como activities e fragments. Pode decidir destruir ou recriar cada componente em resposta a determinadas ações do utilizador ou eventos do dispositivo;
- Se o sistema destruir ou recriar um controlador de UI, qualquer tipo de dado guardado no mesmo será perdido. Por exemplo, se a nossa aplicação incluir uma lista de utilizadores numa das suas Activities, quando a Activity é recreada devido a uma alteração de configuração (ex. rodar ecrã), terá de voltar a construir a lista de utilizadores;
- Os dados subjacentes a cada componente podem ser recriados através dos respetivos ciclos de vida do componente onde foram inicializados.

ViewModel

Exemplo

Recordamos a aplicação “Hello Toast” presente na Ficha Prática nº 1.

- Uma das funcionalidades da aplicação era a presença de um contador. Cada click que o utilizador der no botão COUNT, o valor do contador incrementa em uma unidade.

0

ViewModel

Exemplo

Recordamos a aplicação “Hello Toast” presente na Ficha Prática nº 1.

- Uma das funcionalidades da aplicação era a presença de um contador. Cada click que o utilizador der no botão COUNT, o valor do contador incrementa em uma unidade.

0



COUNT

ViewModel

Exemplo

O que acontece se o utilizador decidir rodar o ecrã?

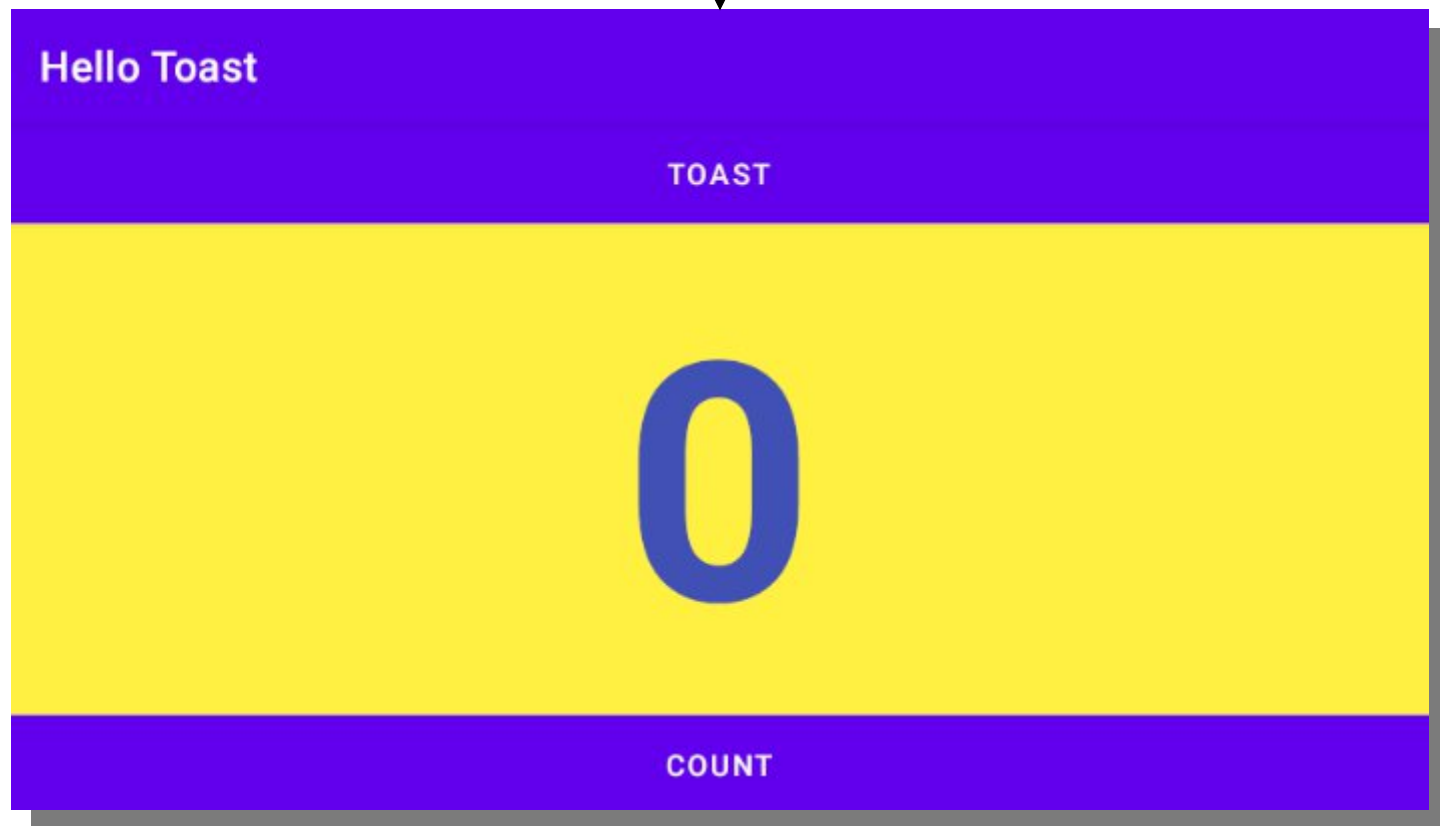
- O comportamento esperado é que a aplicação nos mostre o valor atual do contador e que nos permita incrementá-lo;

