



INSTITUTO FED. DE EDUCAÇÃO, CIÊNC. E TEC. DE PERNAMBUCO
CURSO: TEC. EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS
PROFESSOR: RAMIDE DANTAS
ASSUNTO: PRIMERA APLICAÇÃO ANDROID

Aluno (a):			
Matrícula:		Data:	

Prática 01

Parte 1: Criando sua primeira aplicação

Passo 0: Prepare o ambiente

1. Baixe o Android Studio em:
<http://developer.android.com/sdk/index.html>
2. Instale na sua máquina
Tenha pelo menos 2 GB de memória RAM instalada (3 GB ou mais se for usar o emulador).
3. Configure o seu celular Android para ser usado durante os testes.
Em configurações (Settings), procure opções do desenvolvedor (Developer) e habilite "USB Debugging". (Da versão 4.3 em diante, a opção de Desenvolvedor está oculta. Para exibí-la, vá em Settings->About Phone e toque 7 vezes em "Build number")
Alternativamente, na falta de um dispositivo Android, use o emulador que acompanha o Android Studio, porém o desempenho da máquina será afetado.

Passo 1: Criando um novo projeto

1. Abra o Android Studio e na tela de abertura (Welcome screen) selecione "Start a new Android Studio project".
2. Dê um nome adequado a seu projeto:

Application name: Pratica01

[Nome da aplicação]

Company domain: pdm.tads.ifpe.edu.br

[Domínio da organização,
usado para dar nome aos
pacotes gerados]

Deixe os demais campos com os valores padrão. Ao preencher cada tela, pressione o botão **Next**.

Passo 2: Na próxima tela, selecione o tipo de aplicação a ser criado.

1. Escolhar "Phone and Tablet".
2. Na opção Minimum SDK, deixe a opção padrão (ex.: API 19: Android 4.4)
Essa opção diz qual o requisito mínimo para instalação da aplicação que você está desenvolvendo. Caso sua aplicação necessite de recursos de versões mais novas, mude essa opção de acordo.

Passo 3: Escolha um template de Activity para ser usado inicialmente na sua aplicação.

1. Escolha a opção “Basic Activity” e pressione Next.
2. Na tela seguinte, configure os campos como a seguir:


Activity Name: MainActivity
Layout Name: activity_main
Title: MainActivity
Menu Resource Name: menu_main.

Finalize a criação do projeto clicando em “Finish”.

Passo 4: Explore os arquivos que foram criados para o seu projeto:

1. `app->res->layout->activity_my.xml`: arquivo XML que descreve um “view container” introduzido com as versões recentes do Android. Usa um Layout chamado de CoordinatorLayout que vem preenchido com uma AppBar, e um Floating Button.
2. `app->res->layout->content_main.xml` arquivo XML que descreve o conteúdo visual a ser apresentado ao usuário no centro do CoordinatorLayout.
3. `app->java->[pacote]->MainActivity.java`: código java da atividade. Carrega o arquivo de layout da Activity para montar a interface.
4. `app->manifests->AndroidManifest.xml`: descreve as características da sua aplicação, seus componentes, permissões, etc.
5. `Gradle Scripts->build.gradle`: similar a um Makefile ou build.xml do Ant. Gradle é o software usado pelo Android Studio para montar a aplicação (compilar código e recursos e gerar o arquivo APK no final).
6. pasta `res`: contém os recursos (Resources) usados pela sua aplicação, como layouts, strings, menus, imagens, etc.

Passo 5: Teste a aplicação no seu dispositivo Android.

1. Certifique-se que o dispositivo está conectado.
2. Clique em  ou tecle Shift + F10 para executar a aplicação.
3. Escolha o seu dispositivo na tela que aparece e clique OK. (Caso esteja usando o emulador, escolher Launch Emulator.)
4. Teste a aplicação no seu dispositivo.

Parte 2: Trabalhando a interface com usuário

Passo 1: Mudando para um Linear Layout.

1. Abra o arquivo `content_main.xml` e visualise o seu XML (parte inferior, aba Text).
2. Remova o elemento `<TextView>` do xml.
3. Mude o elemento `<ConstraintLayout>` para `<LinearLayout>`.
4. Adicione o atributo `android:orientation` com valor “horizontal”.

Passo 2: Adicione um campo de texto.

1. No arquivo `content_main.xml`, adicione um elemento `<EditText>` como filho do elemento `<LinearLayout>`. Use como identificador do EditText (atributo `android:id`) o valor `"@+id/edit_message"`.

