03

Android Olá Mundo

Bem-vindo ao curso de Android do alura

Nesse curso, utilizamos o eclipse com o plugin do ADT, este que não é mais recomendado pelo Google, então recomendamos esse curso apenas se você está envolvido com algum projeto legado. Se você está começando com o android recomendamos começar com esse curso : android-studio (http://www.alura.com.br/course/android-studio-do-zero-a-persistencia)

Atualmente, o poder de processamento dos celulares e outros aparelhos móveis salta à vista. Com esse avanço, também há sensíveis melhorias nas telas e na usabilidade desses dispositivos.

Imagine o quão elegante seria, nesse cenário, poder mostrar seu portfólio de aplicações já diretamente na sua mão, sem necessidade de sequer um laptop? Por essas e outras razões, plataformas móveis como iOS e Android, estão cada vez mais em evidência e o mercado para tais dispositivos está em plena ascensão. A explosão do Android é a mais recente dentre essas plataformas.

O Android é um sistema operacional que roda sobre o núcleo Linux. Desenvolvido pela *Open Handset Alliance*, a plataforma permite que os desenvolvedores escrevam software na linguagem Java controlando o dispositivo via bibliotecas desenvolvidas pela Google, com o objetivo de ser uma plataforma flexível, aberta e de fácil migração para os fabricantes.

O link para documentação, downloads e outros é o http://developer.android.com (http://developer.android.com)

Para iniciar com o desenvolvimento do Android, primeiramente baixaremos o Bundle Android para o nosso sistema operacional:

- Bundle para Linux (https://s3.amazonaws.com/caelum-online-public/FJ-57/adt-bundle-linux-x86_64-20140702.zip)
- Bundle para Mac (https://s3.amazonaws.com/caelum-online-public/FJ-57/adt-bundle-mac-x86_64-20140702.zip)
- Bundle para Windows (https://s3.amazonaws.com/caelum-online-public/FJ-57/adt-bundle-windows-x86_64-20140702.zip)

O Bundle vem com:

- *Software Development Kit* do Android (SDK): um conjunto de ferramentas para serem usadas no terminal
- Eclipse: IDE
- *Android Developer Tools* (ADT): um plugin que permite usar o SDK a partir do Eclipse

No download estará presente a última versão do SDK do Android, que pode ser diferente da vista no vídeo. Caso você deseje utilizar outras versões do Android (anteriores ou posteriores), basta seguir a seção abaixo.

(Opcional) Baixando SDKs

No Eclipse, vá em `Window -> Android SDK Manager` .

Na tela que abriu, podemos ver todos os SDKs disponíveis para download, além de outras coisas. Veja que o Eclipse pode ter marcado alguns arquivos para atualização, mas como só queremos baixar o SDK, podemos ir na opção abaixo nessa janelinha chamada *Deselect all*.

Agora, basta localizar nessa janelinha a pasta Android 4.4.2 (API 19), que é a versão usada no vídeo e marcar as seguintes subpastas:

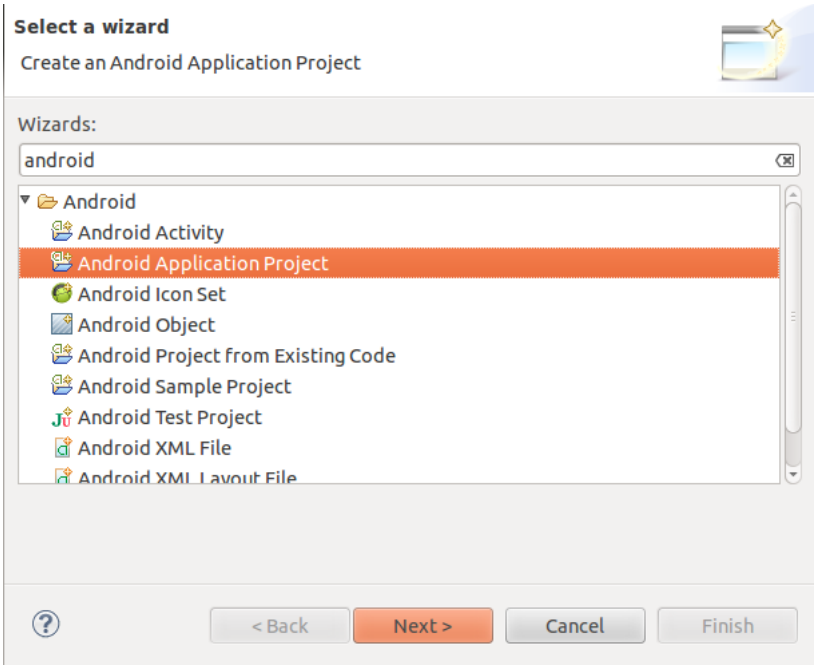
- *SDK Platform*
- *ARM EABI v7a System Image*
- *Intel x86 Atom System Image*
- *Google APIs (x86 System Image)*
- *Google APIs (ARM System Image)*

Com apenas esses pacotes marcados, clique no botão *Install 5 packages...* e proceda com a instalação normalmente.

