03

Completando nosso sistema: Intents

Bem-vindo ao curso de Android do alura

Nesse curso, utilizamos o eclipse com o plugin do ADT, este que não é mais recomendado pelo Google, então recomendamos esse curso apenas se você está envolvido com algum projeto legado. Se você estar começando com o android recomendamos começar com esse curso : android-studio (www.alura.com.br/course/android-studio-do-zero-a-persistencia)

Como editar o aluno quando ele for selecionado? Simplesmente abrir a activity `Formulario` não é o suficiente, precisamos populá-lo com os dados do aluno específico. Como transferir essa informação, sem gerar muito código e classes internas?

Também deixamos preparados para que um `longClick` em um aluno abra as opções específicas de cada item do `listView` do nosso `listaAlunos` . Vamos agora implementá-las com algumas das funcionalidades de um smartphone, como sms e ligações telefônicas. Localizaremos o endereço num mapa usando a própria API do Google Maps integrado ao Android e navegaremos no site informado também através da conectividade do dispositivo.

Agora queremos criar a tela de edição de `Aluno` . Aliás, podemos aproveitar a tela de inserção criada anteriormente. Porém, como utilizar os dados do aluno selecionado no método `onItemClick` do listener registrado em `ListView listaAlunos` ?

A questão fica: como vou acessar um `Aluno` da activity `ListaAlunosActivity` de dentro da nossa `FormularioActivity` ?

Aqui existem várias opções que são bem conhecidas de desenvolvedores Swing: poderíamos ter um atributo estático, com um getter público em `ListaAlunosActivity` , como por exemplo `getAlunoSelecionadoParaEdicao` . Essa solução não tem um design muito elegante. Poderíamos eliminar o static se passássemos uma referência a instância de `ListaAlunosActivity` para o `FormularioActivity` , mas repare que isso não pode ser feito facilmente, já que **não somos responsáveis pelo ciclo de vida da Activity**, isso é de controle do Android. Repare que **nunca** demos `new` em uma Activity, assim como não fazemos isso com servlets, EJBs, e outros componentes que tem ciclo de vida cuidado por um container.

Para casos como esse, as `Intent` s possuem métodos para **passar informações de uma Activity a outra**. Para colocar um atributo, existe o `putExtra` , que adiciona elementos como em um mapa e possui sobrecargas para diversos tipos, como `int` , `String` e `Serializable` .

Analogamente, temos getters específicos para cada tipo, que podem ser invocados através do Intent que iniciou aquela Activity (método `getIntent`).

Então no `onItemClick` do listener registrado em `ListView listaAlunos` , vamos setar um atributo extra:

```
listaAlunos.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> adapter,
        View view, int posicao, long id) {

        Intent edicao = new Intent(ListaAlunosActivity.this,
                                   FormularioActivity.class);
        edicao.putExtra("alunoSelecionado",
            (Aluno) listaAlunos.getItemAtPosition(posicao));
        startActivity(edicao);
    }
});
```

Uma observação: caso seja escolhido passar o objeto inteiro para a outra Activity, é necessário que nossa classe `Aluno` implemente a interface `java.io.Serializable` .

Nome de parâmetros do putExtra

O Android pode também passar alguns parâmetros para a intent. Para evitar ter conflito de nomes de parâmetros, é sugerido que usemos um namespace junto com seus nomes, como, por exemplo, `br.com.caelum.alunos.alunoSelecionado` .

Agora, no nosso `FormularioActivity` , precisamos pegar as informações do `alunoSelecionado` . Ao mesmo tempo temos de tomar um cuidado: essa activity também é usada para inserir, então nem sempre haverá esse parâmetro. Caso ele não exista, trataremos como uma inserção.