Identificadores (Ids) permitem referenciar objetos de outros lugares da aplicação. O símbolo `@` é usado para referenciar Ids dentro de XML. Ele é seguido pelo tipo de recurso, `"/` e o nome do recurso. O sinal de mais `+` indica um novo Id.

2. Defina os atributos `layout_width` e `layout_height` do EditText como `"wrap_content"`.

Essa opção diz para o campo ser grande o suficiente para conter o texto.

3. Defina a dica (atributo `hint`) do campo como sendo o objeto String com id `"@string/edit_message"`.

Como esse string não está definido ainda, o IDE apontará um erro.

O procedimento acima manipula diretamente o arquivo XML que descreve a UI da atividade. Alternativamente, também poderia ser usado o editor gráfico do IDE para adicionar o campo e modificar as propriedades. Experimente fazer isso e compare os resultados.

Passo 3: Adicionando strings à App

1. Abra o arquivo `strings.xml` (em `res->values`)
2. Adicione um elemento `<string>` com nome `"edit_message"` e valor `"Enter a message"`, como seguir:

```
<string name="edit_message">Enter a message</string>
```

3. Faça o mesmo para a string de nome `"button_send"` e valor `"Send"`.

Nesse ponto a aplicação já deve compilar corretamente. Rode a aplicação e veja se o campo aparece na tela como esperado.

Passo 4: Adicionando um botão a interface

1. Abra o arquivo `content_main.xml`
2. Dentro do elemento `<LinearLayout>` defina num novo botão (elemento `<Button>`) imediatamente depois do EditText
3. Configure o botão para ter largura do texto que ele contém (configure `width` e `height` para `"wrap_content"`).
4. Defina o texto do botão (propriedade `android:text`) como sendo o string `button_send` que definimos no passo anterior.

Experimente adicionar botões usando edição gráfica do Android Studio e veja o resultado.

Passo 5: Melhorando a apresentação da caixa de Entrada

1. No arquivo `content_main.xml`, modifique o elemento `EditText` para conter o atributo `android:layout_weight` com valor igual a `"1"`. Configure o `android:layout_width` com valor `"0pt"`.

O atributo peso (`android:layout_weight`) é usado para distribuir o espaço vazio entre elementos visuais. O espaço é distribuído proporcionalmente de acordo com o peso. Nesse caso, a configuração de largura (`"width"`) que havia antes (`"wrap_content"`) é ignorada e portanto foi substituída por (`"0pt"`).

Rode a aplicação e veja a diferença.

Parte 3: Tratando eventos

Passo 1: Tratando o botão "Send"

1. No arquivo `content_main.xml`, dentro do elemento `<Button>`, adicione o atributo `android:onClick` com valor `"sendMessage"`.
Esse é o nome do método que tratará esse evento na classe da atividade.
2. Em `MainActivity.java`, adicione o método com nome `"sendMessage"`.
O método deve ser público, com tipo de retorno `void` e receber como parâmetro um objeto do tipo `View` para poder ser usado para tratar o click do botão.
(O Android Studio cria o método automaticamente com `Alt-Enter`.)

Passo 2: Criando um "Intent"

1. No corpo do método `sendMessage` criado no passo anterior, crie um objeto do tipo `Intent`, como abaixo:

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

O IDE deve reclamar que `DisplayMessageActivity` não existe. Ignore o erro por hora.

2. Ainda no método `sendMessage`, pegue o objeto `EditText` correspondente a caixa de entrada através do seu `Id` usando o método `findViewById`, como abaixo:

```
EditText editText = (EditText) findViewById(R.id.edit_message);
```

3. Importe as classe `"android.content.Intent"` e `"android.widget.EditText"` em `MainActivity.java`.
Use `Alt+Enter` no Android Studio para importar as classes que estiverem faltando automaticamente.
4. Coloque o texto da caixa de entrada em uma variável do tipo `String` (e.g., `message`);

```
String message = editText.getText().toString();
```

5. Coloque a mensagem no Intent, como a seguir:

```
intent.putExtra(EXTRA_MESSAGE, message);
```

6. Declare EXTRA_MESSAGE como a variável estática na classe MainActivity:

```
public final static String EXTRA_MESSAGE =  
    "br.edu.ifpe.tads.pdm.Pratica01.MESSAGE";
```

O valor da string é arbitrário, desde que identifique unicamente o parâmetro que colocamos no Intent (por isso usamos o nome qualificado da aplicação).

