Tópicos Avançados em Programação

Aula 2

Prof. Marcelo Rodrigues do Nascimento

Conversa Inicial

Seja bem-vindo à segunda aula de Tópicos Avançados de Programação. Na aula anterior aprendemos sobre a plataforma Android, seu comportamento e alguns de seus principais fundamentos. Agora chegou a hora de nos aprofundarmos um pouco mais na linguagem de programação, colocando à prova alguns dos conceitos que já absorvemos e ampliando nosso conhecimento.

Ao finalizarmos esta etapa você compreenderá o conceito de Activities, seu ciclo de vida, como enviar mensagens (Intents) e também como criar uma interface que responda aos comandos do usuário. Vamos começar?

No material online, o professor Marcelo apresenta os temas que serão abordados!

Contextualizando

No desenvolvimento de aplicativos para Android contamos com uma extensa biblioteca de componentes que nos auxiliam a tornar a tarefa de criar aplicativos cada vez mais simples. No entanto, ainda que esta biblioteca esteja disponível de forma gratuita, é de suma importância entendermos como funcionam os principais aspectos da plataforma, para que possamos obter um resultado profissional quando decidimos escrever nossos próprios aplicativos.

Nosso objetivo hoje é trabalhar diretamente com o primeiro dos quatro componentes de aplicativo do Android, as Activities. Vamos entender como estas funcionam, qual seu comportamento padrão, seu ciclo de vida e também como transportar informações a respeito de um objeto de uma Activity para o sistema operacional, para que este crie uma nova Activity. Também daremos início ao conceito de interface, customizando nosso projeto para que este receba um objeto que responda aos comandos do usuário.

O professor Marcelo contextualiza esse assunto no material online.

Tema 1 – Activites

Como vimos na primeira aula, um dos blocos de construção de aplicativos Android são as Activities. Cada Activity recebe um layout que representa a interface do usuário, com botões e caixas de texto, por exemplo. Aplicativos geralmente possuem uma entrada principal que é mostrada ao usuário quando este executa o aplicativo pela primeira vez, e a partir daí várias activities vinculadas entre si. Diferentemente dos sistemas desktop, onde a uma janela é delegada várias ações, é recomendado que tenhamos apenas um tipo de ação por Activity, como por exemplo: em um aplicativo de leitura de e-mails, teremos uma Activity principal que listará os e-mails recebidos e que, ao se clicar em uma determinada mensagem iniciará uma nova Activity, que trará detalhes a respeito da mensagem selecionada em um painel de leitura.

Neste painel de leitura teremos opções referentes ao que é possível ao usuário executar após a leitura daquela mensagem, como enviar uma resposta ou encaminhá-la a outro destinatário. Se o usuário decidir por responder a esta mensagem, ao clicar no respectivo botão, uma nova Activity será disparada, fornecendo um editor de texto que permitirá a escrita da resposta.

Essa estrutura, que pode parecer confusa à primeira vista tem como objetivo simplificar as interfaces com as quais o usuário interage. Cada Activity deve ser desenvolvida tendo como um de seus objetivos criar uma experiência agradável ao usuário, de forma que este consiga executar o que deseja da forma mais natural possível.

Como o sistema operacional Android trata este volume de Activities que são abertos para a execução de uma tarefa corriqueira, como a leitura e resposta de um e-mail?

Um dos primeiros conceitos que devemos entender a respeito das Activities é que estas são ordenadas em forma de pilha, no formato UEPS (último a entrar, primeiro a sair). A cada Activity criada, a Activity anterior é interrompida,

mas armazenada na pilha para que possa ser reestabelecida quando o usuário pressionar o botão de retorno.

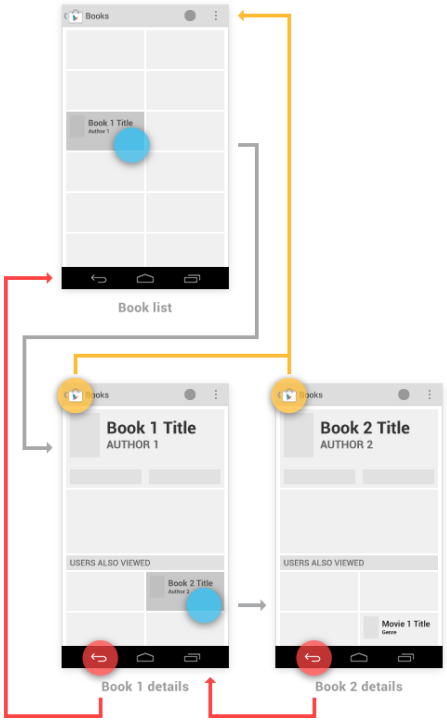

Figura 1 – Activity Stack

Na figura anterior percebemos que, ao usuário tocar no item que representa o Livro 1, uma nova Activity é disparada, trazendo detalhes a respeito do mesmo. Nesta segunda Activity apresentamos ao usuário a possibilidade de obter maiores detalhes a respeito de outros livros que foram visualizados por usuários que também pesquisaram a respeito do Livro 1, neste caso o Livro 2.

Caso o usuário toque no item que representa o Livro 2 uma nova Activity, que também apresenta detalhes a respeito do livro selecionado é disparada. Quando o usuário toca no botão de voltar, comum a todos os dispositivos Android, a Activity atual é destruída e a última Activity disparada é chamada novamente, e assim sucessivamente até que a primeira Activity (ponto de entrada) seja atingida.

Caso o usuário insista em pressionar o botão de retorno ao chegar na primeira Activity, o aplicativo será enviado para segundo plano e lá permanecerá

até que seja chamado novamente ou que o sistema necessite de recursos e acabe por destruí-lo por completo.

Este é um ponto muito interessante a se comentar a respeito do comportamento padrão de aplicativos Android. Ao contrário do que ocorre no Desktop, que quando o botão de encerrar o programa é clicado o aplicativo é imediatamente destruído, no Android o mesmo é enviado para segundo plano. O desenvolvedor pode optar for forçar que o mesmo seja destruído, mas este não é o comportamento recomendado.

Normalmente delega-se a responsabilidade de finalização por completo do aplicativo ao sistema operacional Android, que o fará quando julgar necessário (normalmente quando os recursos do sistema operacional estão escassos). Este é o motivo pelo qual, quando clicamos no botão de tarefas em um dispositivo Android muitas vezes somos surpreendidos com vários aplicativos que estavam em segundo plano.

Vamos entender agora como isto é possível estudando o ciclo de vida de uma Activity. Podemos dizer que uma Activity possui quatro estados e sete métodos principais, sendo eles:

Estados

- A Activity não existe;
- A Activity está em primeiro plano;
- A Activity está em segundo plano;
- A Activity está em pausa.

Métodos

- onCreate();
- onStart();
- onResume();
- onPause();
- onStop();

- onRestart();
- onDestroy();

Mas, como estes estados e estes eventos se relacionam?

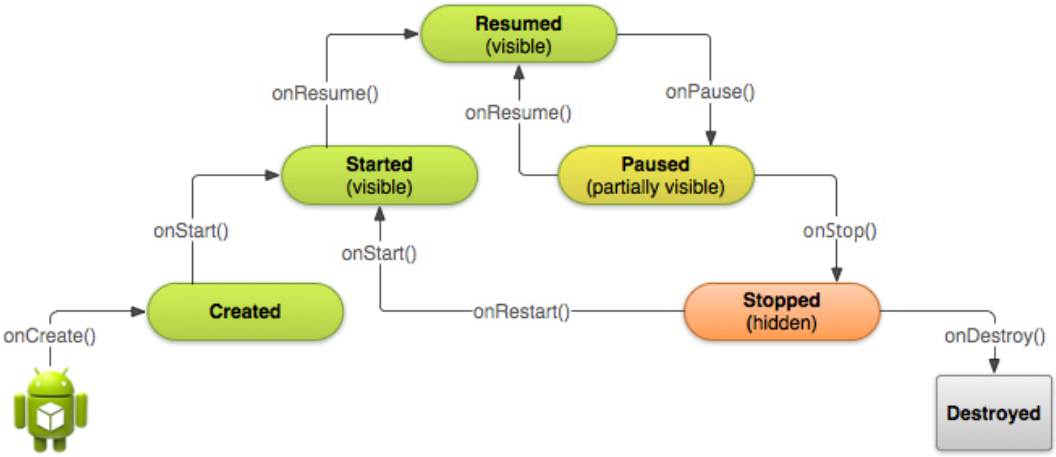

Figura 2 – Activity Life Cycle

Quando iniciamos uma Activity, ela parte do estado de “não existente” para o estado de primeiro plano. Para que o estado da Activity mude para o primeiro plano, passamos por três dos principais métodos de uma atividade: **onCreate()**, **onStart()** e finalmente **onResume()**.

