sãojudas universidade	FTCE Faculdade de Tecnologia e Ciências Exatas

USJT - 2019 – Programação para Web e Dispositivos Móveis

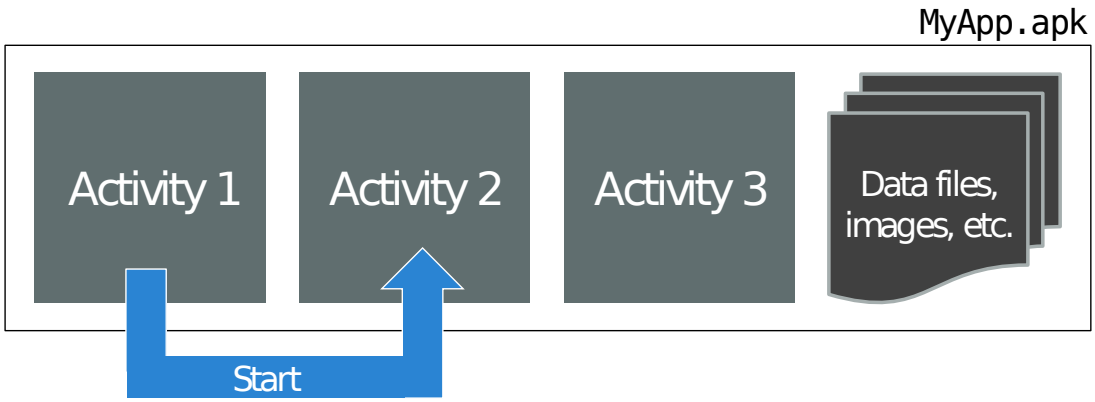
Professor: Anderson e Bossini

Aula: 03
Assunto: Activity - Intent - Stack Navigation

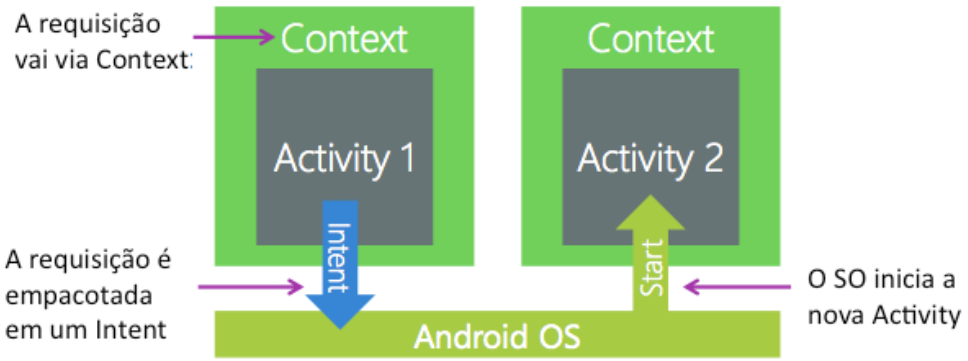
Conceitos básicos

Introdução

Como vimos na aula passada, uma aplicação Android é um conjunto de Activities.



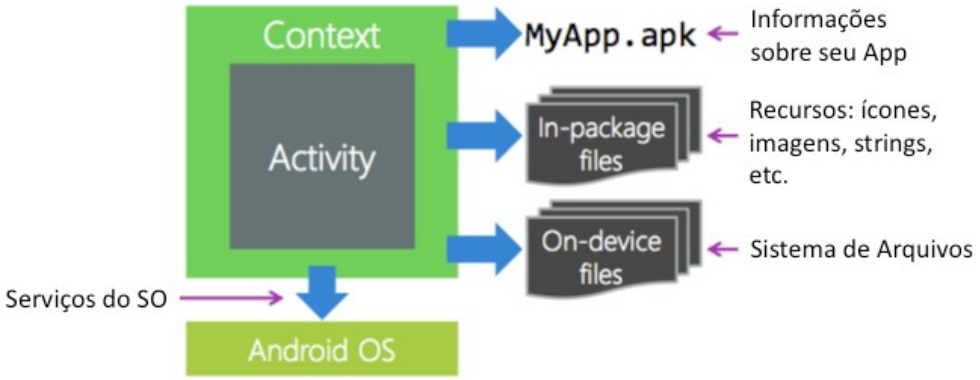
Visão geral do início de uma Activity



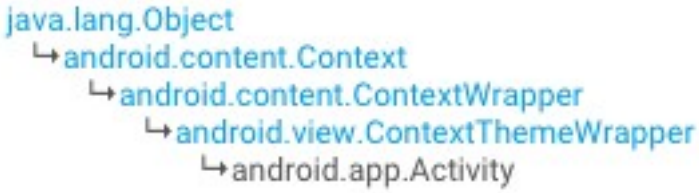
* SO é o Sistema Operacional, ou seja, o Android.

