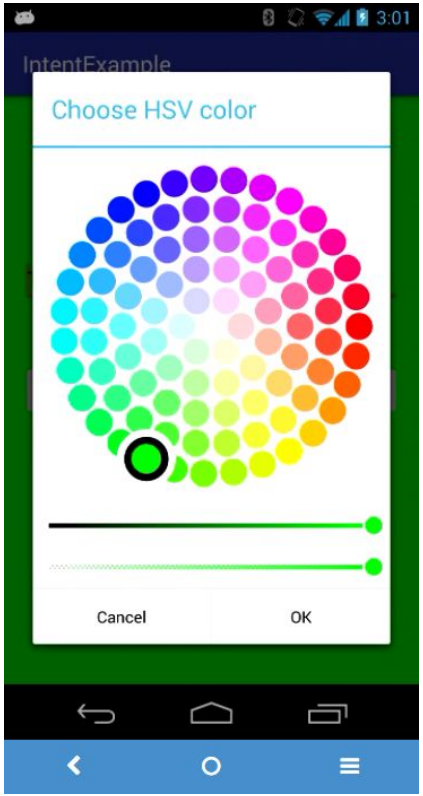
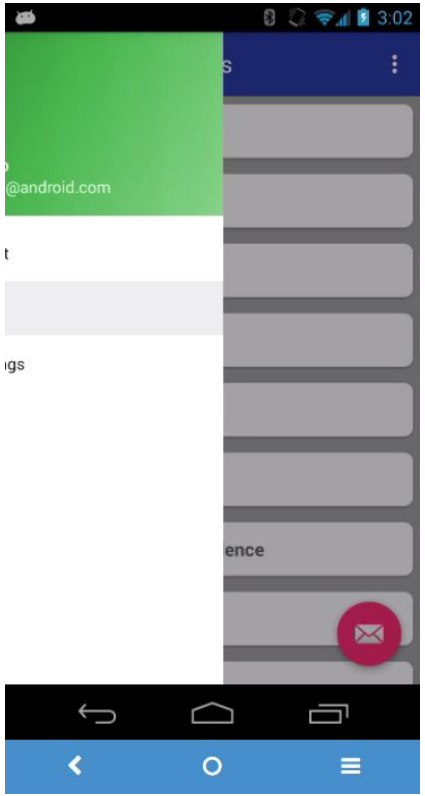
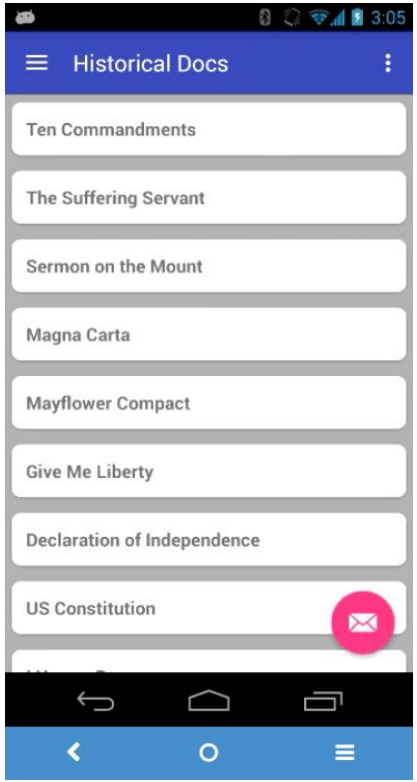
Activities and Fragments





By the end of this unit you should be able to...

- Build apps that use multiple screens
- Build apps that pass data between activities
- Launch system Activities using standard intents
- Create fragment views and controllers
- Add a fragment at runtime
- Manage the Fragment backstack
- Write callbacks to let fragments communicate with activities and other fragments



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Starting code

First download the ColorChanger starting app from

https://github.com/AndroidCourseMaterial/ColorChooser

TODO:

Unzip it and then open it in Studio.

Update if asked

Note: it uses a cool 3rd party color chooser:

https://android-arsenal.com/details/1/1693
which is included in the build.gradle file

Activity

In this lesson you will learn about Activities

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

An Activity is a single screen

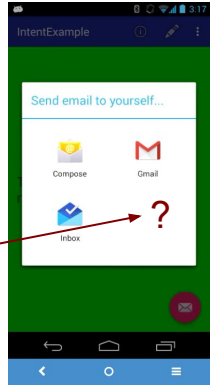
Pretty simple, huh?

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

If your app has multiple activities, which gets launched?

The intent filter in your AndroidManifest.xml file

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.rosehulman.boutell.colorchooser">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="ColorChooser"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="ColorChooser"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".InputActivity" />
    </application>

