

**INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**
RIO GRANDE DO NORTE
Campus Natal - Central

Log, Ciclo de Vida e Diálogos

Prof. Fellipe Aleixo (fellipe.Aleixo@ifrn.edu.br)

Conteúdo

- Log
 - Classe `android.util.Log`
 - `LogCat`
- Ciclo de Vida
 - Pilha de atividades
 - Métodos e estados da atividade
- *Instance State*
 - `Bundle`
- Diálogos
 - `Dialogs`
 - `AlertDialog`, `AlertDialog.Builder`

LogCat

- A classe `android.util.Log` pode ser utilizada para escrever logs (informações) de uma aplicação
- Tipos de log e métodos da classe:
 - `Debug (d)` – mensagens de debug
 - `Verbose (v)` – método para mensagens mais extensas
 - `Informação (i)` – utilizado para informações
 - `Alerta (w)` – para mensagens de alerta (*warnings*)
 - `Erro (e)` – para mensagens de erro

LogCat

- Todos os métodos recebem uma categoria para facilitar o filtro das mensagens no LogCat
 - LogCat – Window
 - Show View
 - Other
 - LogCat

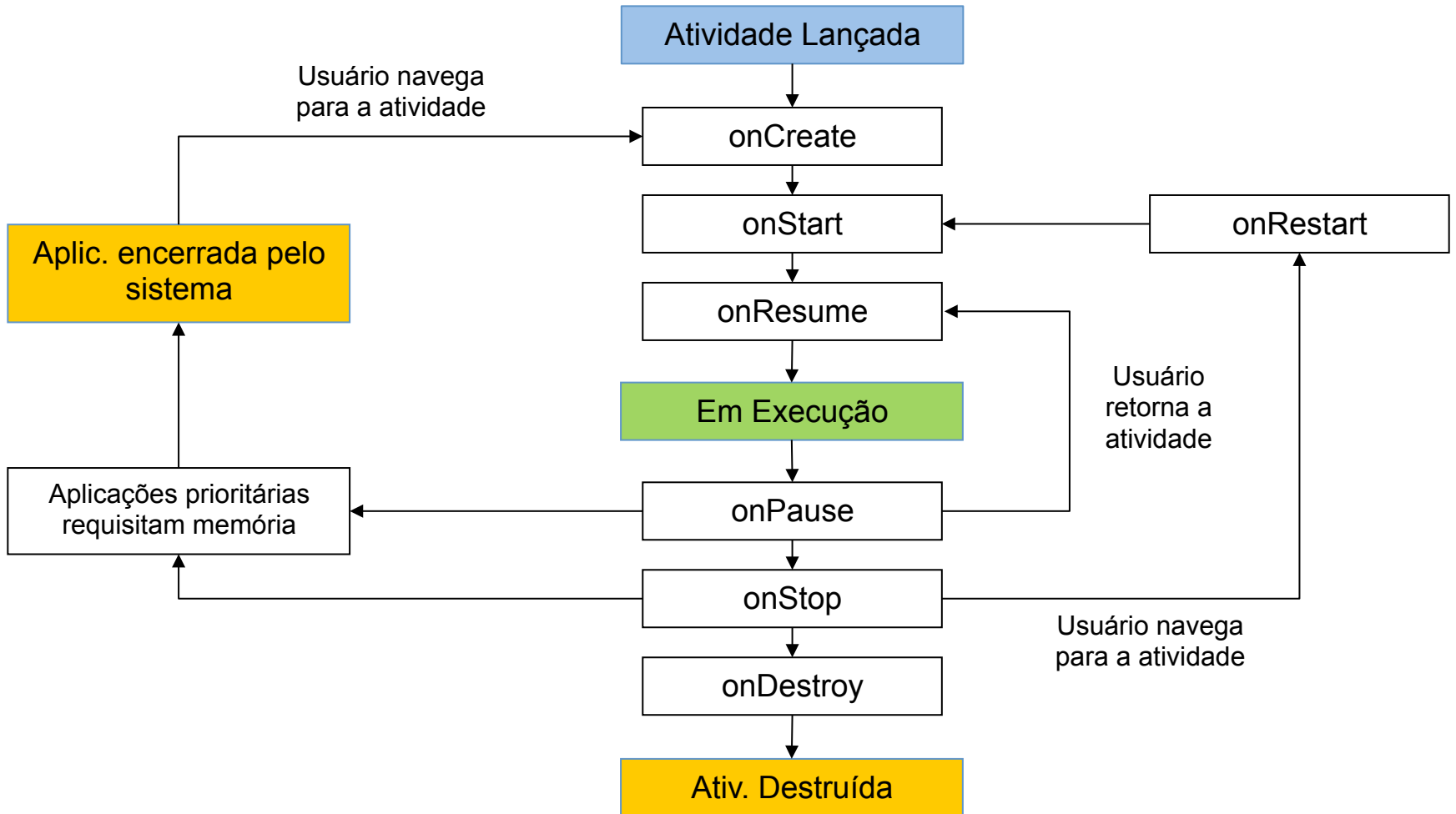
Pilha de Atividades e Ciclo de Vida

- As *activities* são empilhadas pelo sistema em uma *activity stack*
- Quando uma nova atividade é lançada, ela ocupa o topo da pilha – “em execução”
- As demais atividades ficam (i) executando em segundo plano, (ii) pausadas ou (iii) paradas

Pilha de Atividades e Ciclo de Vida

- Ciclo de Vida: compreende os estágios em que a atividade se encontra desde a sua criação até a destruição pelo sistema
- Durante o ciclo de vida, vários métodos da classe **Activity** são chamados e podem ser sobre escritos

Ciclo de Vida



Métodos de Ciclo de Vida

- **onCreate**
 - É chamado uma única vez e deve criar a **View** com **setContentView**
 - O método **onStart** é chamado em seguida
- **onStart**
 - É chamado quando a atividade está ficando visível
 - Ocorre após **onCreate** ou **onRestart**
- **onRestart**
 - É chamado para reiniciar a atividade após uma parada
- **onResume**
 - É chamado quando a atividade está no topo da pilha

Métodos de Ciclo de Vida

- **onPause**
 - É chamado quando algum evento remove a atividade em execução do topo da pilha
- **onStop**
 - É chamado quando uma atividade está sendo encerrada
 - Após o método, a atividade não estará mais visível
- **onDestroy**
 - Encerra em definitivo a execução da atividade
 - Pode ser chamado pelo sistema operacional ou pelo método **finish** da classe **Activity**

