

Have all images in the repo

ViewPager in class lab

See ViewPager3 class demo

with and without threaded pages

1. First create a project (Use the one with a Floating Action Button (FAB)) because it gives you an appBar.
2. Get rid of FAB in MainActivity.java and in MainActivity.xml
Get rid of the fragments (java) and their layouts(XML)
Test to see if working
3. Place images of interest in res/drawable (p0,p1,p2,p3,p4,p5)

The XML

4. Add the viewPager widget to activity_main.xml (replace the content main). Be sure to give it an ID

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">
    <com.google.android.material.appbar.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">
        <androidx.appcompat.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />
    </com.google.android.material.appbar.AppBarLayout>
    <androidx.viewpager2.widget.ViewPager2
        android:id="@+id/view_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="@color/colorAccent">
    </androidx.viewpager2.widget.ViewPager2>
</LinearLayout>
```

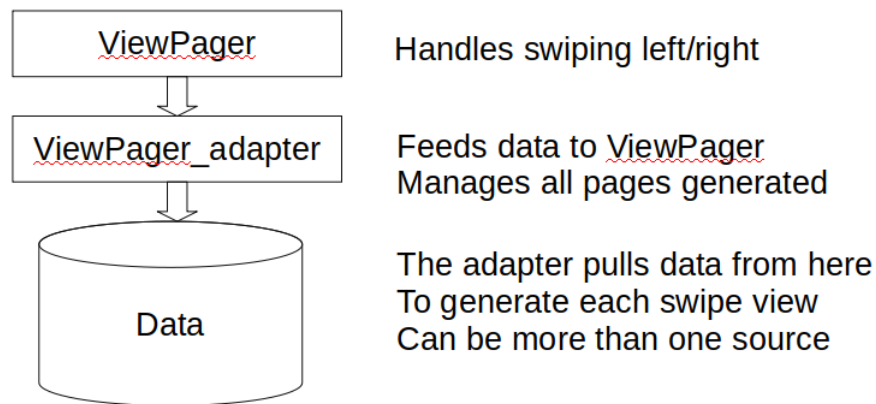
5. Need a layout to define what each page displayed in the viewPager looks like. Here we will have an image and a text (see ViewPager3 for page look demo).

In layout folder create swipe_layout (or any name you want) (from Layout folder→right click →new→xml→layout XML file. Give it a name and (choose FrameLayout for layout type)

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

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:id="@+id/imageView"
        android:scaleType="fitXY"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <TextView
        android:id="@+id/tv"
        android:layout_width="match_parent"
        android:text="sample text"
        android:textSize="60dp"
        android:textStyle="bold"
        android:gravity="center"
        android:layout_gravity="bottom"
        android:layout_height="120dp"
        android:background="#33777777" />
</FrameLayout>
```

The Adapter



6. Create an adapter (the brains of the operation). It supplies the ViewPager with 1 page of data at a time whose appearance is defined by `swipe_layout` above.

Create a new JavaClass `ViewPager2_Adapter`(or any name you like) and have it extend...

```
public class ViewPager2_Adapter extends RecyclerView.Adapter
```

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

7. `ViewPager2_Adapter` is going to serve up images, so add the list of images in the drawable folder to `ViewPager2_Adapter` as member variable array.

```
private int[] image_resources =  
{ R.drawable.p0,R.drawable.p1,R.drawable.p2,R.drawable.p3,R.drawable.p4,R.drawable.p5 };
```

8.Each time a user swipes on a Viewpager image a new image slides in, that image consists of a `swipe_layout` that will be populated with images from `image_resources` and text. But first we have to create it. For that we need a layout inflater (remember its use in the spinner project?). Add one to to `ViewPager2_Adapter` as member variable

```
private final LayoutInflater li;
```

9. And we need a context to get this inflater. Add one to to `ViewPager2_Adapter` as member variable.

```
private final Context ctx;
```

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

```
public ViewPager2_Adapter(Context ctx){  
  
    this.ctx=ctx;  
    li=(LayoutInflater)ctx.getSystemService(Context.LAYOUT_INFLATER_SERVICE);  
}
```

