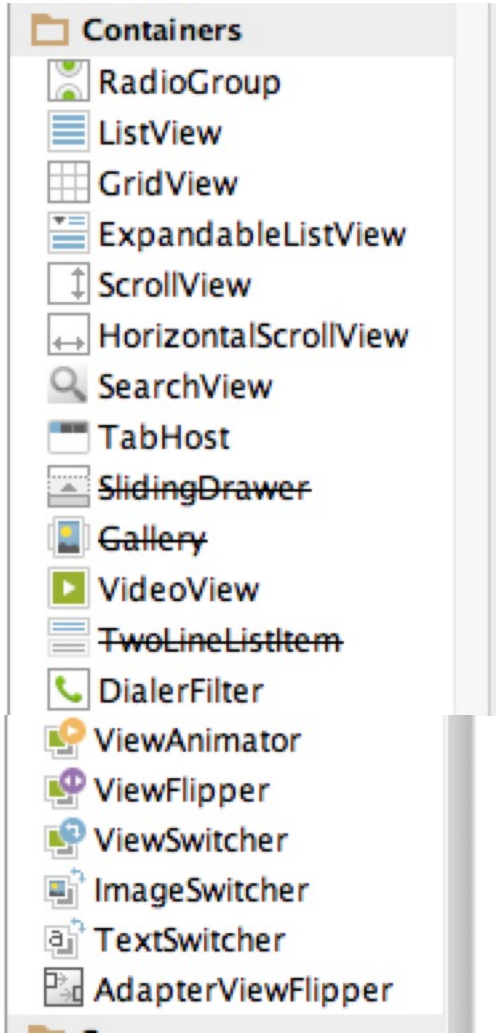
RecyclerView

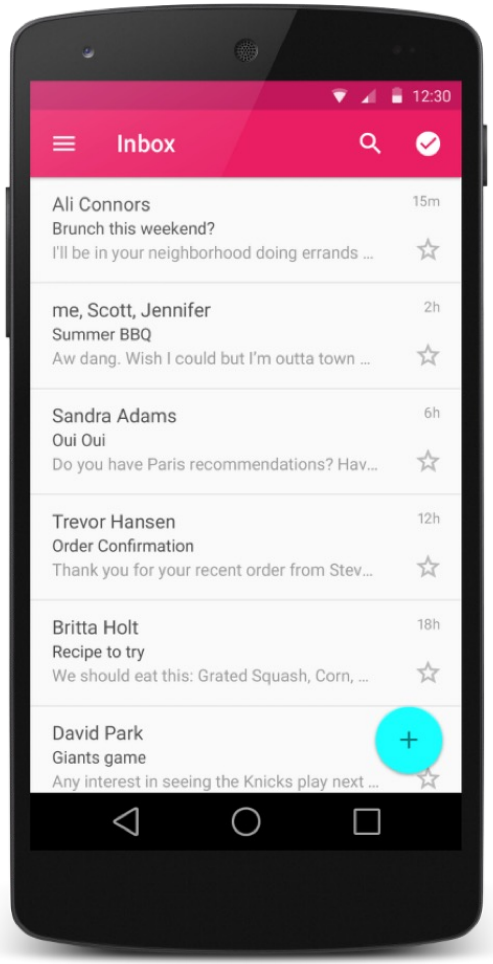
Mobile Computing - Android

Introduction

- LinearLayouts, RelativeLayouts, GridLayouts, etc. contain other Views, but for positional purposes only: they are not interactive
- *Container* classes also contain Views, but the user can interactively browse through them
- Numerous types:
  - RecyclerViews, ListViews, GridViews, ScrollViews, HorizontalScrollViews, various Flippers and Switchers
- The layout of choice if we want to display a largish amount of data in a table



