Dispositivos Móveis

Aula 09 - Estilos, Temas e Compatibilidade

Apresentação

Nesta aula, daremos continuidade ao estudo de interfaces gráficas em aplicações Android. Aprenderemos aqui os conceitos relacionados a temas e estilos dentro de um aplicativo, assim como veremos seu uso através de exemplos. Estudaremos, também, sobre compatibilidade das aplicações em meio à grande gama de configurações de tela existentes no mercado.

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Entender os conceitos e a implementação de estilos e temas.
- Avaliar a importância de um aplicativo que implementa compatibilidade de forma correta.
- Implementar soluções para os possíveis problemas de compatibilidade.

Estilos

Independentemente da plataforma para que desenvolvemos, parte importante do processo é a definição da aparência do nosso produto, a fim de criar uma identidade visual robusta. Com o Android não é diferente e, para facilitar esse processo, é usado o conceito de estilo, muito semelhante ao CSS (Cascade StyleSheet) em aplicações web.

Um estilo, então, é definido como uma coleção de propriedades utilizadas para definir a aparência e formatação das *Views*. Nele, podem ser definidas diversas propriedades diferentes (largura, cor de fonte, margens, entre muitas outras) que serão aplicadas aos elementos escolhidos para utilizá-lo. Um estilo garante, além de fácil acesso e utilização, um ótimo controle de mudanças na parte visual do aplicativo, uma vez que, quando usado corretamente, centraliza as propriedades que definirão a estilização dos componentes.

Diferentemente da formatação *inline* (na qual cada propriedade é inserida juntamente com a declaração do componente no layout), os estilos são definidos em arquivos XML e podem ser utilizados em quantos layouts forem necessários.

Para melhor entendimento, observe as **Listagens 1** e **2** apresentadas a seguir.

```
1 <TextView
2   android:layout_width="fill_parent"
3   android:layout_height="wrap_content"
4   android:textColor="#00FF00"
5   android:typeface="monospace"
6   android:text="@string/hello" />
```

Listagem 1– Definição *inline*

```
1 <TextView
2   style="@style/CodeFont"
3   android:text="@string/hello" />
```

Listagem 2 – Definição usando estilo

Como é possível observar, as diferenças de implementação apresentadas entre as **Listagens 1** e **2** são todas relativas às propriedades de definição do estilo do componente. Na **Listagem 1**, são declaradas as propriedades necessárias ao *TextView* em 4 linhas, enquanto na **Listagem 2**, basta uma linha para se atingir o mesmo resultado.

Para definir um estilo, primeiro deve-se criar um arquivo de extensão ".xml" na pasta *res/values* e declarar como seu elemento principal a tag *<resources>*. Tendo criado o arquivo e sua estrutura, basta declarar uma tag *<style>* para cada estilo que se deseja criar. Cada estilo deve definir um nome (propriedade *name*), que será seu identificador único e utilizado como referência na hora de aplicar um estilo a um determinado componente. Além do nome, cada estilo pode definir diversas propriedades diferentes, e cada uma delas é armazenada em uma tag *<item>*, que por sua vez deve possuir o nome da propriedade que se deseja declarar e seu valor.

Seguindo, ainda, o exemplo apresentado nas **Listagens 1** e **2**, a implementação do arquivo de estilo para manter as configurações do componente após seu uso é apresentada na **Listagem 3**.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3    <style name="CodeFont">
4       <item name="android:layout_width">fill_parent</item>
5       <item name="android:layout_height">wrap_content</item>
6       <item name="android:textColor">#00FF00</item>
7       <item name="android:typeface">monospace</item>
8    </style>
9 </resources>
```

Listagem 3 – Arquivo XML de estilos

Além de definir estilos completamente novos, podemos utilizar os conceitos de herança na hora de sua definição, garantindo que todas as propriedades já definidas em um estilo possam ser utilizadas ou reescritas. Para tal, devemos nos atentar a qual estilo queremos herdar.

Se o estilo que nós queremos utilizar é um dos estilos padrão definidos pelo Android, nós utilizaremos a propriedade *parent* na definição do estilo, conforme visto na **Listagem 4**.

