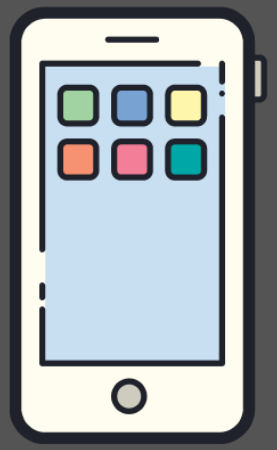


MOBILE DEVELOPMENT

Data-Driven Views

Outline

- Recall: MVC
- Motivation for ListViews
- General Pattern for ListViews
- RecyclerViews
  - Components
  - Implementation

Recall: Model-View-Controller Pattern

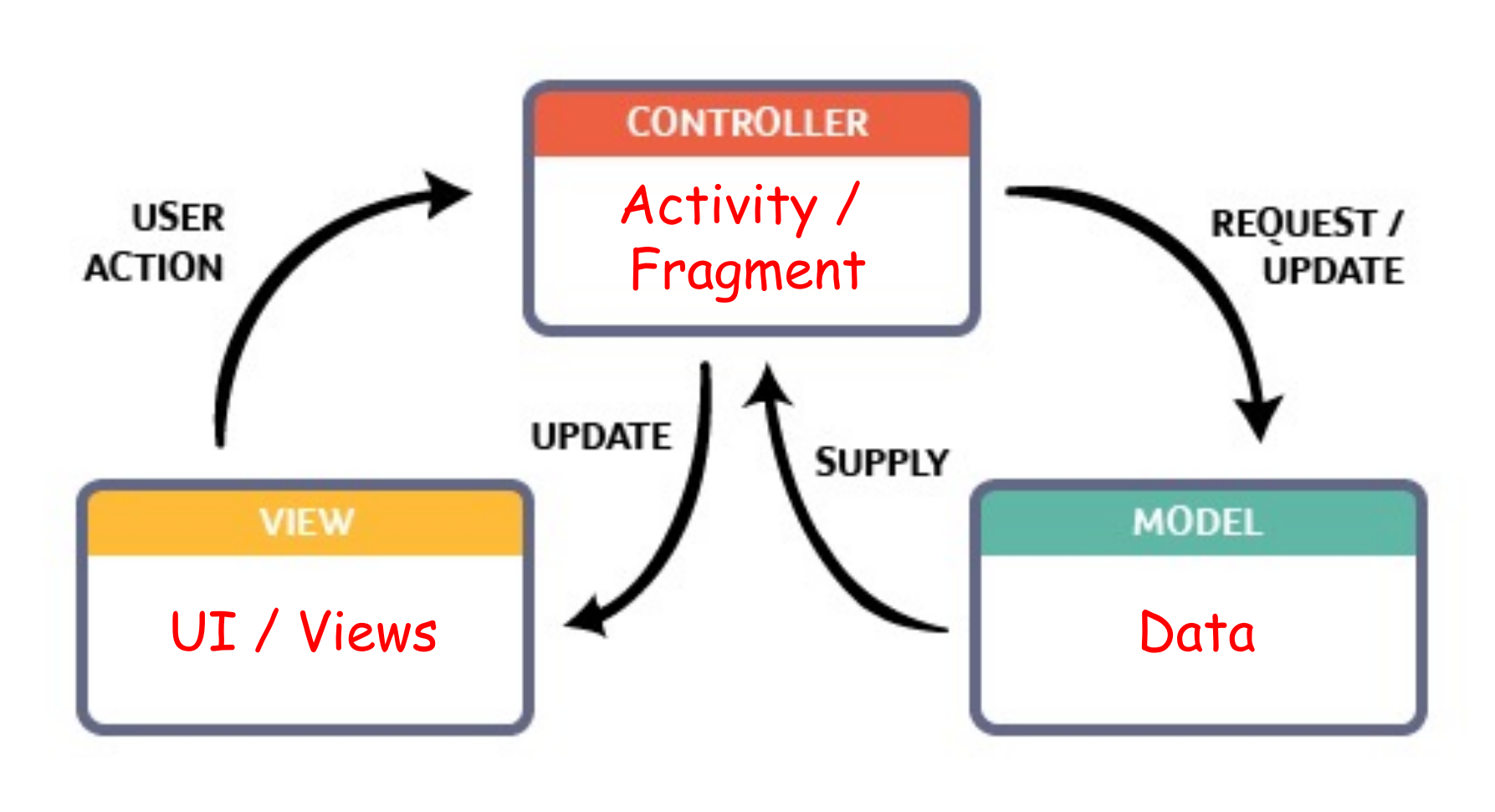


Image: *https://hackernoon.com/beginners-guide-to-ruby-on-rails-mvc-model-view-controller-pattern-4z19196a*

Situation: You have to populate an app's feed (e.g. Instagram, Facebook, Twitter) with 40 photos, what do you do?

Image: https://www.femalefirst.co.uk/features/the-best-instagram-apps-to-wow-your-followers-1211434.html

And if you follow a lot of people, how many profile pictures would be here?

Situation: You have to populate an app's feed (e.g. Instagram, Facebook, Twitter) with 40 photos, what do you do?

Solution: Create a view for each photo

Does anyone see a problem with this design?

Problem: Those photos get stored in memory… what if this scales?
And what happens if the user scrolls past 40? Load everything until the RAM is full? Delete views? (<-more computation)