Dentro do `onCreate` , precisamos fazer:

```
aluno = (Aluno) getIntent().getSerializableExtra("alunoSelecionado");
Button botao = (Button) findViewById(R.id.botao);
if (aluno == null) {
    aluno = new Aluno();
}
else {
    botao.setText("Alterar");
}
```

Além disso, caso haja um id no aluno que estamos editando, isso quer dizer que ele não é novo e precisamos utilizar seus valores antigos.

```
EditText nome = (EditText) findViewById(R.id.nome);
EditText telefone = (EditText) findViewById(R.id.telefone);
//...

nome.setText(aluno.getNome());
telefone.setText(aluno.getTelefone());
//...
```

Como resolvemos delegar a responsabilidade de lidar com os dados do formulario com o `FormularioHelper` vamos deixá-lo atualizar os dados da tela também:

```
public class FormularioHelper {
    private Aluno aluno;

    //... atributos e métodos criados anteriormente

    public void colocaNoFormulario(Aluno aluno) {
        nome.setText(aluno.getNome());
        telefone.setText(aluno.getTelefone());
        site.setText(aluno.getSite());
        nota.setProgress(aluno.getNota().intValue());
        endereco.setText(aluno.getEndereco());

        this.aluno = aluno;
    }
}
```

Poderíamos ter feito um `if (aluno.getId() != null)` , mas nem precisamos, pois, se for `null` , ele vai setar os textos para `null` , fazendo com que o android use os *hints* de cada campo.

Agora criamos um método no DAO para alterar um aluno. É muito análogo a inserção:

```
public void alterar(Aluno aluno) {
    ContentValues values = new ContentValues();

    values.put("nome", aluno.getNome());
    values.put("telefone", aluno.getTelefone());
    values.put("endereco", aluno.getEndereco());
    values.put("site", aluno.getSite());
    values.put("nota", aluno.getNota());
    values.put("foto", aluno.getFoto());

    getWritableDatabase().update(TABELA, values, "id=?",
            new String[] { aluno.getId().toString() });
}
```

E, quando o botão do formulário for clicado, precisamos verificar se é uma edição ou alteração:

```
AlunoDAO dao = new AlunoDAO(FormularioActivity.this);
if ( aluno.getId() != null ) {
       dao.alterar(aluno);
} else {
       dao.inserir(aluno);
}
dao.close();
finish(); // volta para a activity anterior
```

O *AndroidManifest.xml* é o arquivo da nossa applicação onde estão todas as principais configurações e registro das nossas *Activities*. Ele existe desde o hello world que fizemos, mas só agora teremos necessidade de conhecer um pouco mais de seu funcionamento:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="br.com.caelum.cursofj57"
      android:versionCode="1"
      android:versionName="1.0">

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:label="@string/app_name" android:name="ListaAlunosActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="5" />
</manifest>
```

Ele já vem com um editor extremamente prático no próprio plugin que torna desnecessário sequer olhar para este XML, embora seja bastante comum editá-lo à mão. Também é nele que registram-se novas Activities, receptores de *broadcast* (como veremos), etc.

Permissões

E, por fim, também é nele que são relatadas as permissões necessárias para rodar a sua aplicação. Quando ela for instalada em um Android, essas permissões serão listadas para que o usuário autorize sua utilização.

O Android controla, com permissões, o que as aplicações podem acessar. Seguem alguns exemplos mais comuns:

. `<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>` Permite usar provedores de localização

. `<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"/>` Permite acessar comandos adicionais de localização

. `<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION"/>` Permite a criação de provedores de localização simulados

. `<uses-permission android:name="android.permission.INTERNET"/>` Permite acesso à internet

. `<uses-permission android:name="android.permission.CALL_PHONE"/>` Permite fazer chamadas telefônicas

. `<uses-permission android:name="android.permission.RECEIVE_SMS"/>` Permite receber SMS

. `<uses-permission android:name="android.permission.CAMERA"/>` Permite acessar a câmera

. `<uses-permission android:name="android.permission.READ_CONTACTS"/>` Permite ler os contatos

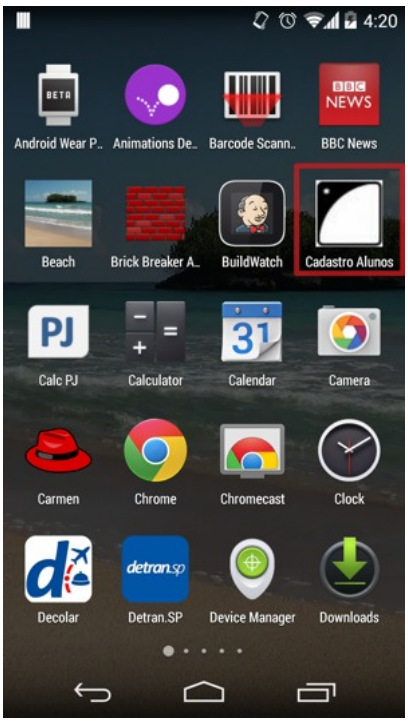
A lista completa de permissões pode ser encontrada nesta página da documentação do Android (http://developer.android.com/reference/android/Manifest.permission.html).

Ícone da sua aplicação

Para colocar um ícone no seu aplicativo, você terá que deixar o arquivo de imagem na pasta *drawable* da aplicação e declará-lo no seu *AndroidManifest.xml*.

Na tag **application** deve ser colocada a informação do local do ícone:

