Universidade Federal de Ouro Preto
Departamento de Computação e Sistemas - DECSI

Computação Móvel
Conceitos Básicos do Android
Ref. Capítulo 3

Vicente Amorim
vicente.amorim.ufop@gmail.com
www.decom.ufop.br/vicente

Sumário

* Estrutura do Projeto no Android Studio

* Acessando Recursos de Texto e Imagem

* *build.gradle*

* Trabalhando com o *Logcat*

* Tratamento de Eventos

Estrutura do Projeto no Android Studio

Estrutura do Projeto Android Studio

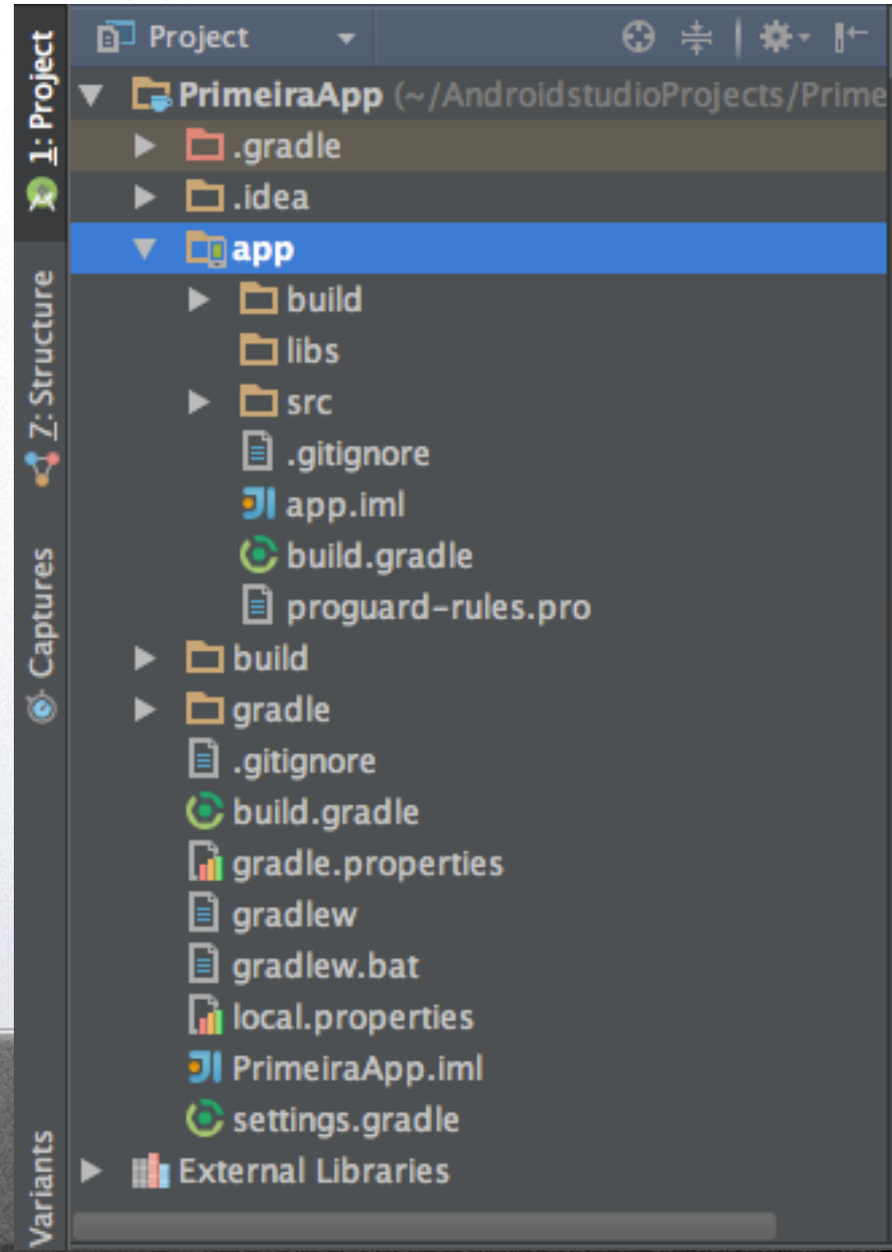
- Detalhes sobre o projeto no Android Studio:

✓ Cada projeto pode conter um ou mais módulos.

✓ A pasta *app* representa o módulo padrão criado juntamente com o projeto.

✓ Dentro da pasta app existe o código-fonte e arquivos de compilação específicos do módulo.

✓ Na raiz do projeto existem arquivos com escopo geral.



Estrutura do Projeto Android Studio

- Descrição dos arquivos da raiz do projeto:

✓ app: Módulo padrão do projeto. Onde a aplicação está contida.

✓ build.gradle: Arquivo de configuração do gradle (*build* da app). Válido para todos os módulos do projeto e gerado automaticamente.

✓ gradle.properties: Arquivo com propriedades para customização do *build* do gradle.

✓ gradle.bat: Arquivo que executa o *build* do gradle para compilação da app.

✓ local.properties: Configurações locais do app.

✓ settings.gradle: Arquivo de configurações do gradle. Indica os módulos a serem compilados.

Estrutura do Projeto Android Studio

- Descrição dos arquivos módulo:

✓ build: Arquivos compilados do projeto. Arquivo de aplicação .apk reside dentro de *build/outputs/apk*.