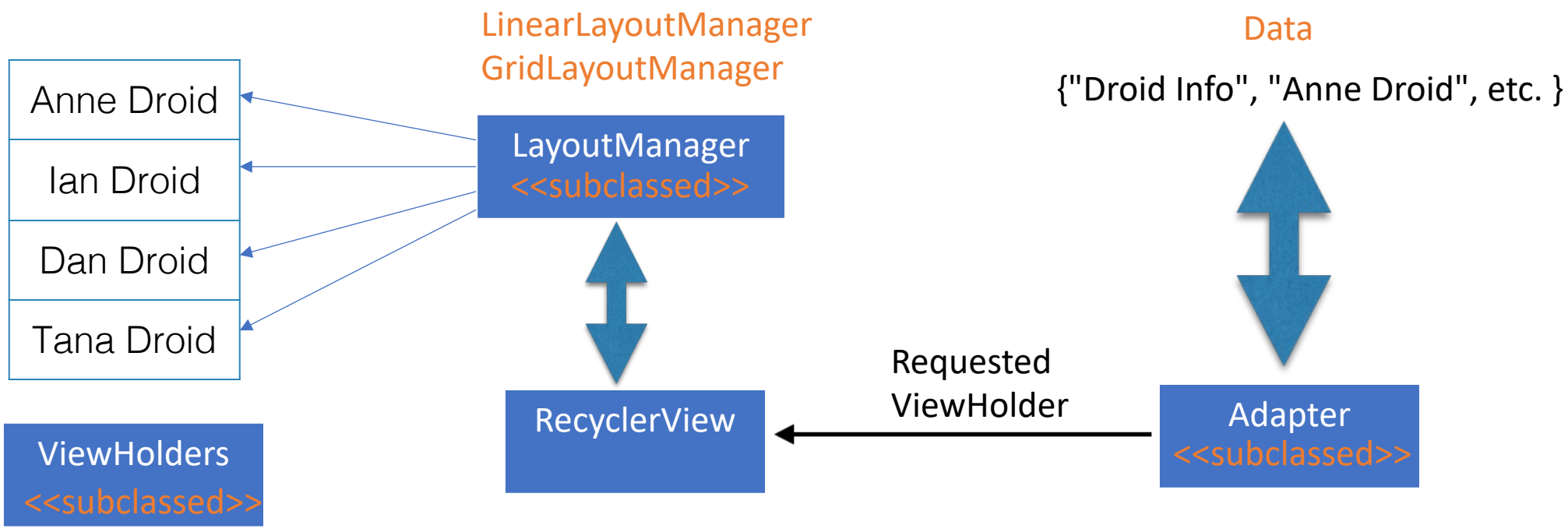
RecyclerView



- A single column table, aka a vertically scrolling, horizontally filled list of Views. This is a more flexible/advanced version of a ListView.

java.lang.Object
↳ android.view.View
↳ android.view.ViewGroup
↳ androidx.recyclerview.widget.RecyclerView

RecyclerView Illustrated



The view holder and contents are provided by the Adapter.
(The general method responsible for providing the ViewHolder is bindViewHolder() method, but the actual work is done in the subclass where it overrides onbindViewHolder()

Recycler.Adapter

- Subclass the recycler adapter to work with a data source and bind data to a ViewHolder.
- Need to override
  - `onCreateViewHolder(ViewGroup parent, int viewType)`
    - Called when the RecyclerView needs a new ViewHolder.
  - `onBindViewHolder(ViewHolder, int pos)`
    - Update the view holder to reflect the data at the given position in the data set.
  - getItemCount()
    - Return the number of items in the data source.

Recycler.LayoutManager

- Can use a predefined manager like `LinearLayoutManager` or `GridLayoutManager`.

- Can subclass `Recycler.LayoutManager` to create a customized layout.

Recycler.Viewholder

- An abstract child view that knows its position in the layout/adapter.
- The actual view is held by the instance variable `itemView`.
- Knows if the view can be recycled.
- Defined as a static inner class of our adapter. (Static inner classes are easier to create since they can be created independently of any instance of the outer class.)

Steps to Create a Basic RecyclerView

1. Add a RecyclerView to the activity layout.
2. Create a layout to use for the subviews.
3. Define a concrete adapter class inheriting from Recycler.Adapter.
4. Define a concrete ViewHolder class inside the adapter inheriting from Recycler.ViewHolder.
5. Create a data model.
6. Override the abstract methods onCreateViewHolder, onBindViewHolder, and getItemCount in the concrete adapter.
7. Create an instance of the adapter with a reference to the data.
8. Attach the adapter to the RecyclerView
9. Create an instance of a LayoutManager and attach to the RecyclerView

A RecyclerView for Planets

- We are going to demo the basic steps by creating an app that will list out planet names that are derived from an array list.

- Precondition: You have created an app from the empty app template.

1. Recycler in the Layout

- In the activity layout XML, drag and drop (add) a RecyclerView widget.
- Set its attributes
  - Give it the id planetRV
  - Lets give it the full width (match parent)
  - Reserve the bottom of the screen for later (300dp) with margin/bias to put it up at the top.

2. New Layout resource

- Create a new layout that we will use for each cell with the name ***planet_view*** that has a constraint view as its root*.
  - Match the parent in width and make the height 50dp
- Add a TextView with the id nameTV
  - Wrap for height width
  - Position to the left and centered vertically.

*We could make the root element just a text view, but we are going to want to display more than one thing in the cell.

3. Adapter Subclass

We need a specialized adapter that will connect the planets model to the RecyclerView. Create a Java class and inherit from RecyclerView.Adapter.

PlanetAdapter.java

```
public class PlanetAdapter extends
RecyclerView.Adapter<PlanetAdapter.PlanetViewHolder> {


// More to come here
}
```

4. ViewHolder Subclass

Inside the Planets Adapter class we will define how the subviews of the Recycler look with our PlanetViewHolder that inherits from RecyclerView.TextHolder.

Inside PlanetAdapter.java

```
public static class PlanetViewHolder
    extends RecyclerView.ViewHolder {
       public PlanetViewHolder(TextView v) {
          super(v);
       }
}
```

5a. Reference the Model

We know that the Adapter must have access to the model (data) that are going to be displayed. This means that some class must have the prime responsibility of creating and holding the data. We have a few choices:

1. Put the data in the MainActivity.
2. Put the data in the Adapter.
3. Put the data in some other class.

For now we will put the data in the MainActivity.

