O aplicativo parece incrivelmente impressionante e bonito! Parece pronto! Então por que mudá-lo?

Isto pode ocorrer em times de desenvolvimento de aplicativos. Você constrói a versão 1 de um aplicativo. Então o time decide melhorar o design, para ter uma experiência de usuário ainda melhor. Então você procede para construir a versão 2 do aplicativo!

Quando você fala com qualquer desenvolvedor Android profissional, eles podem lhe contar muitas histórias sobre como a interface de usuário para seus aplicativos desenvolveu-se nos últimos meses ou até mesmo anos.

Como desenvolvedor, a habilidade importante de se ter é ser capaz de**refatorar** o seu código. Manter a mesma funcionalidade no aplicativo, mas visualmente, o aplicativo parece diferente. Você não pode romper nenhuma funcionalidade existente (por exemplo, você não pode perder as função de reprodução de imagens ou áudio) quando mudar para o novo design. Quando você rompe algo, e o usuário perde a habilidade de fazer algo no aplicativo comparado à versões prévias, isto é chamos **regressão**.

Para ter certeza de que não rompemos nada, vamos prosseguir em pequenas etapas, uma por vez. Tente fazer o aplicativo rodar no seu disposito o mais frequente possível. Você não irá querer passar 5 dias escrevendo um novo código e então perceber que ele não roda mais no seu dispositivo.

Aqui estão os novos designs. Nós queremos deslizar entre as listas de palavra. Isto nos economiza um toque extra para ter que abrir as listas de vocabulário de palavras. Quando iniciamos o aplicativo, podemos ver imediatamente as listas de palavras.

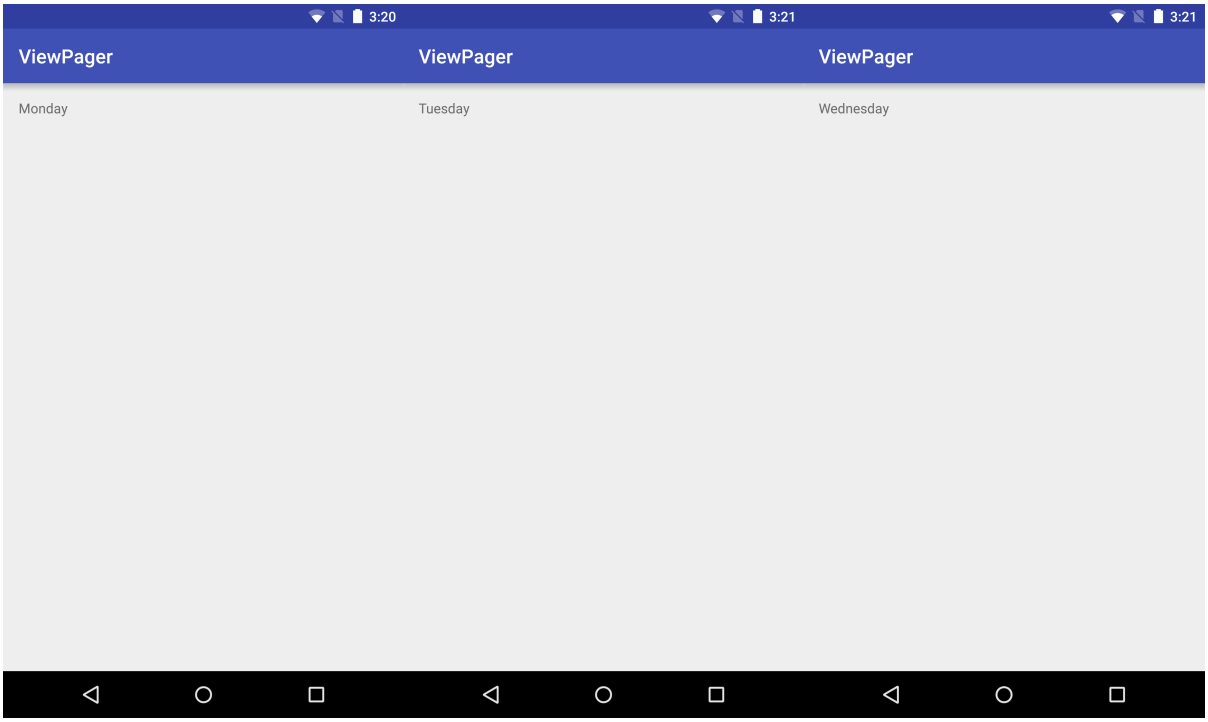
Salve uma cópia da condição atual do aplicativo

IMPORTANTE: Uma vez que vamos mover partes principais do código, definitivamente faça uma cópia da condição atual do seu aplicativo. Salve como um backup, então pelo menos você terá uma versão do aplicativo que funciona para retroceder.

No seu computador, procure pela pasta “AndroidStudioProjects”. Deve haver uma pasta “Miwok” como todo o código do seu aplicativo dentro dela. Faça uma cópia daquela pasta e a salve em algum outro lugar.

Iremos abordar esta alteração em múltiplas etapas.

Primeiramente, você irá testar um aplicativo de amostra que contêm um ViewPager. Você verá como um FragmentPagerAdapter irá proporcionar diferentes Fragmentos para cada “page” que você deslizar. Então aprenderemos um pouco mais sobre Fragmentos, para prepará-lo para a próxima etapa de código.



Você irá refatorar a lógica de 4 atividades (numbers, colors, family, phrases) em 4 fragmentos. O aplicativo terá a mesma aparência, mas toda a lógica para mostrar a lista de palavras estará nos fragmentos, em vez de estar nos arquivos de atividade. Assim que a lógica estiver dentro dos fragmentos, podemos seguir a próxima etapa.

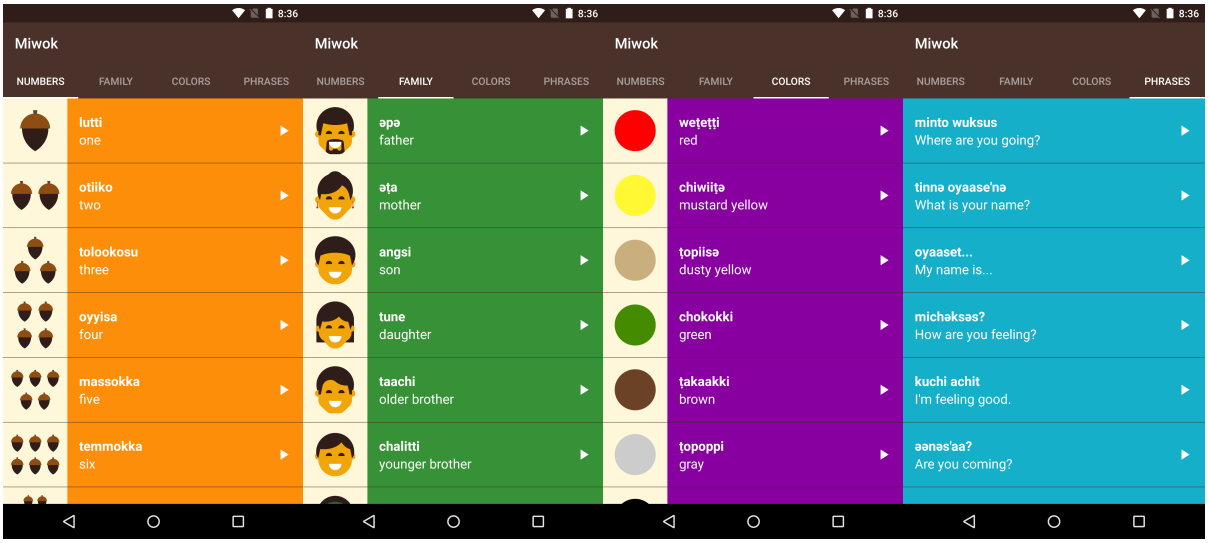
- NumbersActivity conterá NumbersFragment
- FamilyActivity conterá FamilyFragment
- ColorsActivity conterá ColorsFragment
- PhrasesActivity conterá PhrasesFragment

A ViewPager é a que você desliza entre diferentes “pages” ou telas. Iremos modificar o MainActivity para que contenha um ViewPager com 4 páginas, onde cada página é um

Fragmento. Podemos deslizar entre cada Fragmento para ver uma lista diferente de palavras.

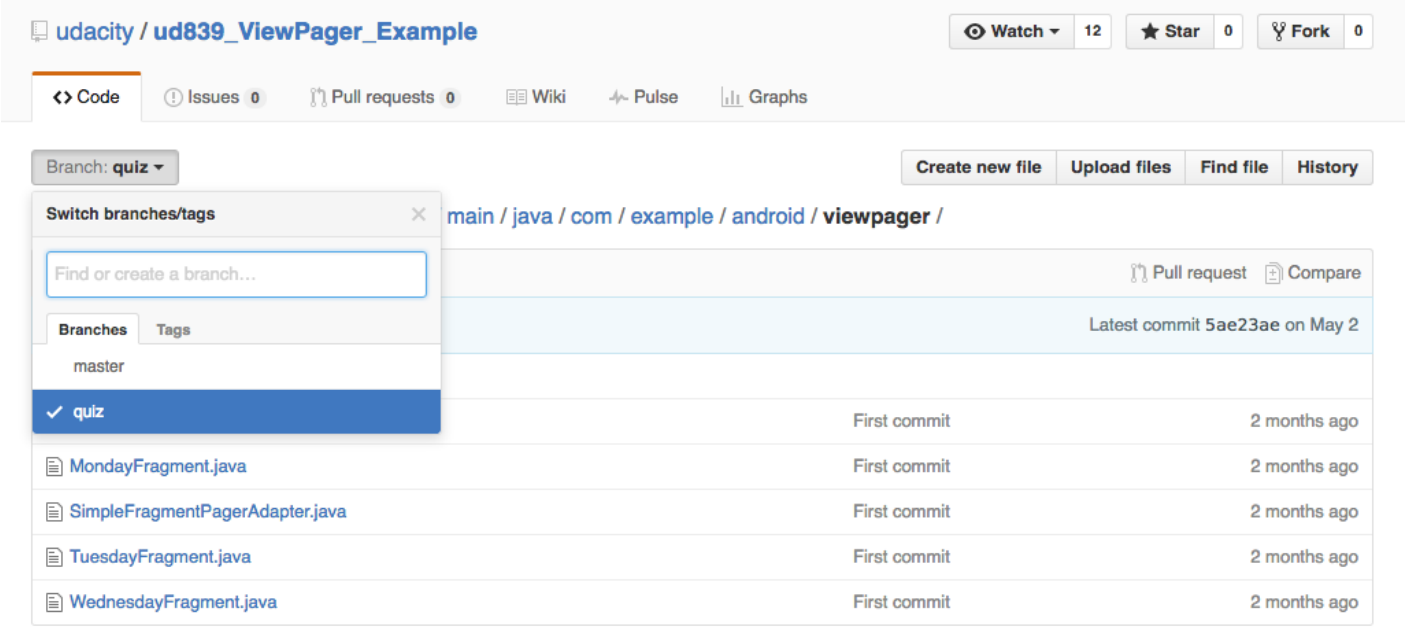
A esta altura, estamos removendo o layout que tinha um botão para cada categoria. Iremos também deletar as atividades de categoria (NumbersActivity, FamilyActivity, ColorsActivity, PhrasesActivity) porque o aplicativo contêm apenas 1 atividade neste momento (MainActivity. Você pode dizer que estamos no MainActivity porque sobre a barra de navegação do aplicativo está escrito “Miwok” em todas as telas.

