03

Adicionando transação do dialog na lista

Transcrição

Baseando-se nas informações que o usuário mandou no *Dialog*, faremos o seguinte procedimento: no momento em que recebermos os dados no *Dialog*, faremos com que eles sejam computados e enviados na lista de transação. Primeiro, removeremos o *Toast* pois não iremos mais precisar dele.

Depois, é necessário ter acesso à lista que está computando as informações tanto em nossa lista, quanto em nosso resumo para fazer o cálculo. Então, ao invés de utilizar a lista `transacoesDeExemplo()` em `ListaTransacoesActivity`, iremos removê-la. Criaremos uma lista que seja acessível por todos os membros.

Na *activity*, criaremos uma *property* para isso:

```
class ListaTransacoesActivity : AppCompatActivity() {

    private val transacoes: List<Transacao>
}
```

A partir desse momento, estamos criando uma *property* que será acessível para todos e irá conter todas as transações que criarmos. Vamos inicializar a nossa *property*:

```
class ListaTransacoesActivity : AppCompatActivity() {

    private val transacoes: List<Transacao> = listOf()
}
```

A linha `val transacoes: List<Transacao> = transacoesDeExemplo` será deletada, pois antes a usávamos para criar a lista de transações. Na parte do `configuraResumo(transacoes)` e no `configuraLista(transacoes)`, estamos enviando as transações, mas agora ela é acessível por todos, e com isso não precisamos mais enviá-la via parâmetro. O código ficará desta forma:

```
class ListaTransacoesActivity : AppCompatActivity() {

    private val transacoes: List<Transacao> = listOf()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_lista_transacoes)

        configuraResumo()

        configuraLista()

        lista_transacoes_adiciona_receita
                .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())
                        viewCriada.form_transacao_data
                                .setOnClickListener {
                                    DatePickerDialog(context: this,
                                            DatePickerDialog.OnDateSetListener {...}
                                            , ano, mes, dia)
                                            .show()
                                }

                        val adapter = ArrayAdapter
                                .createFromResource(context: this,
                                        R.array.categorias_de_receita,
                                        android.R.layout.simple_spinner_dropdown_item)

                        viewCriada.form_transacao_categoria.adapter = adapter

                        AlertDialog.Builder(context: this)
                                .setTitle(R.string.adiciona_receita)
                                .setView(viewCriada)
                                .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/M
                                            val dataConvertida: Date = formatoBrasileiro.parse(datal
                                            val data = Calendar.getInstance()
                                            data.time = dataConvertida

                                            val transacaoCriada = Transacao(tipo = Tipo.RECEITA,
                                                valor = valor,
                                                data = data,
                                                categoria = categoriaEmTexto)
                                        })
                                .setNegativeButton(text: "Cancelar", listener: null)
                                .show()
                    }
        }

        private fun configuraResumo() {
            val view: View = window.decorView
            val resumoView = ResumoView(context: this, view, transacoes)
            resumoView.atualiza()
```

```
    }

    private fun configuraLista() {
        lista_transacoes_listview.adapter = ListaTransacoesAdapter(transacoes, context: this)
    }
}
```

Já que mudamos bastante essa parte do código, é interessante executa-lo novamente para verificarmos se está tudo funcionando.

Temos uma lista vazia! E o Resumo também está zerado. Então precisamos pegar a lista `transacoes`, e adicioná-la no momento em que criamos uma transação nova:

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

    transacoes.add(transacaoCriada)
```

Repare que a função `add()` não foi compilada, mas por que isso acontece? De modo geral, o Kotlin tem um conceito diferente em relação às *collections* do Java. Quando utilizamos uma `List`, estamos usando uma *collection* que é imutável, ou seja, uma lista que não modifica os seus valores internos. Então, quando criamos a lista vazia `listOf()`, estamos dizendo que ela será vazia para sempre, e não vamos conseguir modificar nada nela. Então, como podemos fazer para utilizar uma **lista mutável**?

Podemos utilizar uma outra assinatura, uma outra interface: a ***MutableList***.

```
    private val transacoes: MutableList<Transacao> = listOf()
```

A partir do momento em que utilizamos a `MutableList`, temos a capacidade de utilizar uma *collection*, uma lista, que permite modificar os seus valores internos. Com essa mudança, o `listOf()` deixa de funcionar, pois ele nos devolve uma *lista imutável* (por padrão).

Como vamos devolver uma instância de uma `MutableList` vazia? Podemos fazer assim:

```
    private val transacoes: MutableList<Transacao> = mutableListOf()
```

Agora, nós conseguimos **adicionar** a transação criada. Entretanto, por mais que estejamos adicionando esse item em `transacoes`, tanto a função `configuraResumo()` como a função `configuraLista()`, foram criadas a partir da transação quando ela estava vazia, e agora estamos adicionando valores novos. Para que essas informações sejam imputadas novamente, precisamos chamar o `atualiza()` e também, criar o *adapter*.

Portanto, precisamos configurar a lista e o resumo novamente.

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

transacoes.add(transacaoCriada)
configuraLista()
configuraResumo()
```

No momento em que adicionamos uma transação na lista de transações, precisamos garantir que a lista e o resumo sejam também atualizados. Inclusive, podemos extraí-los para uma função. Utilizaremos o atalho "Ctrl + Alt + M" para extrair após ter selecionado `transacoes`, `configuraLista()` e `configuraResumo()`. Esse código se chamará **atualizaTransacoes**. E então, tudo o que precisará ser atualizado, ficará isolado.

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

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

            }
    }

    private fun atualizaTransacoes(transacao: Transacao) {
            transacoes.add(transacao)
            configuraLista()
            configuraResumo()
    }
```

Vamos executar a *app* e ver o que acontece.

A princípio, está vazio. Então vamos adicionar uma transação. A transação precisa das informações de `valor` que será `100`, de `data` onde podemos colocar `20/10/2017`, e a `categoria` será `Pagamento`. Clicamos em "Adicionar".

Como podemos ver, o item foi adicionado. Entretanto, a camada do botão verde ainda está aparecendo. E como não estamos mais utilizando ele, seria interessante se ele fechasse após o usuário inserir a transação na lista.

Para fechar esse botão, faremos assim. A partir do momento em que estamos finalizando todas as operações no botão verde, ele deverá fechar o seu menu. A propriedade que se refere ao menu do botão é a `lista_transacoes_adiciona_menu` , e depois é só chamar o `.close()` .

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

            atualizaTransacoes(transacaoCriada)
            lista_transacoes_adiciona_menu.close()
        })
```

No momento em que chamamos a função `close()` , temos a possibilidade de fechar **com/sem animação**. Ao inserir `true` , permitimos que a animação seja realizada.

```
        lista_transacoes_adiciona_menu.close(animate: true)
```

Com isso, o usuário pode adicionar suas transações.