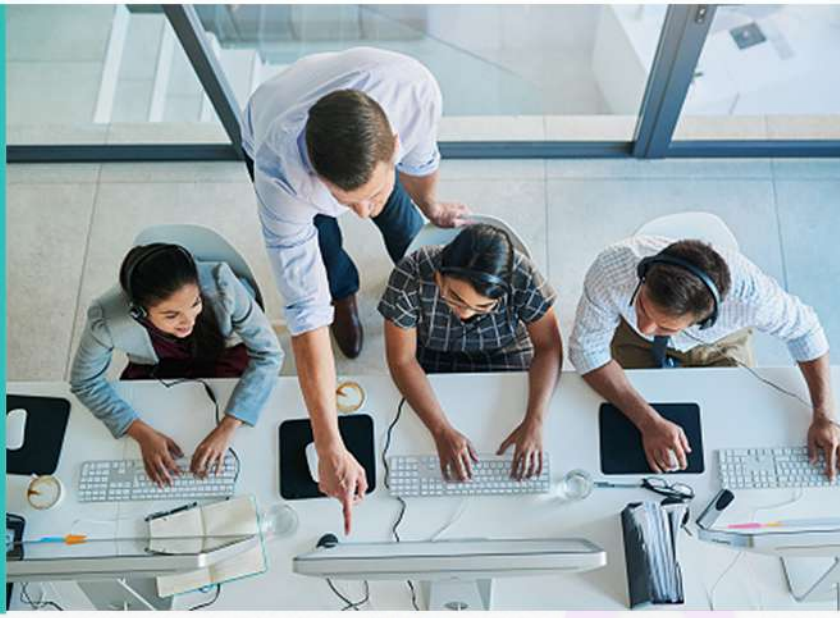
Andhra Pradesh State Skill Development Corporation



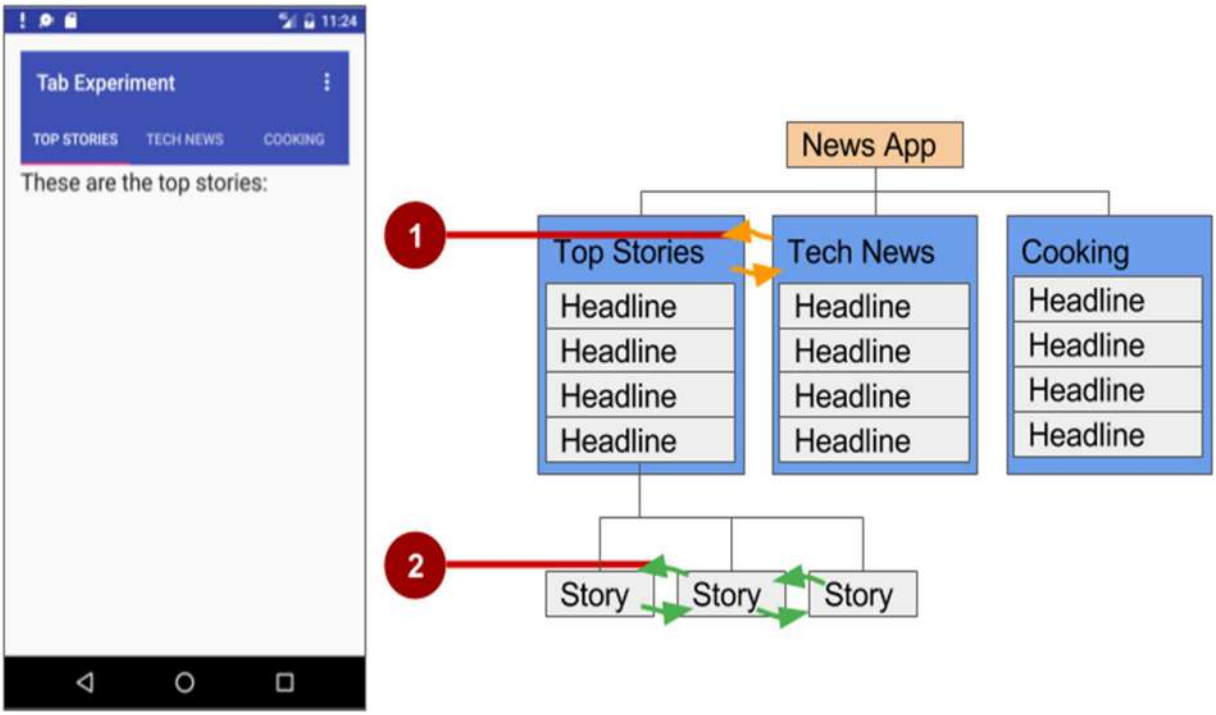
ANDROID APPLICATION DEVELOPMENT

TAB NAVIGATION

Tab Navigation

Use tab navigation with swipe views

With lateral navigation, you enable the user to go from one sibling to another (at the same level in a multitier hierarchy). For example, if your app provides several categories of stories (such as Top Stories, Tech News, and Cooking, as shown in the figure below), you would want to provide your users the ability to navigate from one category to the next, without having to navigate back up to the parent screen. Another example of lateral navigation is the ability to swipe left or right in a Gmail conversation to view a newer or older one in the same Inbox.