✓ R.java

. Contém constantes para facilitar o acesso aos recursos do projeto (XMLs de layout, strings, etc).

. Classe é gerada automaticamente pelo ADT a cada vez que um novo recurso é adicionado ao projeto.

. Importante: Nunca alterar manualmente a classe R.java.

. Gerado na pasta *app/build/generated/source/r* do módulo.

Estrutura do Projeto Android Studio

- Descrição dos arquivos módulo: (cont.)

✓ libs: Contém arquivos .jar que devem ser compilados junto ao projeto.

✓ src/main/java: Pasta com os códigos-fonte da aplicação.

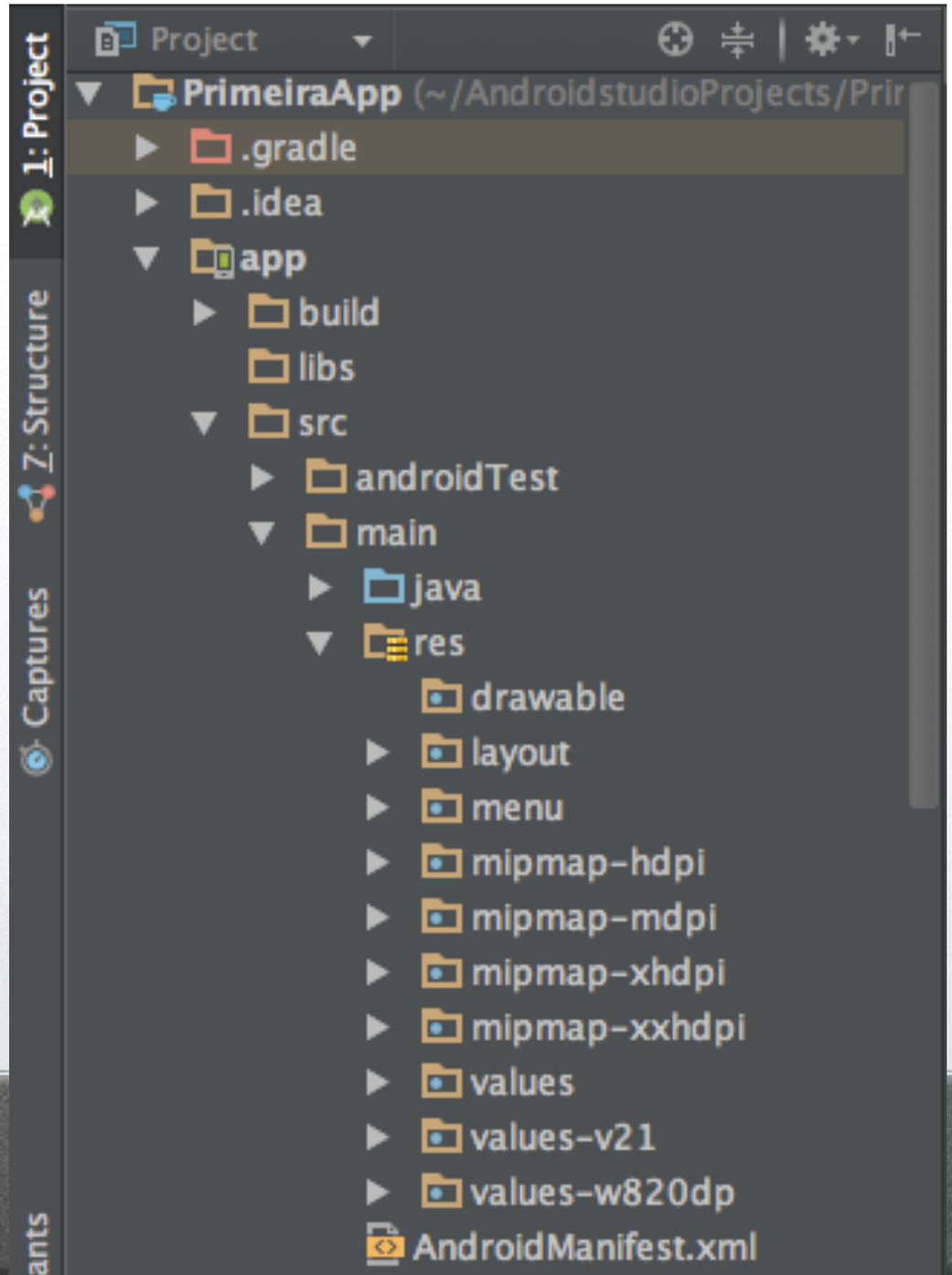
✓ src/main/res: Pasta com os recursos utilizados pela aplicação (imagens, *layouts*, sons, etc). Internamente possui cinco subpastas: *drawable, layout, menu, mipmap* e *values*.

. Cada arquivo (imagem ou XML) dentro dessa pasta contém uma referência na classe R (gerada automaticamente durante a comp.).

. Cada vez que a pasta é alterada, uma nova versão da classe R é gerada.

Estrutura do Projeto Android Studio

- Estrutura do projeto



Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto:

✓ AndroidManifest.xml

. Principal arquivo do projeto.

. Dentre as diversas informações, vemos que a MainActivity foi configurada como a principal do projeto.

. Descreve a versão mínima e *target* da SDK.