O que é um Context?

O Context é um ponto de acesso para o ambiente Android que está rodando seu app.



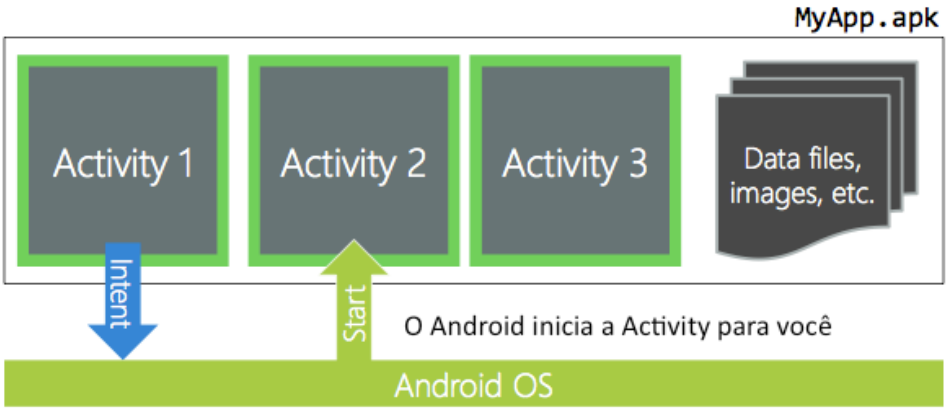
A Activity é subclasse de Context. Isso garante que cada Activity tenha acesso ao ambiente para carregar recursos e interagir com o Android.



O ContextWrapper é uma implementação do pattern Proxy e o ContextThemeWrapper dá suporte para estilos e temas.

O que é um Intent?

Um Intent é uma requisição para o Android começar uma nova Actvity.



O que é um Intent explícito?

É um Intent que identifica exatamente qual a Activity que será iniciada.

```
public Class Intent... {
   public Intent(Context packageContext, Class<?> cls) {
                  ...
   }
}
```

`Context packageContext` deve ser um Context associado com o **.apk** que contem a Activity que você quer iniciar. Use sua Activity atual, uma vez que ela é um Context e que está no mesmo .apk que a Activity que você quer iniciar.

`Class<?> cls` identifica univocamente a Activity a ser iniciada.

Métodos para Iniciar uma Activity

Estes métodos estão na classe Context. Os mais comuns são:

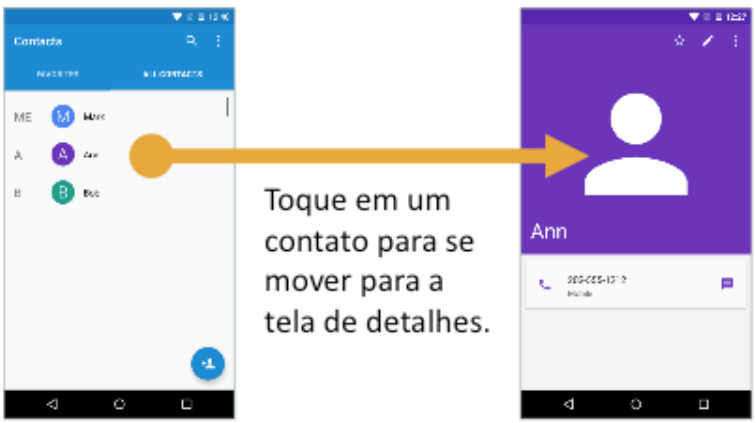
```
public abstract void startActivity(Intent intent);

public abstract void startActivity(Intent intent, Bundle options);
```

Como iniciar uma Activity

```
public class ActivityMain extends Activity{

   public void buscarClientes(View view) {

      Intent intent = new Intent(this, ListaClientesActivity.class);

      startActivity(intent);
    }
}
```