Se uma segunda Activity for disparada, esta entrará em primeiro plano e a Activity inicial passará para o estado de segundo plano, passando pelos métodos **onPause**() e **onStop**(). Lembre-se que isto ocorreu somente com a primeira Activity. A segunda Activity passará então do estado de “não existente” para o estado de primeiro plano, percorrendo os métodos **onCreate()**, **onStart()** e **onResume()**.

Digamos agora que, após a segunda Activity passar para o primeiro plano, nosso usuário receba, por exemplo, uma notificação de chamada telefônica. A segunda Activity passará pelo método de **onPause(),** entrando no estado de “em pausa”, pois está parcialmente visível (existe uma notificação sobre ela). Caso o

usuário opte por atender a chamada, nossa segunda Activity continuará o caminho rumo ao segundo plano, passando pelo método **onStop()**.

Nosso usuário agora terminou de atender a sua ligação, e agora deseja retornar ao nosso aplicativo. Ele selecionará nosso aplicativo da lista de aplicativos ativos e a segunda Activity sairá então do estado de "em segundo plano" e entrará no estado de "primeiro plano", passando pelos métodos **onRestart()**, **onStart()**, e finalmente **onResume()**.

Quando nosso usuário finalizar seu interesse na segunda Activity e clicar no botão de retorno, esta será destruída, passando para o estado de "não existe". Isto ocorre com a chamada dos métodos **onPause()**, **onStop()** e finalmente **onDestroy()**.

Lembra-se de que nossas activities são armazenadas no formato de pilha? Neste caso, acabamos de remover o topo da pilha, tornando visível a Activity que estava logo abaixo dela, ou seja a Activity inicial. Como isso ocorreu? A Activity inicial saiu agora do estado de segundo plano (estava abaixo da segunda Activity) e foi para o primeiro plano, ao chamar os métodos **onRestart()**, **onStart()** e **onResume()**.

Confuso? O professor Marcelo esclarece isso de uma forma mais prática no material online.

Tema 2 – Monitorando o ciclo de vidas de uma Activity

O que acha de testarmos este conceito de ciclo de vida de atividades criando agora uma aplicação Android que monitore todos os eventos disparados para o debugger do Android Studio?

Inicie o Android Studio e clique na opção "**Start a new Android Studio project**"

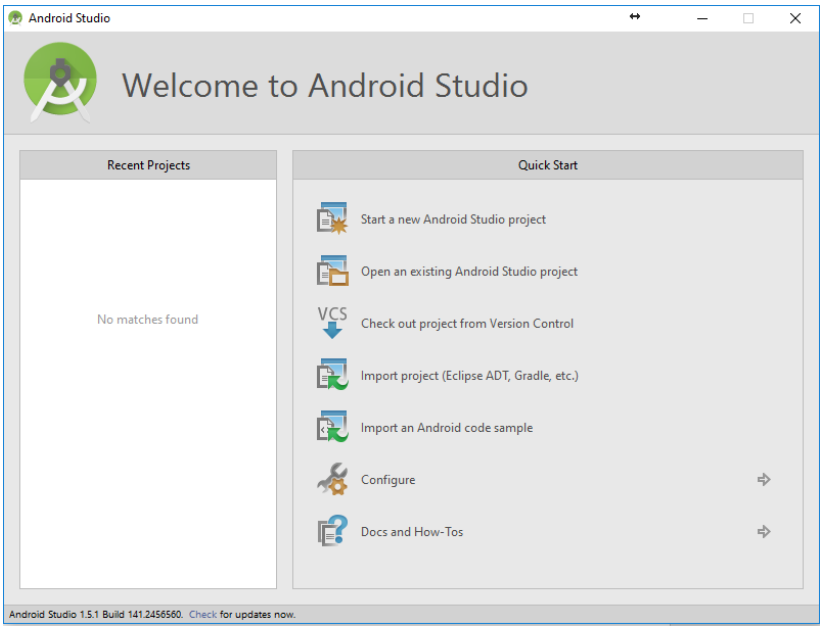

Figura 3 – Criando um novo projeto

Em Application name digite “**ActivityLifeCycle**”

Em Company Domain digite activitylifecycle.grupouninter.com.br

Perceba que, ao colocarmos este nome de domínio, o nome de pacote abaixo é preenchido automaticamente com “br.com.grupouninter.aula2.activitylifecycle”.

Este é o formato padrão de pacotes Java, e é criado desta maneira por questões de indexação.

Em uma estrutura de diretórios, este pacote estaria organizado da seguinte forma:

```
br/com/grupouninter/aula2/activitylifecyle
```

Como podemos perceber todos os pacotes referentes ao mesmo company name estariam alocados no mesmo diretório, modificando apenas o nome do subdomínio e módulo.

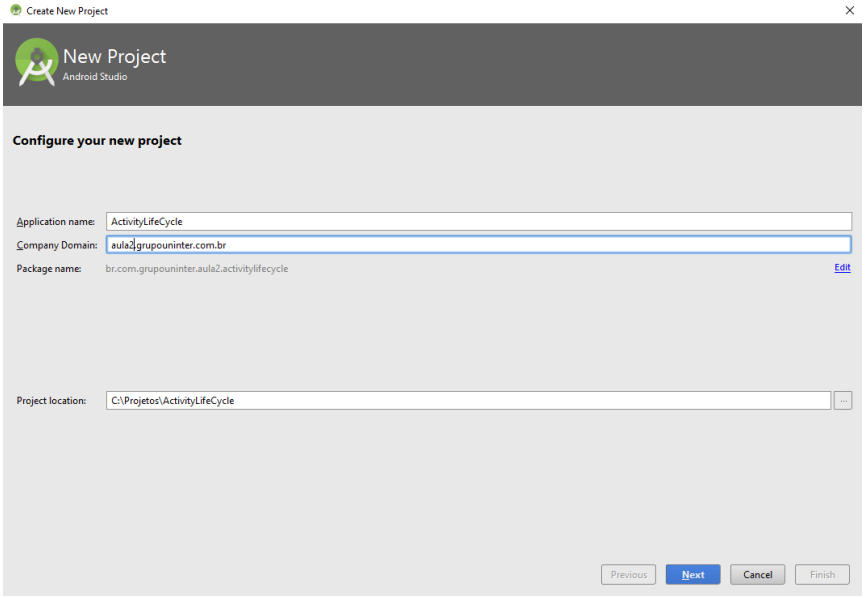

Figura 4 – Nome da aplicação

Clique no botão Next e selecione como dispositivo alvo telefones e tablets. Não se preocupe com o SDK mínimo neste momento, esta é apenas uma aplicação de prova de conceito.

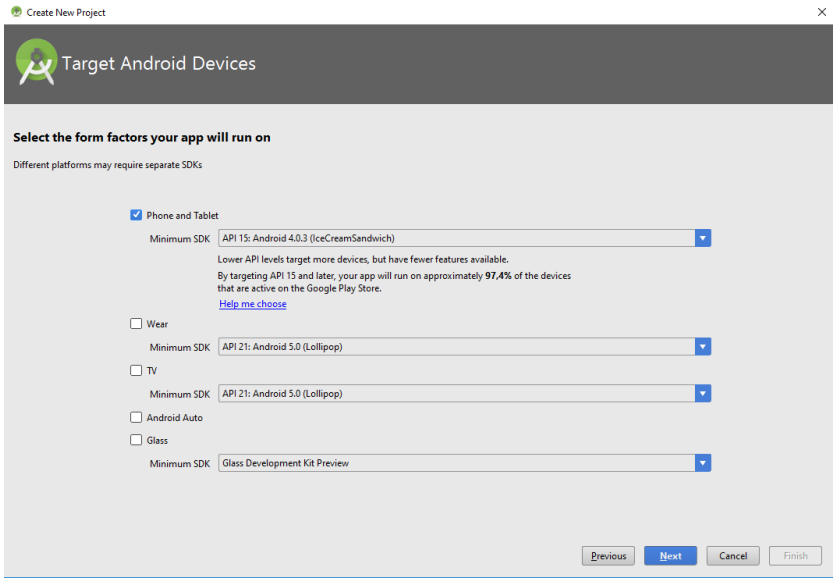

Figura 5 – Seleção de Dispositivos alvo

Clique no botão Next e vamos então adicionar uma Activity para nosso aplicativo. Selecione a opção “Empty Activity” para que seja criada uma Activity em branco, sem o Action Bar (Barra de ações) ou o Floation Action Button (botão de ações flutuante).