In the above figure:

- Lateral navigation from one category screen to another
- Lateral navigation from one story screen to another You can implement lateral navigation with tabs that represent each screen. Tabs appear across the top of a screen, as shown on the left side of the figure above, in order to provide navigation to other screens. Tab navigation is a very popular solution for lateral navigation from one child screen to another child screen that is a sibling—in the same position in the hierarchy and sharing the same parent screen. Tab navigation is often combined with the ability to swipe child screens left-to-right and right-to-left.

The primary class used for displaying tabs is TabLayout in the Android Design Support Library. It provides a horizontal layout to display tabs. You can show the tabs below the app bar, and use the PagerAdapter class to populate screens "pages" inside of a ViewPager. ViewPager is a layout manager that lets the user flip left and right through screens. This is a common pattern for presenting different screens of content within an activity—use an adapter to fill the content screen to show in the activity, and a layout manager that changes the content screens depending on which tab is selected.

You supply an implementation of a PagerAdapter to generate the screens that the view shows. ViewPager is most often used in conjunction with Fragment. By using fragments, you have a convenient way to manage the lifecycle of each screen "page".

To use classes in the Android Support Library, add com.google.android.material:material:xx.xx.x (in which xx.xx.x is the newest version) to the **build.gradle (Module: app)** file.

The following are standard adapters for using fragments with the ViewPager:

FragmentPagerAdapter: Designed for navigating between sibling screens (pages) representing a fixed, small number of screens. **FragmentStatePagerAdapter**: Designed for paging across a collection of screens (pages) for which the number of screens is undetermined. It destroys fragments as the user navigates to other screens, minimizing memory usage. The app for this practical challenge uses FragmentStatePagerAdapter.

Create the layout for tab navigation

1. Create a new project using the Empty Activity template. Name the app Tab Experiment.
2. Edit the build.gradle (Module: app) file, and add the following lines (if they are not already added) to the dependencies section:

```
implementation 'com.google.android.material:material:1.2.0'
```

If Android Studio suggests a version with a higher number, edit the above lines to update the version. Also, if Android Studio suggests a newer version of compileSdkVersion, buildToolsVersion, and/or targetSdkVersion, edit them to update the version.

3. In order to use a Toolbar rather than an action bar and app title, add the following statements to the **res > values > styles.xml** file to hide the action bar and the title:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
 ...
 <item name="windowActionBar">false</item>
 <item name="windowNoTitle">true</item>
</style>
```

4. Open the activity_main.xml layout file. In the Layout Editor, click the Text tab at the bottom of the screen and change the root view group to RelativeLayout, as you've done in previous exercises.
5. In the activity_main.xml layout, remove the TextView supplied by the template, and add a Toolbar, a TabLayout, and a ViewPager within the root layout.

As you type the app:popupTheme attribute for Toolbar as shown below, app will be in red if you didn't add the following statement to RelativeLayout:

```
<LinearLayout xmlns:app="http://schemas.android.com/apk/res-auto"
```

You can click on app and press Option-Return, and Android Studio automatically adds the statement.

Depending on your version of Android Studio, your layout code will look something like the following:

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

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <com.google.android.material.tabs.TabLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tabs"
        android:background="#5D03FD"
        app:tabTextColor="#FFFFFF"
        app:tabRippleColor="#73D305"
        app:tabSelectedTextColor="#E91E63"/>

    <androidx.viewpager.widget.ViewPager
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/pager"/>

</LinearLayout>
```

Create a layout and class for each fragment

1. Add a fragment representing each tabbed screen: (TabFragment1, TabFragment2, and TabFragment3) Fragment Names you can create your own way. To add each fragment:
   - Click com.example.android.tabexperiment(your package name, it is changing based on your system) in the project view.
   - Choose **File > New > Fragment > Fragment (Blank).**
   - Name the fragment **TabFragment1.**
   - Check the "Create layout XML?" option, and change the Fragment Layout Name for the XML file to **tab_fragment1**.
   - Uncheck the "Include fragment factory methods?" and the "include interface callbacks?" options. You don't need these methods.(Android Studio 4.0 then above versions this option is not avliable. So, you can create directly Blank Fragments)
   - Click **Finish.**
   - Repeat the above steps, using **TabFragment2 and TabFragment3** for Step C, and **tab_fragment2** and **tab_fragment3** for Step D.

