ANDROID

Material de estudo – Baseado na Documentação Oficial do Android

Prof. Diego Marques de Carvalho

Atividades (Activity)

Activity é um componente de aplicativo que fornece uma tela com a qual os usuários podem interagir para fazer algo, como discar um número no telefone, tirar uma foto, enviar um e-mail ou ver um mapa. Cada atividade recebe uma janela que exibe a interface do usuário. Geralmente, a janela preenche a tela, mas pode ser menor que a tela e flutuar sobre outras janelas (GOOGLE, 2017).

- Aplicativos geralmente possuem várias atividades;
- Normalmente um aplicativo possui uma atividade que é declarada como “principal” (main) no documento de manifesto que é a apresentada ao usuário ao iniciar o aplicativo pela primeira vez;
- Cada atividade é um ponto de partida para iniciar outra atividade para executar diferentes ações;
- Ao iniciar uma nova atividade, a atividade anterior é interrompida, mas o sistema conserva a atividade em uma pilha (a "pilha de retorno");
- Quando uma atividade é interrompida devido ao início de uma nova atividade, ela é notificada acerca dessa alteração de estado por meio de métodos de retorno de chamada do ciclo de vida da atividade.

Gerenciamento do ciclo de vida da atividade

À medida que o usuário navega no aplicativo, sai dele e retorna a ele, as instâncias de Activity no aplicativo transitam entre diferentes estados no ciclo de vida. Por exemplo, quando a atividade inicia pela primeira vez, ela fica em primeiro plano no sistema e tem o foco do usuário. Durante o processo, o sistema Android chama uma série de métodos do ciclo de vida na atividade, onde você define a interface do usuário e outros componentes. Se o usuário realiza uma ação que começa outra atividade ou muda para outro aplicativo, o sistema chama outro conjunto de métodos de ciclo de vida de sua atividade como ele se move para o fundo (onde a atividade não é mais visível, mas a instância e seu estado permanece intacta)(GOOGLE, 2017).

Dentro dos métodos de retorno de chamada do ciclo de vida, você pode declarar como a atividade deve se comportar quando o usuário sai e retorna da atividade. Por exemplo, se você estiver criando um reprodutor de vídeos de transmissão, poderá pausar o vídeo e encerrar a conexão de rede quando o usuário alternar para outro aplicativo. Quando o usuário retornar, poderá reconectar a rede e permitir que ele reinicie o vídeo de onde parou(GOOGLE, 2017).