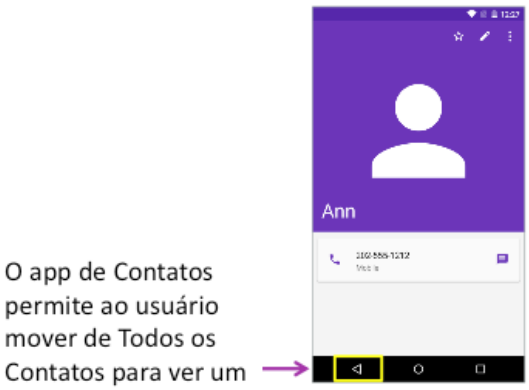
Navegação

A navegação descreve o caminho que você cria no seu App para permitir ao usuário se mover por várias Activities.



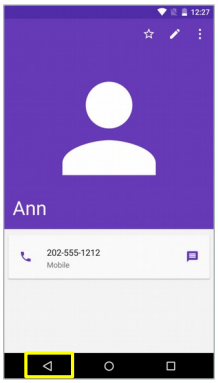
O botão Back

Permite ao usuário voltar para a Activity anterior.

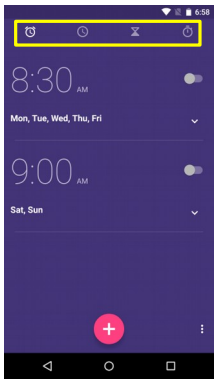


Padrões de Navegação

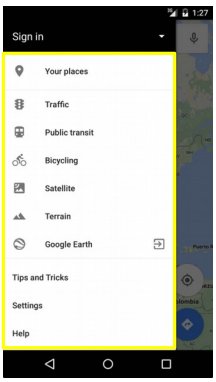
Existem vários no Android. Neste curso iremos aprender somente o Stack.



Stack



Tab



Drawer

Stack Navigation

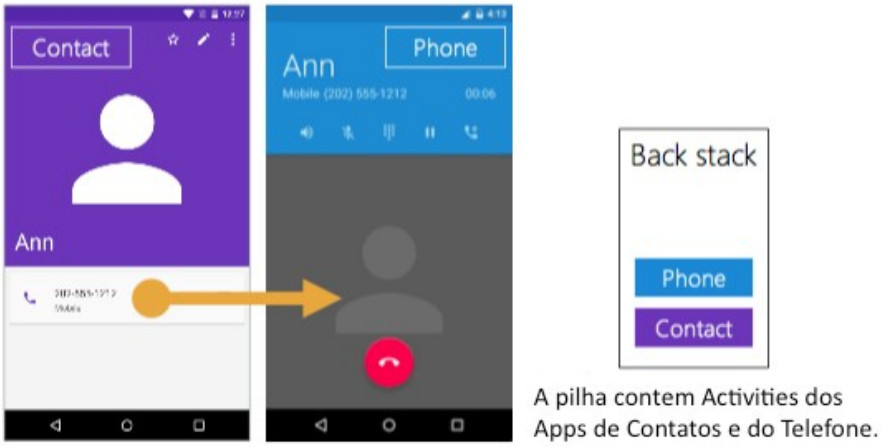
A navegação por pilha (stack navigation) armazena a sequência de Activities pelas quais o usuário passou em uma pilha (stack) e permite ao usuário retornar de qualquer Activity até o início.

O que é o back-stack?

É o histórico armazenado de todas as Activities vivas (live Activities) do usuário.

Escopo do back-stack

As Activities do back-stack podem se expandir por vários apps.



Push e Pop

O Android empilha (push) uma Activity no back-stack assim que você a inicia.

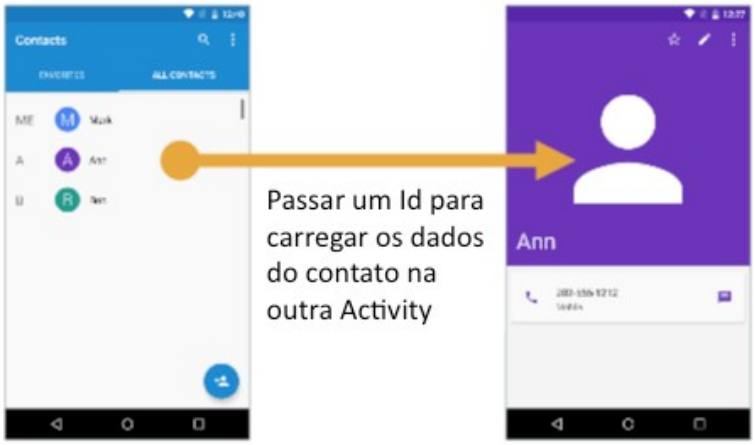
```
startActivity(intent);
```

E desempilha (pop) quando o usuário pressiona o botão Back.