Each fragment **(TabFragment1, TabFragment2, and TabFragment3)** is created with its class definition set to extend Fragment. Also, each fragment inflates the layout associated with the screen (tab_fragment1, tab_fragment2, and tab_fragment3), using the familiar resource-inflate design pattern you learned in a previous chapter with the options menu.

For example, TabFragment1 looks like this:

```
public class TabFragment1 extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
```

```
        return inflater.inflate(R.layout.tab_fragment1, container, false);
        }
}
```

Note: For Android Studio 4.0 Then Above versions The Fragment code Changed.

Android Studio automatically includes the following import statements:

```
import android.os.Bundle;
import androidx.fragment.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

2. Edit each fragment layout XML file (tab_fragment1, tab_fragment2, and tab_fragment3):
   - Change the Root Tag to RelativeLayout.
   - Add a TextView with text such as "These are the top stories".
   - Set the text appearance with android:textAppearance="?android:attr/textAppearanceLarge".
   - Repeat the above steps for each fragment layout XML file, entering different text for the TextView in step B.

3. Examine each fragment layout XML file. For example, **tab_fragment1** should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout 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"
    tools:context=".Chats">

    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="This is Chats Fragment"
        android:textSize="30sp"
        android:gravity="center"
        android:textColor="#2196F3"
        android:fontFamily="serif"
        />

</FrameLayout>
```

4. In the fragment layout XML file tab_fragment1, extract the string for "These are the top stories:" into the string resource tab_1. Do the same for the strings in tab_fragment2, and tab_fragment3.

Add a PagerAdapter

The adapter-layout manager pattern lets you provide different screens of content within an activity—use an adapter to fill the content screen to show in the activity, and a layout manager that changes the content screens depending on which tab is selected.

1. Add a new PagerAdapter class to the app that extends FragmentStatePagerAdapter and defines the number of tabs (mNumOfTabs):

```
class PagerAdapter extends FragmentPagerAdapter{
int mNumOfTabs;
   public PagerAdapter(FragmentManager fm, int NumOfTabs) {
      super(fm);
      this.mNumOfTabs = NumOfTabs;
   }
}
```

While entering the above code, Android Studio automatically imports:

```
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentPagerAdapter;
```

If FragmentManager in the above code is in red, a red lightbulb icon should appear when you click on it. Click the lightbulb icon and choose Import class. Import choices appear. Select the following import choice:

Also, Android Studio underlines the class definition for PagerAdapter and, if you click on PagerAdapter, displays a red bulb icon. Click the icon and choose Implement Methods, and then click OK to implement the already selected getItem() and getCount() methods.

2. Change the newly added getItem() method to the following, which uses a switch caseblock to return the fragment to show based on which tab is clicked

```
@Override
public Fragment getItem(int position) {
      switch (position) {
         case 0:
            return new TabFragment1();
         case 1:
            return new TabFragment2();
         case 2:
            return new TabFragment3();
         default:
            return null;
      }
}
```

3. Change the newly added getCount() method to the following to return the number of tabs:

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

Adding Titles to the Tabs

implement getpageTitle() inside Adapter class.

```
@Nullable
@Override
public CharSequence getPageTitle(int position) {
  switch (position){
    case 0:
      return "Chats";
    case 1:
      return "Status";
    case 2:
      return "Calls";
  }
  return super.getPageTitle(position);
}
```

Use PagerAdapter to manage screen views

1. Below the code you added to the onCreate() method in the previous task, add the following code to use PagerAdapter to manage screen (page) views in the fragments:

```
...
// Using PagerAdapter to manage page views in fragments.
// Each page is represented by its own fragment.
// This is another example of the adapter pattern.
final ViewPager viewPager = (ViewPager) findViewById(R.id.pager);
final PagerAdapter adapter = new PagerAdapter
        (getSupportFragmentManager(), tabLayout.getTabCount());
viewPager.setAdapter(adapter);
// Setting a listener for clicks.
...
```

2. At the end of the onCreate() method, set a listener (TabLayoutOnPageChangeListener) to detect if a tab is clicked, and create the onTabSelected() method to set the ViewPager to the appropriate tabbed screen. The code should look as follows:

```
...
// Setting a listener for clicks.
viewPager.addOnPageChangeListener(new
        TabLayout.TabLayoutOnPageChangeListener(tabLayout));
tabLayout.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
  @Override
  public void onTabSelected(TabLayout.Tab tab) {
    viewPager.setCurrentItem(tab.getPosition());
  }

  @Override
  public void onTabUnselected(TabLayout.Tab tab) {

  }

  @Override
    public void onTabReselected(TabLayout.Tab tab) {
    }
```

```
});
}
```

3. Run the app. Tap each tab to see each "page" (screen). You should also be able to swipe left and right to visit the different "pages".

OutPut: