

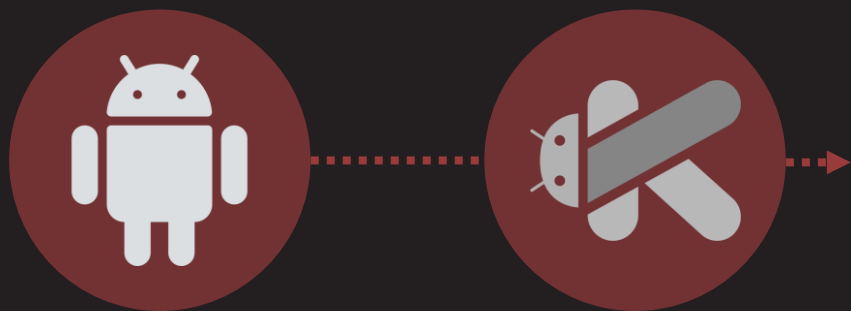
FIA P GRADUAÇÃO

# Hybrid Mobile App Development

Prof. Andrey Masiero

#02 – Adapter, Model e Extension Function

# Começando o caminho Jedi



# Adapter

Construa o seu próprio =D

# Construindo seu próprio Adapter

- Vimos como utilizar o ArrayAdapter para exibir uma lista de strings.
- Mas podemos deixar nossa aplicação, com uma experiência melhor para nosso usuário.
- E se as transações tivessem essa aparência:



# Construindo seu próprio Adapter

- Para isso, vamos criar um Adapter. Crie um pacote dentro de ui chamado adapter, em seguida crie uma classe chamada ListaTransacoesAdapter.

```
1 package br.com.fiap.financas.ui.adapter
2
3 class ListaTransacoesAdapter {
4
5 }
```

# Construindo seu próprio Adapter

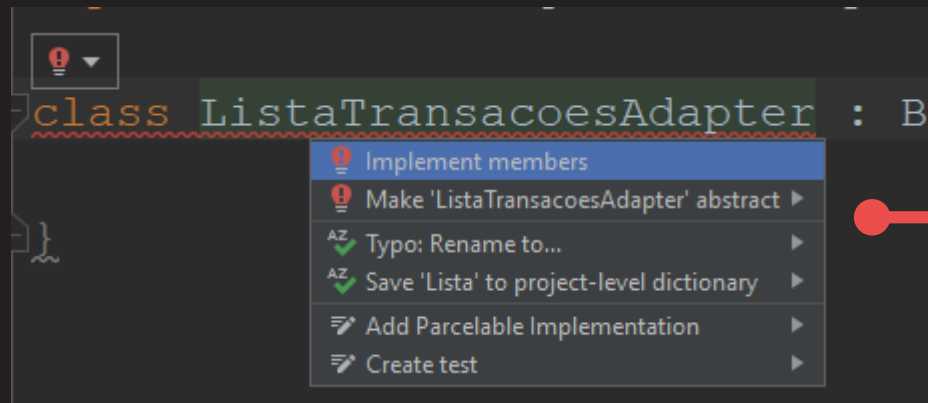
- Ela deve herdar de uma classe chamada BaseAdapter.

```
1 package br.com.fiap.financas.ui.adapter
2
3 import android.widget.BaseAdapter
4
5 class ListaTransacoesAdapter : BaseAdapter() {
6
7 }
```

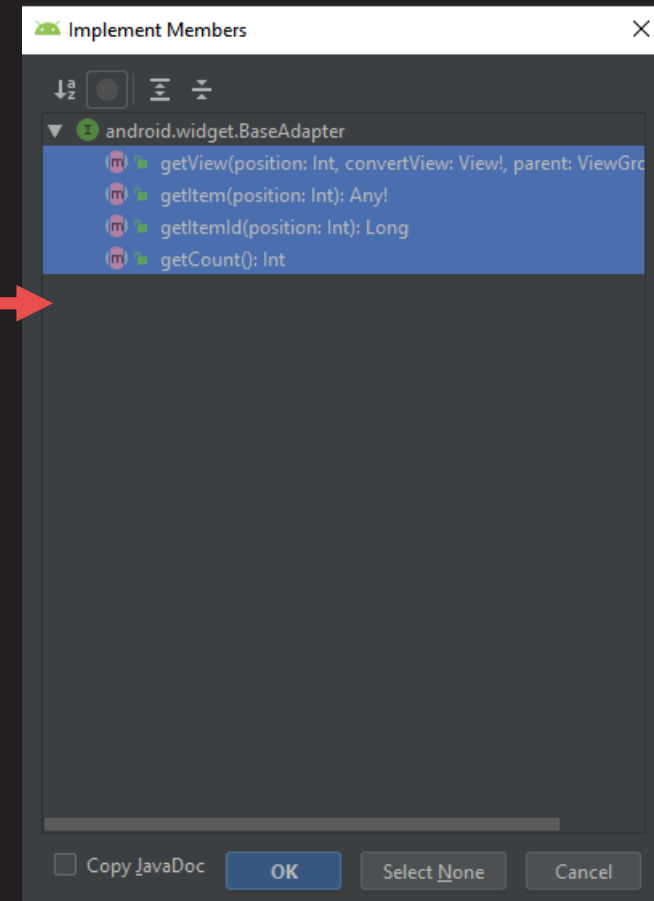
- O nome da classe está em vermelho, pois existem alguns métodos obrigatórios para implementar.

