06

Implementando o DatePickerDialog

Transcrição

Na aula anterior, fizemos a primeira implantação do nosso *Dialog*, entretanto, há detalhes que precisamos nos atentar.

O componente de **valor**, está funcionando como o esperado. Já o campo **data**, está com uma informação de data fixa, e quando clicamos não acontece nada, ou seja, esse campo ainda não foi implementado.

De acordo com o campo **Data** na *app* base, a data é mostrada não como um texto fixo, mas sim no formato de data: `18/10/2017`.

Pensando nisso, é necessário colocar essa informação assim que o *Dialog* abrir. Ainda na *app* base, ao clicar na data, vemos um calendário onde podemos setar a data.

Recapitulando: Precisamos colocar a **data atual** no momento em que o `AlertDialog` é aberto, e após clicar na data, o componente de calendário deverá ser aberto.

A princípio, no código, precisamos pegar o componente que representa a data. Podemos até pensar em pegar esse componente a partir do *synthetics*, já que estamos na *activity*, certo?

Bom, não exatamente, pois o nosso *layout* não foi criado na *activity*, e sim a partir do objeto `viewCriada` que é próprio do *dialog*. Pensando nesse detalhe, pegaremos o componente a partir da `viewCriada`:

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        val view: View = window.decorView
        val viewCriada = LayoutInflater.from(context: this)
            .inflate(R.layout.form_trasacao,
                view as ViewGroup,
                attachToRoot: false)

        viewCriada.form_transacao_data

        AlertDialog.Builder(context: this)
            .setTitle(R.string.adiciona_receita)
            .setView(viewCriada)
            .show()
    }
```

Perceba que `form_transacao_data` é um **EditText**, e com isso, podemos usá-lo para colocar informações através da *property* `text`.

Na implementação do **EditText**, os métodos de acesso possuem uma certa divergência. O `set` do atributo `text` espera de fato, uma `String`, entretanto o `getText` é um *editable*. Por ter essa diferença nos métodos de acesso, não somos capazes de utilizar as *properties*.

Para conseguirmos colocar um texto, somos obrigados a colocar a função de acesso `setText()`, assim como é feito no Java.

```
viewCriada.form_transacao_data.setText("Data atual")
```

Vamos testar a aplicação.

Veja que o texto "Data atual" apareceu no `AlertDialog`, entretanto não queremos essa informação em texto, e sim a data atual. Para isso, é necessário criar uma variável que irá conter essa informação.

Pegaremos uma instância de `Calendar`, e a colocaremos em uma nova variável chamada `hoje`. Esperamos que `hoje` seja mandada no formato brasileiro:

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        val view: View = window.decorView
        val viewCriada = LayoutInflater.from(context: this)
            .inflate(R.layout.form_trasacao,
                view as ViewGroup,
                attachToRoot: false)

        val hoje = Calendar.getInstance()
        viewCriada.form_transacao_data
            .setText(hoje.formataParaBrasileiro())

        AlertDialog.Builder(context: this)
            .setTitle(R.string.adiciona_receita)
            .setView(viewCriada)
            .show()
    }
```

Como podemos ver, a data de hoje está visível. Mas é claro que, se clicarmos na data nada irá acontecer, pois não colocamos nenhum *Listener* para tomar alguma ação.

Implementaremos o evento de **clique**. Utilizaremos a função `setOnClickListener()` no modo **lambda**.

```
viewCriada.form_transacao_data
    .setText(hoje.formataParaBrasileiro())
viewCriada.form_transacao_data
    .setOnClickListener {

    }
```

Dessa forma, estamos pegando o evento de clique que, no momento em que clicarmos, o componente de calendário deverá ser aberto. Esse componente no Android, é conhecido como `DatePicker`. Quando digitamos `DatePicker`, o Android Studio nos mostra duas opções para se utilizar: o `DatePicker()` é um *widget* qualquer, e ele poderia ser criado manualmente, e o `DatePickerDialog()`.

Se repararmos na *app* base, o `DatePicket` parece muito com um *Dialog*, logo, vamos utilizar o `DatePickerDialog()`, fazendo com que esse *Dialog* apareça quando clicarmos. O primeiro parâmetro a ser passado é o **context: this** e o `this` se refere à *activity*. Já o segundo, é um *Listener* que irá pegar as informações que forem clicadas no `DatePicker`.