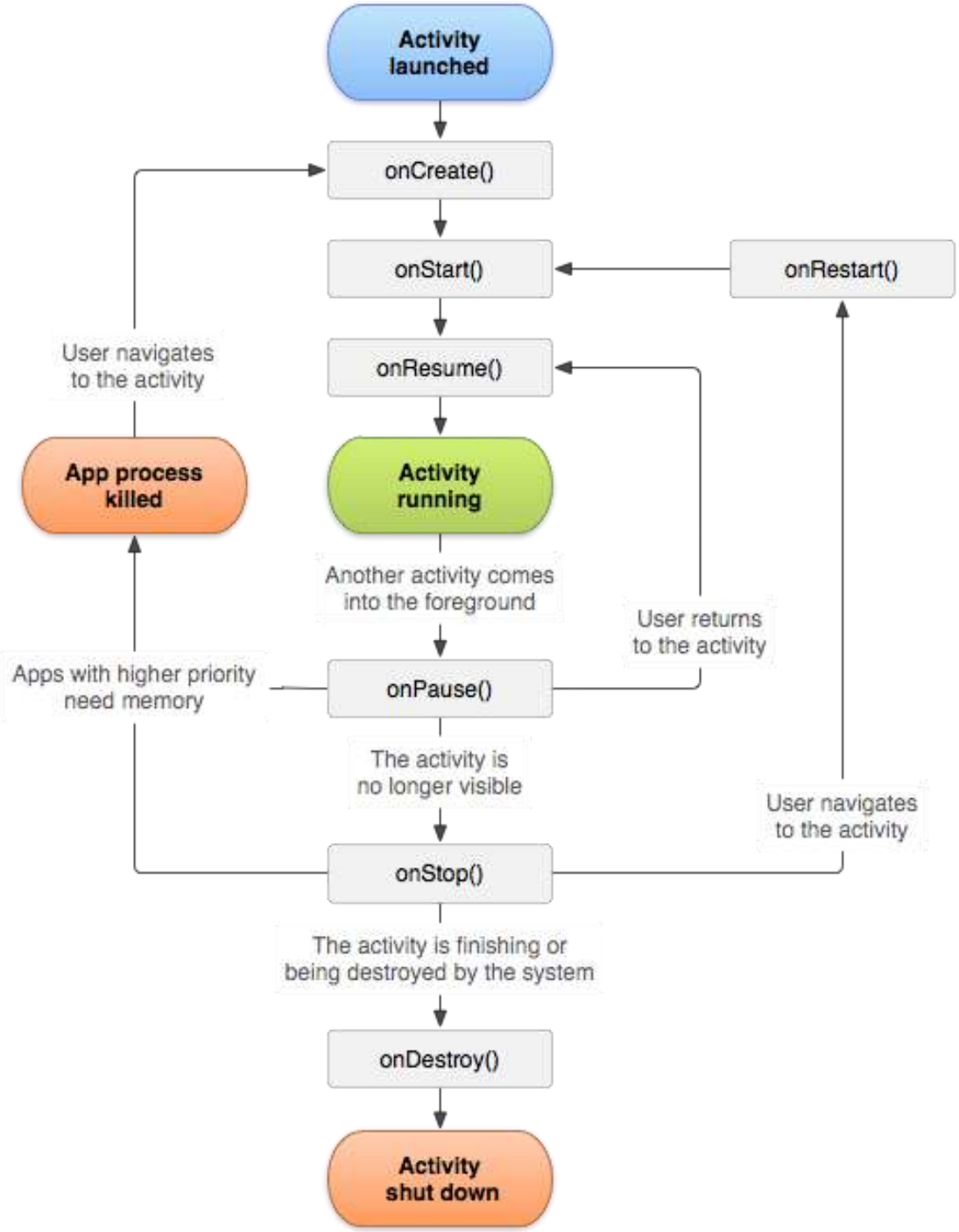


Figura 1 - Ciclo de vida de uma atividade Android

Observe a classe a seguir, ela é uma atividade e possui os métodos do ciclo de vida do Android comentados para ajudar na compreensão.

```
1.  //'extends' aplica o conceito de herança para a classe
2.  public class ExampleActivity extends Activity {
3.  //O @Override serve para reescrever um método que foi herdado, aonde seu comportament
    o na classe PAI, se difere do seu comportamento na classe filha. Ou seja, eles possue
    m o mesmo nome, mas, funcionalidades ou ações diferentes.
4.      @Override
5.  //public é o modificador de acesso do método. Usando este modificador o método pode s
    er acessado por qualquer classe dentro (e fora) do projeto.
6.  //Void significa vazio, ou seja o método não tem retorno
7.      public void onCreate(Bundle savedInstanceState) {
8.          //O 'super' chama o construtor da classe PAI
9.          super.onCreate(savedInstanceState);
10.        // A atividade está sendo criada.
11.     }
12.     @Override
13.     protected void onStart() {
14.         super.onStart();
15.         // A atividade está prestes a se tornar visível.
16.     }
17.     @Override
18.     protected void onResume() {
19.         super.onResume();
20.         // A atividade tornou-se visível (agora é "resumida").
21.     }
22.     @Override
23.     protected void onPause() {
24.         super.onPause();
25.         // Outra atividade está tomando foco
26.         (esta atividade está prestes a ser "pausada").
27.     }
28.     @Override
29.     protected void onStop() {
30.         super.onStop();
31.         // A atividade não é mais visível (agora está "parada")
32.     }
33.     @Override
34.     protected void onDestroy() {
35.         super.onDestroy();
36.         // A atividade está prestes a ser destruída.
37.     }
38. }
```

Tabela 1Resumo dos métodos de retorno de chamada do ciclo de vida da atividade (GOOGLE, 2017).