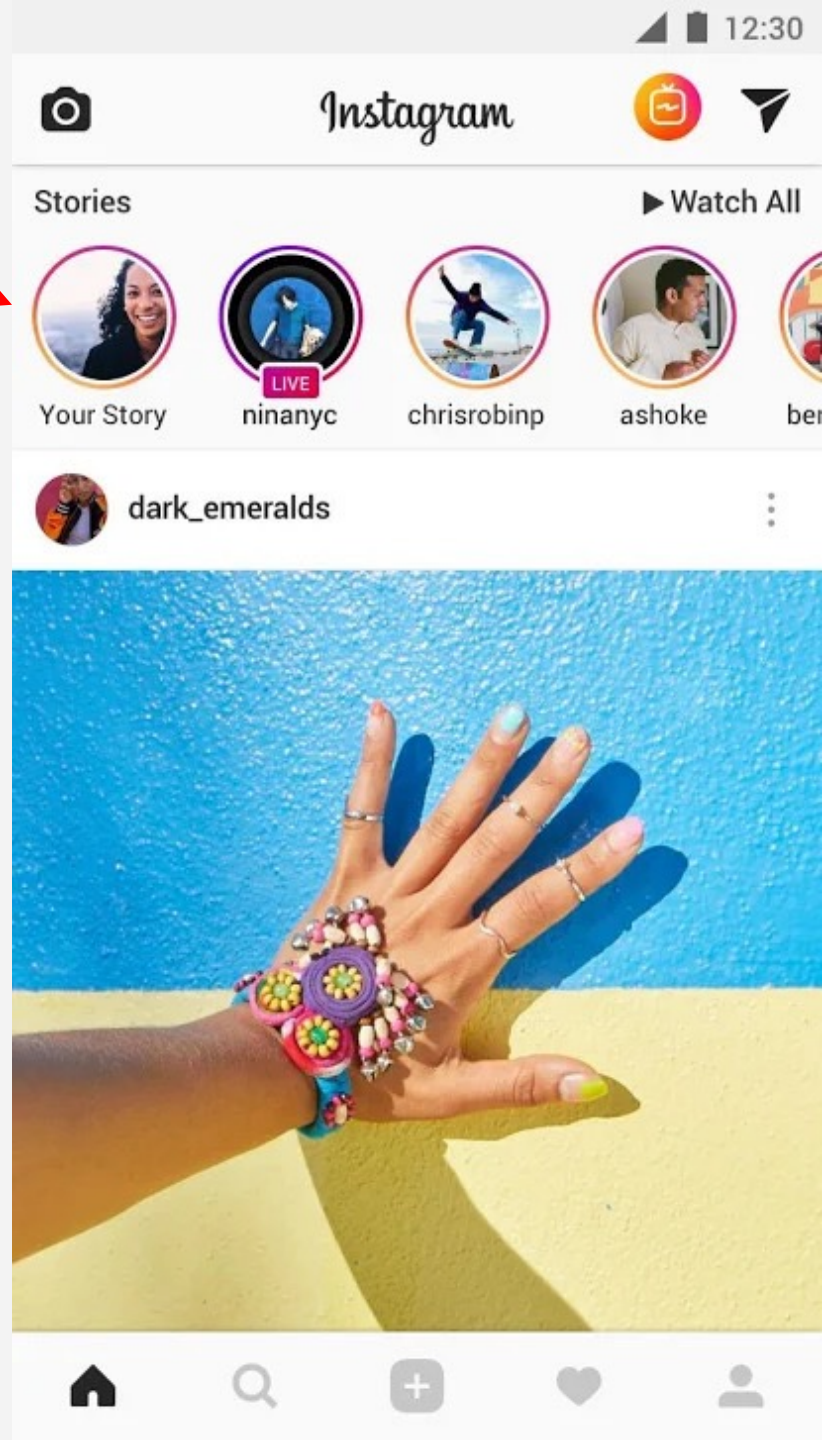
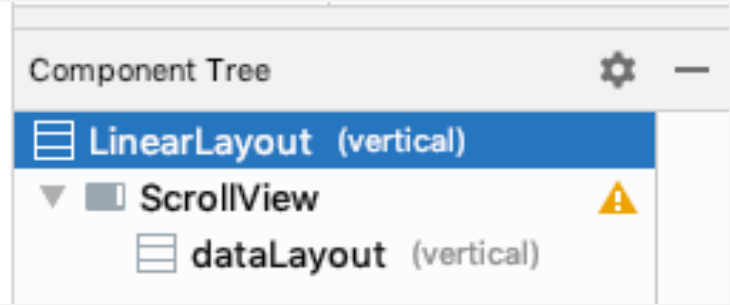


Image: https://www.femalefirst.co.uk/features/the-best-instagram-apps-to-wow-your-followers-1211434.html

Let's take a simpler example…



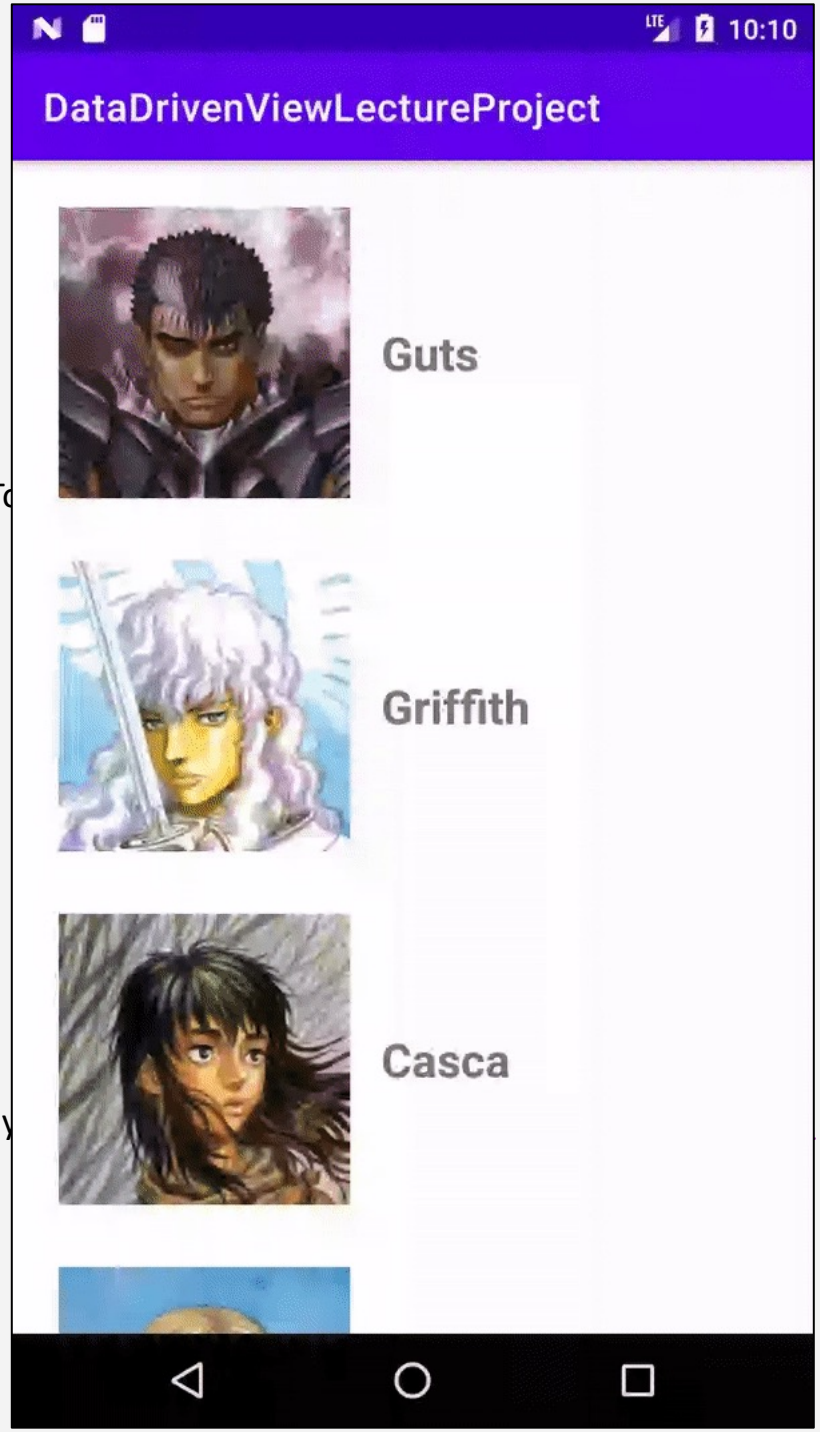

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    populate_data();
    initializeUiElements();
    populateLayoutWithViews();
}
```