. Ícone, versão da aplicação, nome da aplicação, temas, ...

. Contém de forma geral todas as informações necessárias para a execução da aplicação.

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.cea436.primeiraapp" >

    <application
        android:allowBackup="true"              //Permite ou nao backup na cloud
        android:icon="@mipmap/ic_launcher"      //Icone do aplicativo na tela home
        android:label="@string/app_name"        //Nome do aplicativo na tela home
        android:theme="@style/AppTheme" >       //Tema do aplicativo (res/values/styles.xml)
        <activity
            android:name=".MainActivity"        //Classe da Activity que deve ser executada
            android:label="@string/app_name" > //Titulo da Activity para mostrar na action bar
            <intent-filter>                     //Declara um filtro para executar a activity
                <action android:name="android.intent.action.MAIN" /> //Indica que a Activity pode ser
executada como a inicial

                <category android:name="android.intent.category.LAUNCHER" /> //Indica que o icone da activity
deve ficar disponivel na tela inicial
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ AndroidManifest.xml

. Notar uso das referências a itens visuais.

```
...
        android:icon="@mipmap/ic_launcher"
...
```

. A notação com o caractere '@' é utilizada sempre que for necessário acessar um recurso dentro de um XML.

. Se necessário fazer uso de um recurso à partir de uma classe Java, então classe R deve ser utilizada.

Objetivo	XML	Classe R
Acessar icon.png localizada na pasta drawable.	@drawable/icon	R.drawable.icon
Acessar msg hello localizada no arquivo strings.xml	@string/hello	R.string.hello

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ AndroidManifest.xml

. Dentro da *tag* <manifest> é declarado o pacote principal do projeto. No nosso caso: com.cea436.primeiraapp.

. Importante: Nome do pacote deve ser único dentro da Google Play.

. É obrigatório declarar todas as "activities" dentro do arquivo.

. *Tag* `<intent-filter>` é necessária para customizar a forma como a *activity* será iniciada:

- Ação MAIN: Activity pode ser iniciada isoladamente como pt. inicial da app.
- Categoria LAUNCHER: Aplicativo estará disponível na tela inicial.

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ MainActivity.java

. Classe principal do projeto. Representa a tela inicial da aplicação.

. Toda e qualquer *activity* deve extender da classe android.app.Activity.

. android.app.Activity representa uma tela da aplicação. É responsável por controlar o estado e os eventos da tela.

. android.app.Activity apenas representa a tela, não sendo responsável pelo desenho de qualquer elemento.

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ MainActivity.java

```
package com.cea436.primeiraapp;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    } ...
```

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ MainActivity.java

. Método `protected void onCreate(Bundle savedInstanceState)` precisa ser implementado obrigatoriamente.

. `protected void onCreate(Bundle savedInstanceState)` é chamado automaticamente pelo Android quando a tela é criada.

. Elementos gráficos são desenhados na tela através da classe View: TextView, ImageView, etc.

. Criação da tela (e dos elementos gráficos) pode se dar através de arquivos XML ou "programaticamente".

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ MainActivity.java

. Dentro da classe android.app.Activity existem duas assinaturas para o método setContentView().

. 1ª: ID indicando algum recurso de algum arquivo XML.

. 2ª: Argumento do tipo View.

. setContentView() é o método que faz a ligação entre a *Activity* e a *View* (responsável por desenhar a interface gráfica na tela).

. Importante: Verificar o setContentView() presente em onCreate() da MainActivity.java.

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ No Android, o *layout* da tela pode ser criado através de arquivos XML ou diretamente via API.

✓ Recomendado criar sempre utilizando-se XML para separar a lógica de negócios da apresentação.

✓ Pode-se dizer que uma *view* é a representação gráfica de um componente na tela.

✓ Uma *view* pode ser agrupada em várias sub-*views*.

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)
  - ✓ /res/layout/activity_main.xml
    - . Arquivo XML que define a interface gráfica da tela.
    - . Por padrão, o arquivo contém uma tag `<TextView>` que exibe um texto de "Hello world".
    - . Atentar para o valor de `android:text="@string/hello_world"`;
    - . Padrão de *layout* é definido pela tag `<RelativeLayout>`

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ /res/layout/activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">

    <TextView android:text="@string/hello_world" android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