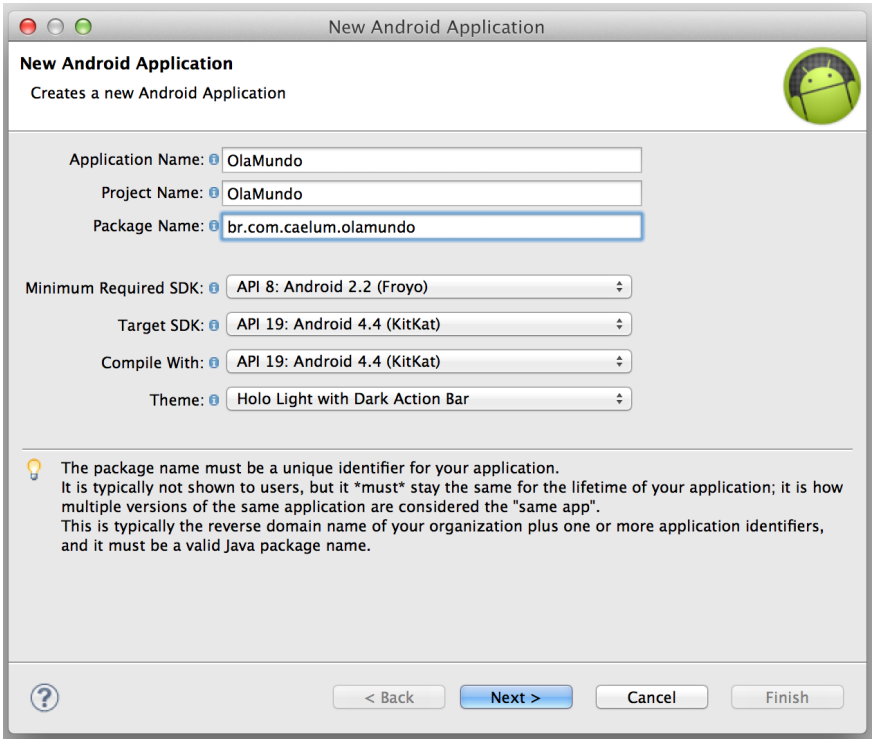
Criando nosso projeto

Começaremos com um primeiro projeto que mostra uma mensagem de boas vindas ao usuário. Criamos um projeto em branco, e preenchemos o nome da aplicação e pacotes de instalação.

No Eclipse, clique em `New -> Android Application Project`:

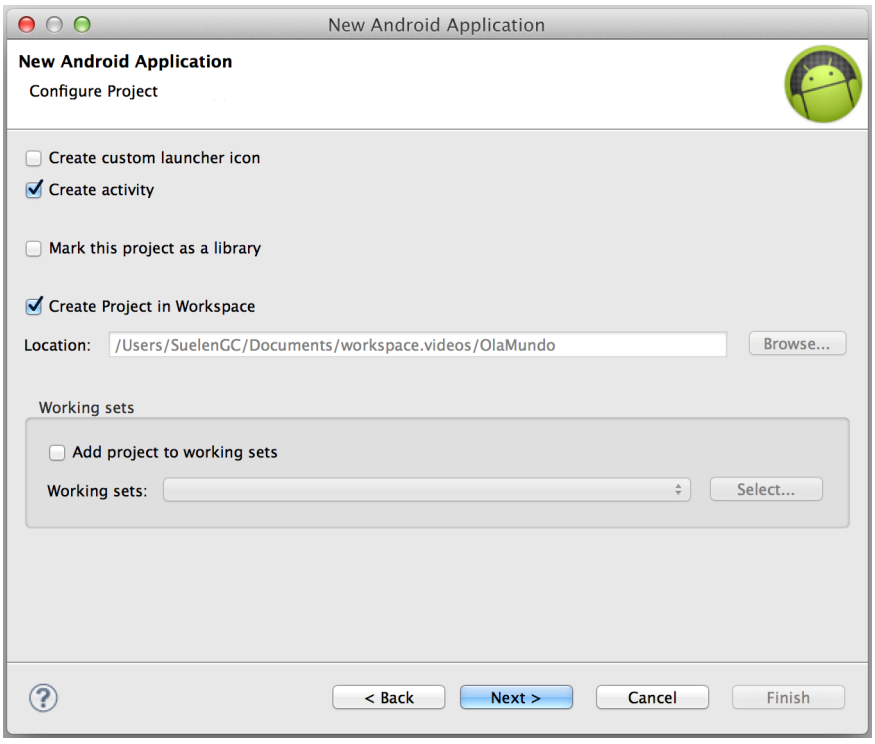


Preencha o nome da aplicação como sendo `OlaMundo` e o *package* como `br.com.caelum.olamundo` como na seguinte figura:



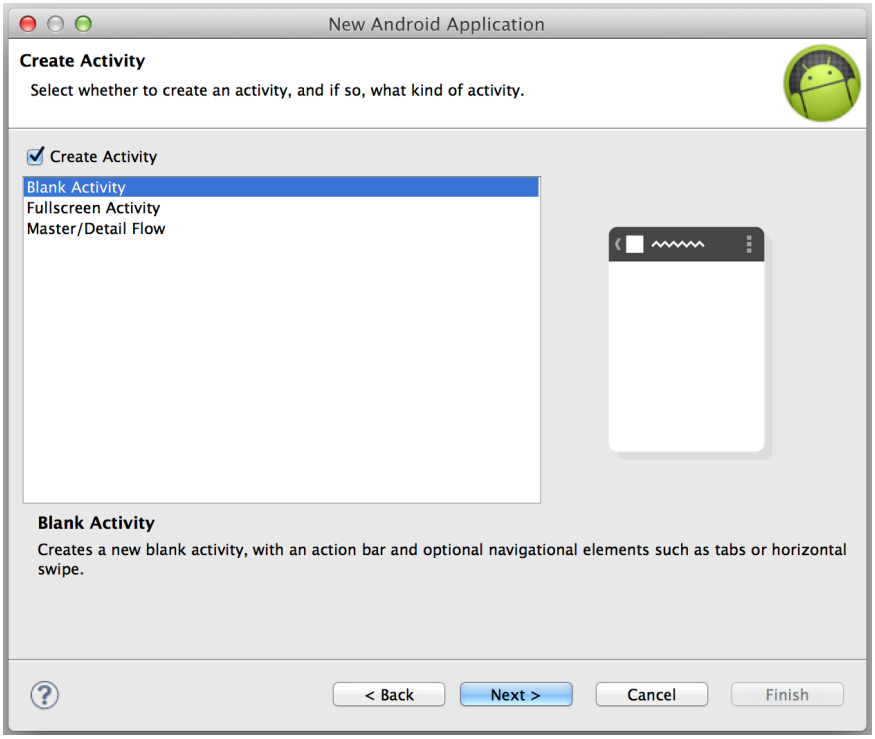
Clique em `Next` .

Na tela de Configuração do Projeto, diremos que não desejamos utilizar um ícone customizado, mas sim o padrão do Android. Desmarque a opção ***Create custom launcher icon***.

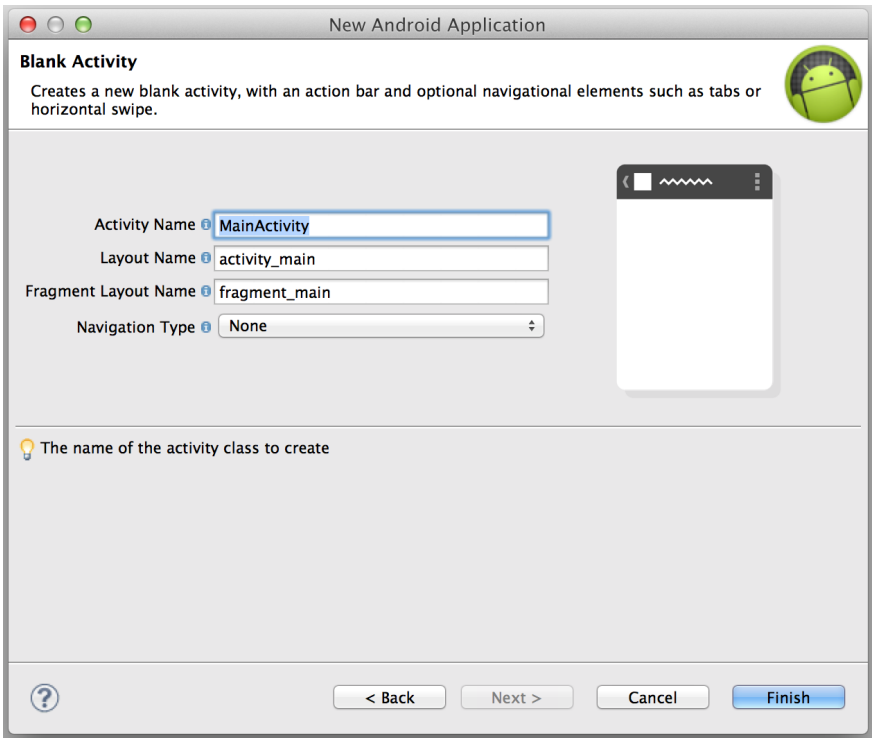


Clique em `Next` .

