02

Melhorando a apresentação da tela para tablets

A apresentação da listagem de provas apresenta o mesmo problema que nossa listagem de alunos. Em um *smartphone* a tela com apenas a listagem já é satisfatória, entretanto em um *tablet* existe bastante espaço para ambas as partes: a listagem das provas e os detalhes na mesma tela.

No capítulo de ***Application Resources*** vimos que é possível termos pastas com ***qualifiers*** para termos recursos diferentes em função de características do dispositivo. Vamos aproveitar essa *feature* do Android para criar um *layout* diferenciado no caso do dispositivo ser um *tablet* na orientação *landscape*.

Basta criarmos um novo layout para a `ProvasActivity` com o mesmo nome e dois `<FrameLayout>` para serem substituídos por *fragments* e não apenas um:

Antes:

```
<LinearLayout ... >

    <FrameLayout android:id="@+id/provas_view"
        ... />

</LinearLayout>
```

Depois:

```
<LinearLayout ... >

    <FrameLayout android:id="@+id/provas_lista"
        ... />

    <FrameLayout android:id="@+id/provas_detalhe"
        ... />

</LinearLayout>
```

Na `ProvasActivity` agora devemos verificar se o aparelho é um *tablet* na horizontal. Uma das abordagens possíveis é a que usamos em nosso `ListaAlunosAdapter`. Podemos buscar a *view* de id `provas_detalhe`, se o retorno for `null` significa que o *layout* inflado não possui esse elemento e, consequentemente, não se trata de um *tablet* na horizontal:

```
FrameLayout frameDetalhes = (FrameLayout) findViewById(R.id.provas_detalhe);

if (frameDetalhes == null ){
    // aqui o código para smartphone comum
} else {
    //aqui o código para tablet
}
```

O problema dessa abordagem é que `if`s semelhantes vão se espalhar pelo código e sempre teremos que pensar qual *view* existirá ou não na situação que queremos diferenciar (no nosso caso, reconhecer um *tablet* na horizontal).

Para resolver esse problema vamos novamente fazer uso dos ***Application Resources*** criando um arquivo **bools.xml** na pasta values:

```
<resources>
    <bool name="isTablet">false</bool>
</resources>
```

Podemos agora criar outra versão do arquivo em uma pasta com os *qualifiers* **land** e **xlarge** colocando o booleano como `true`.

Agora em qualquer ponto da aplicação se quisermos saber se o *device* é um *tablet* na horizontal podemos fazer:

```
boolean ehTablet = getResources().getBoolean(R.bool.isTablet);
```

Com esse artifício podemos acertar o método `onCreate` da `ProvasActivity` para que a tela seja preenchida por um `Fragment` ou dois dependendo da orientação e do tipo do aparelho.

Passando dados de um *fragment* para outro

Com a lista das provas pronta podemos implementar a visualização dos detalhes da prova selecionada. A ideia é ir para uma tela de detalhes quando o usuário fizer um clique em um dos itens da listagem de provas.

Agora vamos tratar o clique no item da lista de provas e descobrir a prova selecionada, da mesma forma que já fizemos no projeto. A diferença é que agora precisaremos mandar a prova selecionada para o *fragment* `DetalhesProvaFragment` que ficará responsável por mostrar os detalhes da prova selecionada. Mas o `DetalhesProvaFragment` pode existir ou não dependendo do dispositivo. Não cabe ao `ListaProvasFragment` cuidar da criação ou localização de outro *fragment*. Vamos fazê-lo delegar para a *acticity* que está no controle da *view*:

```
listViewProvas.setOnItemClickListener(new OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapter, View view, int posicao, long id) {
        Prova selecionada = (Prova) adapter.getItemAtPosition(posicao);

        ProvasActivity calendarioProvas = (ProvasActivity) getActivity();
        calendarioProvas.selecionaProva(selecionada);
    }
});
```

Na implementação do método `selecionaProva` na `ProvasActivity` vamos instanciar o novo *fragment* e passar para ele a prova selecionada.

