Desenvolvimento Mobile

Introdução ao Banco de Dados com Kotlin

Prof. Dr. Marcelo Otone Aguiar

Universidade Federal do Espírito Santo - UFES

25 de Fevereiro de 2025

Conteúdo

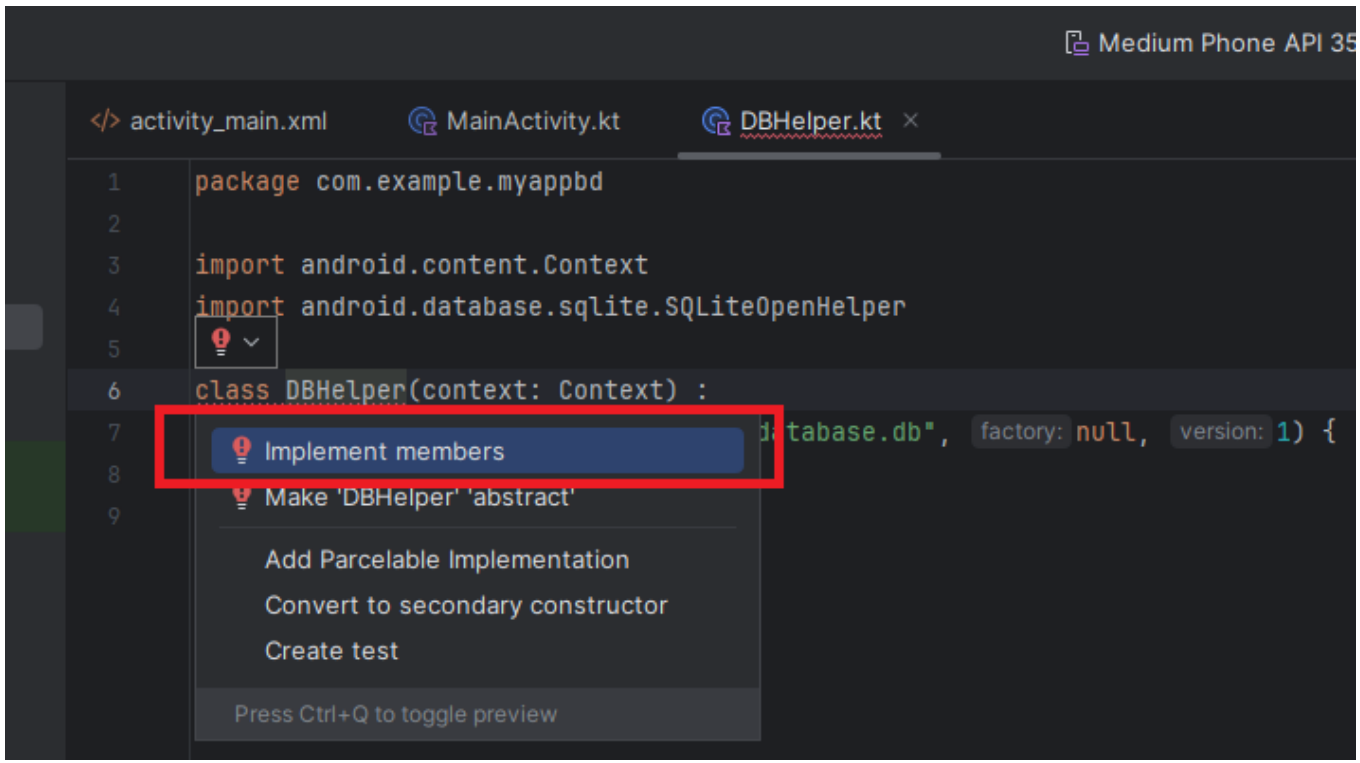
Definição inicial da classe DBHelper

- Vamos utilizar o banco de dados *SQLite*
- Para isso, vamos criar uma classe denominada de *DBHelper* para facilitar a implementação dos acessos ao banco de dados
- A classe deverá estender da classe *SQLiteOpenHelper*
- A implementação abaixo permitirá que, ao acessar os métodos do banco de dados, seja necessário informar apenas o contexto e demais informações, como o nome do banco de dados, serão tratados de forma fixa.

```
1 class DBHelper(context: Context) :
2     SQLiteOpenHelper(context, "database.db", null, 1) {
3
4 }
```

