# 1 App lista de tarefas LAYOUT

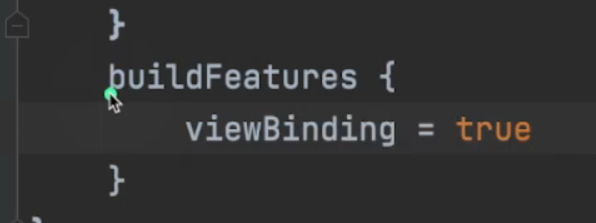
## 1 definindo as cores do aplicativo

<!-- Cores customizadas -->  
  
<color name="primaria">#FF5617</color>  
<color name="primaria\_varicao">#E34D0B</color>  
  
  
<color name="secundaria">#FF6F3A</color>  
<color name="secundaria\_varocao">#FF8051</color>  
  
  
<color name="fundo">#EFEFEF</color>

## 2 definindo o tema

<!-- Customizando o tema. -->  
  
  
<item name="colorPrimary">@color/primaria</item>  
<item name="colorPrimaryVariant">@color/primaria\_varicao</item>  
  
  
<item name="colorSecondary">@color/secundaria</item>  
<item name="colorSecondaryVariant">@color/secundaria\_variacao</item>  
  
<item name="android:windowBackground">@color/fundo</item>

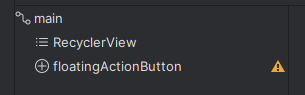
## 3 adicionando viewBinding

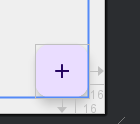


## 4 criando nosso layout

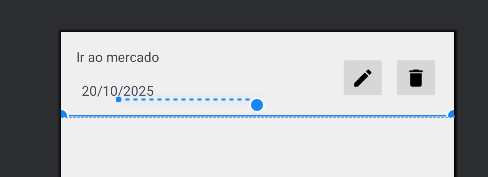
1 primeiro criamos nosso recyclerView

2 vamos adicionar o button float com o ícone de adicionar





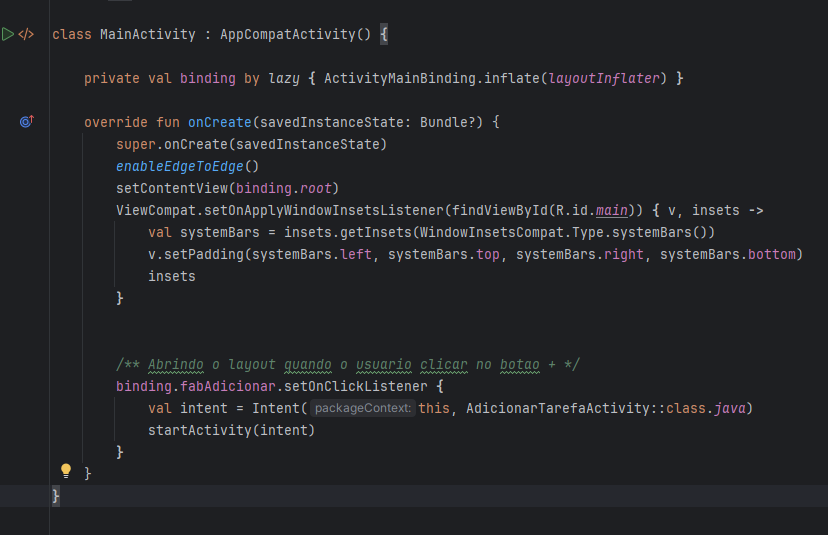
3 criando layout para o recyclerView



4 criando o layout de adicionar tarefa

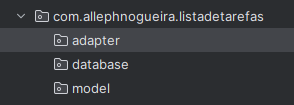


## 5 criando uma ação para quando o usuário clicar no botão ele navegar para a nova activity



# 2 Configurando o banco de dados

## 1 organizando o projeto criando os pacotes



## 2 criando a classe DatabaseHelper

class DatabaseHelper(context: Context) : SQLiteOpenHelper(  
 context, NOME\_BANCO\_DADOS, null, VERSAO  
) {  
  
 companion object {  
 const val NOME\_BANCO\_DADOS = "ListaTarefas.db"  
 const val VERSAO = 1  
  
 const val NOME\_TABELA\_TAREFAS = "tarefas"  
 const val COLUNA\_ID\_TAREFA = "id\_tarefa"  
 const val COLUNA\_DESCRICAO = "descricao"  
 const val COLUNA\_DATA\_CADASTRO = "data\_cadastro"  
  