Assim que o ViewPager estiver funcionando, iremos adicionar transversalmente as abas no topo do ViewPager, então você poderá tocar para ir a qualquer página específica.



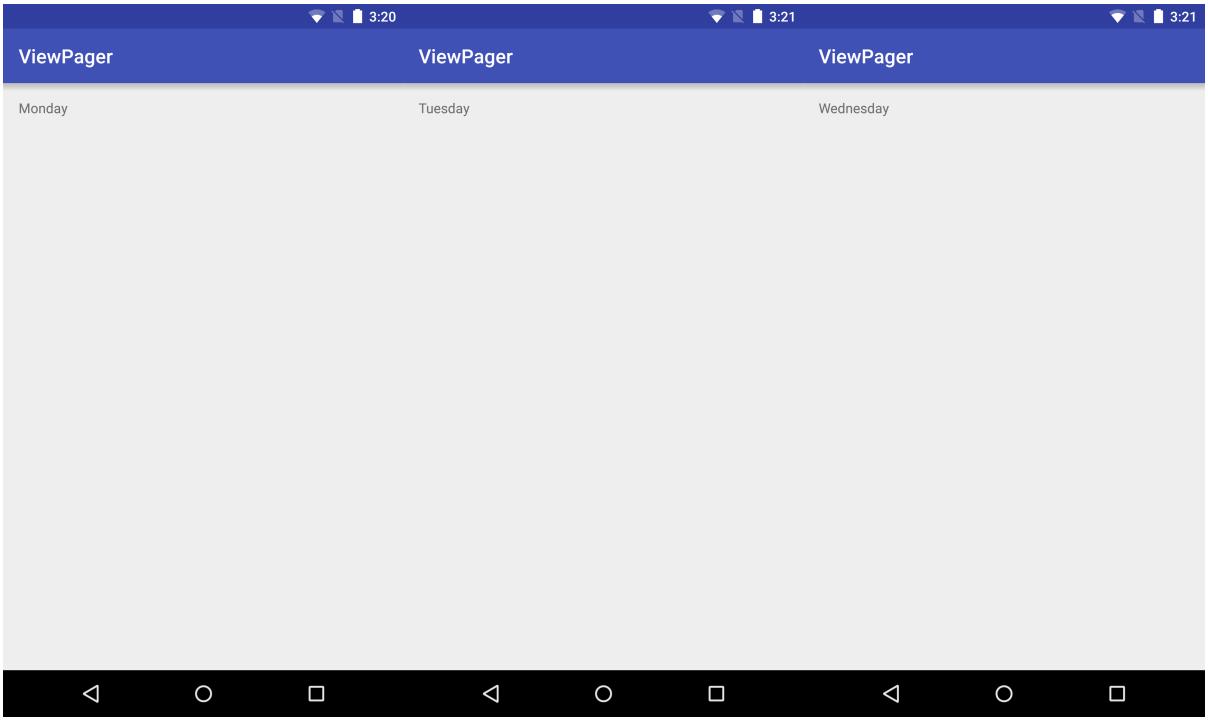
Vamos rodar um aplicativo de exemplo para explorar como um ViewPager trabalha em um cenário simples.

1. Baixe este aplicativo de amostra do **link do GitHub** clicando sobre o "Download zip" (certifique-se que você está no branch "quiz").



2. Importe este projeto no Android Studio e rode o aplicativo no seu dispositivo.

3. Quando você abre o aplicativo, devem ter 3 páginas para deslizar neste ViewPager. Primeiro deve dizer "Monday", depois "Tuesday", depois "Wednesday.



4. Navegue em torno da base de código e veja como o layout e os arquivos Java estão trabalhando juntos. Prepare-se para o quizz, que pedirá para você modificar este aplicativo de

exemplo.

Como funciona?

O ViewPager funciona obtendo seus dados de um adapter - chamado FragmentPagerAdapter.

No nosso caso, queremos personalizar o adapter para exibir nossos próprios fragments, por isso temos que usar a herança para subclasse do FragmentPagerAdapter. Ao herdar, obtemos todas as funcionalidades do FragmentPagerAdapter gratuitamente, e podemos adicionar nossa própria personalização em cima dela. Criamos a classe SimpleFragmentPagerAdapter e estendemos a partir da classe FragmentPagerAdapter.

Quando você inicia o aplicativo em seu dispositivo, primeiro o ViewPager pergunta ao adapter quantas páginas teremos. No nosso caso, o adapter diz que teremos 3 páginas. Veja o método SimpleFragmentPagerAdapter getCount().

Para que o ViewPager exiba a página 0, o ViewPager solicita o adapter pelo fragment 0th. Veja o método SimpleFragmentPagerAdapter getItem(int position). Quando o usuário desliza para a esquerda, passamos para a página 1, o que significa que o ViewPager pede ao adapter para o fragmento na posição 1. Quando chegarmos à página 2, o ViewPager pede o adapter para o fragment na posição 2. Assim, dependendo de qual página (também conhecida como posição) para o qual o usuário deslizou, o fragment correspondente é mostrado.

Exercício

Modifique o View Pager Sample App

Quando o app iniciar, ele deve mostrar a segunda-feira, então a terça, quarta, quinta e sexta-feira.
Sinta-se livre para exerimentar alterações além das discutidas aqui

Para ter 5 páginas, adicionamos um pouco mais de arquivos java contendo Fragments:

- **ThursdayFragment.java**
- **FridayFragment.java**

Nós também adicionamos arquivos de layout correspondentes:

- **fragment_thursday.xml**
- **fragment_friday.xml**

Then we modified the **SimpleFragmentPagerAdapter.java file** to create these 5 different fragments.

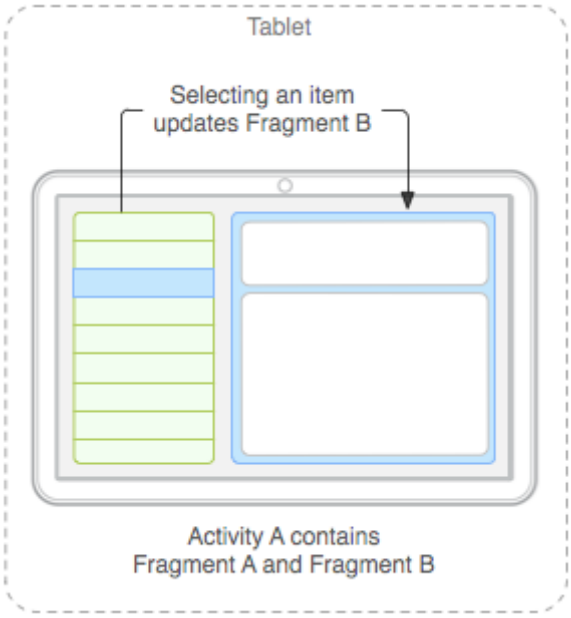
Aqui temos o **código**para a última versão do aplicativo.

Dicas avançadas: Ao invés de ter 5 arquivos de Fragments individuais que fazem praticamente a mesma coisa, você pode também passar um argumento String para o Fragment. Então este Fragment pode conter lógica para mudar o texto mostrado no TextView dependendo da String passada nele. Use o método Fragment **setArguments(Bundle args)**. Veja o **guia de Fragments** para mais detalhes.

Um fragmento é somente uma parte de uma atividade. Você pode ter um fragmento que ocupa parte da tela ou a tela toda. Ou você pode mostrar múltiplos fragmentos ao mesmo tempo para tornar uma tela inteira. Dentro de uma atividade, você também pode trocar diferentes fragmentos entre si (você também pode ter fragmentos invisíveis, que tem alguma função em relação à atividade, mas não veremos isso neste curso).

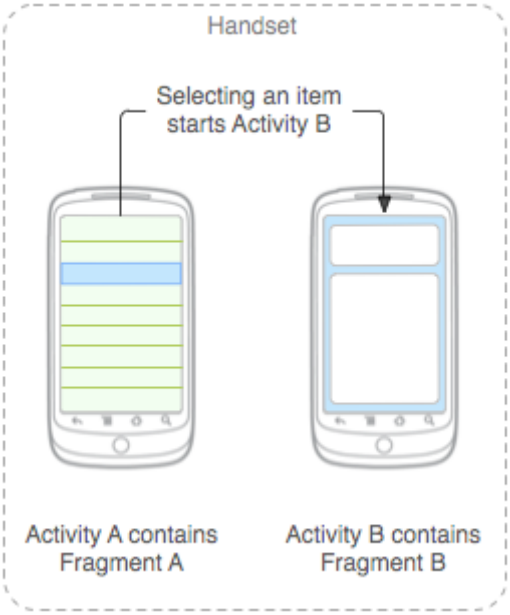
Os fragmentos foram apresentados em Android quando começamos a construir para telas maiores tal como tablets. Deixe-me lhe mostrar um exemplo.

No padrão master/detail, um fragmento lista está no lado esquerdo da tela, enquanto o fragmento detalhe à direita troca dependendo no item da lista que é selecionado.



(**Fonte**)

Neste caso, utilizar fragmentos é conveniente quando se adapta o aplicativo à dispositivos menores, como telefones. Quando o usuário abre o aplicativo em um telefone, ele pode enxergar o fragmento lista. Se ele tocar em um item da lista, ele poderá navegar ao fragmento detalhe.



(**Fonte**)

No telefone, apenas 1 fragmento é mostrado por vez. No tablet, 2 fragmentos são mostrados ao mesmo tempo, um ao lado do outro, para obter vantagem do estado real maior da tela disponível.

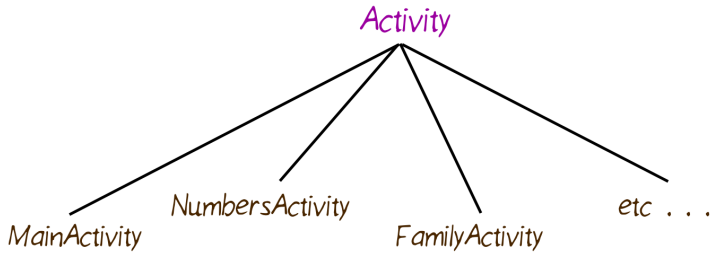
(**Fonte**)

Este é apenas um caso de uso para fragmentos. Você pode dividir a lógica do seu aplicativo em quantos fragmentos você quiser. Entretanto, dividi-lo em muitos fragmentos pode causar um overhead extra porque sua Atividade precisará gerir a comunicação entre os fragmentos.

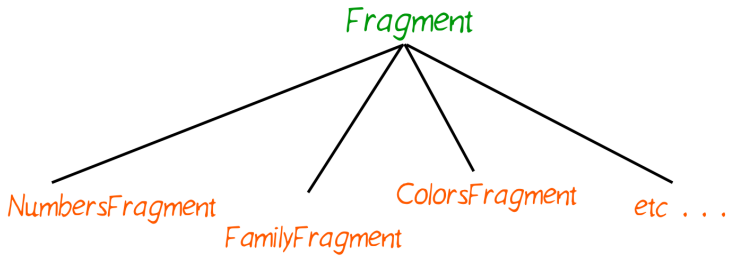
Um Fragmento é uma classe Java. Você cria o seu próprio Framento, assim como você criou suas próprias atividade. Você subclassifica a classe Fragmento no framework Android.