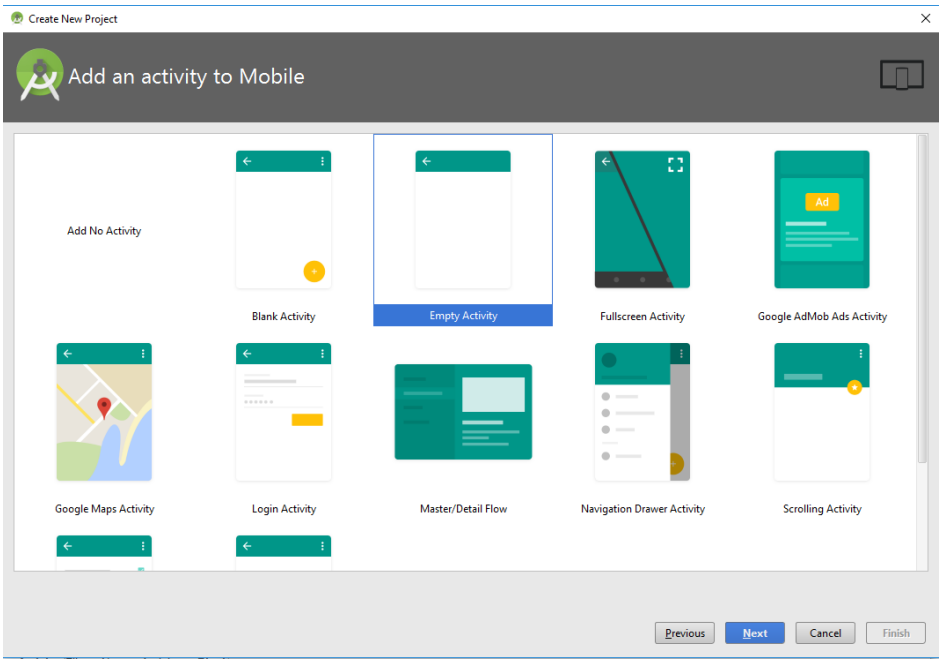

Figura 6 – Empty Activity

Clique em Next e então daremos à Activity que acabamos de criar o nome de sua classe e também a qual arquivo de layout ela está vinculada. Para este exemplo, vamos deixar os valores padrão, ou seja, **MainActivity** para o nome da Activity e **activity_main** para o nome do Layout.

Clique então no botão Finish.

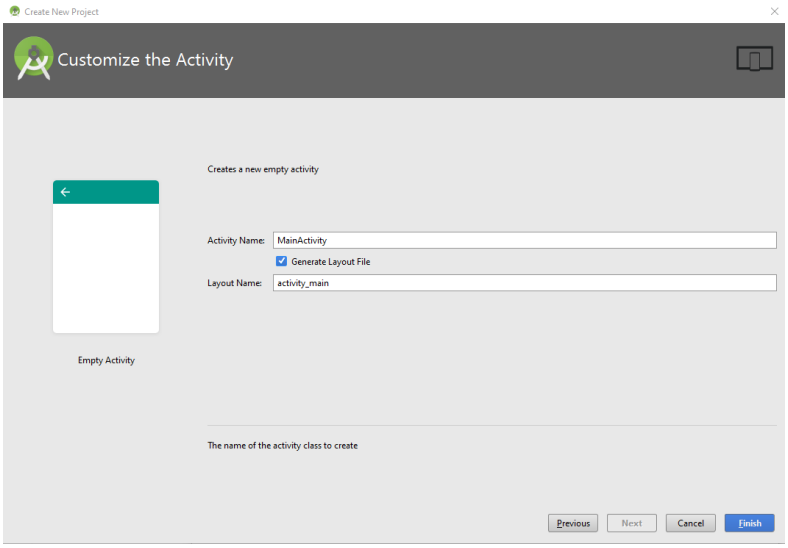

Figura 7 – Activity Name

Aguarde alguns momentos para que o projeto seja gerado e então a tela principal do Android Studio será apresentada, já com o arquivo referente à nossa Activity em evidência. Antes de iniciarmos o desenvolvimento, vamos observar por um momento as janelas que compõe a nossa interface de desenvolvimento. A seguir vemos a estrutura de nosso projeto Android.

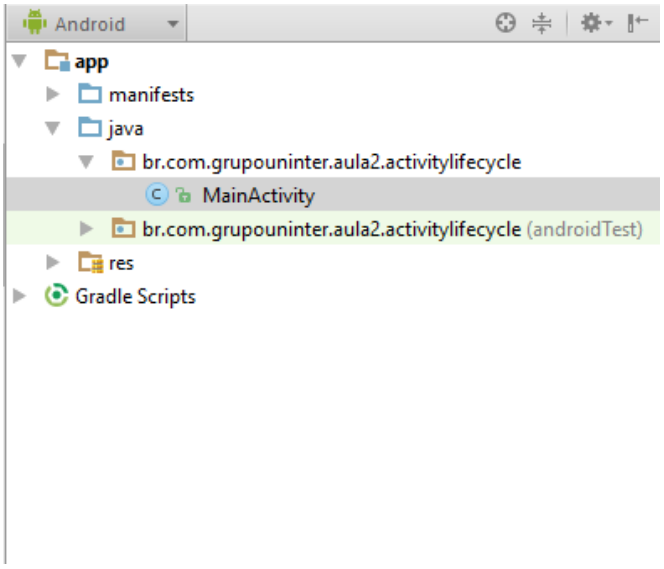

Figura 8 – Estrutura do Projet

Nesta estrutura observamos um conjunto de pastas:

- app – nome de nosso módulo; (Clique nas palavras destacadas):

manifests – nesta pasta temos o nosso arquivo Android Manifest;

java – nesta pasta temos nossos pacotes e as classes criadas por nós ou automaticamente (para casos de retrocompatibilidade, por exemplo);

res – pasta com os recursos deste aplicativo (ícones, layouts, etc).

- Graddle Scritps – aqui temos a estrutura que contém os scripts de inicialização e configuração do Graddle;

Em nossa janela de editor, vemos dois arquivos abertos: MainActivity.java (selecionado) e activity_main.xml. Nossa MainActivity é composta da junção destes dois arquivos: no arquivo ManActivity.java desenvolvemos a classe Java que controla esta Activity enquanto no arquivo activity_main.xml criamos a interface que é apresentada ao usuário quando esta Activity é acionada.

Figura 9 - Editor

Lembre-se de que a janela do editor se modifica de acordo com o tipo de arquivo nela selecionado. Se o arquivo selecionado for do tipo Java, ele abrirá o editor com todas as funcionalidades presentes para a programação. No entanto, ao selecionarmos o arquivo activity_main.xml nosso editor muda para o editor de

layouts, com os componentes de criação de layout disponíveis, uma árvore que mostra a hierarquia entre os objetos disponíveis do layout e também uma janela de propriedades que nos auxilia na modificação do comportamento do componente selecionado.

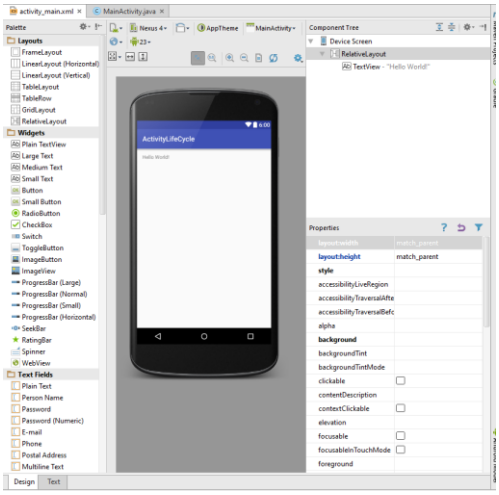

Figura 10 – Editor de Layout

Perceba que nesta janela do editor temos duas abas inferiores, Design e Text. Selecionando-se a opção Design temos acesso a uma visualização renderizada de como ficará nosso layout. Caso seja selecionada a opção Text na barra inferior entraremos no editor de XML em modo texto para este arquivo.