</manifest>
```



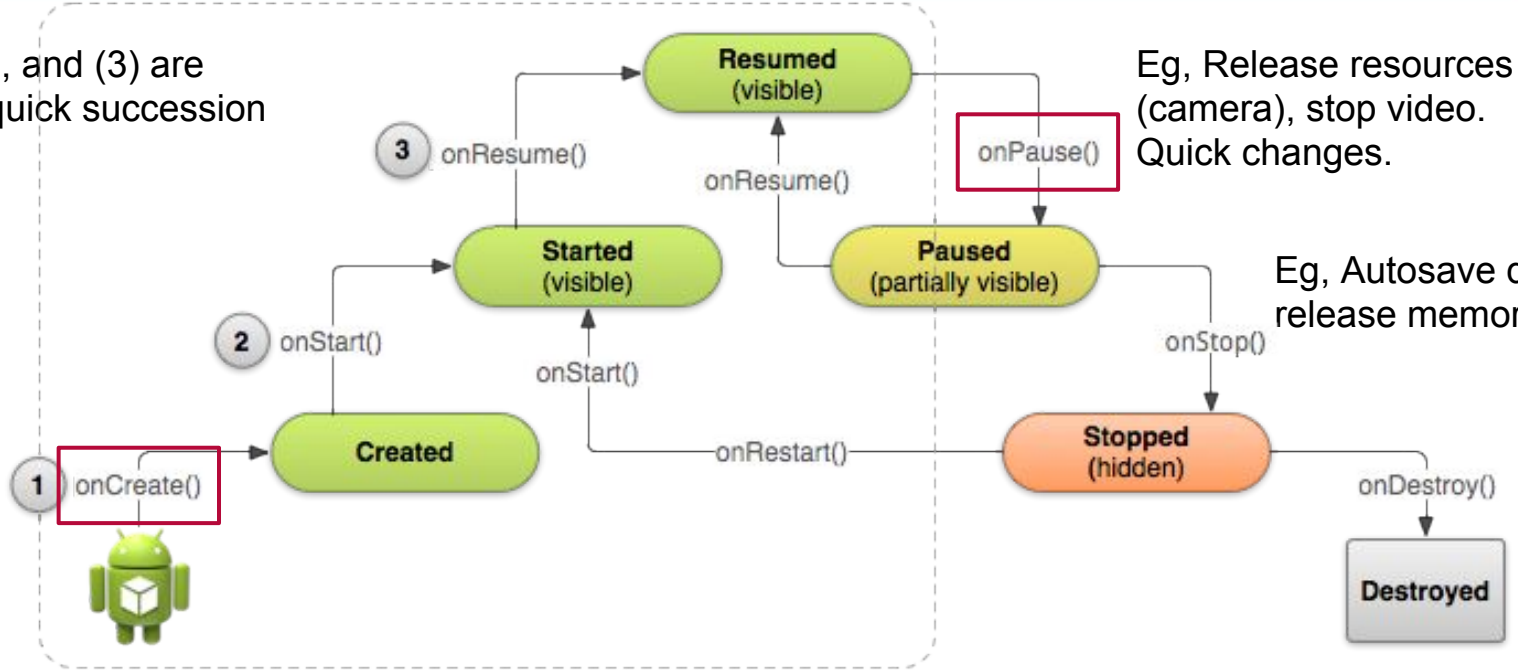
Tip for later: by setting intent filters appropriately, you can let other apps call your activity!

http://developer.android.com/training/basics/intents/filters.html

You could write an email app that people could choose over Gmail!

Activities are managed according to the Activity lifecycle

Steps (1), (2), and (3) are executed in quick succession on startup



Eg, Release resources (camera), stop video. Quick changes.

Eg, Autosave data, release memory

Important Details (essential reading sometime this week):
http://developer.android.com/training/basics/activity-lifecycle/recreating.html

http://developer.android.com/images/training/basics/basic-lifecycle-create.png

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

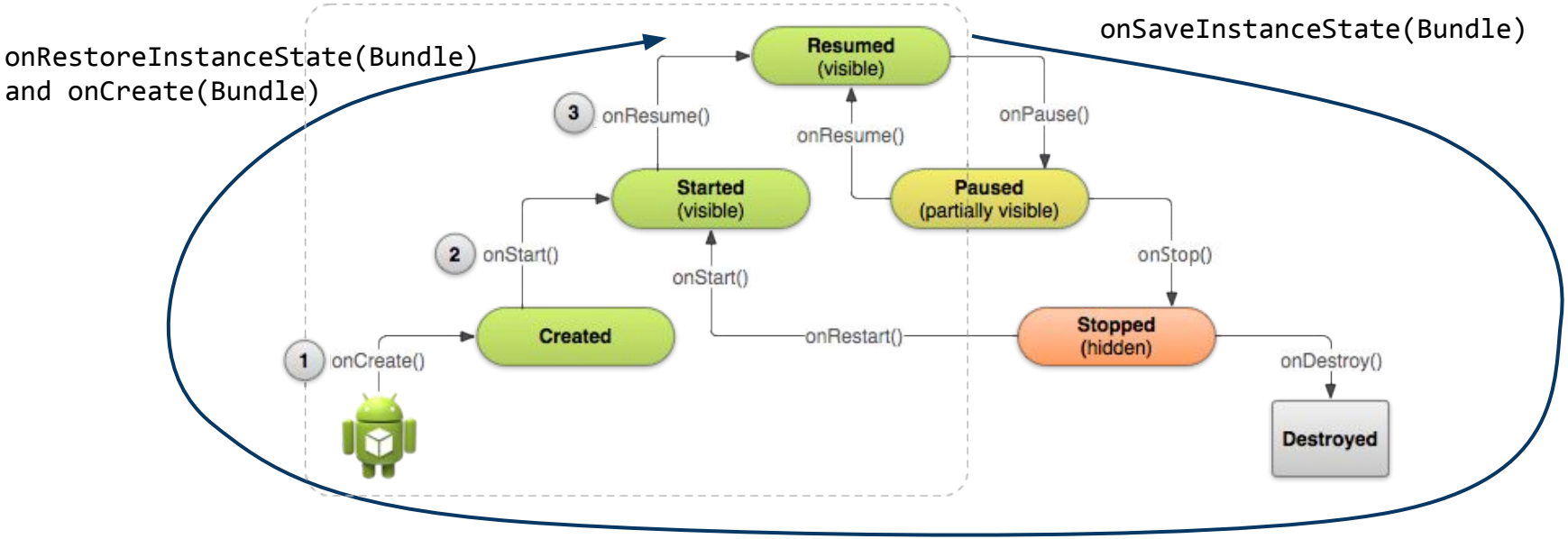
React to changes in the lifecycle using callbacks

The two most important ones:

onCreate(): when it is first launched
onPause(): to save persistent data. If your app is partially obscured, it could get destroyed by the system

Save data using SharedPreferences, SQLite, etc.

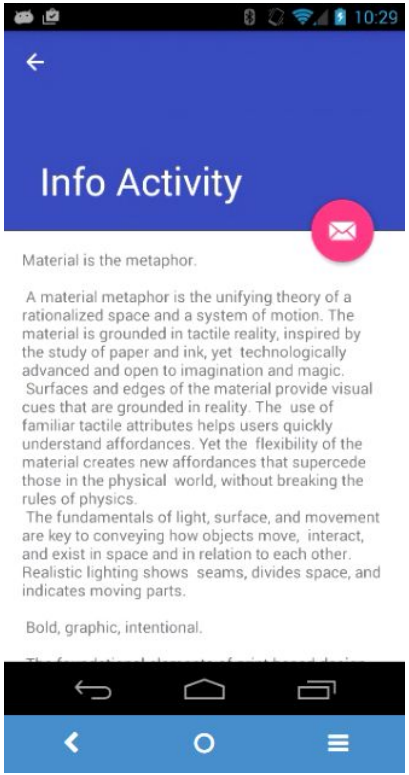
InstanceState: an easier alternative for rotation



http://developer.android.com/training/basics/activity-lifecycle/recreating.html

Creating an Activity

In this lesson you will learn how to create a new Activity and launch it from code



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

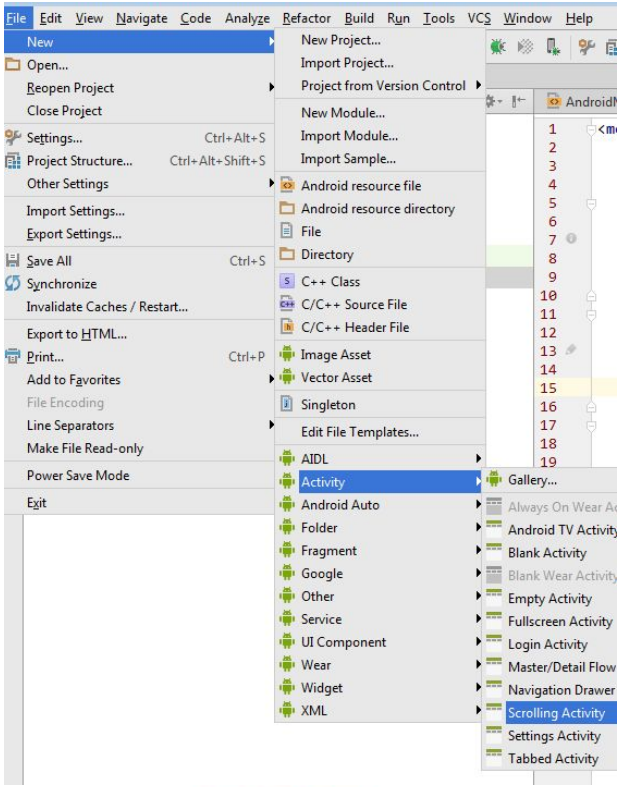
Use the New > Activity wizard

Just like when you create a new project. It will do 3 steps:

1. Create a new Java **class** that extends Activity
   - Overrides the **onCreate** method that sets the content view to the layout

2. Create a **layout** file for the new Activity
   - Add appropriate **resources** (strings, colors, etc)

3. Register the Activity in **AndroidManifest**.xml



Create a new **Scrolling Activity** now:

Name: **InfoActivity**

Hierarchical Parent: **MainActivity**.

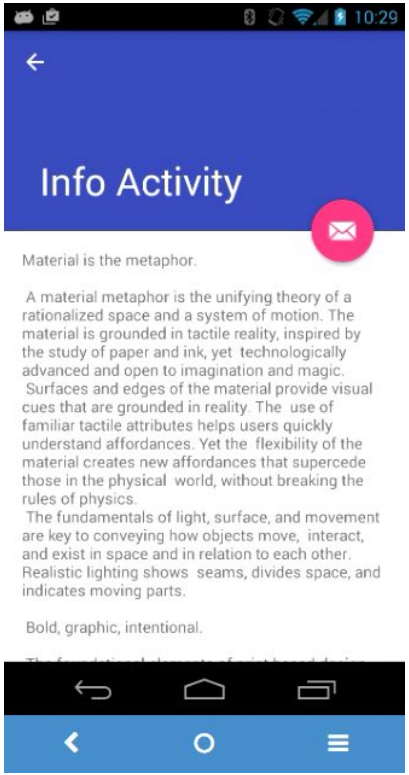
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

How to test the new Activity in isolation? Move the intent-filter to it.

MAIN means it is the main entry point and expects no starting data.

LAUNCHER tells it to use its icon in the launcher app.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="edu.rosehulman.boutell.colorchooser">

   <application
       android:allowBackup="true"
       android:icon="@mipmap/ic_launcher"
       android:label="@string/app_name"
       android:supportsRtl="true"
       android:theme="@style/AppTheme">
       <activity
           android:name=".MainActivity"
           android:label="@string/app_name"
           android:theme="@style/AppTheme.NoActionBar">
       </activity>
       <activity android:name=".InputActivity" />
       <activity
           android:name=".InfoActivity"
           android:label="@string/title_activity_info"
           android:parentActivityName=".MainActivity"
           android:theme="@style/AppTheme.NoActionBar">
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />

               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
           <meta-data
               android:name="android.support.PARENT_ACTIVITY"
               android:value="edu.rosehulman.boutell.colorchooser.MainActivity" />
       </activity>
   </application>

</manifest>
```



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

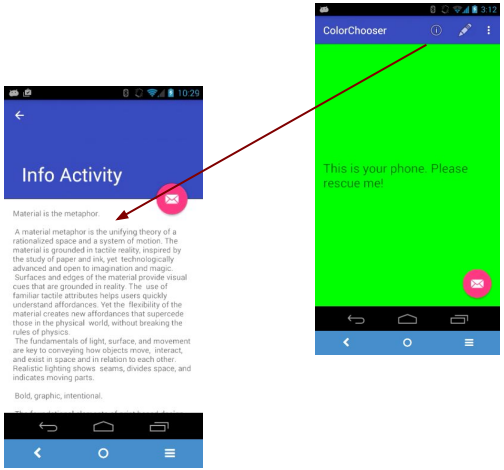
To launch the activity from another, we need a 4th step

4. Start the Activity from another Activity
    - Create an **Intent**
    - Start **Activity** using the Intent
    - If appropriate, communicate info back & forth using **extras**

Step 4: To launch an Activity, you need to create an Intent

An intent is an abstract description of an operation to be performed.

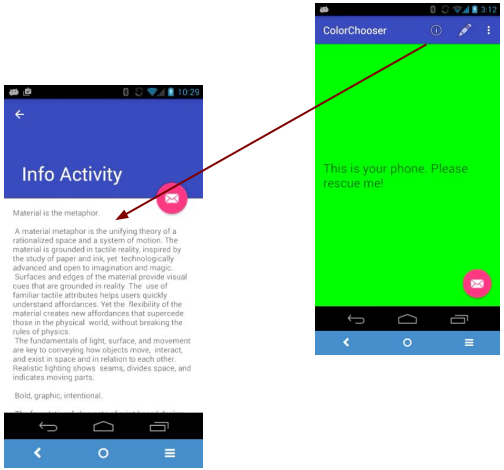
An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities...



It can be used with `startActivity` to launch an `Activity`, `broadcastIntent` to send it to any interested `BroadcastReceiver` components, and `startService(Intent)` or `bindService(Intent, ServiceConnection, int)` to communicate with a background `Service`.

http://developer.android.com/reference/android/content/Intent.html

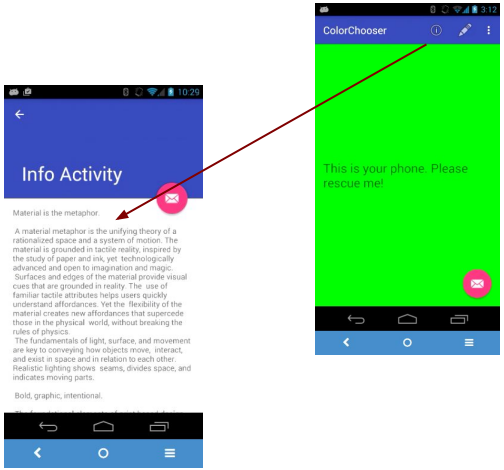
Create an intent for the AboutActivity and use it to start the activity




```java
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {

        case R.id.action_info:
            // TODO: Launch a new Info Activity that is a ScrollingActivity.
            Intent infoIntent = new Intent(this, InfoActivity.class);
            startActivity(infoIntent);
            return true;

        ...
```

Try the back button.
MainActivity is on the back stack.
MainActivity had been *paused*.