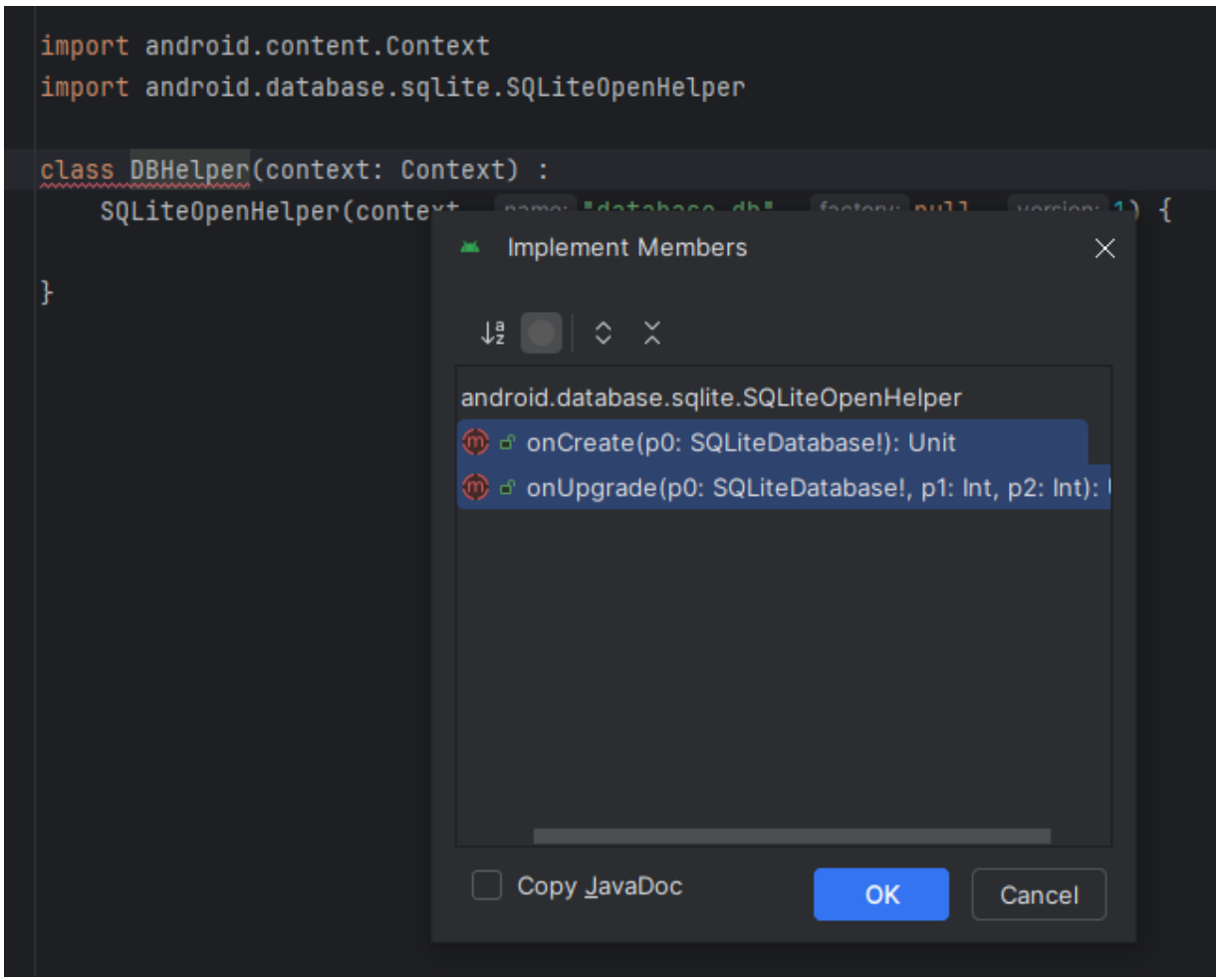
Implementando os métodos da classe *SQLiteOpenHelper*

Como a classe herda de *SQLiteOpenHelper*, alguns de seus métodos deverão ser implementados.