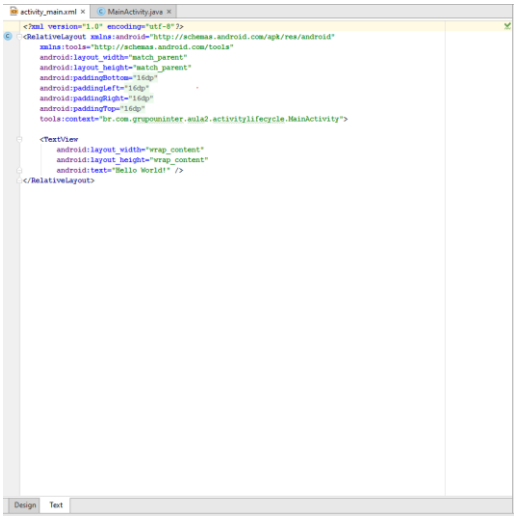

Figura 11 – Editor de Layout na visão XML

Voltando ao nosso arquivo MainActiviy.java, na janela de Edição, temos o código gerado pelo Android Studio para nosso template:

```
package br.com.grupouninter.aula2.activitylifecycle;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Como você pode perceber, os métodos que discutimos anteriormente não foram declarados, à exceção do método **onCreate()**. Isso ocorre porque nossa classe MainActivity herda suas propriedades e métodos da classe APPCompatActivity, que já define o comportamento padrão para estes métodos. Para efeito deste exemplo, vamos então sobrescrever os métodos e assim monitorar seu comportamento. Adicione em seu código os seguintes métodos:

```
@Override
protected void onStart(){
    super.onStart();
}

@Override
protected void onResume(){
    super.onResume();
}

@Override
protected void onPause(){
    super.onPause();
}
```

```
@Override
protected void onStop(){
    super.onStop();
}

@Override
protected void onRestart(){
    super.onRestart();
}

@Override
protected void onDestroy(){
    super.onDestroy();
}
```

Após esta declaração podemos dar início à modificação de comportamento, para que possamos acompanhar através do debugger do Android Studio as chamadas aos métodos à medida que mudamos o estado de nosso aplicativo.

Crie uma variável estática que contenha o nome de nossa Activity, para que se torne mais fácil acompanharmos os resultados através do arquivo de log:

```
...
public class MainActivity extends AppCompatActivity {

    private static String ACTIVITY = "MainActivity";

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

Para fins de debug, utilizaremos a classe **Log,** chamado o método **d,** que envia as mensagens por ela disparadas para o debug do Android Studio, onde poderemos filtrar o resultado.

Modifique agora todos os métodos para que estes enviem ao Debug a mensagem de que foram disparados:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d(ACTIVITY, "onCreate()");
}

@Override
protected void onStart(){
    super.onStart();
    Log.d(ACTIVITY,"onStart()");
}

@Override
protected void onResume(){
    super.onResume();
    Log.d(ACTIVITY, "onResume()");
}

@Override
protected void onPause(){
    super.onPause();
    Log.d(ACTIVITY, "onPause()");
}

@Override
protected void onStop(){
    super.onStop();
    Log.d(ACTIVITY, "onStop()");
}

@Override
protected void onRestart(){
    super.onRestart();
    Log.d(ACTIVITY, "onRestart()");
}