Criaremos agora nossa primeira tela de atividade do usuário com o sistema. A tela apresentada agora é a de criação da `Activity` , selecione a opção `Blank Activity` para começar a tela do zero, como na seguinte figura:



Vá para a próxima tela. Configuraremos os dados de nossa `Activity` como nome da classe e do *layout*. Por enquanto iremos aceitar as sugestões de nomes dada pelo *wizard*. Como na seguinte figura:



Apesar de não termos alterado os nomes sugeridos, isso poderia ser feito, os nomes não precisam seguir nenhum padão específico, faremos algumas customizações mais adiante no curso. Agora clique em *Finish*.

Caso você tenha escolhido *Blank Activity* e não apareceu a opção para digitar o *Fragment Layout Name*, não se preocupe pois criaremos esse tal de `Fragment` mais adiante. Neste caso, **não é necessário** fazer a seção abaixo ("Simplificando nosso código"), pois ele já estará como o esperado.

Simplificando nosso código

Em **algumas versões** do ADT (*Android Developer Tools*) o *template* de um aplicativo simples com apenas uma Activity inicial pode ser bem complexo, se utilizando de classes como `Fragment` e outras que aprenderemos mais adiante nos cursos de Android. O que faremos agora é simplificar nosso código antes de o

evoluirmos para começarmos com o mais simples possível.

Na pasta `src`, vá na classe `br.com.caelum.olamundo.MainActivity` e apague todo o código após o fim do método `onCreate` com exceção ao último símbolo de fechar chaves `"}"` pois este fecha a classe. Dentro do método `onCreate` há um `if`, pode apagá-lo.

Altere também a classe que `MainActivity` herda para `Activity` do pacote `android.app.Activity`. Após estas simplificações, sua classe `MainActivity` deve estar da seguinte forma:

```
package br.com.caelum.olamundo;

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

public class MainActivity extends Activity {

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

Abra o arquivo `activity_main.xml` que se encontra na pasta `res/layout` e substitua todo o conteúdo dele pelo conteúdo do arquivo `fragment_main.xml` que se encontra na mesma pasta. Após isso, apague o arquivo `fragment_main.xml`.

Seu arquivo `activity_main.xml` deve ter o seguinte conteúdo:

```
<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="br.com.caelum.olamundo.MainActivity$PlaceholderFragment" >

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

</RelativeLayout>
```

Mesmo após simplificarmos, são muitos os arquivos gerados automaticamente quando criamos um projeto do Android. Para começarmos a entender o que acontece na nossa aplicação, vamos explorar um pouco a estrutura desse projeto.

Comece abrindo o arquivo `MainActivity.java` em `src`, no pacote `br.com.caelum.olamundo`. Você verá algo como:

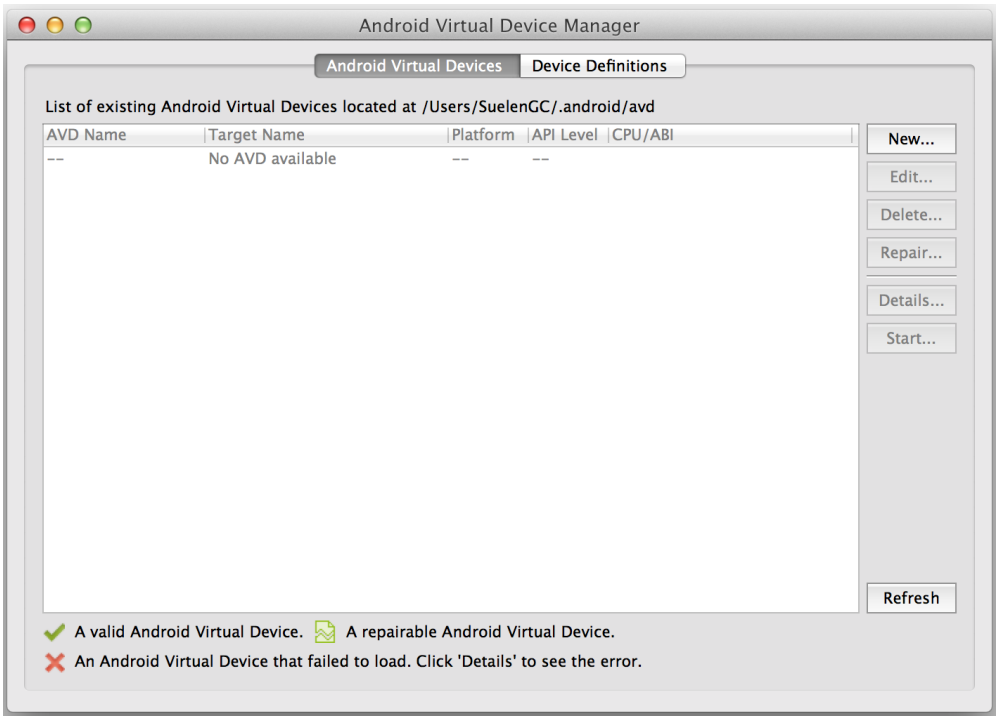
```
public class OlaMundoActivity extends Activity {