Navigating back to the first activity



1. Use the back button.
   MainActivity is on the back stack.
   MainActivity had been *paused*.

2. Set a working up arrow **in the toolbar to navigate back**.

For the arrow, add to child activity's onCreate():

```
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

To tell **where "up" goes,** add to the child activity's tag in the AndroidManifest:

```
android:parentActivityName=".MainActivity"
```

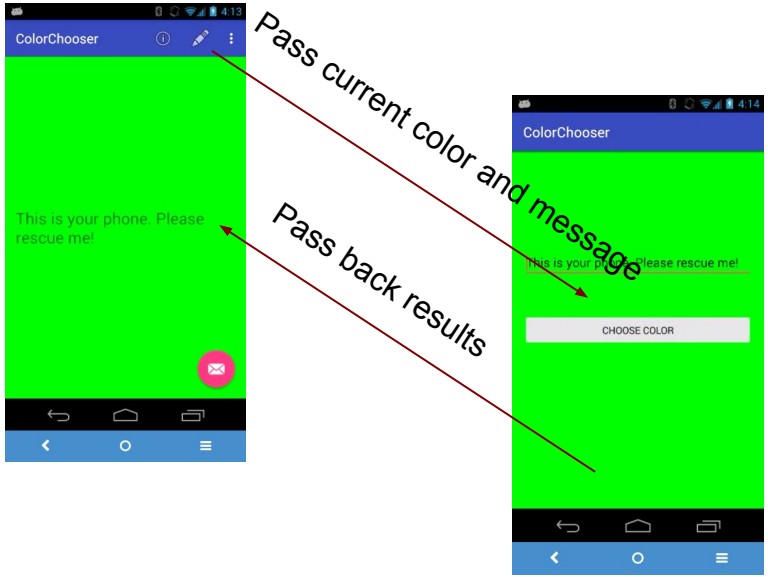
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Use Intent extras to pass info between Activities

In this lesson, you'll learn about:

1. putExtra, get*Type*Extra

2. startActivityForResult, onActivityResult

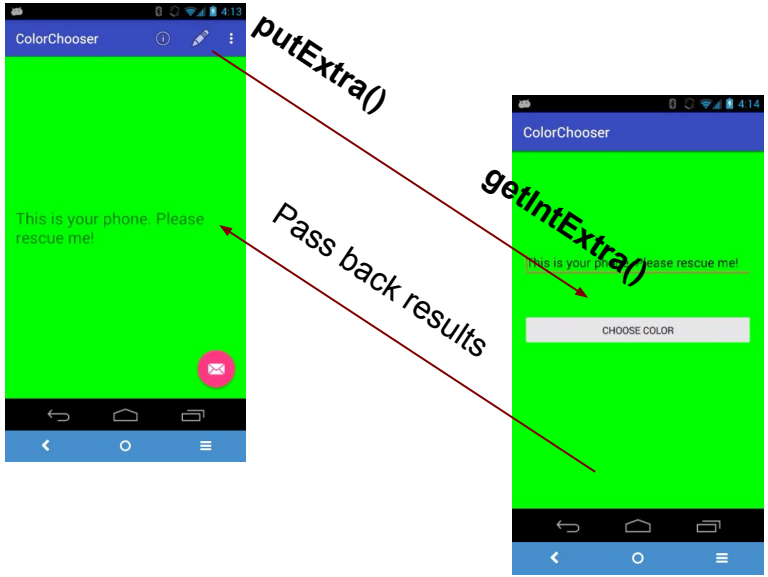


ROSE-HULMAN INSTITUTE OF TECHNOLOGY

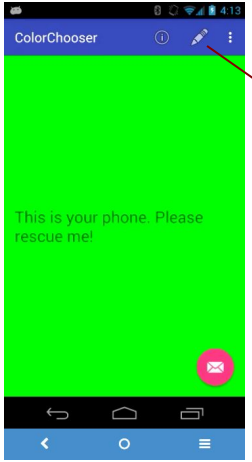
Pass data using extras in Intents

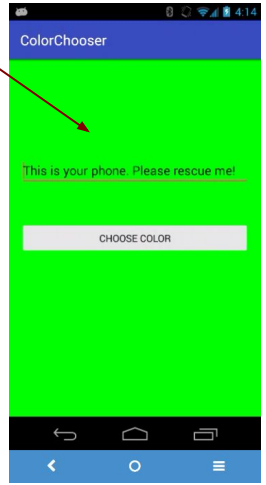
Extras are key-value pairs

Key	Value
`KEY_NUM_BUTTONS`	mNumButtons



In MainActivity, create keys and values for the color and message and add to the Intent






```
public class MainActivity extends AppCompatActivity {
   public static final String EXTRA_MESSAGE = "EXTRA_MESSAGE";
   public static final String EXTRA_COLOR = "EXTRA_COLOR";

   private RelativeLayout mLayout;


   ...

      @Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {

      ...
      case R.id.action_change_color:
         // TODO: Launch the InputActivity to get a result
         Intent inputIntent = new Intent(this, InputActivity.class);
         inputIntent.putExtra(EXTRA_MESSAGE, mMessage);
         inputIntent.putExtra(EXTRA_COLOR, mBackgroundColor);
         startActivity(inputIntent);
         return true;
```

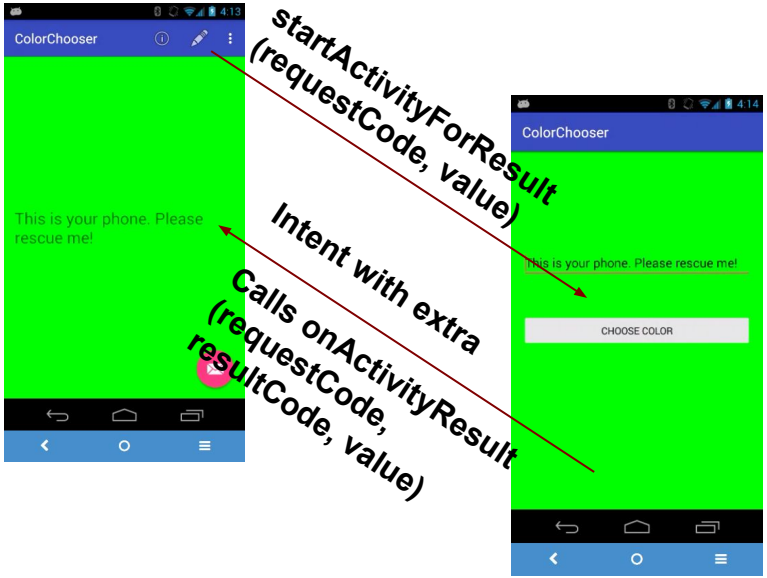
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

In the receiving activity, get the extras from the Intent

```
public class InputActivity extends AppCompatActivity {

    private RelativeLayout mLayout;
    private EditText mEditText;
    private int mCurrentBackgroundColor;
    private String mMessage;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_input);
        mLayout = (RelativeLayout) findViewById(R.id.activity_input_layout);
        mEditText = (EditText) findViewById(R.id.activity_input_message);

        Intent intent = getIntent();
        mMessage = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
        mCurrentBackgroundColor = intent.getIntExtra(MainActivity.EXTRA_COLOR, Color.GRAY);
        updateUI();
```

The last parameter is a default if none were passed.

Passing data back uses a callback



MainActivity start the input activity (ForResult)

```
public class MainActivity extends AppCompatActivity {
   public static final String EXTRA_MESSAGE = "EXTRA_MESSAGE";
   public static final String EXTRA_COLOR = "EXTRA_COLOR";
   private static final int REQUEST_CODE_INPUT = 1;

   private RelativeLayout mLayout;
   ...
   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
       switch (item.getItemId()) {


           case R.id.action_change_color:
               // TODO: Launch the InputActivity to get a result
               Intent inputIntent = new Intent(this, InputActivity.class);
               inputIntent.putExtra(EXTRA_MESSAGE, mMessage);
               inputIntent.putExtra(EXTRA_COLOR, mBackgroundColor);
               startActivityForResult(inputIntent, REQUEST_CODE_INPUT);
               return true;
```

Requires a *request code* to distinguish between multiple activities it could call. Used in the callback we'll write

InputActivity: Create an Intent with the result and the OK flag. Then tell this Activity to finish

```
// From https://android-arsenal.com/details/1/1693
private void showColorDialog() {
   ColorPickerDialogBuilder
      ...
      .setPositiveButton(getString(android.R.string.ok), new ColorPickerClickListener() {
              @Override
              public void onClick(DialogInterface dialog, int selectedColor, Integer[] allColors) {
                  mCurrentBackgroundColor = selectedColor;
                  mMessage = mEditText.getText().toString();

                  updateUI();
                  Intent returnIntent = new Intent();
                  returnIntent.putExtra(MainActivity.EXTRA_MESSAGE, mMessage);
                  returnIntent.putExtra(MainActivity.EXTRA_COLOR, mCurrentBackgroundColor);
                  setResult(Activity.RESULT_OK, returnIntent);
                  finish();
              }
          })
```

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

MainActivity's onActivityResult() is called when startActivityForResult finishes