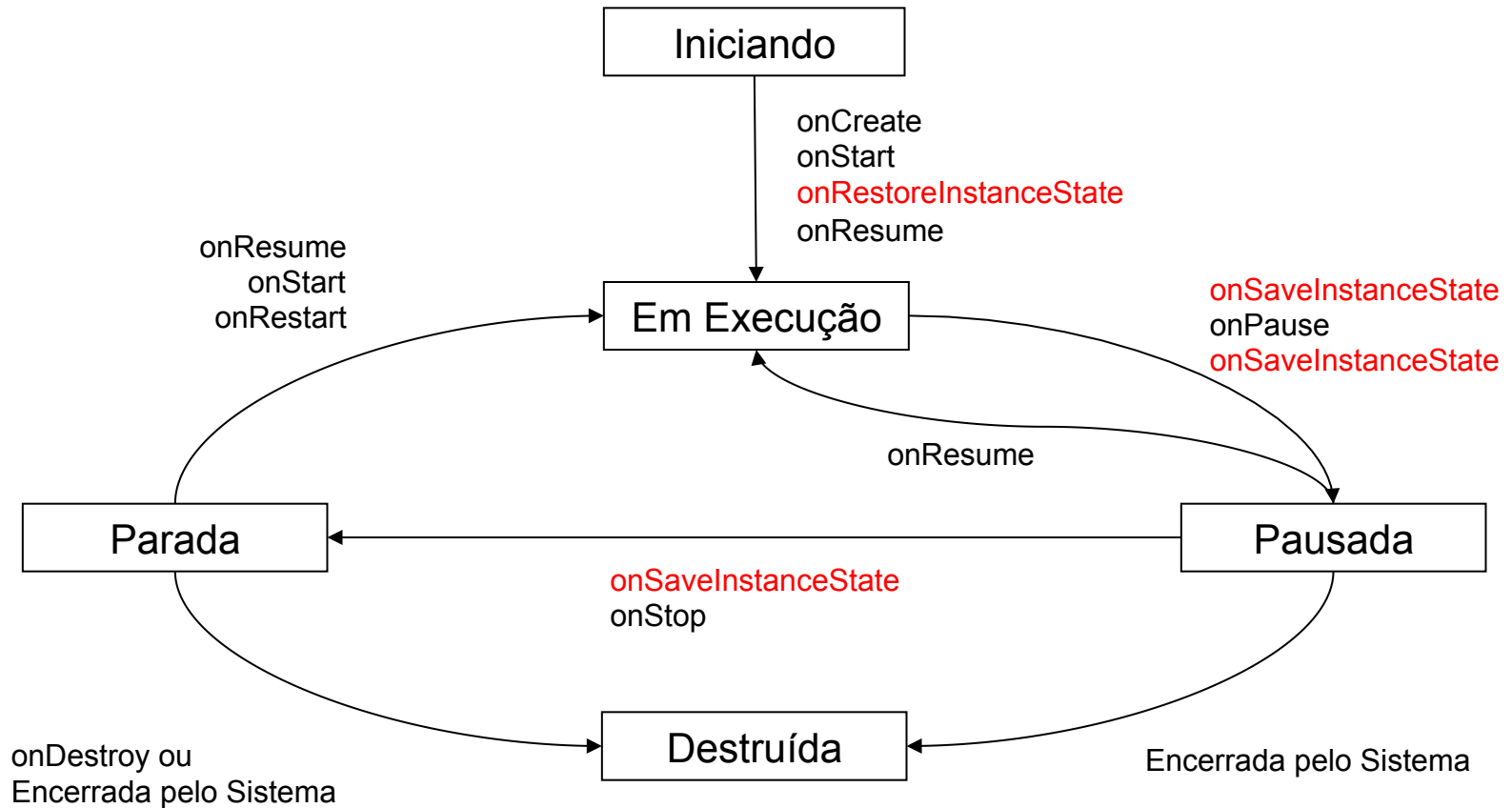
Ciclos de Vida da Atividade

- Ciclo de Vida Completo (**Entire Lifetime**)
 - Do **onCreate** ao **onDestroy**, que são chamados uma única vez
 - Recursos alocados no **onCreate** devem ser liberados no **onDestroy**
- Ciclo de Vida Visível (**Visible Lifetime**)
 - Do **onStart** ao **onStop**
 - Ciclo: **onStart**, **onResume**, **onPause**, **onStop**, **onRestart**, **onStart**
 - A atividade está iniciada, podendo estar visível ou parada em segundo plano

Ciclos de Vida da Atividade

- Ciclo de Vida em 1º Plano (**Foreground Lifetime**)
 - Do **onResume** ao **onPause**
 - A atividade está no topo da pilha em interação com o usuário
 - Este ciclo pode se repetir várias vezes, alterando o estado da aplicação de **Em Execução** para **Pausado**
 - Os métodos devem ser leves, já que podem ser executados várias vezes
 - O método **onPause** pode ser chamado, por exemplo, quando o celular entra no modo dormir para economizar bateria

