RecyclerView: Models and Interaction

Mobile Computing - Android

Improving the RecyclerView

GOALS:

- Add complexity to the data source and display more than one value in a cell.
- Make the cells clickable and retrieve the data associated with the cells position.
- Mutate the list holding the data.

Making the data more interesting

- Add extra components to the container view so we have maximum ability to style and place the information/images.

A Better Model

- We want the model to have a cohesive set of data values
- We typically want to have a single copy of the model.
- We typically want the model to be accessible globally.
- We typically want to avoid parallel lists and instead define a an inner class with instances that get added into a list.

Parallel

```
ArrayList<String> names;
ArrayList<Double> distances;
```

Preferred

```
public static class Planet{
  String name;
  Double distance;
}
ArrayList<Planet> thePlanets;
```

A Better Model

Once we have more complicated models, we also want to separate them from the rest of the code.

- Option 1: Make a class for the model that has static attributes and static methods.
- Option 2: Make a class that has a single static reference that refers to the single instance of the class. Attributes and methods are then typically not static. (*Singleton.*)

Model Class (Start)

```
public class Model {

    // Planet is a domain class, we can define it here
    // as an inner class or make it into its own class
    // The choice depends on how complicated it is.
    // We also have a choice on how to access the attributes.
    // Since this is a simple class with no constraints, I will
    // make the attributes public and will not use accessor/mutator
    // methods, but work with the attributes directly.
    public static class Planet {
        public String name;
        public Double distance;
        public Planet(String name, Double distance) {
            this.name = name;
            this.distance = distance;
        }
    }
```

Model Class (Make Singleton)

```
// Make the model class a singleton
// There will be a unique instance of the model
// The first time someone asks for the model it will create itself.
private static Model theModel = null;
public static Model getModel(){
    if (theModel == null) theModel = new Model();
    return theModel;
}

public ArrayList<Planet> thePlanets;    //If private, need getter/setter

// The constructor must be private so we can guarantee
// a unique model object

private Model(){
    thePlanets = new ArrayList<Planet>();
    loadModel();
}
```

Model Class (Load responsibility)

```
// Since this code might be complicated, lets break
// it off from the constructor.  May get values from a file or DB
    private void loadModel(){
        Planet p = null;
        p =  new Planet("Mercury", 0.39);  thePlanets.add(p);
        p  = new Planet("Venus", 0.72);    thePlanets.add(p);
        p  = new Planet("Earth", 1.0);     thePlanets.add(p);
        p  = new Planet("Mars", 1.52);     thePlanets.add(p);
        p  = new Planet("Jupiter", 5.20);  thePlanets.add(p);
        p  = new Planet("Saturn", 9.58);   thePlanets.add(p);
        p  = new Planet("Uranus", 19.20);  thePlanets.add(p);
        p  = new Planet("Neptune", 30.05); thePlanets.add(p);
    }
}
```

Revise the Cell Layout

- Our goal is that the name will be above the distance with the distance indented.
- Lets change the font appearance of the distance as well.

Add a TextView

```
<TextView
        android:id="@+id/distanceTV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="casual"
        android:text="Default distance"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.1" />
```

Using the new Model

- We no longer need to create the model in the MainActivity.
- We no longer need to pass the model into the adapter.
- We no longer need to hold onto a reference for the model.
- We do need to update the bind and count methods.

Adapter Change

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

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

Because distance is a Double and not a double, we don't use String.valueOf(distance), but do toString()

Colored parts are replaced by:

```
nameTV.setText(Model.getModel().thePlanets.get(position).name);
TextView distanceTV = holder.itemView.findViewById(R.id.distanceTV);
distanceTV.setText(Model.getModel().thePlanets.get(position).distance.toString);
```

```
return Model.getModel().thePlanets.size();
```

Adding in Click Responses

- In the past with a ListView, you would create an **onItemClickListener**, attach it to the ListView and a click would trigger the listener and pass in the position of the item the from an array so you could handle the click.
- RecyclerViews are more complicated.
  - You could create onClickListeners and associate them with the ViewHolder… This creates a lot of anonymous functions.
  - You could use a SelectionTracker, but you will need to have a unique identifier (key) for every item in your list/grid.
  - We are going to create a GestureRecognizer and intercept motion events. If we find the gesture we are looking for (single tap) we will take the appropriate action.

