



Android

Programando Aplicativos para Android

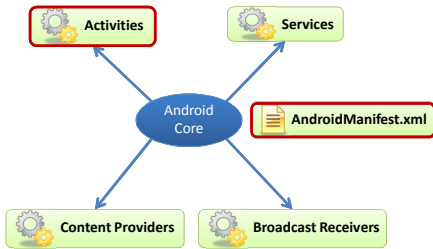
Softblue
cursos online

Tópicos Abordados

- Aplicações no Android
- *AndroidManifest.xml*
- Activities
- Resources
- Logging
- API de compatibilidade
- Permissões de acesso

Aplicações no Android

- Uma aplicação pode ser um conjunto de diversos componentes



```
graph TD; AC((Android Core)) --> A[Activities]; AC --> S[Services]; AC --> CP[Content Providers]; AC --> BR[Broadcast Receivers]; AM[AndroidManifest.xml];
```

AndroidManifest.xml



- Arquivo que declara os componentes do aplicativo e outras informações
- Todo aplicativo possui um arquivo *AndroidManifest.xml*
- O formato do arquivo é XML

AndroidManifest.xml



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="softblue.android"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="softblue.android.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Activities



- Uma activity normalmente representa uma tela da aplicação
- Representada por uma classe que herda de **android.app.Activity**

```
public class MyActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView view = new TextView(this);
        view.setText("Texto da tela");
        setContentView(view);
    }
}
```

A Classe *Context*



- Representa o contexto de execução da aplicação do Android
- Possui muitos métodos que são comumente chamados em aplicações
- Um objeto desta classe está normalmente disponível em vários lugares da aplicação
 - A classe **Activity** herda de **Context**
 - O *Context* é fornecido como parâmetro

A Classe *Context*



- O Android tem basicamente dois tipos de contextos
 - Contexto da activity
 - Cada activity da aplicação tem um contexto
 - Contexto da aplicação
 - É único para a aplicação toda
- É preciso tomar cuidado na escolha do contexto para não causar *memory leaks* na aplicação (vazamento de memória)

A Classe *Context*



- A classe **Activity** herda de **Context**

```
public class MyActivity extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        TextView view = new TextView(this);  
    }  
}
```

Contexto da activity

```
public class MyActivity extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        MyManager m = MyManager.getInstance(getApplicationContext());  
    }  
}
```

Contexto da aplicação

Activities e Views



- Views são os componentes que se agrupam para montar a interface gráfica
- As activities e as views têm uma relação próxima
 - Activities representam a tela
 - Views representam o que é renderizado na tela

Activities e Views



- O método **setContentView()** da activity é utilizado para determinar qual view será renderizada

```
TextView view = new TextView(this);  
view.setText("Texto da tela");  
setContentView(view);
```

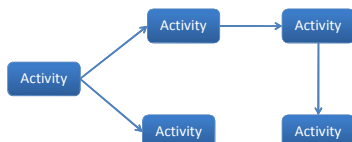
Renderiza a view

Criação de uma
caixa de texto

Invocando Activities



- Uma aplicação normalmente é composta por um conjunto de activities



A comunicação entre activities é feita através de uma **intent**

Invocando Activities



- Um objeto **Intent** é usado para disparar uma nova activity

```
Tela1.java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent it = new Intent(getApplicationContext(), Tela2.class);
    startActivity(it);
}
```

Retornando Informações



- O método **startActivity()** chama outra activity
- Se esta nova activity for finalizada, a activity que fez a chamada não recebe nenhuma informação
- Para notificar a activity chamadora, é possível usar **startActivityForResult()**
- A nova activity carregada é considerada uma sub-activity, já que fica atrelada à activity que a chamou

Retornando Informações



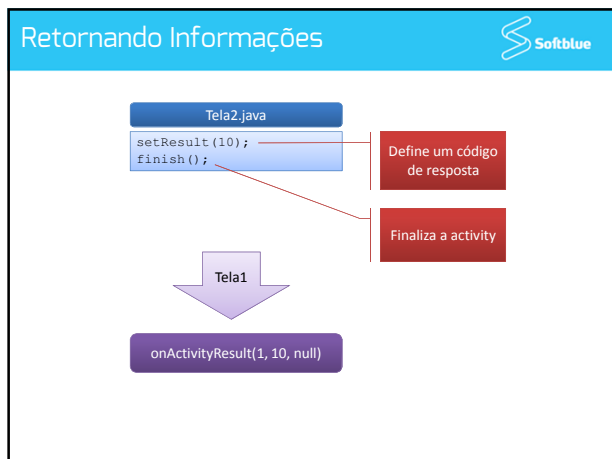
```
Tela1.java
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent it = new Intent(getApplicationContext(), Tela2.class);
    startActivityForResult(it, 1);
}

protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    //...
}
```