```
<application android:icon="@drawable/icon">
```



Intents Implícitas

Para efetuarmos ligações telefônicas com o Android, existem duas maneiras. Na primeira o Android permite que você acesse em baixo nível o telefone e controle as ligações telefônicas, repare que para isso, você precisaria ter um conhecimento profundo de como é o ciclo de vida de uma ligação telefônica.

Da segunda maneira, basta chamar o próprio aplicativo de ligação, caso o aparelho possua. Para que isto se torne funcional, vamos colocar mais uma opção no `ListaAlunosActivity`. Ele já tem um context menu **Ligar** que só aparecerá quando o aluno já existir, após um `clique longo`.

O Android possui **muitas** activities já prontas para utilizar as várias funcionalidades do dispositivo. O construtor da nossa `Intent` recebe uma `String` que representa a ação (*action*) que desejamos executar. Diversas dessas strings podem ser acessadas usando constantes declaradas na própria classe `Intent`.

No caso de ligar para um número, utilizamos `Intent.ACTION_CALL`. Mas para qual número o dispositivo vai realizar a chamada? Para isso temos também os dados que serão transmitidos para a outra activity, pelo método `setData`. A *action* e a *data* são as principais informações de uma `Intent`.

Esta chamada deverá ser colocada no método `onCreateContextMenu`. Utilizaremos uma `Intent`, que já utilizamos anteriormente para iniciar uma nova activity nossa:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);

    MenuItem ligar = menu.add(0,0,0, "Ligar");
    Intent intentLigar = new Intent(Intent.ACTION_CALL);
    intentLigar.setData(Uri.parse("tel:"+alunoSelecionado.getTelefone()));
    ligar.setIntent(intentLigar);

    //...
}
```

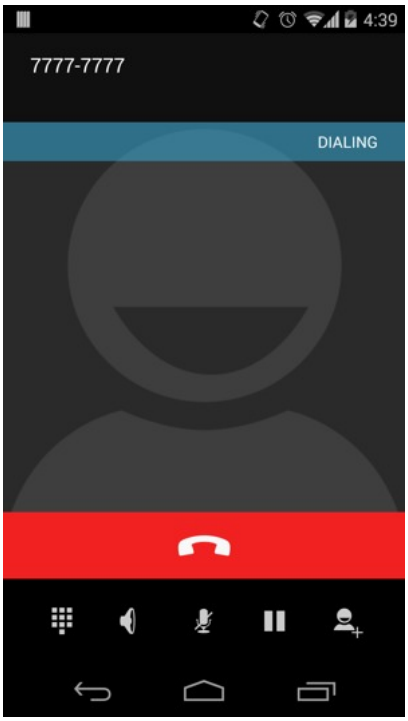
Se você quiser, poderá usar a `Intent.ACTION_DIAL` para chamar o discador somente, deixando a cargo do usuário discar os números.

Repare que estamos construindo um `Intent` diferentemente da maneira que vimos até agora. Antes sempre deixávamos bastante **explícito** qual era a activity que estávamos querendo abrir, passando inclusive sua classe (como `new Intent(ListaAlunosActivity.this, FormularioActivity.class)`). Aqui vamos apenas passar uma `String` que indica a nossa "intenção", e a própria plataforma decidirá, usando uma série de regras que podem ser customizadas, que activity vai responder ao chamado.