Você pode também "chamar o botão Back" via código. Basta finalizar a Activity chamando o método `public void finish()`

Passagem de Dados entre Activities

As Activities geralmente precisam passar dados umas para as outras.

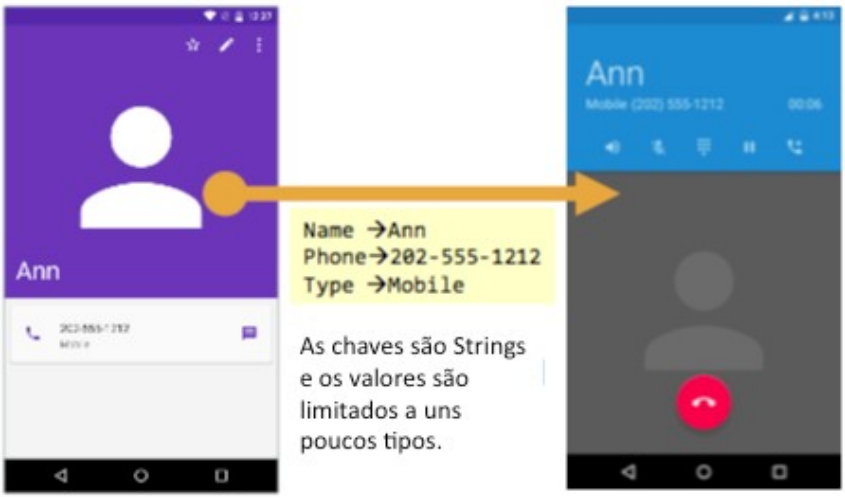


- Cada App roda no seu próprio processo.
- Cada Activity roda no processo do seu próprio App.

- Por este motivo, **apenas tipos primitivos e objetos serializáveis podem ser passados entre Activities;** não é possível passar referências a objetos, uma vez que não há compartilhamento de memória.

O que é um Bundle

Bundle é uma coleção de chave/valor passada entre Activities.



O Bundle tem métodos put/get para os tipos primitivos e strings.

Porém, existe um Bundle interno ao Intent chamado Intent Extra. Geralmente colocamos valores diretamente nele.

Colocar valores no Intent Extra

```
public class MainActivity extends AppCompatActivity {
    private EditText nome;
    public static final String CHAVE = "br.usjt.arqdesis.clientep1.chave";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        nome = (EditText)findViewById(R.id.busca_nome_cliente);

    }

    public void buscarClientes(View view){
        Intent intent = new Intent(this, ListaClientesActivity.class);
        String chave = nome.getText().toString();
        intent.putExtra(CHAVE, chave);
        startActivity(intent);
    }
}
```

Pegar valores do Intent Extra

```
        Intent intent = getIntent();
        String chave = intent.getStringExtra(MainActivity.CHAVE);
```

Recebendo resultados de uma Activity

Para obter um retorno de uma Activity, use o método da classe Activity

```
public void startActivityForResult (Intent intent, int requestCode)
```

Neste caso, a Activity retorna um código de status da sua execução para quem a chamou. Para configurar este status, a Activity chamada usa seu método:

```
public final void setResult (int resultCode)
```

Se a Activity chamada quiser também retornar dados, ela usa o Bundle interno do Intent a ser retornado usando o método

```
protected void onActivityResult (int requestCode, int resultCode,
Intent data)
```

A Activity chamadora pega este resultado chamando o método de callback, que é chamado imediatamente antes do onResume()

```
protected void onActivityResult (int requestCode, int resultCode,
Intent data)
```

Exemplo prático

Neste exemplo iremos criar uma aplicação que possui uma fila de chamados de help desk. Na primeira tela será possível digitar o nome de uma fila da qual desejamos ver os chamados. Na tela a seguir, uma lista exibe os chamados daquela fila. Uma vez que um item seja clicado, uma terceira tela mostra os detalhes de um chamado.