 }  
  
  
 override fun onCreate(db: SQLiteDatabase?) {  
  
 val sql = "CREATE TABLE IF NOT EXISTS $NOME\_TABELA\_TAREFAS(" +  
 "$COLUNA\_ID\_TAREFA INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +  
 "$COLUNA\_DESCRICAO VARCHAR(70)," +  
 "$COLUNA\_DATA\_CADASTRO DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP" +  
 ");"  
  
 try {  
 db?.execSQL(sql)  
 Log.i("info\_db", "Sucesso ao criar tabela!")  
 }catch (e: Exception) {  
 e.printStackTrace()  
 Log.i("info\_db", "Erro ao criar tabela!")  
 }  
  
 }  
  
  
 override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {  
 *TODO*("Not yet implemented")  
 }  
}

## 3 criando a classe tarefa (model)

Aqui vai ser as colunas que estamos criando para o banco de dados

package com.allephnogueira.listadetarefas.model  
  
data class Tarefa (  
 val idTarefa: Int,  
 val descricao: String,  
 val dataCadastro: String  
)

## 4 criando a interface para criar os métodos

Primeiro cria a interface

package com.allephnogueira.listadetarefas.database  
  
import com.allephnogueira.listadetarefas.model.Tarefa  
  
interface ITarefasDAO {  
  
 fun salvar( tarefa: Tarefa ) : Boolean  
 fun atualizar( tarefa: Tarefa ) : Boolean  
 fun remover ( idTarefa: Int ) : Boolean  
 fun listarTarefa () : List<Tarefa>  
}

# 3 criando o tarefaDAO INSERINDO

package com.allephnogueira.listadetarefas.database  
  
import android.content.ContentValues  
import android.content.Context  
import android.util.Log  
import com.allephnogueira.listadetarefas.model.Tarefa  
  
class TarefaDAO(context: Context) : ITarefasDAO {  
  
 private val escrita = DatabaseHelper(context).*writableDatabase* // Fazer escrita (UPDATE, INSERT, DELETE)  
 private val leitura = DatabaseHelper(context).*readableDatabase* // leitura (SELECT)  
  
  
 override fun salvar(tarefa: Tarefa): Boolean {  
  
 val valores = ContentValues()  
 valores.put("${DatabaseHelper.COLUNA\_DESCRICAO}", tarefa.descricao)  
  
  
 try {  
 escrita.insert(  
 DatabaseHelper.NOME\_TABELA\_TAREFAS,  
 null,  
 valores  
 )  
 Log.i("info\_db", "Sucesso ao criar tabela!")  
 }catch (e: Exception) {  
 e.printStackTrace()  
 Log.i("info\_db", "Erro ao criar tabela!")  
 return false  
 }  
  
 return true  
 }  
  
 override fun atualizar(tarefa: Tarefa): Boolean {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun remover(idTarefa: Int): Boolean {  
 *TODO*("Not yet implemented")  
 }  
  
 override fun listarTarefa(): List<Tarefa> {  
 *TODO*("Not yet implemented")  
 }  
}

## 6 capturando a tarefa que o usuário escreveu para salvar.

Vamos la na Activity que criamos e vamos capturar oque o usuário escreveu.

binding.btnSalvar.setOnClickListener **{** val nomeTarefa = binding.editTarefa.*text*.toString()  
 salvarTarefa(nomeTarefa)  
  
 **}**}  
  
private fun salvarTarefa(nomeTarefa: String) {  
  
 if (nomeTarefa.*isNotEmpty*()) {  
 val tarefaDAO = TarefaDAO(this)  
  
 val tarefa = Tarefa(  
 -1,  
 nomeTarefa,  
 "DEFAULT"  
 )  
  
 tarefaDAO.salvar(tarefa)  
 Toast.makeText(this, "Tarefa salva com sucesso!", Toast.*LENGTH\_LONG*).show()  
 finish()  
 }else {  
 Toast.makeText(this, "Adicione uma tarefa", Toast.*LENGTH\_SHORT*).show()  
 }  
}

# 4 Listando as tarefas

Nossa tabela tem esses valores, com essa formatação de data e hora



Tem uma forma de retornar a data da forma que a gente quer.

Atenção tem as funções para formatar no google/IA podemos sempre pedir essas funções para formatar esses dados.

Aqui formatamos para ficar dessa forma



Usamos essa função que podemos achar no código do SQLite

SELECT

Precisamos colocar a função.

Precisamos passar o formatado que vai ser formatado

Precisamos também passar a coluna que vai ser formatada.