Revisar Sucessão

CLASS HIERARCHY



CLASS HIERARCHY



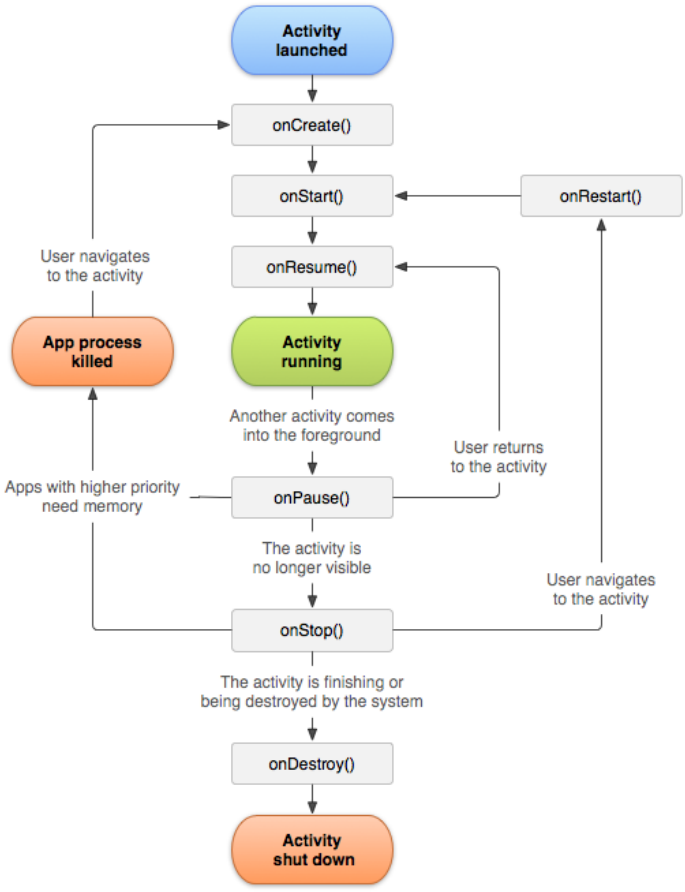
Porque um Fragmento pode ser criado, adicionado à tela, e então removido da tela, ele tem seu próprio ciclo de vida. E adivinhe? O Android provê callbacks assíncronas onde você pode ser notificado dessas mudanças de estado. O estilo desses callbacks é similar a Activity.

Imagine uma Activity contendo um Fragmento que preenche toda a tela. Quando a atividade é criada, o Fragmento também é criado. Quando a Atividade é retomada e ativa, o Fragmento é retomado e ativo, também. Há uma relação próxima entre o ciclo de vida da Atividade e o ciclo de vida do Fragmento.

O Fragmento também tem callbacks adicionais onde você, como desenvolvedor, pode ser notificado quando ele está ligado a Atividade delimitadora ou separado da Atividade.

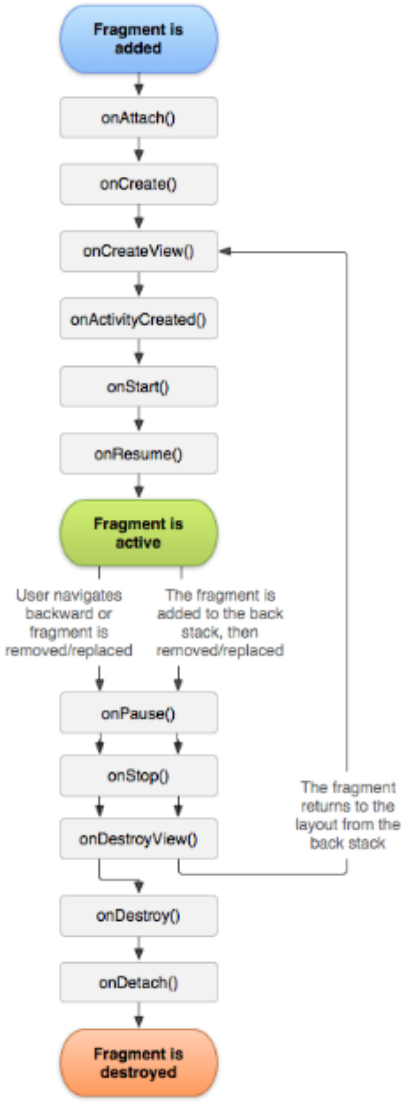
Se múltiplos fragmentos ocupam a tela toda, então todas elas precisam vincular-se à mesma atividade. Elas precisam saber quando a atividade está retomada / pausada, então elas podem retomar ou pausar o que estão fazendo dentro do fragmento. Se você trocar e substituir diferentes fragmentos ao longo do tempo de vida do aplicativo, os Fragmentos precisam ser criados e destruídos de maneira adequada, também.

Lembre-se, este é o ciclo de vida da atividade.



(**Fonte**)

Similarmente, o fragmento tem um ciclo de vida que é vinculado as etapas do do ciclo de vida da atividade.

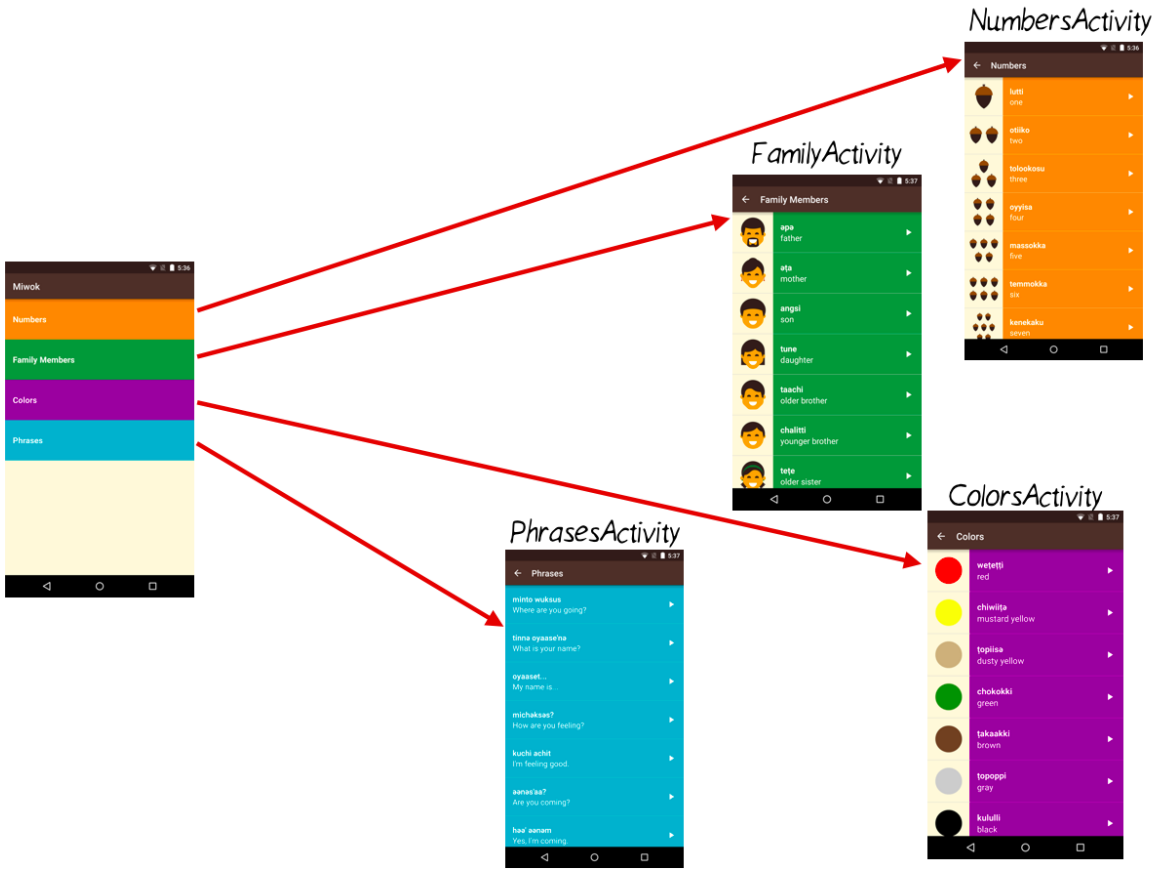


(**Fonte**)

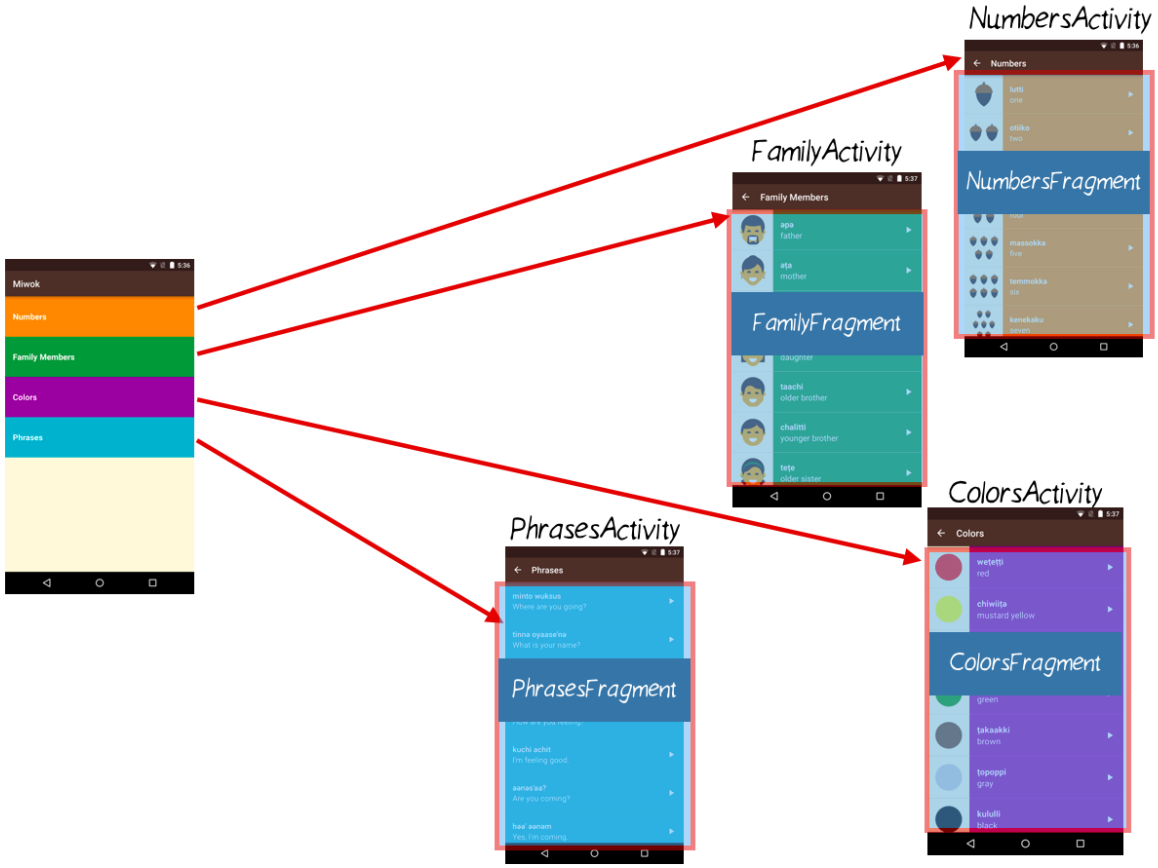
Fragmento é um tópico difícil. Você encontrará mais Fragmentos à medida que você avança em sua jornada como um desenvolvedor Android. Se você está curioso para mais detalhes, neste momento, veja o **guia de desenvolvedor de Fragmento** sobre como **manusear o ciclo de vida do Fragmento** ou como **construir um UI dinâmico com Fragmentos**.

