Alan Felipe

RecyclerView

A RecyclerView é uma das principais classes utilizadas para exibir listas de itens em um aplicativo Android. Ela é mais flexível e eficiente em termos de desempenho do que a antiga ListView. Aqui está um exemplo de como usar a RecyclerView no Android usando Kotlin:

Adicione a dependência do RecyclerView no arquivo build.gradle do seu módulo de aplicativo:

Crie um arquivo XML para o item da lista que será exibido na RecyclerView. Por exemplo, você pode criar um arquivo chamado item_lista.xml na pasta res/layout do seu projeto. Aqui está um exemplo de layout simples para um item da lista:

```
<TextView
    android:id="@+id/textViewItem"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="16sp"
    android:padding="8dp" />
```

Crie uma classe para o adaptador da RecyclerView, que estenderá a classe RecyclerView.Adapter. Este adaptador será responsável por inflar o layout do item da lista e associar os dados aos elementos de visualização correspondentes.

```
class MeuAdapter(private val dados: List<String>) : RecyclerView.Adapter<MeuAdapter.MeuViewHolder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MeuViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_lista, parent, false)
        return MeuViewHolder(view)
    }
    override fun onBindViewHolder(holder: MeuViewHolder, position: Int) {
        val item = dados[position]
        holder.textViewItem.text = item
    }
    override fun getItemCount(): Int {
        return dados.size
    }
    class MeuViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val textViewItem: TextView = itemView.findViewById(R.id.textViewItem)
    }
}
```

Na sua atividade ou fragmento, configure a RecyclerView no layout e defina o adaptador para exibir os dados.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val recyclerView: RecyclerView = findViewById(R.id.recyclerView)
        recyclerView.layoutManager = LinearLayoutManager(this)
        val dados = listOf("Item 1", "Item 2", "Item 3") // Seus dados da lista
        val adapter = MeuAdapter(dados)
        recyclerView.adapter = adapter
    }
}
```

Certifique-se de ter uma RecyclerView com o id recyclerView no seu layout principal (activity_main.xml neste exemplo).

Isso é um exemplo básico de como usar a RecyclerView no Android com Kotlin. Você pode personalizar ainda mais o layout do item da lista, adicionar manipuladores de eventos, implementar ações de clique e muito mais, de acordo com as necessidades do seu aplicativo.

//----------

Retrofit2

Adicione as dependências necessárias no arquivo build.gradle do seu módulo de aplicativo:

Crie uma interface que defina as chamadas da API que você deseja fazer. Por exemplo, se você estiver usando uma API REST que retorna uma lista de objetos, você pode criar uma interface semelhante a esta:

```
import retrofit2.Call

import retrofit2.http.GET

interface ApiService { @GET("endpoint")    fun getDados(): Call<List<ObjetoDados>>}
```

Crie uma instância do Retrofit e configure-o com a URL base da sua API. Por exemplo

```
import retrofit2.Retrofit

import retrofit2.converter.gson.GsonConverterFactory

val retrofit = Retrofit.Builder()

   .baseUrl("https://api.example.com/")

   .addConverterFactory(GsonConverterFactory.create())

   .build()

val apiService = retrofit.create(ApiService::class.java)
```

Faça a chamada à API usando o serviço Retrofit criado. Aqui está um exemplo usando uma chamada assíncrona:

```
apiService.getDados().enqueue(object : Callback<List<ObjetoDados>> {

   override fun onResponse(call: Call<List<ObjetoDados>>, response: Response<List<ObjetoDados>>) {

      if (response.isSuccessful) {

         val dados = response.body()

         // Faça algo com os dados retornados pela API

      } else { // Tratar erros de resposta da API  }

   }

   override fun onFailure(call: Call<List<ObjetoDados>>, t: Throwable) {    // Tratar erros de comunicação }

})
```

Este é um exemplo básico de como acessar dados de uma API usando Retrofit no Android com Kotlin.

//----------

Como converter Json

Crie uma classe de modelo que represente a estrutura do seu JSON. Por exemplo, se o JSON se parecer com isto:

```
{"nome": "John Doe", "idade": 25, "email": "johndoe@example.com"}
```

Você pode criar uma classe Usuario correspondente em Kotlin:

```
data class Usuario( val nome: String,  val idade: Int, val email: String )
```

Use o Gson para converter o JSON em um objeto Usuario. Aqui está um exemplo de como fazer isso:

