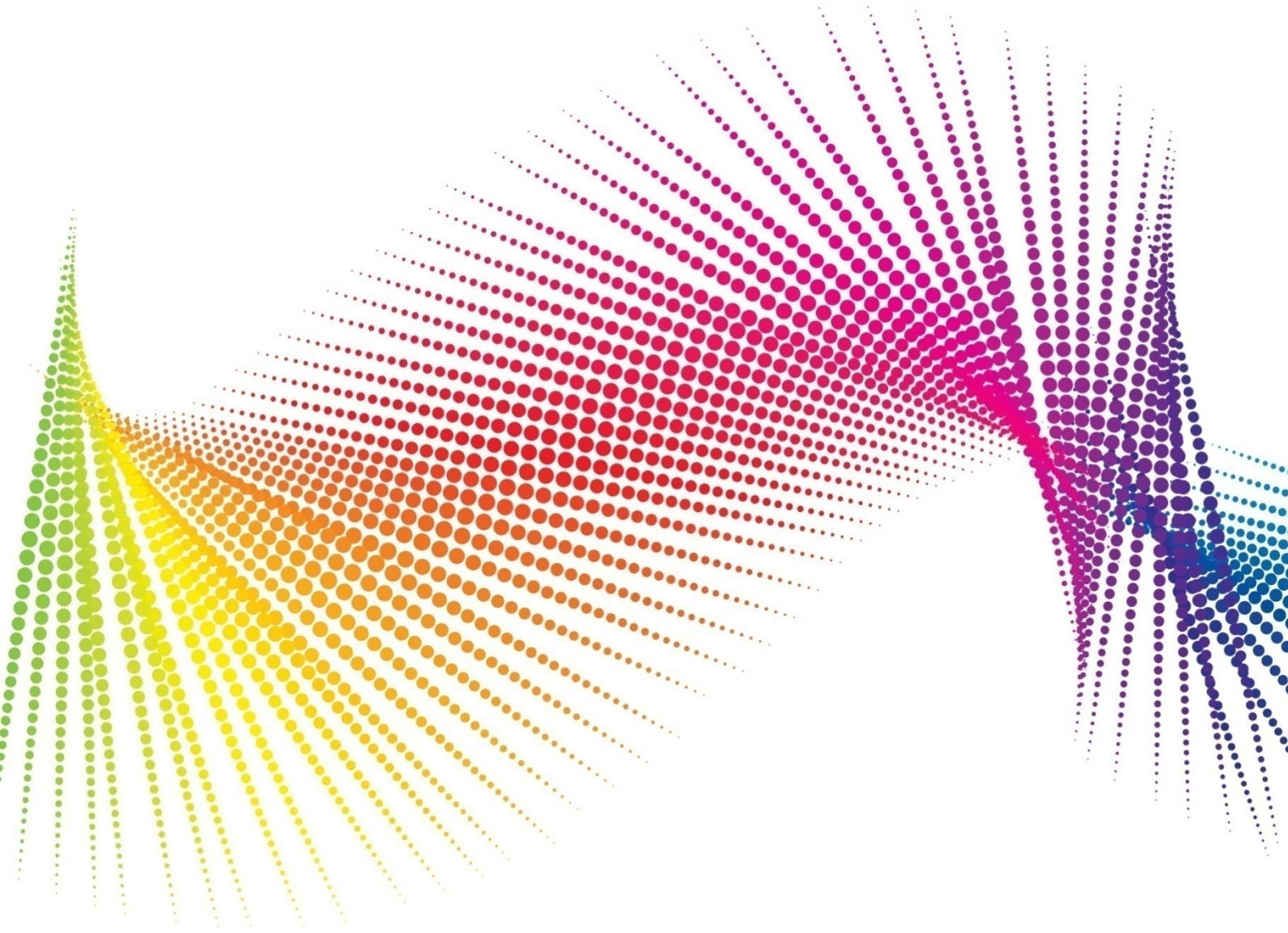
Computação Móvel

Aula 12



Aula 12: Interface com usuário android, notificações, estilo, temas, visualização

Objetivo: Concluir a demonstração das formas de construção de uma interface com usuário android, iniciando com as formas de notificação, os estilos de interfaces, criação de temas baseado nos estilos e as formas de visualização da aplicação.