Há também muitos outros tutoriais sobre Fragmentos escritos por desenvolvedores na comunidade.

Não usaremos um ViewPager ainda. Vamos preparar nossos fragments primeiro. Atualmente, temos 4 Activity's de categoria e 0 Fragments.



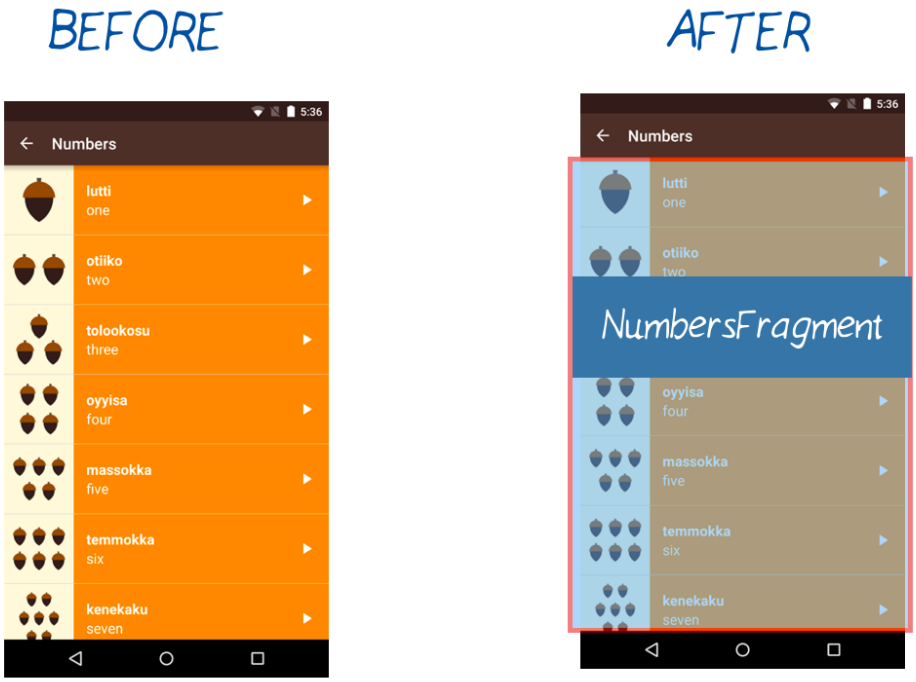
Ao fim desta tarefa de codificação, queremos 4 Activity's de categoria com 4 Fragment's. Cada Activity conterá um Fragment.



A fim de mover a lógica para o fragmento de arquivos java, vamos ter que copiar/colar um monte de código. O código é basicamente o mesmo. No entanto, qualquer código que assumiu que estávamos dentro de uma classe Actividty terá que ser ligeiramente modificada para ter em conta que agora o código está dentro de uma classe Fragment. Por exemplo, os retornos de chamada do ciclo de vida da Activity (onCreate, onStop) não existem quando estamos dentro de uma classe Fragment. Existem métodos semelhantes (onCreateView, onStop), por isso só precisamos de pequenas modificações para ajustar os retornos de ciclo de vida do Fragment.

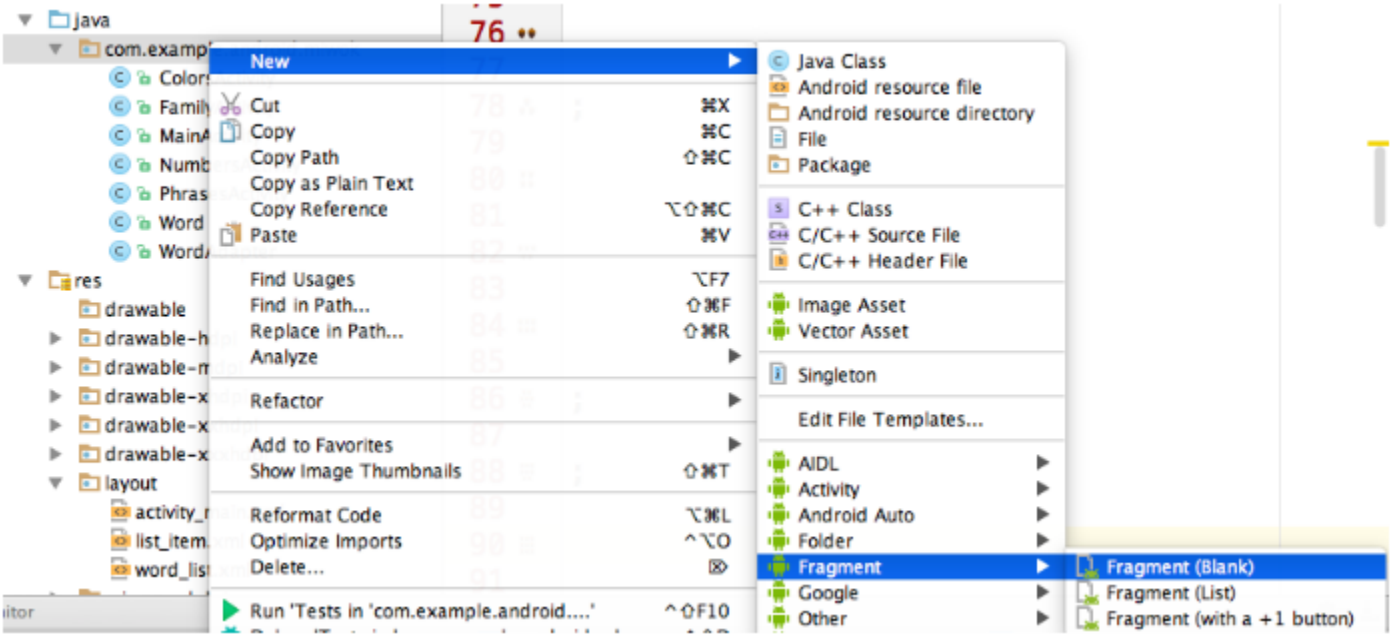
	NumbersActivity.java	NumbersFragment.java
Global variables: ***MediaPlayer, AudioManager, OnCompletionListener, OnAudioFocusChangeListener***	Same	Same
Creating the view	In onCreate() activity lifecycle callback: *Set the content view, setup the UI*	In onCreateView() fragment lifecycle callback: *Inflate the XML layout, setup the UI, return the root view*
Leaving the screen	In onStop() activity lifecycle callback: *Release the media player*	In onStop() fragment lifecycle callback: *Release the media player*
Helper method: ***releaseMediaPlayer()***	Same	Same

Não se preocupe se a tabela é confusa. Vamos passo a passo através das alterações de código para o NumbersActivity, e em seguida é bem provável que o processo se tornará mais claro para você. Como mostrado no diagrama abaixo, vamos deixar de ter um NumbersActivity (antes do estado) para ter um NumbersActivity com um NumbersFragment contido dentro dela (após o estado). Em seguida, para o próximo quiz, você converterá o resto das activity's da categoria por sua conta, seguindo o mesmo processo.

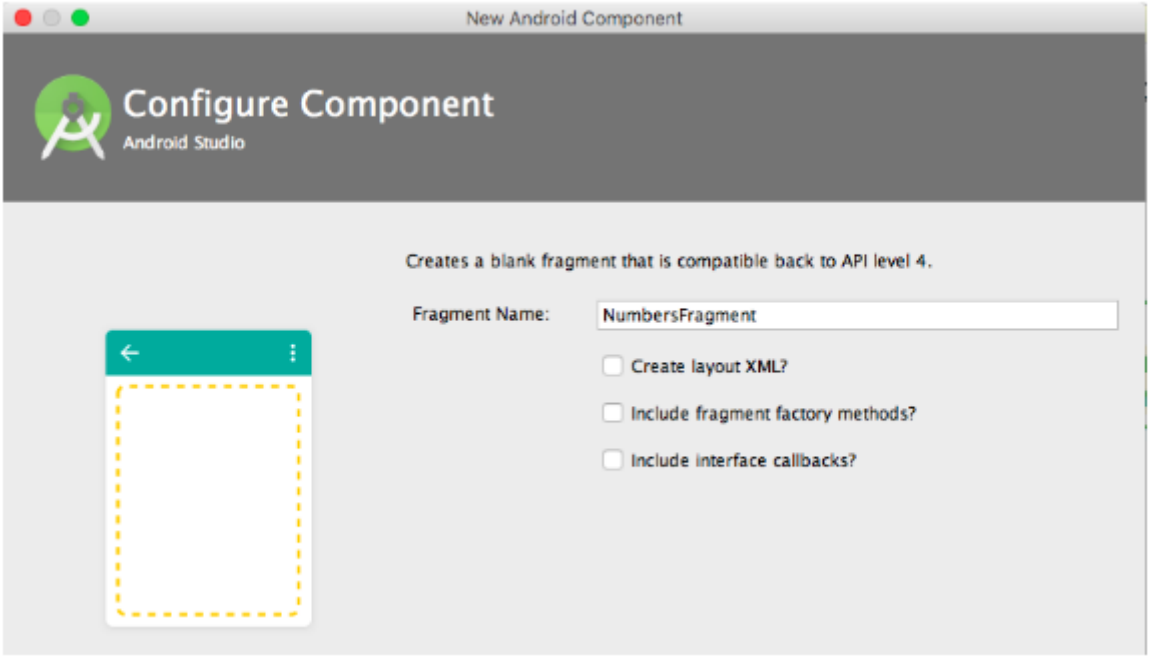


Crie classe NumbersFragment

1) Para começar, crie um novo arquivo Java para o NumbersFragment. Botão direito no diretório "com.example.android.miwok". Vá para New > Fragment > Fragment (Blank).



2) Preencha o wizard. Dê um nome ao fragment: NumbersFragment. Desmarque todas as check boxes porque estaremos adicionando código ao arquivo por nós mesmos.



3) Android Studio irá auto-gerar uma nova classe Fragment para você em um arquivo chamado NumbersFragment.java. Se parecerá com algo assim:

```
package com.example.android.miwok;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

/**
* Uma simples {@link Fragment} sub-classe.
*/
public class NumbersFragment extends Fragment {

    public NumbersFragment() {
       // Requerido construtor público e vazio
   }

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container, Bu
   TextView textView = new TextView(getActivity());
   textView.setText(R.string.hello_blank_fragment);
   return textView;
   }
}
```

Copie código do NumbersActivity e cole-o no NumbersFragment

4) Comece copiando as variáveis globais do NumbersActivity para o NumbersFragment. (Delete-os do NumbersActivity ao fazer isto.) O código deve se parecer com **isto**.

