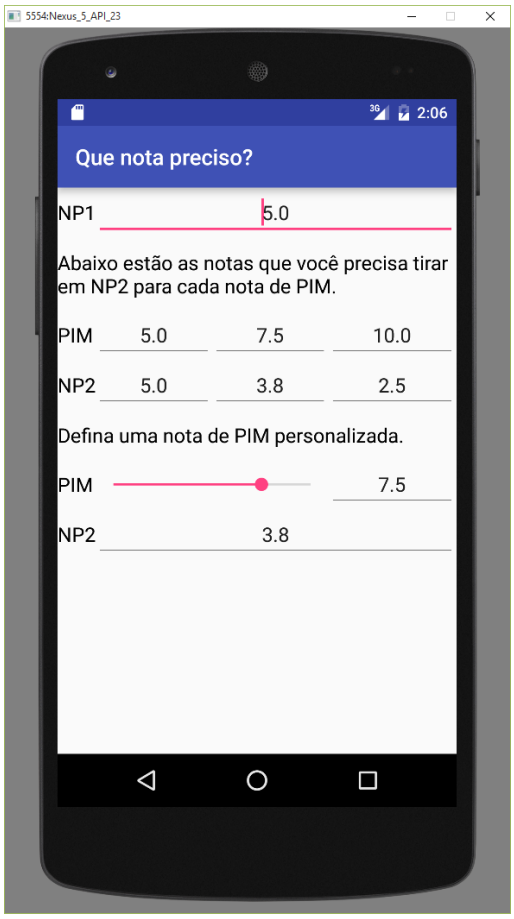
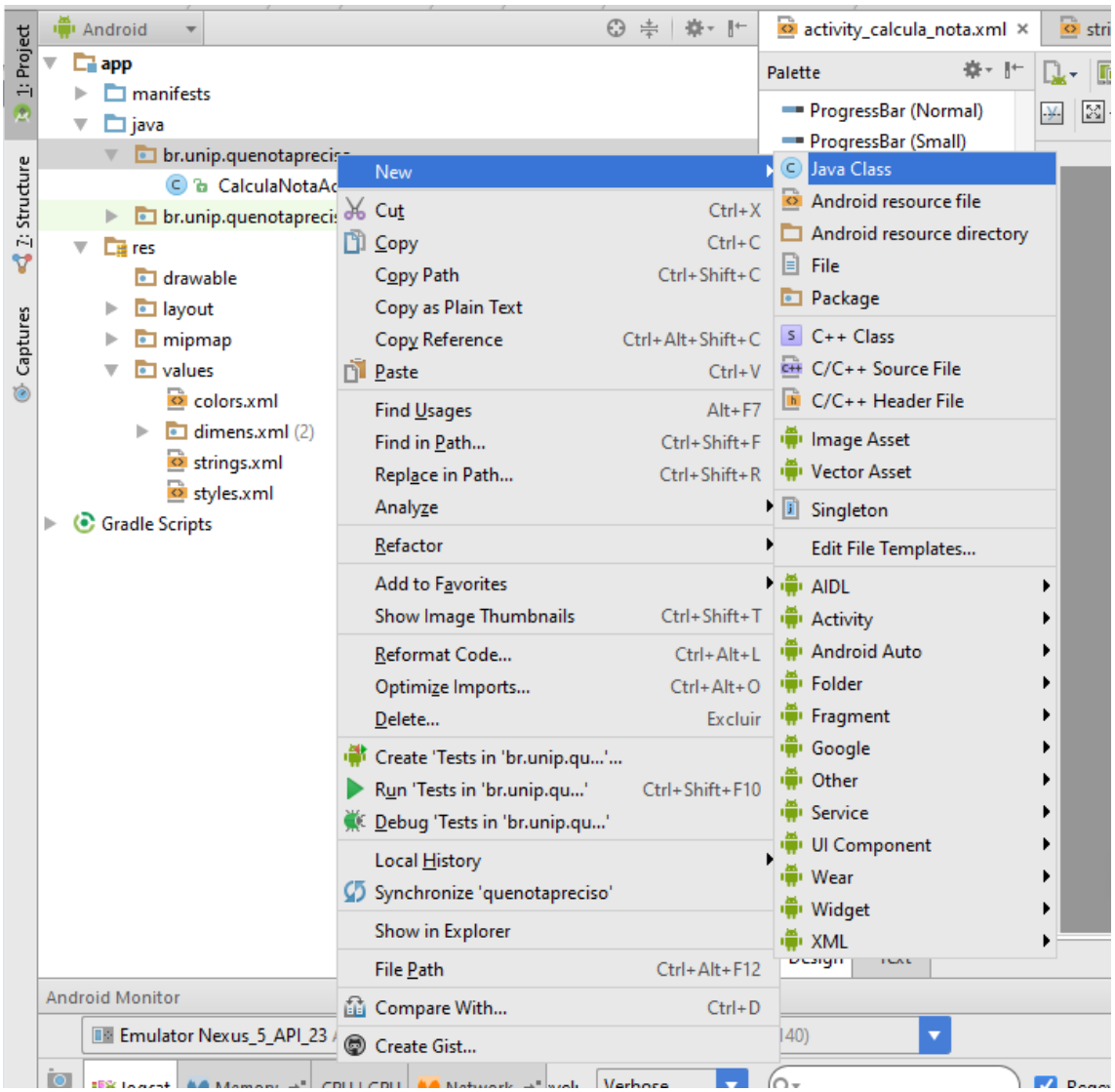
Roteiro de Laboratório 04

