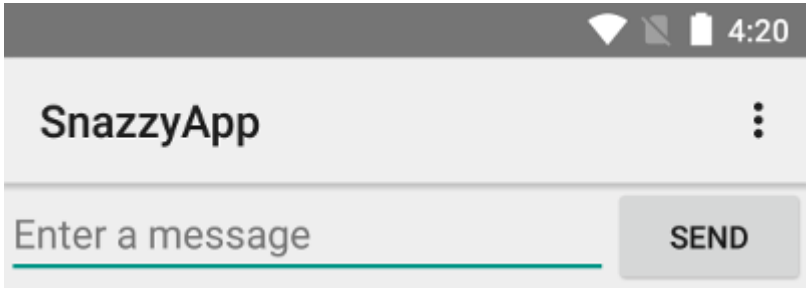
IT – Entrepreneurship
Fragments and ActionBars

ActionBar

- The action bar displays the title for the activity on one side and an overflow menu on the other
- Beginning with Android 3.0 (API level 11), all activities that use the default theme have an ActionBar as an app bar.
- However, app bar features have gradually been added to the native ActionBar



ActionBar

- As a result, the native ActionBar behaves differently depending on what version of the Android system a device may be using.
- By contrast, the most recent features are added to the support library's version of Toolbar.
- For this reason, you should use the support library's Toolbar class to implement your activities' app bars.

Toolbar

- Using the support library's toolbar helps ensure that your app will have consistent behavior across the widest range of devices.
- For example, the Toolbar widget provides a material design experience on devices running Android 2.1 or later
- But the native action bar doesn't support material design unless the device is running Android 5.0 or later.

Add a Toolbar to an Activity

- Make sure the project uses appcompat
- The activity should extend AppCompatActivity
- Change the theme to android:theme="@style/Theme.AppCompat.Light.NoActionBar"

Add a Toolbar to an Activity

- Add a Toolbar to the activity’s layout.

```
<android.support.v7.widget.Toolbar
   android:id="@+id/my_toolbar"
   android:layout_width="match_parent"
   android:layout_height="?attr/actionBarSize"
   android:background="?attr/colorPrimary"
   android:elevation="4dp"
   android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
   app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

Add a Toolbar to an Activity

- In the activity's onCreate() method, call the activity's setSupportActionBar() method, and pass the activity's toolbar.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_my);
    Toolbar myToolbar = (Toolbar) findViewById(R.id.my_toolbar);
    setSupportActionBar(myToolbar);
    }
```

Adding and Handling Actions

- The app bar allows you to add buttons for user actions.
- This feature lets you put the most important actions for the current context right at the top of the app.
- Space in the app bar is limited.
- If an app declares more actions than can fit in the app bar, the app bar send the excess actions to an overflow menu.

Adding and Handling Actions

- The app can also specify that an action should always be shown in the overflow menu, instead of being displayed on the app bar.
- All action buttons and other items available in the action overflow are defined in an XML menu resource.
- Add an <item> element for each item you want to include in the action bar

Adding and Handling Actions

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <!-- "Mark Favorite", should appear as action button if possible -->
    <item
        android:id="@+id/action_favorite"
        android:icon="@drawable/ic_favorite_black_48dp"
        android:title="@string/action_favorite"
        app:showAsAction="ifRoom"/>

    <!-- Settings, should always be in the overflow -->
    <item android:id="@+id/action_settings"
          android:title="@string/action_settings"
          app:showAsAction="never"/>

</menu>
```

Respond to Actions

- When the user selects one of the app bar items, the system calls your activity's onOptionsItemSelected()
- and passes a MenuItem object to indicate which item was clicked.

Respond to Actions

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_settings:
            // User chose the "Settings" item, show the app settings UI...
            return true;

        case R.id.action_favorite:
            // User chose the "Favorite" action, mark the current item
            // as a favorite...
            return true;

        default:
            // If we got here, the user's action was not recognized.
            // Invoke the superclass to handle it.
            return super.onOptionsItemSelected(item);

    }
}
```

Adding an Up Action

- Your app should make it easy for users to find their way back to the app's main screen.
- One simple way to do this is to provide an Up button on the app bar for all activities except the main one.
- When the user selects the Up button, the app navigates to the parent activity.

Adding an Up Action

- To support the up functionality in an activity, you need to declare the activity's parent.

```
<!-- A child of the main activity -->
    <activity
        android:name="com.example.myfirstapp.MyChildActivity"
        android:label="@string/title_activity_child"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >

        <!-- Parent activity meta-data to support 4.0 and lower -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
```

Enable the Up Button

- To enable the Up button for an activity that has a parent activity, call the app bar's setDisplayHomeAsUpEnabled() method.

```
// my_child_toolbar is defined in the layout file
    Toolbar myChildToolbar =
        (Toolbar) findViewById(R.id.my_child_toolbar);
    setSupportActionBar(myChildToolbar);

    // Get a support ActionBar corresponding to this toolbar
    ActionBar ab = getSupportActionBar();

    // Enable the Up button
    ab.setDisplayHomeAsUpEnabled(true);
```

What The Fragments?

- Why fragments? Because This!
- Why not only use fragments? Because This!

Create a Fragment Class

- To create a fragment, extend the Fragment class, then override key lifecycle methods similar to the way you would with an Activity class.
- One difference when creating a Fragment is that you must use the onCreateView() callback to define the layout.

Create a Fragment Class

```
public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

Add a Fragment to an Activity using XML

- Each instance of a Fragment class must be associated with a parent FragmentActivity.
- You can achieve this association by defining each fragment within your activity layout XML file.
- If you're using appcompat, your activity should instead extend AppCompatActivity, which is a subclass of FragmentActivity.

Add a Fragment to an Activity using XML

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <fragment android:name="com.example.android.fragments.HeadlinesFragment"
            android:id="@+id/headlines_fragment"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />

    <fragment android:name="com.example.android.fragments.ArticleFragment"
            android:id="@+id/article_fragment"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />

</LinearLayout>
```

Fragment Basic

- Evaluate the FragmentBasics project to better understand this concept.

Building a Flexible UI

Tabbed Toolbar

- Check gradle to include

```
compile 'com.android.support:appcompat-v7:23.0.1'
compile 'com.android.support:design:23.0.1'
```

- For full width tabs, app:tabGravity=”fill”
- For center aligned tabs, app:tabGravity=”center”
- For scrollable tabs, app:tabMode=”scrollable”
- To set icon to tab, tabLayout.getTabAt(0).setIcon(tabIcons[0]);

Demo

References

- developer.android.com
- cs.dartmouth.edu