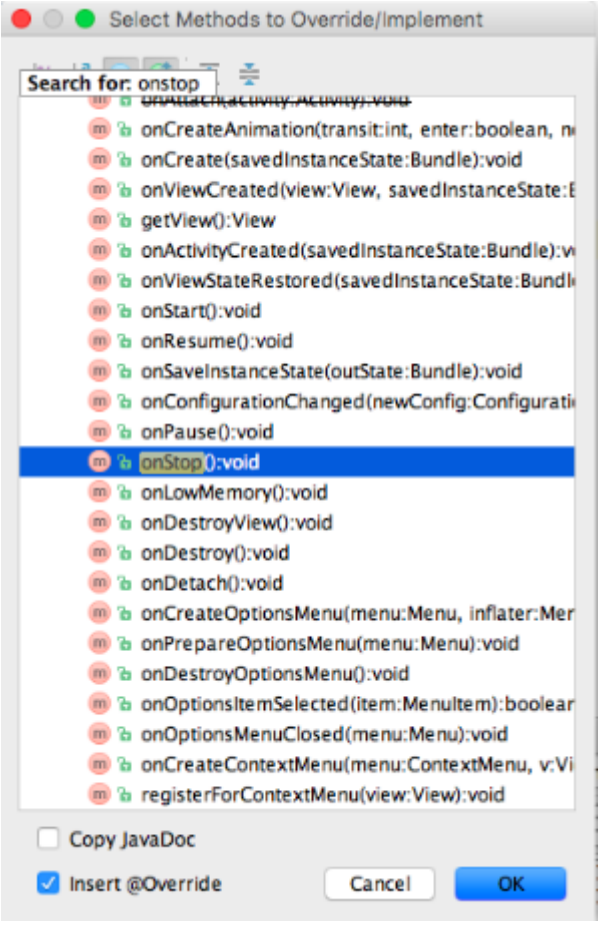
5) Copie o método auxiliar releaseMediaPlayer() do NumbersActivity para o NumbersFragment Código para **releaseMediaPlayer()**.

Adapte o código para o Ciclo de Vida de Fragment (ao invés do Ciclo de Vida de Activity)

6) Sobrescreva o método de Fragment onStop().

Você não pode copiar o método exato onStop() da NumbersActivity porque o método Fragment onStop() tem uma assinatura de método ligeiramente diferente. Caso você esteja se perguntando, a classe Activity usa o modificador "protected" no método, enquanto que a classe Fragment tem o modificador "public" no método. Mas não se preocupe com os detalhes desses modificadores agora.

Mova seu cursor para um espaço vazio da classe, onde você possa adicionar um novo método. Use a atalho de teclado CTRL + O para abrir um dialog e selecionar um método para sobrescrever. Escreva em "onStop" e quando achar este resultado, clique em OK.



Android Studio automaticamente adicionará este método a sua classe NumbersFragment para você:

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

```
}
```

Modifique o método onStop() para que chame o método releaseMediaPlayer:

```
@Override
public void onStop() {
    super.onStop();

    // Quando a activity for parada, libere os recursos media player porque n
    // mais tocando mais sons.
    releaseMediaPlayer();
}
```

7) Sobrescreva o método de Fragment onCreateView().

O método Activity's onCreate() é ligeiramente diferente do método de Fragment onCreateView(). Dentro do método de Activity onCreate(), podemos chamar setContentView() para setar o layout para a activity. No Fragment, precisamos inflar a view do ID de recurso de layout XML e retornar esta view no método onCreateView(). Note que o layout para o Fragment usará o recurso de layout XML word_list porque estará mostrando uma lista de palavras.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundl
    View rootView = inflater.inflate(R.layout.word_list, container, false);

    /** TODO: Insira todo o código do método NumberActivity's onCreate() após

    return rootView;
}
```

Depois de copiar o código do método onCreate() do NumbersActivity, você receberá um monte de erros no Android Studio porque o código supõe que ele seria executado dentro de uma classe Activity, não uma classe Fragment. Abaixo temos instruções sobre como resolver cada um desses erros. Se você não quiser corrigir cada erro um a um, você pode pular para o trecho de código abaixo que mostra como o seu NumbersFragment onCreateView() deve se parecer quando estiver pronto.

Erro #1: Você obterá um erro dizendo que você não consegue resolver o método "findViewById(int)" porque o Fragment não possui um método findViewById, enquanto que na Activity existe este método (veja este **link**).

```
ListView listView = (ListView) findViewById(R.id.list);
```

Conserte o erro chamando findViewById(int) no objeto rootView, que deve conter views filhas como as ListView.`ListView listView = (ListView) rootView.findViewById(R.id.list);`

Erro #2: Você obterá um erro dizendo que você não consegue resolver o método "getSystemService(String)" porque o Fragment não tem acesso ao serviço de sistema, enquanto que a Activity possui (veja **este link**).`mAudioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);`

Conserte o erro obtendo a instância do objeto Activity primeiramente. Esta é a Activity que envolve o Fragment corrente, que será o NumbersActivity para o NumbersFragment. Então chame getSystemService(String) naquele objeto de Activity.

```
mAudioManager = (AudioManager) getActivity().getSystemService(Context.AUDIO_SERVICE);
```

Erro #3: Há um problema com os argumentos passados no construtor WordAdapter porque o primeiro parâmetro "this" se refere a esta classe (que é a NumbersFragment), e um Fragment não é um Contexto válido. No entanto, o código costumava funcionar quando "this" se referia a NumbersActivity porque uma Activity é um Contexto válido.`WordAdapter adapter = new WordAdapter(this, words, R.color.category_numbers);`

Conserte o erro passando em uma referência a Activity que envolve este Fragment como o contexto.`WordAdapter adapter = new WordAdapter(getActivity(), words, R.color.category_numbers);`

Erro #4: Ao criar o objeto MediaPlayer, precisamos passar em um context. Novamente, "this" se refere ao NumbersFragment (e não ao NumbersActivity), e o Fragment não é um Contexto válido.`mMediaPlayer = MediaPlayer.create(NumbersActivity.this, word.getAudioResourceId());`

Conserte o erro passando a activity para o primeiro parâmetro de entrada. `mMediaPlayer = MediaPlayer.create(getActivity(), word.getAudioResourceId());`

Após consertar esses 4 casos, não deve ter mais erros neste arquivo! O método NumbersFragment onCreateView() deve se parecer com **isto**.

Como um teste de integridade, este é o resultado final da classe **NumbersFragment**.

Atualizar NumbersActivity

8) No diretório res/layout, crie um novo arquivo de layout chamado activity_category.xml. A parte importante é que a view tem um ID. Escolhemos dar a view um ID chamado "container".

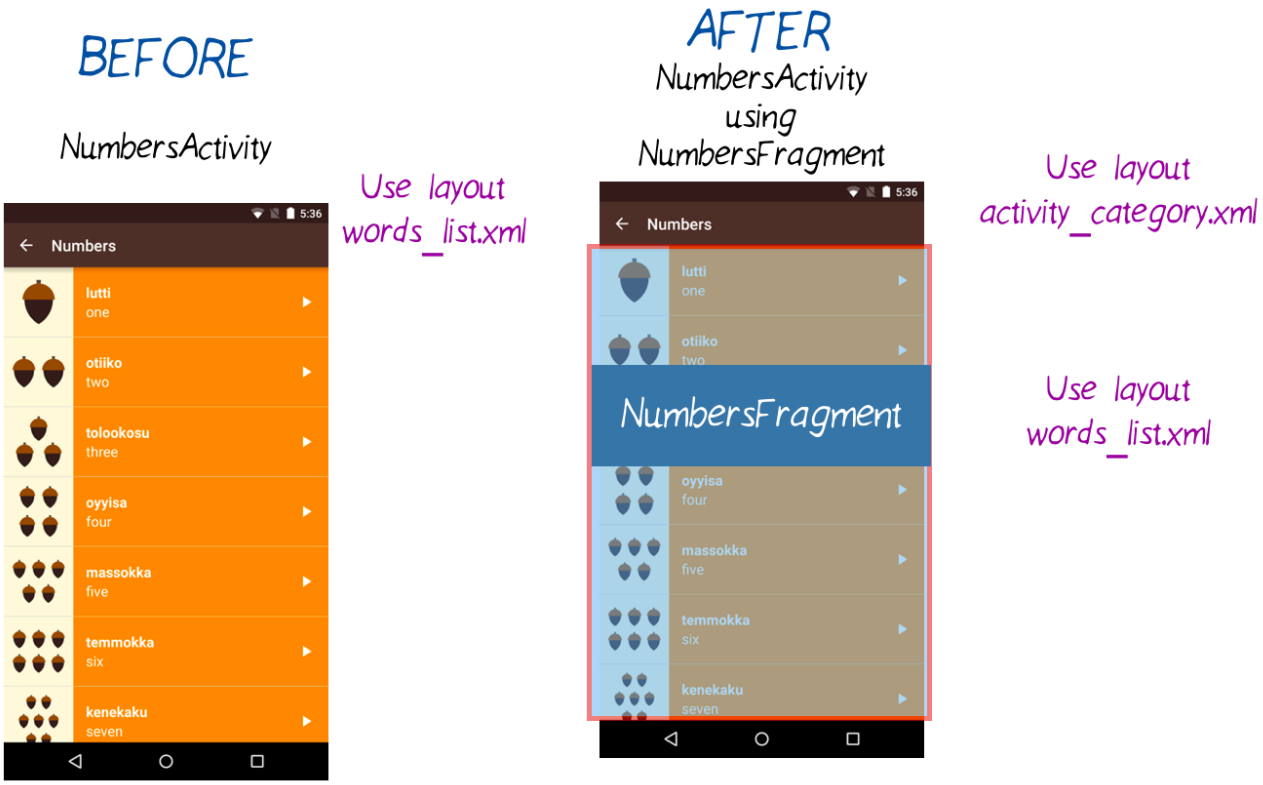
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/container"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"/>
```

9) Agora precisamos atualizar o NumbersActivity para usar o NumbersFragment, senão teremos código duplicado que faz a mesma coisa em ambas as classes.

Substitua o código do NumbersActivity com este trecho de código. Iremos usar esta activity simplificada que seta o recurso de layout XML activity_category como o content view. Então um novo NumbersFragment é criado e inserido no container view, usando um **FragmentTransaction**(não é necessário entender os detalhes disto agora). Como o container tem "match_parent" para width e height, o NumbersFragment pegará todo o width e height da tela.

```
package com.example.android.miwok;

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

public class NumbersActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_category);
   getSupportFragmentManager().beginTransaction()
           .replace(R.id.container, new NumbersFragment())
           .commit();
   }
}
```

10) Para esclarecer, a NumbersActivity costumava mostrar o layout the word_list.xml. Agora, a NumbersActivity mostra o layout activity_category.xml, e o NumbersFragment mostra o layout word_list.xml.



Agora, o NumbersActivity está usando um NumbersFragment! Execute o aplicativo para verificar se a lista de números ainda funciona. Deve se parecer o mesmo porque este é apenas um ponto médio ao longo do caminho para o nosso objetivo, como explicado anteriormente.

Ótimo trabalho! No próximo teste, você repetirá os mesmos passos #1 - 9 para as outras categorias. Todas as activity's de categoria podem usar o recurso de layout activity_category.xml.

No final, seu aplicativo deve ter o mesmo aspecto, mas cada activity de categoria conterá um fragment diferente. Teste seu aplicativo para certificar-se de que a activity está mostrando o fragment certo. Certifique-se de cada fragment tem a cor correta do tema. (Esses foram os dois bugs na minha aplicação quando implementei a solução pela primeira vez.) Certifique-se também de que a reprodução de áudio ainda funciona.

Há um monte de copiar/colar, boa sorte!!

Exercício

Finalize a refatoração transformando as 4 activities em 4 fragmentos.

NumerosActivity vai conter o NumerosFragment
FamilaActivity vai conter o FamiliaFragment
CoresActivity vai conter o CoresFragment
FrasesActivity vai conter o FrasesFragment

Garanta que o app ainda se comporte da mesma maneira e que você ainda é capaz de visualizar as 4 categorias.