@Override
protected void onDestroy(){
    super.onDestroy();
    Log.d(ACTIVITY, "onDestroy()");
}
```

Pressione Shift + F10 (ou clique no botão de execução do aplicativo) para selecionar o emulador a ser utilizado:

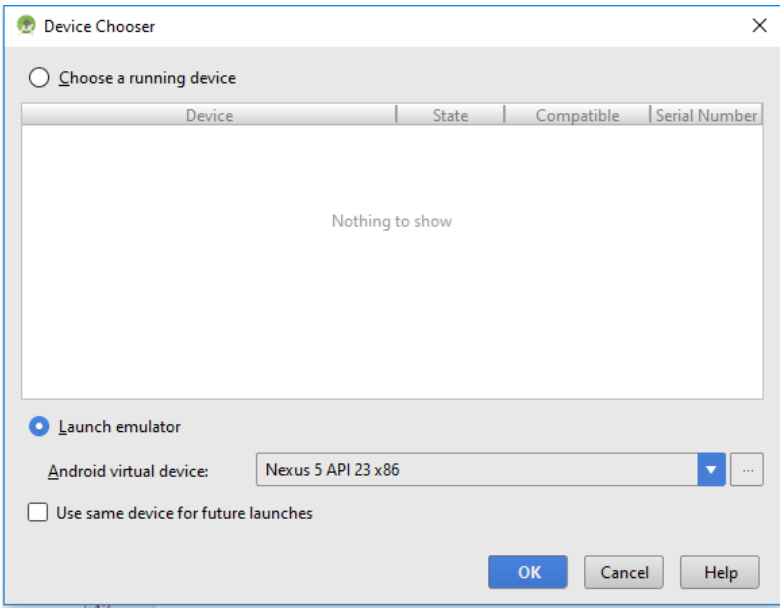

Figura 12 – Escolher o emulador

Selecione o emulador desejado e pressione OK. Marque a opção “Use same device for future launches” para utilizar este emulador como padrão para este projeto. O emulador pode levar algum tempo até que seja iniciado, portanto tenha paciência.

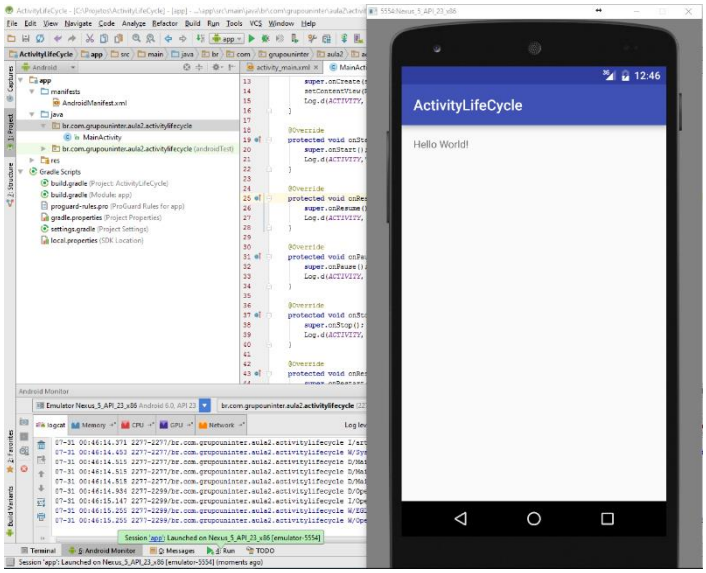

Figura 13 – Emulador em execução

Como podemos ver, nosso emulador está agora em execução e nosso aplicativo, com sua Activity principal está em primeiro plano. Como podemos então capturar as mensagens de Log no debugger?

Volte ao Android Studio (não feche o emulador) e então selecione o Android Monitor. Nesta tela, em Log level selecione a opção Debug e então digite na caixa de pesquisa ao lado o conteúdo de nossa variável estática, no caso "**MainActivity**"

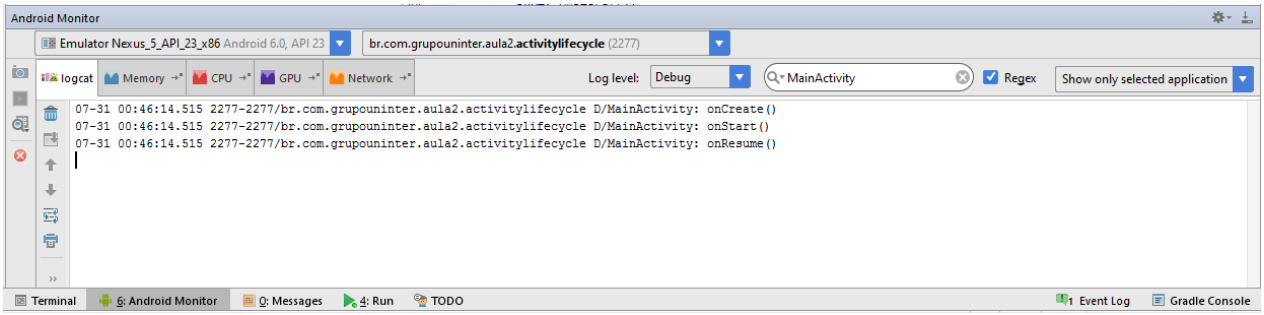

Figura 14 – Android Monitor

Como você pode perceber, os métodos **onCreate()**, **onStart()** e **onResume()** foram disparados quando o aplicativo foi iniciado.

Volte agora ao nosso emulador (lembre-se de deixar a tela do Android Studio visível abaixo dele) e pressione o botão de retorno. O que aconteceu? Em seu emulador, sua aplicação foi encerrada e podemos perceber no Android Monitor que os eventos foram todos corretamente disparados.

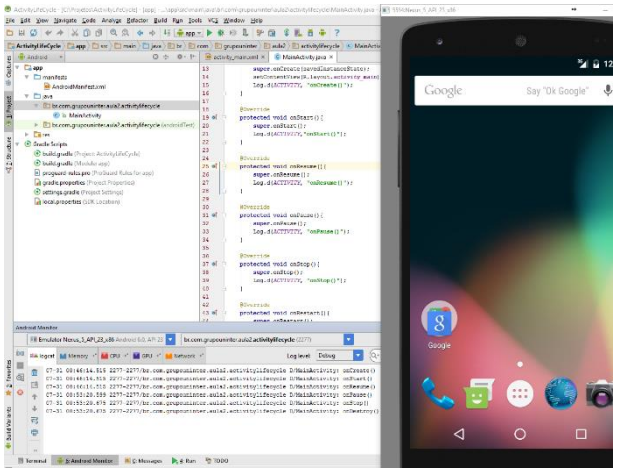

Figura 15 – Eventos disparados

Clique agora no botão de janela e veja que seu aplicativo não foi realmente destruído, mas sim enviado ao segundo plano, aguardando seu retorno (ou então sua destruição ocorreu devido a uma necessidade de recursos). Pressione novamente no aplicativo para colocá-lo em segundo plano.

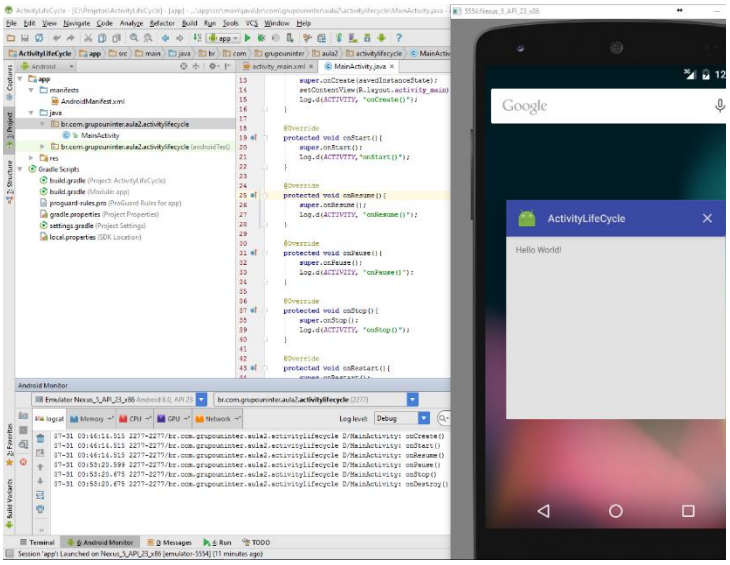

Figura 16 – Aplicativo não destruído

Ao trazermos novamente nosso aplicativo para o primeiro plano percebemos que os métodos foram novamente executados, conforme esperávamos.

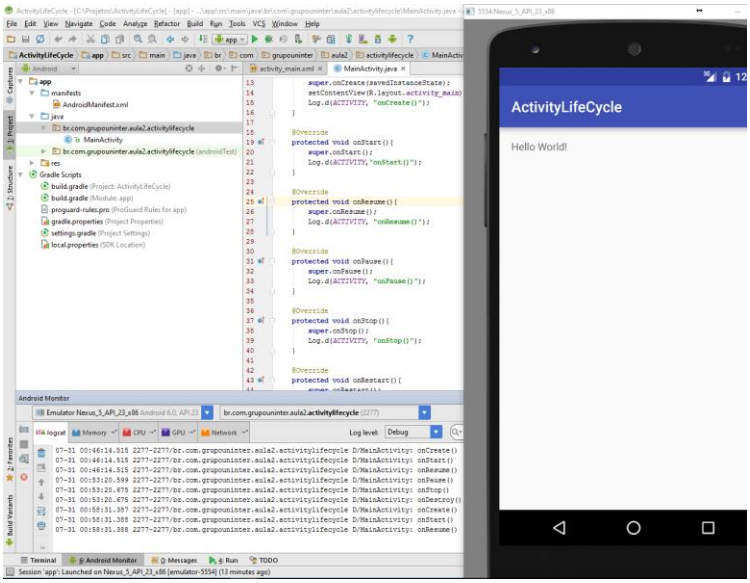

Figura 17 – Eventos disparados

Conseguimos provar então o conceito do ciclo de vida de uma Activity, criando uma aplicação que o monitora e envia mensagens para o Debug do Android Monitor. Apesar deste aplicativo ter se provado bem interessante como forma de teste do ciclo de vida das Activities, que tal expandi-lo um pouco mais?

Saiba como acessando a explicação do professor Marcelo, no material online.

Tema 3 - Intent

Nós já conhecemos os quatro componentes básicos de construção para aplicativos: Android – Activities, Services, Broadcast Receivers e Content Providers. Também aprendemos a respeito de como o primeiro desses componentes – Activity – se comporta e comprovamos seu ciclo de vida.

Entendemos também que aplicativos Android devem levar em consideração primeiramente a experiência para o usuário, portanto uma boa prática é separar as operações a serem executadas em diferentes activities.

Mas, como fazemos isso? Como criar uma nova Activity a partir de uma já existente? Realizamos essa operação criando um objeto de mensagem que será utilizado para solicitar uma ação a outro componente de aplicativo.

Este objeto de mensagem é conhecido como Intent, e trabalha com três dos componentes de aplicativo, mas por enquanto vamos aprender a respeito de seu comportamento na Activity.

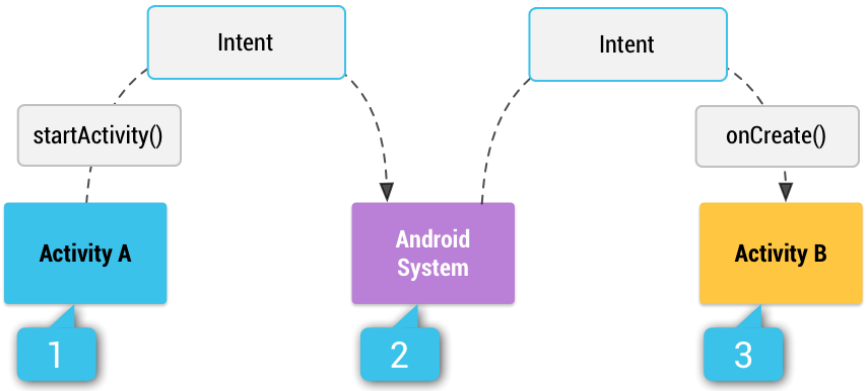

Figura18 - Intent

Iniciamos uma nova Activity passando sua referência a uma Intent e então passando esta Intent ao método **startActivity()**. Nesse caso a Intent descreverá a atividade a iniciar e carregará todos os dados necessários. Caso você necessite obter um resultado quando a nova Activity finalizar, basta passar a Intent ao método **startActivityForResult()**. Assim, sua Activity estará apta a receber o resultado como um objeto Intent separado no retorno de chamada **onActivityResult()**.

Certo, mas que Activity iremos utilizar, já que temos apenas uma até o momento? Este é um bom lugar para começarmos, vamos criar uma nova Activity, sua classe Java e seu layout correspondente. Abra o Android Studio e carregue novamente nosso projeto **ActivityLifeCycle**. Em sua janela de Projeto, selecione o pacote referente a nosso aplicativo e então clique em **New, Java Class.**

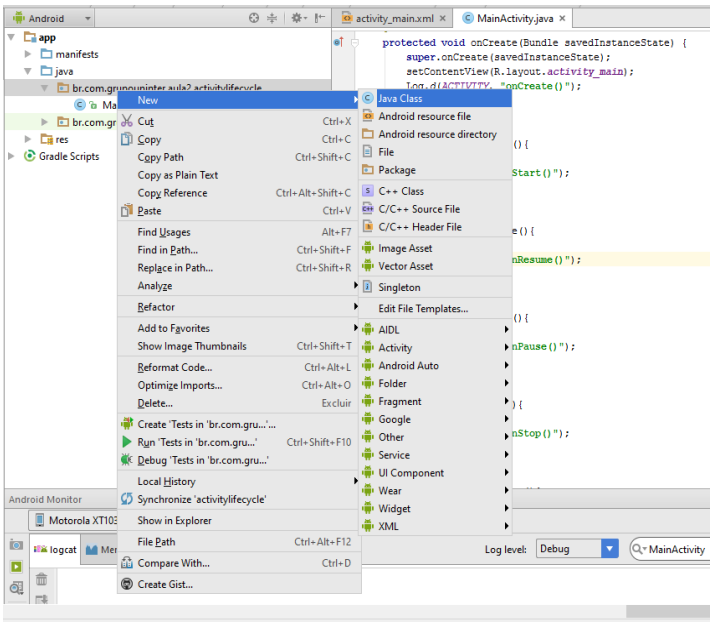

Figura 19 – Nova classe Java

A janela de criação de nova classe será acionada. Em Name preencha o nome de sua nova Activity, em nosso caso, SegundaActivity e pressione OK

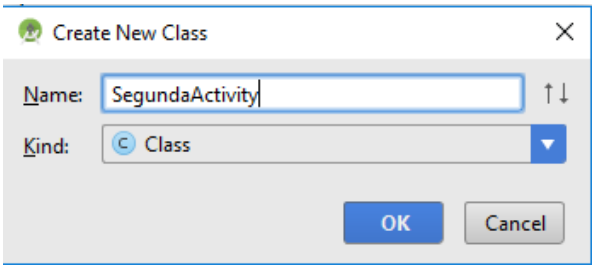

Figura 20 – Segunda Activity

Será criada uma classe pública, ainda sem formato de Activity.

