01

Criando a transação a partir do Dialog

Transcrição

Daremos início ao processo para pegar as informações que o usuário computar em nosso *Dialog* e transformar em uma transação, para que depois venhamos a colocar na lista de transações. No momento em que colocamos o botão positivo, esperamos que o usuário entenda que ele mandou as informações que ele desejava, e nós pegamos esses dados e transformamos em transações. Então, começaremos a partir dessa função.

Para isso, é necessário colocar o *Listener* que será responsável por tomar a ação no momento em que clicarmos no botão. Como vamos implementar esse *Listener*?

Podemos pedir para que ele implemente o `OnClickListener()`, mas o Android Studio nos mostra duas opções. A primeira é aquela que precisamos mandar uma classe anônima, e a segunda opção é utilizar a expressão Lambda, que é bem mais resumido. Vamos utilizar a segunda opção.

```
AlertDialog.Builder(context: this)
        .setTitle(R.string.adiciona_receita)
        .setView(viewCriada)
        .setPositiveButton(text: "Adicionar",
                DialogInterface.OnClickListener { dialogInterface, i ->

                })
        .setNegativeButton(text: "Cancelar", listener: null)
        .show()
```

O Android Studio nos mostra que ocorre uma redundância em `OnClickListener`, e se usarmos o "Alt + Enter", podemos remover essa parte e a expressão lambda ficará bem mais resumida.

```
.setPositiveButton(text: "Adicionar",
        { dialogInterface, i ->

        })
```

Quando estamos fazendo um Lambda de uma interface do Java, podemos colocar direto assim como fizemos no código acima, pois só temos uma única função para implementar.

Muito bem. Faremos o processo para pegar as informações. Utilizaremos o objeto `viewCriada`. A partir desse objeto, somos capazes de acessar todos os componentes, então vamos pegar cada um deles. O primeiro componente é o `form_transacao_valor`. Para pegar o valor desse campo, e já que ele é um *EditText*, podemos pegar a sua *property* `text` que devolve um `Editable`, que é o campo que contém a informação da String, e então devolvemos o `toString()` que devolverá a string que está dentro do input dele.

```
.setPositiveButton(text: "Adicionar",
        { dialogInterface, i ->
            val valorEmTexto = viewCriada
                    .form_transacao_valor.text.toString()
        })
```

Pegamos o primeiro parâmetro do *Dialog*. Vamos pegar o segundo a partir de `viewCriada` e o componente `form_transacao_data`. A partir desse componente, podemos pegar a *property* `text`, e o `toString()` dela. Tudo isso ficará dentro da variável `dataEmTexto` do tipo `val`.

```
.setPositiveButton(text: "Adicionar",
        { dialogInterface, i ->
            val valorEmTexto = viewCriada
                    .form_transacao_valor.text.toString()
            val dataEmTexto = viewCriada
                    .form_transacao_data.text.toString()
        })
```

E agora, nos resta pegar o último campo, a categoria selecionada. Nesse passo, usaremos o componente `form_transacao_categoria`, que diferente dos outros componentes, ele é um *Spinner*, então a *property* será `selectedItem`. Mas já que o item selecionado é uma String, então podemos utilizar o `toString()`.

```
.setPositiveButton(text: "Adicionar",
        { dialogInterface, i ->
            val valorEmTexto = viewCriada
                    .form_transacao_valor.text.toString()
            val dataEmTexto = viewCriada
                    .form_transacao_data.text.toString()
            val categoriaEmTexto = viewCriada
                    .form_transacao_categoria.selectedItem.toString()
        })
```

Aqui temos todos os valores que o usuário colocou. E se fossemos criar uma transação baseando-se nas informações acima, o que poderia acontecer?

Criaremos uma nova transação que será do tipo `Receita`. Em seguida, indicaremos o campo **valor**:

```
.setPositiveButton(text: "Adicionar",
        { dialogInterface, i ->
            val valorEmTexto = viewCriada
                    .form_transacao_valor.text.toString()
            val dataEmTexto = viewCriada
                    .form_transacao_data.text.toString()
            val categoriaEmTexto = viewCriada
                    .form_transacao_categoria.selectedItem.toString()

            Transacao(tipo = Tipo.RECEITA, valor = valorEmTexto)
        })
```

Podemos reparar que `valorEmTexto` nessa última linha não foi compilado, pois `valorEmTexto` é uma **String** ao invés de um **BigDecimal**. Em outras palavras, é necessário fazer o processo de conversão para que essa variável seja compatível com a transação.

Vamos realizar o processo de conversão de uma `String` para um `BigDecimal`. No momento em que construímos o *BigDecimal*, podemos mandar o `valorEmTexto`. Quando mandamos uma String em um *BigDecimal*, ele já realiza a conversão para nós e já transforma em BigDecimal.

```
.setPositiveButton(text: "Adicionar",
        { dialogInterface, i ->
            val valorEmTexto = viewCriada
                    .form_transacao_valor.text.toString()
            val dataEmTexto = viewCriada
                    .form_transacao_data.text.toString()
            val categoriaEmTexto = viewCriada
                    .form_transacao_categoria.selectedItem.toString()

            val valor = BigDecimal(valorEmTexto)

            Transacao(tipo = Tipo.RECEITA,
                    valor = valor)
        })
```

Como podemos ver, a conversão para o *BigDecimal* foi devolvida na variável `valor`. E ao invés de mandar `valorEmTexto` em `valor = valorEmTexto`, podemos simplesmente colocar a variável que retorna a conversão. Manteremos o mesmo processo para a **Data**. Ao invés de realizarmos a conversão de *String* para *BigDecimal*, faremos a conversão de *String* para *Calendar*.