1

ViewModel

Exemplo

utilizador rodou ecrã



Hello Toast

TOAST

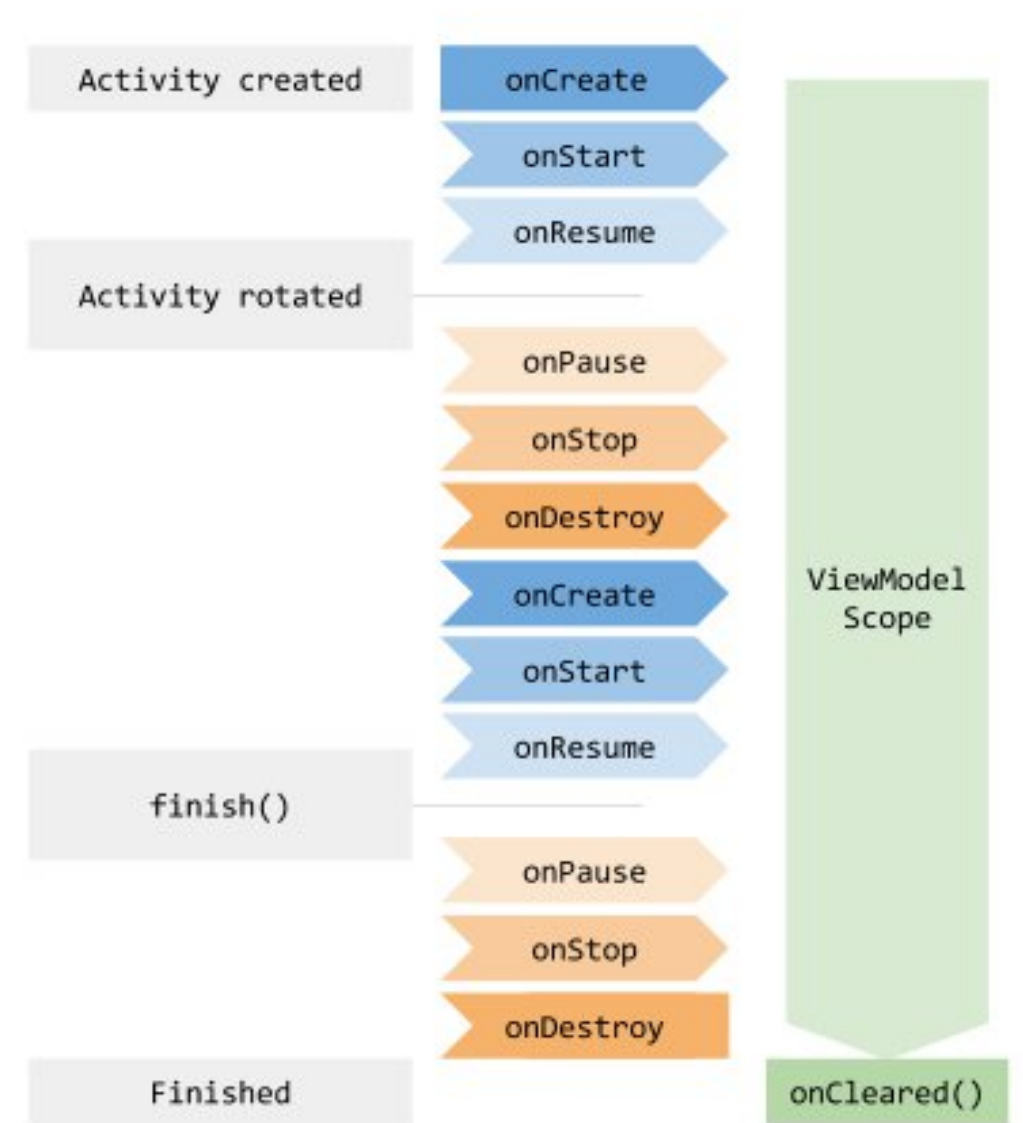
1

COUNT

ViewModel Scope

O scope de um ViewModel pode estar associado a uma activity;

Implica que pode ser utilizado por outros componentes que se enquadram no ciclo de vida da activity como por exemplo fragments.