</RelativeLayout>
```

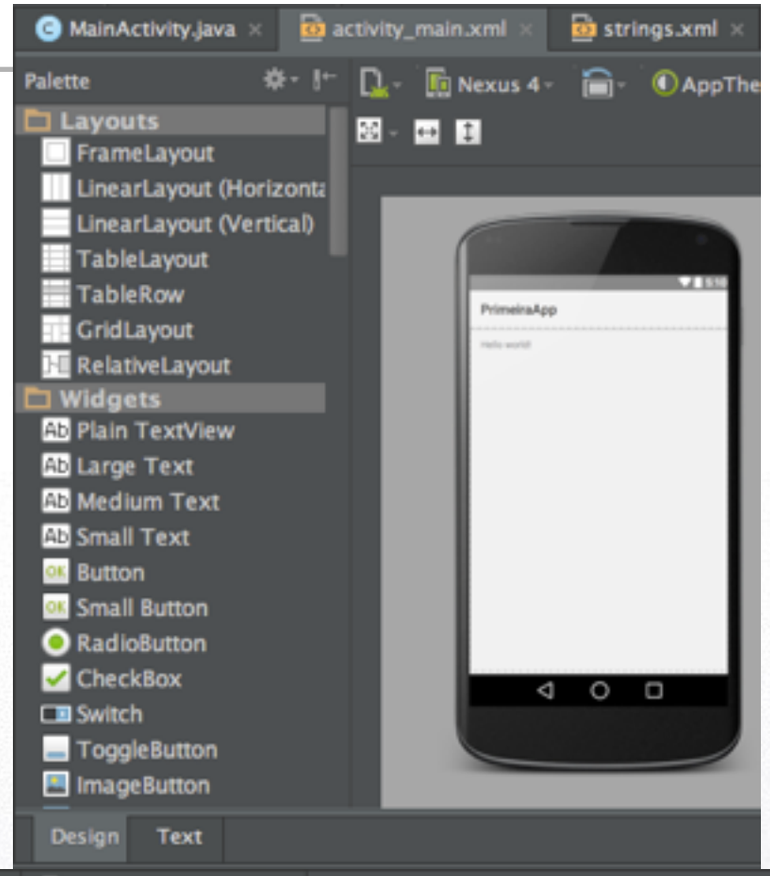
. Sintaxe @dimen é utilizada para acessar valores de espaçamento que foram definidos no arquivo */res/values/dimens.xml*.

Estrutura do Projeto Android Studio

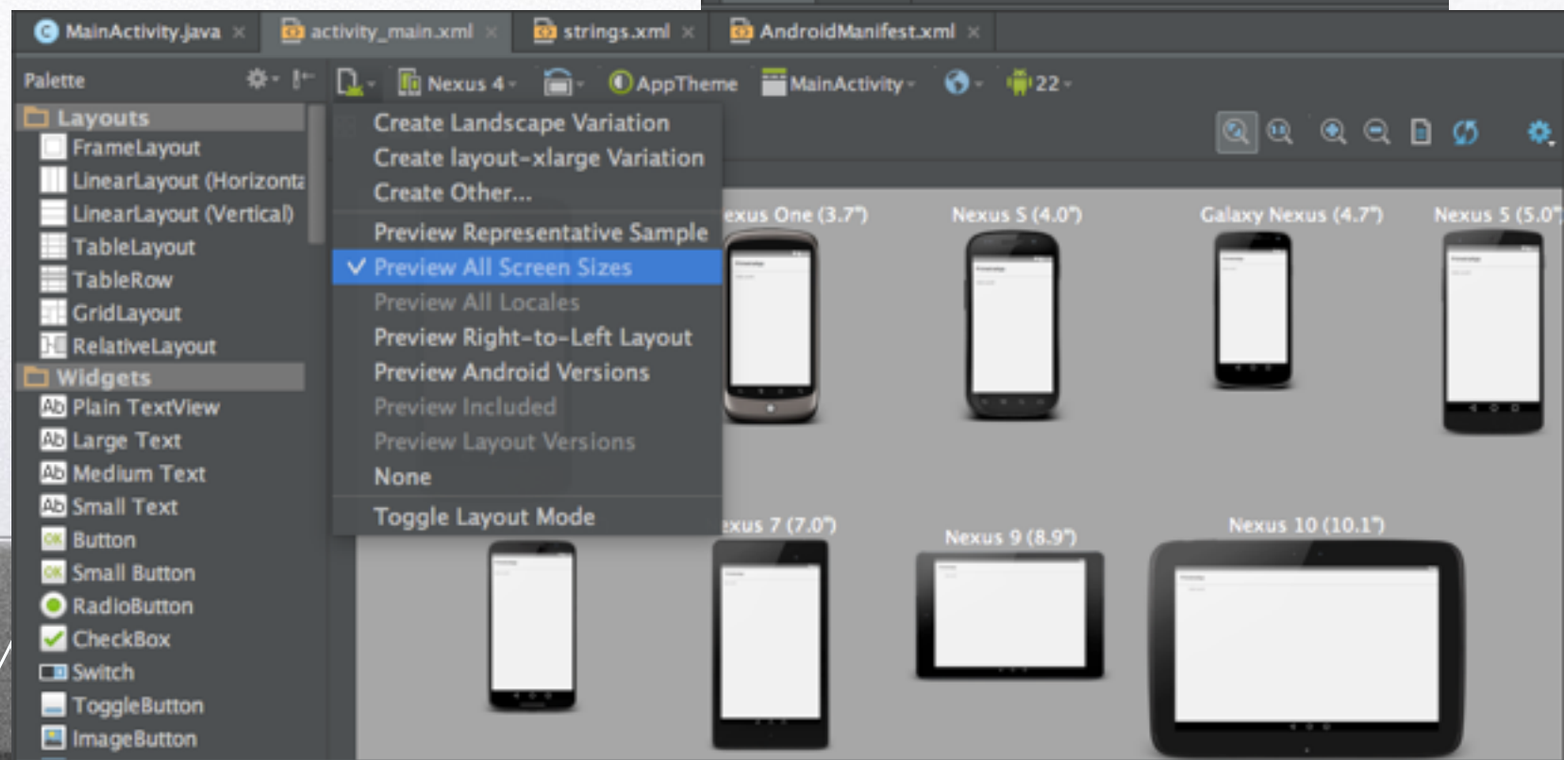
- Arquivos Gerados no Projeto: (cont.)

✓ Visualização

. No Android Studio é possível editar tanto a versão textual de um XML quando verificar sua visualização na tela.



. É possível ainda verificar como um mesmo *layout* se comportaria em diferentes tamanhos de tela.



Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ /res/values/strings.xml

. Arquivo XML que contém as mensagens da aplicação para organizar os textos em um único arquivo centralizado.

. Facilita a internacionalização da aplicação.

. /res/values-en/strings.xml

. /res/values-pt/strings.xml

...

. Inicialmente contém a string do *hello world* e o nome da aplicação (chave app_name).

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)
  - ✓ /res/values/strings.xml