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

Note que essa classe herda de `Activity`, isso quer dizer que ela representa uma tela da nossa aplicação. É aqui que fica o código para interagirmos com o usuário.

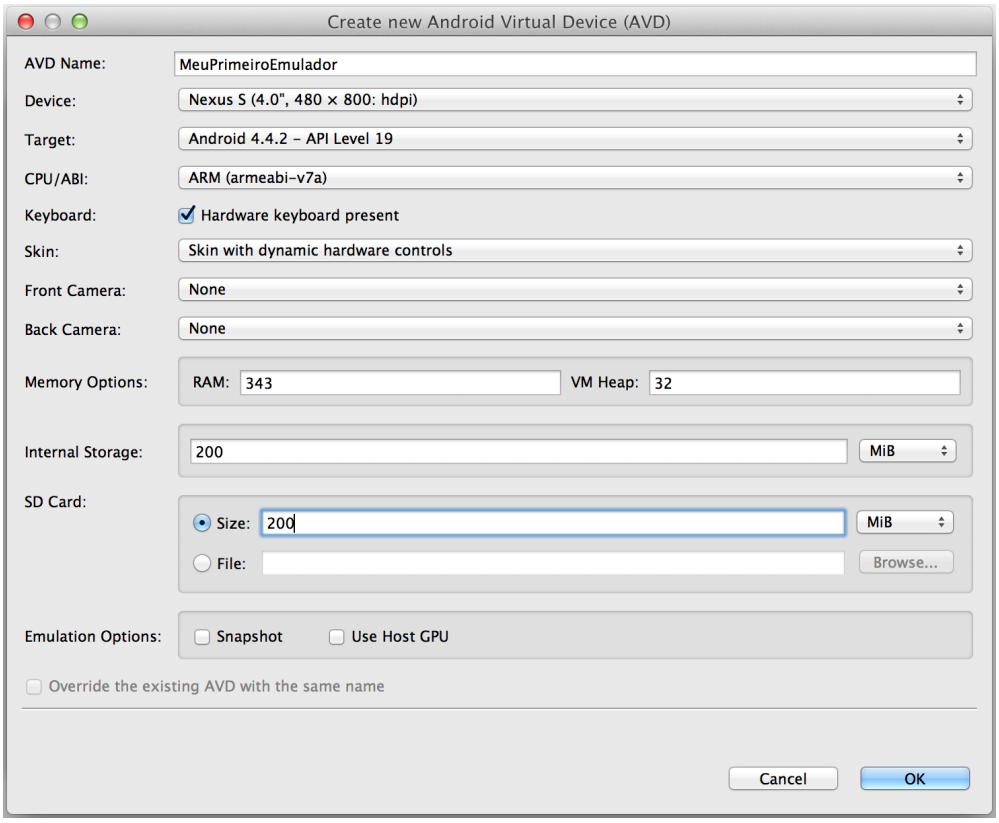
Emulando um dispositivo

Para executarmos nossa aplicação criaremos um dispositivo virtual - emularemos um aparelho real - que roda Android. Para isso basta escolhermos as especificações do aparelho que desejamos. O emulador é bastante útil porque nem sempre possuímos vários dispositivos com diferentes tamanhos de tela e versões do Android disponíveis para teste.