Método	Descrição		Pode ser eliminado depois?	Próximo
onCreate()	Chamado quando a atividade é criada pela primeira vez. É onde se deve fazer toda a configuração estática normal — criar exibições, vincular dados a listas etc. Esse método recebe um objeto Bundle (pacote) contendo o estado anterior da atividade, caso esse estado tenha sido capturado (consulte Gravação do estado da atividademais adiante). Sempre seguido de onStart().		Não	onStart()
	onRestart()	Chamado depois que atividade tiver sido interrompida, logo antes de ser	Não	onStart()

Método	Descrição		Pode ser eliminado depois?	Próximo
		reiniciada. Sempre seguido de onStart().		
	onStart()	Chamado logo antes de a atividade se tornar visível ao usuário. Seguido de onResume() se a atividade for para segundo plano ou onStop() se ficar oculta.	Não	onResume() ou onStop()
		onResume()	Chamado logo antes de a atividade iniciar a interação com o usuário. Nesse ponto, a atividade estará no topo da pilha de atividades com a entrada do usuário direcionada a ela. Sempre seguido de onPause().	Não	onPause()
		onPause()	Chamado quando o sistema está prestes a retomar outra atividade. Esse método normalmente é usado para confirmar alterações não salvas a dados persistentes, animações interrompidas e outras coisas que talvez estejam consumindo CPU e assim por diante. Ele sempre deve fazer tudo bem rapidamente porque a próxima atividade não será retomada até ela retornar. Seguido de onResume() se a atividade retornar para a frente ou de onStop() se	**Sim**	onResume() ou onStop()

Método	Descrição	Pode ser eliminado depois?	Próximo
	ficar invisível ao usuário.		
onStop()	Chamado quando a atividade não está mais visível ao usuário. Isso pode acontecer porque ela está sendo destruída ou porque outra atividade (uma existente ou uma nova) foi retomada e está cobrindo-a. Seguido de onRestart() se a atividade estiver voltando a interagir com o usuário ou onDestroy() se estiver saindo.	**Sim**	onRestart() ou onDestroy()
onDestroy()	Chamado antes de a atividade ser destruída. É a última chamada que a atividade receberá. Pode ser chamado porque a atividade está finalizando (alguém chamou finish() nela) ou porque o sistema está destruindo temporariamente essa instância da atividade para poupar espaço. É possível distinguir entre essas duas situações com o método isFinishing().	**Sim**	*nada*

Declaração de uma atividade no manifesto

É preciso declarar a atividade no arquivo de manifesto para torná-la acessível para o sistema. Para declarar a atividade, abra o arquivo de manifesto e adicione um elemento <activity> como filho do elemento <application> . Por exemplo (GOOGLE, 2017):

```
<manifest...>
<application...>
<activity android:name=".ExampleActivity" />
          ...
</application ... >
     ...
</manifest>
```

Existem outros atributos que podem ser incluídos nesse elemento para definir propriedades da atividade, como o rótulo, um ícone para ela ou um tema para estilizar sua IU. O atributo android:name é o único necessário — ele especifica o nome da classe da atividade. Depois de publicar o aplicativo, não se deve alterar o nome porque, se isso acontecer, podem ocorrer problemas em algumas funcionalidades, como os atalhos do aplicativo(GOOGLE, 2017).

