



POLITÉCNICO
DE LEIRIA
ESCOLA SUPERIOR
DE TECNOLOGIA
E GESTÃO

CTeSP de Programação de Sistemas de Informação

Acesso Móvel a Sistemas de Informação

Desenvolvimento Móvel em Android

Personalização de Aplicações
com Estilos e Temas

Sónia Luz, sonia.luz@ipleiria.pt

David Safadinho, david.safadinho@ipleiria.pt

Departamento de Engenharia Informática

Escola Superior de Tecnologia e Gestão

Instituto Politécnico de Leiria

1º Semestre - 2021/2022

Personalização de Aplicações

- O desenvolvimento de aplicações em Android é muito versátil
 - permitindo a implementação de diversos tipos de aplicações:
 - Mais profissionais, mais formais, mais interativos ou até mais dinâmicos, etc.;
- Cada aplicação tem uma UI com aparência e formato específicos;
 - Que podem ser personalizados de acordo com o seu objetivo:
 - Componente a componente;
 - Tipos de componentes;
 - Uma atividade;
 - Toda a aplicação

Estilos e Temas

- A personalização de aplicações Android pode ser efetuada através de **Estilos e Temas**;
- Resumidamente:
 - Um **estilo** é um conjunto de propriedades que definem a aparência de uma *View* ou janela;
 - Um **tema** é um tipo específico de estilo aplicado sobre uma atividade ou toda a aplicação.

Estilo

- Um estilo é um conjunto de propriedades que especificam a aparência e formato de uma *View* ou janela.
 - Pode especificar propriedades, tais como:
 - altura, cor da fonte, preenchimento, tamanho da fonte, cor de fundo e muito mais;
- Um estilo é definido num recurso XML que é independente do XML que especifica o layout.

Tema

- Um tema é um estilo aplicado
 - A uma *atividade* (a partir do ficheiro de Manifesto);
 - Ou à *aplicação* inteira (a partir do ficheiro de Manifesto);
 - Ao invés de uma *View* individual (a partir de um ficheiro de layout);
- Quando um estilo é aplicado como um tema
 - Todas as *Views* na atividade ou aplicação irão usar todas as propriedades de estilo por ele definidas.

Definir Estilos

- Para definir um conjunto de estilos deve:
 - Criar um ficheiro **.xml** - normalmente designado de **styles.xml**;
 - Na pasta **res/values/**;
- O ficheiro XML:
 - Deve ter como nó raiz um elemento **<resources>**;
 - Para cada estilo a criar deve adicionar um elemento **<style>**;
 - Definindo o atributo (obrigatório) **name** para identificar univocamente o estilo;

Definir um Estilo

- Para cada elemento **<style>** deve:
 - Definir o atributo **name** (obrigatório) para identificar univocamente o estilo;
 - Adicionar um elemento **<item>**, para cada propriedade do estilo
 - Com um atributo **name** (obrigatório) que declara a propriedade de estilo e o seu valor respetivo;
 - O seu valor pode ser:
 - uma string de palavra-chave, uma cor em hexadecimal, uma referência a outro tipo de recurso ou outro valor, dependendo da propriedade de estilo.

Definir um Estilo

- Recorrendo à criação do Estilo no ficheiro **styles.xml**:
 - Que deve ter por base o AppTheme
 - definido como estilo base a aplicar a toda a aplicação;

```
<resources>
  <style name="AppTheme.Button" parent="@android:style/Widget.Button">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:textSize">18sp</item>
    <item name="android:textStyle">bold</item>
    <item name="android:typeface">normal</item>
    <item name="android:textColor">@android:color/holo_blue_dark</item>
  </style>
</resources>
```


Aplicar um Estilo

- Assim, usando o estilo definido é possível usar um layout XML, e transformá-lo:

```
<Button
    android:id="@+id/btnListaContactos"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    android:textStyle="bold"
    android:typeface="normal"
    android:onClick="onClickListaContactos"
    android:text="@string/btnLCText"
/>
```

```
<Button
    android:id="@+id/btnListaContactos"
    style="@style/AppTheme.Button"
    android:onClick="onClickListaContactos"
    android:text="@string/btnLCText"
/>
```

Herdar um Estilo

- O atributo **parent** no elemento **<style>** permite especificar um estilo
 - Do qual o seu estilo deve herdar propriedades;
 - Pode herdar propriedades de um estilo existente e definir apenas as propriedades que deseja alterar ou adicionar;
 - É possível herdar estilos:
 - Criados por nós;
 - Incorporados no Android.