```
private void populateLayoutWithViews() {
    for(Character c : characterArrayList) {
        // Prepare the character's layout
        LinearLayout hll = new LinearLayout(this);
        hll.setGravity(Gravity.CENTER_VERTICAL);
        hll.setPadding(convertPxToDp(16),convertPxTo

        // Prepare the image
        ImageView icon = new ImageView(this);
        icon.setImageResource(c.getImageId());
        icon.setAdjustViewBounds(true);
        icon.setMaxHeight(convertPxToDp(150));
        icon.setMaxWidth(convertPxToDp(150));
        // Add the image to the LinearLayout
        hll.addView(icon);

        // Prepare the text
        TextView name = new TextView(this);
        name.setText(c.getName());
        LinearLayout.LayoutParams lp = new LinearLay
        lp.setMarginStart(convertPxToDp(16));
        name.setLayoutParams(lp);
        name.setTextSize(24);
        name.setTypeface(null, Typeface.BOLD);
        // Add the text to the LinearLayout
        hll.addView(name);
```



Take note: the Character class contains an integer for the image Id and a string for the character's name

```java
private void populateLayoutWithViews() {
  for(Character c : characterArrayList) {
    // Prepare the character's layout
    LinearLayout hll = new LinearLayout(this);
    hll.setGravity(Gravity.CENTER_VERTICAL);
    hll.setPadding(convertPxToDp(16),convertPxToDp(16),convertPxToDp(16),convertPxToDp(16));

    // Prepare the image
    ImageView icon = new ImageView(this);
    icon.setImageResource(c.getImageId());
    icon.setAdjustViewBounds(true);
    icon.setMaxHeight(convertPxToDp(150));
    icon.setMaxWidth(convertPxToDp(150));
    // Add the image to the LinearLayout
    hll.addView(icon);

    // Prepare the text
    TextView name = new TextView(this);
    name.setText(c.getName());
    LinearLayout.LayoutParams lp = new LinearLayout.LayoutParams(ViewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.LayoutParams.WRAP_CONTENT);
    lp.setMarginStart(convertPxToDp(16));
    name.setLayoutParams(lp);
    name.setTextSize(24);
    name.setTypeface(null, Typeface.BOLD);
    // Add the text to the LinearLayout
    hll.addView(name);

