List in class lab
See List_and_RecyclerView class demo
See https://github.com/codepath/android_guides/wiki/Using-the-RecyclerView for basis of this lab.

with and without threaded pages
1. First create an Empty Activity project

The XML
2. Replace the TextView with **RecyclerView** in activity_main.xml

```
        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/rvContacts"
            android:layout_width="0dp"
            android:layout_height="0dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />
```

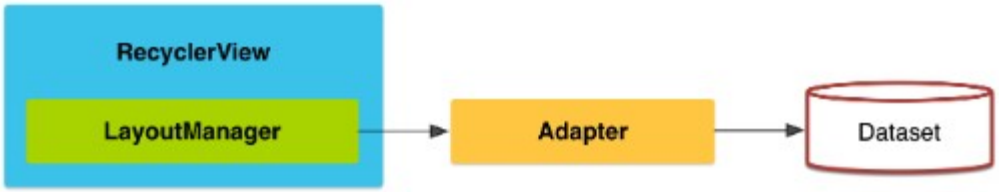
3. Need a layout to define the appearance of each **row** in the RecyclerView. Here we will have 2 textViews (see List_and_RecyclerView for page look demo).

In layout folder create row_layout (or any name you want) (from Layout folder→right click →new→xml→layout XML file. Give it a name and (choose LinearLayout for layout type, because we just have a row of data).

Be sure the height of each row is wrap_content, (if match parent it would take entire screen) fill in row_layout with

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
        android:id="@+id/tvInfo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="TextView" />
    <TextView
        android:id="@+id/tvResult"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="TextView" />
</LinearLayout>
```

The Adapter

4. Create an 

adapter (the brains of the operation). It supplies the RecyclerView with 1 row of data at a time whose appearance is defined by row_layout above.

Create a new JavaClass RecyclerView_Adapter(or any name you like) and have it extend...

```
public class RecyclerView_Adapter extends RecyclerView.Adapter
```

5. Add unimplemented required methods (alt-enter on red squiggly lines)

6. The list will have many rows, each row will consists of a row_layout with three TextViews and will appear as;

2 squared = 4

As the user scrolls the screen (swipe up or down) new rows will appear with new results.

This layout is populated by the adapter in onBindViewHolder using the position argument as the number to double

But first we have to create each row. For that we need a layout inflator (remember its use in the spinner project?). Add one to to RecyclerView_Adapter as member variable

```
private final LayoutInflater li;
```

7. And we need a context to get this inflator. Add one to to RecyclerView_Adapter as member variable.

```
private final Context ctx;
```

8. Now add a constructor. (hover over class name and hit alt-insert) and pass in a reference to Mainactivity save in a member

```
public RecyclerView_Adapter(Context ctx){

    this.ctx=ctx;
    li=(LayoutInflater)ctx.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
}
```

9. Add a RecyclerView.ViewHolder to RecyclerView_Adapter

When each row_layout rolls off the screen do we garbage collect it? Or reuse this fully constructed object to hold the next layout?

Answer: Reuse it. That way we can forgo repeating expensive operations like findViewById)

```
class RowViewHolder extends RecyclerView.ViewHolder {
      TextView tvInfo;
      TextView tvResult;

    //after construction, above member variables hold references
    //to widgets IN THIS PARTICULAR ROW!
    public RowViewHolder(@NonNull View itemView) {
        super(itemView);
        tvInfo = (TextView)itemView.findViewById(R.id.tvInfo );
        tvResult = (TextView)itemView.findViewById(R.id.tvResult );
    }
}
```

10. Fill in the method that **CREATES** a ViewHolder

```
public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

    //call this when we need to create a brand new PagerViewHolder
    View view = li.inflate(R.layout.row_layout, parent, false);
    return new RowViewHolder(view);   //the new one
}
```

11. Fill in the method that **REUSES** the viewholder. Notice that we do not need to reinflate the views in this layout (they have already been created in onCreateViewHolder). We are just reusing them.

```
public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {

    //passing in an existing instance, reuse the internal resources
      //pass our data to our ViewHolder.
      RowViewHolder viewHolder = (RowViewHolder) holder;
      viewHolder.tvInfo.setText(Integer.toString(position) + " squared =");
      viewHolder.tvResult.setText(Integer.toString(position*position));}
```

12. The RecyclerView_Adapter has to know how many rows it will hold. In this case we decide. Create a maxRows field in the RecyclerView_Adapter and add another constructor, 1 uses *DEFAULT_MAX_ROWS* the other allows the user to select maxrows.