José Cassiano Grassi Gunji



No último roteiro desenvolvemos o leiaute do aplicativo de cálculo da nota necessária para a aprovação em uma determinada disciplina. Vamos agora definir seu comportamento. Para tanto, vamos inicialmente criar uma classe Java convencional para implementar a fórmula do cálculo da nota NP2. Clique com o botão direito do mouse no pacote br.unip.quenotapreciso e selecione New -> Java Class. Esta será a classe que implementa as regras de negócio.


Android
app
manifests
java
br.unip.quenotapreci
CalculaNotaAc
br.unip.quenotapreci
res
drawable
layout
mipmap
values
colors.xml
dimens.xml (2)
strings.xml
styles.xml
Gradle Scripts
1: Project
7: Structure
Captures
activity_calcula_nota.xml
Palette
ProgressBar (Normal)
ProgressBar (Small)
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy as Plain Text
Copy Reference Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Find in Path... Ctrl+Shift+F
Replace in Path... Ctrl+Shift+R
Analyze
Refactor
Add to Favorites
Show Image Thumbnails Ctrl+Shift+T
Reformat Code... Ctrl+Alt+L
Optimize Imports... Ctrl+Alt+O
Delete... Excluir
Create 'Tests in 'br.unip.qu...'...
Run 'Tests in 'br.unip.qu...' Ctrl+Shift+F10
Debug 'Tests in 'br.unip.qu...'
Local History
Synchronize 'quenotapreciso'
Show in Explorer
File Path Ctrl+Alt+F12
Compare With... Ctrl+D
Create Gist...
Java Class
Android resource file
Android resource directory
File
Package
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
Singleton
Edit File Templates...
AIDL
Activity
Android Auto
Folder
Fragment
Google
Other
Service
UI Component
Wear
Widget
XML
Android Monitor
Emulator Nexus_5_API_23
Verbose

Crie a classe Calculadora com o seguinte código:

```
1  package br.unip.quenotapreciso;
2
3  /**
4   * Created by Cassiano on 23/10/2016.
5   */
6
7  public class Calculadora {
8
9      static String calculaNp2(double np1, double pim) {
10
11         String saida = "";
12         double np2 = (5.0 - np1 * 0.4 - pim * 0.2) / 0.4;
13
14         if (np2 > 10.0) {
15             saida = ">10;0";
16         } else if (np2 < 0.0) {
17             saida = "<0";
18         } else {
19             saida = String.format("%.1f", np2);
20         }
21
22         return saida;
23     }
24 }
```

Esta é uma classe que assume o estereótipo Control, pois realiza a implementação da lógica do sistema.

