sãojudas universidade	FTCE Faculdade de Tecnologia e Ciências Exatas

USJT - 2019 – Desenvolvimento de Sistemas para Dispositivos Móveis

Professores: Bossini

Aula: 05
Assunto: O padrão ViewHolder e o componente RecyclerView

Passo 1 (O padrão ViewHolder) Na aula passada, passamos a fazer uso da view recebida como parâmetro do método getView para reciclar views.

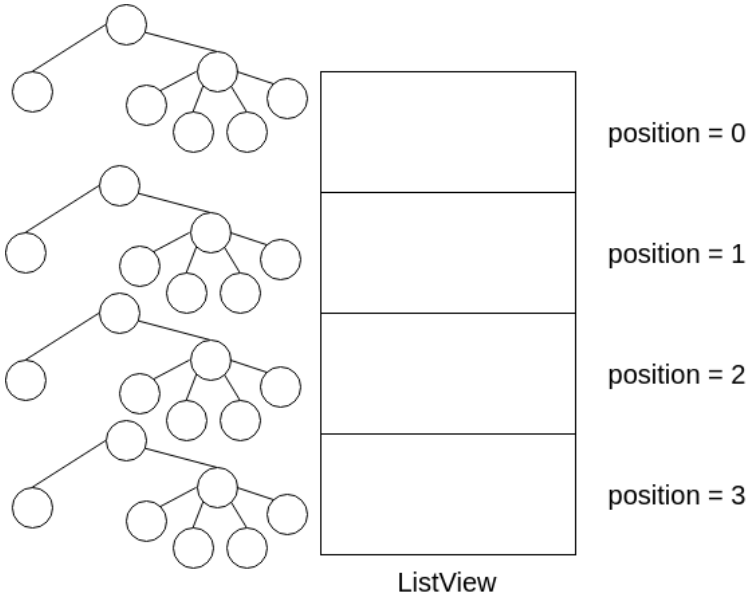
1.1 Para relembrar, veja a Listagem 1.1.

Listagem 1.1

```
@Override
public View getView(int position,View convertView,  ViewGroup
parent) {
    Chamado chamadoDaVez = getItem(position);
    Fila filaDaVez = chamadoDaVez.getFila();
    LayoutInflater inflater = LayoutInflater.from(getContext());
    if (convertView == null){
        convertView = inflater.inflate(R.layout.list_item, parent,
false);
    }
    ImageView filaIconImageView =
convertView.findViewById(R.id.filaIconImageView);
    TextView descricaoChamadoNaFilaTextView =

convertView.findViewById(R.id.descricaoChamadoNaFilaTextView);
    TextView statusChamadoNaFilaTextView =

convertView.findViewById(R.id.statusChamadoNaFilaTextView);
    TextView dataAberturaChamadoNaFilaTextView =

convertView.findViewById(R.id.dataAberturaChamadoNaFilaTextView);
    TextView dataFechamentoChamadoNaFilaTextView =

convertView.findViewById(R.id.dataFechamentoChamadoNaFilaTextView);
    filaIconImageView.setImageResource(filaDaVez.getIconId());

descricaoChamadoNaFilaTextView.setText(chamadoDaVez.getDescricao());
   statusChamadoNaFilaTextView.setText(chamadoDaVez.getStatus());

dataAberturaChamadoNaFilaTextView.setText(DateHelper.format(chamadoD
aVez.getDataAbertura()));
   if (chamadoDaVez.getDataFechamento() != null){

dataFechamentoChamadoNaFilaTextView.setText(DateHelper.format(chamad
oDaVez.getDataFechamento()));
   }
   return convertView;
}
```

1.2 Analise a Figura 1.1. Ela mostra que cada item da lista é uma árvore. Cada chamada ao método findViewById opera sobre uma dessas árvores, dependendo do valor de position. Se position igual a 0, a busca é feita na primeira árvore. Se position igual a 1, a busca é feita na segunda árvore. E assim por diante.