Estados da Atividade



InstanceState

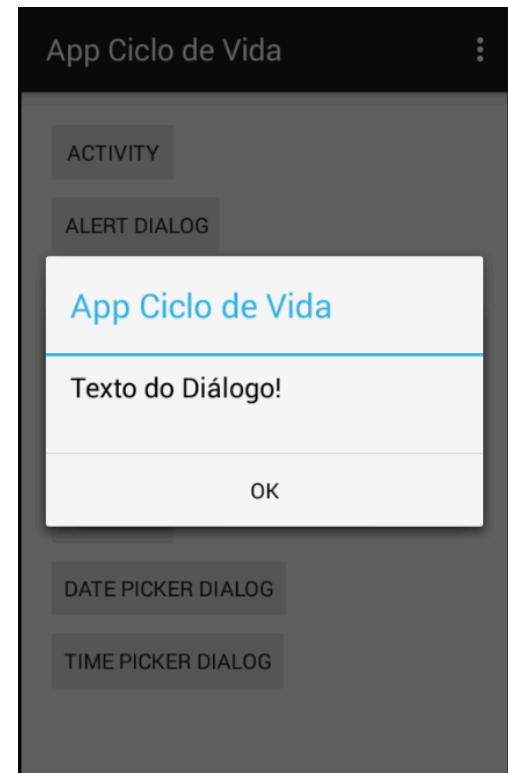
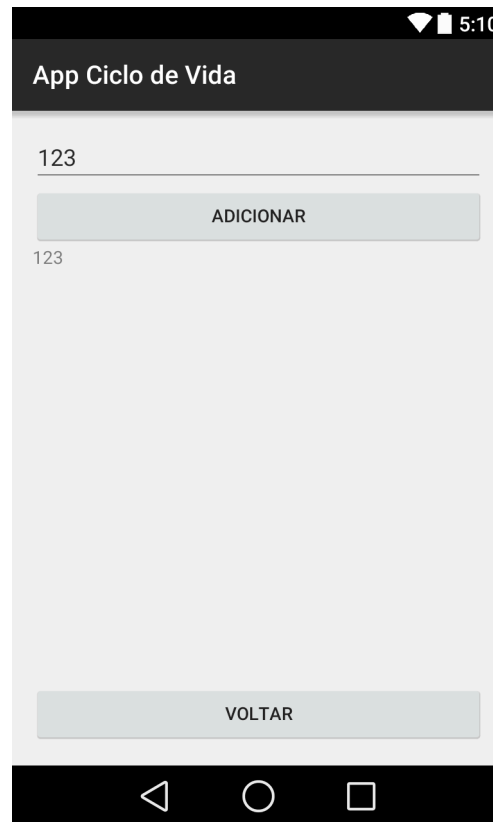
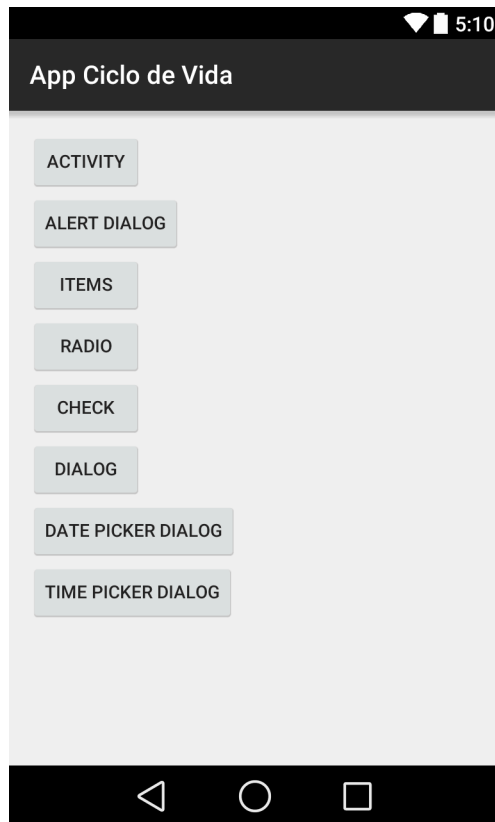
- O estado de uma atividade ([InstanceState](#)) pode ser salvo em um objeto da classe [android.os.Bundle](#) e recuperado após a atividade ser reiniciada
- A classe [Bundle](#) define um mapa onde:
 - A chave de cada item é uma [String](#)
 - O valor é um [Parcelable](#) (tipos primitivos, vetores e listas)
- O método [onSaveInstanceState](#) de [Activity](#) é usado para salvar o estado da atividade antes da sua destruição
- Os métodos [onCreate](#) e [onRestoreInstanceState](#) podem ser usados para recuperar o estado salvo após reiniciar

Diálogos

- Diálogos são janelas modais utilizadas para:
 - Apresentar mensagens para o usuário
 - Listar um conjunto de opções e solicitar uma decisão
 - Requisitar informações adicionais a uma atividade
- Principais Classes
 - `android.app.Dialog`
 - `android.app.AlertDialog`
 - `android.app.AlertDialog.Builder`
 - `android.app.DatePickerDialog`
 - `android.app.TimePickerDialog`

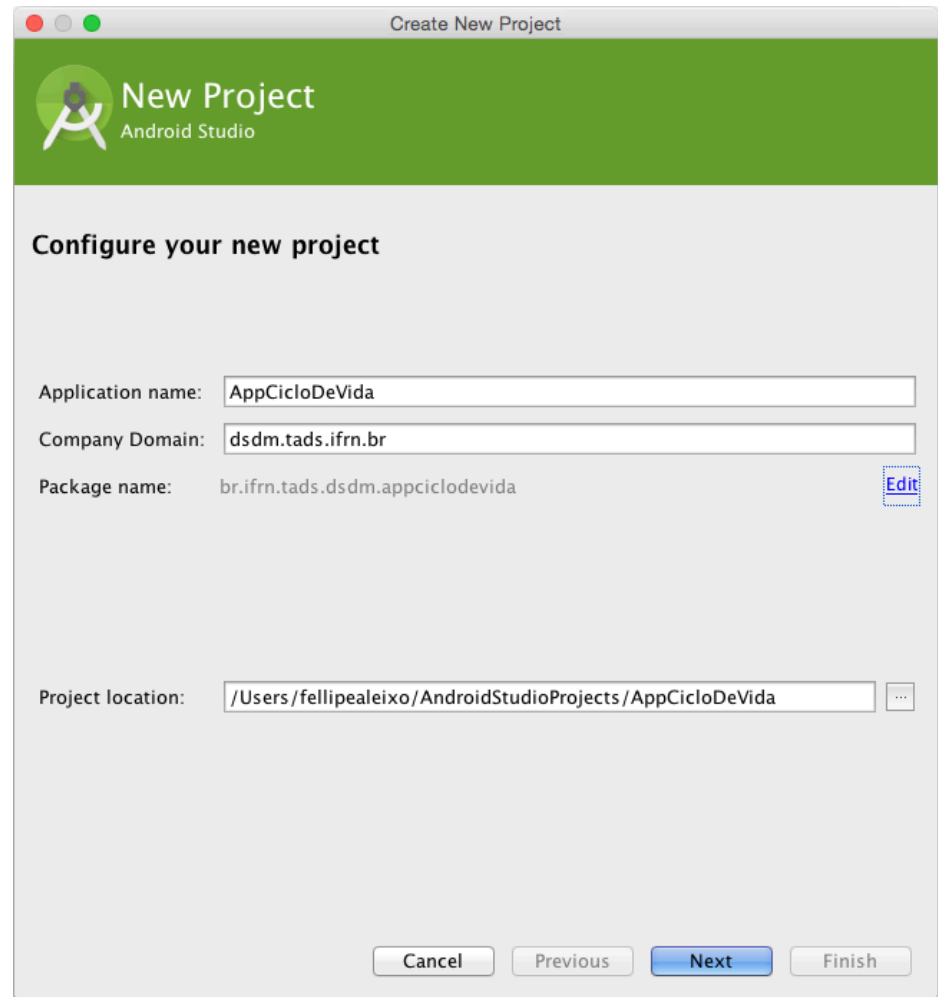
Exemplo – Ciclo de Vida

- A aplicação mostra como utilizar Log, Ciclo de Vida, Instance States e Diálogos




