Computação II – Orientação a Objetos

Fabio Mascarenhas - 2014.1

http://www.dcc.ufrj.br/~fabiom/java

Editor de Figuras

- Vamos usar nosso modelo de editor de figuras para fazer uma versão Android dele

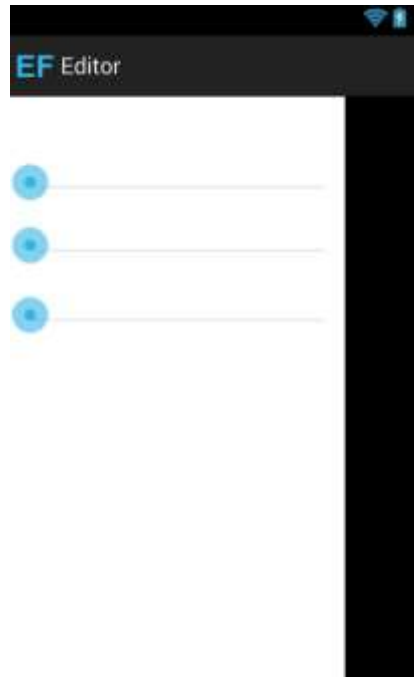
- Ao invés de usarmos botões, vamos usar toda a área da aplicação como área de desenho, e usar ações e itens de menu para controlar o modo atual do editor

- Vamos implementar uma view para ser o Canvas do editor, e conectá-lo com componentes Android implementando as outras interfaces do modelo

- Nosso canvas também vai capturar os eventos de clique e arrasto, e passá-los para um controlador que vai traduzi-lo nos métodos do modelo

Múltiplos layouts

- O layout da nossa atividade de escolha de cor não é o mais adequado quando dispositivo está na orientação “retrato”:



- Podemos criar outro arquivo `escolhe_cor.xml`, em um diretório `layout-port`, com o layout correto

Intents implícitos

- Uma terceira forma de criar um `Intent` é passando apenas uma *ação* que queremos fazer, e deixar o sistema escolher uma aplicação e uma atividade que pode fazer aquela ação

- Essa é a maneira que as aplicações se comunicam umas com as outras no sistema Android sem precisar acoplá-las umas às outras

- Vamos usar esse recurso para implementar outro recurso no editor: enviar o desenho atual como uma imagem

Armazenamento local

- Não podemos enviar uma imagem como extra no Intent, pois uma imagem pode ser grande demais para ter várias cópias dela na memória
- Precisamos primeiro gravá-la em alguma parte permanente do dispositivo
- Vamos gravá-la em um arquivo no *armazenamento local*
- O armazenamento local é uma área privativa da aplicação, e que é apagada quando a aplicação é desinstalada
- Acessamos o armazenamento local pelo métodos `getFilesDir`, ou pelos métodos `openFileInput` e `openFileOutput`

Salvando a Tela

- Usamos um `Bitmap` e outro `Canvas` para salvar o conteúdo da tela, e o método `compress` de `Bitmap` para gravá-lo como um arquivo PNG:

```
public void salvaTela(FileOutputStream arq) {
    Bitmap bm = Bitmap.createBitmap(this.getWidth(),
                                    this.getHeight(),
                                    Config.ARGB_8888);
    Canvas c = new Canvas(bm);
    this.draw(c);
    bm.compress(CompressFormat.PNG, 100, arq);
}
```

- Na atividade, criamos o arquivo com `openFileOutput` e passamos o resultado para `salvaTela`

Enviando

- Agora basta criar o Intent de enviar, e pedir pro sistema começar uma atividade que o responda:

```
Intent ienviar = new Intent(Intent.ACTION_SEND);
ienviar.setType("image/png");
Uri uri = Uri.fromFile(new File(getFilesDir(), "tela.png"));
ienviar.putExtra(Intent.EXTRA_STREAM, uri.toString());
startActivity(Intent.createChooser(ienviar, "Enviar com"));
```

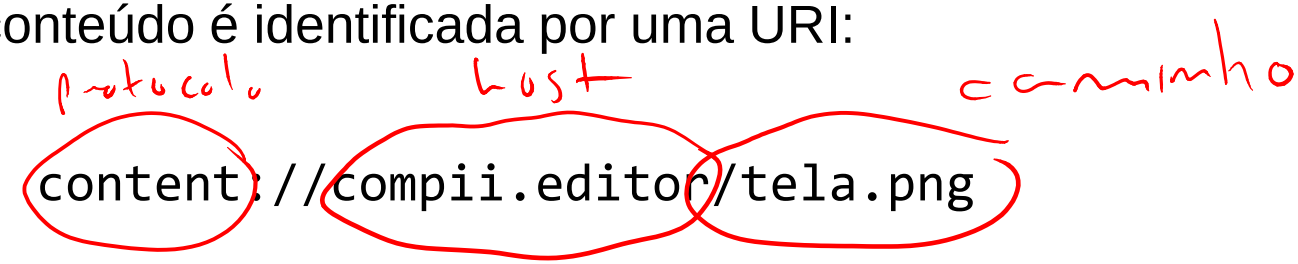
- Só que isso não funciona! O armazenamento local é privado da aplicação, então outras aplicações não conseguem ler o arquivo

- Precisamos criar um *provedor de conteúdo*

Provedor de Conteúdo

- Com um provedor de conteúdo uma aplicação pode compartilhar dados com outra aplicação

- Cada fonte de conteúdo é identificada por uma URI:



- A parte de *host* da URI identifica a aplicação, a parte de *caminho* identifica o conteúdo dentro da aplicação

- Conteúdo pode ser dados não estruturados, como um arquivo, ou uma *tabela* que responde a consultas, como uma tabela de um banco de dados

ContentProvider

- Um provedor de conteúdo é uma subclasse de `ContentProvider`

- Ele implementa uma série de métodos, para consultas a tabelas e a arquivos:

```
// Inicialização
public boolean onCreate();
// Tipo MIME do dado na uri
public String getType(Uri uri);
// Consulta linhas da tabela (como SELECT do SQL)
public Cursor query(Uri uri, String[] prj, String sel, String[] args, String sort);
// Insere linhas na tabela (como INSERT do SQL)
public Uri insert(Uri uri, ContentValues values);
// Apaga linhas da tabela (como DELETE do SQL)
public int delete(Uri uri, String sel, String[] args);
// Modifica linhas da tabela (como UPDATE do SQL)
public int update(Uri uri, ContentValues values, String sel, String[] selArgs);
// Abre e retorna um arquivo na uri
public ParcelFileDescriptor openFile(Uri uri, String mode)
   throws FileNotFoundException;
```

Provedor de conteúdo do Editor

- O provedor de conteúdo do editor só retorna um arquivo, então ele só implementa efetivamente openFile:

```
@Override
public ParcelFileDescriptor openFile(Uri uri, String mode)
          throws FileNotFoundException {
    File arq = new File(getContext().getFilesDir(), uri.getPath());
    return ParcelFileDescriptor.open(arq,
        ParcelFileDescriptor.MODE_READ_ONLY);
}
```

- O provedor também precisa de uma entrada no manifesto da aplicação:

```
<provider
            android:name="compii.editor.ProvedorArq"
            android:authorities="compii.editor"
            android:exported="true" />
```

Enviando

- Agora basta criar o Intent de enviar, e pedir pro sistema começar uma atividade que o responda:

```
Intent ienviar = new Intent(Intent.ACTION_SEND);
ienviar.setType("image/png");
Uri uri = Uri.parse(“content://compii.editor/tela.png");
ienviar.putExtra(Intent.EXTRA_STREAM, uri.toString());
startActivity(Intent.createChooser(ienviar, "Enviar com"));
```

- O emulador não tem muitas opções de aplicação que podem receber uma image, então vamos usar a aplicação RecebeImagem, no projeto RecebeImagem.zip