Se estivéssemos usando apenas *Activities* precisaríamos adotar a mesma abordagem utilizada previamente na listagem de alunos: capturar a prova selecionada no `onItemClick` da `ListView` e então passá-la por meio de uma `Intent` para uma nova *activity*.

```
Intent intent = new Intent(ListaProvasActivity.this,
        DetalhesProvaActivity.class);

intent.putExtra("naoErreADigitacaoDissoAqui", provaSelecionada);
```

O problema dessa abordagem é vincular a uma *string* o recurso que queremos: para recuperarmos a prova na outra *activity* precisamos digitar corretamente o "apelido" utilizado. Alem da possibilidade de erro, quando quisermos alterar esse "apelido" precisamos fazê-lo no mínimo em mais de um lugar.

Para passarmos informações para o *fragment* devemos usar um método chamado `setArguments` que recebe um `Bundle` como parâmetro. Então vamos criar um `Bundle` colocando nele os objetos que queremos enviar para o *fragment*. Depois basta passar a referência do `Bundle` que criamos para o *fragment*, utilizando o `setArguments` .

```
public void selecionaProva(Prova prova) {
    Bundle argumentos = new Bundle();
    argumentos.putSerializable("prova", prova);

    DetalhesProvaFragment detalhesProva = new DetalhesProvaFragment();
    detalhesProva.setArguments(argumentos);

    FragmentTransaction transaction =
        getSupportFragmentManager().beginTransaction();

    transaction.replace(R.id.provas_view, detalhesProva,
        DetalhesProvaFragment.class.getCanonicalName());

    transaction.addToBackStack(null);

    transaction.commit();
}
```

Repare que estamos fazendo um `replace` da associação do `<FrameLayout>` de id **provas_view** (até o momento vinculado com o *fragment* `ListaProvas`) e substituindo com um vínculo ao `DetalhesProvaFragment` .

Vamos invocar o método `addToBackStack` para empilhar nosso *fragment* e ter o correto funcionamento do botão ***back*** que irá desempilhar os *fragments* e só depois (quando o último *fragment* for desempilhado) será invocado o `finish` da `activity` . Quando invocamos o `addToBackStack` empilhamos todo o conteúdo

da ***transaction***, de forma que todas as operações efetuadas naquela transação serão desfeitas ao invocar o `finish` da *activity*.

Para pegarmos as informações enviadas, vamos usar o método `getArguments` dentro do método `onCreateView` do *fragment*.

```
public class DetalhesProvaFragment extends Fragment {

    // outros códigos

    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
        View layout = inflater.inflate(R.layout.provas_detalhe,
            container, false);

        // outros códigos
        if  (getArguments() != null) {
            Prova prova = (Prova) getArguments().getSerializable("prova");
        }

        // outros códigos

        return layout;
    }

    // outros códigos
}
```

Atualmente estamos apenas imprimindo na tela um `Toast` com a prova selecionada. Vamos criar uma nova *view* para exibir os detalhes da prova selecionada na listagem.

Tornando o comportamento para *tablets* diferenciado

Precisamos alterar no `ProvasActivity` o método `selecionaProva` , que lida com a seleção da prova. Na implementação do método estávamos substituindo o vínculo do `<FrameLayout>` do `ListaProvasFragment` para o `DetalhesProvaFragment` .

Implementação anterior:

```
public void selecionaProva(Prova prova) {
    Bundle argumentos = new Bundle();
    argumentos.putSerializable("prova", prova);

    DetalhesProvaFragment detalhesProva = new DetalhesProvaFragment();
    detalhesProva.setArguments(argumentos);

    FragmentTransaction transaction = getSupportFragmentManager()
        .beginTransaction();

    transaction.replace(R.id.provas_view, detalhesFragment);
    transaction.addToBackStack(null);

    transaction.commit();
}
```

Agora levando em conta que nos *tablets* a *view* de detalhes já tem um *fragment* associado, precisamos apenas substituí-lo:

```
public void selecionaProva(Prova prova) {
    Bundle argumentos = new Bundle();
    argumentos.putSerializable("prova", prova);

    DetalhesProvaFragment detalhesProva = new DetalhesProvaFragment();
    detalhesProva.setArguments(argumentos);

    FragmentTransaction transaction =
        getSupportFragmentManager().beginTransaction();

    transaction.replace(R.id.provas_view, detalhesFragment);
    if (!isTablet()) {
        transaction.addToBackStack(null);
    }
    transaction.commit();
}
```

Novamente a chamada ao método `addToBackStack` garante que automaticamente o conteúdo da transação será desfeito se o usuário apertar o botão ***back*** ou fazer qualquer ação que invoque o `finish` da *activity*.