Passo 1 (Criando um novo projeto) Crie um novo projeto no Android Studio com os seguintes dados:

Template: Basic Activity
Application Name: Fila de Chamados
Package Name: br.usjt.filaChamados
Minimum API Level: API 22 Android 5.1 (Lollipop)

Mantenha os demais campos com seu valor padrão.

Passo 2 (Inspecionando os arquivos de layout) Observe que a aplicação possui dois arquivos de layout: activity_main.xml e content_main.xml. O primeiro define a toolbar e um FloatingActionButton, além de incluir o segundo. Por sua vez, o segundo servirá para abrigar o conteúdo principal da tela. Essa é uma forma de modularizar o desenvolvimento de interfaces gráficas.

Passo 3 (Ajustando o FloatingActionButton) O template nos entregou um FloatingActionButton que fica posicionado na parte inferior da tela e à direita. Além disso, ele exibe um ícone que lembra o envio de um e-mail. Vamos fazer as seguintes alterações no arquivo activity_main.xml.

3.1 Altere a propriedade layout_gravity para top | end.

3.2 Note que o botão subiu mas ficou em cima da barra de ação. Adicione a propriedade layout_marginTop e configure seu valor para 75dp, usando um par chave/ valor cuja chave deve ser fab_margin_top.

3.3 Remova a propriedade layout_margin pois ela tem precedência sobre qualquer outra específica.

3.4 Adicione as propriedades layout_marginBottom, layout_marginEnd e layout_marginStart configurando todas elas com o valor associado à chave fab_margin (que vale 16dp).

3.5 Clique com o direito na pasta drawable e escolha New >> Vector Asset. Clique no pequeno ícone do Android para visualizar a lista de ícones disponíveis. Procure pelo ícone chamado “check” e o selecione, clicando em Ok e depois Next e Finish.

3.6 No arquivo activity_main.xml, configure a propriedade do FloatingActionButton srcCompat como valor @drawable/ic_check_black_24dp.

Passo 4 (Adicionando o campo textual para o usuário digitar) Iremos utilizar um componente que permite que o usuário entre com texto e que, a princípio, exibe uma dica. Porém, quando o usuário tocar no componente, ao invés de a dica simplesmente sumir, ela será exibida na parte superior do componente com uma fonte um pouco

menor, ajudando o usuário a não esquecer para que serve o componente. Isso pode ser feito com o uso de dois componentes: um TextInputLayout que deve abrigar um TextInputEditText. Faremos as seguintes alterações no arquivo content_main.xml.

4.1 Remover a tag TextView.

4.2 Alterar o gerenciador de layout para LinearLayout.

4.3 Adicionar o conteúdo da Listagem 4.1 ao corpo do LinearLayout.

Listagem 4.1

```
<android.support.design.widget.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <android.support.design.widget.TextInputEditText
        android:id="@+id/nomeFilaEditText"
        android:hint="@string/dica_nome_fila"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</android.support.design.widget.TextInputLayout>
```

4.4 Note que o EditText usa um recurso de string. Vá em frente e crie-o associado ao valor “Digite o nome da fila”.

Passo 5 (Variável de referência para o EditText) A fim de obter o que o usuário irá digitar, precisamos de uma variável de referência para o EditText. Declare-a na classe MainActivity e inicialize-a no método onCreate, como mostra a Listagem 5.1.

Listagem 5.1

```
private EditText nomeFilaEditText;
//no método onCreate, depois de setContentView
nomeFilaEditText = findViewById(R.id.nomeFilaEditText);
```

Passo 6 (Tratando o evento no fab) Note que o fab já tem uma variável de referência e um observer registrado que reagirá quando ele for tocado. Iremos utilizar o método onClick do observer para obter o que o usuário digitou e navegar para a segunda tela, que será responsável por exibir os chamados da fila cujo nome foi digitado. Veja a implementação do método onClick na Listagem 6.1.

Listagem 6.1

```
public void onClick(View view) {
    String nomeFila =
            nomeFilaEditText.getEditableText().toString();
    Intent intent =
            new Intent (MainActivity.this,
ListaChamadosActivity.class);
    intent.putExtra("nome_fila", nomeFila);
    startActivity(intent);
}
```

Passo 7 (Criando a ListaChamadosActivity) O método onClick usa a classe ListaChamadosActivity que será criado agora. Para tal, clique com o direito no pacote principal da aplicação e escolha New >> Activity >> Empty Activity. O nome deve ser ListaChamadosActivity. Mantenha os demais campos com seu valor padrão.