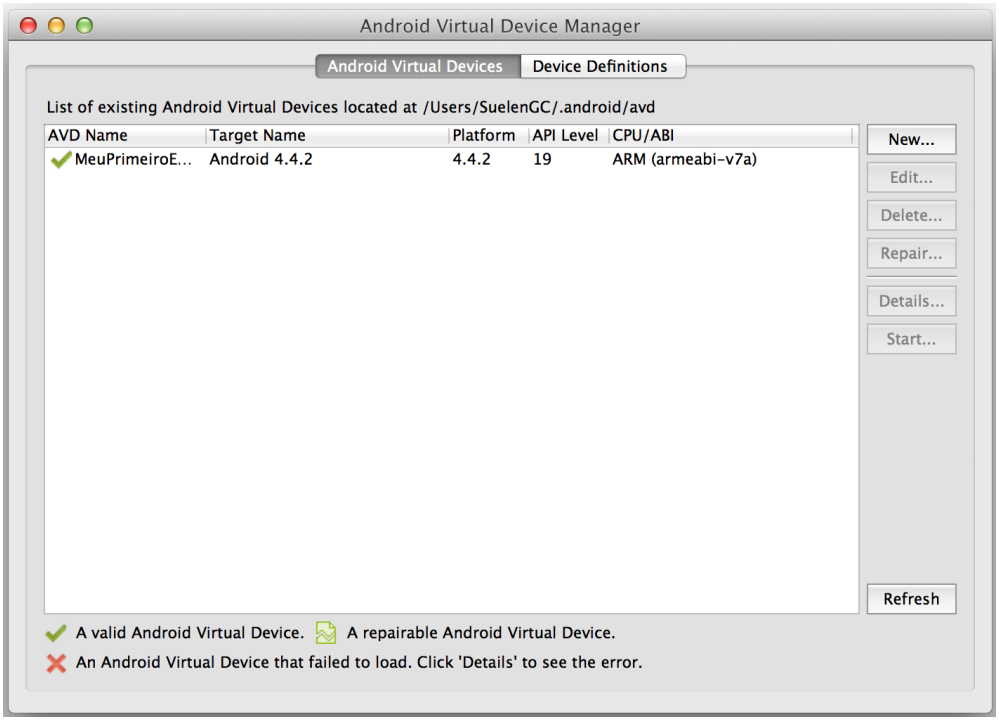
Para criar um novo emulador, clicamos no menu `Window -> Android Virtual Device Manager (AVD)`. Será aberta a janela a seguir:



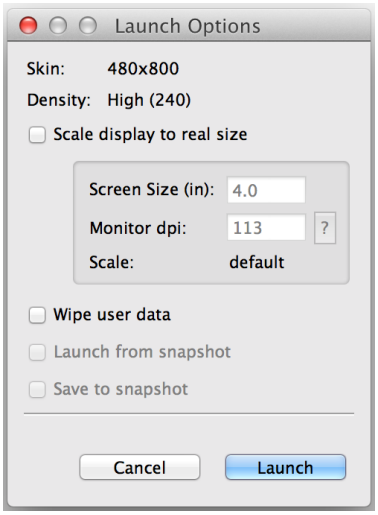
Na janela que foi aberta, clique sobre o botão `New`. Agora preencha as informações conforme a figura abaixo e após clique em `Ok`.



Após criar um novo emulador, selecione-o na lista de AVDs existentes e clique em `Start`.



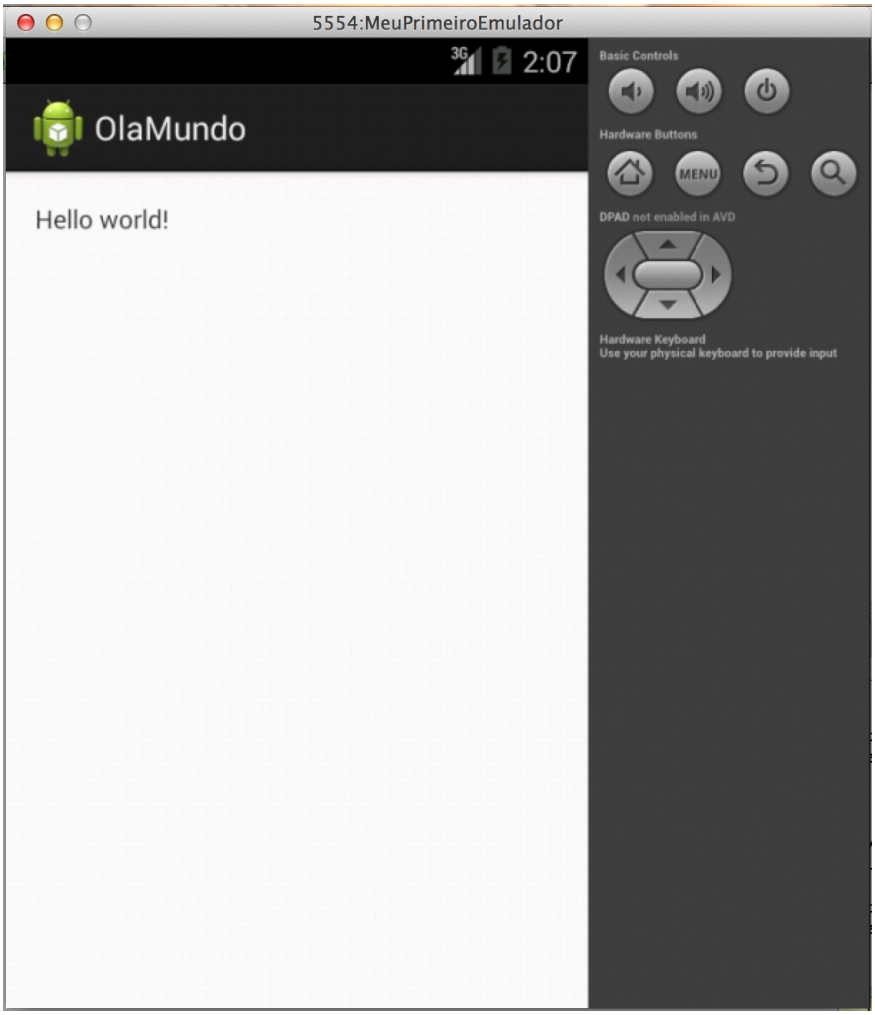
Na janela seguinte, clique em `Launch` .