```
protected void onActivityResult(int requestCode, int resultCode, Intent data){

    // See which child activity is calling us back.
    switch (requestCode) {
        case REQUEST_CODE_INPUT:
            if (resultCode == Activity.RESULT_OK){
                Log.d("COLOR" , "Result ok!");
            }
            else {
                Log.d("COLOR", "Result not okay.  User hit back without a button");
            }
            break;
        default:
            Log.d("COLOR", "Unknown result code");
            break;
    }
}
```

Get and use the extras from the intent

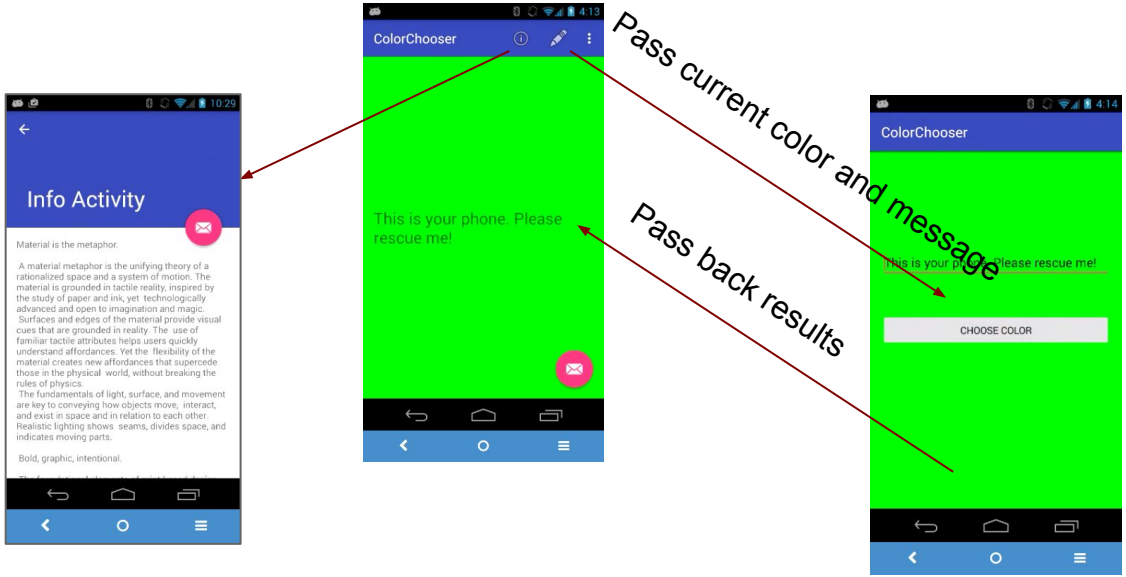
```
protected void onActivityResult(int requestCode, int resultCode, Intent data){
    switch (requestCode) {
        case REQUEST_CODE_INPUT:
            if (resultCode == Activity.RESULT_OK){
                Log.d("COLOR", "Result ok!");
                mMessage = data.getStringExtra(EXTRA_MESSAGE);
                mBackgroundColor = data.getIntExtra(EXTRA_COLOR, Color.GRAY);
                updateUI();

            }
            else {
                Log.d("COLOR", "Result not okay.  User hit back without a button");
            }
            break;
        default:
        Log.d("COLOR", "onActivityResult");
            break;
    }
}
```

Hitting back button will give **Activity.RESULT_CANCELLED**

Summary:
Use intents to launch activities, and extras to send info

Note: extra types can be primitives, Strings, parcelable objects, and arrays of these

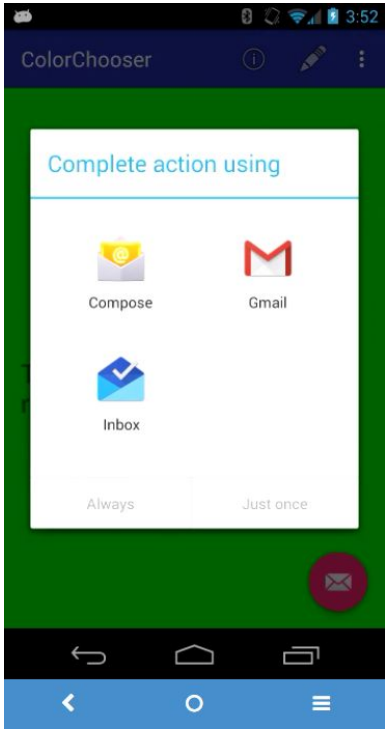
Read this [excellent resource]() for understanding when onPause() etc called when one activity launches another.



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

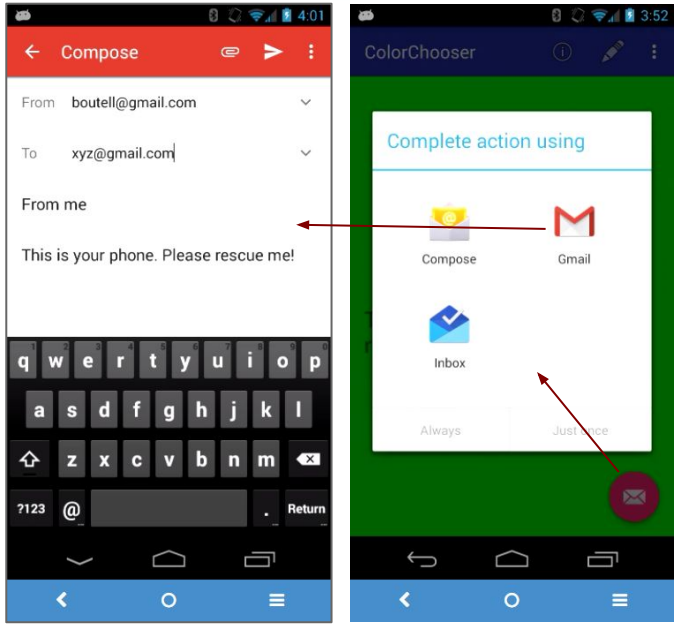
Using system intents

In this lesson you will learn how to launch system activities using intents



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Example: an intent to compose an email