A partir de agora, vamos editar o arquivo CalculaNotaActivity.java.

```
1  package br.unip.quenotapreciso;
2
3
4  import android.os.PersistableBundle;
5  import android.support.v7.app.AppCompatActivity;
6  import android.os.Bundle;
7  import android.text.Editable;
8  import android.text.TextWatcher;
9  import android.widget.EditText;
10 import android.widget.SeekBar;
```

As primeiras linhas do arquivo definem o pacote e as importações. Você pode definir os imports manualmente ou permitir que a correção automática crie estas linhas para você conforme for escrevendo o restante do código.

```
11
12  public class CalculaNotaActivity extends AppCompatActivity {
13
14      // Constantes usadas ao se salvar/restaurar estado:
15      private static final String NP1 = "NP1";
16      private static final String PIM = "PIM";
17
18      // Atributos que armazenam os valores que devem ser mantidos quando
19      // o aplicativo reinicia.
20      private double np1;
21      private double pim;
22
23      // Armazena as referências dos componentes da interface gráfica.
24      private EditText np1EditText;
25      private EditText np250EditText;
26      private EditText np275EditText;
27      private EditText np2100EditText;
28      private SeekBar pimSeekBar;
29      private EditText pimEditText;
30      private EditText np2EditText;
31
```

No início da classe são declarados seus atributos. Nem todos os atributos estão declarados aqui. Deixamos dois atributos mais para o final do código pois eles requerem mais explicações.

```
32      // Método chamado quando a Activity é criada ou reativada.
33      @Override
34      protected void onCreate(Bundle savedInstanceState) {
35          super.onCreate(savedInstanceState);
36          setContentView(R.layout.activity_calcula_nota);
37
38          // Obtém referências aos compontentes da tela:
39          np1EditText = (EditText) findViewById(R.id.np1EditText);
40          np250EditText = (EditText) findViewById(R.id.np250EditText);
41          np275EditText = (EditText) findViewById(R.id.np275EditText);
42          np2100EditText = (EditText) findViewById(R.id.np2100EditText);
43          pimSeekBar = (SeekBar) findViewById(R.id.pimSeekBar);
44          pimEditText = (EditText) findViewById(R.id.pimEditText);
45          np2EditText = (EditText) findViewById(R.id.np2EditText);
46
```

Na linha 34 delcaramos a sobrescrita do método onCreate(), que é chamado sempre que a Activity é carregada ou recarregada. Na linha 35 chamamos o método onCreate() da superclasse. A linha 36 especifica que o arquivo de layout desta Activity é o activity_calcula_nota. Este layout é obtido da classe R, que é uma classe do aplicativo que é escrita para nós pelo compilador Gradle. Essa classe não deve ser alterada. Nas linhas 39 a 45 obtemos referências aos componentes interativos usando o método findViewById(). Só podemos obter referências de componentes que tenham sua propriedade id definida, como fizemos anteriormente.

```
47          // Cria os ouvintes de eventos para as views interativas:
48          np1EditText.addTextChangedListener(ouvinteNp1EditText);
49          pimSeekBar.setOnSeekBarChangeListener(ouvintePimSeekBar);
50
```

Nestas linhas, definimos quais são os objetos que irão tratar os eventos de mudança de texto do np1EditText e de mudança do progresso do pimSeekBar. Note que enquanto você digita estas linhas, os atributos ouvinteNp1EditText e ouvintePimSeekBar ainda não foram declarados e serão destacados em vermelho como um erro. Não se preocupe. Mais abaixo estes atributos serão declarados e este erro desaparecerá.

```
51          // Verifica se o aplicativo acabou de ser iniciado ou se está
52          // sendo restaurado:
53          if (savedInstanceState == null) {
54              np1 = 5.0;
55              pim = 7.5;
56          } else {
57              // o aplicativo está sendo restaurado da memória, não está
58              // sendo executado a partir do zero. Assim, os valores de
59              // np1 e pim são restaurados:
60              np1 = savedInstanceState.getDouble(NP1);
61              pim = savedInstanceState.getDouble(PIM);
62          }
63
64          // Atualiza os componentes gráficos com os valores atualizados:
65          np1EditText.setText(String.format("%.1f", np1));
66          pimSeekBar.setProgress((int) (pim * 10));
67      }
```

Se o aplicativo está sendo reiniciado, por exemplo, se a orientação do dispositivo mudou entre retrato e paisagem, é necessário recarregar os valores de np1 e pim. A linha 53 testa se o aplicativo não está sendo reiniciado. Se é a primeira vez que é executado, as linhas 54 e 55 definem valores padrão para np1 e pim. Caso contrário, as linhas 60 e 61 recuperam os valores de np1 e pim que foram gravados pelo aplicativo anteriormente. O método que grava estes valores será visto mais adiante.

```
68
69          // Atualiza o valor das notas NP2 para PIM 5.0, 7.5 e 10.0
70          private void atualizaNp2Padrao() {
71              np250EditText.setText(Calculadora.calculaNp2(np1, 5.0));
72              np275EditText.setText(Calculadora.calculaNp2(np1, 7.5));
73              np2100EditText.setText(Calculadora.calculaNp2(np1, 10.0));
74          }
75
76          // Atualiza o valor da nota NP2 para PIM personalizado:
77          private void atualizaNp2Personalizado() {
78              np2EditText.setText(Calculadora.calculaNp2(np1, pim));
79          }
80
```