    // Add the character's layout to the layout in the scroll view
    this.dataLayout.addView(hll);
  }
}
```

This type of approach produces too much code and scales horribly [O(n)]

More Appropriate Method…

- If you search for methods to display data online, you might stumble upon ListViews / GridViews…

1 – create view

<default text>

2 – bind data

alpha
beta
charlie
delta
echo
foxtrot
golf
hotel

Alpha

Alpha

Beta

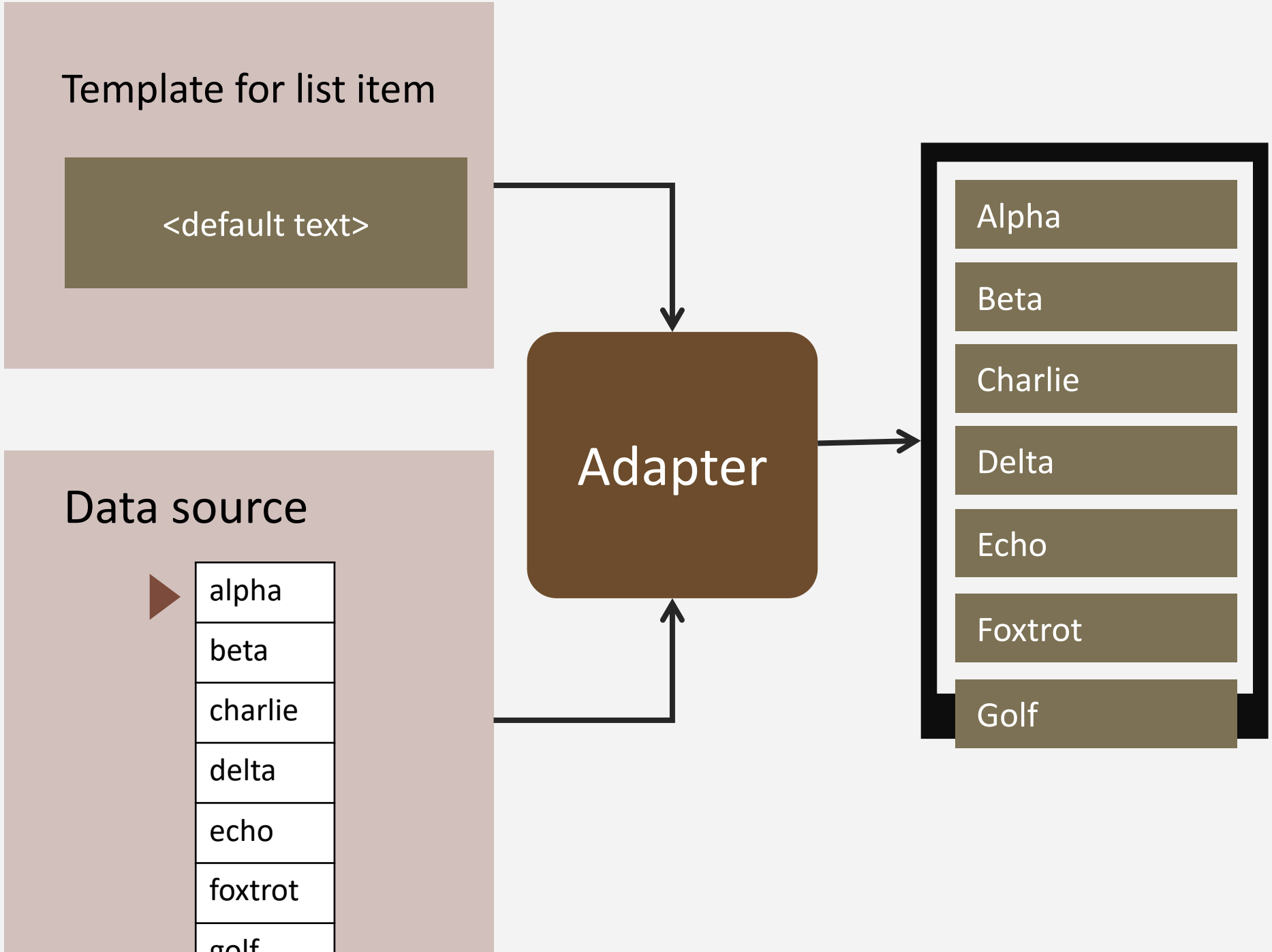
Charlie

Delta

Echo

Foxtrot

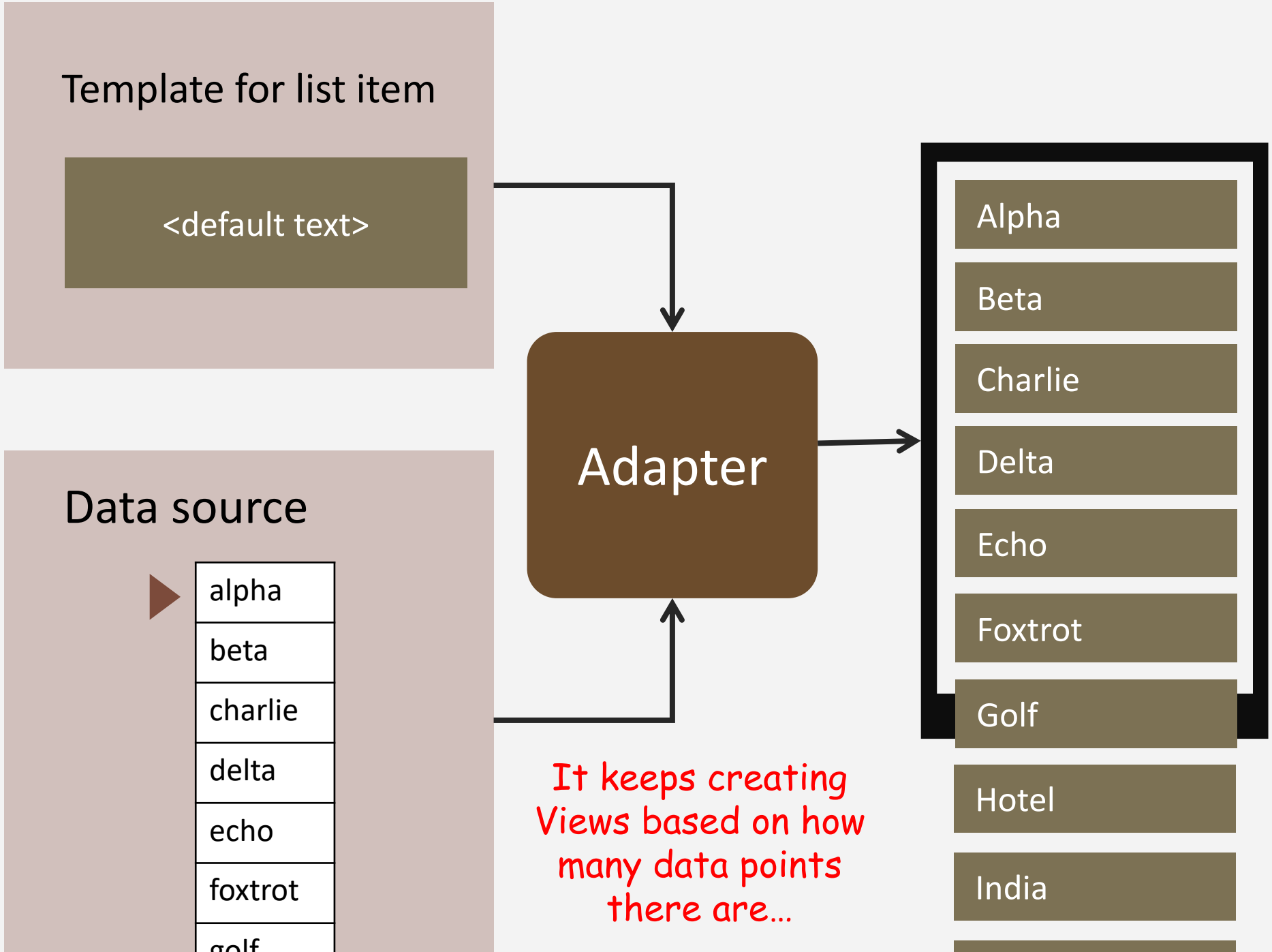
Golf


Template for list item
<default text>
Data source
alpha
beta
charlie
delta
echo
foxtrot
golf
Adapter
Alpha
Beta
Charlie
Delta
Echo
Foxtrot
Golf

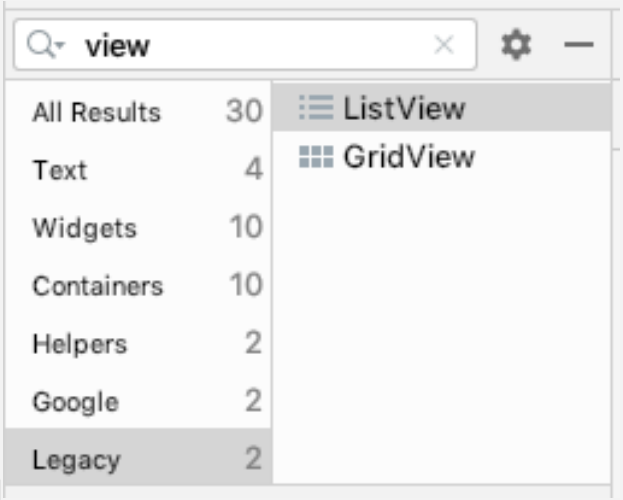
More Appropriate Method…

- …ListViews / GridViews
  - They make life slightly easier by integrating a Layout file instead of needing to code all attributes

So what's the catch?


Template for list item
<default text>
Data source
alpha
beta
charlie
delta
echo
foxtrot
golf
Adapter
Alpha
Beta
Charlie
Delta
Echo
Foxtrot
Golf