```
public class RecyclerView_Adapter extends  RecyclerView.Adapter{

    private static final int DEFAULT_MAX_ROWS = 100;
    private final LayoutInflater li;
    private final Context ctx;
    private final int maxRows;

    //one arg constructor uses DEFAULT_MAX_ROWS
    public RecyclerView_Adapter(Context ctx) {
        this(ctx,DEFAULT_MAX_ROWS);
     }
     //two arg constructor in case user wants to define their own maxrows
    public RecyclerView_Adapter(Context ctx, int maxRows) {
        this.ctx = ctx;
        li = (LayoutInflater)ctx.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        this.maxRows=maxRows;
    }
```

and finaly tell consumers of RecyclerView_Adapter how many rows its going to have (forget this and you will have 0 rows displayed)

```
public int getItemCount() {

    //the expected number of rows
    return this.maxRows;

}
```

In MainActivity.java

Now all we have to do is bind the adapter to the **RecyclerView**

13. Add these 2 member variables

```
public class MainActivity extends AppCompatActivity {

       RecyclerView rv;
       RecyclerView_Adapter rva;   //not Richmond VA, Recyclyer View Adapter
```

14.In on create bind the RecyclerView to the RecyclerView_Adapter.
Also tell the activity how you want it laid out, in a grid or a list.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);


     //get a ref to the viewpager
    rv=findViewById(R.id.rvContacts);
    //create an instance of the swipe adapter
    rva  = new RecyclerView_Adapter(this);
    //set this viewpager to the adapter
    rv.setAdapter(rva);

}
```

Display the data

15. Not done yet, how do you want to display the data?
You can choose between a grid and a linear layout below.

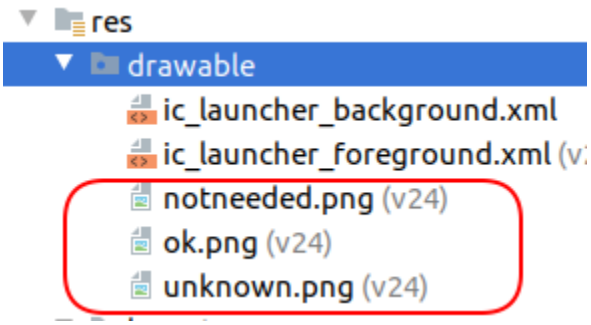
```
     @Override

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        //get a ref to the viewpager
        rv=findViewById(R.id.rvNumbs);
        //create an instance of the swipe adapter
        rva  = new RecyclerView_Adapter(this);
        //set this viewpager to the adapter
        rv.setAdapter(rva);
       // Setup layout manager for items with orientation
        // Also supports `LinearLayoutManager.HORIZONTAL`
```

```
        //LinearLayoutManager layoutManager = new LinearLayoutManager(this,
LinearLayoutManager.VERTICAL, false);
        GridLayoutManager layoutManager = new GridLayoutManager(this,
1,LinearLayoutManager.VERTICAL, false);
        // Optionally customize the position you want to default scroll to
         layoutManager.scrollToPosition(0);
         // Attach layout manager to the RecyclerView
         rv.setLayoutManager(layoutManager);
    }
```

STOP HERE UNTIL NEXT TIME!

Lets make each row view a little snazzier!

Find the 3 drawables in the List_and_RecyclerView drawable folder and add them to your apps drawable folder.



Now add a new row_layout to the res/layoutfolder. Call it row_layout2

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="65dp"
        android:layout_height="65dp"
        android:layout_centerVertical="true"
        android:layout_margin="5dp"
        android:padding="3dp"
        android:scaleType="fitXY"
        android:src="@drawable/unknown" />
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal"
            android:gravity="center_vertical">
            <TextView
                android:id="@+id/tvInfo"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="5dp"
                android:maxLines="1"
                android:text="number"
                android:textColor="@android:color/black"
                android:textSize="15dp"
                android:textStyle="bold" />
            <TextView
                android:id="@+id/tvResult"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_alignParentRight="true"
```

```
                android:layout_marginTop="5dip"
                android:layout_marginRight="10dp"
                android:text="$$$$"
                android:textColor="@android:color/holo_red_light"
                android:textSize="15dp" />
        </LinearLayout>
</LinearLayout>
```

And finally lets make some changes to the RecyclerView_Adapter to accommodate this new layout;

```
    class RowViewHolder extends RecyclerView.ViewHolder {
    TextView tvInfo;
    TextView tvResult;
    ImageView iv;

    public RowViewHolder(@NonNull View itemView) {
        super(itemView);
        tvInfo = (TextView)itemView.findViewById(R.id.tvInfo);
        tvResult = (TextView)itemView.findViewById(R.id.tvResult);
        iv=(ImageView)itemView.findViewById(R.id.imageView1);
    }
}


    @NonNull
    @Override
    public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
        //call this when we need to create a brand new PagerViewHolder
        View view = li.inflate(R.layout.row_layout2, parent, false);
        return new RowViewHolder(view);
    }
    @Override
    public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {
        //passing in an existing instance, reuse the internal resources
        //pass our data to our ViewHolder.
        RowViewHolder viewHolder = (RowViewHolder) holder;
        viewHolder.iv.setImageResource(R.drawable.ok);
        viewHolder.tvInfo.setText(Integer.toString(position) + " squared =");
        viewHolder.tvResult.setText(Integer.toString(position*position));