Uso de filtros de intents

Um elemento <activity> também pode especificar vários filtros de intents — usando o elemento <intent-filter> — para declarar o modo com que os componentes de aplicativo podem ativá-lo. Ao criar um novo aplicativo com as ferramentas do Android SDK, o esboço da atividade criado contém automaticamente um filtro de intent que declara que a atividade responde à ação "main" (principal) e deve ser colocada na categoria "launcher" (inicializador). O filtro de intent tem a seguinte aparência (GOOGLE, 2017):

```
<activity android:name=".ExampleActivity"  android:icon="@drawable/app_icon">
   <intent-filter>
       <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

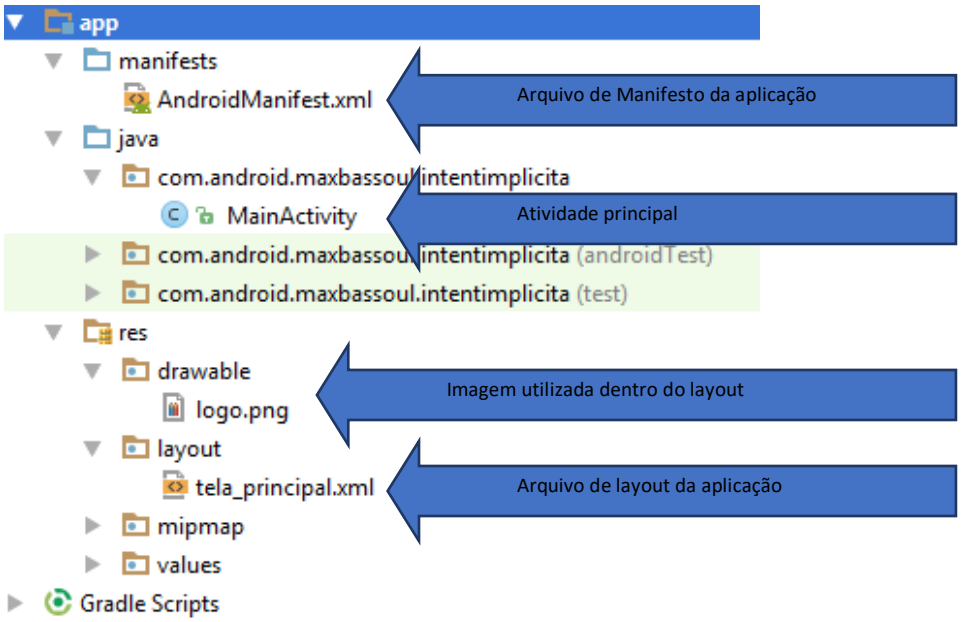
Início de uma atividade

Para iniciar outra atividade, é possível chamar startActivity() passando uma Intent que descreva a atividade que se deseja iniciar. O intent especifica a atividade exata que deve ser iniciada ou descreve o tipo de ação que ela deve executar (e o sistema seleciona a atividade adequada, que pode ser até de um outro aplicativo). Um intent também pode portar pequenas quantidades de dados a serem usados pela atividade iniciada. Ao trabalhar no aplicativo, frequentemente será necessário iniciar uma atividade conhecida. Para isso, pode-se criar um intent que defina explicitamente a atividade que deve ser iniciada por meio de um nome de classe. Por exemplo, eis como uma atividade inicia outra atividade de nome SignInActivity (GOOGLE, 2017):

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

Projeto Android comentado

Vamos criar um projeto que envia um texto utilizando o sistema nativo de SMS do Android. Abaixo a estrutura do projeto:



AndroidManifest.xml

```
//Declaração XML
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.android.maxbassoul.intentimplicita">

<application
android:allowBackup="true"//Permitir que o aplicativo participe da infra-estrutura de backup e
restauração
android:icon="@mipmap/ic_launcher"//Imagem de ícone do APP
android:label="@string/app_name"//Nome do APP
android:supportsRtl="true"//Declara se a sua aplicação está disposta a apoiar layouts da direita para a
esquerda (RTL)
android:theme="@style/AppTheme">//Tema do App
<activity android:name=".MainActivity">//Declaração da atividade
<intent-filter>//Filtro de intenção
<action android:name="android.intent.action.MAIN" />//Atividade declarada como principal,
ou seja, ela aparece primeiro para o usuário
<category android:name="android.intent.category.LAUNCHER" />
//LAUNCHER - A atividade é a atividade inicial de uma tarefa e é listada no inicializador do aplicativo do sistema
</intent-filter>
</activity>
</application>