Implementando os métodos da classe *SQLiteOpenHelper*

Selecione os métodos da lista e pressione **OK**.



Implementando os métodos da classe *SQLiteOpenHelper*

Após pressionar **OK** o resultado obtido será similar ao código a seguir:

```
1  class DBHelper(context: Context) :
2      SQLiteOpenHelper(context, "database.db", null, 1) {
3
4      override fun onCreate(p0: SQLiteDatabase?) {
5          TODO("Not yet implemented")
6      }
7
8      override fun onUpgrade(p0: SQLiteDatabase?, p1: Int, p2: Int) {
9          TODO("Not yet implemented")
10     }
11
12 }
```

Exemplo: Tabela de Usuários

- Os códigos que serão apresentados a seguir são uma possibilidade de implementação dentre várias
- Além disso, a implementação supõe que os registros serão armazenados em uma tabela de usuários com a seguinte estrutura:

usuarios
id **INTEGER** *PRIMARY KEY*
nomeUsuario **TEXT**
senhaUsuario **TEXT**

Evento *onCreate()*

```
1  class DBHelper(context: Context) :
2      SQLiteOpenHelper(context, "database.db", null, 1) {
3
4      val sql = arrayOf(
5          "CREATE TABLE usuarios (id INTEGER PRIMARY KEY AUTOINCREMENT,
               nomeUsuario TEXT, senhaUsuario TEXT)",
6          "INSERT INTO usuarios (nomeUsuario, senhaUsuario) VALUES ('user', '123')
               ",
7          "INSERT INTO usuarios (nomeUsuario, senhaUsuario) VALUES ('user2',
               '123')"
8      )
9
10     override fun onCreate(db: SQLiteDatabase) {
11         sql.forEach {
12             db.execSQL(it)
13         }
14     }
```

Evento *onUpgrade()*

- Não vamos nos preocupar com o versionamento do banco de dados, sendo assim, o único argumento de entrada que vamos utilizar é **db**
- Desta forma, há apenas a necessidade de apagar a tabela e invocar o evento *create* para criar novamente

```
1 override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) {
2     db.execSQL("DROP TABLE usuarios")
3     onCreate(db)
4 }
```

Criando uma classe de entidade para *Usuario*

- No projeto, crie uma nova *Kotlin Class*
- Atribua a nova classe o nome da entidade desejada, que em nosso exemplo será **Usuario**
- Na classe, faça a implementação à seguir:

```
1  class Usuario(val id: Int = 0, val nomeUsuario: String = "", val senhaUsuario:
       String = "") {
2
3      override fun toString(): String {
4          return "Usuario(id=$id, nomeUsuario='$nomeUsuario', senhaUsuario='
               $senhaUsuario')"
5      }
6  }
```

Função para Retornar todos os Usuários

- A função a seguir retorna em um cursor uma lista com todos os registros da tabela
- Note que, o banco de dados foi iniciado com o comando *this.readableDatabase*, pois como faremos apenas consulta no banco, não é necessário abri-lo em modo de escrita.

```
1  fun selectAllUsuarios() : Cursor {
2      val db = this.readableDatabase
3      val cursor = db.rawQuery("SELECT * FROM usuarios", null)
4      db.close()
5      return cursor
6  }
```