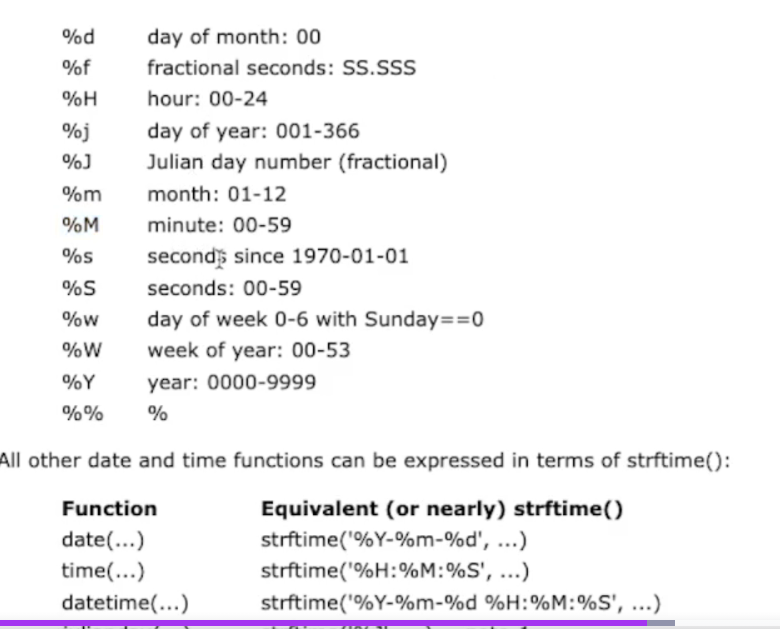
id\_tarefa, descricao,

**strftime**('%d-%m-%Y', data\_cadastro) AS data\_formatada

FROM tarefas;

Também alteramos o nome

Aqui temos mais parâmetros para consultar



## 1 adicionando os códigos dentro do método

Esse método esta dentro do TarefaDAO que é responsável pelos dados do banco de dados

override fun listarTarefa(): List<Tarefa> {  
 val listaTarefas = *mutableListOf*<Tarefa>()  
  
 val comandoSQL = "SELECT ${DatabaseHelper.COLUNA\_ID\_TAREFA}, ${DatabaseHelper.COLUNA\_DESCRICAO}, strftime('%d/%m/%Y', ${DatabaseHelper.COLUNA\_DATA\_CADASTRO}) AS ${DatabaseHelper.COLUNA\_DATA\_CADASTRO} FROM ${DatabaseHelper.NOME\_TABELA\_TAREFAS};"  
  
 val cursor = leitura.rawQuery(comandoSQL, null)  
  
 val indiceID = cursor.getColumnIndex(DatabaseHelper.COLUNA\_ID\_TAREFA)  
 val indiceDescricao = cursor.getColumnIndex(DatabaseHelper.COLUNA\_DESCRICAO)  
 val indiceData = cursor.getColumnIndex(DatabaseHelper.COLUNA\_DATA\_CADASTRO)  
  
 while (cursor.moveToNext()) {  
 val colunaIdTarefa = cursor.getInt(indiceID)  
 val colunaDescricao = cursor.getString(indiceDescricao)  
 val colunaDataCadastro = cursor.getString(indiceData)  
  
 listaTarefas.add(  
 Tarefa(colunaIdTarefa, colunaDescricao, colunaDataCadastro )  
 )  
 }  
   
  
 return listaTarefas  
}

## Adicionando os códigos dentro do Main Activity que é onde vai ser executado o código sempre que o usuário abrir o app

Aqui fizemos o onStart porque ele é sempre chamado quando o usuário volta para a tela.

Antes no onCreate estávamos com problemas, porque ele executa apenas uma vez, imagina que o usuário adiciona uma nova tarefa, então ele agora com o onStart toda vez que o usuário adicionar uma nova tarefa, ele vai voltar automaticamente para a tela inicial e com isso vai ser recarregado os códigos novamente.

class MainActivity : AppCompatActivity() {  
  
 private val binding by *lazy* **{** ActivityMainBinding.inflate(*layoutInflater*) **}** private var listaDeTarefas = *emptyList*<Tarefa>() // Criando uma lista vazia

override fun onStart() {  
 */\*\* Lembra que o onCreate ele executa apenas uma vez  
 \* Então criamos o onStart que sempre quando o usuario sai e volta ele executa novamente, assim sempre atualizando a lista de tarefas  
 \* Para relembrar o onStart ele é chamado toda vez que o usuario sai da tela  
 \* onStart é o metodo que geralmente a gente usa para recuperar dados.  
 \*/* super.onStart()  
  
 val tarefaDAO = TarefaDAO(this)  
 listaDeTarefas = tarefaDAO.listarTarefa()  
 listaDeTarefas.*forEach* **{** tarefas **->** Log.i("info\_db", "ID: ${tarefas.idTarefa} - Descricão: ${tarefas.descricao} - Data: ${tarefas.dataCadastro}\n")  
 **}**}