Notificações

Existem vários tipos de situações que podem surgir e requerem que se notifique o usuário sobre um evento que ocorre em sua aplicação. Alguns eventos exigem que o usuário responda as notificações e outros não, por exemplo: quando um evento que necessita salvar um arquivo está completo, uma pequena mensagem deve aparecer para confirmar que a gravação foi bem sucedida.

Se o aplicativo está sendo executado em segundo plano e precisa de atenção do usuário, ele deve criar uma notificação que permite que o usuário responda a sua conveniência.

Se o aplicativo está executando o trabalho que o usuário deve esperar, o pedido deve mostrar uma barra de progresso.

Cada uma destas tarefas de notificação pode ser conseguida usando uma técnica diferente, como:

- Toast Notification, para breves mensagens que vêm do background.
- Status Notification, para lembretes persistentes que vêm do background e solicitam a resposta do usuário.
- Dialog Notification, para notificações de atividades relacionadas.

Estilos

Um estilo é um conjunto de propriedades que especificam a aparência e formato de uma View ou janela. Um estilo pode especificar propriedades, tais como altura, estofamento, cor de fonte, tamanho de fonte, cor de fundo e muito mais. Ele é definido em um arquivo de recurso que é separado do XML que especifica o layout.

Estilos em android seguem uma filosofia muito semelhante ao estilo de um web-design que permitem separar o design do conteúdo. Para melhor entendimento veja o exemplo 1.

```
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:textColor="#00FF00"
   android:typeface="monospace"
   android:text="@string/hello" />
```

E transformá-lo para isso:

```
<TextView
   style = " @style/CodeFont "
   android:text = "string@/hello"  />
```

Algumas propriedades de estilo, no entanto, não são suportadas por qualquer elemento da View e só podem ser aplicadas como um tema. Essas propriedades de estilo se aplicam a toda janela e não a qualquer tipo de exibição.

Aplicando estilos e temas para a interface do usuário

Há duas maneiras de definir um estilo:

- Para uma vista individual, adicionando o estilo a um elemento da View no XML para seu layout.
- Para uma atividade inteira ou aplicação, adicionando o atributo de tema do android para os elementos <activity> ou <application> no manifesto deste.

Quando se aplica um estilo a um único ao layout de uma view, as propriedades definidas por ele são aplicadas apenas a ela. Se um estilo é aplicado a uma ViewGroup, as visualizações de elementos subordinados a ele não herdarão o estilo diretamente e será aplicado os estilos a cada elemento.

Para aplicar uma definição de estilo como um tema, deve-se aplicar o estilo a uma atividade ou aplicação no manifesto do android. Quando se faz isso, toda a View dentro da atividade ou aplicação terá suas propriedades afetadas pelo tema.

Para aplicar um tema para uma atividade ou aplicação temos:

<application android:theme="@style/CustomTheme">

Usando estilos plataforma e temas

A plataforma android oferece uma grande coleção de estilos e temas que se pode usar em suas aplicações. Pode-se encontrar uma referência de todos os estilos disponíveis na classe R.style.

Definindo uma visualização

Para ajudar a criar uma visualização para uma aplicação, o emulador android inclui um aplicativo chamado "Preview Widget", que cria uma pré-visualização deste aplicativo.

Tipos de visualização

- ListView – uma visão que mostra itens em uma lista com barra de rolagem vertical.
- GridView – uma visão que mostra itens em duas dimensões de uma grade.
- StackView – uma visão como uma espécie de fichário, em que o usuário pode selecionar a ficha que deseja ver.
- AdapterViewFlipper – Um adaptador simples apoiado em ViewAnimator que anima entre dois ou mais pontos de vista.