# Construindo seu próprio Adapter

- Pressione Alt + Enter no nome da classe e escolha Implement Methods:




- Selecione todos os métodos e clique em OK.





# Construindo seu próprio Adapter

```
7 class ListaTransacoesAdapter : BaseAdapter() {  
8     override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View {  
9         TODO(reason: "not implemented")  
10    }  
11  
12    override fun getItem(position: Int): Any {  
13        TODO(reason: "not implemented")  
14    }  
15  
16    override fun getItemId(position: Int): Long {  
17        TODO(reason: "not implemented")  
18    }  
19  
20    override fun getCount(): Int {  
21        TODO(reason: "not implemented")  
22    }  
23  
24 }
```



Essa função impede que o App seja executado.  
É proposital para evitar que alguma função fique sem implementação.  
Deve ser removida antes de executar o App, implementando os métodos necessários.

# Construindo seu próprio Adapter

- Agora devemos criar o atributo de lista de transações. Já vimos, na Activity que a boa prática é que essa lista seja imutável.

```
class ListaTransacoesAdapter : BaseAdapter() {  
  
    private val transacoes  
  
    override fun getView(position: Int, convertView:  
        TODO(reason: "not implemented")
```

Property must be initialized or be abstract

- Perceba, que o Android Studio nos dá uma dica, informando que ela deve ser inicializada. Vamos utilizar o construtor da classe para isso.

# Construindo seu próprio Adapter

- A declaração do atributo é feito logo após o nome da classe:

```
class ListaTransacoesAdapter(transacoes : List<String>) : BaseAdapter(){  
    private val transacoes = transacoes
```

- Após os dois pontos determinamos o tipo do parâmetro, no caso uma lista de strings.
- Depois disso é só inicializar o atributo da classe.

# Construindo seu próprio Adapter

- Nossa classe também precisa receber o contexto que o adapter deverá atuar.

```
class ListaTransacoesAdapter(transacoes : List<String>,  
                             context : Context) : BaseAdapter() {  
  
    private val transacoes = transacoes  
    private val context = context
```

- Como boa prática, a documentação do Kotlin recomenda que parâmetros sejam declarados verticalmente, como no exemplo.

# Construindo seu próprio Adapter

- Vamos trabalhar as funções:
  - **getItem**: retorna o item da lista pela posição

```
override fun getItem(position: Int): String {  
    return transacoes[position]  
}
```

- **getItemId**: retornar o id do item pela posição (Nosso caso, como não temos id, retornamos 0)

```
override fun getItemId(position: Int): Long {  
    return 0  
}
```

# Construindo seu próprio Adapter

- **getCount**: retorna o tamanho da lista

```
override fun getCount(): Int {  
    return transacoes.size  
}
```

- **getView**: cria a view
  - Vamos criar o container que vai construir nossa view. Da classe LayoutInflater use o método from (estático). Ele recebe o contexto como parâmetro.
  - Na sequência o método inflate é chamado recebendo como parâmetro o layout. No caso transacao\_item.
  - Repasse também o parâmetro parent e no último argumento informe false, para que a view seja criada pelo adapter.

# Construindo seu próprio Adapter

- A classe ListaTransacoesAdapter ficou assim:

```
class ListaTransacoesAdapter(transacoes : List<String>,
                             context : Context) : BaseAdapter() {

    private val transacoes = transacoes
    private val context = context

    override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View {
        return LayoutInflater.from(context).inflate(R.layout.transacao_item,
            parent, attachToRoot: false)
    }

    override fun getItem(position: Int): String {
        return transacoes[position]
    }

    override fun getItemId(position: Int): Long {
        return 0
    }

    override fun getCount(): Int {
        return transacoes.size
    }
}
```