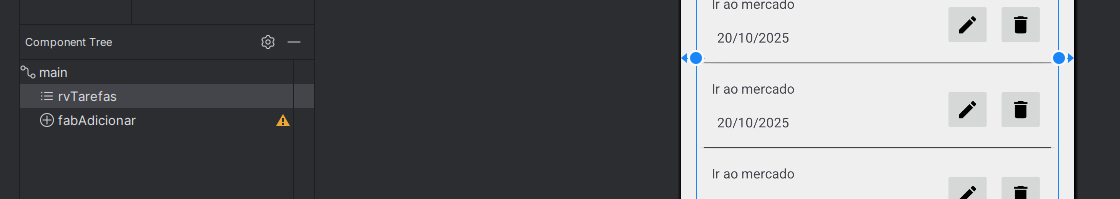
# 5 listando as tarefas dentro do RecyclerView

## 1 construindo o recyclerView

Vamos construir dentro do método onCreate

// Construir o recyclerView  
 tarefaAdapter = TarefaAdapter()  
  
 binding.rvTarefas.*adapter* = tarefaAdapter  
  
 binding.rvTarefas.*layoutManager* = LinearLayoutManager(this)  
  
  
}

## Vamos da uma ID para o RecyclerView



## Criando método para atualizar o recyclerView sempre que entrar mais dados

private fun atualizarListaTarefas(){  
 val tarefaDAO = TarefaDAO(this)  
 listaDeTarefas = tarefaDAO.listarTarefa()  
 tarefaAdapter?.adicionarLista(listaDeTarefas)  
}  
  
override fun onStart() {  
 */\*\* Lembra que o onCreate ele executa apenas uma vez  
 \* Então criamos o onStart que sempre quando o usuario sai e volta ele executa novamente, assim sempre atualizando a lista de tarefas  
 \* Para relembrar o onStart ele é chamado toda vez que o usuario sai da tela  
 \* onStart é o metodo que geralmente a gente usa para recuperar dados.  
 \*/* super.onStart()  
 // Toda vez que a gente sair e entrar na pagina vamos chamar esse metodo passando os novos dados e o RecyclerView vai atualizar.  
 atualizarListaTarefas()  
  
  
  
}

## Criando o adapter e passando os dados para exibir na tela

package com.allephnogueira.listadetarefas.adapter  
  
import android.view.LayoutInflater  
import android.view.ViewGroup  
import androidx.recyclerview.widget.RecyclerView  
import com.allephnogueira.listadetarefas.databinding.ItemTarefaBinding  
import com.allephnogueira.listadetarefas.model.Tarefa  
  
class TarefaAdapter() : RecyclerView.Adapter<TarefaAdapter.TarefaViewHolder>() {  
  
 private var listaTarefas: List<Tarefa> = *emptyList*()  
  
 fun adicionarLista(lista: List<Tarefa>){  
 this.listaTarefas = lista  
 notifyDataSetChanged() // Esse metodo serve para notificar o recyclerView para ele recarregar os dados.  
 }  
  
  
 inner class TarefaViewHolder(itemBinding: ItemTarefaBinding) :  
 RecyclerView.ViewHolder(itemBinding.*root*) {  
 private val binding: ItemTarefaBinding  
  
 init {  
 binding = itemBinding  
 }  
  
 fun bind(tarefa: Tarefa) {  
 binding.textDescricao.*text* = tarefa.descricao  
 binding.textData.*text* = tarefa.dataCadastro  
 }  
 }  
  
 override fun onCreateViewHolder(  
 parent: ViewGroup,  
 viewType: Int  
 ): TarefaAdapter.TarefaViewHolder {  
 val layoutInflater = LayoutInflater.from(parent.*context*)  
 val itemTarefaBinding = ItemTarefaBinding.inflate(  
 layoutInflater, parent, false  
 )  
 return TarefaViewHolder(itemTarefaBinding)  
 }  
  
  
  
 override fun onBindViewHolder(holder: TarefaAdapter.TarefaViewHolder, position: Int) {  
 val tarefa = listaTarefas[position]  
 holder.bind(tarefa)  
 }  
  
  
  
 override fun getItemCount(): Int {  
 return listaTarefas.size  
 }  
}

# 6 removendo tarefas

Quando o usuário clicar no botão de remover, vamos exibir uma confirmação para o usuário.

## 1 passando uma ação para o adapter

class TarefaAdapter(  
 val onClickExcluir: (Int) -> Unit // A entrada de dados vai ser uma função que recebe um tipo Inteiro e nao retorna nada (Unit)  
) : RecyclerView.Adapter<TarefaAdapter.TarefaViewHolder>() {

## 2 chamando o método que vai confirmar a exclusão

Dentro do nosso MainActivity

Quando clicar no botão excluir

Ele vai chamar o método confirmarExclusao(id)

Vamos passar o ID

Dentro do metodo criamos um AlertDialog para confirmar realmente se o usuário quer excluir.