Criação do Projeto

- No Android Studio, siga os passos do exemplo anterior
- A interface padrão e demais arquivos do projeto são criados



Criação do Projeto

Create New Project

 Target Android Devices

Select the form factors your app will run on

Different platforms may require separate SDKs

Phone and Tablet

Minimum SDK: API 18: Android 4.3 (Jelly Bean)

Lower API levels target more devices, but have fewer features available. By targeting API 18 and later, your app will run on approximately 55,0% of the devices that are active on the Google Play Store.
[Help me choose](#)

Wear

Minimum SDK: API 21: Android 5.0 (Lollipop)

TV

Minimum SDK: API 21: Android 5.0 (Lollipop)

Android Auto

Glass (Not Installed) [Download](#)

Minimum SDK:

Cancel Previous Next Finish

strings.xml

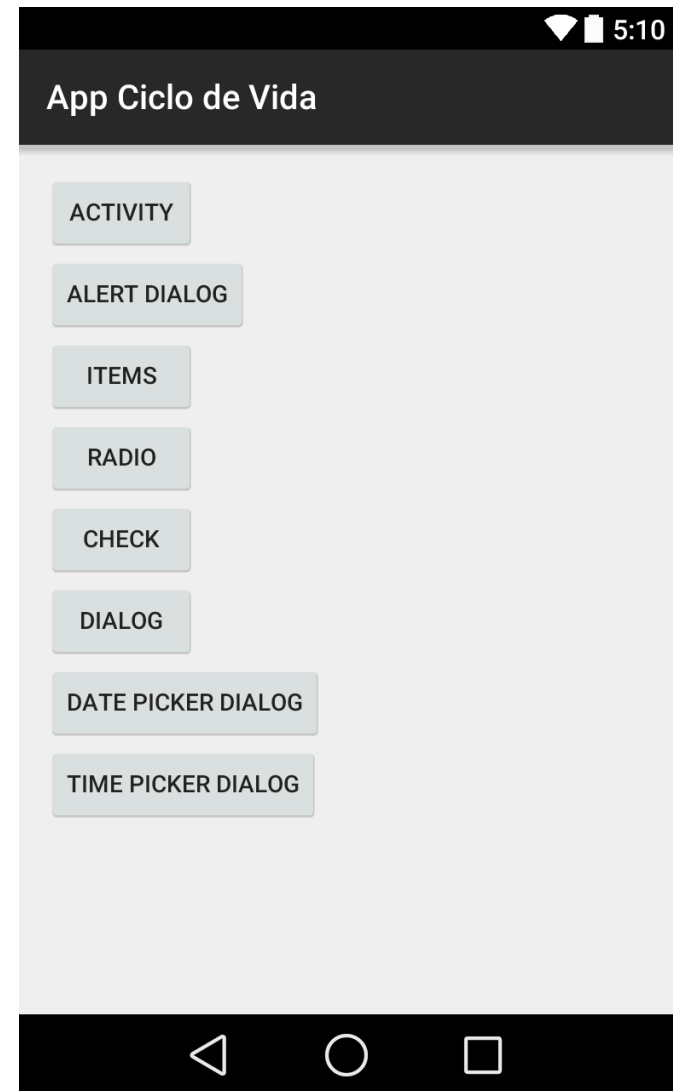
- No arquivo strings.xml, defina o *array de strings* que será utilizado na demonstração dos diálogos:

```
<resources>
  <string name="app_name">App Ciclo de Vida</string>

  <string-array name="listaOpcoes">
    <item>Opção 1</item>
    <item>Opção 2</item>
    <item>Opção 3</item>
    <item>Opção 4</item>
  </string-array>
  ...
</resources>
```

activity_main: LinearLayout

- O arquivo activity_main.xml utiliza um **LinearLayout** e oito botões conforme a figura
- O botão Activity tem id = **btnActivity**
- Adote para os demais o mesmo padrão: **btnAlert**, **btnItems**, **btnRadio**, etc.



activity_slave

- Nova atividade: activity_slave.xml

```
<LinearLayout ...
  android:orientation="vertical">
  <EditText ...
    android:layout_width="match_parent"
    android:id="@+id/editTextSlave" />
  <Button ...
    android:layout_width="match_parent"
    android:id="@+id/btnAdicionar" />
  <TextView ...
    android:layout_width="match_parent"
    android:layout_weight="1"
    android:id="@+id/textViewSlave" />
  <Button ...
    android:layout_width="match_parent"
    android:id="@+id/btnVoltar" />
</LinearLayout>
```



AndroidManifest.xml

- Será criada a classe e definida no *Manifest*

```
<application ...  
  android:label="@string/app_name" >  
  <activity      android:name=".MainActivity"  
                android:label="@string/app_name" >  
    <intent-filter>  
      <action android:name="android.intent.action.MAIN" />  
      <category android:name="android.intent.category.LAUNCHER" />  
    </intent-filter>  
  </activity>  
  <activity      android:name=".SlaveActivity"  
                android:label="@string/title_activity_slave" >  
    </activity>  
</application>
```

MainActivity – Passo 01

- Declare as referências utilizadas na atividade
- O método `onCreate` recupera o vetor de *strings* do recurso e insere no log a mensagem

```
private final String Tag = "Main";
private String[] items;
private int checkedItem;
private boolean[] checkedItems;
private Dialog dialog;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    items = getResources().getStringArray(R.array.listaOpcoes);
    Log.i(Tag, "onCreate");
}
```

MainActivity – Passo 2

- O método `onCreateOptionsMenu` insere no log a mensagem com o mesmo nome do método para verificação de sua chamada

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    getMenuInflater().inflate(R.menu.main, menu);
    Log.i(Tag, "onCreateOptionsMenu");
    return true;
}
```

MainActivity – Passo 3

- Repete nos demais métodos do ciclo de vida

```
@Override
protected void onStart() {
    super.onStart();
    Log.i(Tag, "onStart");
}
@Override
protected void onRestart() {
    super.onRestart();
    Log.i(Tag, "onRestart");
}
@Override
protected void onResume() {
    super.onResume();
    Log.i(Tag, "onResume");
}
```


MainActivity – Passo 4

- Continuando...