A princípio, podemos começar com *Object Expression*, passando a interface `OnDateSetListener`:

```
viewCriada.form_transacao_data
    .setOnClickListener {
        DatePickerDialog(context: this,
            object : DatePickerDialog.OnDateSetListener {

            })
    }
```

Repare que, por mais que estejamos utilizando *Object Expression*, o compilador está reclamando, por que isso ocorre?

A sobrecarga do construtor `DatePickerDialog` só funciona para APIs mínimas do número `24` do Android, porém estamos usando a API mínima `19`, e ele não tem a capacidade de utilizar essa API mínima. Mediante a esse problema, quais seriam as possibilidades para fazer com que isso funcionasse?

A princípio, poderíamos marcar uma *anotação* indicando que isso será executado somente se estivermos dentro da API `24`. Mas, essa não é uma abordagem muito legal, pois se alguém estiver utilizando a API `19`, não poderão usar o que estaremos implementando aqui.

A fim de dar suporte também para a API mínima, podemos utilizar um outro **construtor**, uma nova sobrecarga desse construtor, que recebe `ano`, `mes` e `dia`, tornando assim, uma API compatível com as antigas.

```
viewCriada.form_transacao_data
    .setOnClickListener {
        DatePickerDialog(context: this,
            object : DatePickerDialog.OnDateSetListener {

            }, ano, mes, dia)
    }
```

Com o cursor em cima do `object`, implementaremos os outros membros a partir do atalho "Alt + Enter", e selecionaremos `onDateSet()`. O código ficará assim:

```
viewCriada.form_transacao_data
    .setOnClickListener {
        DatePickerDialog(context: this,
            object : DatePickerDialog.OnDateSetListener {
                override fun onDateSet(p0: DatePicker?, p1: Int, p2: Int, p3: Int) {

                }
            }, ano, mes, dia)
    }
```

Após ter feito esses passos, o próprio Android Studio nos fornece a dica para convertermos para **lambda** para que fique mais simples. Ao converter, o código ficará assim:

```
viewCriada.form_transacao_data
    .setOnClickListener {
        DatePickerDialog(context: this,
            DatePickerDialog.OnDateSetListener {p0, p1, p2, p3 ->

            }
```

```
            , ano, mes, dia)
        }
```

Agora podemos fazer a implementação do `OnDateSetListener`. Sabemos que temos quatro parâmetros, onde o primeiro é a `view` que representa a view do `DatePicker`, o segundo, o terceiro e o quarto parâmetro são respectivamente o `ano`, o `mes` e o `dia` recebidos no momento em que o usuário seleciona no calendário.

```
viewCriada.form_transacao_data
    .setOnClickListener {
        DatePickerDialog(context: this,
            DatePickerDialog.OnDateSetListener { view, ano, mes, dia ->

        }
        , ano, mes, dia)
    }
```

A partir dessas informações, conseguimos montar uma data que irá representar a data que vai aparecer para o usuário. Dentro do escopo do **lambda**, podemos criar uma data baseando-se nessas informações:

```
viewCriada.form_transacao_data
    .setOnClickListener {
        DatePickerDialog(context: this,
            DatePickerDialog.OnDateSetListener { view, ano, mes, dia ->
                Calendar.getInstance()
        }
        , ano, mes, dia)
    }
```

Uma ocorrência aparecerá perguntando se queremos fazer uma substituição em dois pontos, já que estamos chamando o `getInstance()` em dois pontos do código. Vamos escolher a opção de substituir em somente **um ponto**. Após essa decisão, será criada uma variável, e a chamaremos de `dataSelecionada`:

```
viewCriada.form_transacao_data
    .setOnClickListener {
        DatePickerDialog(context: this,
            DatePickerDialog.OnDateSetListener { view, ano, mes, dia ->
                val dataSelecionada = Calendar.getInstance()

        }
        , ano, mes, dia)
    }
```

Feito isso, agora podemos modificar os valores padrões da atual, com os valores recebidos pelo usuário.

```
viewCriada.form_transacao_data
    .setOnClickListener {
        DatePickerDialog(context: this,
            DatePickerDialog.OnDateSetListener { view, ano, mes, dia ->
                val dataSelecionada = Calendar.getInstance()
                dataSelecionada.set(ano, mes, dia)
        }
```

```
            , ano, mes, dia)
        }
```