Hotel
India
It keeps creating Views based on how many data points there are...

More Appropriate Method…



- …ListViews / GridViews
  - They make life slightly easier by integrating a Layout file instead of needing to code all attributes
    - <u>But you're still going to create many views…</u>
  - These views are also considered *ancient*
    - You can still implement them…
- However, RecyclerViews are the way to go now ☺

RecyclerViews

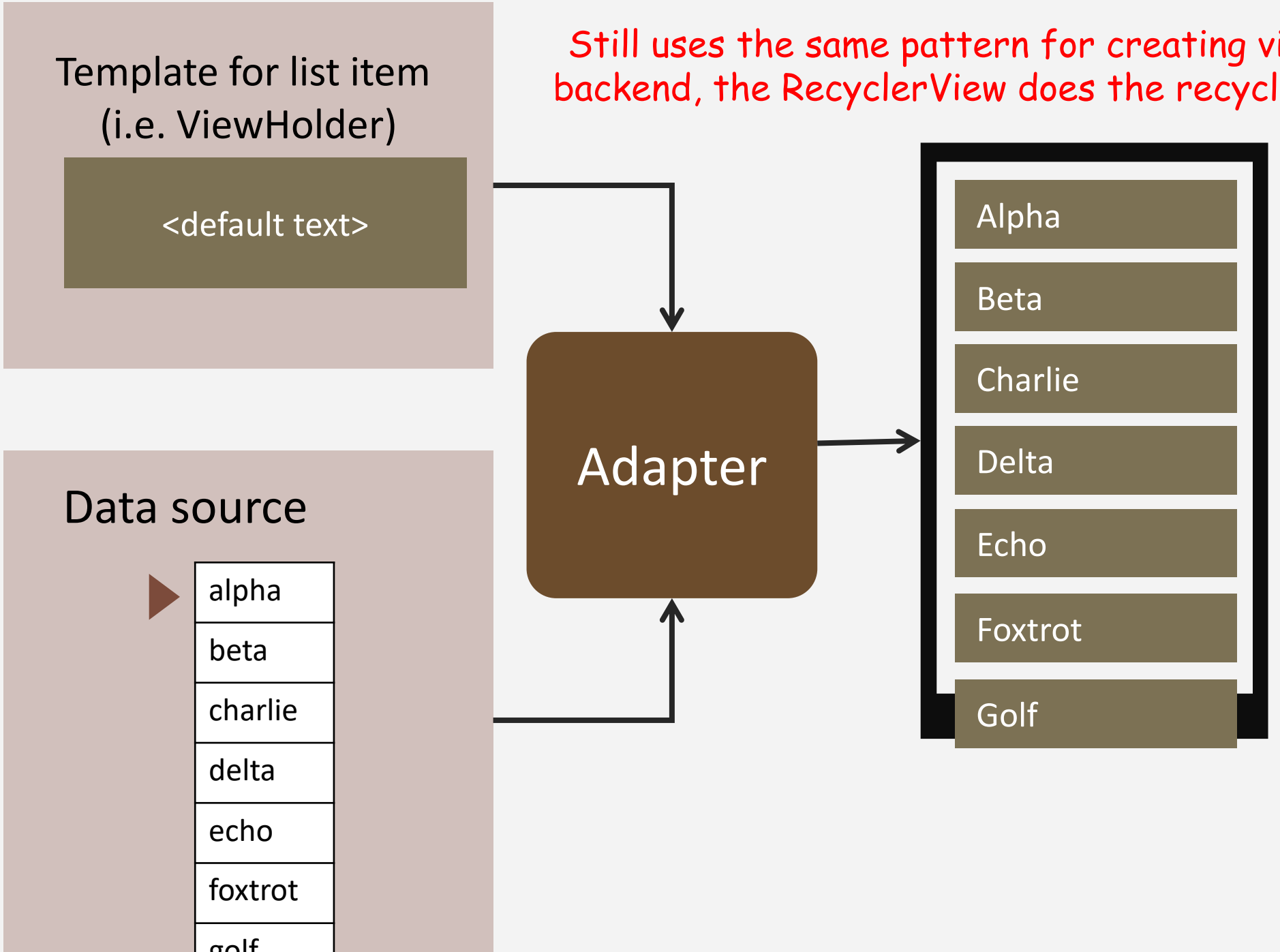
- Is a ViewGroup
- Is a more advanced, robust version of a ListView
  - Can display in list (1d) or grid format (2d)
- Dynamically creates and reuses views when they're needed
- Also makes use of the Adpater+ViewHolder Pattern
  - However, we want to avoid using `findViewById()` too many times since this method is computationally expensive

We will also look at a ViewBinding approach to this, but in our next session ☺

Data that's not immediately to be shown isn't bound to a template view

Intuition is to only bind data when its needed and only create a certain numbers of views based on how many are needed on the screen (i.e. to **recycle** views)