```
<style name="GreenText" parent="@android:style/TextAppearance">
   <item name="android:textColor">#00FF00</item>
</style>
```

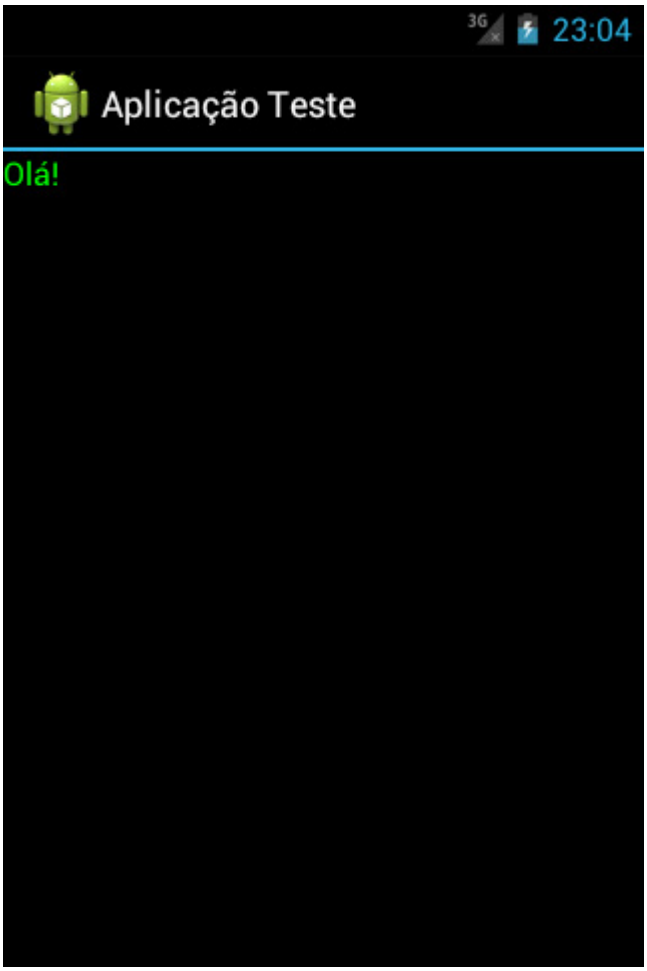
Listagem 4 – Estilo herdando de estilo do Android

Conforme visto na definição da **Listagem 4**, criamos um estilo chamado *GreenText*, que possui como pai o estilo *TextAppearance*, da plataforma. Porém, é definida a propriedade *textColor* com o valor *#00FF00*, que é o código hexadecimal da cor verde. Com isso, ao usar o estilo *GreenText*, teremos todas as propriedades definidas em *TextAppearance* com a diferença da cor do texto ser verde. Tal comportamento pode ser visto na **Listagem 5** e na **Figura 1** abaixo.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello"
      style="@style/GreenText" />
</LinearLayout>
```

Listagem 5 – Implementação do layout usando o estilo anteriormente definido

Figura 01 - Resultado da implementação da Listagem 5.



Para os casos nos quais desejamos herdar estilos definidos na própria aplicação, não utilizamos a propriedade *parent*, mas uma nova forma de declaração do nome do nosso estilo. Essa nova forma consiste em adicionar o nome do estilo pai como prefixo do novo estilo, separando-se os nomes com um "." (ponto). Como exemplo, observe a **Listagem 6** a seguir.

```
1 <style name="GreenText.Big">
2   <item name="android:textSize">30sp</item>
3 </style>
```