Herdar um Estilo Criado

- É possível definir a herança de um estilo criado por nós:
 - Através do atributo **parent**;

```
<style name="MyButton" parent="AppTheme.Button">  
  <item name="android:background">@android:color/background_light</item>  
</style>
```

```
<Button  
  android:id="@+id/btnGrelhaContactos"  
  style="@style/MyButton"  
  android:text="@string/btnGCText" />
```

- Ou diretamente na definição do nome do novo estilo;

```
<style name="AppTheme.Button.MyButton">  
  <item name="android:background">@android:color/background_light</item>  
</style>
```

```
<Button  
  android:id="@+id/btnGrelhaContactos"  
  style="@style/AppTheme.Button.MyButton"  
  android:text="@string/btnGCText" />
```

Herdar um Estilo Incorporado do Android

- A herança de um estilo incorporado do Android
 - É efetuada obrigatoriamente através do atributo **parent**

```
<style name="MyTextView" parent="@android:style/TextAppearance">  
    <item name="android:textColor">#FF0000</item>  
    <item name="android:textSize">24dp</item>  
</style>
```

```
<TextView  
    android:id="@+id/textViewId"  
    style="@style/MyTextView" />
```

Definir um Tema

- Definir um *tema* é idêntico a definir um *estilo* mas:
 - O **parent** do estilo refere-se a um tema do Android;
 - Que tem por base o **Theme.AppCompat**;
 - Este **tema** é usado para garantir que a aplicação corre independente da plataforma;
 - Faz parte do *Android Support Libraries* e permite que dispositivos mais antigos utilizem novas características do Android SDK.

Definir um Tema

- Ao definir um *tema*:
 - É possível aplicar um estilo a todos os componentes de um determinado tipo como botões, ou outras *views*;
 - Podemos aplicar estilos previamente criados por nós;

```
<style name="AppTheme.Light" parent="Theme.AppCompat.Light">  
  <item name="android:typeface">serif</item>  
  <item name="android:buttonStyle">@style/AppTheme.Button</item>  
  <item name="android:textViewStyle">@style/MyText</item>  
</style>
```

Aplicar um Tema

- É possível aplicar um tema através do **AndroidManifest.xml**
 - A uma atividade

```
<activity android:name=".DetalhesContacto"  
          android:theme="@style/AppTheme.Light">
```

- A toda a aplicação

```
<application  
    android:theme="@style/AppTheme.Light">
```

- Mas o estilo de uma atividade **não** se pode sobrepor ao da aplicação;

Selecionar um Tema ajustado à versão Android

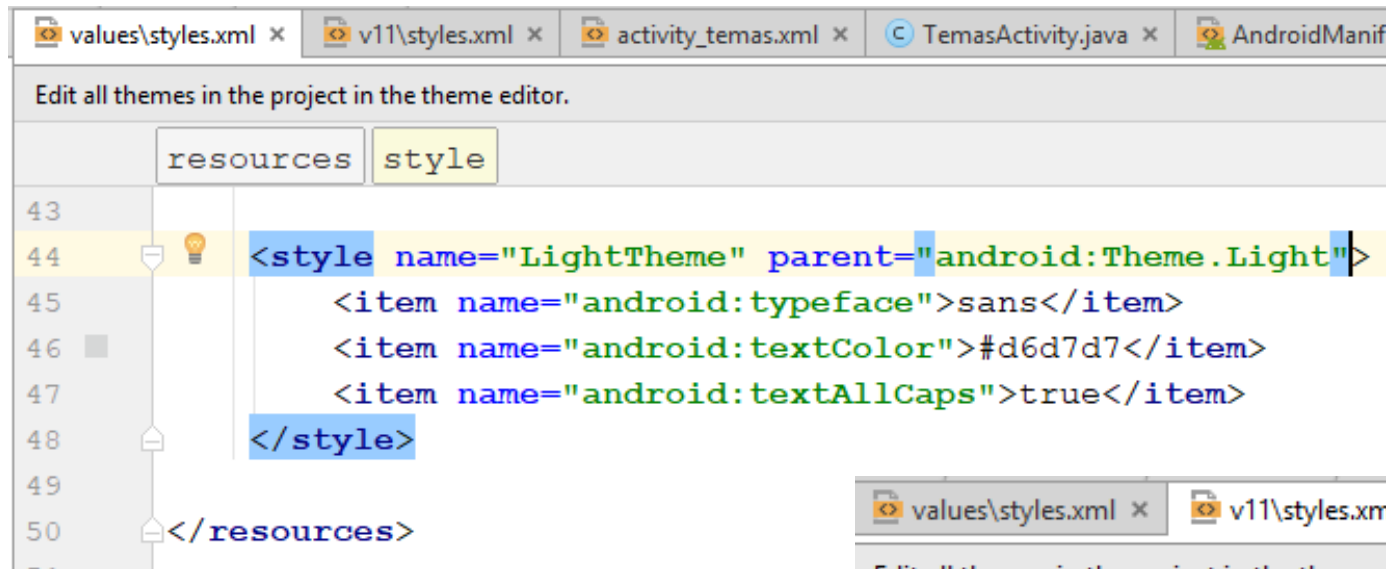
- As versões mais recentes do Android têm temas adicionais disponíveis para as aplicações;
 - Quando não se herda de um **Theme.AppCompact**;
- É possível usá-los e aplicá-los nessas plataformas mantendo a compatibilidade com versões anteriores;
 - Através de um tema personalizado
 - Que usa seleção de recursos para alternar entre temas pai diferentes;
 - Com base na versão da plataforma.