5a. Reference the Model

Inside the Planets Adapter class we will need to have reference to our model. Since we are keeping the model in the main activity, we will pass a reference to the model as an argument of the PlanetAdapter constructer.

Inside PlanetAdapter.java after the previous code.

```
private ArrayList<String> model;

public PlanetAdapter(ArrayList<String> model) {
    super();
    this.model = model;
}
```

5b. Create the Model

We decided that the model would be held by the Main Activity. The onCreate method is a good place to guarantee that things like models get initialized when the app starts since it will be the first activity created.

Inside MainActivity.

```
private ArrayList<String> data = null;

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

    // Load up the model
    data = new ArrayList<String>(); data.add("Mercury");
    data.add("Venus"); data.add("Earth"); data.add("Mars");
}
```

6a. Override OnCreateViewHolder

- This method allows the RecyclerView to ask for a new subview that it can display. Note that we don't have to put anything in the sub view, we just need to use an Inflater with the XML to get the root element viewroot and then pass that into a new ViewHolder.

Inside PlanetAdapter.

```
public PlanetsAdapter.PlanetViewHolder
onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    // Create the view
    View v = LayoutInflater.from(parent.getContext())
            .inflate(R.layout.planet_view, parent, false);
    PlanetViewHolder vh = new PlanetViewHolder(v);
    return vh;
}
```

6b. Override OnBindViewHolder

This method allows the RecyclerView to ask the adapter to use an item in the data model to set the contents of the view in the ViewHolder.

Inside PlanetAdapter.

```
public void onBindViewHolder(
    @NonNull PlanetViewHolder holder,
    int position)
    {
       TextView nameTV = holder.theView.findViewById(R.id.nameTV);
       nameTV.setText(model.get(position));
    }
```

6c. Override itemCount

Ask the model how many items it has… or potentially how many values should be displayed.

Inside PlanetAdapter.

```
@Override
public int getItemCount() {
    return model.size();
}
```

7. Create the Adapter

- Now that we have finished with the definition of the adapter, we want to create an adapter in the activity that has the Recycler view. (Main in our case, but it could be a different activity.)
- Note that we are passing in a reference to the model.

Inside MainActivity at the end of onCreate

```
// Create the Adapter
PlanetsAdapter planetServer = new PlanetsAdapter(data);
```

8. Attach the Adapter

Get a reference to the RecyclerView and then attach the adapter to it.

Inside MainActivity at the end of onCreate

```
// Attach it to the RecyclerView
RecyclerView planetRV = findViewById(R.id.planetRV);
planetRV.setAdapter(planetServer);
```

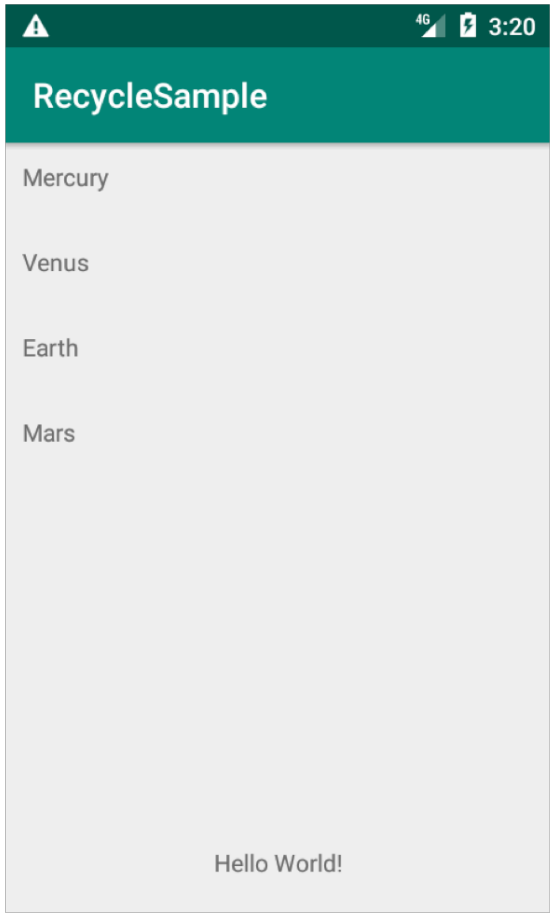
9. Attach a LayoutManager

We will use the predefined LinearLayoutManager so that items will be presented in a vertically stacked fashion.

Inside MainActivity at the end of onCreate

```
// Make and attach a layoutmanager
LinearLayoutManager myManager = new LinearLayoutManager(this);
planetRV.setLayoutManager(myManager);
```

The Result



Questions

- Distinguish between
  - an Adapter and a RecyclerView
  - a RecyclerView and a LayoutManager
  - a ViewHolder and a View
- Describe and be able to sketch the relationships between a RecyclerView, an Adapter, ViewHolders and a LayoutManager.
- To implement a RecyclerView, which classes will you subclass?
- What are the three methods that the Adapter class will need to override?
- What is the purpose of and difference between the methods onCreateViewHolder and onBindViewHolder? Who calls these methods?
- What is the purpose of the LayoutManager? What are the two predefined subclasses that are commonly used and how do they arrange the ViewHolders.