7. Finalize o código do método sendMessage ativando o Intent:

```
startActivity(intent);
```

Compartilhando Informações entre Activities

Nessa prática a mensagem que foi digitada pelo usuário é comunicada à nova activity por meio de um “Extra” no Intent. Existem formas mais convenientes para trocar informações entre Activities de uma mesma App, por exemplo, por meio do padrão de projetos **Singleton**. Nesse caso, é criado um Singleton que armazena informações que podem ser acessadas por qualquer atividade da aplicação, por exemplo, dados sobre o usuário atual.

Parte 4: Criando uma nova Atividade

Passo 1: Criando uma nova atividade

1. Crie uma nova atividade clicando com o botão direito no painel de projeto em
app->New->Activity->Basic Activity
2. No assistente de criação de atividades, forneça os seguintes dados:
Activity Name: DisplayMessageActivity
Layout Name: activity_display_message
Title: My Message
Package Name: br.edu.ifpe.tads.pdm.Pratica01
Finalize a criação clicando em **Finish**. Rode e teste a aplicação e veja o que acontece.

Passo 2: Adicionando um TextView

1. Adicione um componente TextView no arquivo `content_display_message.xml`, com `android:id` igual a `"@+id/message_view"`
Configure a largura (`layout_width`) e altura (`layout_height`) com `wrap_content` (envolver conteúdo).

Passo 3: Recebendo o Intent

1. No método `onCreate()` da classe `DisplayMessageActivity`, obtenha o Intent que ativou a Atividade (importe a classe Intent depois):

```
Intent intent = getIntent();
```

2. Obtenha a mensagem associada ao Intent:

```
String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
```

Passo 4: Exiba a mensagem

1. Ainda no `onCreate()`, obtenho o objeto `TextView` criado no passo 2:

```
TextView textView = (TextView) findViewById(R.id.message_view);
```

2. Configure os atributos de tamanho do texto e mensagem como abaixo:

```
textView.setTextSize(40);  
textView.setText(message);
```

Rode e teste a aplicação. Nesse ponto ela deve ser capaz de exibir a mensagem digitada em uma nova tela.

não exibe a msg



Parte 5: Trabalhando com a App Bar

Atividades em branco criadas pelo Android Studio (versão 1.5) por default possuem um elemento Toolbar, o qual funciona como AppBar da nossa aplicação.

Passo 1: Criando opções de menu para o App Bar.

1. Adicione elementos `<item>` para cada um dos items abaixo dentro do elemento `<menu>` do arquivo `res->menu->menu_main.xml`:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

    <!-- Search, should appear as action button -->
    <item android:id="@+id/action_search"
          android:icon="@android:drawable/ic_search_category_default"
          android:title="@string/action_search"
          app:showAsAction="ifRoom" />

    <!-- Settings, should always be in the overflow -->
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          app:showAsAction="never" />

</menu>
```

O atributo `android:showAsAction` indica se a ação deve aparecer como botão na ActionBar ou como opção no menu lateral (chamado Overflow).

2. Crie as novas strings que estão faltando (`res->values->strings.xml`).

Passo 2: Respondendo às ações da App Bar.

1. Substitua o código do método `onOptionsItemSelected()` pelo código a seguir:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle presses on the action bar items
    switch (item.getItemId()) {
        case R.id.action_search:
            openSearch();
            return true;
        case R.id.action_settings:
            openSettings();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Esse código obtém o item que foi ativado pelo Id que você forneceu no XML de configuração (passo 1) e realiza uma ação correspondente.

2. Crie implementações vazias (Stubs) para os métodos `openSearch()` e `openSettings()`. (Dica: use o atalho `Alt+Enter` do Android Studio).
Dica: use um `Toast` para avisar ao usuário que a opção ainda não está implementada:

```
Toast.makeText(this, "Não implementada.", Toast.LENGTH_SHORT).show();
```

Passo 3: Melhorando a navegação entre as atividades via App Bar.

1. No arquivo `AndroidManifest.xml`, dentro do elemento `<activity>` de atividade `DisplayMessageActivity`, declare a atividade `MainActivity` como atividade-pai (parent activity) fazendo a seguinte modificação:

```
<activity
    android:name=".DisplayMessageActivity"
    android:label="@string/title_activity_display_message" ...>

    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>

</activity>
```

2. No arquivo `DisplayMessageActivity.java`, adicione o trecho de código abaixo ao final do método `onCreate()`:

```
ActionBar ab = getSupportActionBar();
ab.setDisplayHomeAsUpEnabled(true);
```

Essa mudança faz com que a Action Bar da atividade `DisplayMessageActivity` exiba uma ação de voltar (\leftarrow), que leva o usuário de volta à `MainActivity`.