É preciso fornecer, além da intent, um código de identificação

Invocado quando a sub-activity chamada é finalizada



Passagem de Parâmetros

- É possível passar parâmetros de uma activity para outra usando um **Bundle**

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Intent it = new Intent(getApplicationContext(),
        Tela2.class);
    Bundle params = new Bundle();
    params.putString("msg", "Alguma mensagem");
    it.putExtras(params);
    startActivity(it);
}
```

Passagem de Parâmetros

- A leitura do parâmetro é feita a partir da recuperação da intent que disparou a activity

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

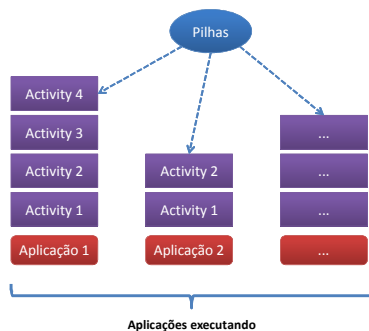
    Intent it = getIntent();
    Bundle params = it.getExtras();
    String msg = params.getString("msg");
}
```

As Activities na Memória



- As activities que fazem parte da sua aplicação são organizadas numa pilha
- A activity que está no topo da pilha é a activity mostrada na tela
- Toda vez que uma nova activity é invocada, ela passa a ficar no topo da pilha
- Quando o botão voltar é pressionado, a activity atual é removida da pilha e dá espaço para a activity que ocupa o topo da pilha ficar ativa

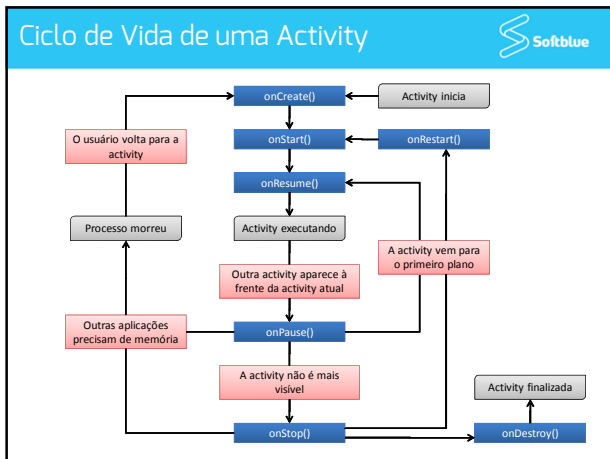
A Pilha de Activities

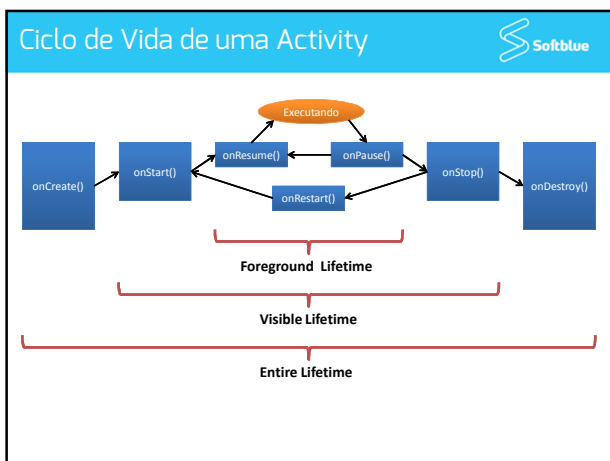


Ciclo de Vida de uma Activity



- Métodos de callback invocados pelo Android durante o ciclo de vida da activity
 - `onCreate()`
 - `onStart()`
 - `onResume()`
 - `onPause()`
 - `onStop()`
 - `onRestart()`
 - `onDestroy()`