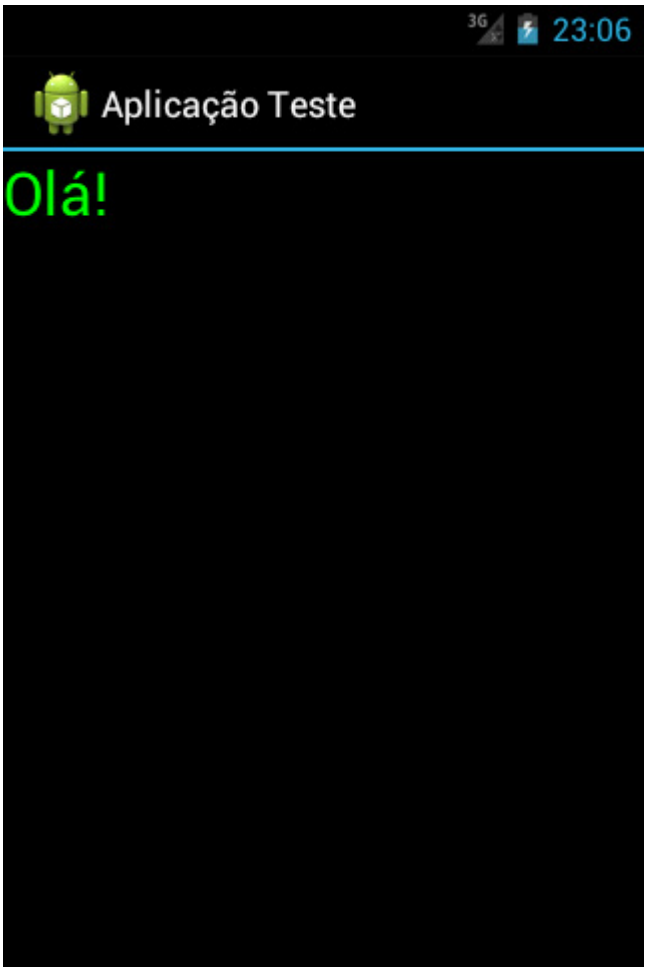
Listagem 6 – Estilo herdando de estilo próprio

Ao utilizar esse estilo, usaremos o mesmo nome indicado na sua definição, no caso *GreenText.Big*, e o comportamento esperado com seu uso é a verificação de todas as propriedades definidas em *GreenText* (inclusive as herdadas de *TextAppearance*) além do novo tamanho de fonte definido no novo estilo. Dessa forma, o uso desse novo estilo ficaria conforme é mostrado na **Listagem 7** e na **Figura 2**.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello"
      style="@style/GreenText.Big" />
</LinearLayout>
```

Listagem 7 – Implementação do layout usando o estilo com parent anteriormente definido

Figura 02 - Resultado da implementação da Listagem 7.



Conforme observado na **Figura 2**, todos os comportamentos anteriormente definidos no estilo *GreenText* foram mantidos, assim como a nova propriedade (referente ao tamanho da fonte) foi adicionada.

Um ponto importante a se notar ao utilizar estilos é que só serão aplicadas à *View* associada aquelas propriedades existentes na definição da *View*. Dessa forma, no processo de definição de estilos para um *TextView*, deve-se atentar àquelas propriedades suportadas por ele, que, no caso, podem ser verificadas em http://developer.android.com/reference/android/widget/TextView.html#lattrs. Dessa forma, qualquer atributo listado que seria utilizado como uma propriedade na definição de um *TextView* poderá ser definido como um item dentro de um estilo.

Temas

Juntamente com o conceito de estilos, estudaremos os temas, que podem ser definidos como estilos aplicados diretamente a uma *Activity* ou aplicação. Pode-se definir como um tema, qualquer estilo definido, e todas as propriedades definidas serão aplicadas à *View* ou aplicação, conforme definido no *AndroidManifest*.

Porém, existem certas propriedades que podem ser definidas dentro de uma tag *style* que são de uso exclusivo para temas. Tais propriedades não podem ser aplicadas diretamente sobre uma *View*, mas apenas utilizadas como temas. Isso acontece porque essas propriedades são aplicadas à janela como um todo, e não apenas a uma determinada *View*. Esses atributos, usados especificamente para temas, podem ser vistos em http://developer.android.com/reference/android/R.attr.html e são facilmente identificáveis por começar com o prefixo "window".

Diferentemente dos estilos, o uso de um tema, deve ser definido no arquivo *AndroidManifest* da aplicação. Aplicar um tema a uma *Activity* é o mesmo que definir o estilo utilizado para todas as *Views* presentes naquela *Activity*, enquanto que aplicar um tema a uma aplicação é o mesmo que aplicá-lo em todas as *Activities* dela.

O uso de temas pode ser melhor observado nas **Listagens 8** e **9**, logo a seguir.