Disposição linear

LinearLayout é um grupo vista que alinha todas as Views em uma única direção, verticalmente ou horizontalmente. Para melhor entendimento veja o exemplo 2.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:paddingLeft="16dp"
        android:paddingRight="16dp"
        android:orientation="vertical" >
        <EditText
           android:layout_width="fill_parent"
           android:layout_height="wrap_content"
           android:hint="@string/to" />
        <EditText
           android:layout_width="fill_parent"
           android:layout_height="wrap_content"
           android:hint="@string/subject" />
        <EditText
           android:layout_width="fill_parent"
           android:layout_height="0dp"
           android:layout_weight="1"
           android:gravity="top"
           android:hint="@string/message" />
        <Button
           android:layout_width="100dp"
           android:layout_height="wrap_content"
           android:layout_gravity="right"
           android:text="@string/send" />
</LinearLayout>
```

Disposição relativa

RelativeLayout é um grupo de vista que apresenta as Views em posições relativas. A posição de cada um pode ser vista como especificado em relação às outras.

A RelativeLayout é um utilitário muito poderoso para a concepção de uma interface de usuário, pois pode eliminar grupos aninhados e manter o plano de hierarquia do layout, que melhora o desempenho. Para melhor entendimento veja o exemplo 3.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:paddingLeft="16dp"
        android:paddingRight="16dp" >
        <EditText
           android:id="@+id/name"
           android:layout_width="fill_parent"
           android:layout_height="wrap_content"
           android:hint="@string/reminder" />
        <Spinner
           android:id="@+id/dates"
           android:layout_width="0dp"
           android:layout_height="wrap_content"
           android:layout_below="@id/name"
           android:layout_alignParentLeft="true"
           android:layout_toLeftOf="@+id/times" />
        <Spinner
           android:id="@id/times"
           android:layout_width="96dp"
           android:layout_height="wrap_content"
           android:layout_below="@id/name"
           android:layout_alignParentRight="true" />
```

```
        <Button
            android:layout_width="96dp"
            android:layout_height="wrap_content"
            android:layout_below="@id/times"
            android:layout_alignParentRight="true"
            android:text="@string/done" />
</RelativeLayout>
```

ListView

ListView é um grupo de exibição que mostra uma lista de itens de rolagem. Os itens da lista são automaticamente inseridos na lista usando um adaptador que puxa conteúdo de uma fonte, como uma consulta matriz ou banco de dados e converte cada item no resultado uma visão de que é colocado na lista. Para melhor entendimento veja o exemplo 4.

```
public class ListViewLoader extends ListActivity
        implements LoaderManager.LoaderCallbacks<Cursor> {

        SimpleCursorAdapter mAdapter;

        static final String[] PROJECTION = new String[]
            {ContactsContract.Data._ID,
                        ContactsContract.Data.DISPLAY_NAME};

        static final String SELECTION = "((" +
            ContactsContract.Data.DISPLAY_NAME + " NOTNULL) AND (" +
            ContactsContract.Data.DISPLAY_NAME + " != " ))";

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);

            ProgressBar progressBar = new ProgressBar(this);
```

```
        progressBar.setLayoutParams(new
                LayoutParams(LayoutParams.WRAP_CONTENT,
                LayoutParams.WRAP_CONTENT, Gravity.CENTER));
        progressBar.setIndeterminate(true);
        getListView().setEmptyView(progressBar);

        ViewGroup root = (ViewGroup findViewById(android.R.id.content);
                        root.addView(progressBar);

        String[] fromColumns =
                        {ContactsContract.Data.DISPLAY_NAME};
        int[] toViews = {android.R.id.text1}; // The TextView in
                 simple_list_item_1

         mAdapter = new SimpleCursorAdapter(this,
             android.R.layout.simple_list_item_1, null,
             fromColumns, toViews, 0);
        setListAdapter(mAdapter);

        getLoaderManager().initLoader(0, null, this);
    }

    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
        return new CursorLoader(this,
                ContactsContract.Data.CONTENT_URI,
                PROJECTION, SELECTION, null, null);
    }

    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
        mAdapter.swapCursor(data);
    }

    public void onLoaderReset(Loader<Cursor> loader) {
        // This is called when the last Cursor provided to onLoadFinished()
```

```
        // above is about to be closed.  We need to make sure we are no
        // longer using it.
        mAdapter.swapCursor(null);
    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Do something when a list item is clicked
    }
}
```

Grade

GridView é um ViewGroup que exibe itens em uma grade bidimensional, rolável. Os itens da grade são automaticamente inseridos no layout usando um ListAdapter.