Salvando o Estado de uma Activity

- Uma activity pode ser destruída pelo Android para liberar recursos
- Quando ela for recriada, é possível que informações da activity sejam perdidas
- Para evitar que isto aconteça, é necessário salvar o estado da activity
- Não é preciso se preocupar com o estado das views, pois ele é gravado e depois recuperado
 - É preciso que a view tenha um ID especificado

O método `onSaveInstanceState()`



- O Android chama o método **`onSaveInstanceState()`** na activity quando ela não fica mais visível

```
public class MyActivity extends Activity {
    private String nome;
    private int idade;

    protected void onSaveInstanceState(Bundle outState) {
        outState.putString("nome", nome);
        outState.putInt("idade", idade);
        //...
        super.onSaveInstanceState(outState);
    }
}
```

Bundle para salvar os dados

Chama o método da superclasse

- Quando a activity é destruída, o estado dela fica guardado

Restaurando o Estado de uma Activity



- Quando a activity for recriada, o Android chama o método **`onCreate()`** passando como parâmetro o bundle que contém os dados salvos

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    if (savedInstanceState != null) {
        this.nome = savedInstanceState.getString("nome");
        this.idade = savedInstanceState.getInt("idade");
    }
}
```

Se o bundle não é null, a activity está sendo restaurada

- O método **`onRestoreInstanceState(Bundle)`** também é chamado (depois de `onStart()`)

Resources



- Exemplos de recursos (resources)
 - Textos
 - Imagens
 - Sons
 - Layouts de tela
- O ideal é não referenciar os resources diretamente no código
 - Repetição de código
 - Difícil de manter
- O Android possui mecanismos para trabalhar com resources de forma fácil e centralizada

Localização dos Resources

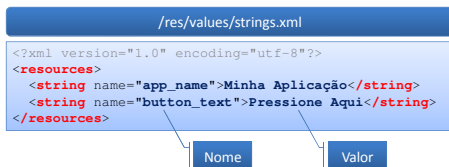


- Os resources ficam localizados dentro do diretório **/res**
- Este diretório contém subdiretórios de acordo com tipo de recurso que representam
 - /res/values-xxx
 - /res/layout
 - /res/drawable-xxx
 - etc.
- Dentro dos subdiretórios ficam os arquivos de resources

String Resources



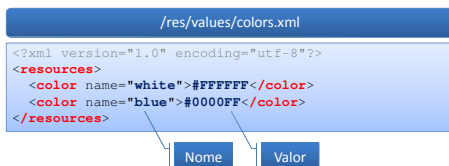
- Representam textos (strings) como recursos
- Estão localizados no diretório **/res/values**
- A representação dos resources é feita em XML
 - O nome do XML não é importante para o Android



Color Resources



- Semelhantes às string resources
- Também localizados em **/res/values**



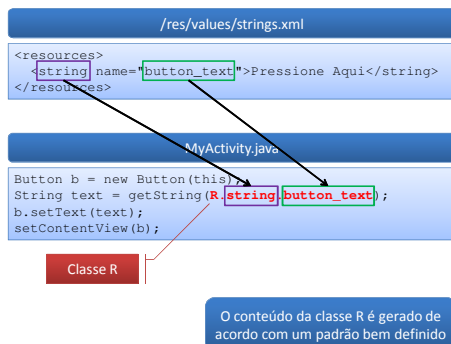
É possível misturar os tipos de resources dentro de um mesmo arquivo XML

A Classe R



- A classe **R** é um recurso muito interessante do Android para referenciar resources no código
- Ela é gerada automaticamente

Usando a Classe R



Implementação da Classe R



```

R.java

/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found. It
 * should not be modified by hand.
 */

package softblue.android;

public final class R {
    public static final class attr {
    }
    public static final class string {
        public static final int button_text=0x7f050003;
        public static final int hello_world=0x7f050002;
    }
}

```

A classe não deve ser editada

Inner classes e constantes representam os resources

Drawable Resources


- Resources que representam imagens
- Localizados no diretório **/res/drawable**
 - Extensões .png, .jpg e .gif são suportadas




MyActivity.java