// Construir o recyclerView  
 tarefaAdapter = TarefaAdapter(  
 */\*\* Aqui estamos passando a função que vamos criar, passando o ID do item que queremos remover  
 \*  
 \*/* **{** id **->** confirmarExclusao(id) **}** )  
  
 binding.rvTarefas.*adapter* = tarefaAdapter  
  
 binding.rvTarefas.*layoutManager* = LinearLayoutManager(this)  
  
  
}  
  
private fun confirmarExclusao(id: Int) {  
 */\*\* Aqui dentro vamos criar um Alert Dialog para a confirmação  
 \*  
 \*/* val alertDialog = AlertDialog.Builder(this)  
 alertDialog.setTitle("Confirmar exclusão?")  
 alertDialog.setMessage("Deseja realmente excluir a tarefa?")  
  
 */\*\*Aqui temos 2 valores que não vamos utilizar, então para eles nao ficar ocupando espaço na memoria colocamos o \_  
 \* Aqui estamos definindo um texto e a funçao lambda para quando o usuario clicar no sim \*/* alertDialog.setPositiveButton("Sim") **{** \_, \_ **->  
  
 }** alertDialog.setNegativeButton("Não") **{** \_, \_ **->}** alertDialog  
 .create()  
 .show()  
  
}

## 3 fazendo a exclusao dentro do TarefaAdapter

Já estamos recebendo os dados, então vamos dentro do bind e vamos configurar

fun bind(tarefa: Tarefa) {  
 binding.textDescricao.*text* = tarefa.descricao  
 binding.textData.*text* = tarefa.dataCadastro  
  
 binding.btnExcluir.setOnClickListener **{** onClickExcluir(tarefa.idTarefa)  
 **}**}

Agora sim quando clicamos em excluir ele aparece o alertaDialog, mas nada acontece porque não configuramos la dentro do SIM ou NÃO

## 4 configurando a confirmação no alertDialog

alertDialog.setPositiveButton("Sim") **{** \_, \_ **->** val tarefaDAO = TarefaDAO(this)  
 tarefaDAO.remover(id) // Aqui vamos passar o id da tarefa que recebemos como parametro.  
**}**

Agora sim estamos passando o ID da tarefa que estamos pegando do recyclerView

## 5 configurando o método remover no tarefaDAO

override fun remover(idTarefa: Int): Boolean {  
  
  
  
 val args = *arrayOf*(idTarefa.toString())  
  
 try {  
 escrita.delete(  
 */\*\* Vai remover da TABELA  
 \* Onde id\_tarefa for igual a ...  
 \* (args é onde esta o ID ele esta recebendo do MainActivity)  
 \*  
 \* Agora apos remover precisamos usar o metodo para atualizar o recyclerView se tudo deu certo.  
 \*/* DatabaseHelper.NOME\_TABELA\_TAREFAS,  
 "${DatabaseHelper.COLUNA\_ID\_TAREFA} = ?",  
 args  
 )  
 Log.i("info\_db", "Sucesso ao remover tarefa!")  
 }catch (e: Exception) {  
 e.printStackTrace()  
 Log.i("info\_db", "Erro ao remover tarefa!")  
 return false  
 }  
  
 return true  
}

## 6 colocando o toast para confirmar que a tarefa foi removida

Colocamos para exibir o toast apenas se o usuário clicar em SIM dentro do AlertDialog

Esse método esta dentro do MainActivity

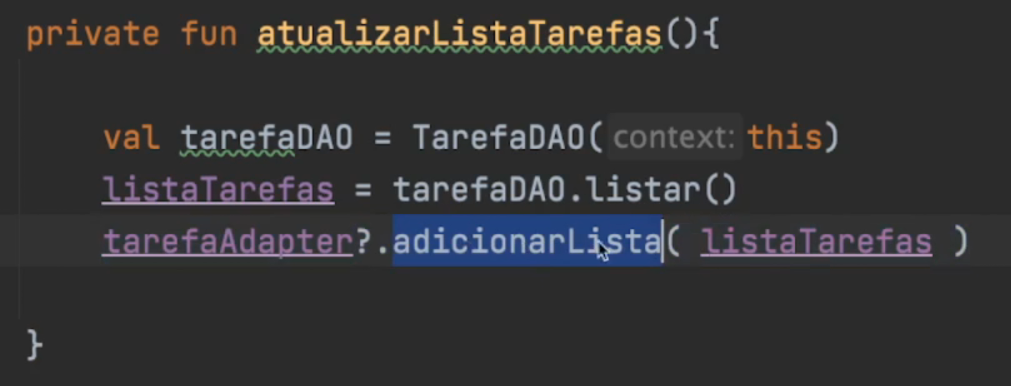
alertDialog.setPositiveButton("Sim") **{** \_, \_ **->** val tarefaDAO = TarefaDAO(this)  
 tarefaDAO.remover(id) // Aqui vamos passar o id da tarefa que recebemos como parametro.  
  