ViewModel

Configuração [build.gradle (Project)]

```
ext{  
    lifecycle_version = "2.8.7"  
}
```

ViewModel

Configuração [build.gradle (Module)]

```
dependencies {  
  
    // ViewModel  
    implementation "androidx.lifecycle:lifecycle-viewmodel:$lifecycle_version"  
  
}
```

ViewModel

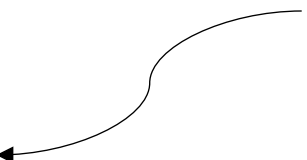
Criação [CounterViewModel.java]

```
public class CounterViewModel extends ViewModel {  
  
    public int counter = 0;  
  
    public void count(){  
        counter++;  
    }  
  
    @Override  
    protected void onCleared() {  
        super.onCleared();  
    }  
}
```

Extende ViewModel



Procedimento é chamado quando a Activity termina, e é utilizado para libertar recursos



ViewModel

Adicionar o ViewModel [MainActivity.java]

```
private CounterViewModel counterViewModel;
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    (...)
```

```
    counterViewModel = new ViewModelProvider(this).get(CounterViewModel.class);
```

```
    updateCounter(counterViewModel.counter);
```

```
}
```

```
@Override
```

```
public void onClick(View v) {
```

```
    switch (v.getId()){
```

```
        case R.id.btn_count:
```

```
            counterViewModel.count();
```

```
            updateCounter(counterViewModel.counter);
```

```
            break;
```

```
        }
```

```
    }
```

```
private void updateCounter(int count){
```

```
    txtCount.setText(String.valueOf(count));
```

```
}
```

Definição do ViewModel

counter inicialmente tem o
valor de zero

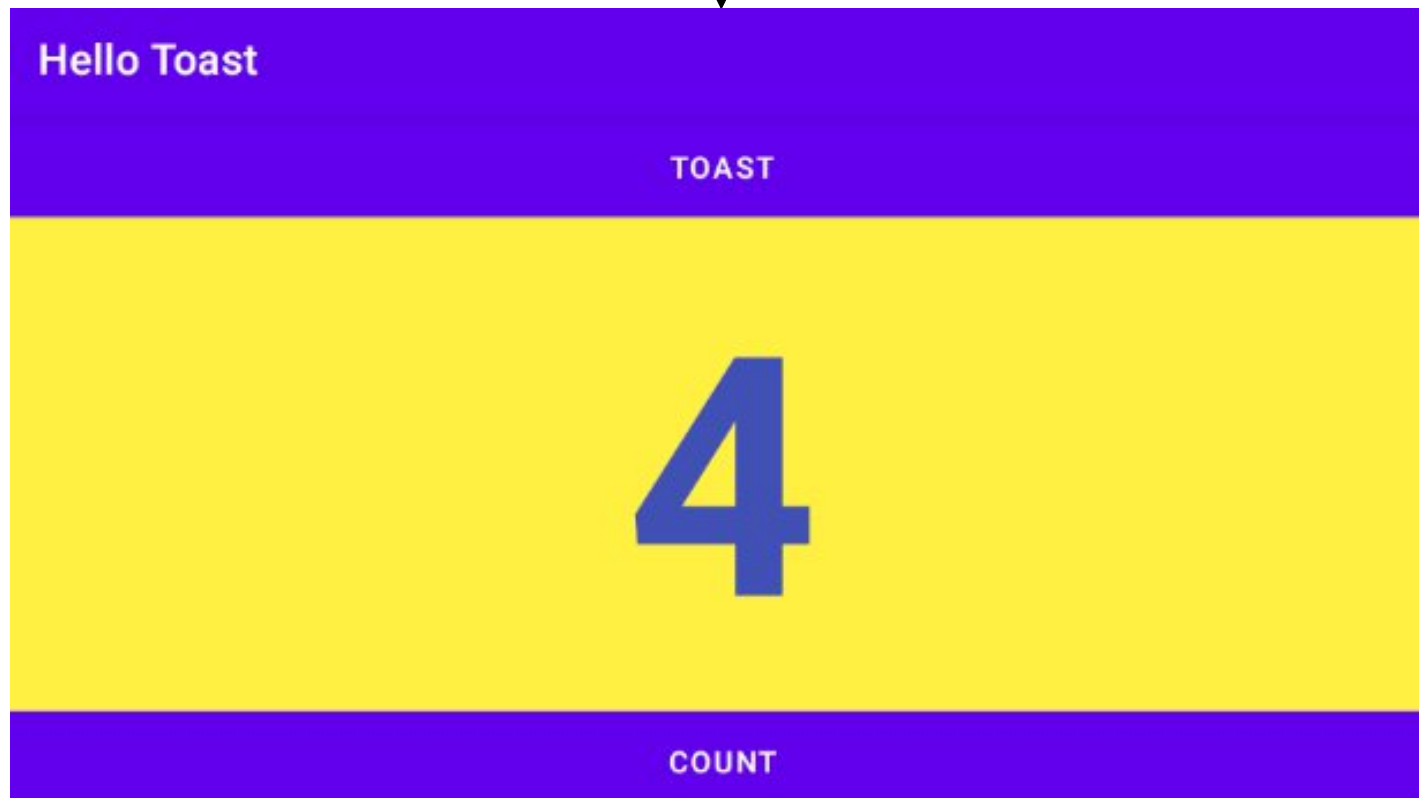
Mudar o valor da TextView
para o valor da variável
counter