```

ImageView img = new ImageView(this);
img.setImageResource(R.drawable.android);
setContentView(img);

```

Layout Resources


- Permitem representar o layout das telas da aplicação em formato XML
- A composição da interface fica separada do código-fonte da aplicação
- Utilizar arquivos de layout é a forma recomendada para criar a interface gráfica da aplicação
- Os layouts ficam localizados em **/res/layout**

Arquivo de Layout


/res/layout/main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

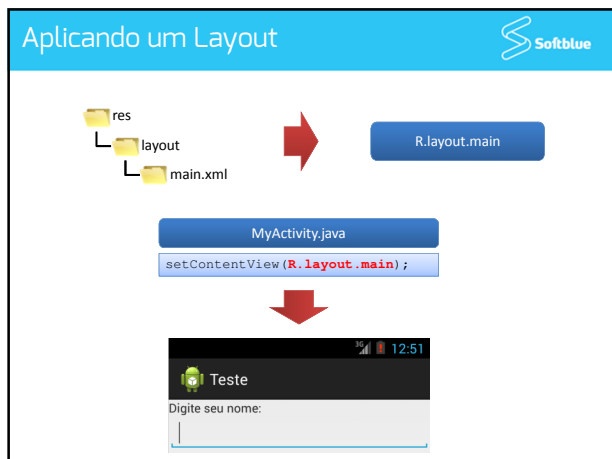
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Digite seu nome:"
    />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
    />
</LinearLayout>

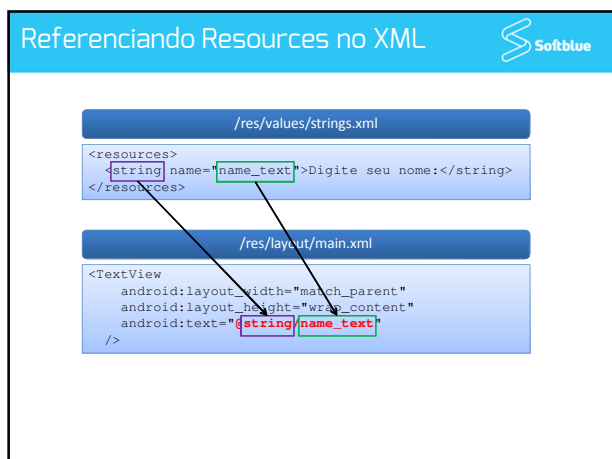
```

Texto

Caixa de texto



- ### Referenciando Resources no XML
- A classe R é utilizada para referenciar resources no código
 - Às vezes é necessário fazer a referência dentro de outro resource (definido em um XML)
 - Layout resources são exemplos típicos
 - O Android possui uma convenção para este tipo de referência



Referenciando Resources no XML



- Convenção para referenciar resources

Resource	Dentro do Código	Dentro do XML
String	R.string.<nome_res>	@string/<nome_res>
Color	R.color.<nome_res>	@color/<nome_res>
Drawable	R.drawable.<nome_res>	@drawable/<nome_res>
...

Definindo Resource IDs



- Algumas vezes é necessário definir seus próprios IDs para resources para referência posterior dentro do código
 - Um exemplo típico é a recuperação de dados fornecidos pelo usuário na interface gráfica

Exemplo de Definição de ID



/res/layout/main.xml

```
<EditText  
    android:id="@+id/name_box"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
>
```

MyActivity.java

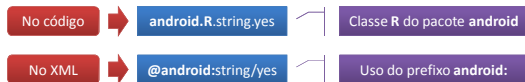
```
EditText e = findViewById(R.id.name_box);  
String s = e.getText().toString();
```

@+id/<algum_id> permite criar um identificador para o componente da interface gráfica

System Resources



- Além dos resources que você define, o próprio Android tem alguns previamente definidos
- Os system resources são referenciados pela classe **android.R**



Outros Tipos de Resources



- Existem outros tipos de resources no Android

Resource	Descrição
anim	Define animações
menu	Define os menus da aplicação
raw	Arquivos de qualquer formato que não são compactados
xml	Arquivos XML
...	...

- Resources também podem ser específicos para uma língua ou hardware
 - Durante a execução, o Android escolhe qual resource utilizar

Arquivos JAR Externos



- É comum aplicações Java precisarem referenciar arquivos JAR externos
 - Estes arquivos contêm classes que podem ser usadas pela aplicação
- Aplicações em Android também podem tirar proveito disso
 - Basta adicionar os arquivos JAR ao classpath da aplicação e eles serão empacotados junto com o código da aplicação que será instalada em um dispositivo
 - Adicionar os arquivos JAR à pasta *libs* do projeto também funciona

Logging



- Utilizar logs é uma forma de mostrar informações úteis durante a execução do programa
- A forma mais simples de fazer isso no Java é através de `System.out.println()`
 - Não é possível categorizar o erro