Still uses the same pattern for creating views, but backend, the RecyclerView does the recycling for us



Recycle in RecyclerView

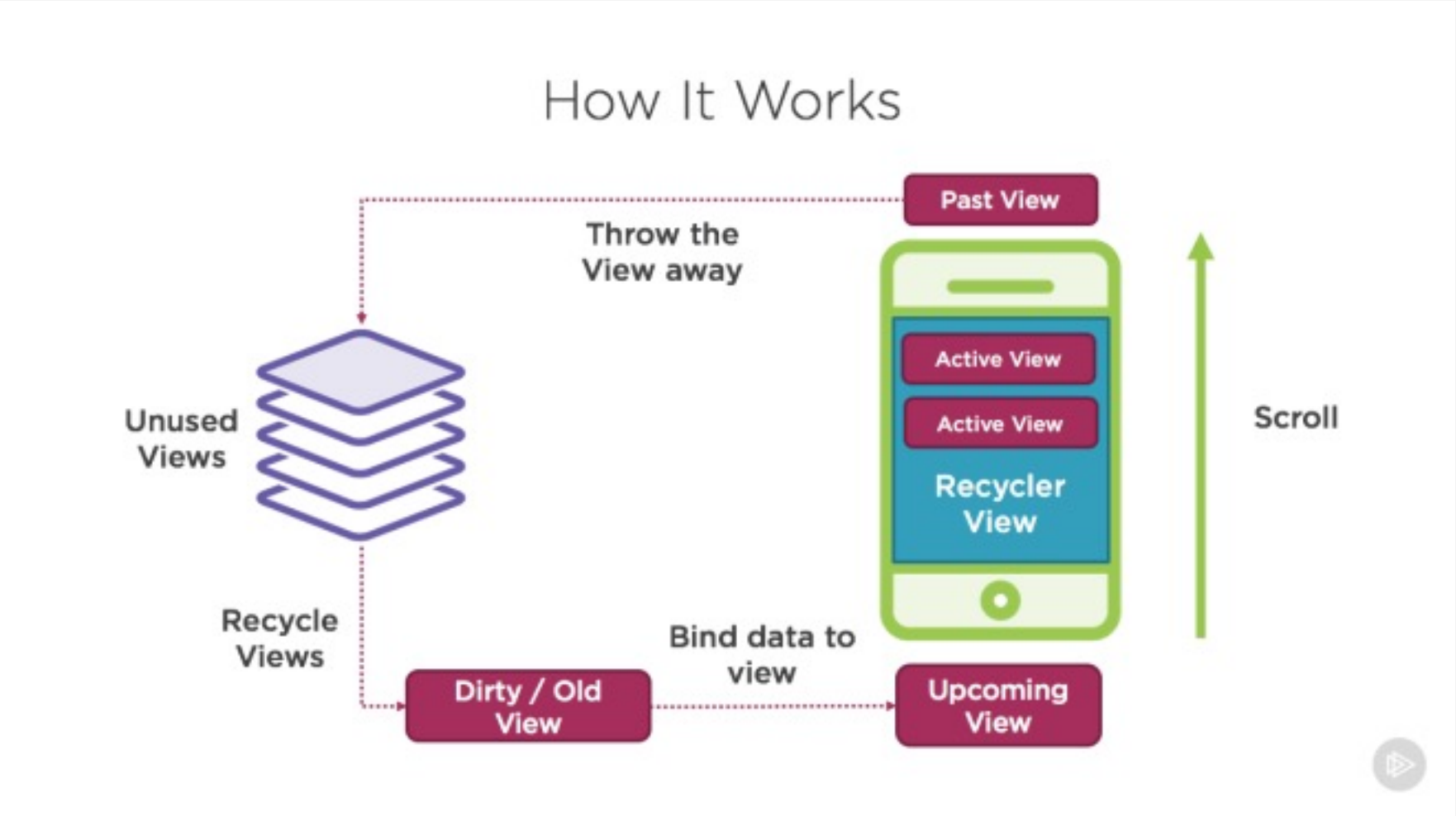
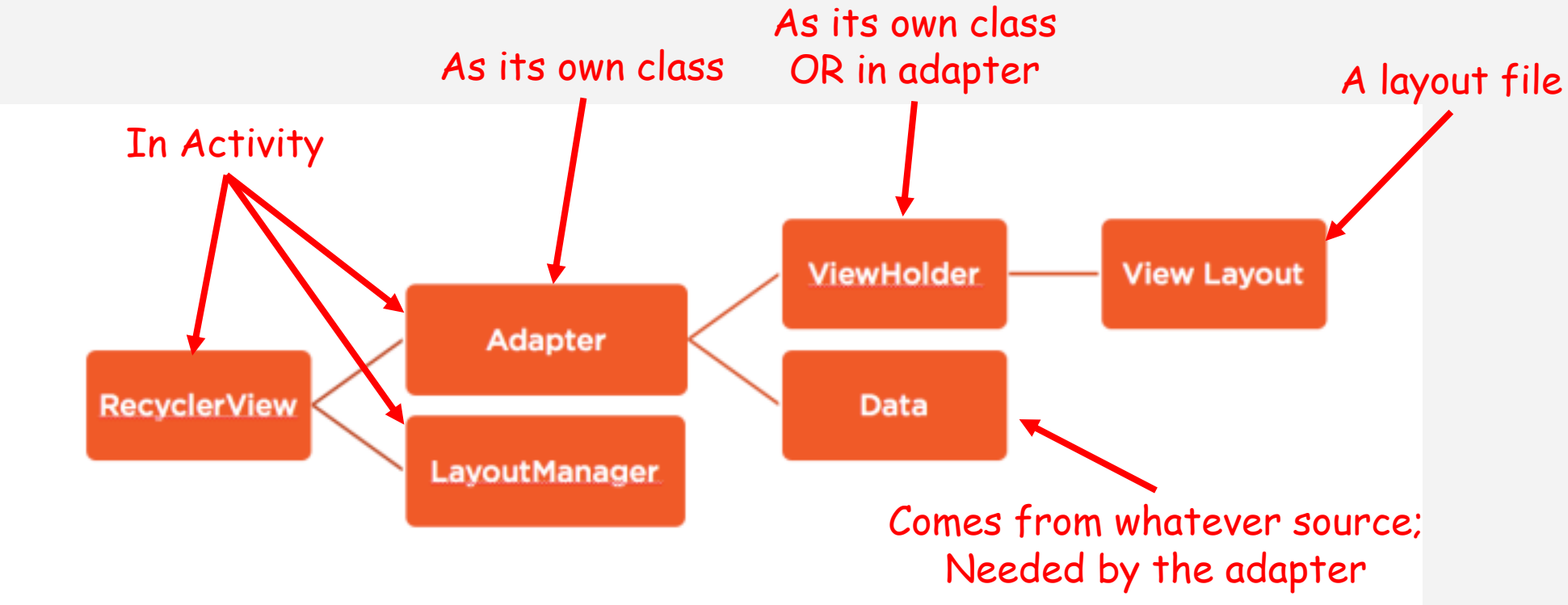


Image from https://sarah3133.wordpress.com/2018/12/07/recylerview/

Questions so far?

Components we need…



You'll also need to add a dependency in your Gradle. Add the following:

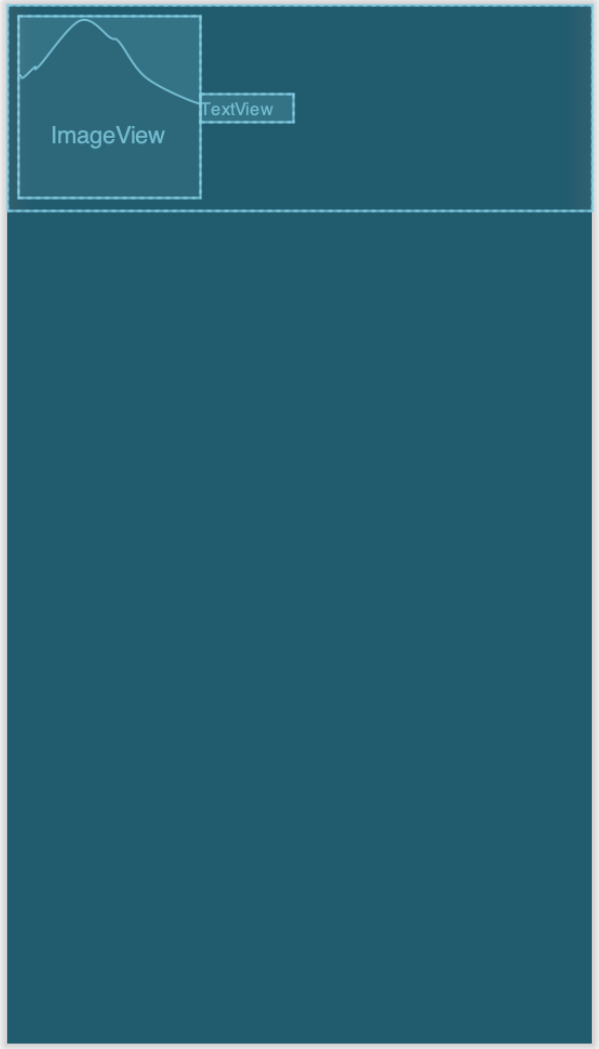
```
dependencies {
    … other dependencies …
    implementation "androidx.recyclerview:recyclerview:1.2.1"
}
```

Tho its better to check Android Documentation for the most updated version ☺

Image from https://sarah3133.wordpress.com/2018/12/07/recylerview/

1. Define your **Layout** in an XML file

- Instead of dynamically creating the ViewGroup, create the UI as if you were creating a Layout for an Activity
- For our example, we just need an ImageView and a TextView



2. Declare your ViewHolder

Inherit from the appropriate class