```
1 <activity android:theme="@style/Tema">
```

Listagem 8 – Definição de tema em uma *Activity*

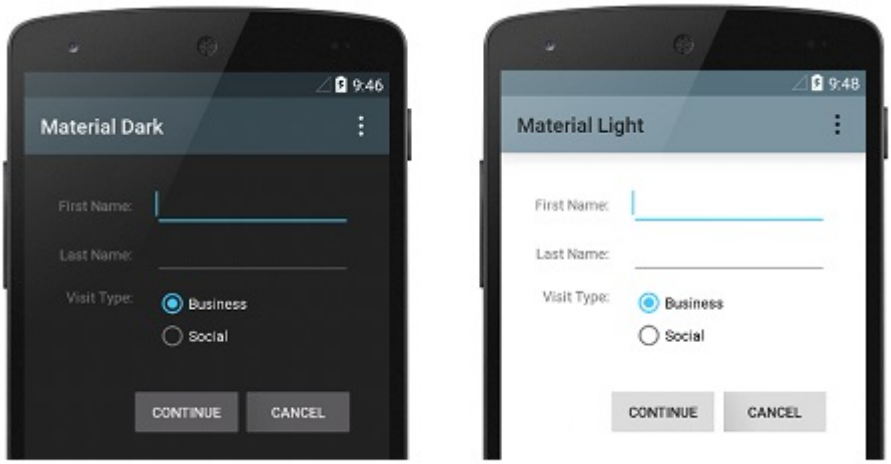
```
1 <application android:theme="@style/Tema">
```

Listagem 9 – Definição de tema na aplicação

Como a estrutura de temas segue a mesma dos estilos, é possível alterar os temas já existentes na plataforma, criando um tema que herde de um previamente existente, assim como a utilização direta de temas já existentes no Android.

O Android nos oferece algumas opções de temas como o *dark material theme* e o *light material theme*, que podemos ver na **Figura 3**. Podemos também criar nossos próprios temas através de sites que nos ajudam nesta tarefa, mostrando a aparência final dos temas criados.

Figura 03 - dark material theme (esquerda), light material theme (direita).



Vídeo 02 - Praticando Temas e Estilos Avançados

Atividade 01

1. Destaque semelhanças e diferenças entre temas e estilos.

Compatibilidade

Conforme já mostrado nas aulas anteriores, o Android é uma plataforma que trabalha com um grande número de aparelhos. Isso traz diversas configurações diferentes de aparelhos possíveis, e, ao desenvolver um aplicativo, é importante notar que quanto mais aparelhos conseguirmos atingir com o aplicativo, melhor será.

Por isso, o Android oferece uma série de mecanismos para facilitar o trabalho com compatibilidade. É possível, por exemplo, definir que alguns usuários, dependendo das funcionalidades e recursos disponíveis em seus aparelhos, nem mesmo vejam o seu aplicativo na Google Play e não possam, portanto, realizar sua instalação.

Cada aparelho que usa Android tem instalado uma determinada versão do sistema. A versão determina, entre outras coisas, o nível de API que o aparelho tem acesso por padrão e, portanto, quais recursos estão disponíveis a ele. É importante, também, salientar que o uso das APIs é independente do aparelho, ou seja, não existem seções opcionais das APIs. Dessa forma, definindo um nível de API mínimo utilizado pela aplicação, temos a certeza de seu correto funcionamento em qualquer aparelho que utilize essa API.

Apesar disso, podem existir recursos de hardware que limitem o funcionamento correto de determinadas partes da API. Felizmente, da mesma forma que podemos limitar o acesso a nossos aplicativos a partir dos níveis de API, podemos fazer algo semelhante com recursos de hardware.

Para limitar o acesso e instalação de sua aplicação, usamos, basicamente, duas abordagens. A primeira é definindo os recursos de software que utilizaremos, através das tags de SdkVersion, definidas no build.gradle, dentro dos Gradle Scripts de sua aplicação. Através dessas tags, indicaremos qual API utilizamos para desenvolver o aplicativo e quais os limites de API que desejamos liberar a aplicação, através das propriedades *minSdkVersion*, *targetSdkVersion* e *maxSdkVersion*. Tal comportamento pode ser melhor observado na **Listagem 10** a seguir.