Adding in Click Responses

```
private GestureDetectorCompat detector = null;


// We need a gesture listener
private class RecyclerViewOnGestureListener extends GestureDetector.SimpleOnGestureListener {


    @Override
    public boolean onSingleTapConfirmed(MotionEvent e) {
        RecyclerView planetRV = findViewById(R.id.planetRV);
        View view = planetRV.findChildViewUnder(e.getX(), e.getY());
        if (view != null) {
            RecyclerView.ViewHolder holder = planetRV.getChildViewHolder(view);
            if (holder instanceof PlanetsAdapter.PlanetViewHolder) {
                int position = holder.getAdapterPosition();


                // handle single tap
                Log.d("click", "clicked on item "+ position);
                return true;  // Use up the tap gesture
            }
        }


        // we didn't handle the gesture so pass it on
       return false;
    }
}
```

Handy reference

Code for handling the tap

Put it in Main

Adding in Click Responses

```
// Make a Listener for taps
detector = new GestureDetectorCompat(this,
                    new RecyclerViewOnGestureListener());


// add the listener to the recycler
planetRecycler.addOnItemTouchListener(new RecyclerView.SimpleOnItemTouchListener(){
    @Override
    public boolean onInterceptTouchEvent(RecyclerView rv, MotionEvent e) {
        return detector.onTouchEvent(e);


    }
});
```

We observe motion events looking for the ones we want.

The detector consumes tap events, but others like scrolling pass through.

Put it at the end of Main onCreate

A More Interesting Handler

Get the name from the model and use it to set a Textview.

```
// handle single tap
Log.d("click", "clicked on item "+ position);

TextView outputTV = findViewById(R.id.outputTV);
outputTV.setText("Clicked on " +
     Model.getModel().thePlanets.get(position).name);

return true;  // Use up the tap gesture
```

Can we change the Model

Suppose we want a tap to make that item go away…
What happens if we change the model?

```
// handle single tap
    Log.d("click", "clicked on item "+ position);
    TextView outputTV = findViewById(R.id.outputTV);
    outputTV.setText("Clicked on " +
            Model.getModel().thePlanets.get(position).name);

    // Remove the selected data from the model
    Model.getModel().thePlanets.remove(position);

    return true;  // Use up the tap gesture
```

Can we change the Model

Suppose we want a tap to make that item go away…
What happens if we change the model?

```
java.lang.IndexOutOfBoundsException:
Inconsistency detected.
Invalid item position 7(offset:7).state:8
androidx.recyclerview.widget.RecyclerView
```

Dynamically changing the Model

If an event causes the model to change, the Adapter needs to notify the RecyclerView so that it can change the view inside any affected ViewHolders.

- notifyDataSetChanged() - Redo it all.
- notifyItemChanged(int position) - Contents of a particular value has changed.
- notifyItemInserted(int position) - we have a new value
- notifyItemMoved(int fromPosition, int toPosition) - position shuffle
- notifyItemRemoved(int position) - lost a value
- There are other notifications that work with a range of item positions.

Change with Notification

```
// handle single tap
    Log.d("click", "clicked on item "+ position);
    TextView outputTV = findViewById(R.id.outputTV);
    outputTV.setText("Clicked on " +
            Model.getModel().thePlanets.get(position).name);

    // Remove the selected data from the model
    Model.getModel().thePlanets.remove(position);
    RecyclerView planetRV = findViewById(R.id.planetRV);
    PlanetAdapter planetServer =
              (PlanetAdapter)planetRV.getAdapter();
    planetServer.notifyItemRemoved(position);

    return true;  // Use up the tap gesture
```

Adding an item to the Model

We are going to use a Button to add a new planet. In real life we would get the values used in the construction from an EditText

```
Button addBTN = findViewById(R.id.addBTN);

addBTN.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Add a new value into the Model at the end
        Model.getModel().thePlanets.add(new Model.Planet("Planet X", 55.0));
        planetServer.notifyItemInserted(Model.getModel().thePlanets.size()-1);
    }
});
```

Create an anonymous Listener.

Planet is static, so we can create it easily.

Put the code at the end of Main onCreate

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.