Implementando uma View

O primeiro método necessário é o getView(). Este método cria uma nova View para cada visualização adicionada à ImageAdapter. Quando este é chamado, uma View é passada, o que normalmente é um objeto, e não necessita de verificação para ver saber se o objeto é nulo. Podem-se configurar as propriedades de uma View nas seguintes formas:

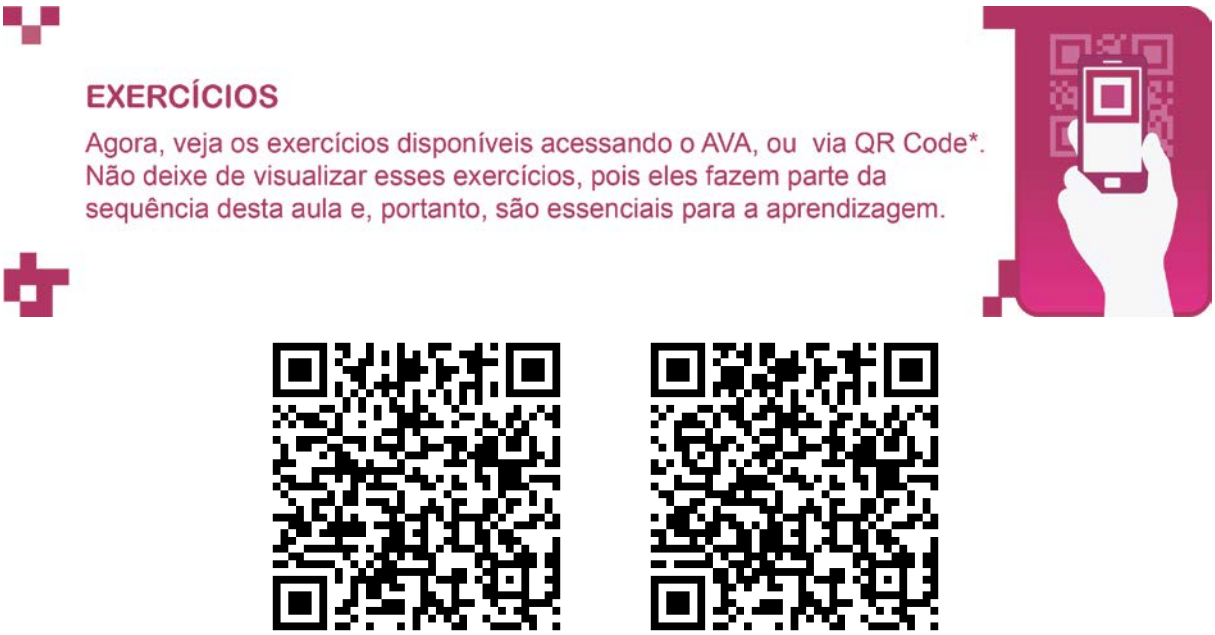
- setLayoutParams (ViewGroup.LayoutParams) define a altura e largura para a View.
- setScaleType (ImageView.ScaleType) declara que as View devem ser colocadas em direção ao centro (se necessário).
- setPadding (int, int, int, int) define o preenchimento de todos os lados.

Resumo

Nesta aula concluímos os métodos de construção da interface com o usuário de uma aplicação android, em que explicamos a criação e as necessidades de utilização das notificações, apresentamos a construção e utilização de estilos estilo e temas para a melhoria das atividades da aplicação e finalizamos com a demonstração das visualizações que podem ser adotadas em atividade no andorid.

Exercícios

Para termos certeza de que entendemos a aula, vamos realizar uma atividade sobre interface do usuário.



Próxima aula

Uma vez que já temos os conhecimentos sobre o funcionamento da arquitetura android, podemos dar o próximo passo em nosso aprendizado. Na próxima aula – **Aula 13** – aprenderemos a manipular os recursos de mensagens e filtros oferecidos pelo android.

REFERÊNCIAS

LECHETA, Ricard R. *Android*: aprenda a criar aplicações para dispositivos móveis com o android SDK. 2. ed. São Paulo: Novatec, 2010.

ROGERS, Rick; et al. *Desenvolvimento de aplicações android*. São Paulo: Novatec, 2009.