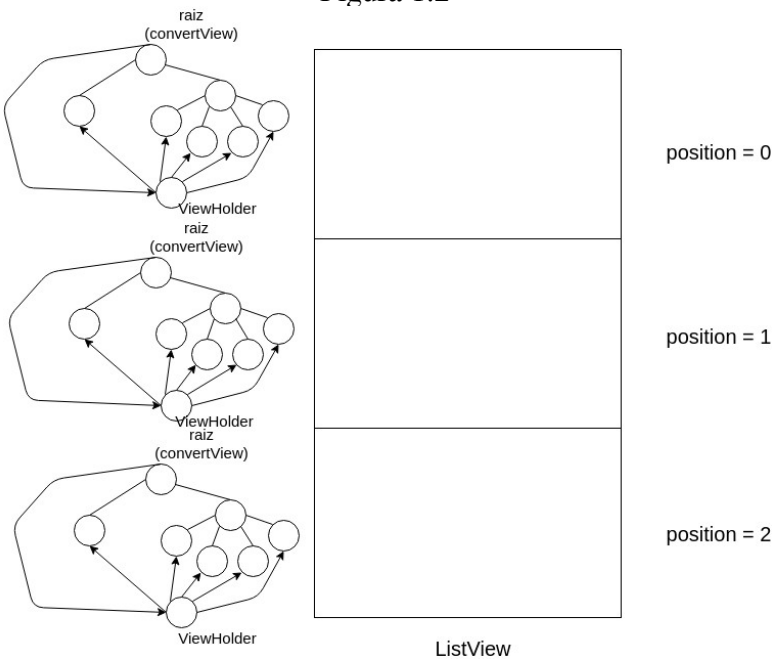
Figura 1.1



1.3 Pense agora no funcionamento do método findViewById. Dado que as views estão sendo recicladas, é verdade que para cada position as buscas se dão sobre a mesma árvore. Por exemplo, para cada item que precisa ser exibido na posição 0 da lista, uma nova chamada ao método findViewById é feita para cada view da lista.

1.4 O padrão de projeto ViewHolder será aplicado de modo que, após a primeira busca pelos componentes da árvore, a localização de cada um seja guardada (daí o nome ViewHolder) para que das próximas vezes não seja necessário realizar a busca novamente. O ViewHolder irá guardar uma referência para cada parte da árvore que seja de interesse e a raiz dela irá guardar uma referência para o ViewHolder. Veja a Figura 1.2.

Figura 1.2



1.5 Para utilizar o padrão ViewHolder e otimizar o funcionamento de nossa ListView, precisamos primeiro criar uma classe que o representa. Ela deve ter campos que irão fazer referência a todos os componentes de interesse na árvore. Veja a Listagem 1.2. Dado que somente o Adapter irá utilizar essa classe, ela pode ser uma classe interna do Adapter.

Listagem 1.2

```
private class ViewHolder{
    public ImageView filaIconImageView;
    public TextView statusChamadoNaFilaTextView;
    public TextView dataAberturaChamadoNaFilaTextView;
    public TextView dataFechamentoChamadoNaFilaTextView;
    public TextView descricaoChamadoNaFilaTextView;
}
```

1.6 O método getView deve agora fazer uso da classe ViewHolder. O algoritmo será o seguinte:

1. Quando o getView for chamado:
   1.1 Se a árvore ainda não existe:
      1.1.1 Crie a árvore.
      1.1.2 Crie um ViewHolder.
      1.1.3 Faça as buscas na árvore guardando as referências no ViewHolder.
      1.1.4 Guarde o ViewHolder na raiz da árvore.
   1.2 Se a árvore já existe:
      1.2.1 Pegue o ViewHolder da raiz da árvore.
2. Pegue o item de interesse no ArrayList.
3. Coloque os dados nos componentes visuais e devolva a raiz da árvore.

Veja a Listagem 1.3.