```
package br.com.grupouninter.aula2.activitylifecycle;

/**
 * Created by Marcelo on 31/07/2016.
 */
public class SegundaActivity {
}
```

A primeira coisa que devemos fazer é estender essa classe, para que ela comece a se comportar como uma Activity. Fazemos isso utilizando a palavra-chave **extends** seguida da classe que queremos herdar o comportamento, ou seja, **Activity**.

Ao se digitar este comando a palavra Activity ficará em vermelho, indicando que ocorreu um erro ou alguma condição não foi satisfeita. Em nosso caso, é porque ainda não importamos os pacotes que descrevem nossa Activity. Coloque o cursor sobre a palavra Activity e pressione Alt+Enter.

Este é o atalho que nos traz sugestões de como proceder quando esses eventos ocorrem. Como você pode ver, ele traz como sua primeira opção a importação da classe para nosso projeto, e é justamente isso que faremos.

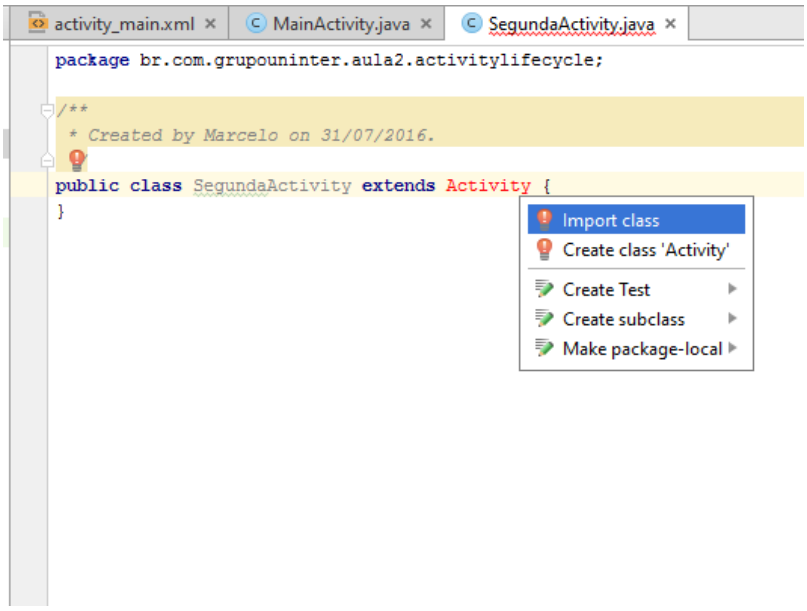

Figura 21 – Importanto a classe Activity

Foi adicionada a linha de importação da classe Activity ao nosso código fonte. Agora devemos redefinir o método **onCreate()** desta classe, para que ela responda da maneira como desejamos e não somente como a classe Activity responderia normalmente. Não esqueça também de adicionar a importação de *android.os.Bundle,* já que utilizaremos o objeto Bundle (agrupamento de objetos) como forma de armazenamento do status atual de nossa Activity.

```
package br.com.grupouninter.aula2.activitylifecycle;

import android.app.Activity;
import android.os.Bundle;

/**
 * Created by Marcelo on 31/07/2016.
 */
public class SegundaActivity extends Activity {

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

    }
}
```

Até agora estendemos então a classe Activity e sobrescrevemos o método **onCreate()** para que este represente o que desejamos. No entanto no momento ele está apenas passando para a classe da qual estendemos, no caso **Activity** o valor de nossa variável Bundle savedInstance.

O que precisamos agora é criar um layout que representará a interface com nosso usuário. Em sua janela de Projeto, expanda a pasta **res** e depois a pasta **layout**. Nesta pasta são armazenados os layouts XML que serão utilizados em nossas interfaces. Novamente clique com o botão direito na pasta **layout** e então em **New, Layout Resource File**. A tela de criação de layout será chamada.

Em File Name digite **activity_segunda**.

Em Root Element deixe o valor **LinearLayout** e então pressione Ok.

A janela de edição de layout será criada, e como podemos ver temos agora um layout XML que representa nossa Activity, e este está em branco.

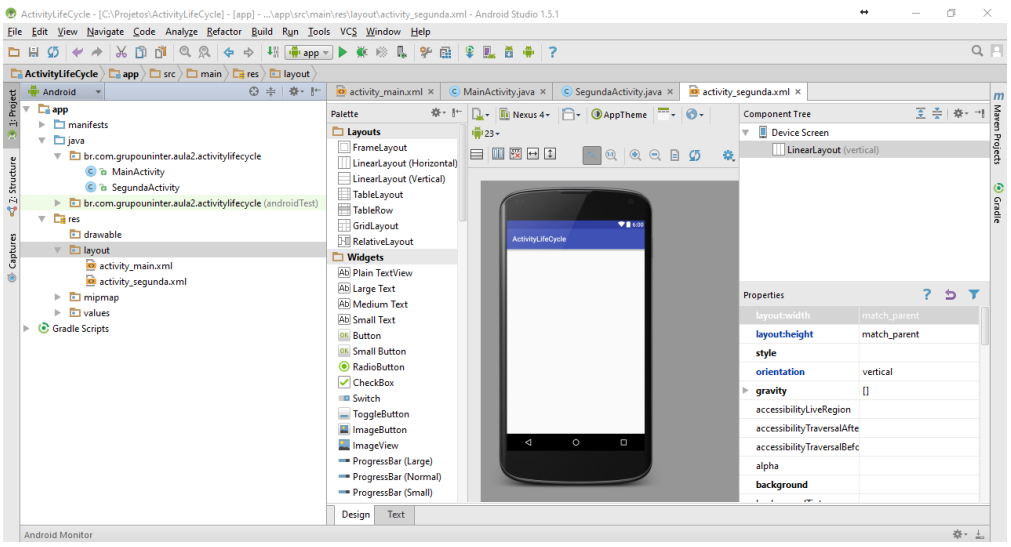

Figura 22 – Layout nova Activity

Por enquanto não vamos nos preocupar com isso, então volte para o arquivo **SegundaActivity.java**.

Dentro do método onCreate, logo após passarmos à classe base (Activity) o valor de nossa variável Bundle savedInstanceState, devemos informar que o layout utilizado para esta Activity será nosso arquivo activity_segunda.xml.

Isto é feito chamando-se o método setContentView() e passando como argumento para o mesmo o nome de nosso layout. No entanto, é importante frisarmos que não digitaremos ali o nome de um arquivo, mas sim a referência a este arquivo, que foi pré-compilado pelo Android Studio e armazenado na classe R.