Para converter uma **data** que está em texto, basicamente precisamos utilizar uma classe específica para isso, o ***SimpleDateFormat***. Ele já foi utilizado para criar o `formataParaBrasileiro()`, e agora ele será capaz de os ajudar novamente. Vamos criar uma instância do `SimpleDateFormat()` e indicar o padrão que esperamos:

```
.setPositiveButton(text: "Adicionar",
         { dialogInterface, i ->
            val valorEmTexto = viewCriada
                    .form_transacao_valor.text.toString()
            val dataEmTexto = viewCriada
                    .form_transacao_data.text.toString()
            val categoriaEmTexto = viewCriada
                    .form_transacao_categoria.selectedItem.toString()

            val valor = BigDecimal(valorEmTexto)

            val formatoBrasileiro = SimpleDateFormat(pattern: "dd/MM/yyyy")

            Transacao(tipo = Tipo.RECEITA,
                    valor = valor)
         })
```

Após ter feito o `formatoBrasileiro`, passaremos a `dataEmTexto` para que ele seja convertido para uma data compatível com a API do Java.

```
val formatoBrasileiro = SimpleDateFormat(pattern: "dd/MM/yyyy")
val dataConvertida = formatoBrasileiro.parse(dataEmTexto)
```

Com o "Alt + Enter" na variável `dataConvertida`, podemos descobrir o tipo que foi devolvido. Clicamos na opção "*Specify type explicitly*", e mostrado o tipo `Date`:

```
        val formatoBrasileiro = SimpleDateFormat(pattern: "dd/MM/yyyy")
        val dataConvertida: Date = formatoBrasileiro.parse(dataEmTexto)
```

Então, na hora da conversão foi devolvido a API de Date, entretanto estamos lidando com a API do *Calendar*, e o que podemos fazer para colocar informações no Calendar? Primeiro, podemos criar uma instância do *Calendar* e chamá-la de `data`, e agora podemos inserir a *property* `time` para poder colocar a `dataConvertida`:

```
        val formatoBrasileiro = SimpleDateFormat(pattern: "dd/MM/yyyy")
        val dataConvertida: Date = formatoBrasileiro.parse(dataEmTexto)
        val data = Calendar.getInstance()
        data.time = dataConvertida
```

Então, depois de todo esse processo, temos a data convertida. Agora podemos passar a `data` na transação:

```
.setPositiveButton(text: "Adicionar",
        { dialogInterface, i ->
                val valorEmTexto = viewCriada
                        .form_transacao_valor.text.toString()
                val dataEmTexto = viewCriada
                        .form_transacao_data.text.toString()
                val categoriaEmTexto = viewCriada
                        .form_transacao_categoria.selectedItem.toString()

                val valor = BigDecimal(valorEmTexto)

                val formatoBrasileiro = SimpleDateFormat(pattern: "dd/MM/yyyy")
                val dataConvertida: Date = formatoBrasileiro.parse(dataEmTexto)
                val data = Calendar.getInstance()
                data.time = dataConvertida

                        Transacao(tipo = Tipo.RECEITA,
                                valor = valor,
                                data = data,
                                categoria = categoriaEmTexto)
        })
```

Perceba que já inserimos o último campo que faltava: a Categoria. E como a categoria já é uma String, então colocamos `categoriaEmTexto` direto.

Após ter criado a transação que o usuário espera, podemos devolver esse objeto para `transacaoCriada`:

```
        val transacaoCriada = Transacao(tipo = Tipo.RECEITA,
                valor = valor,
                data = data,
                categoria = categoriaEmTexto)
```

Inclusive podemos até ver se a informação da transação está recebendo as informações que esperamos. Criaremos um *toast* somente para testar se a transação está sendo criada corretamente. Usaremos o recurso do **String Template**

```
val transacaoCriada = Transacao(tipo = Tipo.RECEITA,
        valor = valor,
        data = data,
        categoria = categoriaEmTexto)

Toast.makeText(context: this, "${transacaoCriada.valor} - " +
        "${transacaoCriada.categoria} - " +
        "${transacaoCriada.data.formataParaBrasileiro} - " +
        "${transacaoCriada.tipo}", Toast.LENGTH_LONG).show
```

Vamos executar a *app*. No botão de "Adicionar receita", colocaremos o valor de `100`, a data do dia `28/10/2017`, e escolheremos a categoria `Empréstimo`. Clicaremos no botão "Adicionar", e veremos que não estamos tendo nenhum problema. O nosso próximo passo será colocar a transação na lista de transações, assim o usuário será capaz de colocar as transações na app dele. Até mais.