```
Write and call this method when the FAB is pressed:
import android.net.Uri;
private void sendEmail() {
    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setData(Uri.parse("mailto:"));
    intent.putExtra(Intent.EXTRA_EMAIL, new String[] {"x@gmail.com"});
    intent.putExtra(Intent.EXTRA_SUBJECT, "From me");
    intent.putExtra(Intent.EXTRA_TEXT, mMessage);
    if (intent.resolveActivity(getPackageManager()) != null) {
        startActivity(intent);
    }
    // or startActivity(Intent.createChooser(emailIntent, "Send email
to yourself..."));
}
```

Email needs to be an array

It will let you choose your email client if you have more than 1.

If you test on an emulator, you'll need to go into the emulator's email app and set up your email address. Otherwise, you'll get: "Unsupported action. That action is not currently supported".

http://developer.android.com/guide/components/intents-common.html

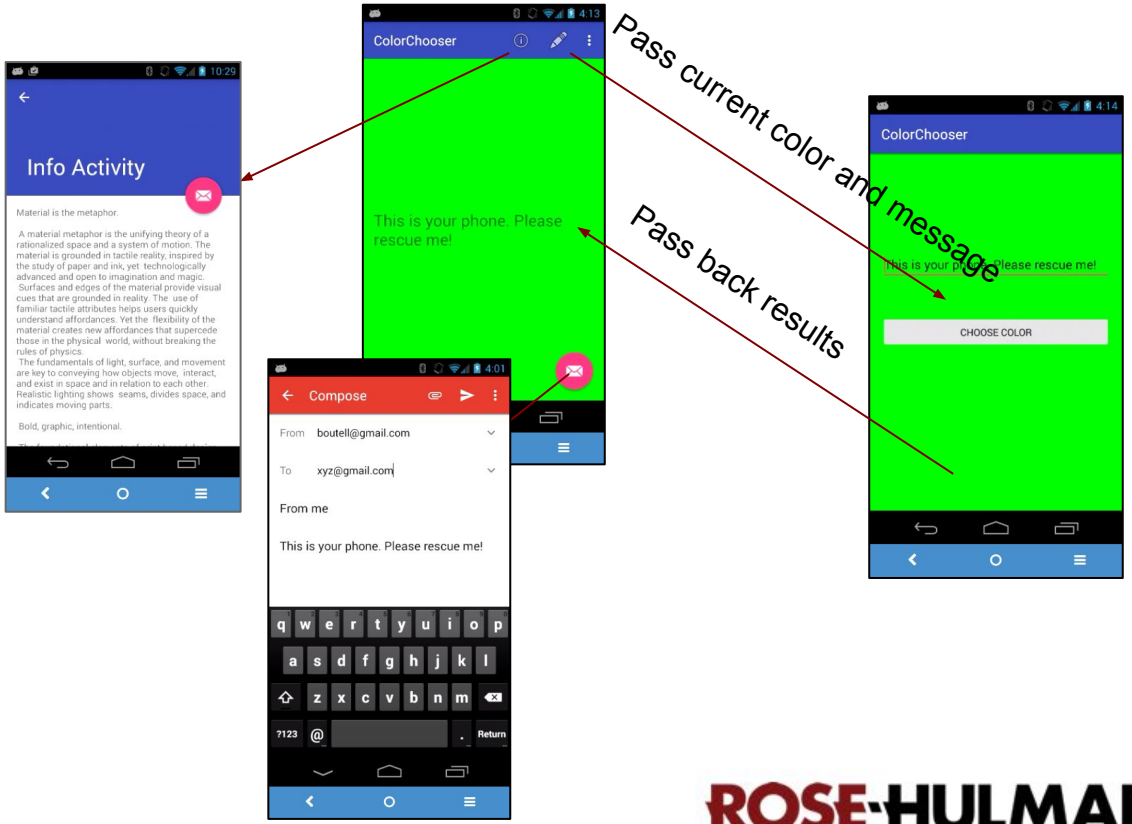
More system intents?

Here are many examples:

1. Alarm Clock
2. Calendar
3. Camera
4. Contacts/People App
5. Email
6. File Storage
7. Local Actions
8. Maps
9. Music or Video
10. New Note
11. Phone
12. Search
13. Settings
14. Text Messaging
15. Web Browser

http://developer.android.com/guide/components/intents-common.html

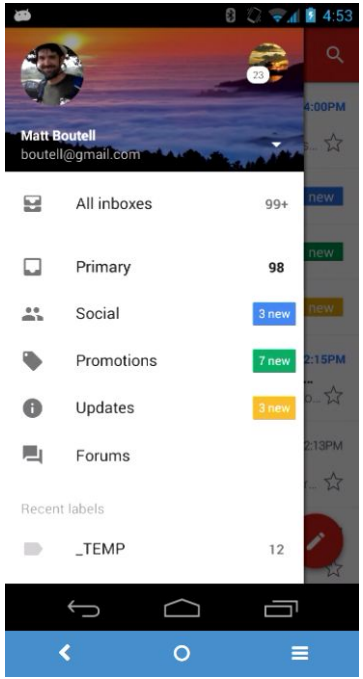
Summary:
Use intents to launch activities, and extras to send info

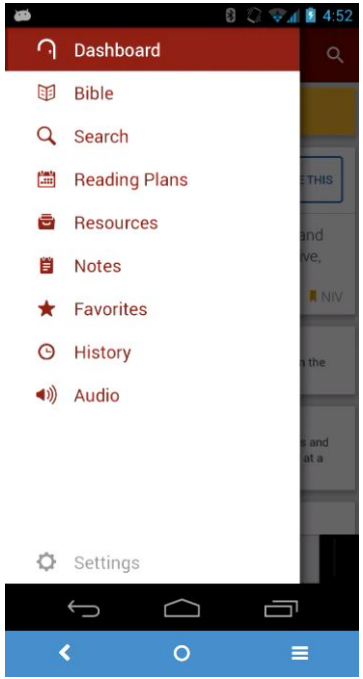


ROSE-HULMAN INSTITUTE OF TECHNOLOGY

NavigationDrawer

In this lesson, you'll learn about Navigation Drawers and study the starting code



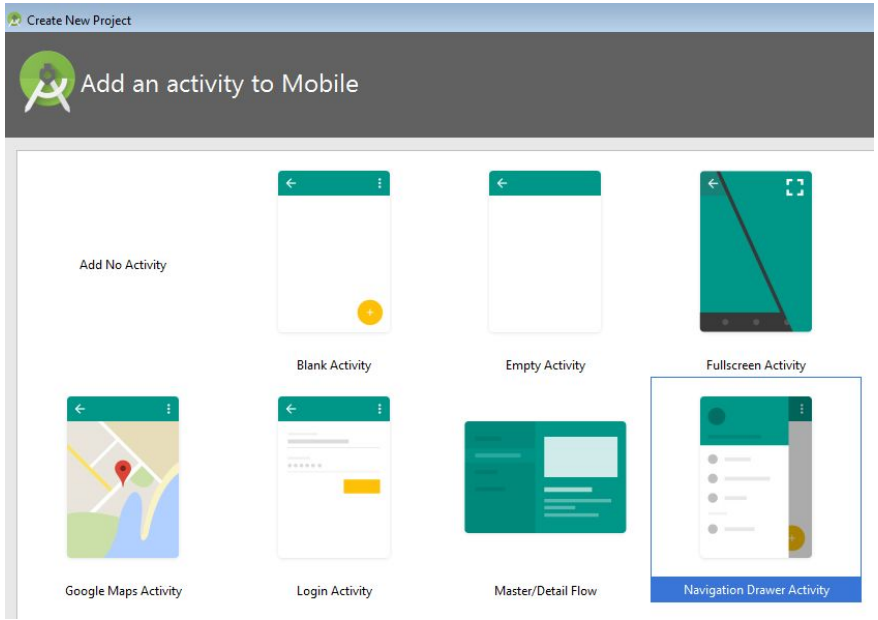


Create a new project with a NavigationDrawer activity

Name: HistoricalDocs

Activity:
NavigationDrawerActivity

Run it and check out the NavDrawer action!



Layout structure

Key: = has

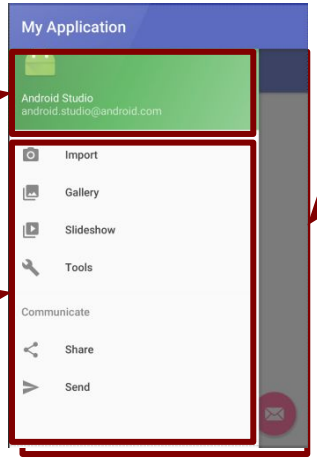
= is (visual)

activity_main.xml:
DrawerLayout

NavigationView

nav_header_main.xml:
LinearLayout

menu/
activity_main_drawer.xml



app_bar_main.xml:
CoordinatorLayout

content_main.xml
(will be fragment)

New things in MainActivity.java

It is a **NavigationView.OnNavigationItemSelectedListener**

onNavigationItemSelected() called

Capture the drawer

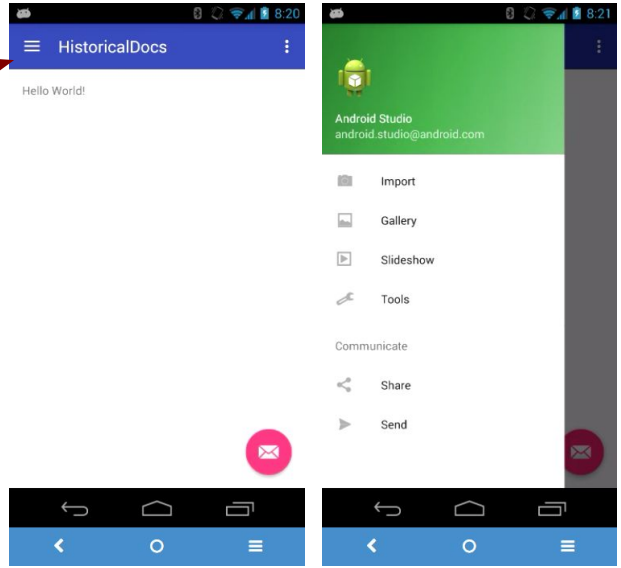
Create a toggle

Tie them together

If back is pressed:

If the drawer is open, close it.

Otherwise back out of app



Download/add the HistoricalDocs code and re-run: zip

res/

- New **menu/**activity_main_drawer.xml with menu items for info, docs, and settings.
- New values/strings.xml
- raw/* for the text of several historical document/excerpts (copy whole folder into res/)
  - http://stackoverflow.com/questions/4087674/android-read-text-raw-resource-file
- layout/row_view_doc.xml

java/ (**all files + folders** go into your edu.rosehulman.historicaldocs package)

- utils/DocUtils.java to read the documents (need to fix imports)
- Add `implementation 'commons-io:commons-io:2.4'`
  - and `implementation 'com.android.support:cardview-v7:27.1.0'` (or your versn)
  - to your build.gradle
- Doc model object
- DocListAdapter (keep commented out for now)

Copy all to your project, overwriting as needed.

Remove if/else in MainActivity.onNavigationItemSelected

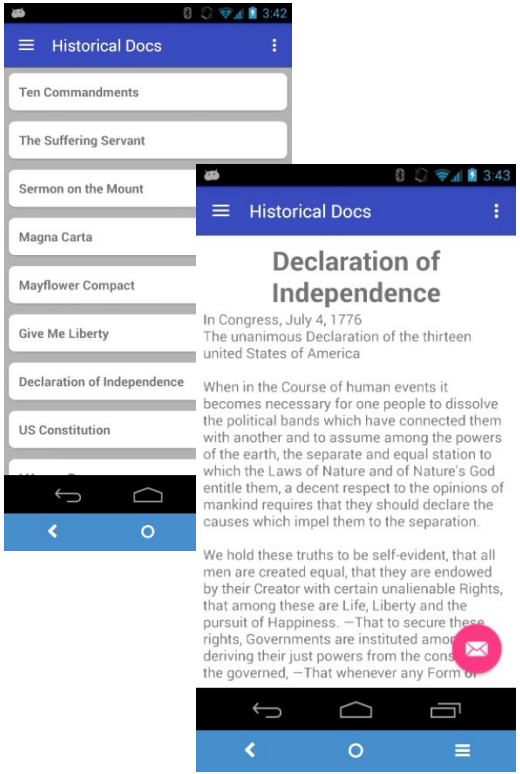
github if you get stuck

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Switching between views

In the HistoricalDocs app, we will switch between 3 displays:

1. **About**: About the authors and the app (from nav item)
2. **Doc List**: a RecyclerView that shows a list of all the document titles (from nav item)
3. **Doc Detail**: the title and all the text from each document (from clicking on a doc in the list)



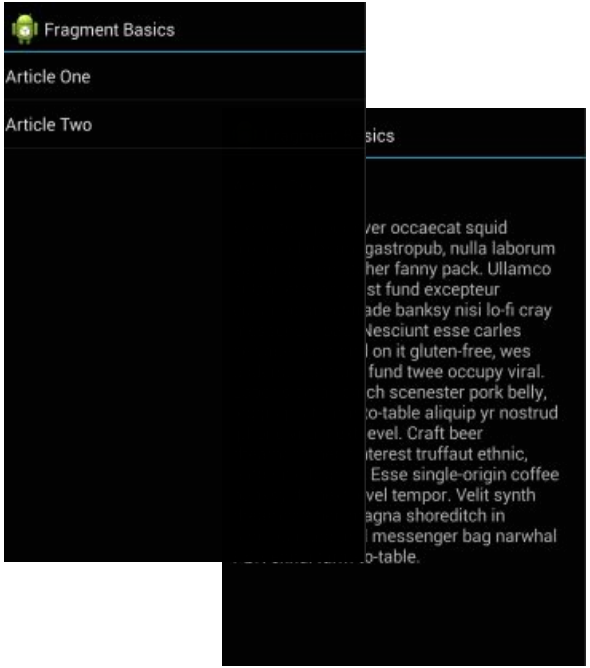
There is a nicer way to do this than using an Activity for each: **Fragments (next)**

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Introduction to Fragments

In this lesson you will

learn how to use Fragments

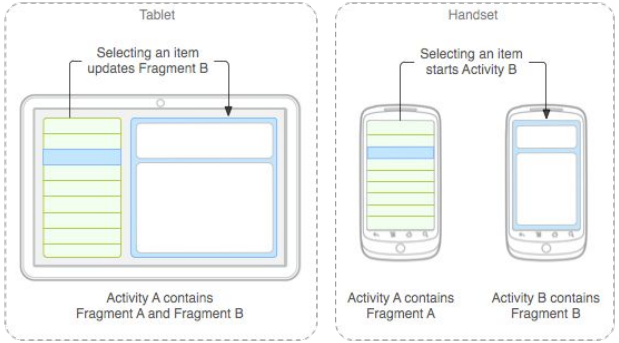


ROSE-HULMAN INSTITUTE OF TECHNOLOGY

What is a Fragment?

A re-usable “sub-activity”, with it’s own:

- layout
- lifecycle



They always live within an activity

They can be dynamically added or removed at runtime

They can communicate with other parts of the activity

Great for reconfiguring layouts on different-size devices

Less overhead than Activities, so snappy transitions while swapping

Fragment:Activity as JPanel:JFrame (analogy)

http://developer.android.com/guide/components/fragments.html

Make an AboutFragment

1. Make a `fragments` package in your java folder for our fragments.

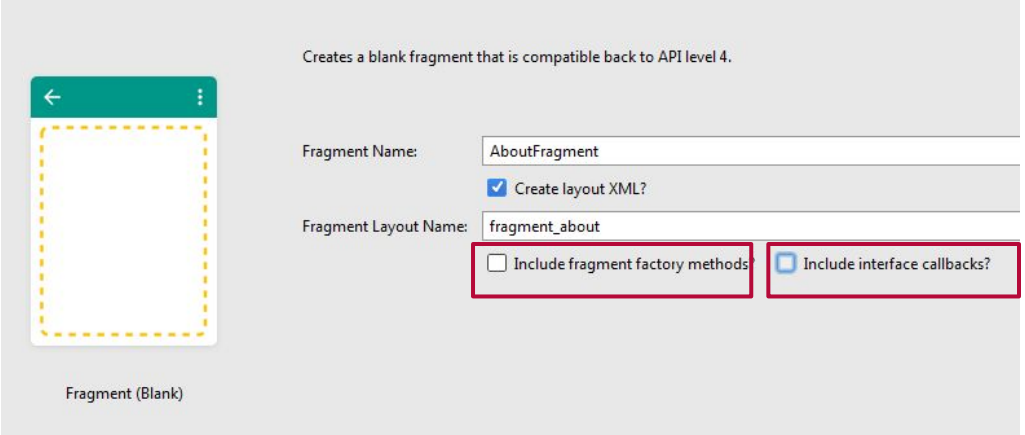
Then click the package, then New > Fragment > Fragment (Blank)

Unselect both includes

Name: AboutFragment

Check out starting code

Note onCreate**View**()



Copy the contents of fragment_about.xml