11. Add a RecyclerView.ViewHolder to the class

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

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

```
class PagerViewHolder extends RecyclerView.ViewHolder {  
  
    ImageView iv;  
    TextView tv;  
  
    //with a view in onBindViewHolder  
    public PagerViewHolder(@NonNull View itemView) {  
        super(itemView);  
        iv = (ImageView)itemView.findViewById(R.id.imageView);  
        tv = (TextView)itemView.findViewById(R.id.tv);  
    }  
}
```

12. 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.swipe_layout, parent, false);  
    return new PagerViewHolder(view); //the new one  
}
```

13. 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.  
    PagerViewHolder viewHolder = (PagerViewHolder) holder;  
    viewHolder.iv.setImageResource(image_resources[position]);  
    viewHolder.tv.setText("Image : " + position);  
}
```

14. The ViewPager2_Adapter has to know how many items it will hold

```

public int getItemCount() {

    //the size of the collection that contains the items we want to display
    return image_resources.length;
}

```

In MainActivity

Now all we have to do is bind the adapter to the viewPager2

15. Add these 2 member variables

```

public class MainActivity extends AppCompatActivity {

    ViewPager2 vp;
    ViewPager2_Adapter csa;
}

```

16. In on create bind the viewPager

```

@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
    vp=findViewById(R.id.view_pager);
    //create an instance of the swipe adapter
    csa = new ViewPager2_Adapter(this);
    //set this viewPager to the adapter
    vp.setAdapter(csa);
}

```

Now lets do multithreaded

Heavy lifting time - Lets retrieve the images 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 an image located on another server.

You can't pause the ViewPager2_Adapter pipeline while waiting to download the image (what would a http timeout do to your apps performance? You would be locked to a particular view waiting for the network request to complete before you move on).

So;

- generate and show all the easy to get stuff,
- show a temp image while waiting for real image to be downloaded
- launch a thread to get the time consuming stuff
- when the thread finishes it will update the appropriate view.

First add a waiting image

got error.png from another project placed in drawable

The Adapter (ViewPager2_Adapter)

modify the PagerViewHolder

```
class PagerViewHolder extends RecyclerView.ViewHolder {  
  
    private static final int UNINITIALIZED = -1;  
    ImageView iv;  
    TextView tv;  
    int position=UNINITIALIZED;    //start off uninitialized, set it when we are  
    populating  
    //with a view in onBindViewHolder  
    public PagerViewHolder(@NonNull View itemView) {  
        super(itemView);  
    }  
}
```

```

        iv = (ImageView)itemView.findViewById(R.id.imageView);
        tv = (TextView)itemView.findViewById(R.id.tv);
    }
}

```

Create inner class AsyncTask in ViewPager2_Adapter :

It just sleeps for a bit and then loads the proper image

Problem: What if in between launching the thread that retrieves the image and the image finally being retrieved, the user swipes the view off the screen? Would the PagerViewHolder 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 downloading,
- when the thread is done, see if what it downloaded is the same thing that the PagerViewHolder says is being downloaded (if not the PagerViewHolder has been recycled, discard the threads result).

```

private class GetImage extends AsyncTask<Void, Void, Void> {
    //ref to a viewholder, this could change if
    //PagerViewHolder myVH is recycled and reused!!!!!!!
    private PagerViewHolder 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_position;

    public GetImage(PagerViewHolder 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_position = myVh.position;
    }
    @Override
    protected Void 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(2000); //sleep for 2 seconds

```

```

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return null;
    }
    @Override
    protected void onPostExecute(Void 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.position == this.original_position){
            //still valid
            //set the result on the main thread
            myVh.iv.setImageResource(image_resources[this.myVh.position ]);
        }
        else
            Toast.makeText(ViewPager2_Adapter.this.ctx,"YIKES! Recycler view reused, my result
is useless", Toast.LENGTH_SHORT).show();
    }
}

```

And finally 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.
    PagerViewHolder viewHolder = (PagerViewHolder) holder;
    //set to some default image
    viewHolder.iv.setImageResource(R.drawable.error);
    viewHolder.tv.setText("Image : " + position);

    //remember which image this view is bound to
    viewHolder.position=position;

    //launch a thread to 'retriev' the image
    GetImage myTask = new GetImage(viewHolder);
    myTask.execute();
}

```