```
defaultConfig {
    applicationId "br.ufrn.imd.myapplication"
    minSdkVersion 16
    targetSdkVersion 23
    versionCode 1
    versionName "1.0"
}
```

Listagem 10 – Declaração de uso de APIs no arquivo build.gradle

Por outro lado, se necessitarmos limitar o acesso a nosso aplicativo devido a recursos de hardware, devemos utilizar a tag *uses-feature*. Diferentemente da verificações de versão do SDK, que são feitas no momento de instalação do aplicativo, a declaração de elementos através da *uses-feature* é apenas informacional, sendo utilizada para verificações antes da instalação do aplicativo pela Play Store.

Um exemplo de declaração de uso de um recurso pode ser melhor observado na **Listagem 11** abaixo.

```
<uses-feature android:name="android.hardware.camera" />
```

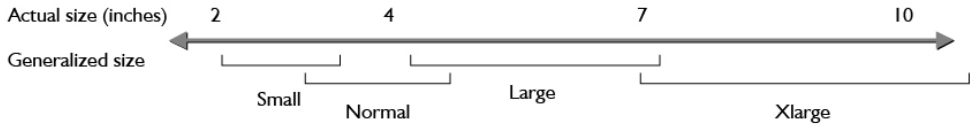
Listagem 11 – Declaração de uso de recurso de câmera no *AndroidManifest*

Além das grandes diferenças possíveis entre recursos de hardware de um modo geral, outra grande preocupação que devemos ter ao desenvolver uma aplicação é sobre seu funcionamento em diferentes configurações de tela. Para isso, devemos levar em consideração alguns conceitos que aprenderemos agora: tamanho, densidade, orientação e resolução da tela.

Vídeo 03 - Compatibilidade com Diferentes Telas

O tamanho da tela é a medida física dela. Essa medida é feita diagonalmente e, geralmente, é expressa em polegadas. Para facilitar os possíveis tratamentos, o Android faz uma separação de acordo com os tamanhos de tela, conforme observado na **Figura 4**.

Figura 04 - Separação de telas por tamanho (em polegadas).



A densidade da tela faz referência à quantidade de pixels que consegue ser armazenada em uma determinada região da tela e geralmente é expressada em dpi (*dots per inch* – pontos por polegada). O Android também provê uma separação de configurações de tela de acordo com suas densidades de tela. Tal separação pode ser observada na **Figura 5**.

Figura 05 - Separação de telas por densidade (em dpi).

Juntamente com o conceito de densidade, temos o de pixel independente de densidade, o *density-independent pixel* (dp), que é uma unidade virtual utilizada quando desejamos definir nossos layouts com medidas independentes da densidade da tela trabalhada. A medida de 1 dp equivale à medida de um pixel físico, mostrado em uma tela com densidade de 160dpi (que é a densidade utilizada pelo sistema como base para telas com densidade média).

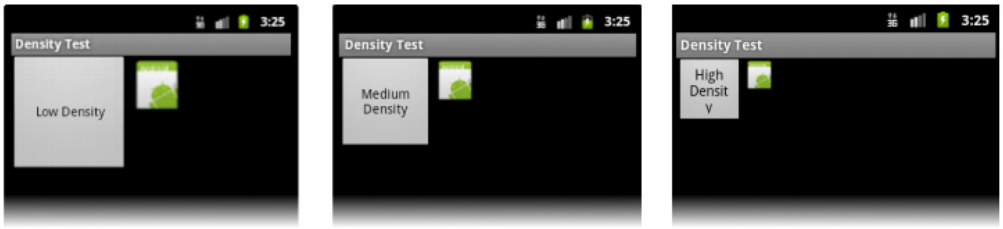
Ainda relacionado à densidade da tela, temos mais uma medida, dessa vez definida para utilização com trabalhos de fontes. Essa medida é o pixel independente de escala (*scale-independent pixel* – sp), que trabalha de forma semelhante ao dp, com a diferença de levar em consideração as configurações de fontes definidas pelo usuário.

Outro importante conceito que deve ser cuidadosamente trabalhado em uma aplicação é a orientação da tela, que é como a tela está disposta ao usuário naquele momento. A disposição é separada em duas: horizontal (*landscape*) ou vertical (*portrait*). Para cada uma dessas disposições, pode ser interessante apresentar o conteúdo de forma diferente ao usuário, uma vez que a tela fica mais alta ou larga de acordo com sua disposição.