```
<resources>
    <string name="app_name">PrimeiraApp</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
</resources>
```

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

  ✓ app/build/generated/source/r/debug/<packageName>/R.java

  . Contém constantes para facilitar o acesso aos recursos do projeto (XMLs de layout, strings, etc).

  . Classe é gerada automaticamente pelo ADT a cada vez que um novo recurso é adicionado ao projeto.

  . Importante: Nunca alterar manualmente a classe R.java.

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

✓ app/build/generated/source/r/debug/<packageName>/R.java

```
/* AUTO-GENERATED FILE.  ...
package com.cea436.primeiraapp;

public final class R {
    public static final class attr {
    }
    public static final class dimen {
        public static final int activity_horizontal_margin=0x7f050000;
        public static final int activity_vertical_margin=0x7f050001;
    }
    public static final class id {
        public static final int action_settings=0x7f080000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int menu_main=0x7f070000;
    }
    public static final class mipmap {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class string {
        public static final int action_settings=0x7f060000;
        public static final int app_name=0x7f060001;
        public static final int hello_world=0x7f060002;
...
```

Estrutura do Projeto Android Studio

- Arquivos Gerados no Projeto: (cont.)

  ✓ Exercício 1:

  . Trocar o nome da aplicação para “compMovel<seu_nome>”.

  . Traduzir a mensagem “Hello World” para português.

  . Incluir uma nova mensagem “Bem-vindo à minha primeira App!!”.

Acessando Recursos de Texto e Imagem

Acessando Recursos de Texto e Imagem

- Criação da interface visual:

  ✓ Pode ser feita via XML ou código-fonte em Java.

  ✓ Criação da tela em XML deixa o código-fonte mais limpo e inteligível.

  ✓ Para criar uma nova *Activity* à partir de um *layout* XML basta adicionar o arquivo a pasta de *layouts* nos recursos do projeto.

  ✓ Classe R irá atualizar as constantes.

Acessando Recursos de Texto e Imagem

- Criação da interface visual:

  ✓ Criar o arquivo layout_exemplo2.xml na pasta /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="match_parent"
    android:layout_height="match_parent">

    <TextView
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:text="Segunda app Computacao Movel"
    />
</LinearLayout>
```

Acessando Recursos de Texto e Imagem

- AppNum2

  ✓ Uma aplicação Android é uma coleção de recursos relacionados através de diferentes arquivos XML.

  ✓ Como os recursos podem ser relacionados?

. Anteriormente na primeiraApp - *AndroidManifest.xml*

```
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
...
```

. Dentro da pasta *res/mipmap/* deverá existir pelo menos uma figura de nome *ic_launcher.(png/jpg)*.

Acessando Recursos de Texto e Imagem

- AppNum2

✓ E se os recursos precisarem ser utilizados diretamente no layout?

. Cada componente visual deve ser declarado no XML ou programaticamente.

. Ao ser declarado, elemento visual (*view*) deve receber um *id* único.

. Mesma lógica que quando aplicado ao *AndroidManifest.xml*.

- Adiciona-se o recurso na pasta */res* específica (imagens em *drawable*, texto em *values/strings.xml*, etc).

- Cria-se uma referência no *layout* que fará uso do recurso.

Acessando Recursos de Texto e Imagem

- AppNum2
  - ✓ E se os recursos precisarem ser utilizados diretamente no layout?

. Após adicionar a figura *smile1.png* na pasta */res/drawable*:

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

    <TextView
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:text="Segunda app Computacao Movel"
    />

    <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/smile1" />
</LinearLayout>
```

Acessando Recursos de Texto e Imagem

- AppNum2
  - ✓ Exercício 2:
    a) Passar o texto do `android:text` para o arquivo /res/values/strings.xml.

    b) Alterar a função `void onCreate(Bundle)` para fazer referência a esse novo *layout*.

Acessando Recursos de Texto e Imagem

- Interface visual pode ser criada também diretamente via código.

  ✓ Criar um novo projeto: AppNum3.

  ✓ Substituir o código do TextView no *layout* por:

```
package com.cea436.appnum3;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView textView = new TextView(this);
        textView.setText("Disciplina de Comp. Movel");
        setContentView(textView);
    }
...
```

Acessando Recursos de Texto e Imagem

- Interface visual pode ser criada também diretamente via código.

✓ Independentemente se a tela está sendo construída por API ou XML, *setContentView(View)* deve ser chamado informando a view responsável por desenhar a tela.

✓ Para telas mais complexas, o uso do XML é melhor aceito e mais fácil.

. Ex.: Criação de formulários e elementos dispostos em colunas.

✓ Separação entre interface gráfica e lógica de negócios: Lembra modelo **MVC** do iOS.

Acessando Recursos de Texto e Imagem

- Acesso aos elementos da tela

✓ *Layout* da tela construído em XML: Como recuperar no código-fonte os objetos definidos no XML?

✓ Em um formulário, por exemplo, faz-se necessário recuperar de alguma forma os dados preenchidos pelo usuário para posterior processamento.

✓ Dados do formulário devem estar acessíveis via código-fonte.

Acessando Recursos de Texto e Imagem

- Acesso aos elementos da tela
  - ✓ AppNum4
    - ./res/layout/layout_exemplo4.xml

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

    <TextView android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Exemplo utilizando ImageView e recuperacao via codigo-fonte" />

    <ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:src="@drawable/smile1" />

    <ImageView android:id="@+id/smile2"
    android:layout_width="wrap_content" android:layout_height="wrap_content" />
</LinearLayout>
```

Acessando Recursos de Texto e Imagem

- Acesso aos elementos da tela
  - ✓ AppNum4
    - . Primeira imagem definida diretamente no XML:

```
<ImageView android:layout_width="wrap_content" android:layout_height="wrap_content"
android:src="@drawable/smile1" />
```

. Notação "@drawable/<nome_imagem>" é utilizada para referenciar a imagem.

. Origem da segunda imagem não é definida:

- Utilização do conceito para imagens dinâmicas (definidas em tempo de execução de um BD ou internet).

- Conceito de um container para imagens.

- Como fazer para exibir então uma imagem na tela?

Acessando Recursos de Texto e Imagem

- Acesso aos elementos da tela

✓ AppNum4

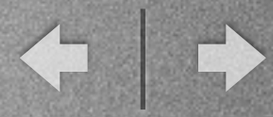
. Aplicação necessita recuperar uma referência aquela imagem via código-fonte.

. Um *id* único foi associado ao *ImageView* correspondente:

```
<ImageView android:id="@+id/smile2"
    android:layout_width="wrap_content" android:layout_height="wrap_content" />
```

. *id* será utilizado como um índice para a recuperação do componente em código.

. A definição de um *id* para os componentes é feita através da notação "@+id/<nome_do_id>". Automaticamente tal id é reconhecido pela classe R.

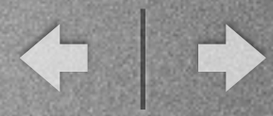



Acessando Recursos de Texto e Imagem

- Acesso aos elementos da tela
  - ✓ AppNum4
    - . Recuperação do componente se dá através do método *findViewById()*:

```
package com.cea436.appnum4;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ImageView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_exemplo4);

        ImageView imageView = (ImageView) findViewById(R.id.smile2);
        imageView.setImageResource(R.drawable.smile2);
    }
...
```

build.gradle

build.gradle

- No Android Studio, o sistema de *build* das aplicações é baseado no *Gradle*.

- O projeto possui internamente o arquivo *app/build.gradle* com as configurações para compilação do módulo *app*.

- Tal arquivo armazena a versão do aplicativo e a versão mínima do Android (*API level*) que o aplicativo suporta.

- Também são listadas no arquivo as bibliotecas necessárias para o funcionamento do aplicativo.


build.gradle

- *build.gradle* do AppNum4:

```
apply plugin: 'com.android.application'

    android {
        compileSdkVersion 22
        buildToolsVersion "21.1.2"

        defaultConfig {
            applicationId "com.cea436.appnum4"
            minSdkVersion 22
            targetSdkVersion 22
            versionCode 1
            versionName "1.0"
        }
        buildTypes {
            release {
                minifyEnabled false
                proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
            }
        }
    }

    dependencies {
        compile fileTree(dir: 'libs', include: ['*.jar'])
    }
```

Trabalhando com o *Logcat*

Trabalhando com o *Logcat*

- Introdução

✓ Em Java, para imprimir uma mensagem na tela faz-se uso do comando System.out.println().

✓ Por que no Android tal comando não funciona?

. Na máquina virtual Dalvik o stdout e stderr redirecionam sua saída para a pasta /dev/null.

✓ Em Android utiliza-se a classe android.util.Log para criar logs de informação, debug, alertas e erros.

✓ Implementação e suporte a diversas categorias de logs.

Trabalhando com o *Logcat*

- Introdução

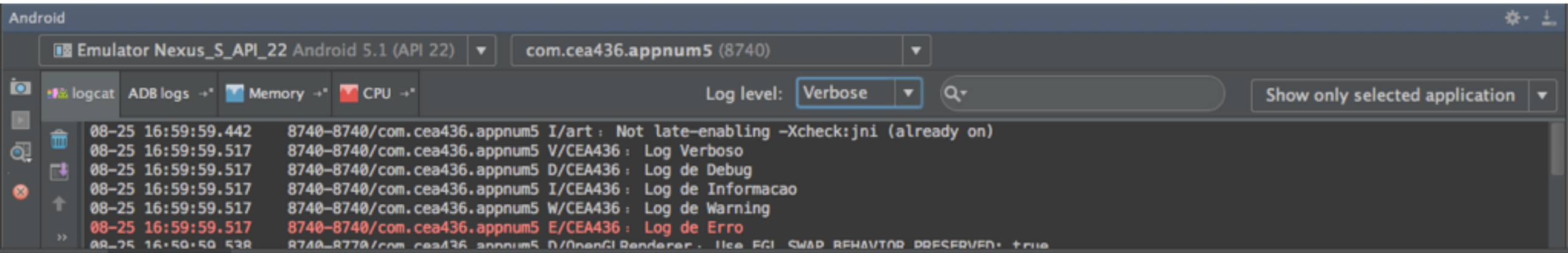
  ✓ Logs gerados com a classe android.util.Log são direcionados para a ferramenta LogCat.