 if (tarefaDAO.remover( id )) {  
 */\*\* Lembrar que o tarefa remover vai retornar um verdadeiro ou falso  
 \* Se retornar verdadeiro caimos nesse toast aqui  
 \* Se nao caimos no ELSE  
 \*/* Toast.makeText(this, "Tarefa removida", Toast.*LENGTH\_SHORT*).show()  
 }else {  
 Toast.makeText(this, "Falha ao remover", Toast.*LENGTH\_SHORT*).show()  
 }  
**}**

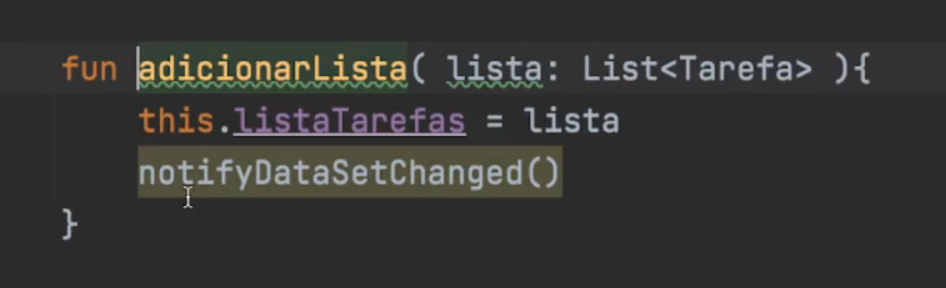
# 7 atualizando o recyclerView com as novas tarefas

Quando ele clicar em sim e tiver tudo certo para remover, vamos chamar o método para atualizar a lista de tarefas

alertDialog.setPositiveButton("Sim") **{** \_, \_ **->** val tarefaDAO = TarefaDAO(this)  
 tarefaDAO.remover(id) // Aqui vamos passar o id da tarefa que recebemos como parametro.  
  
 if (tarefaDAO.remover( id )) {  
 */\*\* Lembrar que o tarefa remover vai retornar um verdadeiro ou falso  
 \* Se retornar verdadeiro caimos nesse toast aqui  
 \* Se nao caimos no ELSE  
 \*/* Toast.makeText(this, "Tarefa removida", Toast.*LENGTH\_SHORT*).show()  
 atualizarListaTarefas() // Atualizando o RecyclerView para recarregar lista de tarefas  
  
 }else {  
 Toast.makeText(this, "Falha ao remover", Toast.*LENGTH\_SHORT*).show()  
 }  
**}**

Resumindo esse método



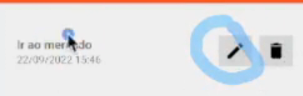


Ele vai listar as tarefas novamente e adicionar a lista de tarefas novamente e após adicionar a nova lista ele vai chamar o método para atualizar o recyclerView

# 8 atualizar uma tarefa

Quando o usuário clicar em atualizar a tarefa vamos enviar ela para a mesma tela de adicionar, mas na verdade ele vai esta atualizando a tarefa.

Vamos também recuperar a tarefa que o usuário esta adicionando e colocar ela la para o usuário poder apagar.





Entao vamos começar fazendo a ação de ir para a outra tela

## 1 criando a ação de ir para a activity

Primeiro vamos no nosso adapter e vamos passar uma função que ele vai receber

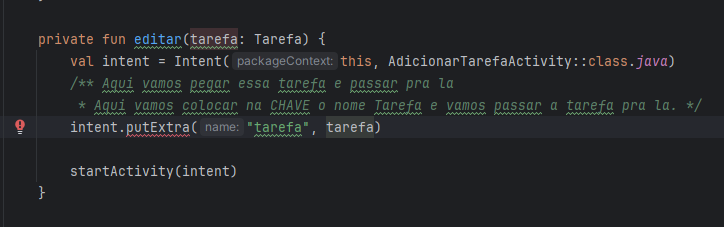
val onClickEditar: (Tarefa) -> Unit

Onde estamos construindo o nosso layout vamos passar a função lambda e vamos criar um método para fazer a função

// Construir o recyclerView  
tarefaAdapter = TarefaAdapter(  
 */\*\* Aqui estamos passando a função que vamos criar, passando o ID do item que queremos remover  
 \*  
 \*/* **{** id **->** confirmarExclusao(id) **}**,  
 */\*\* Aqui vamos fazer a funçao de editar uma tarefa, dentro do metodo que vamos fazer os detalhes \*/* **{**tarefa **->** editar(tarefa)**}** */\*\* Abrindo o layout para quando o usuario clicar em atualizar \*/*

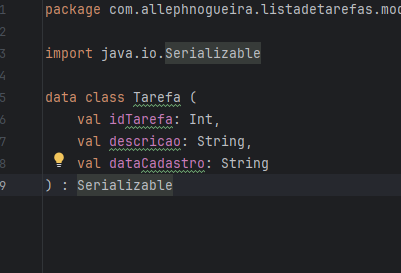
*Aqui no parâmetro eu passo a tarefa que acabei de utilizar*)

## 2 Metodo da tarefa



Aqui estamos com esse erro porque ele é um objeto customizado, então vamos precisar serializar ele.