Listagem 1.3

```
@Override
public View getView(int position,View convertView,  ViewGroup
parent) {
    Chamado chamadoDaVez = getItem(position);
    Fila filaDaVez = chamadoDaVez.getFila();
    LayoutInflater inflater = LayoutInflater.from(getContext());
    ViewHolder vh = null;
    if (convertView == null){
        convertView = inflater.inflate(R.layout.list_item, parent,
false);
        vh = new ViewHolder();
        vh.filaIconImageView =
convertView.findViewById(R.id.filaIconImageView);
        vh.descricaoChamadoNaFilaTextView =

convertView.findViewById(R.id.descricaoChamadoNaFilaTextView);
        vh.statusChamadoNaFilaTextView =

convertView.findViewById(R.id.statusChamadoNaFilaTextView);
        vh.dataAberturaChamadoNaFilaTextView =

convertView.findViewById(R.id.dataAberturaChamadoNaFilaTextView);
        vh.dataFechamentoChamadoNaFilaTextView =
```

```
convertView.findViewById(R.id.dataFechamentoChamadoNaFilaTextView);
        convertView.setTag(vh);
    }
    vh = (ViewHolder) convertView.getTag();
    vh.filaIconImageView.setImageResource(filaDaVez.getIconId());

vh.descricaoChamadoNaFilaTextView.setText(chamadoDaVez.getDescricao(
));

vh.statusChamadoNaFilaTextView.setText(chamadoDaVez.getStatus());

vh.dataAberturaChamadoNaFilaTextView.setText(DateHelper.format(chama
doDaVez.getDataAbertura()));
   if (chamadoDaVez.getDataFechamento() != null){

vh.dataFechamentoChamadoNaFilaTextView.setText(DateHelper.format(cha
madoDaVez.getDataFechamento()));
   }
   return convertView;
}
```

Passo 2 (Entendendo o componente RecyclerView) Agora que aprendemos sobre o padrão ViewHolder, podemos fazer uso do componente RecyclerView, que é justamente um tipo de ListView que aplica o padrão ViewHolder. O componente RecyclerView tem as seguintes características:

- As Views que ele exibe são fornecidas por um LayoutManager que pode ser próprio da API ou definido pelo programador.

- As Views na lista são representadas por ViewHolders. Cada ViewHolder é responsável por somente um elemento na lista.

- Um RecyclerView criar somente a quantidade necessária de ViewHolders para preencher a parte visível da tela e alguns poucos a mais, mesmo que existam muito mais itens a serem exibidos (mas que não cabem na tela).

- Os poucos ViewHolders extras criados são utilizados para que a lista tenha bom desempenho caso o usuário role a tela. Quando for necessário exibir um novo item, um ViewHolder já estará pronto para isso.

- Um RecyclerView mantém os ViewHolders que saíram da tela em memória. Caso o usuário role a tela de volta, eles podem ser exibidos novamente. E se o usuário continuar rolando a tela, aqueles que deixaram de ser visíveis primeiro, serão os primeiros a serem reutilizados para exibir novos itens.

Passo 3 (Adaptando a aplicação para usar um RecyclerView) Agora vamos ajustar a aplicação para usar um RecyclerView.

3.1 O primeiro passo é informar a nova dependência ao Gradle. Abra o arquivo build.gradle (module: app) e adicione o conteúdo da Listagem 3.1 à seção dependencies.

Listagem 3.1

```
implementation 'com.android.support:recyclerview-v7:28.0.0'
```

3.2 A seguir, criamos um ViewHolder. Ele deve herdar de uma classe específica e deve ter campos para todos os componentes visuais que um item da lista tiver. Veja a Listagem 3.2. Note que essa classe também poderá ser interna do novo Adapter que será criado a seguir.

Listagem 3.2

```
public class ChamadoViewHolder extends RecyclerView.ViewHolder {
    public ImageView filaIconImageView;
    public TextView statusChamadoNaFilaTextView;
    public TextView dataAberturaChamadoNaFilaTextView;
    public TextView dataFechamentoChamadoNaFilaTextView;
    public TextView descricaoChamadoNaFilaTextView;
    public ChamadoViewHolder (View v){
        super (v);
        filaIconImageView = v.findViewById(R.id.filaIconImageView);
        descricaoChamadoNaFilaTextView =
                v.findViewById(R.id.descricaoChamadoNaFilaTextView);
        statusChamadoNaFilaTextView =
                v.findViewById(R.id.statusChamadoNaFilaTextView);
        dataAberturaChamadoNaFilaTextView =

v.findViewById(R.id.dataAberturaChamadoNaFilaTextView);
        dataFechamentoChamadoNaFilaTextView =
v.findViewById(R.id.dataFechamentoChamadoNaFilaTextView);
    }
}
```

3.3 Um RecyclerView também faz uso de um Adapter. Ele herda de uma classe específica e se encarrega de dizer o que ocorre quando um ViewHolder é criado pela primeira vez e também o que ocorre quando um ViewHolder é revinculado no processo de reciclagem. Veja a Listagem 3.3.