Chamada da função
responsável por
incrementar o contador

ViewModel

Resultado

utilizador rodou ecrã



Hello Toast

TOAST

4

COUNT

LiveData

Em Android LiveData refere-se a um objeto dentro de um observable data holder class (ViewModel).

- Permite a atualização de componentes em tempo real usando o padrão de software Observable, de forma parecida com publish/subscribe;
- É lifecycle-aware, ou seja, respeita o do ciclo de vida de componentes da aplicação

LiveData

Vantagens

Entre as vantagens desta abordagem encontramos:

- Garante que a UI está actualizada com o último estado do modelo de dados;
- Evita memory leaks;
- Evita erros e crashes devido a atividades paradas;
- Não necessita de cuidados manuais com o ciclo dos componentes em Android;

LiveData em Android

Desde que associada ao mesmo contexto, neste caso a Activity em Android é possível subscrever alterações ao modelo de dados através do método observable e do callback que será executado sempre que houverem alterações ao modelo de dados.

- A classe MutableLiveData e LiveData serão responsáveis por executar cada callback registado em cada componente Android;
- Quando o componente principal onde o LiveData é utilizado (ex: Activity) é removido ou destruído, também o objeto de LiveData é destruído.

LiveData

Implementação

- Para usar o conceito de LiveData em Android vamos usar ViewModels para encapsular os nossos modelos de dados;
- Os nossos modelos de dados devem representar live data, que queremos usar nos diferentes componentes de Android;

LiveData

Configuração [build.gradle (Project)]

```
ext{  
    lifecycle_version = "2.8.7"  
}
```

LiveData

Configuração [build.gradle (Module)]

```
dependencies {  
  
    // ViewModel  
    implementation "androidx.lifecycle:lifecycle-viewmodel:$lifecycle_version"  
    // LiveData  
    implementation "androidx.lifecycle:lifecycle-livedata:$lifecycle_version"  
  
}
```

LiveData

Definição [CounterViewModel.java]

```
public class CounterViewModel extends ViewModel {
```

Estende ViewModel

```
    private MutableLiveData<Integer> counter;
```

```
    public void count(){
```

```
        if (counter == null) {
```

```
            counter = new MutableLiveData<>();
```

```
            counter.setValue(0);
```

```
        } else {
```

```
            counter.setValue(counter.getValue() + 1);
```

```
        }
```

```
    public MutableLiveData<Integer> getCount(){
```

```
        if (counter == null) {
```

```
            counter = new MutableLiveData<>();
```

```
            counter.setValue(0);
```

```
        }
```

```
        return counter;
```

```
    }
```

```
}
```

Verificar se o counter foi iniciado. Em caso negativo iniciamos a zero.