```
package br.com.grupouninter.aula2.activitylifecycle;

import android.app.Activity;
import android.os.Bundle;

/**
 * Created by Marcelo on 31/07/2016.
 */
public class SegundaActivity extends Activity {

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

Pronto, sua classe foi agora corretamente estendida da classe base Activity, teve seu método onCreate () reformulado e até mesmo adicionamos à ela um layout próprio. E agora? Como você deve imaginar, executar este projeto agora não apresentará ao usuário final nenhuma diferença, já que não existe a chamada a SegundaActivity.

Vamos então analisar o fluxo de nosso aplicativo até o momento: quando o usuário o inicia, a MainActivity é chamada, e esta sobrescreve os métodos do ciclo de vida da Activity, mostrando então nossa interface com o usuário (layout activity_main.xml). Como ainda não sabemos como criar um objeto que responda ao evento dentro de uma Activity, vamos utilizar um destes métodos sobrescritos para chamar nossa Segunda Activity.

Abra o arquivo MainActivity.java e, olhando no ciclo de vida de uma Activity, qual método - onCreate(), onStart() ou onResume() - melhor se encaixa em nossa necessidade?

- O método **onCreate()** será executado somente uma vez, durante a criação da classe.

- O método **onStart()** será executado quando a aplicação iniciar, logo após do método **onCreate().** No entanto ele também será executado

caso a MainActivity entre no estado de segundo plano e esteja retornando para primeiro plano, logo após o método **onRestart().**

- Finalmente o método **onResume** será acionado na criação do aplicativo, quando este estiver voltando do estado do segundo plano e ainda quando este estiver saindo do estado de "pausado", ou seja, sua tela estava parcialmente visível (coberta por uma notificação, por exemplo).

Bom, desejamos para esse exemplo que o evento somente ocorra uma única vez, portanto utilizaremos o método **onCreate()**.

Necessitamos então instanciar o objeto Intent, passando a ele um context e também a classe que servirá como base para a nova Activity. Não se preocupe com a palavra context por enquanto, ela será coberta nas próximas aulas.

```
Intent intent = new Intent(this, SegundaActivity.class);
```

Agora temos o objeto de mensagens que carrega o tipo da classe que devemos instanciar e também quem é o responsável por ela (context).

Vamos agora instanciar a nova Activity, chamando o método startActivity(), que recebe uma Intent como seu argumento.

```
Intent intent = new Intent(this, SegundaActivity.class);
startActivity(intent);
```

Para deixar esse exemplo mais rico, antes de executarmos o código, vamos mudar a cor e fundo do layout de nossa segunda activity (activity_segunda.xml). Desta maneira teremos certeza que as Activities foram trocadas.

Selecione o layout activity_segunda.xml e, na janela de propriedades (Properties), vá até a propriedade background (cor de fundo) e digite a seguinte sequência: #c6c6c6

Este é o código hexadecimal para a cor cinza.

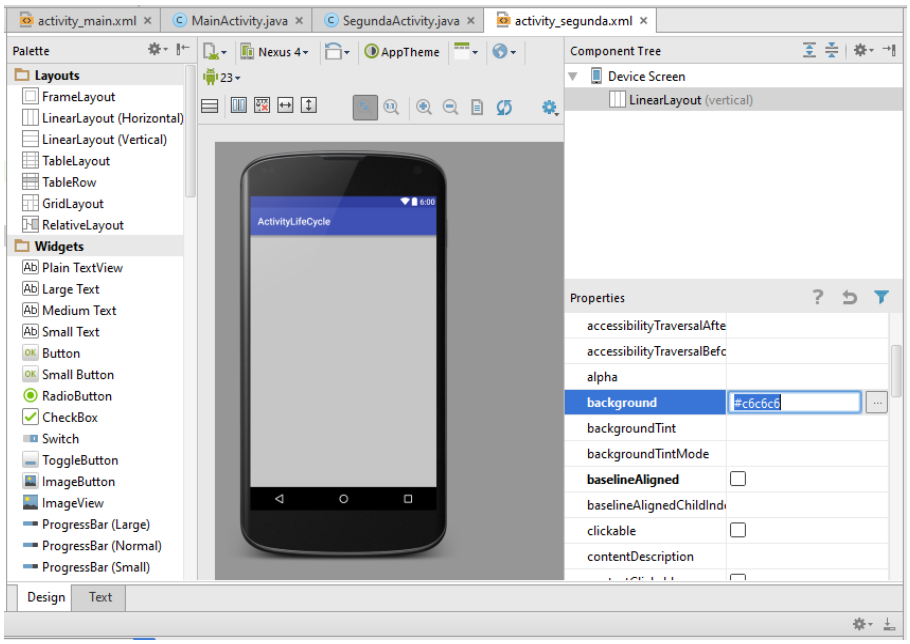

Figura 23 – Mudando a cor da Activity

Um último passo antes de podermos executar este exemplo é registrar a SegundaActivity em nosso AndroidManifest. Caso não façamos isso, o sistema operacional Android não será capaz de localizar nossa Activity, gerando uma mensagem de erro. Em nossa tela de Projects, expanda a pasta manifest e abra o arquivo AndroidManifest.xml.

Adicione ali o seguinte código, para registrar em nossa aplicação nossa SegundaActivity:

```
<activity android:name=".SegundaActivity">
</activity>
```

Aqui temos o código completo de nosso Android Manifest, para sua referência:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.grupouninter.aula2.activitylifecycle">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
        <activity android:name=".SegundaActivity">
        </activity>
    </application>