</manifest>
```

MainActivity.java

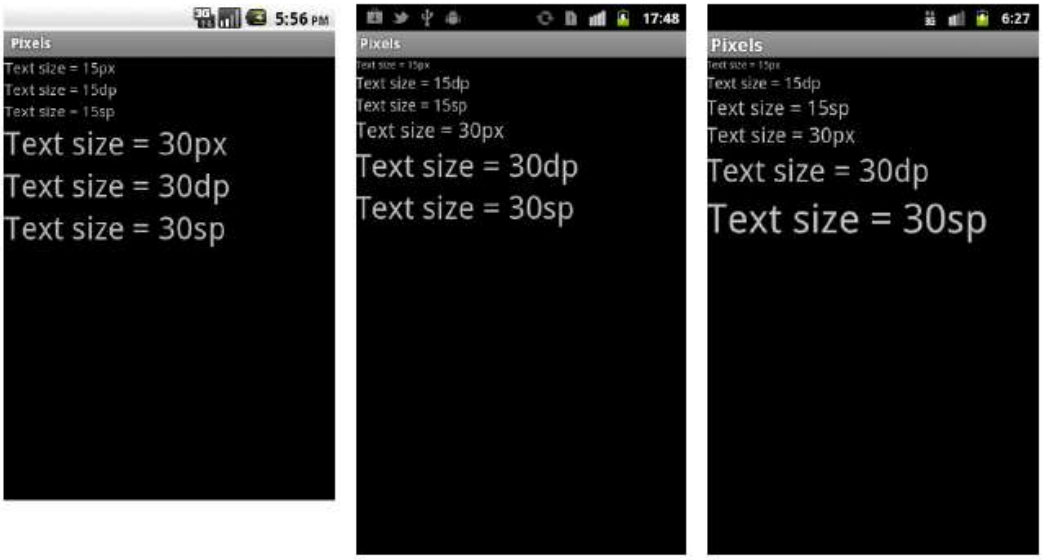
```
//Pacote onde está esta classe
package com.android.maxbassoul.intentimplicita;

import android.content.Intent; //Para utilizarmos a Intent
import android.os.Bundle; //Para salvar informações de estado
import android.support.v7.app.AppCompatActivity; //Classe base
import android.view.View; //Esta classe representa o bloco de construção
básico de componentes de interface do usuário
import android.widget.Button; //Para utilizarmos um botão
//Classe MainActivity do aplicativo
public class MainActivity extends AppCompatActivity {
//Atributo para botão
private Button btn_sms;
//Chamado quando a atividade é criada
@Override
protected void onCreate(Bundle savedInstanceState) {
        //Chama a versão da superclasse (PAI)
        super.onCreate(savedInstanceState);
        //Exibe a interface do usuário
        setContentView(R.layout.tela_principal);

       //Obtém referência para o botão manipulado via programação
btn_sms = (Button) findViewById(R.id.btn_sms);
//Utilização típica do botão para ativar uma ação no click
btn_sms.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
//Intent: Uma intenção é uma descrição abstrata de
                 umaoperação a ser executada
               //Criamos uma intent de nome 'sendIntent' e iniciamos
                ela com 'new'
Intent sendIntent = new Intent();
//Vamos utilizar a intent e atribuir uma ação que envia
                algo para o app de SMS
sendIntent.setAction(Intent.ACTION_SEND);
//O adiciona dados estendidos à intenção
               //Colocamos o texto via putExtra na Intent que se
                chama sendIntent (Nome que criamos para ela)
sendIntent.putExtra(Intent.EXTRA_TEXT, "Uma mensagem de texto...");
//setType: Definir um tipo de dados MIME explícito
               //MIME - do inglês Multipurpose Internet Mail
                Extensions, uma norma da internet para o formato
                das mensagens de correio eletrônico.
sendIntent.setType("text/plain");
//O método startActivity() inicia uma instância
                especificada pela Intent
startActivity(sendIntent);
           }
        });
    }
}
```

tela_principal.xml