Estes métodos são responsáveis por calcular os valores da nota np2 para os diversos casos, tanto para os valores padrão de pim (método atualizaNp2Padrao()) quanto para o valor personalizado de pim (atualizaNp2Personalizado()).

```
80
81      //Define o objeto ouviente de mudança de texto do np1EditText:
82      private TextWatcher ouvinteNp1EditText = new TextWatcher() {
83          @Override
84          public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {
85              // Este método deve ser sobrescrito, mas não é usado neste aplicativo.
86          }
87
88          @Override
89          public void onTextChanged(CharSequence charSequence, int i, int i1, int i2) {
90              try {
91                  np1 = Double.parseDouble(np1EditText.getText().toString());
92              } catch (NumberFormatException e) {
93                  np1 = 0.0;
94              }
95              atualizaNp2Padrao();
96              atualizaNp2Personalizado();
97          }
98
99          @Override
100         public void afterTextChanged(Editable editable) {
101             // Este método deve ser sobrescrito, mas não é usado neste aplicativo.
102         }
103     };
104
```

Esta é a declaração do atributo ouvinteNp1EditText que foi atribuído anteriormente, na linha 48, como ouvinte de mudança de texto do np1EditText. Este atributo recebe uma instância de uma classe anônima, ou seja, uma instância de uma classe que não recebeu um nome. Esta classe implementa a interface TextWatcher que define os métodos que estão sobrescritos nas linhas 84, 89 e 100. Todos eles devem ser sobrescritos, mas o único que é utilizado pelo nosso aplicativo é o método onTextChanged(). Ele obtém o valor da NP1 digitada pelo usuário, converte para double, armazena do atributo np1 e chama os métodos que atualizam o cálculo da NP2.