Com o dispositivo criado, clique com o botão direito do mouse sobre o nome do projeto e selecione `Run As -> Android Application` .

Com isso teremos o *upload* da aplicação para o emulador e sua instalação, rodando a aplicação logo em seguida. Este processo pode ser acompanhado pela janela `Console` , para abrí-la digite `Ctrl+3 + CONSOLE` e clique em `ENTER` .

O emulador pode demorar bastante para inicializar, já que está realmente dando *boot* num Android para o teste. Mas, quando finalmente terminar, o resultado é a nossa aplicação:



Apesar da linguagem de alto nível que utilizamos para programar ser o Java, todo código do Android roda sobre uma máquina virtual chamada Dalvik ou, mais recentemente *Android Runtime* (ART), que utiliza um *bytecode* diferente.

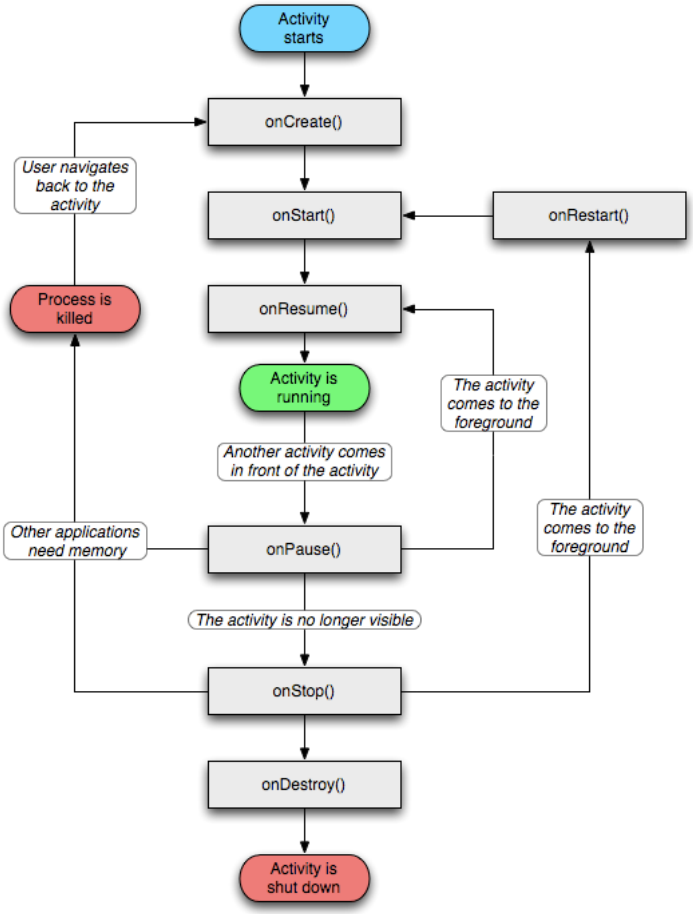
O que acontece é que o SDK do Android já vem munido de um tradutor do formato *bytecode* Java para o formato dex (Dalvik Executable) da máquina virtual, já fazendo uma série de otimizações focadas em reduzir o espaço do executável gerado.

Ciclo de vida da Activity

Para construirmos aplicações robustas é bem importante entendermos sobre o ciclo de vida da Activity no Android. A seguir, vamos entender em detalhe a ordem e quais são os métodos principais do ciclo de vida da `Activity`.

O método `onCreate` é chamado quando uma `Activity` é executada. As classes que estendem `Activity`, de uma forma mais abrangente, interagem tanto com usuários como com serviços.

Toda `Activity` tem o ciclo de vida representado na figura a seguir.



Por exemplo, quando nossa aplicação é criada, ocorre o evento `onCreate` e a classe `Activity` permite que sobrescrevamos ele para executar alguma tarefa especifica nossa, no entanto, temos que garantir que o `onCreate` da classe `Activity`, classe-mãe, seja chamado, fazemos isso chamando o `super.onCreate(...)` antes de fazermos nosso código específico. Na prática usamos esse instante para criar a nossa tela (a *view*).