Retornando os dados em um *ArrayList*

- O comando *cursor.moveToFirst()* posiciona o cursor no primeiro registro da tabela, isso é importante pois quando a tabela é aberta, o cursor está posicionado no último registro
- Para garantir que não haverá erros, as linhas 8-10 são responsáveis por obter o index das colunas

```
1  fun listSelectAllUsuarios(): ArrayList<Usuario> {
2      val db = this.readableDatabase
3      val cursor = db.rawQuery("SELECT * FROM usuarios", null)
4      val listaUsuarios: ArrayList<Usuario> = ArrayList()
5      if (cursor.count > 0) {
6          cursor.moveToFirst()
7          do {
8              val idIndex = cursor.getColumnIndex("id")
9              val nomeIndex = cursor.getColumnIndex("nomeUsuario")
10             val senhaIndex = cursor.getColumnIndex("senhaUsuario")
11             val id = cursor.getInt(idIndex)
12             val nomeUsuario = cursor.getString(nomeIndex)
13             val senhaUsuario = cursor.getString(senhaIndex)
14             listaUsuarios.add(Usuario(id, nomeUsuario, senhaUsuario))
15         } while (cursor.moveToNext())
16     }
17     db.close()
18     return listaUsuarios
19 }
```

Retornando os dados em um *ArrayList*

- A linha 11 obtém o *id* do registro atual
- As linhas 12 e 13 são responsáveis por obter os valores de nome e senha do usuário do registro atual
- A linha 14 é responsável por adcionar um objeto do usuário, instanciado a partir dos dados lidos no cursor atual, no *ArrayList* que foi criado na linha 4
- Por fim, o laço **while** será executado enquanto for possível mover para o próximo registro, preenchendo assim o *ArrayList* com todos os registros retornados na pesquisa

Carregando os dados em um *ListView*

- Adicione um ListView na **Activity** desejada
- Implemente as instruções no *onCreate()* da **Activity** desejada

```
<ListView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/List_view"/>
```

```
private lateinit var binding : ActivityMainBinding
private lateinit var adapter : ArrayAdapter<Usuario>

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    val db = DBHelper(this)
    val listaUsuarios = db.listSelectAllUsuarios()

    adapter = ArrayAdapter(this, android.R.layout.simple_list_item_1,
        listaUsuarios)
    binding.ListView.adapter = adapter
}
```

Função para Inserir Usuário

- A função a seguir retorna um *Long*, pois é o tipo retornado pela função *db.insert()*
- Caso a inserção tenha sucesso, o valor retornado é o *id* do registro inserido, em caso de erro o retorno será 0 ou negativo
- *contentValues* permitirá guardar de forma estruturada os dados que pretende-se armazenar na tabela

```
1  fun usuarioInsert(nome: String, senha: String) : Long {
2      val db = this.writableDatabase
3      val contentValues = ContentValues()
4      contentValues.put("nomeUsuario", nome)
5      contentValues.put("senhaUsuario", senha)
6      val res = db.insert("usuarios", null, contentValues)
7      db.close()
8      return res
9  }
```

Implementação do botão Inserir

- Incluir na **Activity** um *EditText* para cada campo
- Incluir um botão para a operação Inserir
- Implementar o código a seguir para o botão:

```
1  binding.buttonInserir.setOnClickListener{
2      val nomeUsuario = binding.editNomeUsuario.text.toString()
3      val senhaUsuario = binding.editSenhaUsuario.text.toString()
4
5      val res = db.usuarioInsert(nomeUsuario, senhaUsuario)
6      if (res > 0) {
7          Toast.makeText(applicationContext, "Sucesso ao inserir: $res", Toast.
               LENGTH_SHORT).show()
8          listaUsuarios.add(Usuario(res.toInt(), nomeUsuario, senhaUsuario))
9          adapter.notifyDataSetChanged()
10     }
11     else {
12         Toast.makeText(applicationContext, "Erro ao inserir: $res", Toast.
               LENGTH_SHORT).show()
13     }
14 }
```

Carregando dados do item selecionado na *ListView*

- Para implementarmos as operações de alteração e exclusão, vamos antes ajustar o *ListView* para que, ao selecionar um item, os seus dados sejam carregados nos componentes *TextView* e *EditText*
- Para isso, declare uma variável global para guardar a posição e adicione o evento *setOnItemClickListener* na **Activity** conforme o código a seguir:

```
1  private var pos : Int = -1
2
3  binding.ListView.setOnItemClickListener { _, _, position, _ ->
4      binding.textId.setText("ID: ${listaUsuarios[position].id}")
5      binding.editNomeUsuario.setText(listaUsuarios[position].nomeUsuario)
6      binding.editSenhaUsuario.setText(listaUsuarios[position].senhaUsuario)
7      pos = position
8  }
```