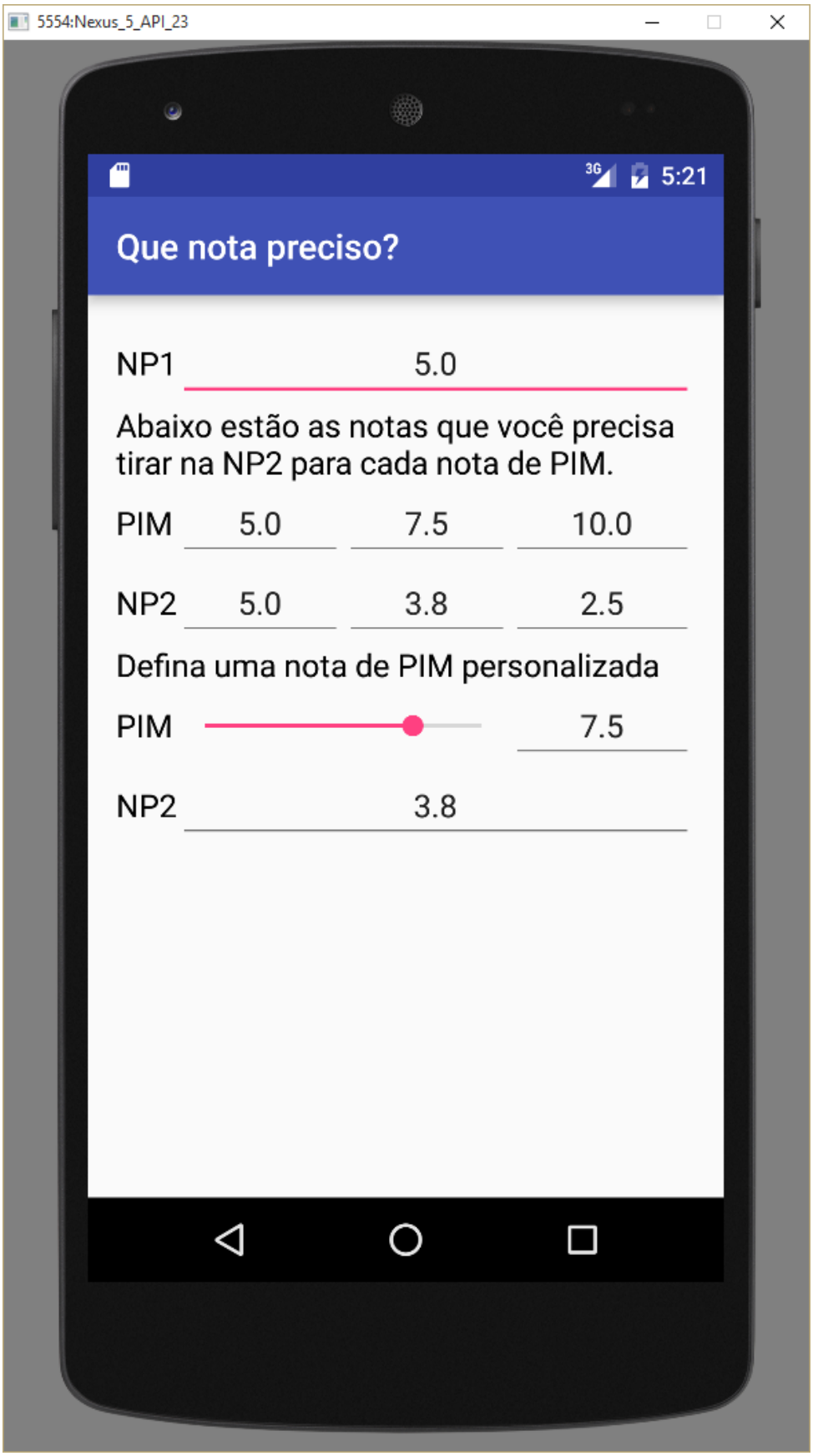
```
104
105     // Define o objeto ouvinte de mudança no pimSeekBar
106     private SeekBar.OnSeekBarChangeListener ouvintePimSeekBar =
107             new SeekBar.OnSeekBarChangeListener() {
108                 @Override
109                 public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
110                     pim = (double) pimSeekBar.getProgress() / 10;
111                     pimEditText.setText(String.format("%.1f", pim));
112                     atualizaNp2Personalizado();
113                 }
114
115                 @Override
116                 public void onStartTrackingTouch(SeekBar seekBar) {
117                     // Este método deve ser sobrescrito, mas não é usado neste aplicativo.
118                 }
119
120                 @Override
121                 public void onStopTrackingTouch(SeekBar seekBar) {
122                     // Este método deve ser sobrescrito, mas não é usado neste aplicativo.
123                 }
124             };
125
```