Passo 8 (Ajustando o layout da ListaChamadosActivity) A segunda tela irá exibir uma lista com os chamados da lista cujo nome foi digitado. Um componente capaz de exibir uma lista de componentes visuais é o ListView. Vá em frente e atualize o arquivo activity_lista_chamados.xml com o código da Listagem 8.1.

Listagem 8.1

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ListaChamadosActivity">
    <ListView
        android:fastScrollEnabled="true"
        android:id="@+id/chamadosListView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

Passo 9 (A base de chamados) No momento iremos utilizar uma base fictícia na memória, representada por um ArrayList. Ela pode ser obtida por meio do método exibido na Listagem 9.1, que faz parte da ListaChamadosActivity.

Listagem 9.1

```
public List<String> geraListaChamados(){
    ArrayList<String> lista = new ArrayList<>();
    lista.add("Desktops: Computador da secretária quebrado.");
    lista.add("Telefonia: Telefone não funciona.");
    lista.add("Redes: Manutenção no proxy.");
    lista.add("Servidores: Lentidão generalizada.");
    lista.add("Novos Projetos: CRM");
    lista.add("Manutenção Sistema ERP: atualizar versão.");
    lista.add("Novos Projetos: Rede MPLS");
    lista.add("Manutenção Sistema de Vendas: incluir pipeline.");
    lista.add("Manutenção Sistema ERP: erro contábil");
    lista.add("Novos Projetos: Gestão de Orçamento");
    lista.add("Novos Projetos: Big Data");
    lista.add("Manoel de Barros");
    lista.add("Redes: Internet com lentidão");
    lista.add("Novos Projetos: Chatbot");
    lista.add("Desktops: troca de senha");
    lista.add("Desktops: falha no Windows");
    lista.add("Novos Projetos: ITIL V3");
    lista.add("Telefonia: liberar celular");
    lista.add("Telefonia: mover ramal");
    lista.add("Redes: ponto com defeito");
    lista.add("Novos Projetos: ferramenta EMM");
    return lista;
}
```

Passo 10 (Método para busca de chamados) A aplicação irá exibir os chamados relacionados à fila cujo nome foi digitado. Porém, se o usuário não digitar nada, ela exibirá todos os chamados. Essa busca é realizada pelo método da Listagem 10.1, também pertencente à classe ListaChamadosActivity.

Listagem 10.1

```
public List<String> buscaChamados(String chave){
    List<String> lista = geraListaChamados();
    if (chave == null || chave.length() == 0){
        return lista;
    }
    else {
        List<String> subLista = new ArrayList<>();
        for(String nome:lista){
            if(nome.toUpperCase().contains(chave.toUpperCase())){
                subLista.add(nome);
            }
        }
        return subLista;
    }
}
```

Passo 11 (Obtendo o nome da fila) A fim de obter o nome da fila (vindo da Activity anterior) a Activity atual precisa obter o Intent que deu origem à ela. Isso pode ser feito usando o método getIntent. Com o Intent em mãos, podemos extrair o valor de interesse usando a mesma chave que foi usada para o envio. Veja a Listagem 11.1.

Listagem 11.1

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_lista_chamados);
    Intent origemIntent = getIntent();
    String nomeFila = origemIntent.getStringExtra("nome_fila");
    final List <String> chamados = buscaChamados(nomeFila);
}
```

Passo 12 (Referência para a ListView) A ListView também precisa de uma variável de referência, declarada e inicializada como mostra a Listagem 12.1.

Listagem 12.1

```
private ListView chamadosListView;
//no método onCreate, depois de setContentView
chamadosListView = findViewById(R.id.chamadosListView);
```

Passo 13 (Criando um adapter) Temos agora um cenário em que dados (representados como um ArrayList) devem ser exibidos por um componente visual (neste caso, uma ListView). É claro que deve haver baixo acoplamento entre esses componentes. O ArrayList não pode saber detalhes de implementação da ListView e vice-versa. Um cenário muito parecido com o que ocorre quando usamos o padrão composto MVC. No caso do Android, iremos criar um objeto Adapter. Iremos informar ao Adapter três coisas: o contexto atual (a Activity, um layout a ser usado

para exibir cada item, e a coleção de itens a serem exibidos. Veja sua construção, que deve ser realizada no método onCreate, na Listagem 13.1. Além de construir o adapter, precisamos informar a ListView que ela deve consultá-lo quando desejar se atualizar graficamente.

Listagem 13.1