  ✓ AppNum5

```
package com.cea436.appnum5;

import android.app.Activity;
...

public class MainActivity extends Activity {

    private static final String CATEGORIA = "CEA436";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Verbose
        Log.v(CATEGORIA,"Log Verboso");

        //Debug
        Log.d(CATEGORIA,"Log de Debug");

        //Info
        Log.i(CATEGORIA,"Log de Informacao");

        //Warning
        Log.w(CATEGORIA,"Log de Warning");

        //Error
        Log.e(CATEGORIA,"Log de Erro");
    }
...
```

Trabalhando com o *Logcat*

- Introdução
  - ✓ AppNum5



Tratamento de Eventos

Tratamento de Eventos

- Introdução

✓ Aplicações normalmente possuem embutidas o conceito de interatividade.

✓ Devem responder às entradas providas pelo usuário.

✓ Comumente, interações acontecem quando usuários entram em contato com algum componente visual:

. Clique em um botão;

. Acesso a um menu;

. etc.

✓ Como controlar os eventos e prover *feedback* ao usuário?

Tratamento de Eventos

- Introdução

✓ No Android os eventos são controlados por funções callback específicas.

✓ Exemplo: Para botões, o método setOnClickListener() necessita inicialmente ser invocado para o cadastro do *listener* de determinado evento.

. Método recebe como parâmetro uma instância da interface android.view.View.OnClickListener.

. Interface OnClickListener é quem define o método onClick(View) - chamado automaticamente quando da ocorrência de um evento.

Tratamento de Eventos

- Exemplo:
  - ✓ AppNum6
    - ./res/values/strings.xml

```
<resources>
    <string name="app_name">AppNum6</string>
    <string name="action_settings">Settings</string>
    <string name="usuario">Usuário</string>
    <string name="senha">Senha</string>
    <string name="login">Login</string>
</resources>
```

Tratamento de Eventos

- Exemplo:

✓ AppNum6

./res/layout/activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="wrap_content" android:padding="16dp" android:gravity="center_vertical"
android:orientation="vertical">

    <TextView
    android:layout_width="match_parent" android:layout_height="wrap_content" android:text="@string/usuario"/>

    <EditText
    android:id="@+id/tLogin"
    android:layout_width="match_parent" android:layout_height="wrap_content"
    android:inputType="text"
    android:singleLine="true"/>

    <TextView
    android:layout_width="match_parent" android:layout_height="wrap_content" android:layout_marginTop="10dp"
    android:text="@string/senha"/>

    <EditText
    android:id="@+id/tSenha" android:layout_width="match_parent" android:layout_height="wrap_content"
    android:inputType="textPassword" android:singleLine="true"/>