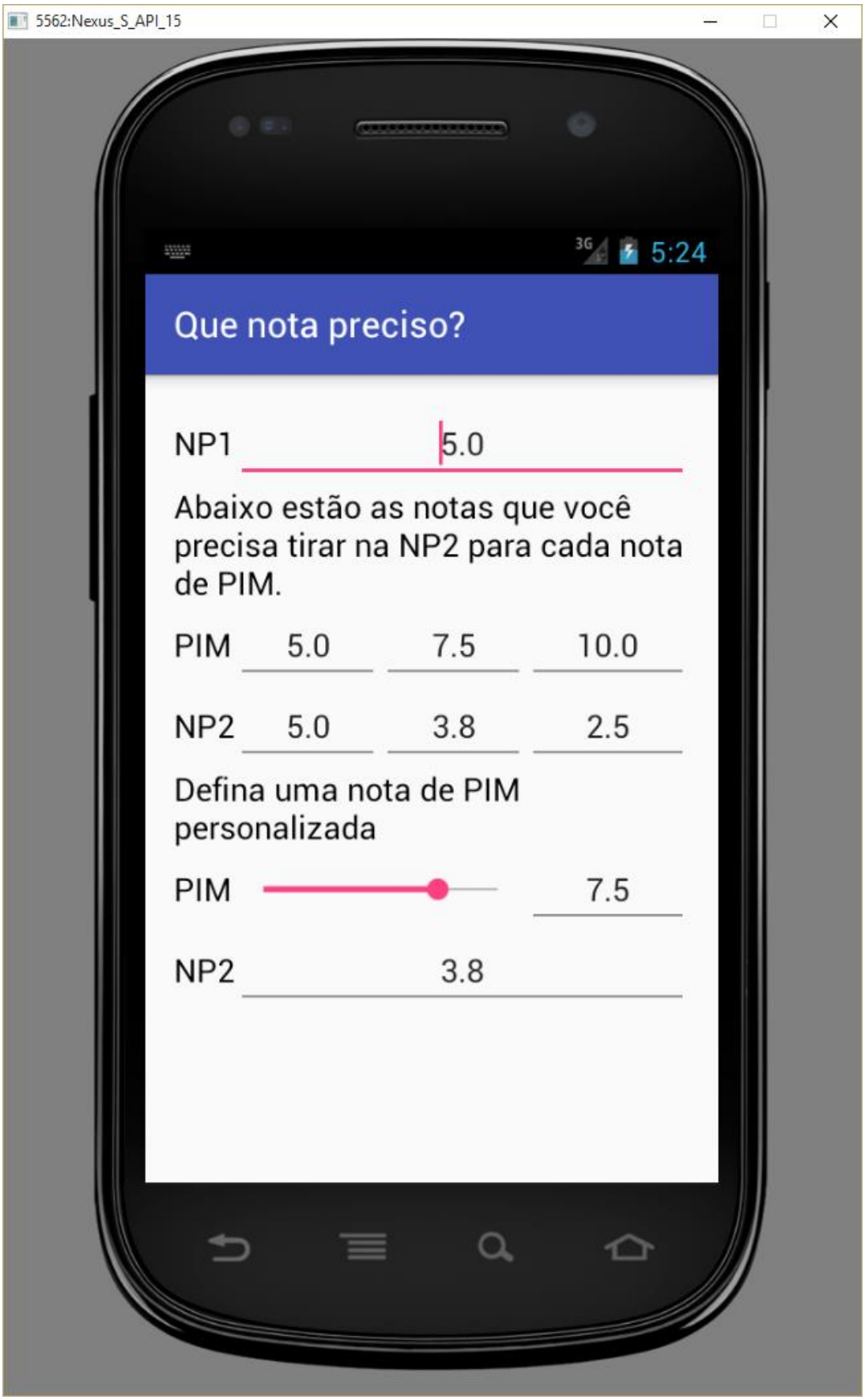
Esta é a declaração do atributo ouvintePimSeekBar, que foi utilizado na linha 49 quando foi atribuído como ouvinte de mudança de progresso do pimSeekBar. Sua lógica de criação é análoga ao do atributo ouvinte anterior.

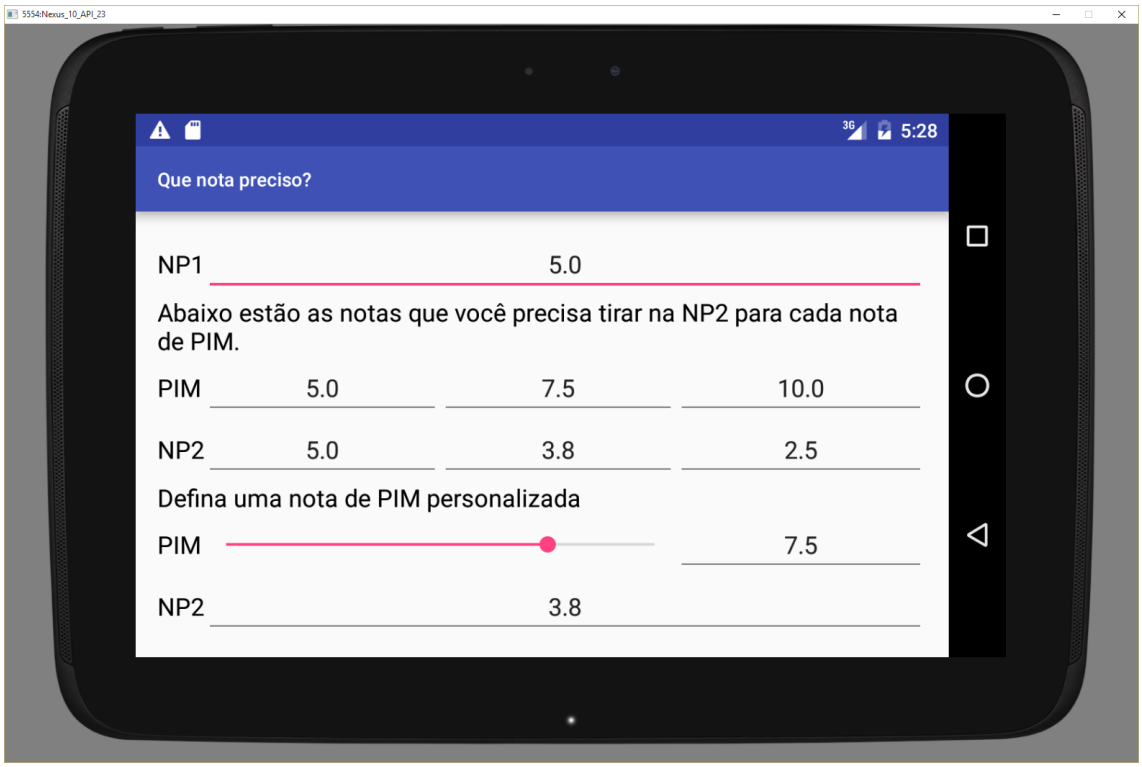
```
125
126         // Este método é chamado quando o aplicativo é interrompido. Nele criamos nossa lógica
127         // para armazenar as informações que devem ser recuperadas quando o aplicativo é reiniciado.
128         @Override
129         public void onSaveInstanceState(Bundle outState, PersistableBundle outPersistentState) {
130             super.onSaveInstanceState(outState, outPersistentState);
131             outState.putDouble(NP1, np1);
132             outState.putDouble(PIM, pim);
133         }
134
135     }
```