Selecionar um Tema ajustado à versão Android

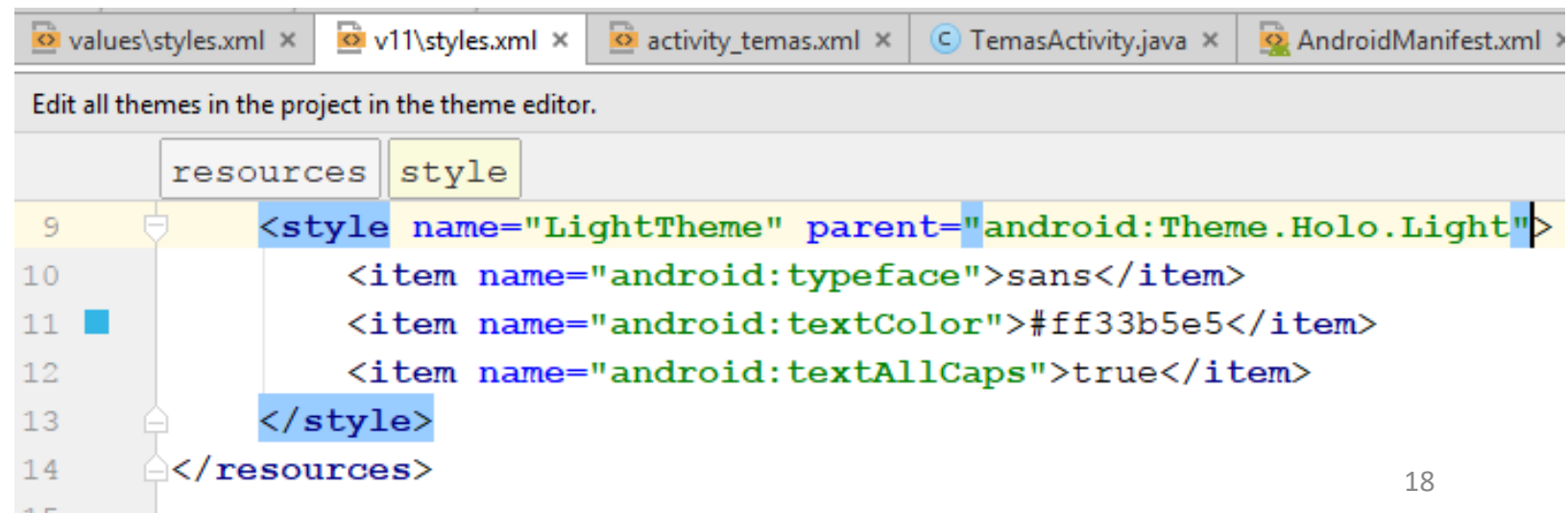
- Por exemplo:
 - Podemos definir um tema **LightTheme** que tem por base o tema Light (claro) padrão da plataforma;
 - E definir o mesmo tema mas holográfico disponível para versões mais recentes (**Android 3.0 - API nível 11**)
 - Para tal deve definir um novo ficheiro xml em **res/values-v11**
 - Indicar o tema holográfico como o **parent**
- Assim é possível usar esse tema como qualquer outro e a aplicação alternará automaticamente para o tema holográfico
 - se executado no Android 3.0 ou posterior