MutableLiveData significa que o valor irá sofrer alterações durante a execução

Se já tivermos um counter ativo, incrementamos o seu valor

Esta função será observada e chamada sempre que o valor de counter sofrer alterações

LiveData

Adicionar o ViewModel [MainActivity.java]

```
private CounterViewModel counterViewModel;  
  
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    (...)  
  
    counterViewModel = new ViewModelProvider(this).get(CounterViewModel.class);  
  
    Observer<Integer> countObserver = new Observer<Integer>() {  
        @Override  
        public void onChanged(Integer count) {  
            updateCounter(count);  
        }  
    };  
  
    counterViewModel.getCount().observe(this, countObserver);  
}  
  
@Override  
public void onClick(View v) {  
    switch (v.getId()) {  
        case R.id.btn_count:  
            counterViewModel.count();  
            break;  
    }  
}  
  
private void updateCounter(int count) {  
    txtCount.setText(String.valueOf(count));  
}
```

Definição do ViewModel

Criação de um Observer.
Sempre que ele for
invocado vai chamar a
função updateCounter

Adição do Observer à
função getCount. Esta
função devolve um
MutableLiveData

LiveData, ViewModel e Fragments

Com combinação de LiveData, ViewModel e Fragments é possível estabelecer comunicação entre fragments sem necessidade de “passar” essa comunicação pela Activity.

- Vamos utilizar o exemplo da aplicação “Hello Toast” para demonstrar esse comportamento.

LiveData, ViewModel e Fragments

Exemplo

DisplayFragment.java

CounterFragment.java

Hello Toast

TOAST

0

COUNT

LiveData, ViewModel, Fragments

Definição [SharedViewModel.java]

```
public class SharedViewModel extends ViewModel {  
  
    private MutableLiveData<Integer> counter;  
  
    public void count(){  
        if (counter == null) {  
            counter = new MutableLiveData<>();  
            counter.setValue(0);  
        }else{  
            counter.setValue(counter.getValue() + 1);  
        }  
    }  
  
    public MutableLiveData<Integer> getCount(){  
        if (counter == null) {  
            counter = new MutableLiveData<>();  
            counter.setValue(0);  
        }  
        return counter;  
    }  
}
```

Estende ViewModel

Verificar se o counter foi iniciado. Em caso negativo iniciamos a zero.

Se já tivermos um counter ativo, incrementamos o seu valor

Esta função será observada e chamada sempre que o valor de counter sofrer alterações

LiveData, ViewModel, Fragments

Definição [CounterFragment.java]

```
public class CounterFragment extends Fragment implements View.OnClickListener {  
  
    private Button btnCounter;  
    private SharedViewModel sharedViewModel;  
  
    @Nullable  
    @Override  
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {  
        View mContentView = inflater.inflate(R.layout.counter_ui, container, false);  
        btnCounter = mContentView.findViewById(R.id.btn_add);  
        btnCounter.setOnClickListener(this);  
  
        sharedViewModel = new ViewModelProvider(requireActivity()).get(SharedViewModel.class);  
  
        return mContentView;  
    }  
  
    @Override  
    public void onClick(View v) {  
        switch (v.getId()) {  
            case R.id.btn_add:  
                sharedViewModel.count();  
                break;  
        }  
    }  
}
```

Definição do ViewModel

Cada vez que o botão é pressionado, chamamos a função responsável por incrementar o contador

LiveData, ViewModel, Fragments

Definição [DisplayFragment.java]

```
public class DisplayFragment extends Fragment {  
  
    private TextView txtCounter;  
    private SharedViewModel sharedViewModel;  
  
    @Nullable  
    @Override  
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,  
                             @Nullable Bundle savedInstanceState) {  
        View mContentView = inflater.inflate(R.layout.counter_display, container, false);  
  
        txtCounter = mContentView.findViewById(R.id.txt_counter);  
  
        sharedViewModel = new ViewModelProvider(requireActivity()).get(SharedViewModel.class);  
  
        Observer<Integer> counterObserver = count -> updateCounter(count);  
        sharedViewModel.getCount().observe(getViewLifecycleOwner(), counterObserver);  
  
        return mContentView;  
    }  
  
    private void updateCounter(int count){  
        txtCounter.setText(String.valueOf(count));  
    }  
}
```

Definição do ViewModel

Criação de um Observer.
Sempre que ele for
invocado vai chamar a
função updateCounter

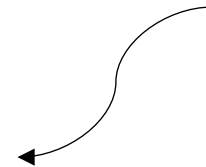
Adição do Observer à
função getCount. Esta
função devolve um
MutableLiveData