```
<?xml version="1.0" encoding="utf-8"?><!--Declaração XML -->
<!--Um layout que organiza seus filhos em uma única coluna ou uma única
linha.-->
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:background="#fffaad" //Cor de fundo do layout
android:orientation="vertical" //Orientação vertical
android:padding="10dp" //Margin interna de 10dp
android:gravity="center" //Todos os elementos alinhados ao centro
android:layout_width="match_parent" //Largura = Largura da tela
android:layout_height="match_parent">//Altura = Altura da tela
<TextView
android:fontFamily="casual"//Família de fonte
android:textStyle="bold"//Estilo da fonte
android:textSize="44sp"//Tamanho da fonte
android:text="PROJETO"//Texto
android:layout_width="wrap_content"//Largura = Tamanho do conteúdo
android:layout_height="wrap_content" />//Altura = Tamanho do conteúdo
<ImageView
android:layout_marginTop="20dp"//Margem externa de 20dp
android:src="@drawable/logo"//Localização da imagem
android:layout_width="wrap_content"//Largura = Tamanho do conteúdo
android:layout_height="wrap_content" />//Altura = Tamanho do conteúdo
<Button
android:textColor="#ffffff"//Cor do texto
android:background="#154908"//Cor do fundo do botão
android:layout_marginTop="10dp"//Margem do topo de 10 dp
android:layout_width="match_parent"//Largura = Largura da tela
android:layout_height="wrap_content"//Altura = Tamanho do conteúdo
android:text="ENVIAR SMS"//Texto
android:id="@+id/btn_sms" />//Identificador ID
</LinearLayout>
```

Density Bucket	Screen Density	Physical Size	Pixel Size
ldpi	120 dpi	0.5 x 0.5 in	0.5 in * 120 dpi = 60x60 px
mdpi	160 dpi	0.5 x 0.5 in	0.5 in * 160 dpi = 80x80 px
hdpi	240 dpi	0.5 x 0.5 in	0.5 in * 240 dpi = 120x120 px
xhdpi	320 dpi	0.5 x 0.5 in	0.5 in * 320 dpi = 160x160 px
xxhdpi	480 dpi	0.5 x 0.5 in	0.5 in * 480 dpi = 240x240 px
xxxhdpi	640 dpi	0.5 x 0.5 in	0.5 in * 640 dpi = 320x320 px

Unit	Description	Units Per Physical Inch	Density Independent	Same Physical Size On Every Screen
px	Pixels	Varies	No	No
in	Inches	1	Yes	Yes
mm	Millimeters	25.4	Yes	Yes
pt	Points	72	Yes	Yes
dp	Density Independent Pixels	~160	Yes	No
sp	Scale Independent Pixels	~160	Yes	No



Entenda a compatibilidade com várias telas no link

https://developer.android.com/guide/practices/screens_support.html?hl=pt-br

Serviços (Service)

Um **Service** é um componente do aplicativo que pode realizar operações longas e não fornece uma interface do usuário. Outro componente do aplicativo pode iniciar um serviço e ele continuará em execução em segundo plano mesmo que o usuário alterne para outro aplicativo. Além disso, um componente poderá se vincular a um serviço para interagir com ele e até estabelecer comunicação entre processos (IPC). Por exemplo, um serviço pode lidar com transações de rede, reproduzir música, executar E/S de arquivos, ou interagir com um provedor de conteúdo, tudo a partir do segundo plano(GOOGLE, 2017).

Um serviço pode essencialmente ter duas formas:

Iniciado: Um serviço é "iniciado" quando um componente do aplicativo (como um atividade) inicia-o chamando startService(). Quando iniciado, um serviço pode ficar em execução em segundo plano indefinidamente, mesmo que o componente que o iniciou seja destruído. Geralmente, um serviço iniciado realiza uma única operação e não retorna um resultado para o autor da chamada. Por exemplo, ele pode fazer download ou upload de um arquivo pela rede. Quando a operação for concluída, o serviço deverá ser interrompido(GOOGLE, 2017).

Vinculado: Um serviço é "vinculado" quando um componente do aplicativo chama bindService() para vinculá-lo. Um serviço vinculado oferece uma interface servidor-cliente que permite que os componentes interajam com a interface, enviem solicitações, obtenham resultados, mesmo em processos com comunicação interprocessual (IPC). Um serviço vinculado permanece em execução somente enquanto outro componente do aplicativo estiver vinculado a ele. Vários componentes podem ser vinculados ao serviço de uma só vez, mas quando todos desfizerem o vínculo, o serviço será destruído. Quando a atividade é fechado este serviço também morre(GOOGLE, 2017).

MainActivity.java