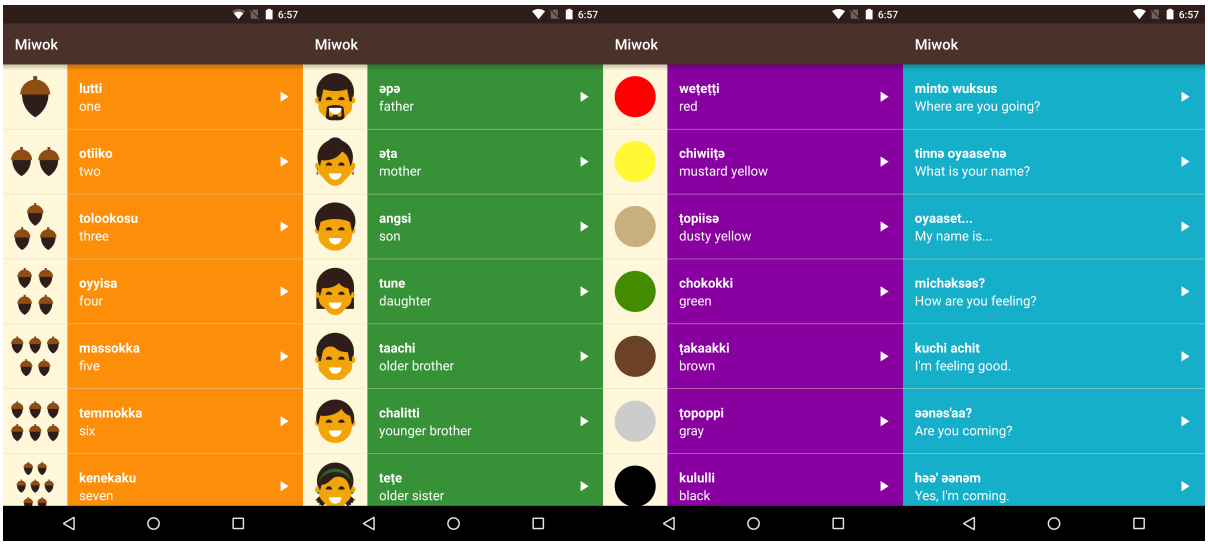
Estas são as mudanças necessárias para completar o quiz.

Este é o estado do código após completar o quiz.

Agora que a lógica para exibir listas de palavras está no Fragment, você pode passar a usar um ViewPager na MainActivity.

Você concluirá esta tarefa por conta própria, mas fique à vontade para consultar o vídeo dos Padrões de Desenvolvimento Android que você assistiu anteriormente. Você também pode revisar o aplicativo de amostra do ViewPager.

Isso é como o aplicativo deve se parecer quando você tiver terminado. Quando o aplicativo é aberto, você verá imediatamente a lista de palavras numéricas. Então você poderá deslizar horizontalmente entre as listas de palavras. Nos bastidores, temos uma única activity (MainActivity) que contém um ViewPager com 4 Fragments diferentes. Você pode dizer que é uma activity única porque o nome "Miwok" permanece na barra de aplicativos enquanto desliza entre as telas.



Exercício

Adicione ViewPager para a MainActivity

Modifique a MainActivity para contar um ViewPager que permita ao usuário trocar entre as listas de palavras.
Crie uma classe CategoriaAdapter que extende de FragmentPagerAdapter. Forneça o fragmento apropriado para cada página no ViewPager.
Exclua os arquivos não utilizados.

Estas são as mudanças necessárias para completar o quiz. Este é o estado do código após completar o quiz.

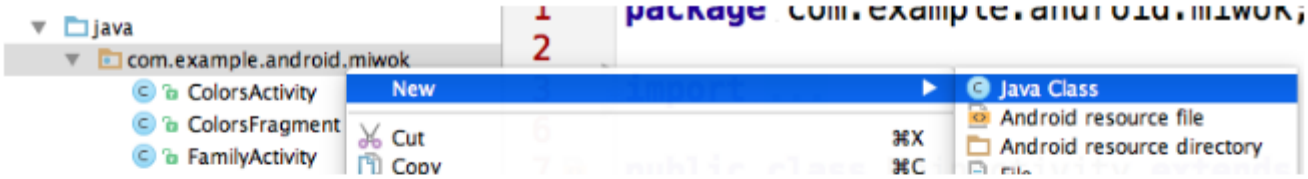
Aqui temos as mudanças necessárias para trocar para um ViewPager e FragmentPagerAdapter no aplicativo Miwok.

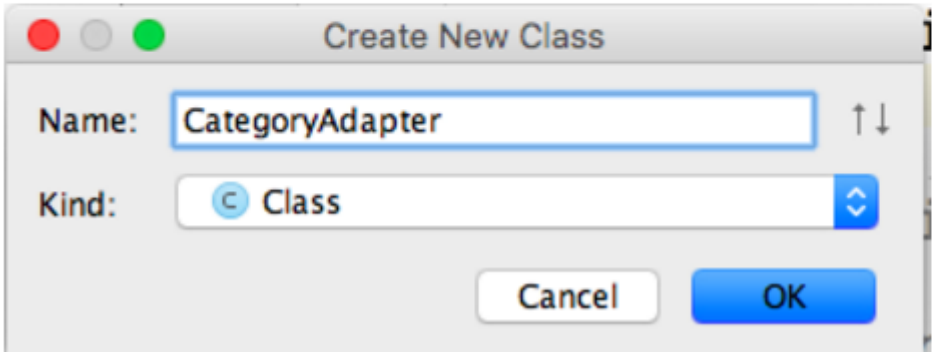
1) Primeiramente modifique o layout activity_main.xml para conter um ViewPager. Você pode deletar os 4 TextViews de categoria que constumavam estar neste arquivo de layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="@color/tan_background"
   android:orientation="vertical"
   tools:context="com.example.android.miwok.MainActivity">

   <android.support.v4.view.ViewPager
       android:id="@+id/viewpager"
       android:layout_width="match_parent"
       android:layout_height="match_parent"/>

</LinearLayout>
```

2) A fim de popular páginas no ViewPager, precisamos de um adapter.

Crie um novo arquivo para o adapter clicando com botão direito no diretório com.example.android.miwok na aba Project directory. Então vá em New > Java class. Crie uma nova classe chamada CategoryAdapter.





3) Android Studio automaticamente criará uma nova classe Java no arquivo CategoryAdapter.java com estes conteúdos:

```
package com.example.android.miwok;

public class CategoryAdapter {

}
```

4) Modifique a linha de declaração de classe CategoryAdapter para estender a classe FragmentPagerAdapter. O Android Studio relatará erros, avisando que você precisa preencher os métodos necessários. Você pode pressionar Alt + Enter para uma correção rápida para o arquivo, ou clique direito e tentar resolver o erro. O Android Studio adicionará uma versão nua dos métodos que você precisa implementar. Android Studio também adicionará um construtor para o CategoryAdapter que recebe um FragmentManager. Veja abaixo:

```
package com.example.android.miwok;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

/**
  * Criado por katherinekuan em 4/14/16.
*/
public class CategoryAdapter extends FragmentPagerAdapter {

   public CategoryAdapter(FragmentManager fm) {
       super(fm);

   }

   @Override
   public Fragment getItem(int position) {
   return null;
   }

   @Override
   public int getCount() {
       return 0;
   }
}
```

5) Sobrescreva os métodos com a lógica que queremos em nosso aplicativo Miwok. Precisamos pensar sobre:

Questão: Quantas páginas precisamos no ViewPager? *Resposta:* 4 páginas, então devemos retornar 4 no método CategoryAdapter getCount().

Questão: Qual Fragment devemos mostrar se a posição for 0? Ou 1 ou 2? *Resposta:* Dentro do método CategoryAdapter getItem(int position), criamos um condicional if/else statement para retornar o fragment da categoria apropriada para uma dada posição.

A classe CategoryAdapter resultante se parece com isto:

```
package com.example.android.miwok;

import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

/**
 * {@link CategoryAdapter} é um {@link FragmentPagerAdapter} que pode prover
 * cada item de lista baseado em um recurso de dado que é uma lista de {@lir
 */
public class CategoryAdapter extends FragmentPagerAdapter {

/**
 * Cria um novo {@link CategoryAdapter} objeto.
 *
 * @param fm é o fragment manager que manterá cada estado de fragment no ada
 *           ao longo de cada deslizada.
 */
public CategoryAdapter(FragmentManager fm) {
    super(fm);
}

/**
 * Retorna o {@link Fragment} que deve ser mostrado para o dado número de pá
 */
@Override
public Fragment getItem(int position) {
    if (position == 0) {
        return new NumbersFragment();
    } else if (position == 1) {
        return new FamilyFragment();
    } else if (position == 2) {
        return new ColorsFragment();
    } else {
        return new PhrasesFragment();
    }
}

/**
 * Retorna o número total de páginas.
 */
@Override
public int getCount() {
    return 4;
}
```

}

6) Então na MainActivity, podemos relacionar o CategoryAdapter para alimentar o ViewPager. Delete todos os códigos previamente relacionados para os 4 TextViews de categoria.

Tudo que precisamos fazer é achar o ViewPager que foi declarado no layout XML. Então crie um novo CategoryAdapter, e sete o adapter para o ViewPager (usando o método setAdapter).

```
package com.example.android.miwok;

import android.os.Bundle;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

        // Sete o conteúdo da activity para usar o arquivo de layout activity
        setContentView(R.layout.activity_main);

        // Ache o view pager que habilitará o usuário a deslizar entre fragme
        ViewPager viewPager = (ViewPager) findViewById(R.id.viewpager);

        // Crie um adapter que saiba qual fragment deve ser mostrado em cada
        CategoryAdapter adapter = new CategoryAdapter(getSupportFragmentManag