Função para Atualizar dados do Usuário

- Muito similar à função de inserir usuário, contudo, nesse caso, será necessário também o *id* como argumento de entrada
- O retorno nesse caso é *Int*, pois é o tipo retornado pela função *db.update()*
- O valor retornado será o número de linhas afetadas ou negativo em caso de erro no processamento
- O argumento *whereArgs* da função *db.update()* requer uma entrada do tipo *array*, pois deve permitir a entrada de mais de um argumento

```
1  fun usuarioUpdate(id: Int, nome: String, senha: String) : Int {
2      val db = this.writableDatabase
3      val contentValues = ContentValues()
4      contentValues.put("nomeUsuario", nome)
5      contentValues.put("senhaUsuario", senha)
6      val res = db.update("usuarios", contentValues, "id=?" , arrayOf(id.toString
           ()))
7      db.close()
8      return res
9  }
```

Implementação do botão Atualizar

- Implementar o código a seguir no botão Atualizar

```
1  binding.buttonAtualizar.setOnClickListener {
2      if (pos >= 0) {
3          val id = listaUsuarios[pos].id
4          val nomeUsuario = binding.editNomeUsuario.text.toString()
5          val senhaUsuario = binding.editSenhaUsuario.text.toString()
6          val res = db.usuarioUpdate(id, nomeUsuario, senhaUsuario)
7          if (res > 0) {
8              Toast.makeText(applicationContext, "Sucesso ao atualizar: $res",
                                                  Toast.LENGTH_SHORT).show()
9              listaUsuarios.set(pos, Usuario(id, nomeUsuario, senhaUsuario))
10             adapter.notifyDataSetChanged()
11         } else {
12             Toast.makeText(applicationContext, "Erro ao atualizar: $res",
                                          Toast.LENGTH_SHORT).show()
13         }
14     }
15 }
```

Função para Excluir o Usuário

- Para excluir um registro do banco não será necessário utilizar o *contentValues*, basta o *id* do registro como argumento de entrada para que a função *db.delete()* seja capaz de encontrar o registro
- O valor retornado será o número de linhas afetadas ou negativo em caso de erro no processamento

```
1  fun usuarioDelete(id: Int) : Int {
2      val db = this.writableDatabase
3      val res = db.delete("usuarios", "id=?", arrayOf(id.toString()))
4      db.close()
5      return res
6  }
```

Implementação do botão Excluir

- Implementar o código a seguir no botão Excluir

```
1  binding.buttonExcluir.setOnClickListener {
2      if (pos >= 0) {
3          val id = listaUsuarios[pos].id
4          val res = db.usuarioDelete(id)
5          if (res > 0) {
6              Toast.makeText(applicationContext, "Sucesso ao Excluir: $res", Toast
                   .LENGTH_SHORT).show()
7              listaUsuarios.removeAt(pos)
8              adapter.notifyDataSetChanged()
9          } else {
10             Toast.makeText(applicationContext, "Erro ao Excluir: $res", Toast.
                   LENGTH_SHORT).show()
11         }
12     }
13 }
```

Função para retornar um usuário

- A função a seguir retorna um *cursor* do registro do usuário a partir do *id* informado no argumento de entrada

```
1  fun selectUsuarioByID(id : Int): Cursor {
2      val db = this.readableDatabase
3      val cursor = db.rawQuery("SELECT * FROM usuarios WHERE id = ?", arrayOf(id.
           toString()))
4      db.close()
5      return cursor
6  }
```

Função para retornar um *objeto* do usuário

- Para retornar um *objeto*, o processo é similar ao realizado para retornar um **ArrayList**

```
1  fun selectObjectUsuarioByID(id : Int): Usuario {
2      val db = this.readableDatabase
3      val cursor = db.rawQuery("SELECT * FROM usuarios WHERE id = ?", arrayOf(id.
           toString()))
4      var usuario = Usuario()
5      if (cursor.count == 1) {
6          cursor.moveToFirst()
7          val idIndex = cursor.getColumnIndex("id")
8          val nomeIndex = cursor.getColumnIndex("nomeUsuario")
9          val senhaIndex = cursor.getColumnIndex("senhaUsuario")
10         val id = cursor.getInt(idIndex)
11         val nomeUsuario = cursor.getString(nomeIndex)
12         val senhaUsuario = cursor.getString(senhaIndex)
13         usuario = Usuario(id, nomeUsuario, senhaUsuario)
14     }
15     db.close()
16     return usuario
17 }
```