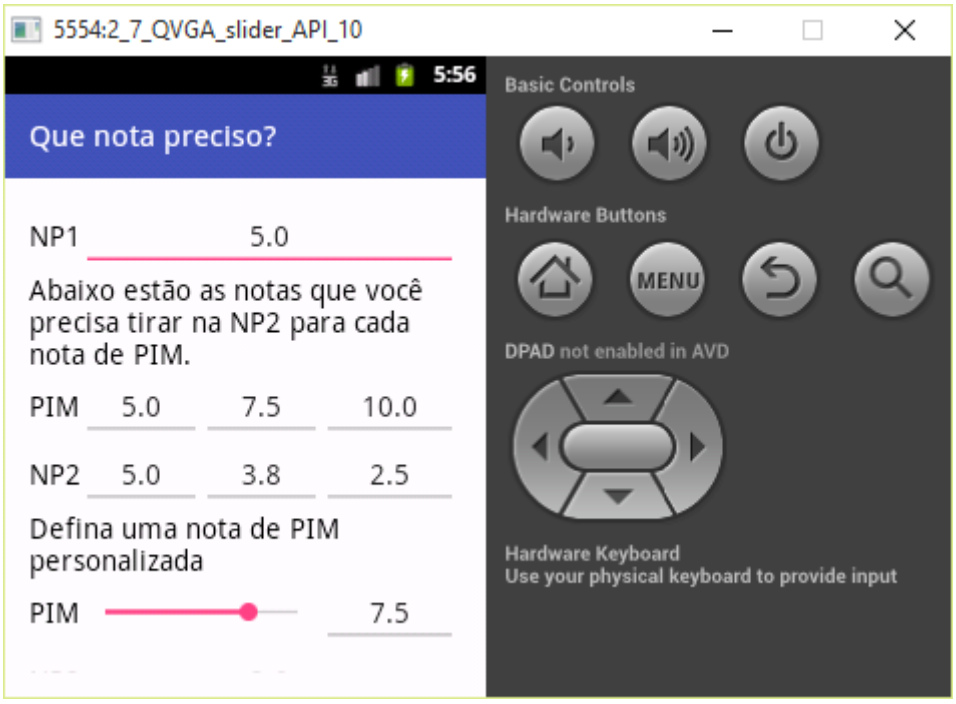
Por fim, o método onSaveInstanceState() é chamado pelo Android quando ele está a ponto de suspender o aplicativo. É neste método que gravamos as informações que devem ser recuperadas quando o aplicativo é recarregado. Note que estas informações só são mantidas enquanto o aplicativo está suspenso. Se ele for totalmente descarregado, estas informações serão perdidas.

Execute seu aplicativo em uma máquina virtual Android ou em um dispositivo real. Note que o aplicativo funciona bem em diversas resoluções de tela.