Existem outros métodos, são eles: `onStart`, `onResume`, `onPause`, `onStop` e `onDestroy`.

Nosso layout

Os layouts são armazenados na pasta `res/layout`. Você pode conferir a existência do arquivo `activity_main.xml` em seu projeto:

```
<?xml version="1.0" encoding="utf-8"?>
<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="br.com.caelum.olamundo.MainActivity$PlaceholderFragment" >

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

</RelativeLayout>
```

Nesse exemplo, estamos declarando um *layout* relativo e avisando que, mesmo que não houver conteúdo o bastante para preencher a tela, a aplicação se esticará para ocupar toda a tela. Isso se deve aos atributos `android:layout_width` e `android:layout_height` com `match_parent`.

Já na tag `TextView`, note que a altura é determinada como o apenas necessário para mostrar seu conteúdo, configurada como `wrap_content`.

res/values/

Repare na linha `android:text="@string/hello_world` no layout acima que `@string/hello` define o texto que aparecerá na tela quando a aplicação rodar. Perceba que ela é apenas uma chave para algum texto. Esse texto está definido em **res/values**, dentro do arquivo `strings.xml` .

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World!</string>
    <string name="app_name">Curso</string>
</resources>
```

Essa é uma ideia simples que facilita na internacionalização da sua aplicação. Para traduzí-la para outros idiomas, basta criar outros arquivos de `strings.xml` e colocá-los nos devidos diretórios **values**, por exemplo, se for um `strings.xml` inglês deve ficar em **values-en**. Além disso, isolar as strings do sistema, também potencializa a clareza de alguns pontos do código como por exemplo, chamadas ao banco de dados do Android.

res/raw/

Esse é o local para armazenar arquivos de mídia que futuramente serão utilizados no sistema, como mp3, vídeos e outros.

res/xml/

Nessa pasta armazenaremos arquivos `xml` . Se ela não existir, é só criá-la na pasta `/res`

res/drawable/

Local para armazenar imagens, sejam `.gif` , `.jpg` ou `.png` .

Vale relembrar que as imagens da sua aplicação são guardadas em três ou mais resoluções, cada uma com sua pasta no sistema:

- `res/drawable-ldpi` : *low*, dispositivos de baixa resolução (quase inexistentes);
- `res/drawable-mdpi` : *medium*, dispositivos de média resolução (raros);
- `res/drawable-hdpi` : *high*, dispositivos com alta resolução (maioria).

Classe R.java

Existe também uma classe central que é automaticamente gerada e é responsável pelo mapeamento dos elementos da *view* (*resources*) com o *model* e o *controller* (códigos java), essa é a classe `R.java` que fica dentro do diretório `gen` (*generated*). Veja que tudo o que é criado ou declarado na pasta `res` ganha uma representação nesta classe, que será bastante usada para pegar valores e referenciar itens.

Note, no entanto, que essa classe não deve ser alterada manualmente. Ela é apenas um recurso que o Android disponibiliza para facilitar a referência a objetos visuais, imagens, strings, etc. Apesar disso, caso ocorra de alterá-la por engano, basta apagar a pasta `gen` que o Android reconstruirá da maneira correta.

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int caelum=0x7f020000;
        public static final int icon=0x7f020001;
    }
    public static final class id {
        public static final int FrameLayout01=0x7f060000;
        public static final int botao=0x7f060002;
        public static final int dest=0x7f060003;
        public static final int linearl=0x7f060001;
        public static final int send=0x7f060005;
        public static final int texto=0x7f060004;
    }
    public static final class layout {
        public static final int helloworld=0x7f030000;
        public static final int layout1=0x7f030001;
        public static final int main=0x7f030002;
        public static final int player=0x7f030003;
        public static final int smsexample=0x7f030004;
        public static final int widgets=0x7f030005;
    }
    public static final class raw {
        public static final int musica=0x7f040000;
```

```
        public static final int video=0x7f040001;
    }
    public static final class string {
        public static final int app_name=0x7f050001;
        public static final int hello=0x7f050000;
    }
}
```

Note que, dessa forma, das nossas classes java, para acessar qualquer recurso declarado na pasta `res`, fazemos uma referência aos atributos públicos dessa estranha classe `R`. Por exemplo:

Acesso a imagens:

- `R.drawable.icon`
- `R.drawable.caelum`

Acesso a itens da tela declarados nos `XMLs` de *layout*:

- `R.id.botao`
- `R.id.texto`

Acesso a arquivos:

- `R.raw.musica`
- `R.raw.video`

Acesso a strings:

- `R.string.app_name`
- `R.string.hello`