Vamos la na nossa classe Tarefa e vamos implementar uma interface serializable



Ficando dessa forma, ai você pode reparar que já saiu o erro.

Lembrando que a recomendação é usar o parsebal pq ele tem o desempenho um pouco melhor.

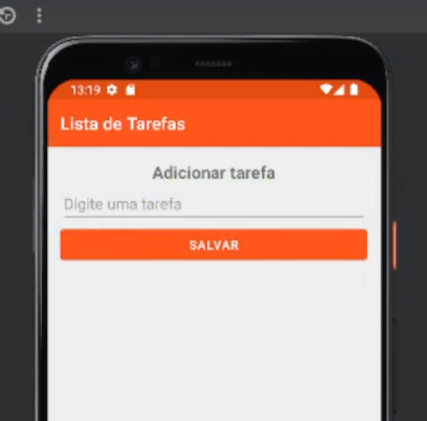
Resumindo agora sim vamos abrir uma nova activity passando a chave tarefa com os valores.

## 3 ajustando a função de atualizar a tarefa no adapter

fun bind(tarefa: Tarefa) {  
 binding.textDescricao.*text* = tarefa.descricao  
 binding.textData.*text* = tarefa.dataCadastro  
  
 binding.btnExcluir.setOnClickListener **{** onClickExcluir(tarefa.idTarefa)  
 **}** binding.btnEditar.setOnClickListener **{** onClickEditar(tarefa)  
 **}**

Aqui vamos passar uma tarefa completa.

Agora ele já esta abrindo uma nova tarefa quando clicamos no botão editar, mas vamos precisar recuperar essa tarefa que clicamos em cima

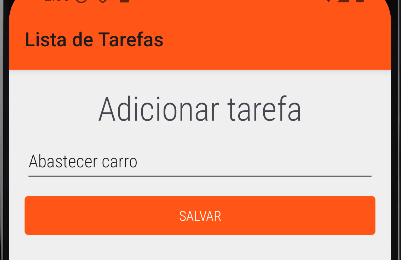


## 4 recuperando a tarefa para editar

Isso só vai acontecer quando for uma edição, fizemos esse teste exatamente para isso.

override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(binding.*root*)  
  
 // Recuperar tarefa que foi passada  
 val bundle = *intent*.*extras* // Vamos fazer um teste para saber se a pessoa esta editando a tarefa ou adicionando uma nova tarefa  
 if (bundle!=null) {  
 val tarefaRecuperado = bundle.getSerializable("tarefa") as Tarefa // as Tarefa para converter  
 binding.editTarefa.setText(tarefaRecuperado.descricao)  
 }

Dessa forma gora estamos pegando a tarefa e exibindo la.



So que precisamos salvar a tarefa ou editar e para isso vamos fazer dessa forma

## 5 Verificando se o usuário quer editar ou salvar uma tarefa

package com.allephnogueira.listadetarefas  
  
import android.os.Bundle  
import android.widget.Toast  
import androidx.activity.enableEdgeToEdge  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.view.ViewCompat  
import androidx.core.view.WindowInsetsCompat  
import com.allephnogueira.listadetarefas.database.TarefaDAO  
import com.allephnogueira.listadetarefas.databinding.ActivityAdicionarTarefaBinding  
import com.allephnogueira.listadetarefas.databinding.ActivityMainBinding  
import com.allephnogueira.listadetarefas.model.Tarefa  
  
class AdicionarTarefaActivity : AppCompatActivity() {  
  
 private val binding by *lazy* **{** ActivityAdicionarTarefaBinding.inflate(*layoutInflater*) **}** override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(binding.*root*)  
  
  
 // Tarefa inicia como null, se for null quer dizer que estamos salvando  
 // Se receber algo dentro dela quer dizer que estamos alterando.  
 var tarefa: Tarefa? = null  
  
 // Recuperar tarefa que foi passada  
 val bundle = *intent*.*extras* // Vamos fazer um teste para saber se a pessoa esta editando a tarefa ou adicionando uma nova tarefa  
 if (bundle!=null) {  
 tarefa = bundle.getSerializable("tarefa") as Tarefa // as Tarefa para converter  
 binding.editTarefa.setText(tarefa.descricao)  
 }  
  
  
 binding.btnSalvar.setOnClickListener **{** // Primeiro vamos testar se objeto esta vazio  
 if (binding.editTarefa.*text*.*isNotEmpty*()) {  
 // Agora antes de salvar, vamos verificar se o usuario quer salvar ou atualizar  
 // Lembra que se a tarefa que fizemos la em cima for null quer dizer que estamos salvando  
 // Se tiver algo dentro dela, quer dizer que estamos editando  
  
 if (tarefa != null) {  
 editar( tarefa )  
 }else {  
 salvar()  
 }  
  
 }else {  
 Toast.makeText(this,  
 "Preencha uma tarefa",  
 Toast.*LENGTH\_SHORT*)  
 .show()  
 }  
  