```
public class MyViewHolder extends RecyclerView.ViewHolder {
    private ImageView iv;
    private TextView tv;

    public MyViewHolder(@NonNull View itemView) {
        super(itemView);
        this.iv = itemView.findViewById(R.id.imageView);
        this.tv = itemView.findViewById(R.id.textView);
    }

    public void setIv(int iv) {
        this.iv.setImageResource(iv);
    }

    public void setTv(String tv) {
        this.tv.setText(tv);
    }
}
```

Declare the View variables needed

Initialize the views

Add in setters so you can bind the data to the ViewHolder's Views

3. Define your **Adapter**

- Your adapter is responsible for the following:
  1. Holding a copy of your data
  2. Creating a ViewHolder
  3. Binding data to the ViewHolder
- First off, create a new Adapter w/ the ViewHolder specified

```
public class MyAdapter extends RecyclerView.Adapter<MyViewHolder> {

}
```

3.1. Getting a copy of **data** to the Adapter

- This part is easy and can be done in multiple ways!
- Assuming your data is in an ArrayList<>…

```
public class MyAdapter extends RecyclerView.Adapter<MyViewHolder> {

    private ArrayList<Character> data;

    public MyAdapter(ArrayList data) {
        this.data = data;
    }
}
```

3.2. Creating the ViewHolder(s)

- Here, we need to inflate the Layout we created in step 1

```
public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View view = inflater.inflate(R.layout.data_layout, parent, false);

    MyViewHolder myViewHolder = new MyViewHolder(view);
    return myViewHolder;
}
```

This takes the parent ViewGroup (i.e. RecyclerView) and inflates a layout within it

Creates our custom ViewHolder and gives it a new View w/ the layout we just inflated

3.3. Binding data to our ViewHolder

- Once we have the ViewHolder created / ready, we bind the data
  - Model → Controller → View

These are the setter we created in our custom ViewHolder

```
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
    holder.setIv(data.get(position).getImageId());
    holder.setTv(data.get(position).getName());
}
```

Position refers to the point in the model's list (i.e. index of the ArrayList)

3.4. Finishing up the Adapter

- Lastly, we need to make sure our Adapter knows how many items are in the Model
  - If this isn't done, nothing will display
  - Don't forget this simple portion!
- Modify the **`getItemCount()`** to return the size / length of your data

```
public int getItemCount() {
    return data.size();
}
```

4. Setup your **RecyclerView**

- Link the RecyclerView in your Activity to the custom Adapter and ViewHolder so its guided properly

```
void setupRecyclerView() {
    this.recyclerView = findViewById(R.id.recyclerView);

    this.myManager = new LinearLayoutManager(this);
    this.recyclerView.setLayoutManager(this.myManager);

