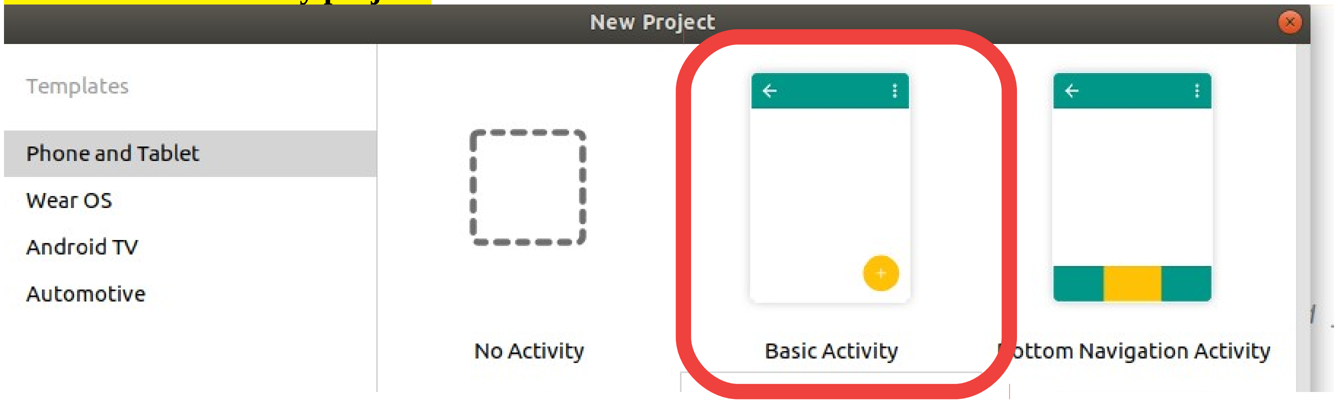
NOTE: Hover over apis that are in red and hit ‘alt-enter’ and then select import to import needed packages.

Create a Basic Activity project:



Chop out fragments and navigation.

1. The manifest

```
<uses-permission android:name="android.permission.INTERNET"/>
```

2. Add an imageview to content_main.xml. Code below

```
<ImageView
    android:id="@+id/imageView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="left"
    android:layout_weight="1"
    android:scaleType="fitXY"
    android:src="@drawable/ic_launcher_foreground" />
```

3. Add a second fab in activity_main.xml

```
<com.google.android.material.floatingactionbutton.FloatingActionButton

    android:id="@+id/fabgetjson"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|left"
    android:layout_marginLeft="@dimen/fab_margin"
    android:layout_marginBottom="16dp"
    app:srcCompat="@android:drawable/ic_dialog_info" />
```

4. Add a ViewModel class to track data

```
public class DataVM extends ViewModel
```

5. Add these as member variables to the ViewModel (the stuff we will get)

```
//gotta define this somewhere

String links[] = {  "https://www.pcs.cnu.edu/~kperkins/pets/p33.png",
            "https://www.pcs.cnu.edu/~kperkins/pets/p44.png",
            "https://www.pcs.cnu.edu/~kperkins/pets/p55.png",
            "https://www.pcs.cnu.edu/~kperkins/pets/p22.png"};
String jsonlink="https://www.pcs.cnu.edu/~kperkins/pets/pets.json";
//this is the value that we want to keep track of through rotations
int currentLink=0;
//threads that we may launch
GetImageThread myGetImageThread;
GetTextThread myGetTextThread;
```

6. Add 2 methods to the ViewModel that launch threads that get images and json

```
public void getJSON(){

      myGetTextThread=new GetTextThread(jsonlink);
      myGetTextThread.start();
}
public void getImage(String url){
      myGetImageThread=new GetImageThread(url);
      myGetImageThread.start();
}
```

7. Add the live data that the viewmodel threads will be updating and that the Activity will be listening for changes on

```
//lets add some livedata

//the bitmap we are looking for
private MutableLiveData<Bitmap> bmp ;
public MutableLiveData<Bitmap> getbmp() {
      if(bmp==null)
            bmp=new MutableLiveData<Bitmap>();
      return bmp;
}
//any communications from thread
private MutableLiveData<String> result ;
public MutableLiveData<String> getresult() {
      if(result==null)
            result=new MutableLiveData<String>();
      return result;
}
```