# Utilizando o Adapter

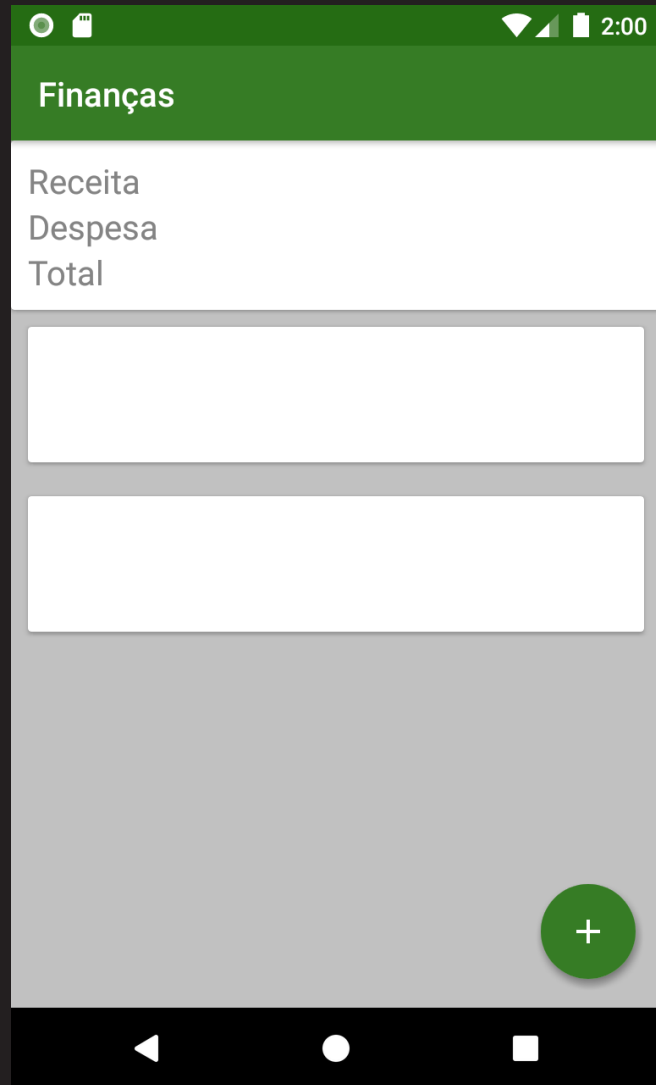
- Agora precisamos utiliza-lo na nossa Activity

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_lista_transacoes)  
  
    val transacoes = listOf("Comida - R$ 20,50", "Economia - R$ 100,00")  
  
    lista_transacoes_listview.adapter = ListaTransacoesAdapter(transacoes, context: this)  
}
```

- Removemos o ArrayAdapter e criamos a nosso Adapter.



# Resultado do Novo Adapter



# String não!

Cadê nosso modelo?

# Classe Model

- O uso de string é bom didaticamente, contudo para um sistema devemos criar um modelo que represente efetivamente uma transação.
- Crie um pacote model, no mesmo nível que o pacote ui. Nele crie a classe Transacao.

# Classe Model

- Utilizaremos atributos val e passaremos como parâmetro os três dados da transação através do construtor.

```
class Transacao(valor : BigDecimal,  
                descricao : String,  
                data : Calendar) {  
  
    private val valor : BigDecimal = valor  
    private val descricao : String = descricao  
    private val data : Calendar = data  
  
}
```

# Utilizando o Modelo

- Nesse momento, vamos trocar a nossa lista de transações em String para o modelo de Transacao:

```
val transacoes = listOf(Transacao(BigDecimal(20.5), descricao: "Comida", Calendar.getInstance()),  
    Transacao(BigDecimal(100.0), descricao: "Rendimento", Calendar.getInstance()))
```

- Agora precisamos trocar o adapter, pois ele está preparado para receber uma lista de Strings.

```
lista_transacoes_listview.adapter = ListaTransacoesAdapter(transacoes, context: this)
```

Type mismatch.  
Required: List<String>  
Found: List<Transacao>

# Modificando o Adapter

- Primeiro lugar no construtor:

```
class ListaTransacoesAdapter(transacoes : List<Transacao>,  
                             context : Context) : BaseAdapter() {
```

- Na sequência o método getItem

```
    override fun getItem(position: Int): Transacao {  
        return transacoes[position]  
    }
```

# Modificando o Adapter

- Agora precisamos mexer no método do getView. A nossa view precisa receber os dados da transação.
- Vamos atribuir a view criada pelo LayoutInflater para uma variável.

```
override fun getView(position: Int, convertView: View?, parent: ViewGroup?): View {  
    val viewCriada = LayoutInflater.from(context).inflate(R.layout.transacao_item,  
        parent, attachToRoot: false)  
  
    return viewCriada  
}
```

# Modificando o Adapter

- Utilizando o synthetic para acessar os dados do layout pela viewCriada, vamos preparando o layout.