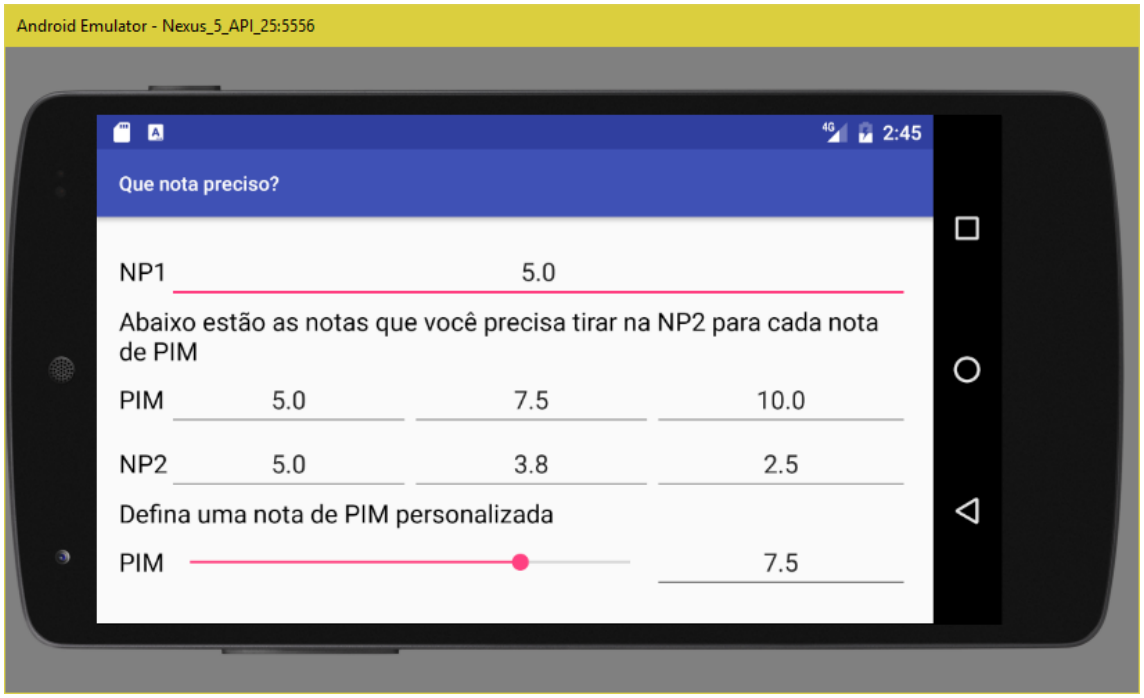

5554:Nexus_5_API_23
5:21
Que nota preciso?
NP1
5.0
Abaixo estão as notas que você precisa tirar na NP2 para cada nota de PIM.
PIM
5.0
7.5
10.0
NP2
5.0
3.8
2.5
Defina uma nota de PIM personalizada
PIM
7.5
NP2
3.8


5562:Nexus_S_API_15
5:24
Que nota preciso?
NP1
5.0
Abaixo estão as notas que você precisa tirar na NP2 para cada nota de PIM.
PIM
5.0
7.5
10.0
NP2
5.0
3.8
2.5
Defina uma nota de PIM personalizada
PIM
7.5
NP2
3.8


5554:Nexus_10_API_23
5:28
Que nota preciso?
NP1
5.0
Abaixo estão as notas que você precisa tirar na NP2 para cada nota de PIM.
PIM
5.0
7.5
10.0
NP2
5.0
3.8
2.5
Defina uma nota de PIM personalizada
PIM
7.5
NP2
3.8


5554:2_7_QVGA_slider_API_10
5:56
Que nota preciso?
NP1
5.0
Abaixo estão as notas que você precisa tirar na NP2 para cada nota de PIM.
PIM
5.0
7.5
10.0
NP2
5.0
3.8
2.5
Defina uma nota de PIM personalizada
PIM
7.5
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input


Android Emulator - Nexus_S_API_16:5554
2:47
Que nota preciso?
PIM 5.0 7.5 10.0
NP2 5.0 3.8 2.5
Defina uma nota de PIM personalizada
PIM 7.5
NP2 3.8


Android Emulator - Nexus_5_API_25:5556
2:45
Que nota preciso?
NP1 5.0
Abaixo estão as notas que você precisa tirar na NP2 para cada nota de PIM
PIM 5.0 7.5 10.0
NP2 5.0 3.8 2.5
Defina uma nota de PIM personalizada
PIM 7.5