```
import com.google.gson.Gson

// ... val jsonString = """{ "nome": "John Doe", "idade": 25, "email": "johndoe@example.com"}"""

val gson = Gson()

val usuario = gson.fromJson(jsonString, Usuario::class.java)
```

// Agora você pode usar o objeto Usuario

println("Nome: ${usuario.nome}")

Neste exemplo, fromJson() é usado para converter a string JSON em um objeto Usuario.

//----------

Layouts

Para construir um layout no Android usando Kotlin, você precisa criar um arquivo XML que define a estrutura e a aparência do layout e, em seguida, inflar esse layout em uma atividade Kotlin. Aqui está um exemplo básico para criar um layout simples:

Crie um arquivo XML para o layout. Por exemplo, você pode criar um arquivo chamado activity_main.xml na pasta res/layout do seu projeto. Aqui está um exemplo de um layout com um botão:

Abra a classe da sua atividade Kotlin (por exemplo, MainActivity.kt) e vincule o layout inflando-o na função onCreate():

O layout define a estrutura de uma interface do usuário no aplicativo, como acontece na atividade. Todos os elementos do layout são criados usando a hierarquia de objetos View e ViewGroup. A View geralmente desenha algo que o usuário pode ver e com que pode interagir. Já um ViewGroup é um contêiner invisível que define a estrutura do layout para View e outros objetos ViewGroup

Um layout pode ser declarado de duas maneiras:

Declarar elementos da IU em XML. O Android fornece um vocabulário XML direto que corresponde às classes e subclasses de visualização, como as de widgets e layouts.

Também é possível usar o Layout Editor do Android Studio para criar o layout XML usando uma interface de arrastar e soltar.

Instanciar elementos do layout no momento da execução. O aplicativo pode criar objetos View e ViewGroup (e processar suas propriedades) programaticamente.

//----------

SharedPreferences

SharedPreferences é uma classe no Android que permite armazenar e recuperar dados persistentes simples em pares de chave-valor. É frequentemente usado para armazenar preferências do usuário, configurações do aplicativo e outras informações pequenas.

As SharedPreferences são implementadas como um arquivo XML que é armazenado no diretório de dados privados do aplicativo. Elas fornecem um mecanismo fácil de usar para armazenar e acessar dados sem a necessidade de criar um banco de dados ou lidar diretamente com arquivos.

Aqui estão alguns conceitos-chave relacionados às SharedPreferences:

Chave-valor: Os dados são armazenados em pares de chave-valor, onde a chave é uma String usada para identificar o dado e o valor pode ser de qualquer tipo primitivo do Java (como String, int, boolean, etc.).