Uma medida muito vista nos detalhes de qualquer aparelho é a resolução da sua tela. Essa medida faz referência ao total de pixels armazenados na tela. É importante notar que, ao trabalhar a fim de suportar múltiplas telas, esse tipo de medida não deve ser levado em consideração, mas sim as medidas de tamanho e densidade.

Mas, no final, qual a importância de se estudar todos esses conceitos para a implementação de um aplicativo? Isso é notado apenas quando testamos nossa aplicação em diversas configurações de tela diferentes. Na **Figura 6**, podemos observar como ficaria uma aplicação desenvolvida sem preocupação com a densidade da tela.

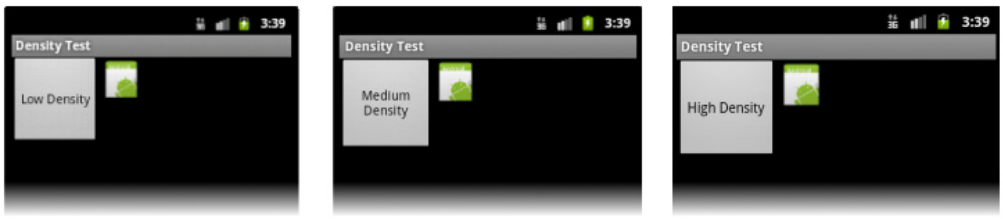
Figura 06 - Aplicação desenvolvida sem preocupação com densidade das telas.



Conforme observado, levando em conta os conhecimentos previamente adquiridos a respeito de densidade de tela, sabemos que a medida de 1 pixel em telas de diferentes densidades irá ocupar porções também diferentes da tela. Dessa forma, um botão cujo tamanho é definido em pixels ou uma imagem que não possui o devido tratamento vão aparecer maiores em telas de densidade pequena e, à medida que formos aumentando a densidade da tela, esses componentes irão ocupando cada vez menos espaço na tela.

Se modificarmos a aplicação mostrada na **Figura 6**, levando em consideração as possíveis mudanças de densidade, podemos atingir o resultado mostrado logo abaixo, na **Figura 7**.

Figura 07 - Aplicação desenvolvida levando-se em consideração as densidades das telas.



Como pudemos ver, na forma apresentada na **Figura 7**, independente da densidade da tela, temos a nossa interface, conforme planejado na hora do desenvolvimento. Para se atingir esse resultado, muitas vezes, basta declarar todas as unidades utilizadas pela aplicação como dp, no lugar de outras medidas que vão variar com a densidade da tela.

Existem algumas formas de realizar o tratamento específico de diferentes configurações de telas, que vão desde flexibilizar o layout da aplicação para funcionar como esperado, independente das condições, até indicar quais telas são suportadas pela aplicação.

Para fazer a declaração dos tamanhos de tela suportados, é necessário o uso da tag *supports-screen* no *AndroidManifest*. Feito isso, você garante o uso do seu aplicativo de maneira adequada àquelas pessoas que possuírem aparelhos dentro das suas telas, ditas como suportadas.

Apesar de ser uma forma mais simples de garantir o correto funcionamento da aplicação, limitamos o número de possíveis usuários do nosso sistema, o que nem sempre é interessante. Para os demais casos, quando desejamos disponibilizar uma interface sólida para o maior número possível de aparelhos, podemos — e, muitas vezes, devemos — utilizar de duas formas de tratamento dos layouts e imagens, além de utilizar medidas independentes de pixel.