```
ArrayAdapter <String> adapter =
        new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1,chamados);
chamadosListView.setAdapter(adapter);
```

Passo 14 (Tratando o evento de toque em um item da lista) Precisamos registrar um observador que será notificado quando um item da lista for tocado. Ele possui um método onClick que irá abrir uma terceira Activity, responsável por exibir os detalhes do chamado escolhido. Veja a Listagem 14.1.

Listagem 14.1

```
chamadosListView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
        Intent intent =
                new Intent (ListaChamadosActivity.this,
DetalhesChamadoActivity.class );
        intent.putExtra("chamado_escolhido",
chamados.get(position));
        startActivity(intent);
    }
});
```

Passo 15 (Criando a classe de detalhes) Agora iremos criar a classe que exibe o nome e a descrição do chamado escolhido. Ela se chama DetalhesChamadoActivity. Para isso, clique com o direito no pacote principal da aplicação e escolha New >> Activity >> Empty Activity.

Passo 16 (Extraindo nome da fila e descrição do chamado) A Listagem 16.1 mostra como pegar o chamado escolhido e separar o nome de sua fila de sua descrição.

Listagem 16.1

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_detalhes_chamado);
    Intent origemIntent = getIntent();
    String chamadoEscolhido =
origemIntent.getStringExtra("chamado_escolhido");
    String [] partes = chamadoEscolhido.split(":");
    String nomeFila = partes[0];
    String descricaoChamado = partes[1];
}
```

Passo 17 (Exibindo o nome e a descrição em um CardView) Agora vamos ajustar o arquivo activity_detalhes_chamado.xml para exibir nome da fila e descrição do chamado em um CardView.

17.1 Adicione a dependência da Listagem 17.1 ao Gradle.

Listagem 17.1

```
implementation 'com.android.support:cardview-v7:28.0.0'
```

17.2 Ajuste o conteúdo do arquivo activity_detalhes_chamado.xml conforme mostra a Listagem 17.2.

Listagem 17.2

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_gravity="center"
    android:layout_width="200dp"
    android:layout_height="200dp"
    card_view:cardCornerRadius="4dp">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <TextView
            android:gravity="center_horizontal"

android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
            android:id="@+id/nomeFilaTextView"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"/>
        <TextView
            android:gravity="center_horizontal"

android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
            android:id="@+id/descricaoChamadoTextView"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1" />
    </LinearLayout>
</android.support.v7.widget.CardView>
```

17.3 Declare as variáveis de referência da Listagem 17.3 na classe de detalhes. Inicialize-as no onCreate.

Listagem 17.3

```
private TextView nomeFilaTextView;
private TextView descricaoChamadoTextView;
//no onCreate, depois de setContentView
nomeFilaTextView = findViewById(R.id.nomeFilaTextView);
descricaoChamadoTextView =
findViewById(R.id.descricaoChamadoTextView);
```

17.4 Por fim, complete o método onCreate como mostra a Listagem 17.4.

Listagem 17.4

```
nomeFilaTextView.setText(nomeFila);
descricaoChamadoTextView.setText(descricaoChamado);
```

Exercício Prático

1. Ajuste o aplicativo da aula passada (aquele que usa o GPS) para capturar localizações do usuário conforme ele se movimenta e guardar em um ArrayList. O ArrayList deve guardar as 50 localizações mais recentes. Entre cada par de localizações deve haver, ao menos, 200 metros de distância e a aplicação deve receber atualizações no intervalo de 2 em 2 minutos. Note que sua coleção de dados deve mostrar as 50 localizações mais recentes. Isso quer dizer que, quando a quinquagésima primeira for recebida, primeira deve ser removida da coleção.

2. A aplicação deve ter uma tela inicial com um FloatingActionButton. Quando clicado, a tela a seguir deve mostrar uma lista com as localizações. Basta mostrar latitude e longitude em cada item.

3. Ao clicar em um item, exiba um mapa próximo daquela localização.

Bibliografia

Android API, package android.app; disponível em http://developer.android.com/reference/android/app/package-summary.html ; consultado em 02/09/15.