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

Create an empty activity project

1. The manifest

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

2. Add an imageview and 2 floatig action buttons to activity_main.xml. Code below

```
<ImageView
    android:id="@+id/imageView1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:gravity="left"
    android:scaleType="fitXY"
    android:src="@drawable/ic_launcher_foreground"
    tools:layout_editor_absoluteX="59dp"
    tools:layout_editor_absoluteY="-31dp" />

<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:srcCompat="@android:drawable/ic_dialog_email" />

<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"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:srcCompat="@android:drawable/ic_dialog_info" />
```

3. Add a ViewModel class to project to track data

```
public class DataVM extends ViewModel
```

4. In DataVM, add 2 methods to the ViewModel that launch threads that get images and json

```
public void getJSON(String url){
    GetTextThread myThread = new GetTextThread(url);
    myThread.start();
}

public void getImage(String url){
    GetImageThread myThread = new GetImageThread(url);
    myThread.start();
}
```

5. In DataVM, add the live data that the viewmodel threads will be updating and that the Activity will be listening for changes on

```
//the bitmap we are looking for
private MutableLiveData<Bitmap> bmp ;
public MutableLiveData<Bitmap> getbmp() {
    if(bmp==null)
        bmp=new MutableLiveData<Bitmap>();
    return bmp;
}

//the json we will download
private MutableLiveData<String> result ;
public MutableLiveData<String> getresult() {
    if(result==null)
        result=new MutableLiveData<String>();
    return result;
}
```

6. In DataVM, add threads that will get the json and the bitmap

```
private class GetTextThread extends Thread{
    private String  url;
    public GetTextThread(String url) {
        this.url=url;
    }
    public void run() {
        //run the task
        Download_https mytask = new Download_https(this.url);
        result.postValue(mytask.get_text());
    }
}
private class GetImageThread extends Thread {
    private String url;
    public GetImageThread(String url) {
        this.url = url;
    }

    public void run() {
        //run the task
        Download_https mytask = new Download_https(this.url);
        bmp.postValue(mytask.get_Bitmap());
    }
}
```

In MainActivity.java

7. Add some member vars to track the viewmodel and the imageview and where to get data from

```java
private DataVM myVM;
private ImageView iv;

//base url of json and bitmap
private static final String MYURL =
"https://raw.githubusercontent.com/CNUClasses/475_web_data/m
aster/";
```

8. Setup infrastructure in onCreate

```java
setContentView(R.layout.activity_main);
iv=findViewById(R.id.imageView1);
```

9. Get a ref to the viewmodel

```java
// 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);
```

10. 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);
```

11. 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) {
        myVM.getImage(MYURL+"p0.png");
    }
});

findViewById(R.id.fabgetjson).setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        myVM.getJSON(MYURL+"pets.json");
    }
});
```