Modo de acesso: Ao usar as SharedPreferences, você pode especificar um modo de acesso, que determina quem pode acessar e modificar os dados armazenados. Alguns modos comuns são MODE_PRIVATE, que permite que apenas o próprio aplicativo acesse as SharedPreferences, e MODE_MULTI_PROCESS, que permite o compartilhamento de dados entre vários processos.

Obtendo uma instância das SharedPreferences: Você pode obter uma instância das SharedPreferences usando o método getSharedPreferences() ou getDefaultSharedPreferences() de um contexto (como uma Activity ou Application). A diferença entre esses métodos é que o getSharedPreferences() permite que você defina um nome personalizado para o arquivo de SharedPreferences, enquanto o getDefaultSharedPreferences() usa um nome padrão baseado no pacote do aplicativo.

Editor: Para modificar os dados das SharedPreferences, você precisa obter uma instância do Editor chamando o método edit() nas SharedPreferences. O Editor fornece métodos como putString(), putInt(), putBoolean() para adicionar ou atualizar os valores. Depois de concluir as modificações, você deve chamar commit() ou apply() para salvar as alterações.

Recuperando dados: Você pode recuperar dados das SharedPreferences chamando os métodos getString(), getInt(), getBoolean() etc., fornecendo a chave correspondente. Esses métodos também aceitam um valor padrão como parâmetro, que será retornado se a chave não existir.

As SharedPreferences são uma maneira simples e conveniente de armazenar pequenos conjuntos de dados persistentes no Android. No entanto, para armazenar grandes quantidades de dados estruturados, é recomendável usar um banco de dados SQLite ou outras soluções de armazenamento mais robustas.

//----------

Definição das entidades(modelo)

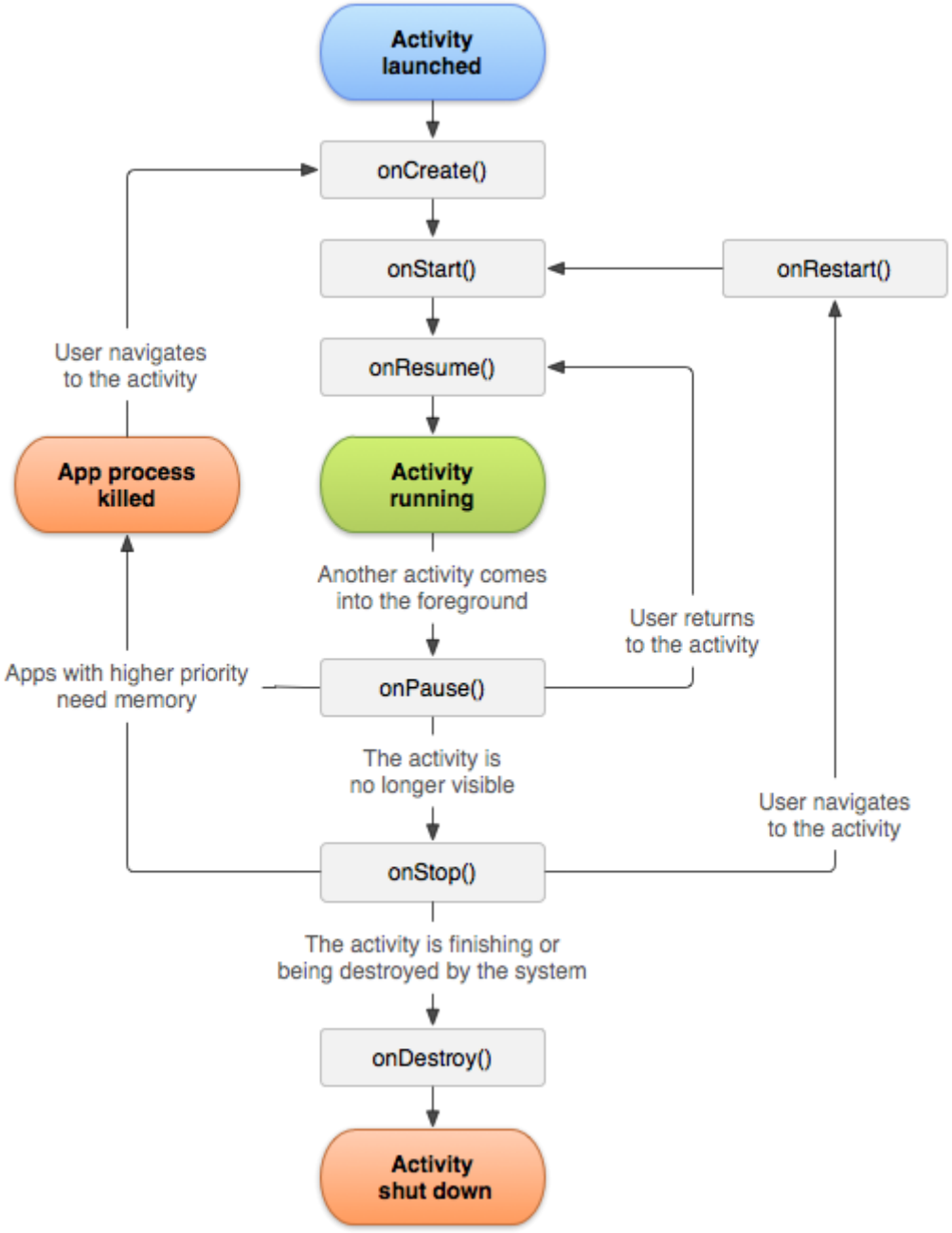
Em Kotlin, uma entidade (modelo) geralmente é representada como uma classe de dados (data class). As classes de dados são uma maneira conveniente de representar estruturas de dados imutáveis, onde cada propriedade na classe tem getters, setters, equals(), hashCode() e toString() automaticamente gerados pelo compilador. Aqui está um exemplo de como definir uma entidade (modelo) em Kotlin:

data class Usuario(val id: Int, val nome: String, val idade: Int, val email: String)

Neste exemplo, temos uma entidade chamada Usuario que possui quatro propriedades: id, nome, idade e email. As propriedades são definidas usando a palavra-chave val para torná-las somente leitura (imutáveis). O compilador Kotlin gera automaticamente os getters correspondentes para cada propriedade.

//----------

LifeCycle


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down