Esse tipo de intent é **implícito**: não sabemos exatamente quem vai ser iniciado, qual a activity específica. Dessa forma, cada dispositivo e aplicação pode configurar como cada tipo de `Intent` deve ser resolvida.

Não se esqueça da permissão no `AndroidManifest.xml`. Lembre-se que para acessar um recurso do dispositivo externo a aplicação, devemos declarar que precisamos das permissões específicas. Portanto, dentro da *tag* `<manifest>`, declaramos:

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



Repare que, se sua aplicação usar o `Intent.ACTION_DIAL`, não é sua app de fato quem estará fazendo a ação de ligar, sua *app* apenas abrirá o discador com o número de telefone por exemplo, e o usuário clicará em Ligar dentro do discador, neste caso, sua aplicação não precisaria da permissão, pois a ação não está partindo dela.

Gostaríamos também de enviar um SMS para o aluno contendo a nota, logo após a apuração de uma prova, por exemplo, então precisaremos de permissão para tal. Precisamos de algumas permissões no nosso *manifest*:

```
<uses-permission android:name="android.permission.READ_SMS" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.WRITE_SMS" />
```

Agora, vamos responder ao menu de SMS do `ListaAlunosActivity`. Repare que o código é parecido com o de ligar para um telefone específico, porém com outra *action* e *data*.

```
MenuItem sms = menu.add(0,1,0, "Enviar SMS");
Intent intentSms = new Intent(Intent.ACTION_VIEW);
intentSms.setData(Uri.parse("sms:"+alunoSelecionado.getTelefone()));
intentSms.putExtra("sms_body", "Mensagem");
sms.setIntent(intentSms);
```

Aqui utilizamos o método `putExtra` que serve para passar dados de uma intent para outra, além do `setData`. Veremos que podemos recuperar essa informação de dentro da nossa activity.

Isso abrirá a tela de envio de SMS da sua plataforma Android. Caso você queira enviar a mensagem de dentro da sua aplicação, sem chamar uma activity de fora, você pode usar a classe `SmsManager` que trabalha com envio direto:

```
SmsManager smsManager = SmsManager.getDefault();
PendingIntent sentIntent = PendingIntent.getActivity(this, 0, null, 0);

if (PhoneNumberUtils.isWellFormedSmsAddress(aluno.getTelefone())) {
    smsManager.sendTextMessage(aluno.getTelefone(), null,
            "Sua nota é " + aluno.getNota(), sentIntent, null);
    Toast.makeText(this,"SMS enviado com sucesso!!!",
        Toast.LENGTH_LONG).show();
```

```
} else {
    Toast.makeText(this, "Falha no SMS - tente novamente!!!",
        Toast.LENGTH_LONG).show();
}
```

A intent de tipo `Intent.ACTION_VIEW` pode abrir o nosso *browser*, no caso de uma URI ser passada como dado. Ficamos então com um código bem simples:

```
MenuItem site = menu.add(0,3,0, "Navegar no site");
Intent intentSite = new Intent(Intent.ACTION_VIEW);
intentSite.setData(Uri.parse(alunoSelecionado.getSite()));
site.setIntent(intentSite);
```

Opcionalmente, há também a possibilidade de criar uma nova Activity `VerSite` apenas para isso, com um `site.xml` próprio dentro de `res/layout`:

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

    <WebView
        android:id="@+id/vwweb"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Com um `WebView`, podemos visualizar sites da web, como se criássemos um navegador. Isso possibilitaria colocar botões e menus, além de ter o browser dentro de nossa tela. Em alguns casos pode ser necessário customizar a utilização do browser assim.