```xml
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:padding="12dp"
   android:layout_height="match_parent"
   >

   <ImageView
       android:id="@+id/image_view"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_alignParentLeft="true"
       android:layout_alignParentStart="true"
       android:layout_alignParentTop="true"
       android:scaleType="fitCenter"
       android:src="@android:drawable/sym_def_app_icon" />

   <TextView
       android:id="@+id/text_view"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_toRightOf="@id/image_view"
       android:text="@string/about_fragment_text"
       android:textSize="24sp"/>

</RelativeLayout>
```

Make a container to hold fragments in content_main.xml

Change the ConstraintLayout to a FrameLayout

Give it an id like `android:id = "@+id/fragment_container"`

Remove the “hello world” TextView.

Fragments are usually added via FragmentTransactions

```
public class MainActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
        navigationView.setNavigationItemSelectedListener(this);

        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
        ft.add(R.id.fragment_container, new AboutFragment());
        ft.commit();
    }
```

Be sure to use and import the support versions

It will add this fragment whenever the Activity is created

You can also replace and remove fragments and add to the backstack

Instead, load the AboutFragment when requested

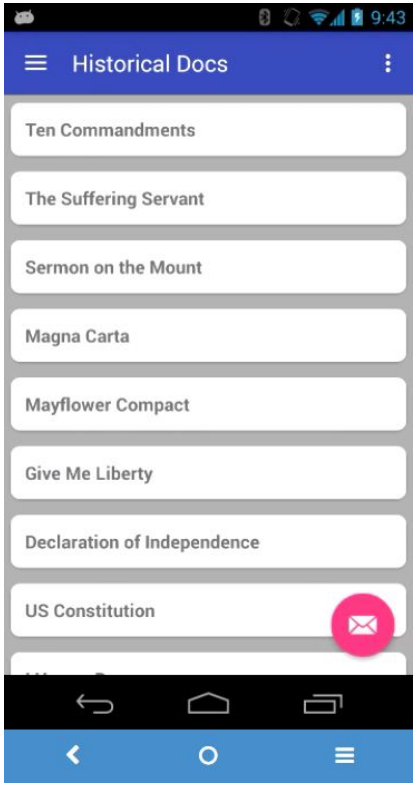
```
public class MainActivity extends AppCompatActivity
   protected void onCreate(Bundle savedInstanceState) {
//         FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
//         ft.add(R.id.fragment_container, new AboutFragment());
//         ft.commit();
   }

      public boolean onNavigationItemSelected(MenuItem item) {
         Fragment switchTo = null;
         switch (item.getItemId()) {
             case R.id.nav_about:
                 switchTo = new AboutFragment();
                 break;

         }

         if (switchTo != null) {
             FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
             ft.replace(R.id.fragment_container, switchTo);
             ft.commit();
         }
         DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);

      ...
```

Replace() is correct when swapping fragments

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

ListFragment

In this lesson you will learn how to create a ListFragment that uses an adapter and a callback



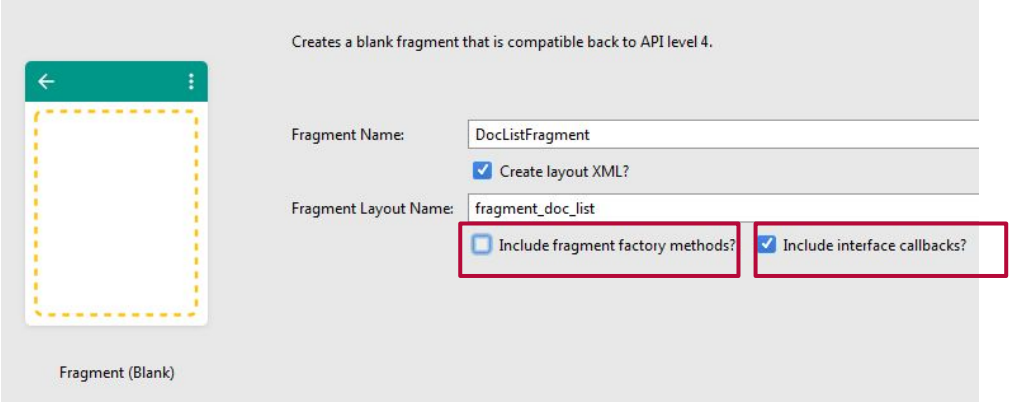
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Make another new Fragment

Name: DocListFragment

Type: Blank Fragment (ListFragment has lots of starting code, including a dummy model class and a RecyclerAdapter)

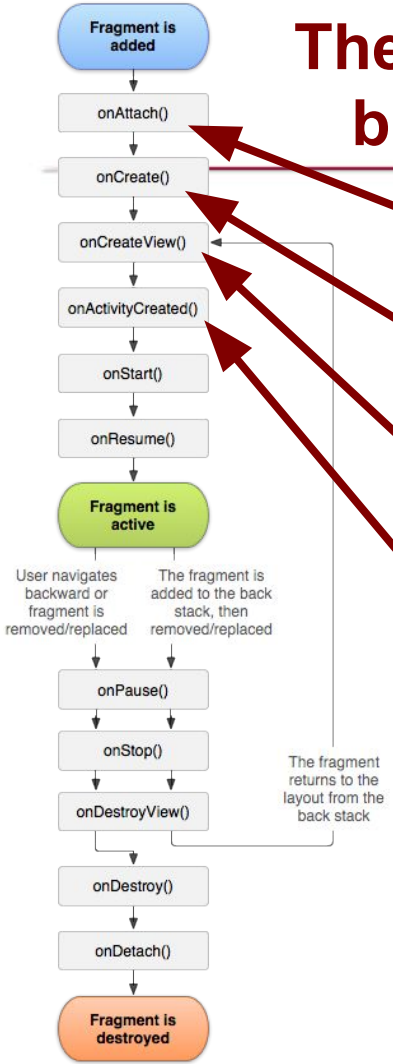
Include Interface callbacks



Copy the contents of fragment_doc_list