Para suportar a questão de layouts, podemos definir layouts independentes para os diferentes tamanhos de tela existentes. Devemos utilizar essa técnica quando a configuração de auto redimensionamento do Android não for satisfatória para o nosso propósito. Para tal, devemos criar novas pastas de layout, indicando que essas pastas serão utilizadas pelo Android (essa indicação é feita de forma automática, desde que as pastas que criemos, sigam os padrões definidos pelo Android, mostrados na **Tabela 1**). Os arquivos que essas novas pastas armazenarão devem ter o mesmo nome de seus arquivos armazenados na pasta padrão (sem qualificador), para que o Android possa fazer a escolha de recursos de acordo com seus qualificadores. Caso o arquivo contido na pasta com qualificador não possua o mesmo nome do arquivo padrão, o Android não conseguirá tratar esses dois recursos como versões de um só recurso.

Característica da Tela	Qualificador	Descrição
Tamanho	small	Recursos para telas pequenas.
	normal	Recursos para telas de tamanho *normal* (Tamanho padrão).
	large	Recursos para telas grandes.
	xlarge	Recursos para telas extragrandes.

Tabela 1 – Qualificadores de tamanho para recursos do Android

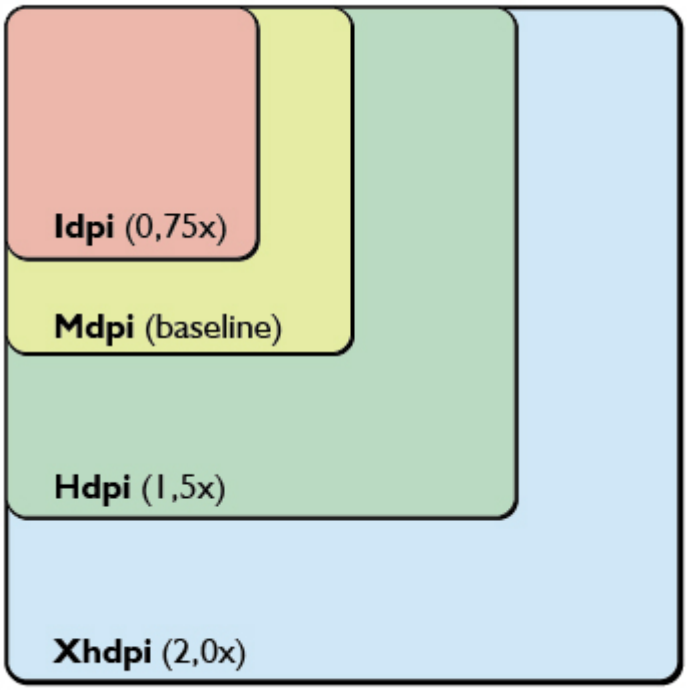
É importante salientar que, a partir da API 13, os conceitos de tamanho de tela foram extintos, e substituídos por outro tipo de qualificador. Esse novo qualificador não limita os tamanhos por padrão, mas deixa por responsabilidade do programador o fazer, dando mais liberdade na modificação de seus layouts. Para utilizar o novo qualificador, basta seguir o modelo: sw*<N>*dp (Sendo *<N>* um número inteiro que indica o tamanho mínimo que a tela deve ter para que os layouts contidos nela sejam aplicados). Por exemplo, uma tela que precise de pelo menos 500 dp para funcionar adequadamente deve ser colocada em uma pasta *layout-sw500dp/*.

Outro ponto a se considerar, é criar layouts independentes para as diferentes orientações de tela, de forma a melhorar a disposição dos componentes de layout de acordo com o espaço disponível, utilizando os qualificadores equivalentes.

Da mesma forma que fazemos com os layouts, podemos fazer com as imagens dos nossos programas. A diferença fica por conta do atributo que verificamos para modificar as imagens, já que, ao invés de usar diferentes imagens para diferentes tamanhos de tela, usaremos diferentes imagens para diferentes densidades. Outro tipo de tratamento que podemos utilizar para imagens é garantir o uso de arquivos de imagem *nine-patch* (extensão .9.png), para que o processo de redimensionamento ocorra sem perda de qualidade da imagem que criamos inicialmente.

Para as imagens *nine-patch*, basta criar os arquivos seguindo o padrão do formato e o redimensionamento será feito de acordo. Já para imagens normais, que vão ser carregadas de acordo com a densidade da tela do usuário, é importante seguir uma proporção pré-determinada entre os tamanhos das imagens, de forma a otimizar o trabalho do Android e a qualidade da imagem que aparecerá para o usuário. Essa proporção é de 3:4:6:8 para as densidades baixa, média, alta e extra-alta, respectivamente. Tal proporção pode ser melhor observada na **Figura 8** apresentada a seguir.

Figura 08 - Proporção indicada para imagens diferentes de acordo com as densidades.