Selecionar um Tema ajustado à versão Android

• Exemplo:



```
values\styles.xml x v11\styles.xml x activity_temas.xml x TemasActivity.java x AndroidManifest.xml
Edit all themes in the project in the theme editor.
resources style
43
44 <style name="LightTheme" parent="android:Theme.Light">
45     <item name="android:typeface">sans</item>
46     <item name="android:textColor">#d6d7d7</item>
47     <item name="android:textAllCaps">>true</item>
48 </style>
49
50 </resources>
```



```
values\styles.xml x v11\styles.xml x activity_temas.xml x TemasActivity.java x AndroidManifest.xml
Edit all themes in the project in the theme editor.
resources style
9 <style name="LightTheme" parent="android:Theme.Holo.Light">
10     <item name="android:typeface">sans</item>
11     <item name="android:textColor">#ff33b5e5</item>
12     <item name="android:textAllCaps">>true</item>
13 </style>
14 </resources>
```

Modificar o Tema em tempo de execução

- Quando temos dois temas diferentes:
 - Podemos permitir ao utilizador alterar o tema ativo;
 - Iremos recorrer às preferências para guardar o tema atual;

```
public class TemasActivity extends AppCompatActivity
implements SharedPreferences.OnSharedPreferenceChangeListener {

    //chave para o grupo de preferências
    public static final String PREFERENCE_THEME = "Pref_Theme";
    //chave para a preferência referente ao nome do tema
    private static final String TEMA = "ThemeName";
    private SharedPreferences preferences;
    private SharedPreferences.Editor editor;
    private String themeName;

}
```

Modificar o Tema em tempo de execução

- Temos de definir/atribuir o tema para a atividade:
 - Ou para todas as atividades que quisermos alterar;

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //aceder ao tema guardado
    preferences = getSharedPreferences(PREFERENCE_THEME, MODE_PRIVATE);
    themeName = preferences.getString(TEMA, "AppTheme.Light");
    int themeId = getResources().getIdentifier(themeName, "style",
                                                getPackageName());
    //efetuar o setTheme antes de instanciar qualquer view
    setTheme(themeId);
    //registar a atividade como Listener de alterações nas preferências
    preferences.registerOnSharedPreferenceChangeListener(this);

    setContentView(R.layout.activity_temas);
}
```

Modificar o Tema em tempo de execução

- Para testar:
 - Podemos recorrer a um botão para alternar o tema aplicado