A partir desses parâmetros, estamos construindo uma data com as informações que o usuário selecionou. Agora que está tudo selecionado, podemos pegar o componente `form_transacao_data` a partir da `viewCriada` e setar a informação com o `setText()`, passando a `dataSelecionada.formataParaBrasileiro()`.

```
    viewCriada.form_transacao_data
        .setOnClickListener {
            DatePickerDialog(context: this,
                DatePickerDialog.OnDateSetListener { view, ano, mes, dia ->
                    val dataSelecionada = Calendar.getInstance()
                    dataSelecionada.set(ano, mes, dia)
                    viewCriada.form_transacao_data
                        .setText(dataSelecionada.formataParaBrasileiro())
            }
            , ano, mes, dia)
        }
```

Recapitulando: Estamos criando uma data com o dia atual, a modificamos com as informações que o usuário mandou, e computamos essas informações no `form_transacao_data`, campo que é usado para representar a data.

O que falta agora é ajustar as informações que mandamos, os parâmetros de `ano`, `mes` e `dia`. No momento em que o `DatePicker` for criado, serão considerados somente a informação desses três parâmetros. Por exemplo, ao selecionarmos uma data qualquer no `DatePicker`, a data selecionada ficará no campo. Ao clicarmos novamente no `DatePicker`, a data que já está selecionada é a data de hoje, pois a deixamos fixa.

Sendo assim, por enquanto, criaremos essas variáveis da seguinte maneira:

```
    .setOnClickListener {
        val view: View = window.decorView
        val viewCriada = LayoutInflater.from(context: this)
            .inflate(R.layout.form_transacao,
                view as ViewGroup,
                attachToRoot: false)

        val ano = 2017
        val mes = 9
        val dia = 18

        val hoje = Calendar.getInstance()
        viewCriada.form_transacao_data
            .setText(hoje.formataParaBrasileiro())
```

> Em relação aos meses, as APIs funcionam de forma diferente, iniciando a sua contagem a partir do zero. Então, Janeiro é representado como o `0`, e Outubro, por exemplo, é representado pelo `9`.

Legal. Agora que já conseguimos implementar essa primeira parte do `DatePickerDialog`, precisamos pedir que ele apareça, sempre que clicamos no componente. Para isso, basta acrescentar a função `show()`. Em seguida, mostraremos o código da *activity* por completo:

```
lista_transacoes_adiciona_receita
    .setOnClickListener {
        val view: View = window.decorView
        val viewCriada = LayoutInflater.from(context: this)
            .inflate(R.layout.form_trasacao,
                view as ViewGroup,
                attachToRoot: false)

        val ano = 2017
        val mes = 9
        val dia = 18

        val hoje = Calendar.getInstance()
        viewCriada.form_transacao_data
            .setText(hoje.formataParaBrasileiro())
        viewCriada.form_transacao_data
            .setOnClickListener {
                DatePickerDialog(context: this,
                    DatePickerDialog.OnDateSetListener { view, ano, mes, dia ->
                        val dataSelecionada = Calendar.getInstance()
                        dataSelecionada.set(ano, mes, dia)
                        viewCriada.form_transacao_data
                            .setText(dataSelecionada.formataParaBrasileiro())
                    }
                    , ano, mes, dia)
                    .show()
            }

        // continuação do código
    }
```

Vamos testar para ver se está funcionando.

Podemos ver que agora, ao clicar na data, o `DatePickerDialog` aparece. Selecionaremos o dia `19` do mês vigente. Ao selecionarmos e clicarmos em "Ok", vemos que a data mudou para o dia `19`. E se clicarmos novamente para abrir o `DatePicker`, o dia voltará para o `18`. Por mais que venhamos setar outra data, e depois voltar ao `DatePicker`, sempre irá para o `18` de Outubro.

Os parâmetros `ano, mes, dia` estão fixos atualmente, mas em breve daremos uma melhorada para que eles sempre peguem o que selecionamos. Adiante, veremos como fazer para melhorar o aspecto da *categoria*, já que não temos nenhuma cadastrada e de acordo com a *app* base, temos várias categorias.