Após o trabalho de ajustar as imagens e o layout, é hora de testar o aplicativo utilizando as diferentes configurações de tela que se pretende suportar para garantia de qualidade. Para isso, basta seguir a **Tabela 2** a seguir, que mostra como criar diferentes cenários de tela utilizando o emulador Android.

	Densidade Baixa (120), *ldpi*	Densidade Média (160), *mdpi*	Densidade Alta (240), *hdpi*	Densidade Muito Alta (320), *xhdpi*
Tela pequena	**QVGA(240x320)**		480x640	
Tela normal	**WQVGA400 (240x400) WQVGA432 (240x432)**	**HVGA (320x480)**	**WVGA800 (480x800) WVGA854 (480x854)** 600x1024	640x960
Tela grande	**WVGA800** (480x800) WVGA854** (480x854)**	**WVGA800* (480x800) WVGA854* (480x854)** 600x1024		
Tela extra grande	1024x600	**WXGA (1280x800)** 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

Tabela 2 – Configurações de telas para utilizar no emulador
* Para emular esse comportamento, especifique uma densidade de 160 enquanto utiliza a configuração de tela WVGA800 ou WVGA854.
** Para emular esse comportamento, especifique uma densidade de 120 enquanto utiliza a configuração de tela WVGA800 ou WVGA854.

Vídeo 04 - Entendendo os Temas e Estilos da Plataforma

Atividade 02

1. Explique por que devemos nos preocupar com compatibilidade do nosso aplicativo ao desenvolver para Android, dando um exemplo prático.

Leitura Complementar

Styles and Themes. Disponível em: <http://developer.android.com/guide/topics/ui/themes.html>. Acesso em: 20 de mai de 2015.

Android Compatibility. Disponível em: <http://developer.android.com/guide/practices/compatibility.html>. Acesso em: 20 de mai de 2015.

Supporting Multiple Screens. Disponível em: <http://developer.android.com/guide/practices/screens_support.html>. Acesso em: 20 de mai de 2015.

Material online (em inglês) sobre aplicações Android.

Resumo

Nesta aula, você estudou os conceitos de estilos e temas, seguidos da compatibilidade de aplicações, apresentando todos os conceitos associados e mostrando as formas de tratar as diferentes configurações de hardware e software que podemos encontrar nos aparelhos Android. Por fim, vimos como flexibilizar os recursos de nossa aplicação e como testar diferentes configurações de tela através do emulador.

Autoavaliação

1. Descreva, brevemente, os conceitos de temas e estilos, apontando ao menos uma diferença e uma semelhança.
2. Um estilo pode utilizar todas as propriedades que um tema pode? Explique.

3. Como podemos garantir que os componentes utilizados terão as mesmas medidas proporcionais independente da densidade da tela?

4. Explique, em poucas palavras, a utilidade dos qualificadores de recursos disponibilizados pela plataforma.

Referências

Android Developers. 2015. Disponível em: http://developer.android.com. Acesso em 20 mai. 2015.

DIMARZIO, J. *Android: A Programmer's Guide*. McGraw-Hill, 2008. Disponível em: <http://books.google.com.br/books?id=hoFI5pxjGesC>. Acesso em: 30 ago. 2012.

HASEMAN, C. *Android Essentials*. Berkeley, CA. USA: Apress, 2008.

MEIER, R. *Professional Android 2 Application Development*. John Wiley e Sons, 2010. Disponível em: <http://books.google.com.br/books?id=ZthJlG4o-2wC>. Acesso em: 30 ago. 2012.