    this.myAdapter = new MyAdapter(this.characterArrayList);
    this.recyclerView.setAdapter(this.myAdapter);
}
```

This defines the ViewGroup. In our example, this is a vertical linear layout. You can modify this so that Views are displayed horizontally or in Grid format

This just assigned the Adapter we created to our RecyclerView

Remember our Adapter's constructor? This is where you'd pass in the data.

And there you have it! You should have a working **RecyclerView**! ☺

Some clarifications…

- The ViewHolder class doesn’t have to be separate from the Adapter classes
- You can even do away with separate setter methods in the ViewHolder and exchange them with one “binding” method

- Implement according to what you feel comfortable with ☺

Parting Notes

- RecyclerViews is an efficient View for displaying data
- RecyclerViews require a ViewHolder pattern
  - There's no getting around it…

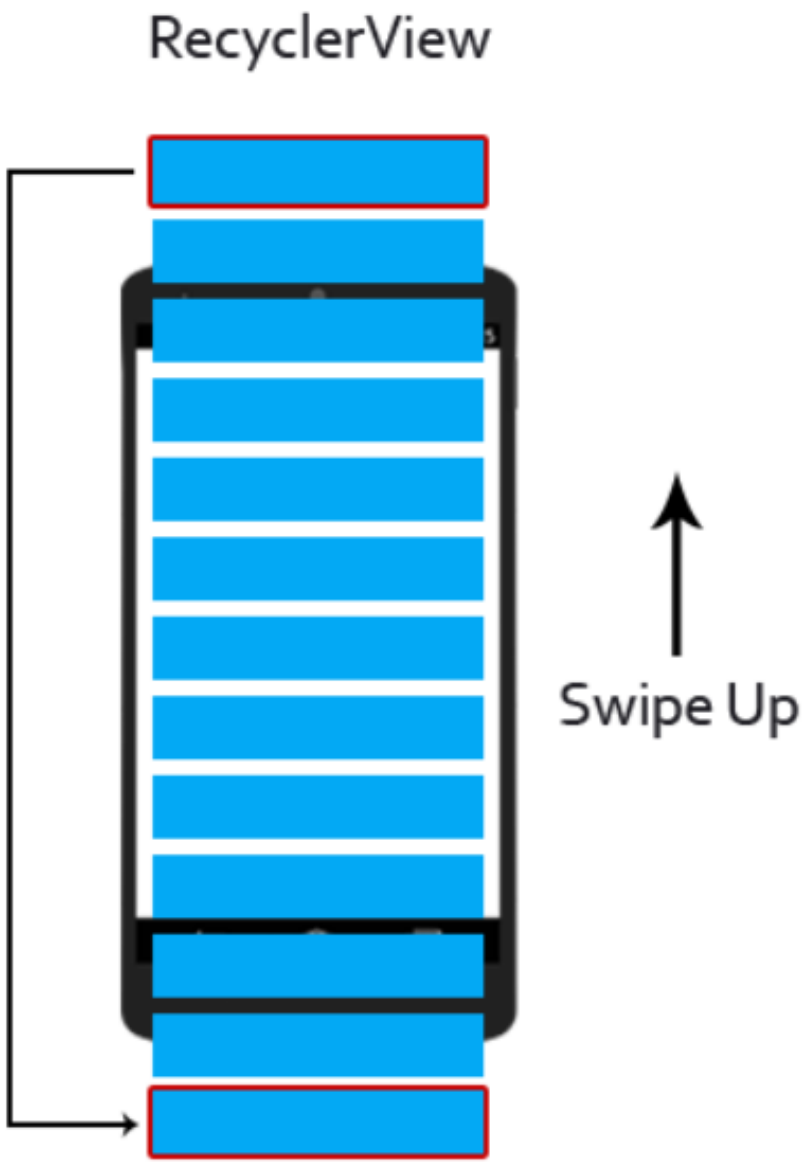


Image modified from https://sarah3133.wordpress.com/2018/12/07/recylerview/

Parting Notes

- RecyclerViews allow for customization
- Here are some common features
  - Swipe to delete
    - https://stackoverflow.com/questions/33985719/android-swipe-to-delete-recyclerview
  - Drag + Swipe
    - https://medium.com/@ipaulpro/drag-and-swipe-with-recyclerview-b9456d2b1aaf
  - Add item dividers
    - https://stackoverflow.com/questions/24618829/how-to-add-dividers-and-spaces-between-items-in-recyclerview

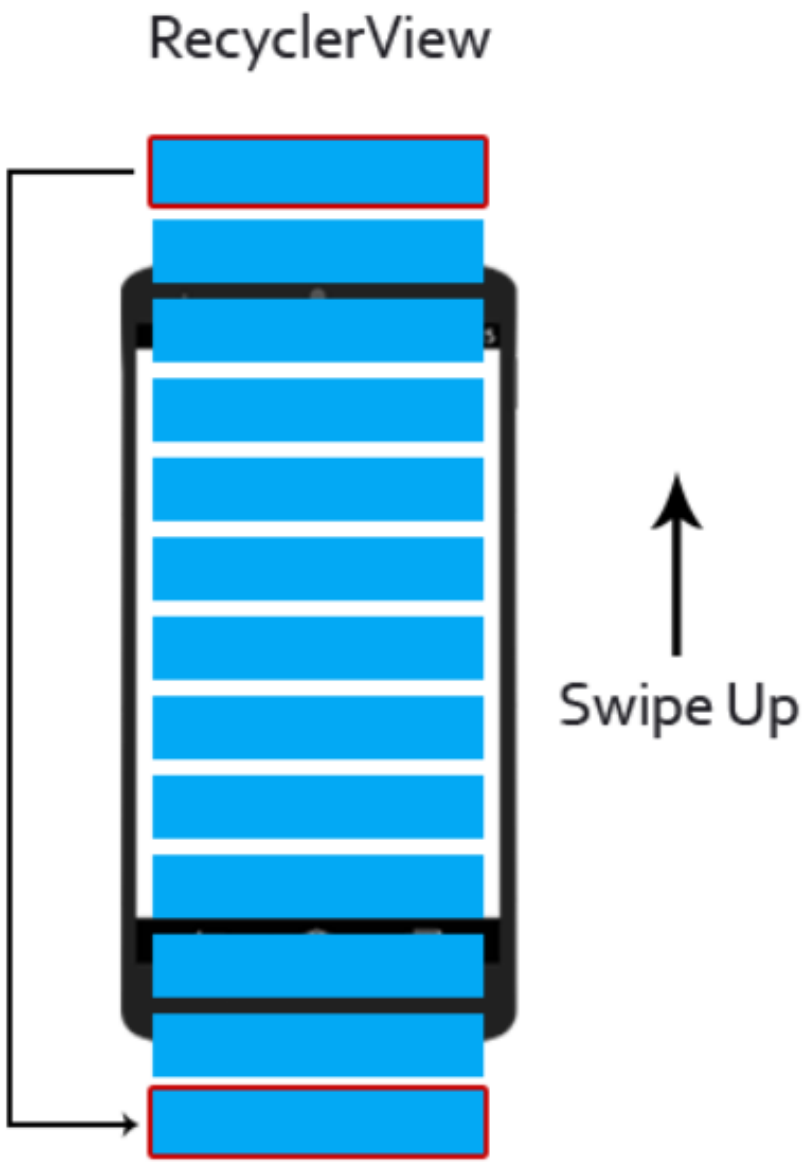
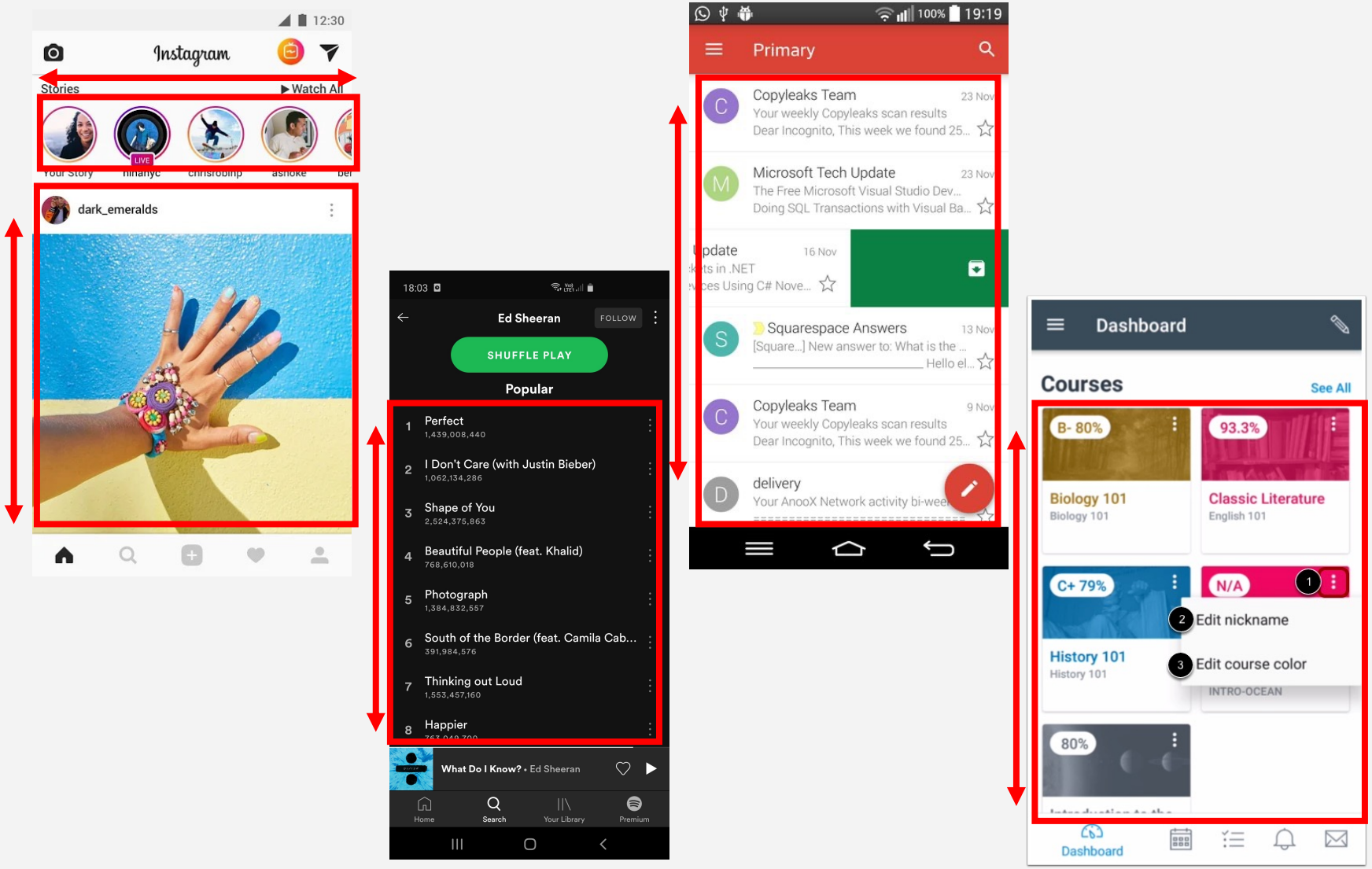


Image modified from https://sarah3133.wordpress.com/2018/12/07/recylerview/

No meme this session. Appreciate the developers that implement RecyclerViews!

Thanks everyone!

See you next meeting! ☺



Images from: https://tacomacc.teamdynamix.com/TDClient/1903/Portal/KB/ArticleDet?ID=51520, https://medium.com/techloop/10-tools-and-widgets-every-android-developer-should-know-about-1a98c089c54e , https://stackoverflow.com/questions/33985719/android-swipe-to-delete-recyclerview , : https://www.femalefirst.co.uk/features/the-best-instagram-apps-to-wow-your-followers-1211434.html