```
@Override
protected void onPause() {
    super.onPause();
    Log.i(Tag, "onPause");
}
@Override
protected void onStop() {
    super.onStop();
    Log.i(Tag, "onStop");
}
@Override
protected void onDestroy() {
    super.onDestroy();
    Log.i(Tag, "onDestroy");
}
```

MainActivity – Passo 5

- Faça o mesmo para os métodos que controlam o **InstanceState**

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    Log.i(Tag, "onSaveInstanceState");
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState)
{
    super.onRestoreInstanceState(savedInstanceState);
    Log.i(Tag, "onRestoreInstanceState");
}
```

MainActivity – Passo 6

- Os métodos abaixo são a chamada para a atividade **SlaveActivity** e o respectivo retorno à MainActivity

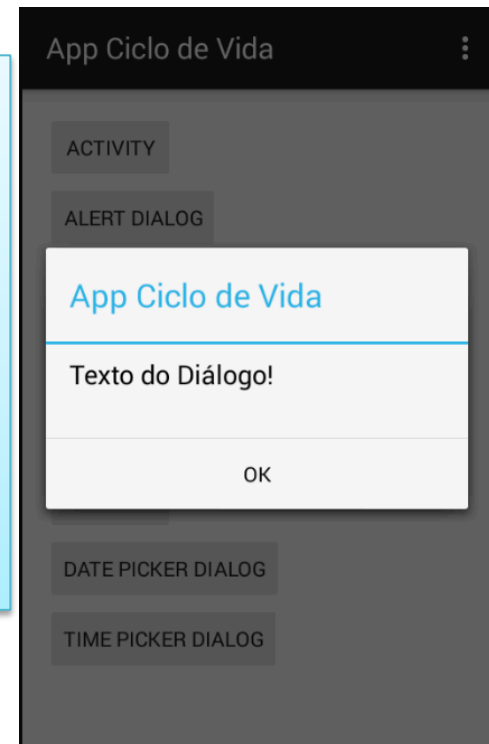
```
public void btnActivityClick(View v) {
    Intent i = new Intent(this, SlaveActivity.class);
    Log.i(Tag, "btnActivityClick");
    startActivityForResult(i, 0);
}

protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == 0)
        if (resultCode == RESULT_OK) {
            Log.i(Tag, "onActivityResult");
            Toast.makeText(this, data.getCharSequenceExtra("ret").toString(),
                Toast.LENGTH_SHORT).show(); }
}
```

MainActivity – Passo 7

- O método `btnDialog1Click` mostra como abrir um diálogo para mostrar uma mensagem para o usuário

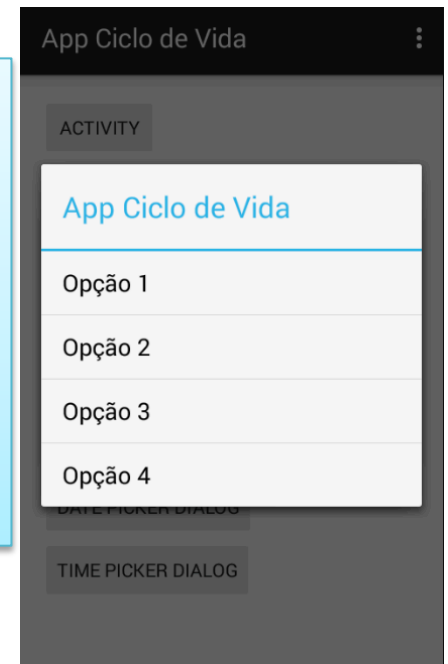
```
public void btnAlertClick(View v) {  
    AlertDialog.Builder builder = new AlertDialog.Builder(this);  
    builder.setTitle(R.string.app_name);  
    builder.setPositiveButton("OK", null);  
    builder.setMessage(R.string.textoDialogo);  
    AlertDialog dialog = builder.create();  
    dialog.show();  
}
```



MainActivity – Passo 8

- O método `btnItemsClick` mostra um diálogo com uma lista de opções
- O método `OnClick` do objeto `listenerItemClick` é chamado na seleção de um item da lista

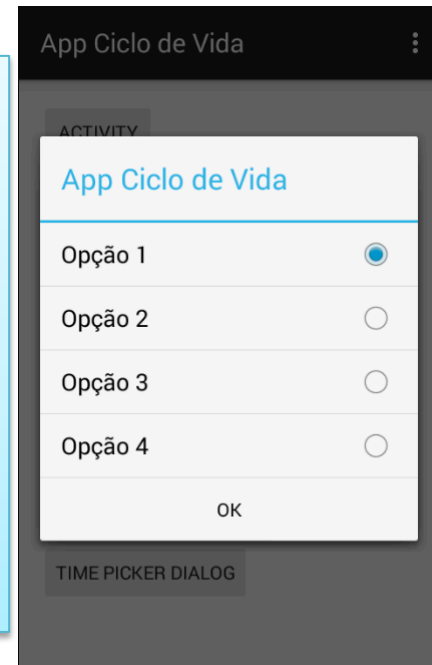
```
public void btnItemsClick(View v) {  
    AlertDialog.Builder builder = new AlertDialog.Builder(this);  
    builder.setTitle(R.string.app_name);  
    builder.setItems(items, listenerItemClick);  
    AlertDialog dialog = builder.create();  
    dialog.show();  
}
```