8. Add the 2 threads to the viewmodel

```
public class GetTextThread extends Thread{

      private static final String    TAG = "GetTextThread";
      private static final int       DEFAULTBUFFERSIZE = 8096;
      private static final int       TIMEOUT = 1000;    // 1 second
      protected int                  statusCode = 0;
      private String                 url;
      public GetTextThread(String url) {
            this.url=url;
      }
      public void run() {
            try {
                  URL url1 = new URL(url);
                  // this does no network IO
                  HttpURLConnection connection = (HttpURLConnection) url1.openConnection();
                  // can further configure connection before getting data
                  // cannot do this after connected
                  connection.setRequestMethod("GET");
                  connection.setReadTimeout(TIMEOUT);
                  connection.setConnectTimeout(TIMEOUT);
                  connection.setRequestProperty("Accept-Charset", "UTF-8");
                  // wrap in finally so that stream bis is sure to close
                  // and we disconnect the HttpURLConnection
                  BufferedReader in = null;
                  try {
                        // this opens a connection, then sends GET & headers
                        connection.connect();
                        // lets see what we got make sure its one of
                        // the 200 codes (there can be 100 of them
                        // http_status / 100 != 2 does integer div any 200 code will = 2
                        statusCode = connection.getResponseCode();
                        if (statusCode / 100 != 2) {
                              result.postValue("Failed! Statuscode returned is " +
Integer.toString(statusCode));
                              return;
                        }
                        in = new BufferedReader(new
InputStreamReader(connection.getInputStream()), DEFAULTBUFFERSIZE);
                        // the following buffer will grow as needed
                        String myData;
                        StringBuffer sb = new StringBuffer();
                        while ((myData = in.readLine()) != null) {
                              sb.append(myData);
                        }
                        result.postValue(sb.toString());
                  } finally {
                        // close resource no matter what exception occurs
                        if(in != null)
                              in.close();
                        connection.disconnect();
                  }
            } catch (Exception exc) {
                  Log.d(TAG, exc.toString());
                  result.postValue(exc.toString());
            }
      }
}
public class GetImageThread extends Thread{
      private static final String TAG = "GetImageThread";
      private static final int       DEFAULTBUFFERSIZE = 50;
      private static final int       NODATA = -1;
      private int                    statusCode=0;
      private String                 url;
      public GetImageThread(String url) {
            this.url=url;
      }
      public void run(){
            // note streams are left willy-nilly here because it declutters the
            // example
            try {
                  URL url1 = new URL(url);
                  // this does no network IO
                  HttpURLConnection connection = (HttpURLConnection) url1.openConnection();
                  // can further configure connection before getting data
                  // cannot do this after connected
                  // connection.setRequestMethod("GET");
                  // connection.setReadTimeout(timeoutMillis);
                  // connection.setConnectTimeout(timeoutMillis);
                  // this opens a connection, then sends GET & headers
                  connection.connect();
                  // lets see what we got make sure its one of
                  // the 200 codes (there can be 100 of them
                  // http_status / 100 != 2 does integer div any 200 code will = 2
                  int statusCode = connection.getResponseCode();
                  if (statusCode / 100 != 2) {
                        result.postValue("Failed! Statuscode returned is " +
Integer.toString(statusCode));
                        return;
                  }
                  // get our streams, a more concise implementation is
                  // BufferedInputStream bis = new
                  // BufferedInputStream(connection.getInputStream());
                  InputStream is = connection.getInputStream();
                  BufferedInputStream bis = new BufferedInputStream(is);
                  // the following buffer will grow as needed
                  ByteArrayOutputStream baf = new
ByteArrayOutputStream(DEFAULTBUFFERSIZE);
                  int current = 0;
                  // wrap in finally so that stream bis is sure to close
                  try {
                        while ((current = bis.read()) != NODATA) {
                              baf.write((byte) current);
                        }
                        // convert to a bitmap
                        byte[] imageData = baf.toByteArray();
                        //some live data here
                        //can only postValue from background thread not setValue
                        bmp.postValue(BitmapFactory.decodeByteArray(imageData, 0,
imageData.length));
                        result.postValue(url);
                  } finally {
                        // close resource no matter what exception occurs
                        if(bis!= null)
                              bis.close();
                  }
            } catch (Exception exc) {
                  Log.d(TAG, exc.toString());
                  result.postValue(exc.toString());
            }
      }
}
```

And now to the mainactivity

9. dump the navigation and binder and fragment stuff

10. Add some member vars to track the viewmodel and the imageview

```
//persists accross config changes

DataVM myVM;
ImageView iv;
```

11. Setup infrastructure in onCreate

```
setContentView(R.layout.activity_main);

iv=findViewById(R.id.imageView1);
setSupportActionBar(findViewById(R.id.toolbar));
```

12. Get a ref to the viewmodel

```
// Create a ViewModel the first time the system calls an activity's

// onCreate() method.  Re-created activities receive the same
// MyViewModel instance created by the first activity.
myVM = new ViewModelProvider(this).get(DataVM.class);
```

13. Create some observers (in onCreate) on the MutableLiveData in the ViewModel. These will be notified when the contents in the ViewModel change

```
// Create the observer which updates the UI.

final Observer<Bitmap> bmpObserver = new Observer<Bitmap>() {
        @Override
        public void onChanged(@Nullable final Bitmap newbmp) {
                // Update the UI, in this case, a TextView.
                iv.setImageBitmap(newbmp);
         }
};
// Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.
myVM.getbmp().observe(this,bmpObserver);
// Create the observer which updates the UI.
final Observer<String> resultObserver = new Observer<String>() {
        @Override
        public void onChanged(@Nullable final String result) {
                // Update the UI, in this case, a TextView.
                Toast.makeText(MainActivity.this,result,Toast.LENGTH_SHORT).show();
        }
};
// Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.
myVM.getresult().observe(this,resultObserver);
```

14. And finally, set up the onclick listeners on the fabs (in onCreate)

```
findViewById(R.id.fab).setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View view) {
                String url=myVM.links[myVM.currentLink++%myVM.links.length];
                myVM.getImage(url);
        }
});
findViewById(R.id.fabgetjson).setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
                myVM.getJSON();
        }
});
```