Dentro do `onCreate` dessa Activity, pegamos a setamos a view do `R.layout.site`:

```
super.onCreate(savedInstanceState);
setContentView(R.layout.site);

WebView view = (WebView) findViewById(R.id.vwweb);

String site = // pega URL que desejamos abrir
view.loadUrl(site);
```

A permissão necessária para esses casos é:

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

Lembrando que, caso usássemos a opção de criar uma nova Activity, precisamos declarar `VerSite` no `AndroidManifest.xml` :

```
<activity android:name=".VerSite" android:label="Site do aluno"></activity>
```

Vamos fazer agora uma Intent para usar o mapa do Android, somente para achar a posição do aluno conforme cadastrado. Novamente usamos `ACTION_VIEW` , mas passamos uma URI do tipo `geo:` .

```
MenuItem acharNoMapa = menu.add(0,2,0, "Achar no mapa");
Intent intentMapa = new Intent(Intent.ACTION_VIEW);
intentMapa.setData(
    Uri.parse("geo:0,0?z=14&q="+alunoSelecionado.getEndereco()));
acharNoMapa.setIntent(intentMapa);
```

Trabalharemos bastante com mapa um pouco mais para frente desse curso.

Para mandar um email, podemos utilizar uma intent de `ACTION_SEND` , com uma série de parâmetros a mais opcionais passados pelo `putExtra` :

```
MenuItem email = menu.add(0,5,0, "Enviar E-mail");
Intent intentEmail = new Intent(Intent.ACTION_SEND);
intentEmail.setType("message/rfc822");
intentEmail.putExtra(Intent.EXTRA_EMAIL,
     new String[] { "caelum@caelum.com.br" });
intentEmail.putExtra(Intent.EXTRA_SUBJECT, "Elogios do curso de android");
intentEmail.putExtra(Intent.EXTRA_TEXT, "Este curso é ótimo!!!");
email.setIntent(intentEmail);
```

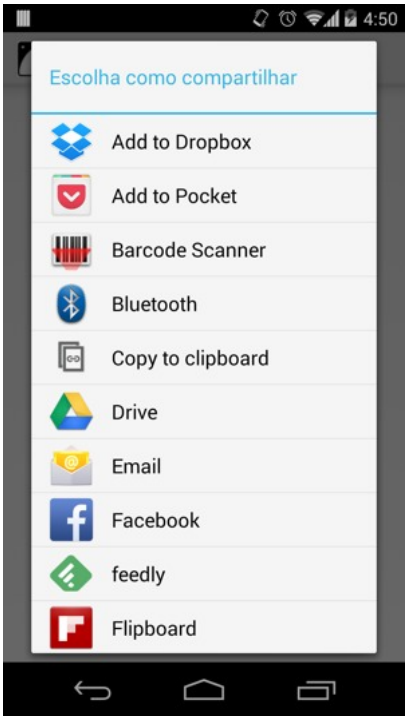
Assim como enviar email, diversas intents do Android podem ser realizadas por mais de uma aplicação. Nesse caso, o Android abre uma tela perguntando qual aplicação deverá ser utilizada, e possibilitar que uma delas seja selecionada como favorita. Há como forçar que esse painel de possibilidades sempre seja aberto, usando um `Intent.createChooser` . Como precisaremos invocar explicitamente o método `startActivity` precisamos registrar um `OnMenuItemClickListener` no item de menu:

```
MenuItem email = menu.add(0,5,0, "Enviar E-mail");
email.setOnMenuItemClickListener(new OnMenuItemClickListener() {
    @Override
    public boolean onMenuItemClick(MenuItem item) {
        //código anterior que cria a intent para enviar o email

        startActivity(Intent.createChooser(intentEmail,
            "Selecione a sua aplicação de email!"));
    }
});
```

Outra funcionalidade que está em alta hoje em dia nos principais dispositivos é o compartilhamento de conteúdo. Imagine que você quer mostrar para seus familiares ou amigos das redes sociais as quais pertence que venceu uma fase do jogo. Para isso, podemos chamar uma *Intent* implícita juntamente com o `createChooser` , para que seja exibida uma lista com todos os aplicativos que podem ser usados para compartilhar o conteúdo que você deseja:

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_SUBJECT, "assunto do que será compartilhado");
intent.putExtra(Intent.EXTRA_TEXT, "texto do que será compartilhado");
startActivity(Intent.createChooser(intent, "Escolha como compartilhar"));
```


4:50
Escolha como compartilhar
Add to Dropbox
Add to Pocket
Barcode Scanner
Bluetooth
Copy to clipboard
Drive
Email
Facebook
feedly
Flipboard