MainActivity – Passo 9

- O método `btnDialog3Click` mostra um diálogo com uma lista de opções (botões de rádio)
- O método `OnClick` do objeto `listener23` é chamado na seleção de um item da lista

```
public void btnDialog3Click(View v) {  
    checkedItem = 0;  
    AlertDialog.Builder builder = new AlertDialog.Builder(this);  
    builder.setTitle(R.string.app_name);  
    builder.setSingleChoiceItems(items, checkedItem, listenerIC);  
    builder.setPositiveButton("OK", null);  
    AlertDialog dialog = builder.create();  
    dialog.show();  
}
```



MainActivity – Passo 10

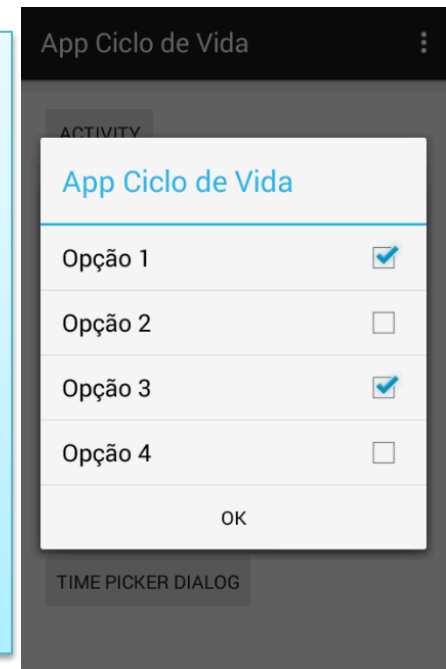
- O código para o objeto listener23 é apresentado abaixo

```
private DialogInterface.OnClickListener listenerItemClick = new
    DialogInterface.OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(MainActivity.this, items[which], Toast.LENGTH_SHORT).show();
    }
};
```

MainActivity – Passo 11

- O método `btnCheckClick` mostra um diálogo com uma lista de opções (caixas de seleção)
- O método `OnClick` do objeto `listenerM` é chamado na seleção de um item da lista

```
public void btnCheckClick(View v) {  
    checkedItems = new boolean[items.length];  
    AlertDialog.Builder builder = new AlertDialog.Builder(this);  
    builder.setTitle(R.string.app_name);  
    builder.setMultiChoiceItems(items, checkedItems, listenerM);  
    builder.setPositiveButton("OK", null);  
    AlertDialog dialog = builder.create();  
    dialog.show();  
}
```



MainActivity – Passo 12

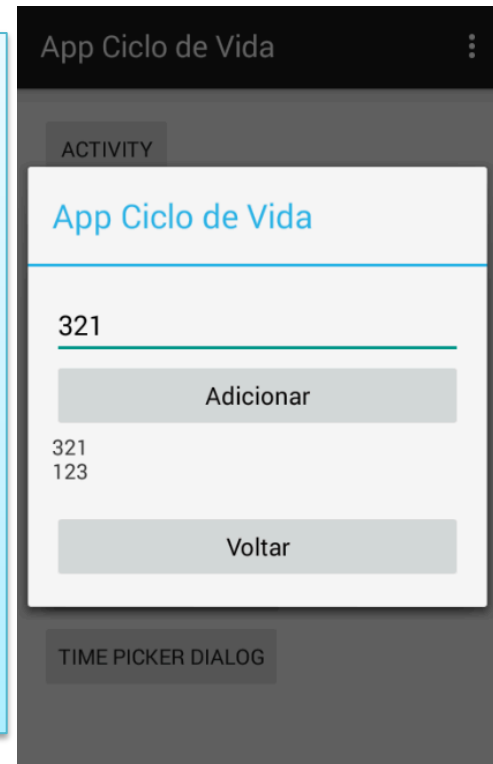
- O código para o objeto `listenerM` é apresentado abaixo

```
private OnMultiChoiceClickListener listenerM = new OnMultiChoiceClickListener() {  
    @Override  
    public void onClick(DialogInterface dialog, int which, boolean isChecked) {  
        Toast.makeText(MainActivity.this, items[which], Toast.LENGTH_SHORT).show();  
    }  
};
```

MainActivity – Passo 13

- O método `btnDialogClick` mostra um diálogo construído a partir de um layout do usuário – Foi utilizado o mesmo layout da atividade `Slave`

```
public void btnDialogClick(View v) {  
    Button btnAdd, btnRet;  
    dialog = new Dialog(this);  
    dialog.setContentView(R.layout.activity_slave);  
    dialog.setTitle(R.string.app_name);  
    btnAdd = (Button) dialog.findViewById(R.id.btnAddicionar);  
    btnRet = (Button) dialog.findViewById(R.id.btnVoltar);  
    btnAdd.setOnClickListener(listenerAdicionar);  
    btnRet.setOnClickListener(listenerVoltar);  
    dialog.show();  
}
```



MainActivity – Passo 14

- Os objetos **listenerAdicionar** e **listenerVoltar**

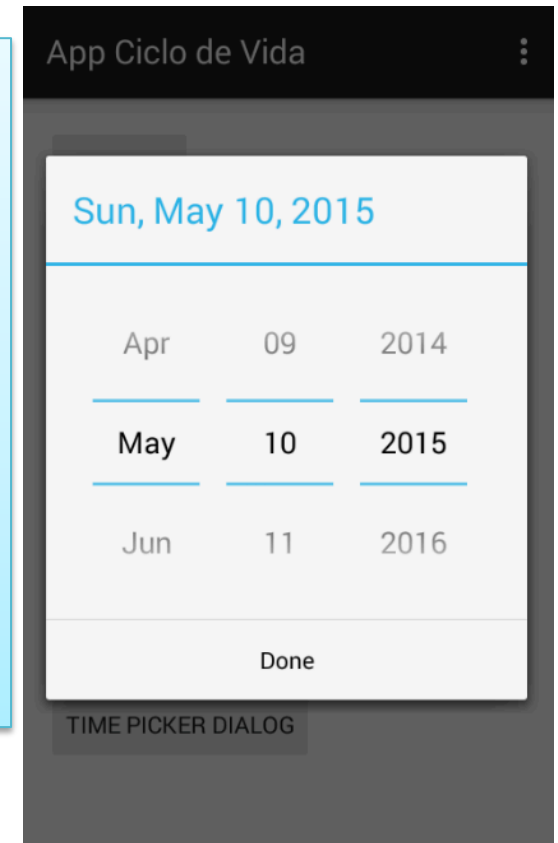
```
private View.OnClickListener listenerAdicionar = new View.OnClickListener() {  
    public void onClick(View v) {  
        EditText editText1 = (EditText) dialog.findViewById(R.id.editTextSlave);  
        TextView textView1 = (TextView) dialog.findViewById(R.id.textViewSlave);  
        String s1 = editText1.getText().toString();  
        String s2 = textView1.getText().toString();  
        textView1.setText(s1 + "\n" + s2); }  
};
```

```
private View.OnClickListener listenerVoltar = new View.OnClickListener() {  
    public void onClick(View v) {  
        TextView textView1 = (TextView) dialog.findViewById(R.id.textViewSlave);  
        Toast.makeText(MainActivity.this, textView1.getText().toString(), Toast.LENGTH_SHORT).show();  
        dialog.dismiss();  
        dialog = null; }  
};
```