Atualizando para *RecyclerView*

- Para atualizarmos de *ListView* para *RecyclerView* será necessário executarmos os seguintes passos:
  - Criar uma classe **ListAdapter** para a entidade afim de estabelecer a associação com o adapter do *RecyclerView*
  - Criar uma classe **ViewHolder** para a entidade que permitirá definir as informações disponíveis na *View*
  - Criar uma classe **OnClickListener** para a entidade que permitirá associar a entidade com o evento *OnClick* do *RecyclerView*
  - Criar um **Resource File** para o **Layout** que será associado ao *RecyclerView*
  - Por fim, substituir o *ListView* pelo *RecyclerView* na **Activity**

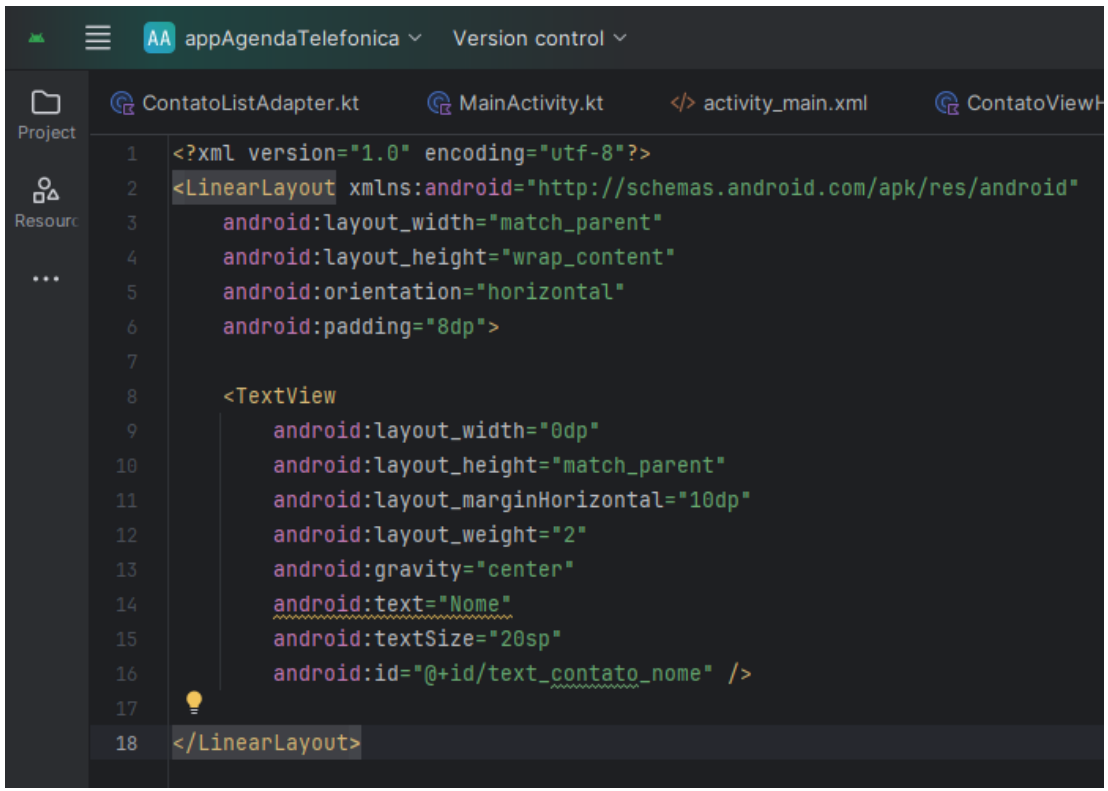
Classe *OnClickListener*

- Em nosso exemplo hipotético a nossa entidade é a classe **ContatoModel** que possui informações de contato da agenda.
- Dessa forma, o nome atribuído à classe **OnClickListener** foi: *ContatoOnClickListener*

```
1  package com.example.appagendatelefonica.adapter.listener
2  import com.example.appagendatelefonica.Model.ContatoModel
3
4  class ContatoOnClickListener(val clickListener: (contato : ContatoModel) -> Unit
       ) {
5
6      fun onClick(contato: ContatoModel) = clickListener
7
8  }
```

