Creating Swipe Views with Tabs

Swipe views provide lateral navigation between sibling screens such as tabs with a horizontal finger gesture (a pattern sometimes known as horizontal paging). This lesson teaches you how to create a tab layout with swipe views for switching between tabs, or how to show a title strip instead of tabs.

> **Swipe View Design**
>
> Before implementing these features, you should understand the concepts and recommendations as described in Designing Effective Navigation (/training/design-navigation/descendant-lateral.html) and the Swipe Views (/design/patterns/swipe-views.html) design guide.

THIS LESSON TEACHES YOU TO

1. Implement Swipe Views
2. Add Tabs to the Action Bar
3. Change Tabs with Swipe Views
4. Use a Title Strip Instead of Tabs

YOU SHOULD ALSO READ

- Providing Descendant and Lateral Navigation
- Android Design: Tabs
- Android Design: Swipe Views

TRY IT OUT

Download the sample app

EffectiveNavigation.zip

Implement Swipe Views

You can create swipe views in your app using the `ViewPager` `(/reference/android/support/v4/view/ViewPager.html)` widget, available in the Support Library (/tools/support-library/index.html). The `ViewPager` `(/reference/android/support/v4/view/ViewPager.html)` is a layout widget in which each child view is a separate page (a separate tab) in the layout.

To set up your layout with `ViewPager` `(/reference/android/support/v4/view/ViewPager.html)`, add a `<ViewPager>` element to your XML layout. For example, if each page in the swipe view should consume the entire layout, then your layout looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

To insert child views that represent each page, you need to hook this layout to a `PagerAdapter` `(/reference/android/support/v4/view/PagerAdapter.html)`. There are two kinds of adapter you can use:

`FragmentPagerAdapter`
: This is best when navigating between sibling screens representing a fixed, small number of pages.

`FragmentStatePagerAdapter`
: This is best for paging across a collection of objects for which the number of pages is undetermined. It destroys fragments as the user navigates to other pages, minimizing memory usage.

For example, here's how you might use `FragmentStatePagerAdapter` `(/reference/android/support/v4/app/FragmentStatePagerAdapter.html)` to swipe across a collection of `Fragment` `(/reference/android/app/Fragment.html)` objects:

```
public class CollectionDemoActivity extends FragmentActivity {
    // When requested, this adapter returns a DemoObjectFragment,
    // representing an object in the collection.
    DemoCollectionPagerAdapter mDemoCollectionPagerAdapter;
    ViewPager mViewPager;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_collection_demo);

        // ViewPager and its adapters use support library
        // fragments, so use getSupportFragmentManager.
        mDemoCollectionPagerAdapter =
                new DemoCollectionPagerAdapter(
                        getSupportFragmentManager());
        mViewPager = (ViewPager) findViewById(R.id.pager);
        mViewPager.setAdapter(mDemoCollectionPagerAdapter);
    }
}

// Since this is an object collection, use a FragmentStatePagerAdapter,
// and NOT a FragmentPagerAdapter.
public class DemoCollectionPagerAdapter extends FragmentStatePagerAdapter {
    public DemoCollectionPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int i) {
        Fragment fragment = new DemoObjectFragment();
        Bundle args = new Bundle();
        // Our object is just an integer :-P
        args.putInt(DemoObjectFragment.ARG_OBJECT, i + 1);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public int getCount() {
        return 100;
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return "OBJECT " + (position + 1);
    }
}

// Instances of this class are fragments representing a single
// object in our collection.
public static class DemoObjectFragment extends Fragment {
    public static final String ARG_OBJECT = "object";

    @Override
    public View onCreateView(LayoutInflater inflater,
            ViewGroup container, Bundle savedInstanceState) {
        // The last two arguments ensure LayoutParams are inflated
        // properly.
        View rootView = inflater.inflate(
                R.layout.fragment_collection_object, container, false);
        Bundle args = getArguments();
        ((TextView) rootView.findViewById(android.R.id.text1)).setText(
                Integer.toString(args.getInt(ARG_OBJECT)));
        return rootView;
    }
```

```
}
```

This example shows only the code necessary to create the swipe views. The following sections show how you can add tabs to help facilitate navigation between pages.

Add Tabs to the Action Bar

Action bar tabs (/design/building-blocks/tabs.html) offer users a familiar interface for navigating between and identifying sibling screens in your app.