```
public void onClickMudarTema(View view) {  
    //atualizar preference com o tema atual  
    themeName = themeName=="AppTheme.Light"?  
                "AppTheme.Dark":"AppTheme.Light";  
    editor = preferences.edit();  
    editor.putString(TEMA, themeName);  
    editor.apply();  
}
```

Modificar o Tema em tempo de execução

- É necessário recarregar o tema cada vez que o utilizador alterar as preferências da atividade
 - Recorrendo ao método **recreate**
 - Para que a atividade seja terminada;
 - E reiniciada;

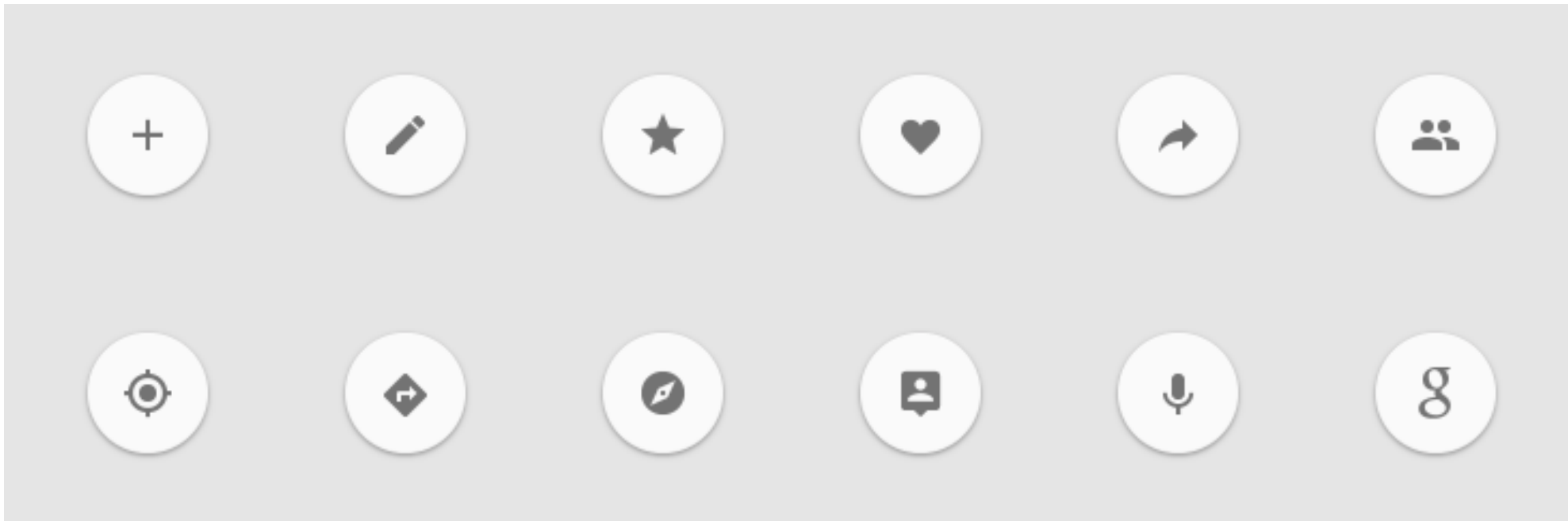
```
@Override
public void onSharedPreferenceChanged(
    SharedPreferences sharedPreferences, String key) {
    if ( key.equals(TEMA) ) {
        //recrir a atividade para que o tema seja atualizado
        this.recreate();
    }
}
```

Personalizar: Floating Action Button

- Os FAB são usados para destacarem uma única ação
 - Que o utilizador pode executar quando está numa secção da aplicação;
- Representados por um ícone flutuante e circular colocado sobre a UI;
- Podem ser definidos em dois tamanhos:
 - Default (Normal) e o mini
 - O tamanho pode ser controlado através:
 - do atributo **fabSize**;
 - do método **setSize()**;

Personalizar: Floating Action Button

- Deve representar ações mais comuns e positivas:
 - Criar, Favoritos, Partilhar, Navegar e Explorar



Personalizar: Floating Action Button

- Esta classe herda da *ImageView*
 - Por isso é possível controlar o ícone apresentado através:
 - Do atributo `src`;
 - Do método `setImageDrawable(Drawable)` ;
- A cor de fundo por omissão corresponde ao tema *colorAccent*
 - Pode ser alterado em *runtime* através do método `setBackgroundTintList(ColorStateList)` .

Personalizar: Floating Action Button

- Exemplo de um FAB
 - Para adicionar um novo contacto

```
<com.google.android.material.floatingactionbutton.  
FloatingActionButton  
    android:id="@+id/fabAddTeste"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="end"  
    android:clickable="true"  
    android:focusable="true"  
    app:backgroundTint="#FF4081"  
    app:srcCompat="@android:drawable/ic_input_add"  
>
```



Desafio:

- Criar um estilo personalizado para aplicar a cada botão;
- Criar um tema personalizado, que herde do estilo anterior e que seja aplicado a uma atividade e depois a toda a aplicação;
- Criar um *FloatingActionButton* que saliente uma única operação numa determinada atividade;

Fontes e Mais Informação

- Estilos e Temas
 - <https://developer.android.com/guide/topics/ui/look-and-feel/themes>
- Estilos do Android
 - <https://android.googlesource.com/platform/frameworks/base/+refs/heads/master/core/res/res/values/styles.xml>
- Temas do Android
 - <https://android.googlesource.com/platform/frameworks/base/+refs/heads/master/core/res/res/values/themes.xml>
- *FloatingAction Button*
 - <https://developer.android.com/reference/android/support/design/widget/FloatingActionButton.html>
 - <https://developer.android.com/guide/topics/ui/floating-action-button>

Próximo Tema:

Persistência de Dados

- Opções de Armazenamento
 - <https://developer.android.com/training/data-storage>
- Recriar uma Atividade
 - <https://developer.android.com/guide/components/activities/activity-lifecycle>
- Armazenamento Interno
 - <https://developer.android.com/training/data-storage#filesInternal>
 - <https://developer.android.com/training/data-storage/app-specific>