  
  
  
 **}** }  
  
 private fun editar(tarefa: Tarefa) {  
  
 }  
  
 private fun salvar() {  
 // Caso nao esteja vazio vamos recuperar o texto dentro dele  
 val descricaoTarefa = binding.editTarefa.*text*.toString()  
 // Vamos criar um novo objeto de tarefa passando os dados como ID, DESCRICAO e DATA  
 // Lembrando que descricao é oque o USUARIO digitou  
 val tarefa = Tarefa(  
 -1,  
 descricaoTarefa,  
 "DEFAULT"  
 )  
  
 // Vamos iniciar o TAREFADAO passando o contexto de onde vai ser executado  
 // Vamos instanciar o TAREFADAO para a gente poder pegar os metodos que estao la dentro, como salvar.  
 val tarefaDAO = TarefaDAO(this)  
  
  
 // TarefaDAO retorna sempre um verdadeiro ou falso  
 if (tarefaDAO.salvar(tarefa)) {  
 // Se for verdadeiro salvamos a tarefa e exibimos o toast e fechando a actvitiy  
 Toast.makeText(this, "Tarefa salva com sucesso!", Toast.*LENGTH\_LONG*).show()  
 finish()  
 } else {  
 // Se nao salvar exibimos um texto  
 Toast.makeText(this, "Falha ao salvar tarefa", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
 private fun salvarTarefa(nomeTarefa: String) {  
  
 }  
}

## 6 usuario editando uma tarefa

Então se foi edição queremos atualizar o valor da tarefa

Então vamos recuperar a descrição da tarefa e vamos modificar ela.

private fun editar(tarefa: Tarefa) {  
  
 val tarefaRecuperada = binding.editTarefa.*text*.toString()  
 // Agora vamos criar uma nova tarefa do 0  
 val tarefaAtualizada = Tarefa(  
 tarefa.idTarefa,  
 tarefaRecuperada,  
 "DEFAULT"  
 )  
 // Precismaos pegar o idTarefa para fazer atualização nesse dado.  
  
 val tarefaDAO = TarefaDAO(this)  
 tarefaDAO.atualizar(tarefaAtualizada)  
  
}

## 7 agora modificando o método atualizar no TarefaDAO

override fun atualizar(tarefa: Tarefa): Boolean {  
  
  
 val conteudo = ContentValues()  
 conteudo.put("${DatabaseHelper.COLUNA\_DESCRICAO}", tarefa.descricao) // Aqui é a nova descrição que estamos atualizando.  
 val args = *arrayOf*(tarefa.idTarefa.toString()) // Aqui é onde vamos pegar o ID da tarefa que estamos querendo atualizar.  
  
 try {  
 escrita.update(  
 */\*\* Parametros que precisamos para atualizar uma tabela \*/* DatabaseHelper.NOME\_TABELA\_TAREFAS,  
 conteudo,  
 "${DatabaseHelper.COLUNA\_ID\_TAREFA} = ?",  
 args  
 )  
 Log.i("info\_db", "Sucesso ao atualizar tarefa!")  
 }catch (e: Exception) {  
 e.printStackTrace()  
 Log.i("info\_db", "Erro ao atualizar tarefa!")  
 return false  
 }  
  
 return true  
  
}

## 8 verificação se deu tudo certo no AdicionarTarefaActivity

private fun editar(tarefa: Tarefa) {  
  
 val tarefaRecuperada = binding.editTarefa.*text*.toString()  
 // Agora vamos criar uma nova tarefa do 0  
 val tarefaAtualizar = Tarefa(  
 tarefa.idTarefa,  
 tarefaRecuperada,  
 "DEFAULT"  
 )  
 // Precismaos pegar o idTarefa para fazer atualização nesse dado.  
  
 val tarefaDAO = TarefaDAO(this)  
  
 // TarefaDAO retorna sempre um verdadeiro ou falso  
 // Aqui estamos verificando se a tarefa foi atualizada mesmo.  
 if (tarefaDAO.atualizar(tarefaAtualizar)) {  
 // Se for verdadeiro salvamos a tarefa e exibimos o toast e fechando a actvitiy  
 Toast.makeText(this, "Tarefa atualizada com sucesso!", Toast.*LENGTH\_LONG*).show()  
 finish()  
 } else {  
 // Se nao salvar exibimos um texto  
 Toast.makeText(this, "Falha ao atualizar tarefa", Toast.*LENGTH\_SHORT*).show()  
 }  
  
}