</manifest>
```

Pressione então Shift+F10 para iniciar o emulador, e verifique se a segunda Activity foi corretamente disparada.

Como você pode ver, o aplicativo foi executado com sucesso, e a tela cinza foi trazida para o primeiro plano, dando a impressão de que a SegundaActivity foi iniciada no lugar da MainActivity.

No entanto, se observarmos em nosso Debug, podemos entender facilmente que nossa aplicação saiu do estado “não existe” para “primeiro plano” (onCreate(), onStart(), onResume()) e então foi enviada para o segundo plano (onPause(), onStop()).

Pressione agora o botão retornar de seu emulador Android e veja quais mensagens seu Debugger captura.

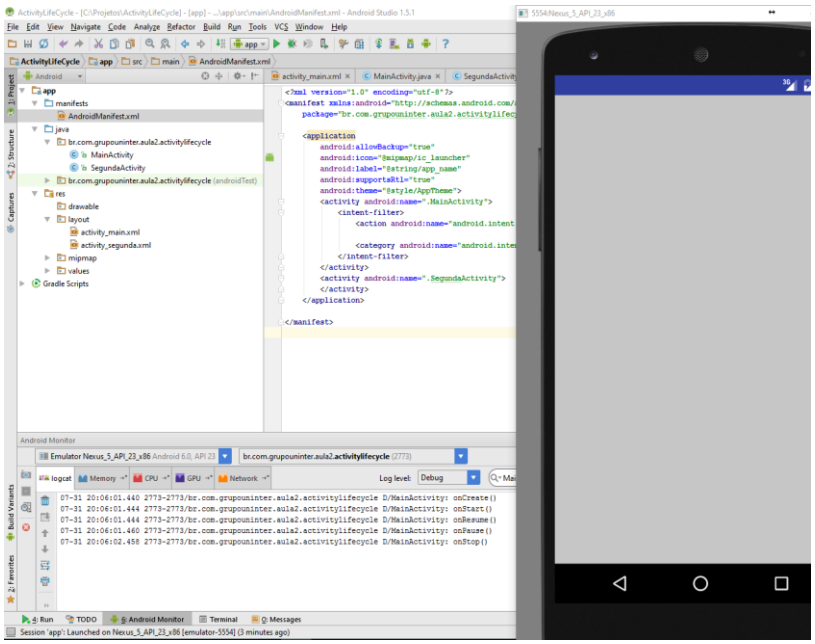

Figura 24 – SegundaActivity em execução

Acredito que agora o ciclo de vida de uma Activity não seja mais um mistério para você, inclusive porque você já está tomando decisões em cima do próprio ciclo. No entanto, este aplicativo seria muito mais interessante se fornecêssemos ao nosso usuário o poder de decidir em que momento a SegundaActivity deveria ser chamada, não acha?

No material online, a seguir, o professor Marcelo fala sobre isso, além dos principais aspectos da Intent.

Tema 4: Button

Agora que temos entendimento sobre como funciona o ciclo de vida das Activities, também entendemos a necessidade de criação de um objeto de mensagem para que possamos executar uma segunda Activity. Por isso vamos adicionar na MainActivity um botão para disparar a criação da SegundaActivity. Para isso, nosso primeiro passo é modificar o layout, fornecendo um botão a nossa interface.

- No painel de Projeto, expanda a pasta res, layout e abra o arquivo activity_main.
- Na Paleta de componentes (Pallete) localize o componente Button, e o arraste para dentro de nosso layout.

Pronto, agora temos um botão no centro do layout, mas infelizmente o mesmo não está atrelado a nenhuma ação. Mesmo que você execute agora o aplicativo, nada acontecerá quando o usuário pressionar este botão.

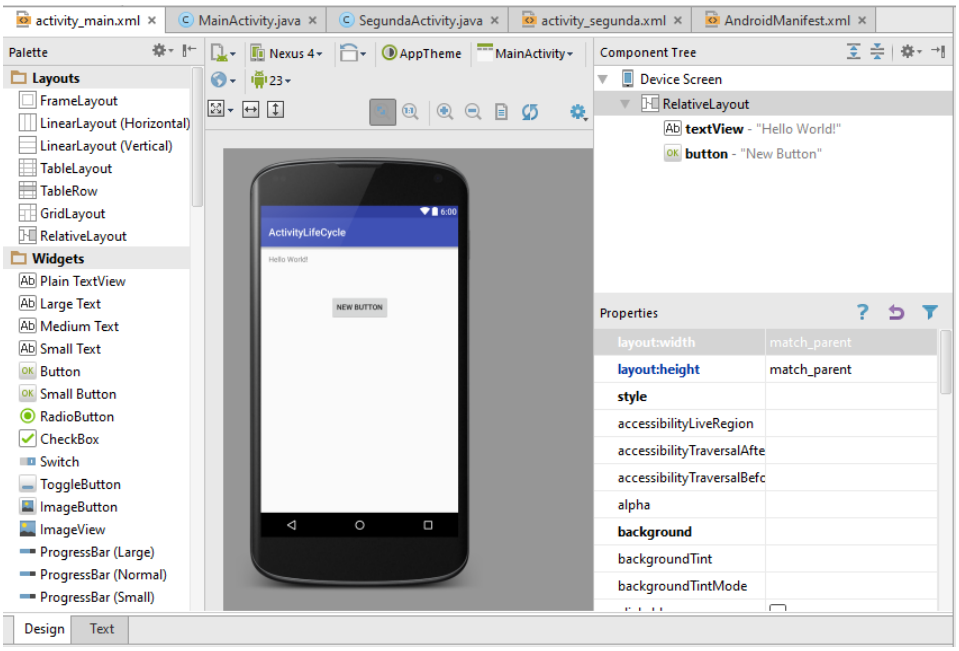

Figura 25 – MainActivity com um botão

Com o layout pronto, podemos agora criar os objetos que gerenciarão o comportamento deste botão.

Como já sabemos, uma Activity é composta de um layout e sua classe Java correspondente. Em nosso layout, já adicionamos o componente gráfico do botão. Agora precisamos criar dentro de MainActivity o objeto que corresponde a seu comportamento. Neste caso, o objeto **Button**.

A ligação entre o objeto Button e o xml é feita através de uma referência (assim como foi feita a referência ao layout), porém informando ao processador que esta referência deve comportar-se como um objeto Button.

```
Button btnSegundaActivity = (Button) findViewById(R.id.button);
```

O comando findViewById se encarrega em localizar o componente mapeado na classe R. Com esta referência pronta, podemos finalmente estender as funcionalidades do objeto Button btnSegundaActivity para que este execute a chamada à SegundaActivity. Isto é feito através do método Button.setOnClickListener (). Este método cria um evento que será disparado todas as vezes que este botão for pressionado. No argumento do método devemos passar o evento desejado, no nosso caso um novo evento OnClickListener().

```
btnSegundaActivity.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) {

    }
});
```

Agora vamos mover a criação do objeto Intent e o método startActivity para dentro desse evento, garantindo assim que os mesmos somente sejam disparados quando o usuário pressionar o botão.

```
btnSegundaActivity.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(this, SegundaActivity.class);
        startActivity(intent);
    }
});
```

Aos movermos, no entanto, somos informados que nosso context “this” não pode mais ser acessado, e que devemos providenciar um novo para que o projeto possa ser executado com sucesso.

O argumento do método onClick que acabamos de criar traz exatamente isso: uma instância da classe View que nos mostra quem tem o controle daquele objeto. Tudo que precisamos é utilizá-la, solicitando que ela nos retorne seu contexto, ou seja: v.getContext();

```
btnSegundaActivity.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(v.getContext(),
SegundaActivity.class);
        startActivity(intent);
    }
});
```

Aqui está o código completo de nosso MainActivity, para sua referência:

```
package br.com.grupouninter.aula2.activitylifecycle;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    private static String ACTIVITY = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(ACTIVITY, "onCreate()");

        Button btnSegundaActivity = (Button)
findViewById(R.id.button);

        btnSegundaActivity.setOnClickListener(new
View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(v.getContext(),
SegundaActivity.class);
                startActivity(intent);
            }
        });


    }
```

```
@Override
protected void onStop(){
    super.onStop();
    Log.d(ACTIVITY, "onStop()");
}

@Override
protected void onRestart(){
    super.onRestart();
    Log.d(ACTIVITY, "onRestart()");
}

@Override
protected void onDestroy(){
    super.onDestroy();
    Log.d(ACTIVITY, "onDestroy()");
```

Ao executarmos esta aplicação (Shift+F10) percebemos que agora a MainActivity voltou a ser a primeira tela a ser apresentada ao usuário, que a mesma possui um botão e que, ao se pressionar este botão a SegundaActivity é disparada. Também observe em nosso Debugger as mensagens que vão sendo disparadas.

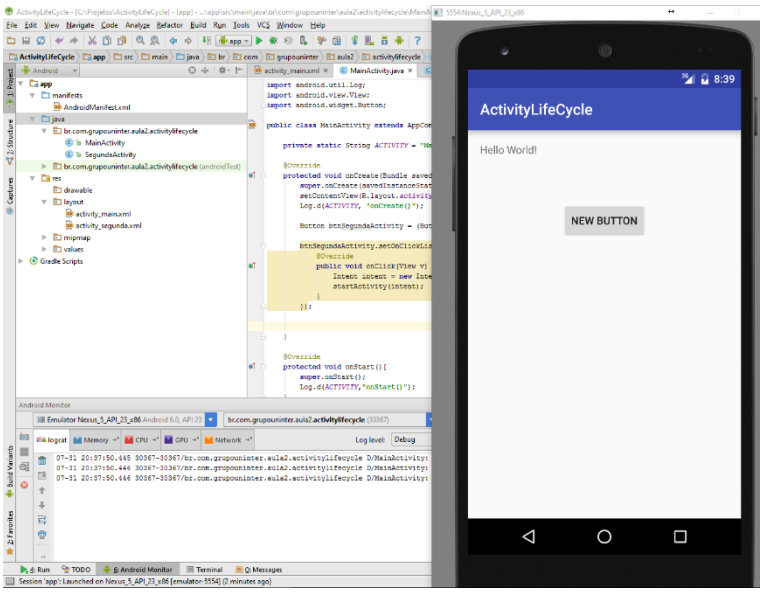

Figura 26 – MainActivity em execuçã

No material online, o professor Marcelo fala mais sobre o Button. Confira!

Síntese

Nesta aula foram apresentados os conceitos de Activities, Ciclo de Vida de uma Activity, Intents e Buttons. O objetivo é que você esteja apto para criar Activities, estender sua funcionalidade e chamar a novas Activities durante a execução do aplicativo.

Confira a síntese do professor Marcelo sobre os conteúdos abordados no material online.

Referências

ANDROID. **Intenções e filtros de intenções**. Disponível em: <https://developer.android.com/guide/components/intents-filters.html>. Acessado em: 12 ago. 2016.

DEITEL, Paul; DEITEL, Harvey; OSWALD, Alexander. **Android 6 para Programadores** – Uma Abordagem Baseada em Aplicativos. Boockman: 2015.