```
val transacao = transacoes[position]

viewCriada.transacao_valor.text = transacao.v

return viewCriada
}
```

```
val      val name = expression
var      var name = expression
v javaClass for T .. Class<Transacao> π
```

- Veja que não estamos conseguindo recuperar o atributo valor. Isso ocorre porque, ele está privado!



# Properties

- O Kotlin deixa transparente a implementação de um get e set, quando o suprimimos o modificador de acesso.
- Nós acessamos como se fosse um atributo publico, mas ele cria de maneira automática os métodos getters e setters.
- A isso damos o nome de **Property**

```
val valor : BigDecimal = valor  
val descricao : String = descricao  
val data : Calendar = data
```

# Simplificando o nosso modelo

- Como utilizamos os métodos get e set padrão, o Kotlin nos permite simplificar a declaração da classe:

```
class Transacao(val valor : BigDecimal,  
                val descricao : String,  
                val data : Calendar)
```

- Basta declara a property direto no construtor da classe.

# Construindo a View

- Agora, podemos acessar os nossos atributos de maneira segura e simples, graças ao Kotlin 😊

```
viewCriada.transacao_valor.text = transacao.valor.toString()
```

- Vamos continuar com as demais informações. A categoria é simples, pois é apenas uma String:

```
viewCriada.transacao_categoria.text = transacao.descricao
```

# Construindo a View

- Para a data, devemos utilizar o mesmo passo a passo do Java. Formatando a data através do objeto SimpleDateFormat do pacote java.text

```
val sdf = SimpleDateFormat(pattern: "dd/MM/yyyy")  
val dataFormatada = sdf.format(transacao.data.time)  
viewCriada.transacao_data.text = dataFormatada
```

- Hora de testar nosso programa!

# Resultado do novo layout



# Extension Function

Abrindo novos caminhos!

# Analizando o getView

- Veja que a formatação da data, não é de responsabilidade do método getView.
- Como resolver isso? Simples, faremos uma função na classe Adapter para isso =D

```
fun dataFormatada(data : Calendar) : String {  
    val sdf = SimpleDateFormat( pattern: "dd/MM/yyyy")  
    return sdf.format(data.time)  
}
```

# Analizando o getView

- Entretanto, isso também não é de responsabilidade do Adapter.
- E se, nós conseguimos criar uma função a mais na classe Calendar. Se ela formatasse para nós?
- Isso é possível através de um padrão que o Kotlin possui, chama-se **Extension Function**.
- Vamos conferir?



# Implementando Extension Function

- A função fica assim:

```
fun Calendar.dataFormatada() : String {  
    return SimpleDateFormat ( pattern: "dd/MM/yyyy")  
        .format (this.time)  
}
```

- E agora fica simples chamar ela no getView:

```
viewCriada.transacao_data.text = transacao.data.dataFormatada()
```

# Deixando ainda melhor

- Vamos aproveitar o paradigma funcional que o Kotlin nos permite e criar um arquivo de funções chamado de CalendarExtension, dentro do pacote extension nas raiz.

```
package br.com.fiap.financas.extension

import java.text.SimpleDateFormat
import java.util.Calendar

fun Calendar.dataFormatada() : String {
    return SimpleDateFormat(pattern: "dd/MM/yyyy")
        .format(this.time)
}
```

# Pronto! Agora tá Biito! =D



# Atenção!

- Este tipo de comportamento é muito **poderoso**, e permite que coloquemos ações em classes que não são nossas, o que é **incomum** no **Java**.
- Isso tende a ser **perigoso** dependendo da forma com que lidamos com isso.
- Se **definirmos comportamentos não esperados em funções**, por exemplo, e outro programador tiver que trabalhar com o código, ele pode acabar não conseguindo ou não entendendo o porquê daquilo.

# Atenção!

- Se para você **fizer muito sentido** estender uma classe para um comportamento que é muito comum em sua aplicação, aí sim a *Extension function* é recomendada.
- Caso contrário, **evite o máximo possível**, pois você poderá estar piorando seu código ao invés de melhorá-lo.

# Exercícios

Hora do descanso.

# Exercícios

- Crie uma calculadora, utilizando Kotlin.
- Para auxiliar no desenvolvimento do layout, utilize o seguinte livro (grátis):

<https://leanpub.com/google-android>



# Próximos Passos

O que veremos na próxima aula



# Na próxima aula...

- Enum
- Sobrecarga de Construtor
- Formatações de Números
- String templates
- Refatorando o código (Boas Práticas)



**Copyright © 2020**  
**Prof. Andrey Masiero**

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

*Do or do not. There is no try – Mestre Yoda*