Resource File para o **Layout**

Ao *Resource File* foi atribuído o nome: **rowcontato**



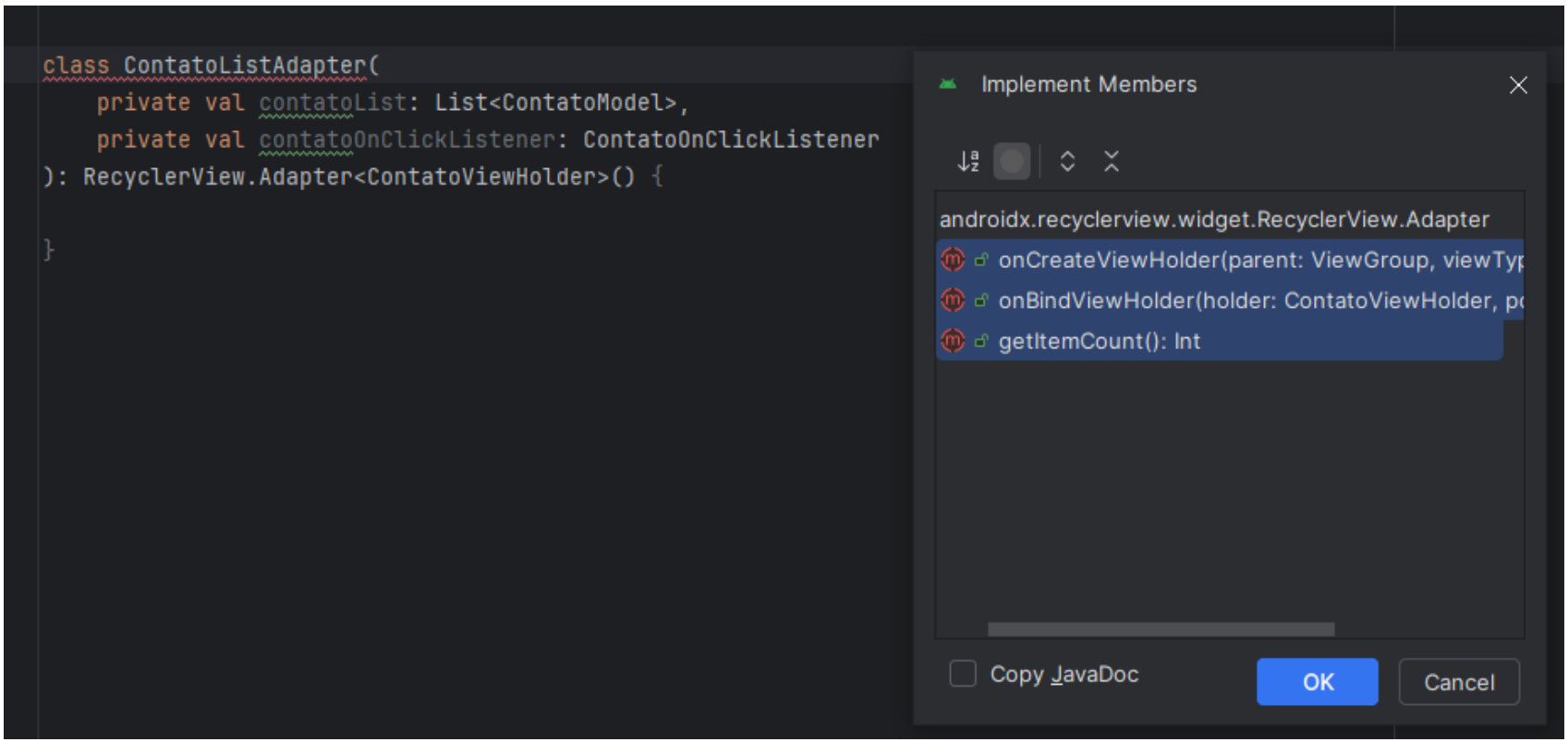
Classe **ViewHolder**

- O nome atribuído à classe **ViewHolder** foi: *ContatoViewHolder*

```
1  package com.example.appagendatelefonica.adapter.viewholder
2
3  import android.view.View
4  import android.widget.TextView
5  import androidx.recyclerview.widget.RecyclerView
6  import com.example.appagendatelefonica.R
7
8  class ContatoViewHolder(view: View): RecyclerView.ViewHolder(view) {
9
10     val textoNome: TextView = view.findViewById(R.id.text_contato_nome)
11 }
```

Classe **ListAdapter**

O nome atribuído à classe **ListAdapter** foi: *ContatoListAdapter*



Classe **ListAdapter**

```kotlin
package com.example.appagendatelefonica.adapter

class ContatoListAdapter(
    private val contatoList: List<ContatoModel>,
    private val contatoOnClickListener: ContatoOnClickListener
): RecyclerView.Adapter<ContatoViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
        ContatoViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.
            row_contato, parent, false)
        return ContatoViewHolder(view)
    }

    override fun getItemCount(): Int {
        return contatoList.size
    }

    override fun onBindViewHolder(holder: ContatoViewHolder, position: Int) {
        val contato = contatoList[position]
        holder.textoNome.text = contato.nome

        holder.itemView.setOnClickListener{
            contatoOnClickListener.clickListener(contato)
        }
    }
}
```

Substituir o *ListView* pelo *RecyclerView* na **Activity**

- O primeiro passo é comentar e/ou remover o código relativo ao *setOnItemClickListener* do **ListView**
- O segundo passo é ajustar o *Adapter* para o novo **Adapter** criado para o *RecyclerView*

```
1  //private lateinit var adapter : ArrayAdapter<ContatoModel>