Listagem 3.3

```
public class ChamadoRecyclerViewAdapter extends RecyclerView.Adapter
<ChamadoViewHolder> {
    private List<Chamado> chamados;
    public ChamadoRecyclerViewAdapter (List <Chamado> chamados){
        this.chamados = chamados;
    }
    @NonNull
    @Override
    public ChamadoViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i)
{
        //ViewHolder sendo criado pela primeira vez.
        //inflamos a view, criamos o viewHolder e devolvemos para o RecyclerView
        View raiz =
LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.list_item, viewGroup,
false);
        return new ChamadoViewHolder(raiz);
    }
    @Override
    public void onBindViewHolder(@NonNull ChamadoViewHolder chamadoViewHolder, int i)
{
        //view holder sendo vinculado
        //ele já existe, basta colocar os dados de interesse (eles estão na posição
```

```
i da coleção)
        Chamado chamadoDaVez = chamados.get(i);

chamadoViewHolder.filaIconImageView.setImageResource(chamadoDaVez.getFila().getIconId
());

chamadoViewHolder.descricaoChamadoNaFilaTextView.setText(chamadoDaVez.getDescricao())
;

chamadoViewHolder.statusChamadoNaFilaTextView.setText(chamadoDaVez.getStatus());

chamadoViewHolder.dataAberturaChamadoNaFilaTextView.setText(DateHelper.format(chamado
DaVez.getDataAbertura()));
        if (chamados.get(i).getDataFechamento() != null){

chamadoViewHolder.dataFechamentoChamadoNaFilaTextView.setText(DateHelper.format(chama
doDaVez.getDataFechamento()));
        }
    }
    @Override
    public int getItemCount() {
        return chamados.size();
    }
}
```

3.4 A seguir, vamos adaptar o layout que contém a lista. Atualmente ele tem um ListView. Devemos trocar a tag por uma RecyclerView, como na Listagem 3.4.

Listagem 3.4

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ListaChamadosActivity">
    <android.support.v7.widget.RecyclerView
        android:id="@+id/chamadosRecyclerView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fastScrollEnabled="true"
        android:scrollbars="vertical"/>
</LinearLayout>
```

3.5 A seguir, vamos ajustar a classe ListaChamadosActivity, pois ela é quem vai usar o novo componente. No método onCreate, devemos

- trocar o tipo da variável de ListView para RecyclerView.
- buscar por um RecyclerView na árvore, trocando o id de acordo com o novo conteúdo do layout.
- Configurar o Gerenciador de Layout do RecyclerView.
- Instanciar um novo adapter, usando a mesma coleção de chamados.
- vincular o RecyclerView com o novo adapter.

Veja os trechos na Listagem 3.5.

Listagem 3.5

```
private RecyclerView chamadosRecyclerView;
chamadosRecyclerView = findViewById(R.id.chamadosRecyclerView);
chamadosRecyclerView.setLayoutManager(new
LinearLayoutManager(this));
chamadosRecyclerView.setAdapter(adapter);
```

Nota: Será necessário trocar de match_parent para wrap_content a propriedade layout_height do LinearLayout que é raiz no arquivo list_item.xml para que ele não tome a lista toda.

A fim de testar a aplicação, comente momentaneamente o registro do observador de toque nos itens. Execute a aplicação e veja se está tudo ok.

Exercício Prático

1. Ajuste sua aplicação para usar o padrão de projeto ViewHolder por meio do componente RecyclerView.

2. Faça upload do projeto no Github e gere uma release.

3. O prazo de entrega é sempre de uma semana a partir da aula em que a atividade foi proposta.