    }
    @Override
    public int getItemCount() {
        return this.maxRows;
    }
}
```

Run the app to see the result

Now lets do multithreaded

Heavy lifting time - Lets do the calcs in a thread and update the recyclerview at a later time. Why? Because often screens consists of easy to get data, like the image number, and hard to get data, like the result of the calculation.

You can't pause the RecyclerView_Adapter pipeline while waiting on calculation. You would be locked to a particular view waiting for the calculation to complete before you move on.

So;

- **generate and show all the easy to get stuff,**
- **show a temp image while waiting for calc to complete and ?? for the result**
- **launch a thread to get the time consuming stuff**
- **when the thread finishes it will update the appropriate view.**

The Adapter (RecyclerView_Adapter)

modify the RowViewHolder

```
class RowViewHolder extends RecyclerView.ViewHolder {
    private static final int UNINITIALIZED = -1;
    int numb = UNINITIALIZED;
    TextView tvInfo;
    TextView tvResult;
    ImageView iv;
    public RowViewHolder(@NonNull View itemView) {
        super(itemView);
        tvInfo = (TextView)itemView.findViewById(R.id.tvInfo);
        tvResult = (TextView)itemView.findViewById(R.id.tvResult);
        iv=(ImageView)itemView.findViewById(R.id.imageView1);
    }

}
```

Create inner class asynctask in RecyclerView_Adapter :

It just sleeps for a bit and then calculates the square of the number
Problem: What if in between launching the thread that retreives the image and the image finally being retreived, the user swipes the view off the screen? Would the PageViewHolder be reused and point to another image after the thread returns?

Maybe, so you must guard against this!

How?

- **have the thread keep track of what its calculating,**
- **when the thread is done, see if what it calculated is the same thing that the PagerViewHolder says is being calculated (if not the PagerViewHolder has been recycled, discard the threads result).**

```
private class GetNumber extends AsyncTask<Void, Void, Integer> {
    //ref to a viewholder, this could change if
    //RowViewHolder myVH is recycled and reused!!!!!!!!!
    private RowViewHolder myVh;
    //since myVH may be recycled and reused
    //we have to verify that the result we are returning
    //is still what the viewholder wants
    private int original_number;
    public GetNumber(RowViewHolder myVh) {
        //hold on to a reference to this viewholder
        //note that its contents (specifically iv) may change
        //iff the viewholder is recycled
        this.myVh = myVh;
        //make a copy to compare later, once we have the image
        this.original_number = myVh.numb;
    }
    @Override
    protected Integer doInBackground(Void... params) {
        //just sleep for a bit to simulate long running downloaded
        //but could just as easily make a network call
        try {
            Thread.sleep(100); //sleep for 2 seconds
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return original_number*original_number;
    }
    @Override
    protected void onPostExecute(Integer param) {
        //got a result, if the following are NOT equal
        // then the view has been recycled and is being used by another
        // number DO NOT MODIFY
        if (this.myVh.numb == this.original_number){
            //still valid
            //set the result on the main thread
            myVh.iv.setImageResource(R.drawable.ok);
            myVh.tvInfo.setText(Integer.toString(this.myVh.numb) + " squared =");
            myVh.tvResult.setText(Integer.toString(param));
         }
        else{
            myVh.iv.setImageResource(R.drawable.notneeded);
            myVh.tvInfo.setText("DANG! work wasted");
            myVh.tvResult.setText("");
         }
    }
}
```

Update **RowViewHolder** so it can track what number it is operating on

```
class RowViewHolder extends RecyclerView.ViewHolder {
     private static final int UNINITIALIZED = -1;
     int numb = UNINITIALIZED;
     TextView tvInfo;
     TextView tvResult;
     ImageView iv;
     public RowViewHolder(@NonNull View itemView) {
          super(itemView);
          tvInfo = (TextView)itemView.findViewById(R.id.tvInfo );
          tvResult = (TextView)itemView.findViewById(R.id.tvResult );
          iv=(ImageView)itemView.findViewById(R.id.imageView1);
     }
}
```

And finaly modify onBindViewHolder to default load error image, then launch a thread which will load real image after a wait

```
public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {
       //passing in an existing instance, reuse the internal resources
       //pass our data to our ViewHolder.
       RowViewHolder viewHolder = (RowViewHolder) holder;
       viewHolder.numb= position;

       //initialize the UI
       viewHolder.iv.setImageResource(R.drawable.unknown);
       viewHolder.tvInfo.setText("Hold on a sec...");
       viewHolder.tvResult.setText("");

       //launch a thread to 'retreive' the image
       GetNumber myTask = new GetNumber(viewHolder);
       myTask.execute();

}
```