        // Sete o adapter no view pager
        viewPager.setAdapter(adapter);
    }
}
```

7) Teste o código para se certificar que tudo funciona.

8) Se o aplicativo se comportar como esperado, delete os arquivos desnecessários:

- NumbersActivity.java
- FamilyActivity.java
- ColorsActivity.java
- PhrasesActivity.java
- Arquivo de layout activity_category.xml

Também delete as declarações de activity do arquivo AndroidManifest.xml.

9) Após todas essas deleções, teste o aplicativo NOVAMENTE para se certificar que tudo funciona e você não deletou algo importante. Certifique-se que a execução do áudio ainda funciona.

Se você quer chegar 2x seu código em relação ao nosso, esta é a **diferença** (ou delta) para o que mudou nesta tarefa de código. Este é o **estado completo** do aplicativo neste ponto.

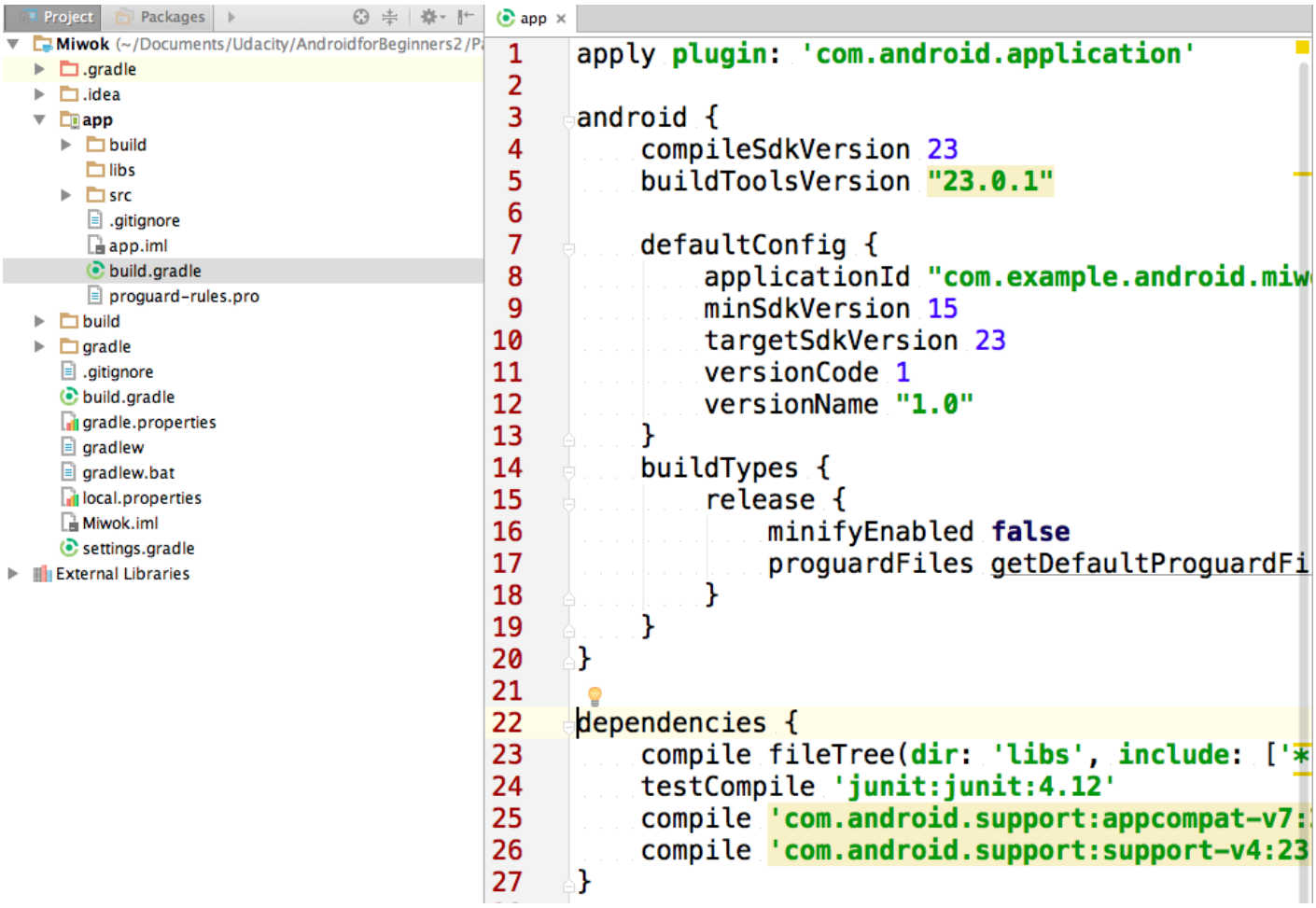
Agora vamos adicionar abas para que o usuário possa dizer que há mais páginas para deslizar.

Primeiramente você precisará usar a Biblioteca de Suporte de Design Android. Isto o ajuda a criar o aplicativo Material Design que roda até mesmo em versões antigas do Android. Aprenda mais sobre a biblioteca de suporte nesta **postagem de blog**. Há atualizações periódicas a biblioteca de suporte, então você pode checar este **site**para as últimas atualizações ou se inscrever no **blog Android Developers**.

Adicione Biblioteca de Suporte de Design Android ao Seu Projeto

1) No Android Studio, dentro da aba do Project Directory, navegue para Miwok > app > build.gradle e abra o arquivo build.gradle.

Gradle é a ferramenta que o Android Studio usa para gerar o apk (arquivo do aplicativo) que vai para o dispositivo. Para mais informações sobre configurar o arquivo build.gradle, veja este **artigo**. Há também um avançado **curso da Udacity** sobre este tópico, então não se preocupe se você não entender agora.



2) Este arquivo build.gradle contém informação sobre como o aplicativo deve ser empacotado. Vá para a seção dependencies, e adicione uma linha que inclua a Biblioteca de Suporte de Design do

Android. Isto indica que nosso aplicativo depende da Biblioteca de Design do Android, e que o código do nosso aplicativo Miwok pode agora se referir a componentes da biblioteca de design.

```
compile 'com.android.support:design:23.3.0'
```

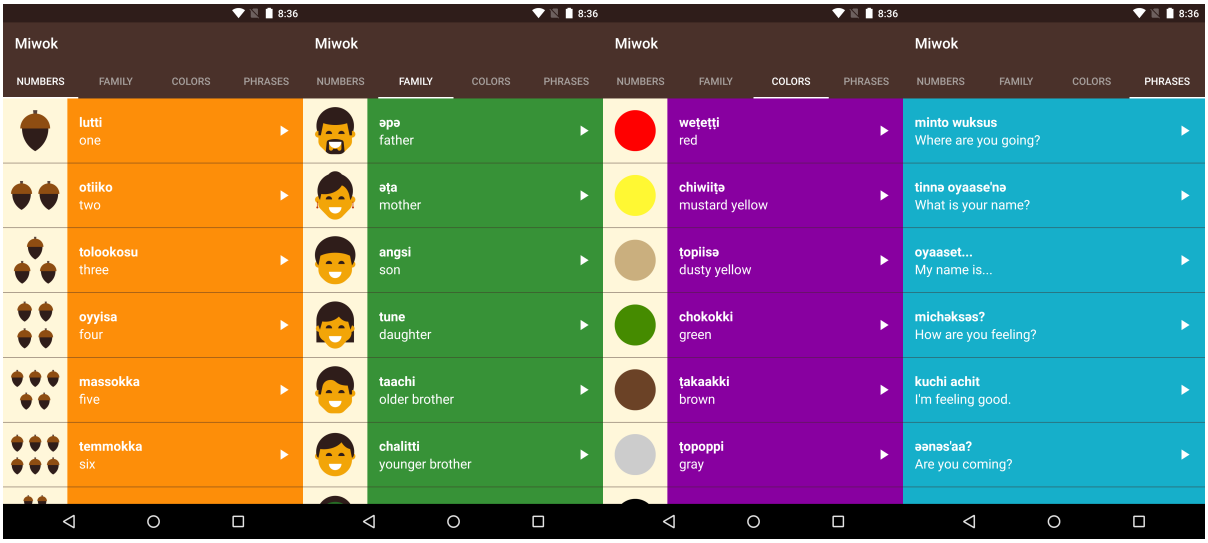
Agora, deve se parecer com isto:

```
dependencies {
   compile fileTree(dir: 'libs', include: ['*.jar'])
   testCompile 'junit:junit:4.12'
   compile 'com.android.support:appcompat-v7:23.2.1'
   compile 'com.android.support:support-v4:23.2.1'
   compile 'com.android.support:design:23.3.0'
}
```

3) Você verá um aviso amarelo no topo do seu arquivo dizendo que o arquivo Gradle mudou e uma sincronização de projeto é necessária. Clique no botão "Sync Now".

Agora você pode começar a usar componentes da Biblioteca de Suporte de Design do Android, como o TabLayout para o ViewPager!

Nesta próxima tarefa de código, modifique o aplicativo para adicionar abas ao ViewPager na MainActivity. Aqui temos como o código deve se parecer quando você tiver terminado. Você pode consultar este **tutorial Codepath** como ajuda.

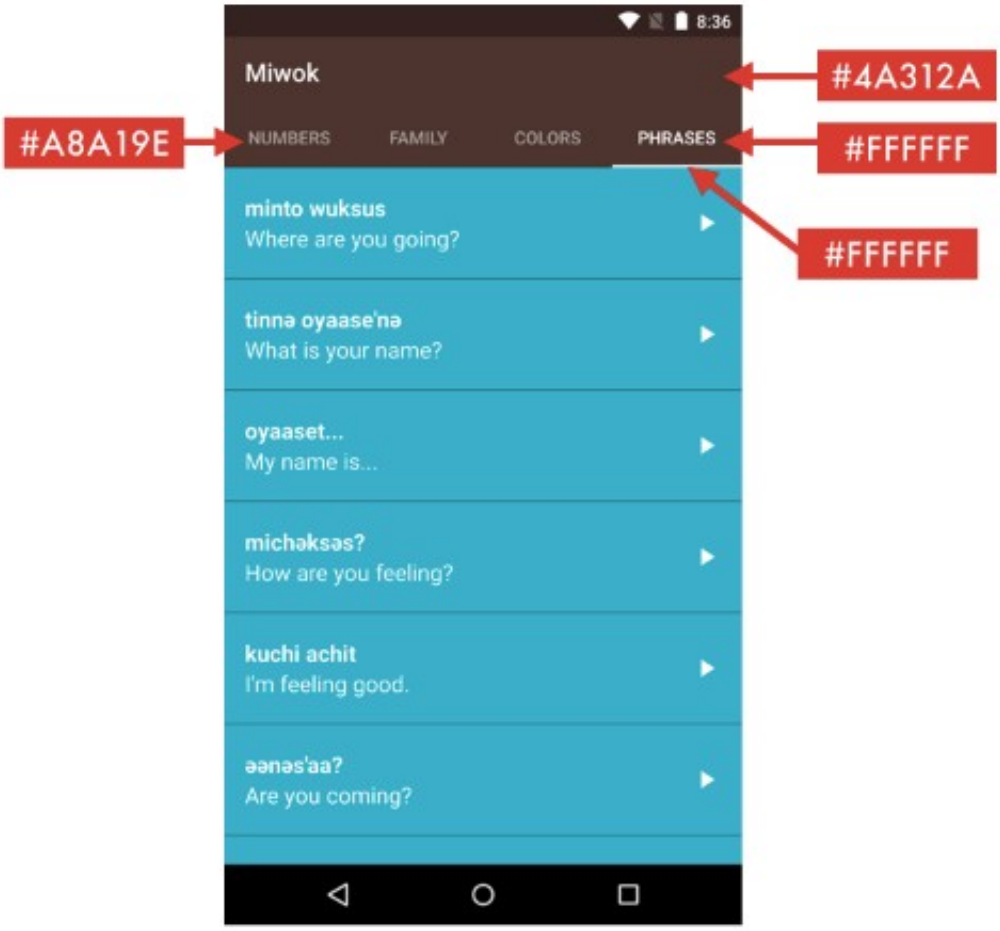

8:36
Miwok
NUMBERS
FAMILY
COLORS
PHRASES
lutti
one
otiiko
two
tolookosu
three
oyyisa
four
massokka
five
temmokka
six
əpə
father
əṭa
mother
angsi
son
tune
daughter
taachi
older brother
chalitti
younger brother
weṭeṭṭi
red
chiwiiṭə
mustard yellow
ṭopiisə
dusty yellow
chokokki
green
ṭakaakki
brown
ṭopoppi
gray
minto wuksus
Where are you going?
tinnə oyaase'nə
What is your name?
oyaaset...
My name is...
michəksəs?
How are you feeling?
kuchi achit
I'm feeling good.
əənəs'aa?
Are you coming?

Exercício

Adicione abas ao ViewPager

Siga o tutorial deixado em link para ver como adicionar abas ao ViewPager na MainActivity. Cada aba deve conter o nome da categoria.

Adicione polimento visual para bater com as diretrizes mostradas abaixo.



Estas são as mudanças necessárias para completar o quiz.

Este é o estado do código após completar o quiz.

Primeiramente, pegaremos as abas mostrando na tela. Então nos preocuparemos sobre o polimento visual para resolver as linhas vermelhas após.

1) Como a Biblioteca de Design deve ser vinculada como uma dependência no projeto, modifique o layout MainActivity para conter ViewPager e TabLayout. O arquivo de layout activity_main.xml deve ser semelhante a este. Observe que o TabLayout tem um ID de view atribuído a ele porque precisaremos nos referir à view no código Java.

O arquivo xml deve se parecer com isto:

```
<?xml version="1.0" encoding="utf-8"?>
  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     xmlns:tools="http://schemas.android.com/tools"
     android:layout_width="match_parent"
     android:layout_height="match_parent"
     android:background="@color/tan_background"
     android:orientation="vertical"
     tools:context="com.example.android.miwok.MainActivity">

     <android.support.design.widget.TabLayout
         android:id="@+id/tabs"
         android:layout_width="match_parent"
         android:layout_height="wrap_content" />

     <android.support.v4.view.ViewPager
         android:id="@+id/viewpager"
         android:layout_width="match_parent"
         android:layout_height="match_parent"/>

  </LinearLayout>
```

2) Modifique o método MainActivity onCreate() para conectar o TabLayout com o ViewPager.

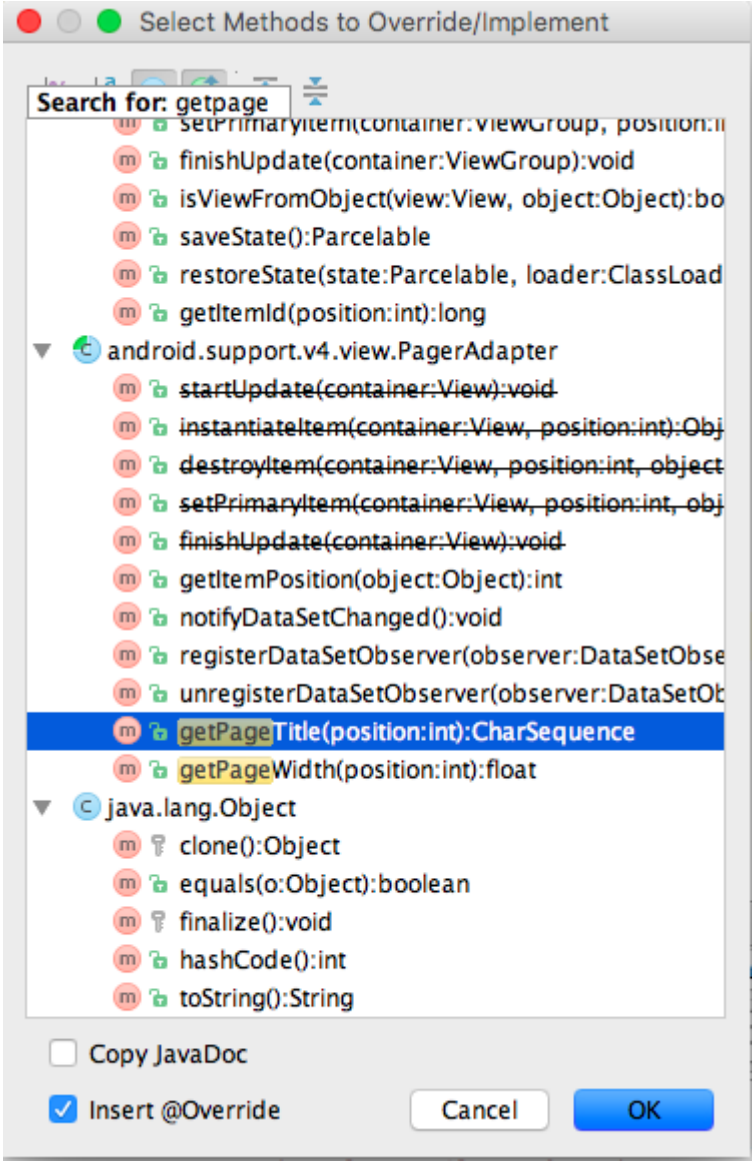
Seu código deve se parecer com isto:

```
package com.example.android.miwok;
import android.os.Bundle;
import android.support.design.widget.TabLayout;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

```
        // Sete o conteúdo da activity para usar o arquivo de layout activity
        setContentView(R.layout.activity_main);

        // Ache o view pager que habilitará o usuário a deslizar entre fragme
        ViewPager viewPager = (ViewPager) findViewById(R.id.viewpager);

        // Crie um adapter que saiba qual fragment deve ser mostrado em cada
        CategoryAdapter adapter = new CategoryAdapter(getSupportFragmentManag

        // Sete o adapter no view pager
        viewPager.setAdapter(adapter);

        // Ache o layout da aba que mostra as abas
        TabLayout tabLayout = (TabLayout) findViewById(R.id.tabs);

        // Conecte o layout da aba com o view pager. Isto irá
        //   1. Atualizar o layout da aba quando o view pager for deslizado
        //   2. Atualizar o view pager quando uma aba for selecionada
        //   3. Definir os nomes da aba do layout da aba com os títulos do ad
        //      chamando onPageTitle()
        tabLayout.setupWithViewPager(viewPager);
    }
}
```

3) Agora temos de dizer ao aplicativo o texto a ser exibido em cada guia. Vá para o arquivo CategoryAdapter.java e substitua o método getPageTitle() da classe. Este método foi originalmente definido na superclasse (o FragmentPagerAdapter), mas queremos substituir o método para personalizar o texto da guia (que é conhecido como o título da página no código).

Para sobrescrever o método, use o atalho de teclado CTRL + O. Digite "getPageTitle" e clique OK.



Android Studio irá automaticamente adicionar este método na classe CategoryAdapter.

```
@Override
public CharSequence getPageTitle(int position) {
   return super.getPageTitle(position);
}
```

4) Modifique a implementação padrão de CategoryAdapter getPageTitle(int position) para retornar o nome da categoria apropriada por página. Poderíamos retornar uma string codificada como "Números", "Família" e assim por diante. No entanto, como enfatizado no final da Lição 4, não queremos restringir o nosso aplicativo para apenas dar suporte ao idioma Inglês. Em vez disso, devemos usar o recurso de string para esses nomes de categoria.

Infelizmente, isto também significa que precisamos de um objeto Contexto a fim de tornar o ID de recurso string em uma String de verdade. Então modificamos o construtor CategoryAdapter para também requerer uma entrada de Contexto para que possamos obter o texto string apropriado.

Isto é como a classe CategoryAdapter se parece com o apropriado código de getPageTitle(int position), onde nós contamos com recursos string em vez de string hardcode codificadas no código Java.

```
package com.example.android.miwok;

import android.content.Context;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;

/**
* {@link CategoryAdapter} é um {@link FragmentPagerAdapter} que pode prover
* cada item da lista baseado em um recursos de dados que é uma lista de obj
*/
public class CategoryAdapter extends FragmentPagerAdapter {

    /** Contexto do aplicativo */
    private Context mContext;

    /**
     * Crie um novo {@link CategoryAdapter} objeto.
     *
     * @param context é o contexto do aplicativo
     * @param fm é o fragment manager que manterá cada estado do fragmento n
     */
    public CategoryAdapter(Context context, FragmentManager fm) {
        super(fm);
        mContext = context;
    }

    /**
     * Returna o {@link Fragment} que deve ser mostrado para o dado número d
     */
    @Override
    public Fragment getItem(int position) {
        if (position == 0) {
            return new NumbersFragment();
        } else if (position == 1) {
            return new FamilyFragment();
        } else if (position == 2) {
            return new ColorsFragment();
        } else {
            return new PhrasesFragment();
        }
    }

    /**
     * Retorna o número total de páginas.
     */
    @Override
    public int getCount() {
        return 4;
    }

    @Override
    public CharSequence getPageTitle(int position) {
        if (position == 0) {
```

```
            return mContext.getString(R.string.category_numbers);
        } else if (position == 1) {
            return mContext.getString(R.string.category_family);
        } else if (position == 2) {
            return mContext.getString(R.string.category_colors);
        } else {
            return mContext.getString(R.string.category_phrases);
        }
    }
}
```

5) Como modificamos o construtor CategoryAdapter, nós também precisamos atualizar a MainActivity (que usa este construtor). Quando criamos um CategoryAdapter, passamos um Contexto (que é "this" ou a activity) e o FragmentManager.

Esta linha de declaração de código deve se parecer com isto:

```
CategoryAdapter adapter = new CategoryAdapter(this, getSupportFragmentManag
```

6) Se você rodar o aplicativo agora, as abas devem ser mostradas na tela com sucesso! No entanto, não parece tão bom com o tema de cores atual do resto do aplicativo. Vamos mover no estágio de polimento visual, usando as linhas vermelhas do designer para nos guiar.

7) A primeira coisa que notamos é que o texto "Family Members" é muito longo, comparado com os outros labels das abas. No arquivo strings.xml, podemos mudar a string para Family ao invés de Family Members.

Quando você modificar strings, tenha cuidado com todos os lugares que usam essa string. Neste caso, nós só usamos a string em um lugar, então é seguro mudar.

```
<!-- Categoria de nome para o vocabulário de palavras para membros de famíl
<string name="category_family">Family</string>
```

8) Faça ajustes visuais para o aplicativo seguindo o **tutorial Codepath** que nós indicamos para você.

Modifique o arquivo activity_main.xml. Mude também o fundo do LinearLayout principal, para que as abas sejam da mesma cor da barra do aplicativo. Nós também queremos aplicar um estilo as abas, para mudar a aparência visual deles.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/primary_color"
    android:orientation="vertical"
    tools:context="com.example.android.miwok.MainActivity">

    <android.support.design.widget.TabLayout
        android:id="@+id/tabs"
        style="@style/CategoryTab"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <android.support.v4.view.ViewPager
        android:id="@+id/viewpager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

Defina o tema do layout da aba no seu arquivo styles.xml. Note que não deve ter sombra abaixo da barra do aplicativo. Veja esta **discussão do StackOverflow**.

Quando você tiver terminado, o arquivo styles.xml deve se parecer com isto:

```
<resources>

    <!-- Tema base da aplicação. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <item name="colorPrimary">@color/primary_color</item>
        <item name="colorPrimaryDark">@color/primary_dark_color</item>
        <item name="actionBarStyle">@style/MiwokAppBarStyle</item>
        <item name="android:windowContentOverlay">@null</item>
    </style>

    <!-- Estilo da barra do aplicativo -->
    <style name="MiwokAppBarStyle" parent="style/Widget.AppCompat.Light.Acti
        <!-- Remove a sombra abaixo da barra do aplicativo -->
        <item name="elevation">0dp</item>
    </style>

    <!-- Estilo para uma aba que mostra uma categoria de nome -->
    <style name="CategoryTab" parent="Widget.Design.TabLayout">
        <item name="tabIndicatorColor">@android:color/white</item>
        <item name="tabSelectedTextColor">@android:color/white</item>
        <item name="tabTextAppearance">@style/CategoryTabTextAppearance</ite
    </style>

    <!-- Aparência do estilo do texto para uma categoria de aba -->
    <style name="CategoryTabTextAppearance" parent="TextAppearance.Design.Ta
        <item name="android:textColor">#A8A19E</item>
    </style>
</resources>
```

9) Após todas estas modificações de código (você adivinhou isto), devemos rodar o aplicativo para certificar que ele se parece e funciona como esperado. :)

Se você quer checar 2x seu código em relação ao nosso, este é o **diff do código** para o que mudou neste quiz. Este é o **estado final completo do aplicativo**. (Difícil acreditar que é o fim, eu sei!)

No futuro, se você quiser aprender mais, confira o aplicativo de código-aberto **Cheesesquare** criado pelo Engenheiro Desenvolvedor de Programas **Chris Banes**. Baixe o código, importe-o no Android Studio, e rode o Cheesesquare no seu dispositivo para checar mais coisas legais que você pode fazer com a Biblioteca de Suporte de Design do Android.