2  private lateinit var adapter: ContatoListAdapter
3  .
4  .
5  .
6  override fun onCreate(savedInstanceState: Bundle?) {
7  .
8  .
9
10     /**binding.listViewContatos.setOnItemClickListener { _, _, position, _ ->
11         val intent = Intent(applicationContext, ContatoDetailActivity::class.
               java)
12         intent.putExtra("id", contatosList[position].id)
13         result.launch(intent)
14     }**/
```

Substituir o *ListView* pelo *RecyclerView* na **Activity**

- Substituir o *ListView* pelo *RecyclerView* no **xml** da **Activity**

```
<androidx.recyclerview.widget.RecyclerView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_above="@id/button_add"
    android:layout_alignParentTop="true"
    android:id="@+id/recyclerViewContatos" />

<!--<ListView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/list_view_contatos"
    android:layout_above="@+id/button_add"
    android:layout_alignParentTop="true"/>
-->
<com.google.android.material.floatingactionbutton.Float
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

Substituir o *ListView* pelo *RecyclerView* na **Activity**

- Associar o *layoutManager* do *RecyclerView* no *OnCreate()* da **Activity**
- Ajustar a função responsável pela atualização da lista

```
1  override fun onCreate(savedInstanceState: Bundle?) {
2      .
3      .
4      .
5      binding.recyclerViewContatos.layoutManager = LinearLayoutManager(
           applicationContext)
6      .
7      .
8      .
9  }
10
11 private fun loadList() {
12     contatosList = dbHelper.getAllContato()
13     adapter = ContatoListAdapter(contatosList, ContatoOnClickListener { contato
           ->
14         val intent = Intent(applicationContext, ContatoDetailActivity::class.
               java)
15         intent.putExtra("id", contato.id)
16         result.launch(intent)
17     } )
18
19     binding.recyclerViewContatos.adapter = adapter
20 }
```