```
<android.support.v7.widget.RecyclerView
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="@android:color/darker_gray"
   tools:context=".fragments.DocListFragment">

</android.support.v7.widget.RecyclerView>
```

The Fragment lifecycle is similar to an Activity's, but accounts for relationship with the Activity



where you store reference to parent Activity

initialize data variables

inflates the View (capture and add listeners)

do anything relying on parent UI

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

The interface callbacks are onAttach(), onDetach()

Goal: Clicking on an item in the list will display that item's detail view.

So the DocListFragment must be **replaced** with a DocDetailFragment.

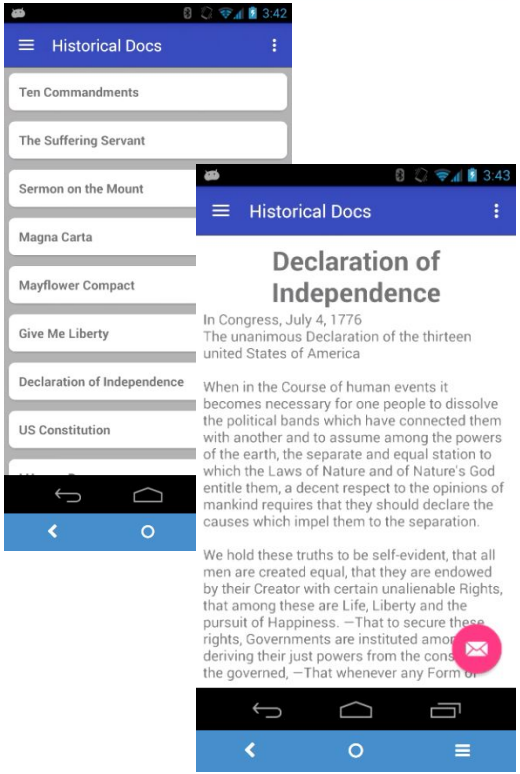
What class can do the replacing? The activity.

We call activity.onDocSelected() to do the replacing.

To require the activity to implement that method, we declare it to implement an interface.

OnFragmentInteractionListener is that interface.

Using interfaces to do this is good OO design (Separation of Concerns pattern)



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

OnFragmentInteractionListener

Rename OnFragmentInteractionListener to Callback, and mListener to mCallback.

Rename its method to onDocSelected(Doc doc)

In onAttach(), we guarantee that the context that created the fragment is a Callback

Delete onButtonPressed() after noting the sample call,
`mCallback.onDocSelected()`

Finish onCreateView

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
   RecyclerView view = (RecyclerView)inflater.inflate(R.layout.fragment_doc_list, container, false);
   view.setLayoutManager(new LinearLayoutManager(getContext()));
   DocListAdapter adapter = new DocListAdapter(getContext(), mCallback);
   view.setAdapter(adapter);
   return view;
}
```

Uncomment the DocListAdapter class. Note: when a ViewHolder is clicked, it calls mCallback.onDocSelected(doc)

Create the DocListFragment when requested

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    Fragment switchTo = null;
    switch (item.getItemId()) {
        case R.id.nav_about:
            switchTo = new AboutFragment();
            break;
        case R.id.nav_docs:
            switchTo = new DocListFragment();
            break;
    }

    if (switchTo != null) {
        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
        ft.replace(R.id.fragment_container, switchTo);
        ft.commit();
    }
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    drawer.closeDrawer(GravityCompat.START);
    return true;
}
```

If you run, you'll get a ClassCastException

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Stub in onDocSelected

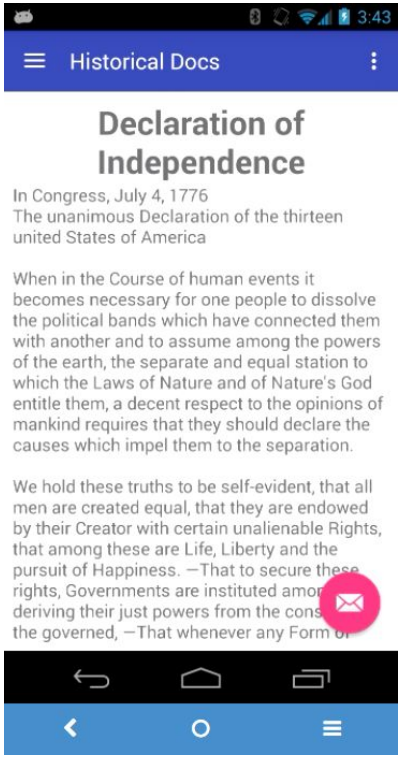
Eventually, it will do the fragment transaction.

Run it now to see the list of docs.

You may want to remove the padding from content_main.xml since I used a gray background for the fragment.

DetailFragment

In this lesson you will learn how to create a DetailFragment for a doc using the newInstance() pattern.



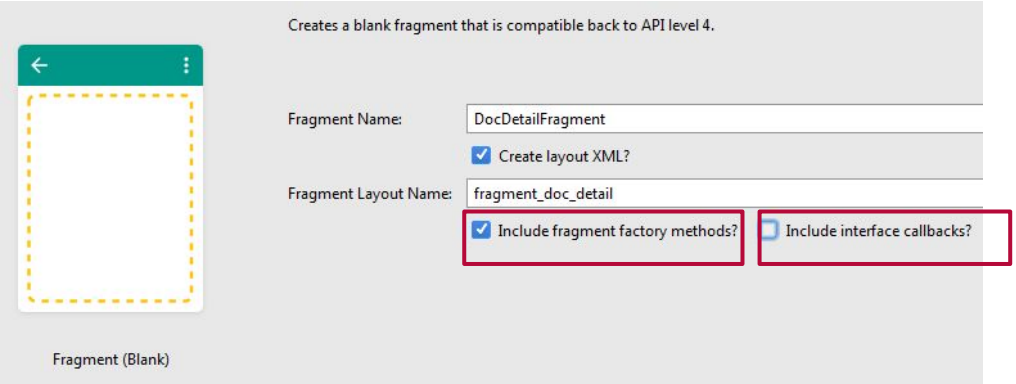
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

One more new Fragment

Name: DocDetailFragment

Type: Blank Fragment

Include factory methods



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Copy the contents of fragment_doc_detail

```
<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:orientation="vertical"
   android:padding="8dp"
   tools:context=".fragments.DocDetailFragment">

   <TextView
       android:id="@+id/fragment_doc_detail_title"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:gravity="center"
       android:textSize="30sp"
       android:textStyle="bold" />

   <ScrollView
       android:id="@+id/scrollView"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:layout_gravity="center_horizontal">

       <TextView
           android:id="@+id/fragment_doc_detail_body"
           android:layout_width="match_parent"
           android:layout_height="wrap_content"
           android:textSize="16sp" />

   </ScrollView>
</LinearLayout>
```

It stubs in a newInstance() method

We need to pass in data (the doc) to our fragment.

Why not use a constructor with args?

If Android has to kill your Fragment and re-create it, it will write the data to disk:

- It only ever recreates Fragments using the **empty** constructor

- It persists data through a Bundle: note onCreate()'s savedInstanceState!

So **we** should use the default constructor and data in a Bundle.

Best practice is to encapsulate this in a the **newInstance()** factory method that will package any "constructor" arguments into a Bundle

```java
public BlankFragment() {
    // Required empty public constructor
}

public static BlankFragment newInstance(String param1, String param2) {
    BlankFragment fragment = new BlankFragment();
    Bundle args = new Bundle();
    args.putString(ARG_PARAM1, param1);
    args.putString(ARG_PARAM2, param2);
    fragment.setArguments(args);
    return fragment;
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getArguments() != null) {
        mParam1 = getArguments().getString(ARG_PARAM1);
        mParam2 = getArguments().getString(ARG_PARAM2);
    }
}
```

We will remove param2 and change param1 to a doc (next slide)

http://www.androiddesignpatterns.com/2012/05/using-newinstance-to-instantiate.html
http://stackoverflow.com/questions/9245408/best-practice-for-instantiating-a-new-android-fragment
http://stackoverflow.com/a/14655001

DocDetailFragment with the Doc argument

```java
public class DocDetailFragment extends Fragment {
   private static final String ARG_DOC = "doc";
   private Doc mDoc;
   public DocDetailFragment() { /* Required empty public constructor */ }
   public static DocDetailFragment newInstance(Doc doc) {
       DocDetailFragment fragment = new DocDetailFragment();
       Bundle args = new Bundle();
       args.putParcelable(ARG_DOC, doc);
       fragment.setArguments(args);
       return fragment;
   }
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       if (getArguments() != null) {
           mDoc = getArguments().getParcelable(ARG_DOC);
       }
   }
   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
       // Inflate the layout for this fragment, using the saved doc
       return inflater.inflate(R.layout.fragment_doc_detail, container, false);
   }
}
```

But Doc is a custom object. How do I pass it via a Bundle?

Bundle arguments, like extras, can be objects, as long as they are serializable. In Android, developers use Parcelable instead.

The Parcelable interface has **lots** of boilerplate code, but don't type it!

Just type `implements Parcelable` and Studio will write it **all** for you with three clicks. :)

```java
public class Doc implements Parcelable {
    private String title;
    private String text;

    public Doc(String title, String text) {
        this.title = title;
        this.text = text;
    }

    protected Doc(Parcel in) {
        title = in.readString();
        text = in.readString();
    }

    public static final Creator<Doc> CREATOR = new Creator<Doc>() {
        @Override
        public Doc createFromParcel(Parcel in) {
            return new Doc(in);
        }

        @Override
        public Doc[] newArray(int size) {
            return new Doc[size];
        }
    };

    public String getTitle() {
        return title;
    }

    public String getText() {
        return text;
    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(title);
        dest.writeString(text);
    }
}
```

Serialization uses reflection and is slow; parcels have been shown to be over 10x faster! http://www.developerphil.com/parcelable-vs-serializable/

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Cause onCreateView() to display the saved doc

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_doc_detail, container, false);
    TextView titleView = (TextView) view.findViewById(R.id.fragment_doc_detail_title);
    titleView.setText(mDoc.getTitle());

    TextView bodyView = (TextView) view.findViewById(R.id.fragment_doc_detail_body);
    bodyView.setText(mDoc.getText());

    return view;
}
```

Implement MainActivity's onDocSelected()

A fragment transaction that replaces the fragment with a new DocDetailFragment that you create using the newInstance() method.

