Computação II – Orientação a Objetos

Fabio Mascarenhas - 2014.1

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

Inflando layouts

- Se quisermos um layout mais complexo nas linhas de uma `ListView`, podemos criar objetos layout e adicionar objetos view a ele

- Ou podemos declarar um layout em XML e *inflar* esse layout

- Para isso usamos o *serviço* `LAYOUT_INFLATER_SERVICE`, uma instância de LayoutInflater:

```
LayoutInflater inf =
    (LayoutInflater)lv.getContext().getSystemService(
        Context.LAYOUT_INFLATER_SERVICE);
view = inf.inflate(R.layout.item_lv, null);
```

AsyncTask<E,I,S>

- Uma tarefa é parametrizada por três tipos:
  - `E` é o tipo dos parâmetros de entrada para a tarefa
  - `I` é o tipo dos resultados intermediários, caso ela reporte progresso
  - `S` é o tipo do resultado da tarefa
- O método `S doInBackGround(E... params)` executa a tarefa
- O método `void onPostExecute(S res)` atualiza a interface com o resultado

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

Canvas

- Para poder desenhar em uma `View`, redefinimos seu método `onDraw`
- Esse método recebe uma instância de `Canvas`
- Para desenhar também de uma instância de `Paint`, que dá a cor e o estilo de desenho (no nosso caso, queremos que as figuras sejam preenchidas)

```
pen = new Paint();
pen.setARGB(255,0,0,0);
pen.setStyle(Style.FILL);
```

```
@Override
public void onDraw(Canvas c) {
  super.onDraw(c);
  c.drawRGB(255,255,255);
  c.drawRect(10, 10, 100, 100, pen);
}
```

onTouchEvent

- Para capturar toques na nossa área de desenho, redefinimos o método `onTouchEvent`, que recebe uma instância de `MotionEvent`
- Estamos interessados em três partes do evento: a ação, a coordenada `x` e a coordenada y
- A ação diz se um toque começou (o usuário encostou o dedo na tela), se um toque terminou (o usuário removeu o dedo), ou se ele se moveu (o usuário deslizou o dedo sobre a tela)
- Vamos traduzir isso em eventos aperto, solta e arrasto no nosso controlador

Checked menus

- Para seleção do modo de edição, vamos usar *checked menus* no menu da barra de aplicação

- Esses menus têm uma checkbox ao lado; a checkbox do modo atual aparecerá ticada

```
<item android:id="@+id/modo_mover"
      android:showAsAction="never"
      android:checkable="true"
      android:checked="true"
      android:title="Mover"/>
```

- O estado de cada item virá de objetos que implementam `Toggle`, e ficam conectados ao modelo

Mudando entre Atividades

- Até agora nossas aplicações Android têm apenas uma atividade, o que simplifica a sua estrutura
- Mas nem sempre é possível fazer uma aplicação assim
- Vamos acrescentar cores ao Editor de Figuras, e fazer a escolha da cor ser feita por uma segunda atividade
- As mudanças no modelo são simples, e precisamos também mudar os métodos de desenho em `Tela`

Intent

- Cada atividade é como uma miniaplicação, então a comunicação entre elas é indireta, através de instâncias de Intent

- Usando um `Intent` podemos mandar uma mensagem para outra atividade na mesma aplicação, ou para alguma outra aplicação instalada

- Na forma mais simples, criamos um `Intent` dando a atividade atual, e *classe* atividade que queremos disparar:

```
Intent icor = new Intent(this, EscolheCor.class);
```

Extras

- Podemos acrescentar dados a um `Intent` que a outra atividade vai poder usar, usando o método `putExtra`

- Esses dados podem ser qualquer objeto serializável

- Isso quer dizer que poderíamos passar o modelo inteiro do editor na nossa mensagem, mas isso seria um desperdício, então passamos apenas a cor corrente

```
icor.putExtra("cor", modelo.getCor());
```

`startActivityForResult`

- Uma vez que temos todos os dados no `Intent`, disparamos a outra atividade com o método `startActivityForResult` da atividade atual

- Passamos o `Intent`, e um código numérico que serve para identificar a razão de estarmos chamando outra atividade

- Quando a outra atividade quer voltar para quem chamou, ela usa o método `setResult` para dar um código de retorno (outro número), e um `Intent` com dados que ela queira passar de volta, depois chama o método `finish`

- Esse `Intent` pode ser criado com um construtor vazio

onActivityResult

- Quando a atividade que disparamos termina, o método `onActivityResult` na atividade original é chamado

- Esse método recebe o código da razão, o código de resultado, e o `Intent` retornado pela outra atividade

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch(requestCode) {
    case ESCOLHE_COR:
        if(resultCode == RESULT_OK) {
            modelo.setCor(data.getIntExtra("cor", 0));
        }
        break;
    }
}
```