LogCat



- O LogCat é um mecanismo versátil de logging do Android
- Acessado através da classe **android.util.Log**, que possui alguns métodos estáticos

Método	Tipo de Mensagem
Log.v()	Verbose (depuração detalhada)
Log.d()	Debug (depuração)
Log.i()	Info (informação)
Log.w()	Warning (aviso)
Log.e()	Error (erro)

Usando Logging no Código



- Invocar os métodos da classe *Log* de acordo com o tipo de mensagem

```
Log.i("Tela Inicial", "Iniciando o processo");
```

Log de mensagem de informação

A tag permite categorizar a mensagem

API de Compatibilidade



- Também chamada de *support library*
- Novas versões do Android costumam trazer novas funcionalidades
- O que fazer se você quiser desenvolver um aplicativo que suporte recursos mais novos em dispositivos mais antigos?
- A API de compatibilidade torna isso possível
- Ela ainda traz alguns utilitários que facilitam o uso de alguns recursos

API de Compatibilidade



- A integração da API de compatibilidade com o projeto é feita através de um ou mais arquivos JAR
- Exemplos
 - **v4 support library**
 - Android 1.6+
 - Notification, view pager, drawer layout
 - **v7 support library**
 - Android 2.2+
 - Action bar, dialogs
 - **v13 support library**
 - Android 3.2+
 - Suporte a fragments

Permissões de Acesso



- Para realizar determinados tipos de operações, a aplicação deve solicitar permissão
- As permissões são configuradas no *AndroidManifest.xml*

```
<manifest>
<uses-permission
  android:name="android.permission.RECEIVE_SMS"
/>
</manifest>
```

Exemplos de Permissões de Acesso



Permissão	Descrição
android.permission.RECEIVE_SMS	Recebimento de SMS
android.permission.VIBRATE	Ativação da vibração do telefone
android.permission.READ_CONTACTS	Leitura dos contatos cadastrados no dispositivo
android.permission.INTERNET	Acesso à internet
android.permission.CALL_PHONE	Realização de chamadas telefônicas
android.permission.CAMERA	Acesso à câmera

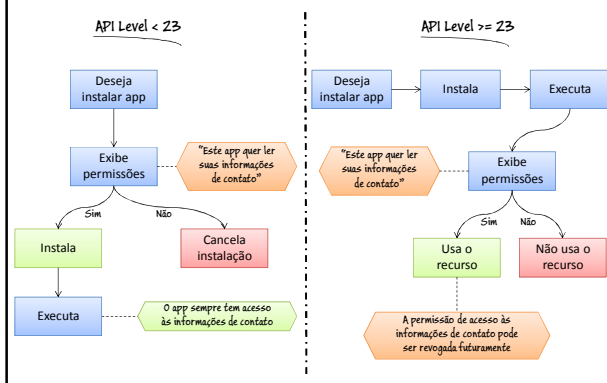
- A lista completa pode ser obtida na documentação do Android

Modelos de Permissões



- Até o Android 5 (API Level 22), as permissões eram checadas no momento da instalação
 - A instalação do aplicativo garantia o aceite das de todas as permissões
- A partir do Android 6 (API Level 23), as permissões são verificadas durante a execução do aplicativo
 - Podem ser aceitas ou revogadas a qualquer momento

Modelos de Permissões



Tipos de Permissões



- Existem dois tipos de permissões
 - Normais
 - Basta declarar no *AndroidManifest.xml* para funcionar
 - Exemplos
 - `android.permission.INTERNET`
 - `android.permission.SET_TIME_ZONE`
 - Perigosas
 - Manipulam dados relacionados à privacidade
 - Precisam ser declarados no *AndroidManifest.xml* e checadas durante a execução
 - Exemplos
 - `android.permission.ACCESS_FINE_LOCATION`
 - `android.permission.CALL_PHONE`

Grupos de Permissões



- Toda permissão pertence a um grupo
- Durante a execução o Android autoriza ou nega a permissão para o grupo

`android.permission-group.STORAGE` { `android.permission.READ_EXTERNAL_STORAGE`
`android.permission.WRITE_EXTERNAL_STORAGE`

Se o grupo for autorizado, todas as permissões dele são autorizadas automaticamente. O mesmo vale para a negação da permissão.