To create tabs using `ActionBar` (/reference/android/app/ActionBar.html), you need to enable `NAVIGATION_MODE_TABS` (/reference/android/app/ActionBar.html#NAVIGATION_MODE_TABS), then create several instances of `ActionBar.Tab` (/reference/android/app/ActionBar.Tab.html) and supply an implementation of the `ActionBar.TabListener` (/reference/android/app/ActionBar.TabListener.html) interface for each one. For example, in your activity's `onCreate()` (/reference/android/app/Activity.html#onCreate(android.os.Bundle)) method, you can use code similar to this:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    final ActionBar actionBar = getActionBar();
    ...

    // Specify that tabs should be displayed in the action bar.
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

    // Create a tab listener that is called when the user changes tabs.
    ActionBar.TabListener tabListener = new ActionBar.TabListener() {
        public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {
            // show the given tab
        }

        public void onTabUnselected(ActionBar.Tab tab, FragmentTransaction ft) {
            // hide the given tab
        }

        public void onTabReselected(ActionBar.Tab tab, FragmentTransaction ft) {
            // probably ignore this event
        }
    };

    // Add 3 tabs, specifying the tab's text and TabListener
    for (int i = 0; i < 3; i++) {
        actionBar.addTab(
                actionBar.newTab()
                        .setText("Tab " + (i + 1))
                        .setTabListener(tabListener));
    }
}
```

How you handle the `ActionBar.TabListener` (/reference/android/app/ActionBar.TabListener.html) callbacks to change tabs depends on how you've constructed your content. But if you're using fragments for each tab with `ViewPager` (/reference/android/support/v4/view/ViewPager.html) as shown above, the following section shows how to switch between pages when the user selects a tab and also update the selected tab when the user swipes between pages.

Change Tabs with Swipe Views

To switch between pages in a `ViewPager` `(/reference/android/support/v4/view/ViewPager.html)` when the user selects a tab, implement your `ActionBar.TabListener` `(/reference/android/app/ActionBar.TabListener.html)` to select the appropriate page by calling `setCurrentItem()` `(/reference/android/support/v4/view/ViewPager.html#setCurrentItem(int))` on your `ViewPager` `(/reference/android/support/v4/view/ViewPager.html)`:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...

    // Create a tab listener that is called when the user changes tabs.
    ActionBar.TabListener tabListener = new ActionBar.TabListener() {
        public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {
            // When the tab is selected, switch to the
            // corresponding page in the ViewPager.
            mViewPager.setCurrentItem(tab.getPosition());
        }
        ...
    };
}
```

Likewise, you should select the corresponding tab when the user swipes between pages with a touch gesture. You can set up this behavior by implementing the `ViewPager.OnPageChangeListener` `(/reference/android/support/v4/view/ViewPager.OnPageChangeListener.html)` interface to change the current tab each time the page changes. For example:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    ...

    mViewPager = (ViewPager) findViewById(R.id.pager);
    mViewPager.setOnPageChangeListener(
            new ViewPager.SimpleOnPageChangeListener() {
                @Override
                public void onPageSelected(int position) {
                    // When swiping between pages, select the
                    // corresponding tab.
                    getActionBar().setSelectedNavigationItem(position);
                }
            });
    ...
}
```

Use a Title Strip Instead of Tabs

If you don't want to include action bar tabs and prefer to provide scrollable tabs (/design/building-blocks/tabs.html#scrollable) for a shorter visual profile, you can use `PagerTitleStrip` `(/reference/android/support/v4/view/PagerTitleStrip.html)` with your swipe views.

Below is an example layout XML file for an activity whose entire contents are a `ViewPager` `(/reference/android/support/v4/view/ViewPager.html)` and a top-aligned `PagerTitleStrip` `(/reference/android/support/v4/view/PagerTitleStrip.html)` inside it. Individual pages (provided by the adapter) occupy the remaining space inside the `ViewPager` `(/reference/android/support/v4/view/ViewPager.html)`.

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.view.PagerTitleStrip
        android:id="@+id/pager_title_strip"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#33b5e5"
        android:textColor="#fff"
        android:paddingTop="4dp"
        android:paddingBottom="4dp" />

</android.support.v4.view.ViewPager>
```