```
1.  package  com.android.maxbassoul.servicolog;
2.  import  android.content.Intent;
3.  import  android.os.Bundle;
4.  import  android.support.v7.app.AppCompatActivity;
5.  import  android.view.View;
6.  import  android.widget.Button;
7.  public  class  MainActivity  extends  AppCompatActivity  {
8.      private  Button  btn_mensagem;
9.      @Override
10.     protected  void  onCreate(Bundle  savedInstanceState)  {
11.         super.onCreate(savedInstanceState);
12.         setContentView(R.layout.tela_principal);
13.         btn_mensagem  =  (Button)  findViewById(R.id.btn_mensagem);
14.         btn_mensagem.setOnClickListener(new  View.OnClickListener()  {
15.             @Override
16.             public  void  onClick(View  v)  {
17.                 Intent  intent  =  new  Intent(MainActivity.this,
    DelayedMessageService.class);
18.                 intent.putExtra(DelayedMessageService.EXTRA_MESSAGE,  getResources().
    getString(R.string.mensagem));
19.                 startService(intent);
20.             }
21.         });
```

```
    }
}
```

DelayedMessageService.java

```
package com.android.maxbassoul.servicolog;
import android.app.IntentService;
import android.content.Intent;
import android.util.Log;
public class DelayedMessageService extends IntentService {
     //Usa uma constante para passar uma mensagem da atividade para o serviço
    public static final String EXTRA_MESSAGE = "messagem";
    public DelayedMessageService(){
       super("DelayedMessageService");
    }
//Este método contém o código a ser executado quando o serviço receber uma intenção
    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized(this) {
            try { //10 segundos
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        //Chama o método showText()
        showText(text);
    }
    private void showText(final String text) {
       //Registra um texto no log para que possamos //velo no logcat
        Log.v("DelayedMessageService", "A nossa mensagem é: " + text);
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android=http://schemas.android.com/apk/res/androidpackage="com.android.maxbass
oul.servicolog">
    <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:supportsRtl="true"
       android:theme="@style/AppTheme">
            <activity android:name=".MainActivity">
                <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category android:name="android.intent.category.LAUNCHER" />
                 </intent-filter>
            </activity>
            <service
                android:name=".DelayedMessageService"
```

```
                android:exported="false"></service>
        </application>
</manifest>
```

Transmissão (Broadcast)

As aplicações do Android podem enviar ou receber mensagens de transmissão do sistema operacional e de outras aplicações do celular, de forma semelhante ao publish-subscribe design pattern(GOOGLE, 2017).

Na arquitetura de software, publish-subscribe é um padrão de mensagens onde remetentes de mensagens, chamados editores, não programam as mensagens a serem enviadas diretamente para receptores específicos, chamados de assinantes, mas sim caracterizam as mensagens publicadas em classes sem conhecimento de quais assinantes, se houver , pode ser. Da mesma forma, assinantes expressam interesse em uma ou mais classes e só recebem mensagens que são de interesse, sem conhecimento de quais editores, se houver, existem (WIKIPÉDIA, 2017).

Estas transmissões são enviadas quando ocorre um evento de interesse. Por exemplo, o sistema Android envia transmissões quando ocorrem vários eventos do sistema, como quando o sistema é inicializado ou o dispositivo começa a ser carregado. As aplicações também podem enviar difusões personalizadas, por exemplo, para notificar outros aplicativos de algo que eles possam estar interessados (por exemplo, alguns novos dados foram baixados)(GOOGLE, 2017).

Os aplicativos podem se registrar para receber transmissões específicas. Quando uma transmissão é enviada, o sistema encaminha automaticamente as transmissões para os aplicativos que se inscreveram para receber esse tipo específico de transmissão(GOOGLE, 2017).

Em geral, as transmissões podem ser usadas como um sistema de mensagens entre aplicativos e fora do fluxo normal do usuário, ou seja, o usuário não precisa ver o serviço acontecendo pois ele opera em segundo plano(GOOGLE, 2017).

Provedores de Conteúdo

Provedores de conteúdo gerenciam o acesso a um conjunto estruturado de dados. Eles encapsulam os dados e fornecem mecanismos para definir a segurança dos dados. Provedores de conteúdo são a interface padrão que conecta dados em um processo com código em execução em outro processo(GOOGLE, 2017).

Exemplo de uma Splash Screen

A Splash Screen é uma tela de abertura muito utilizada em jogos e aplicativos mobile, ela pode servir como resposta gráfica a um processo que está em segundo plano ou simplesmente para apresentar a marca do desenvolvedor ou um anúncio.

Abaixo o exemplo de uma atividade que funciona como Splash Screen:

SplashActivity.java

```
1.  package com.tecnologia.flyover.myapplication;
2.  import android.app.Activity;
3.  import android.content.Intent;
4.  import android.os.Bundle;
5.  import android.os.Handler;
6.  public class SplashActivity extends Activity {
7.     // Tempo da splash screen
8.      private static int SPLASH_TIME_OUT = 4000;
9.      @Override
10.     protected void onCreate(Bundle savedInstanceState) {
11.         super.onCreate(savedInstanceState);
12.         setContentView(R.layout.activity_splash);
13.         new Handler().postDelayed(new Runnable() {
14.             //Exibindo splash com um tempo determinado
15.             @Override
16.             public void run() {
17.                 // Esse método será executado sempre que o tempo acabar
18.                 //E inicia a activity principal
19.                 Intent i = new Intent(getBaseContext(), MainActivity.class);
20.                 startActivity(i);
21. // Fecha esta activity
22.                 finish();
23.             }
24.         }, SPLASH_TIME_OUT);
25.     }
26. }
```

Contexto

Interface para informações globais sobre um ambiente de aplicação. Context é uma classe abstrata, cuja implementação é fornecido pelo sistema Android. Ela permite o acesso a recursos e classes específicas de aplicativos, bem como envio de chamadas para operações em nível de aplicativo como o lançamento de atividades, transmitindo e recebendo intenções, etc.

Context
public abstract class Context
extends Object

java.lang.Object
android.content.Context

getBaseContext

adicionado no nível API 1

Contexto getBaseContext ()

Retorno
Contexto contexto base como definido pelo construtor ou setBaseContext

Qual a diferença de this e getBaseContext()?

Ambos os elementos se referem a um contexto (contexto). No Android um context é um ponto de acesso para informações globais sobre um ambiente de aplicativo. Trata-se de uma classe abstrata cuja implementação é fornecida pelo sistema Android. Ela permite acesso a recursos e classes específicas de aplicativo, bem como chamadas para operações em nível de aplicativo como iniciar activities, enviar ou receber intents por broadcast, etc. Muitas vezes o 'this' refere-se a um contexto recém criado e não a qualquer objeto do contexto, diferente do getBaseContext(). No caso de uma intent() para iniciar a tela o efeito dos dois é o mesmo.

Simulado

https://goo.gl/forms/i1EqGOln3M00rvUn2

Referências

GOOGLE (Estados Unidos da América). Developers. Documentação oficial: Android. 2017. Licença Creative Commons Attribution 2.5. disponível em https://creativecommons.org/licenses/by/2.5/. Disponível em: <https://developer.android.com/index.html>. Acesso em: 10 fev. 2017.

Publish–subscribe pattern. (2016, December 19). In Wikipedia, The Free Encyclopedia. Retrieved 21:02, March 23, 2017, from https://en.wikipedia.org/w/index.php?title=Publish%E2%80%93subscribe_pattern&oldid=755661806