MainActivity – Passo 15

- O método `btnDatePickerClick` mostra um diálogo para definição de uma data

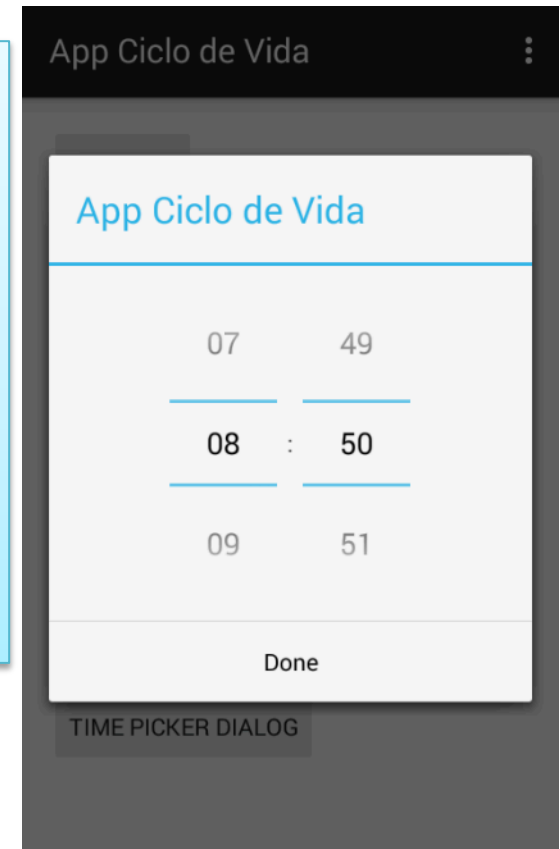
```
public void btnDatePickerClick(View v) {  
    Calendar data = Calendar.getInstance();  
    int dia = data.get(Calendar.DAY_OF_MONTH);  
    int mes = data.get(Calendar.MONTH);  
    int ano = data.get(Calendar.YEAR);  
    DatePickerDialog dialog = new  
        DatePickerDialog(this, listenerDate, ano, mes, dia);  
    dialog.setTitle(R.string.app_name);  
    dialog.show();  
}
```



MainActivity – Passo 16

- O método `btnTimePickerClick` mostra um diálogo para definição de um horário

```
public void btnTimePickerClick(View v) {  
    Calendar data = Calendar.getInstance();  
    int hor = data.get(Calendar.HOUR_OF_DAY);  
    int min = data.get(Calendar.MINUTE);  
    TimePickerDialog dialog = new  
        TimePickerDialog(this, listenerTime, hor, min, true);  
    dialog.setTitle(R.string.app_name);  
    dialog.show();  
}
```



MainActivity – Passo 17

- Os objetos `listenerDate` e `listenerTime` referentes aos diálogos de data e horário

```
private OnDateSetListener listenerDate = new OnDateSetListener() {  
    public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {  
        String msg = String.format("%d/%d/%d", dayOfMonth, monthOfYear, year);  
        Toast.makeText(MainActivity.this, msg, Toast.LENGTH_SHORT).show(); }  
};  
  
private OnTimeSetListener listenerTime = new OnTimeSetListener() {  
    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {  
        String msg = String.format("%d:%d", hourOfDay, minute);  
        Toast.makeText(MainActivity.this, msg, Toast.LENGTH_SHORT).show(); }  
};
```

SlaveActivity – Passo 1

- Declare as referências utilizadas na atividade
- O método `onCreate` insere no log a mensagem para mapear o ciclo de vida da atividade

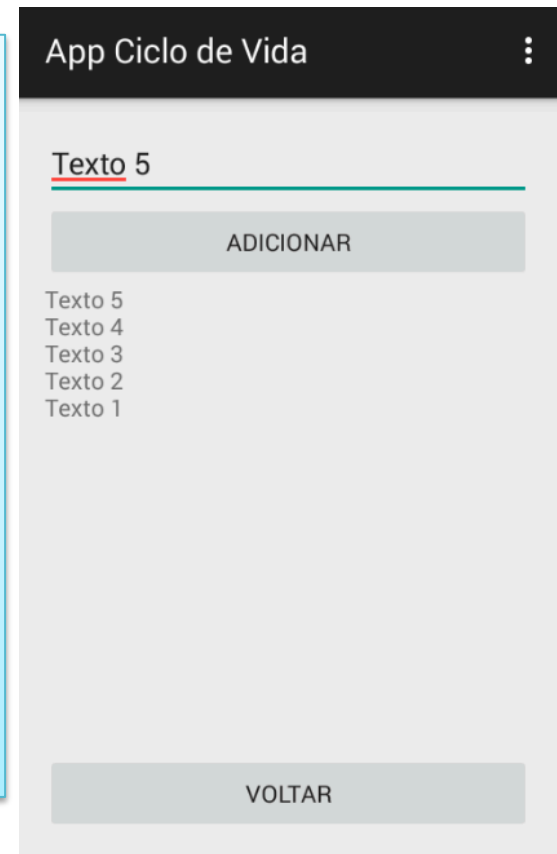
```
private final String Tag = "Slave";
private EditText editText1;
private TextView textView1;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_slave);
    editText1 = (EditText) findViewById(R.id.editTextSlave);
    textView1 = (TextView) findViewById(R.id.textViewSlave);
    Log.i(Tag, "onCreate");
}
```