LiveData, ViewModel, Fragments

Definição [MainActivity.java]

```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

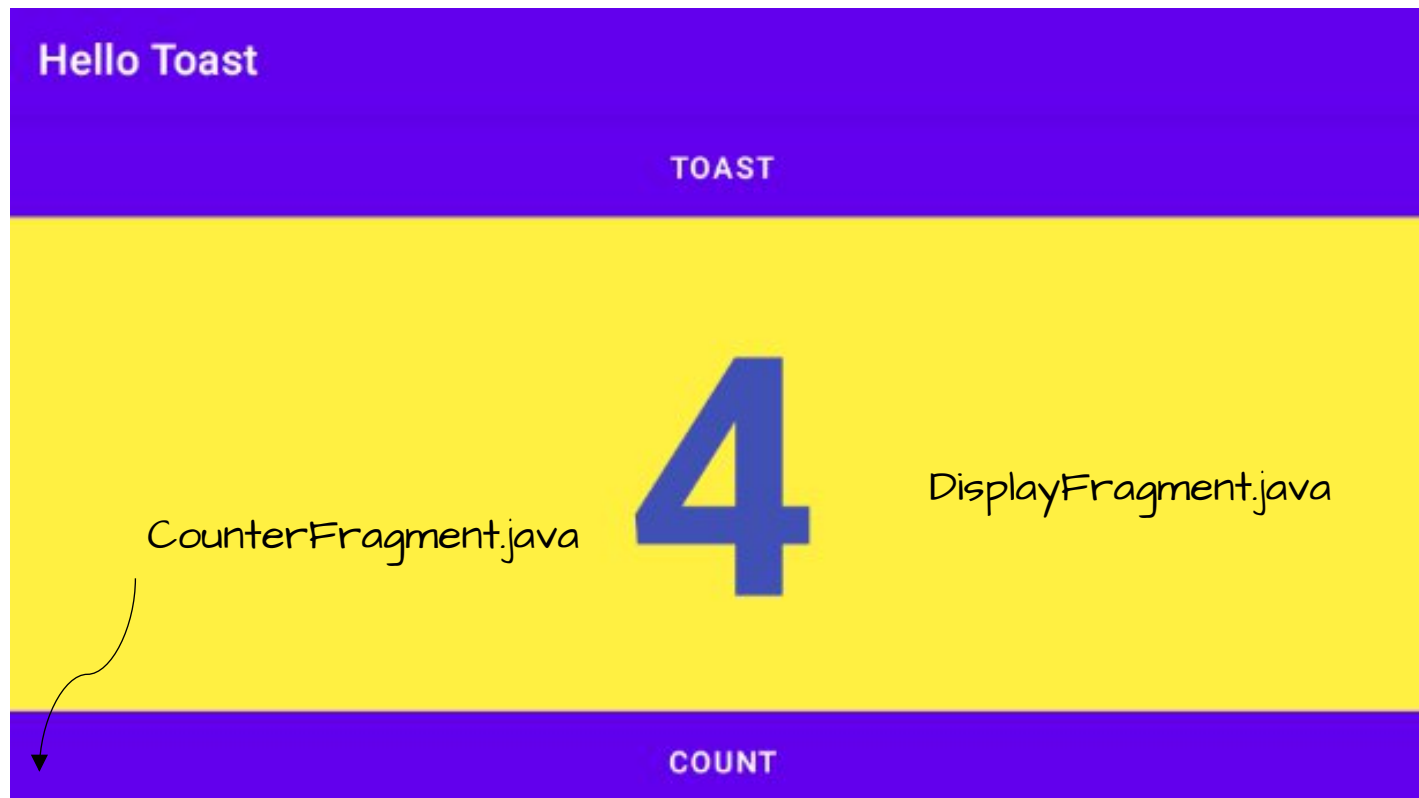
Fragments foram
definidos no layout



MainActivity não tem informação
sobre as comunicações feitas
entre os fragments

LiveData, ViewModel, Fragments Resultado

utilizador rodou ecrã



Hello Toast

TOAST

DisplayFragment.java

4

CounterFragment.java

COUNT

Leitura Adicional

Live Data:

<https://developer.android.com/topic/libraries/architecture/livedata>

View Model:

<https://developer.android.com/topic/libraries/architecture/viewmodel.html>

Programação Para Dispositivos Móveis I

VIEW MODEL
LIVE DATA

2024/_25 CTeSP – Desenvolvimento para a Web e Dispositivos Móveis

Ricardo Barbosa , rmb@estg.ipp.pt

Carlos Aldeias, cfpa@estg.ipp.pt

Adaptação do conteúdo dos slides de João Ramos jrmr@estg.ipp.pt e Fábio Silva fas@estg.ipp.pt