    <Button
    android:id="@+id/btLogin" android:layout_width="200dp" android:layout_height="wrap_content"
    android:layout_marginTop="6dp" android:text="@string/login"
    android:textColor="#ffffff" android:layout_gravity="center"/>
</LinearLayout>
```

Tratamento de Eventos

- Exemplo:

✓ AppNum6

. As *tags* <*Button*>, <*TextView*> e <*EditText*> correspondem às classes Button, TextView e EditText, respectivamente.

. TextView apenas exibe uma informação, enquanto que EditText é um campo para entrada de dados.

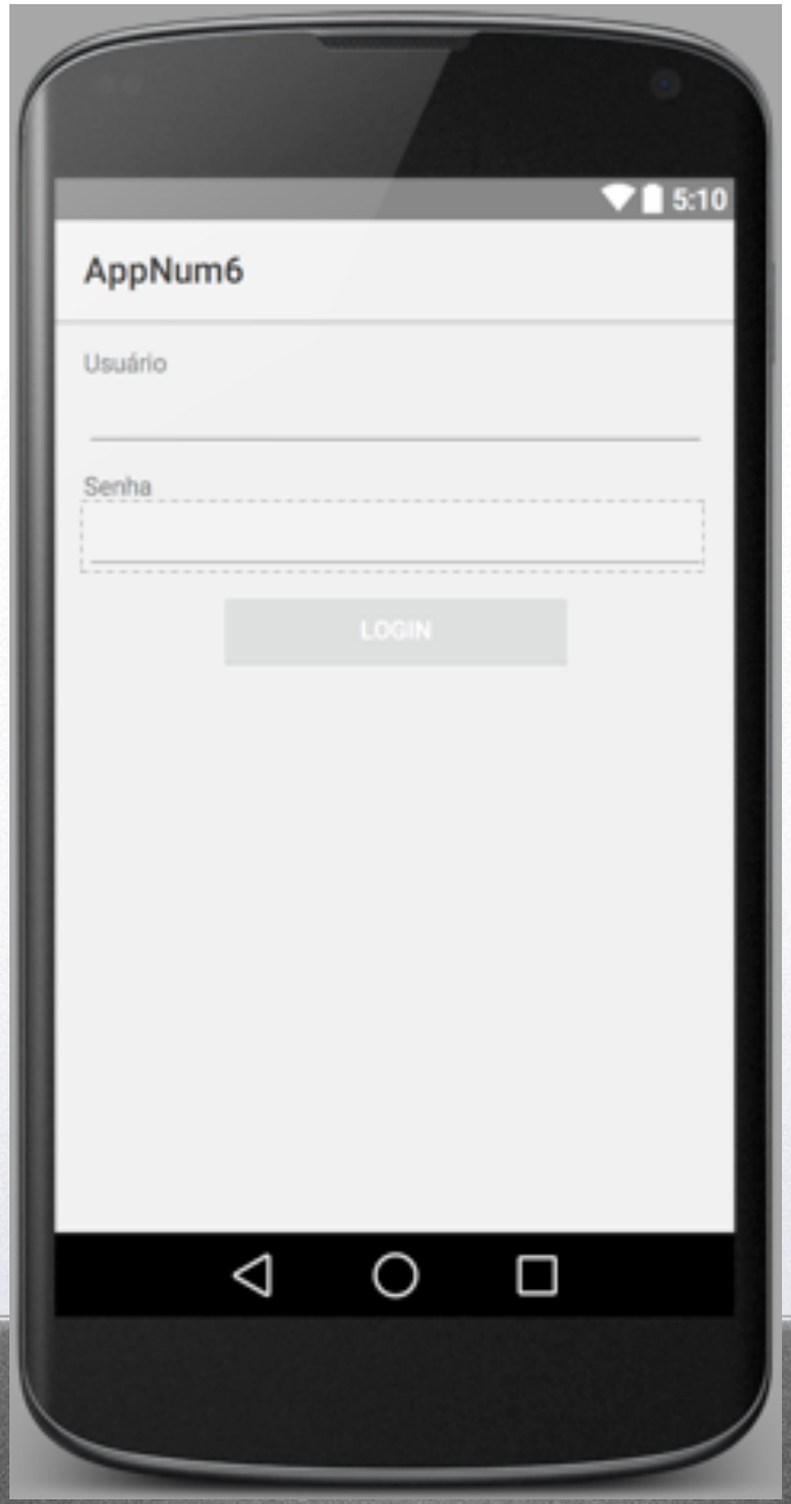
. Notar os *ids* definidos para cada um dos elementos no XML.

. Android Studio permite uma pré-visualização da tela através da aba *design*.

. Futuramente, funcionalidade útil para a edição de telas.

Tratamento de Eventos

- Exemplo:
  - ✓ AppNum6



Tratamento de Eventos

- Exemplo:
  - ✓ AppNum6

```
...
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button btLogin = (Button) findViewById(R.id.btLogin);
        btLogin.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                TextView tLogin = (TextView) findViewById(R.id.tLogin);
                TextView tSenha = (TextView) findViewById(R.id.tSenha);

                String login = tLogin.getText().toString();
                String senha = tSenha.getText().toString();

                if ("vicente".equals(login) && "123456".equals(senha)) {
                    alert("Bem-vindo! Login realizado com sucesso!!!");
                }
                else {
                    alert("Login e/ou senha incorretos!!!");
                }

            }
        });
    }

    private void alert(String s) {
        Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
    }
...
```

Tratamento de Eventos

- Exemplo:

✓ AppNum6

. Implementação atual de tratamento dos eventos utiliza o conceito de classes anônimas do Java.

. Ainda é possível se tratar um evento de outra forma, generalizando-o dentro da própria classe.

. Vantagem: Código mais simples de se compreender.

. Desvantagem: Necessária a implementação de vários tratamentos quando mais um componente estiver na mesma *View*.

. Outra opção é ainda indicar o método de tratamento diretamente no XML de *layout:* <Button android:onClick="onClickBtLogin" …>

Tratamento de Eventos

- Exemplo:
  - ✓ AppNum6

```
...
public class MainActivity_alternate extends Activity implements View.OnClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_activity_alternate);

        Button bt1 = (Button) findViewById(R.id.bt1);
        Button bt2 = (Button) findViewById(R.id.bt2);
        bt1.setOnClickListener(this);
        bt2.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if(v.getId() == R.id.bt1) {
            alert("Clicou no botao 1!!");
        }
        else if (v.getId() == R.id.bt2) {
            alert("Clicou no botao 2");
        }
    }

    private void alert(String s) {
        Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
    }
...
```

Tratamento de Eventos

- Exercícios:

3) Crie um programa em Android que simule uma calculadora. Sua aplicação deve conter dois campos EditText para os operandos, um TextView para o resultado e quatro campos Button para os operadores (+, -, * e /).

✓ Considere o uso das seguintes funções para conversão:

. float a = Float.parseFloat(“”);

. int a = Integer.parseInt(“”);

4) Incremente o programa acima. Se o resultado for maior que 100, um *smile* feliz deverá ser exibido. Caso contrário, um *smile* triste será mostrado.

✓ Baixe do site da disciplina as imagens: http://www.decom.ufop.br/vicente/disciplinas/2015_2/comp_movel/