SlaveActivity – Passo 2

- `buttonAdicionarClick` e `buttonVColtarClick` são os “*listeners*” dos botões Adicionar e Voltar

```
public void btnAdicionarClick(View v) {  
    String s1 = editText1.getText().toString();  
    String s2 = textView1.getText().toString();  
    textView1.setText(s1 + "\n" + s2);  
}  
  
public void btnVoltarClick(View v) {  
    Intent i = new Intent();  
    i.putExtra("ret", textView1.getText().toString());  
    setResult(RESULT_OK, i);  
    finish();  
}
```



SlaveActivity – Passo 3

- Repita os passos 2 a 4 da [MainActivity](#) definindo a mensagem de log para os métodos:
 - onStart
 - onRestart
 - onResume
 - onPause
 - onStop
 - onDestroy

SlaveActivity – Passo 4

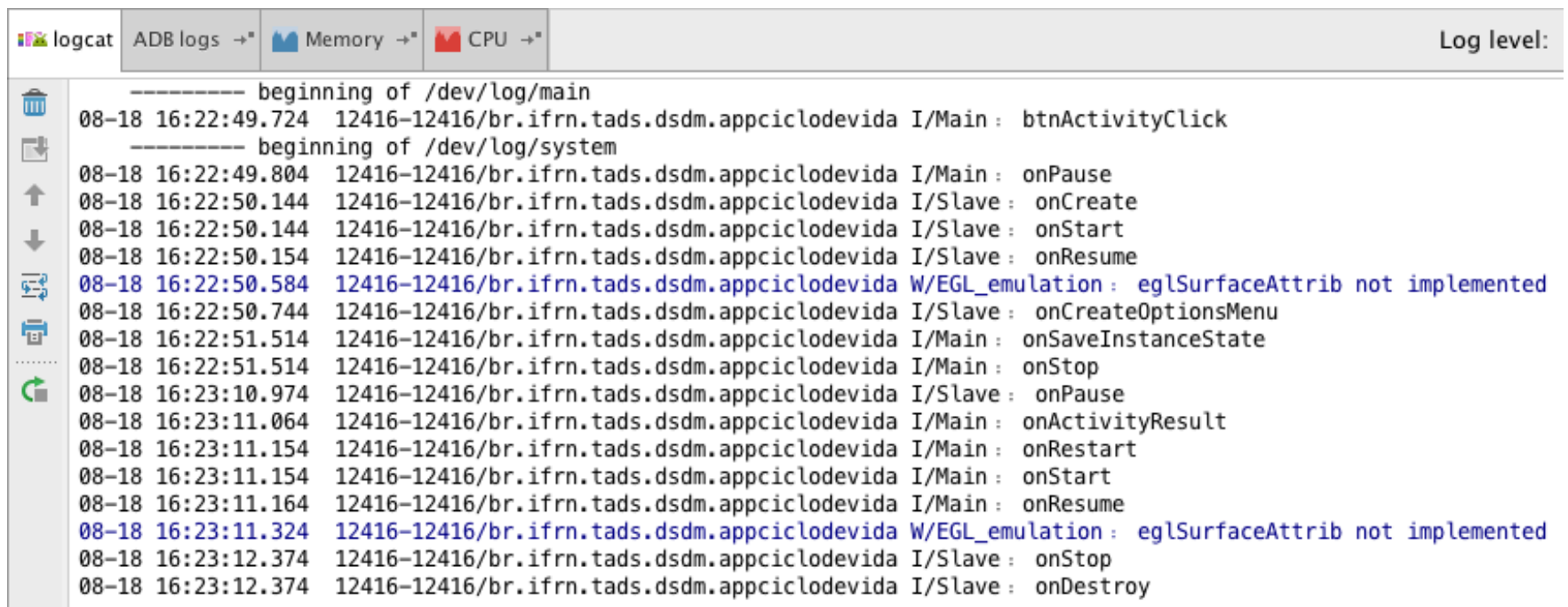
- `saveInstanceState` salva o estado da *activity*
- `onRestoreInstanceState` recupera o estado

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    savedInstanceState.putCharSequence("texto", textView1.getText().toString());
    Log.i(Tag, "onSaveInstanceState");
}

@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    textView1.setText(savedInstanceState.getCharSequence("texto"));
    Log.i(Tag, "onRestoreInstanceState");
}
```

Execute a Aplicação

- Teste a aplicação
- Teste a chamada para a atividade **Slave**, observando as mensagens na janela **LogCat**



The screenshot shows the LogCat window in Android Studio. The window title is "logcat" and it has tabs for "ADB logs", "Memory", and "CPU". The "Log level" is set to "Info". The log output shows the following messages:

```
----- beginning of /dev/log/main
08-18 16:22:49.724 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Main : btnActivityClick
----- beginning of /dev/log/system
08-18 16:22:49.804 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Main : onPause
08-18 16:22:50.144 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Slave : onCreate
08-18 16:22:50.144 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Slave : onStart
08-18 16:22:50.154 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Slave : onResume
08-18 16:22:50.584 12416-12416/br.ifrn.tads.dsdm.appciclodevida W/EGL_emulation : eglSurfaceAttrib not implemented
08-18 16:22:50.744 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Slave : onCreateOptionsMenu
08-18 16:22:51.514 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Main : onSaveInstanceState
08-18 16:22:51.514 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Main : onStop
08-18 16:23:10.974 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Slave : onPause
08-18 16:23:11.064 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Main : onActivityResult
08-18 16:23:11.154 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Main : onRestart
08-18 16:23:11.154 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Main : onStart
08-18 16:23:11.164 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Main : onResume
08-18 16:23:11.324 12416-12416/br.ifrn.tads.dsdm.appciclodevida W/EGL_emulation : eglSurfaceAttrib not implemented
08-18 16:23:12.374 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Slave : onStop
08-18 16:23:12.374 12416-12416/br.ifrn.tads.dsdm.appciclodevida I/Slave : onDestroy
```

Recuperando o Estado

- Abra a atividade **Slave** e insira dados
- Gire o dispositivo (emulador = Ctrl+F12)
- E confira o que acontece no LogCat

Referências

- Android para Programadores – Uma abordagem baseada em aplicativos. Paul Deitel ... [et al.]. Bookman, 2013
- Google Android – Aprenda a criar aplicações para dispositivos móveis com o Android SDK. Ricardo R. Lecheta. Novatec, 2013
- <http://developer.android.com/reference>