Do it now (solution on next slide)

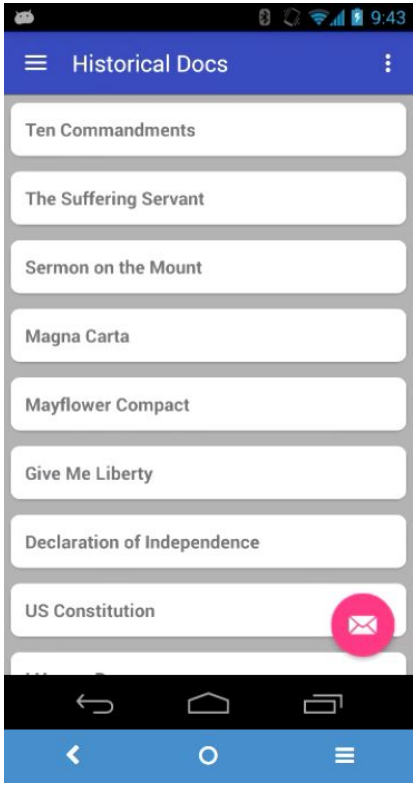
Implement MainActivity's onDocSelected()

A fragment transaction that replaces the fragment with a new DocDetailFragment that you create using the newInstance() method:

```
@Override
public void onDocSelected(Doc doc) {
   FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
   Fragment detailFragment = DocDetailFragment.newInstance(doc);
   ft.replace(R.id.fragment_container, detailFragment);
   ft.commit();
}
```

Managing the fragment backstack

In this lesson you will manage the backstack



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Why do we need the backstack?

Do this now: navigate to a document detail and press the back button.

What happens?

What *should* happen is to go back to the DocListFragment.

With fragments, you need to manage your own back stack.

Easy one-liner:

Run it again.

```
@Override
public void onDocSelected(Doc doc) {
    FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
    Fragment detailFragment = DocDetailFragment.newInstance(doc);
    ft.replace(R.id.fragment_container, detailFragment);
    ft.addToBackStack("detail"); // adds this transaction to the stack
    ft.commit();
}
```

Ah but new problems arise!

Replicate the bug: navigate to a document detail, go to the About fragment, then press back.

When we switch to the about or list fragments, we want to clear the backstack:

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
   Fragment switchTo = null;
   ...
   if (switchTo != null) {
       FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
       ft.replace(R.id.fragment_container, switchTo);
       for (int i = 0; i < getSupportFragmentManager().getBackStackEntryCount(); ++i) {
           getSupportFragmentManager().popBackStackImmediate();
       }
       ft.commit();
   }
   DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
   drawer.closeDrawer(GravityCompat.START);
   return true;
}
```

Discussion of options here:
http://stackoverflow.com/questions/6186433/clear-back-stack-using-fragments

Parting comment: what if we want it to start with the AboutFragment?

In onCreate, uncomment out these lines:

```
//        FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
//        ft.add(R.id.fragment_container, new AboutFragment());
//        ft.commit();
```

Now run the app, navigate to the doc list and rotate the screen.

:)

You only want to add the fragment the first time the activity is created.

How do we know if it's the 'first time'? savedInstanceState would be null

Guard it with this condition:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   ...
   if (savedInstanceState == null) {
       FragmentTransaction ft = getSupportFragmentManager().beginTransaction();
       ft.add(R.id.fragment_container, new AboutFragment());
       ft.commit();
   }
}
```

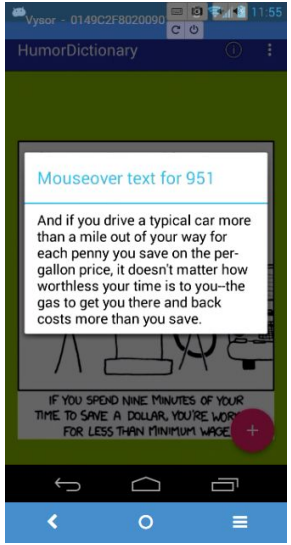
Lab: ComicViewer

Write an app to scroll through selected xkcd comics

Based on **ViewPager,** another fragment-based UI pattern

Extra stuff to learn:

- Populating model objects from remote json
- Loading images stored remotely
  - Uses an **AsyncTask** (see video)
- Image scroll and zoom

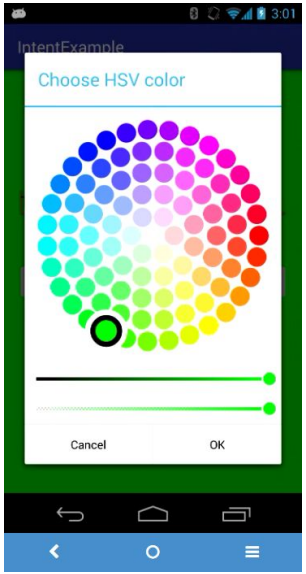


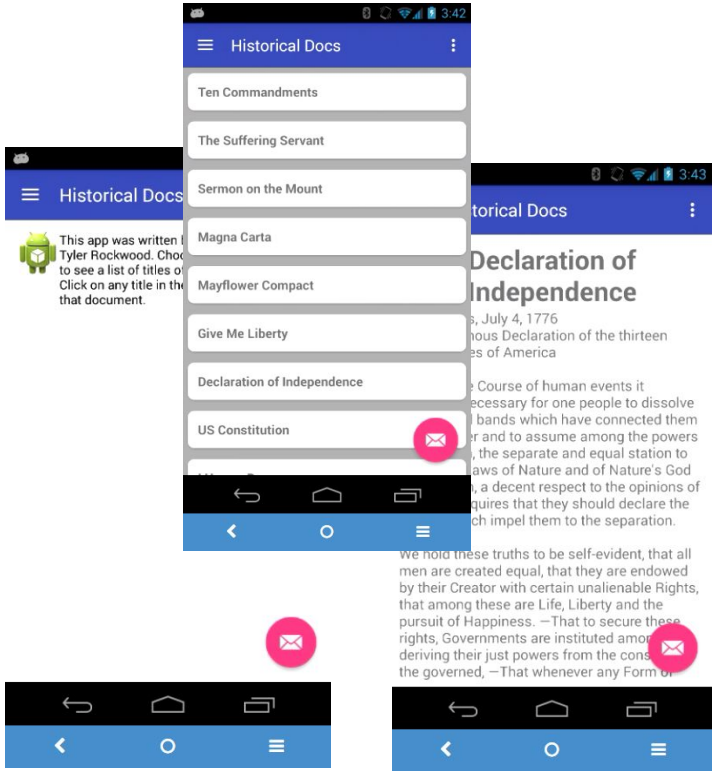
https://xkcd.com/

Summary: UIs are built out of activities and fragments

Activities are screens

Fragments are reusable components with an encapsulated layout and controller





ROSE-HULMAN INSTITUTE OF TECHNOLOGY

FAQ

1. Can I declare fragments in xml?
- Yes. (to right)

- Capture using fragmentManager.findFragmentById(id)

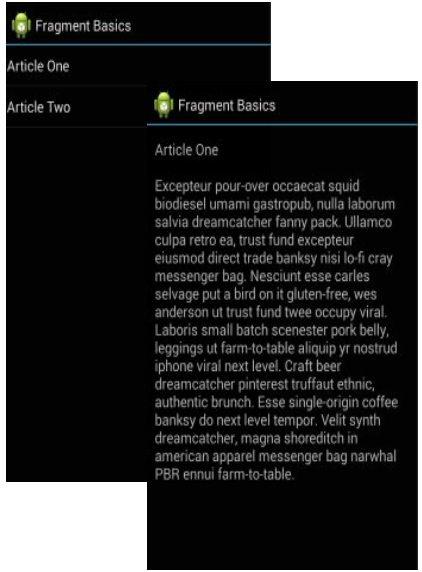
2. Why aren't Fragments in the Android Manifest?

They only exist within an activity!

```
<LinearLayout xmlns:android="ht
    android:orientation="horizo
    android:layout_width="match_
    android:layout_height="matc

    <fragment android:name="com
              android:id="@+id/
              android:layout_we
              android:layout_wi
              android:layout_he

    <fragment android:name="com
              android:id="@+id/
              android:layout_we
              android:layout_wi
              android:layout_he

</LinearLayout>
```

You can place fragments in different workflows depending on the device

Nexus 4: Intents for articles



Nexus 7: multiple fragments on screen

See developer.android.com download in lab.
http://developer.android.com/guide/components/fragments.html

Next steps for further study

1. The fragments guide has a great example at the end to make sure you understand how to make fragments. It also has more about using fragments differently on different devices.

http://developer.android.com/guide/components/fragments.html

2. Experiment with other fragment-based UI patterns besides the NavDrawer:
   a. TabbedActivity uses a ViewPager (this week's lab)
   b. Master/Detail Flow (related to the example above)

On the app:

- You could make the Settings menu item launch a Preference Fragment
- You could make the FAB email the current doc.