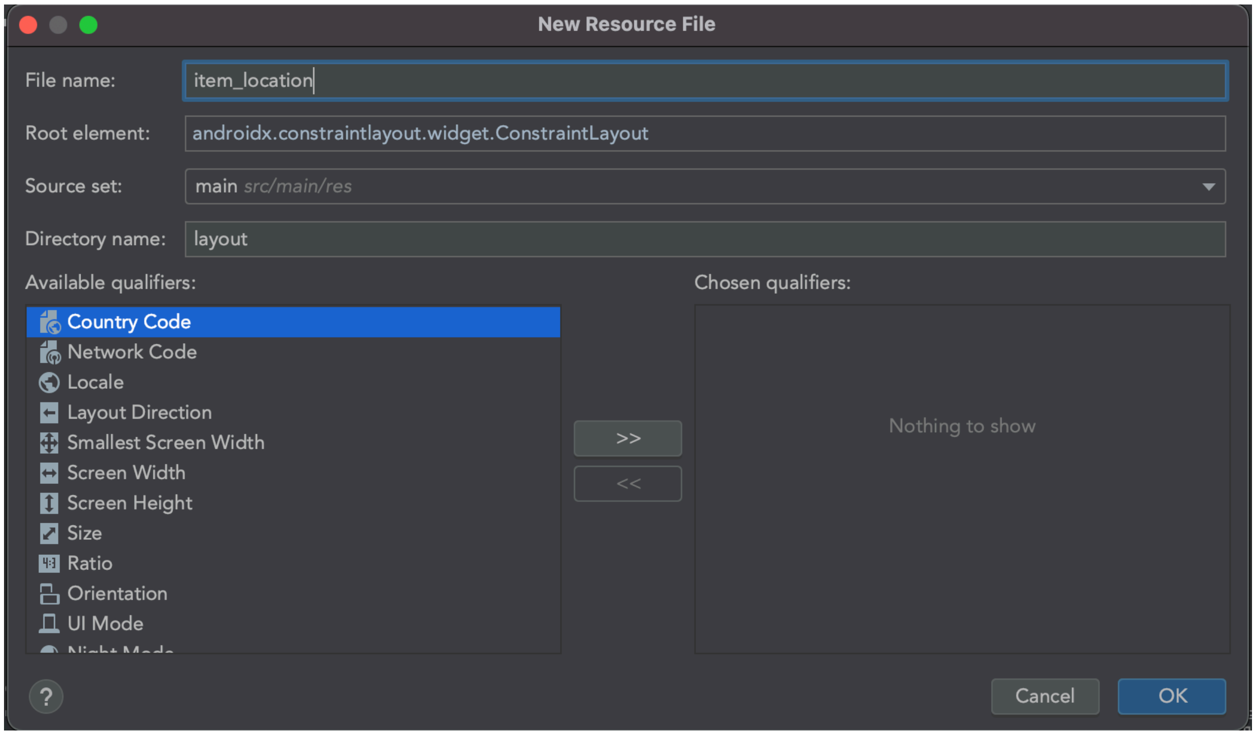
Building a list of items using RecyclerView

Setup

Create a new project in Android Studio (API 26+, Kotlin)

Create a layout file for a single item in the list

This view will be populated with data from a list in the adapter. Use this to represent what a generic view will contain in your list.
Right click on the layout directory under the res directory and select “New -> Layout Resource File”
Name the file “item_location” and click Ok.



New Resource File

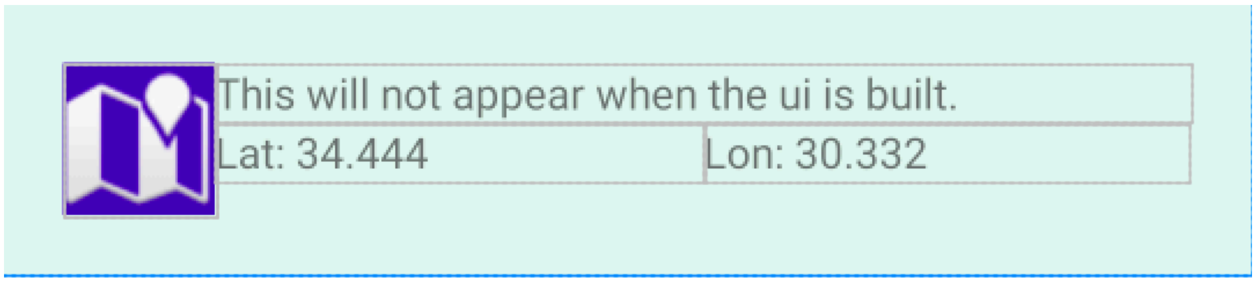
File name: item_location

Root element: androidx.constraintlayout.widget.ConstraintLayout

Source set: main *src/main/res*

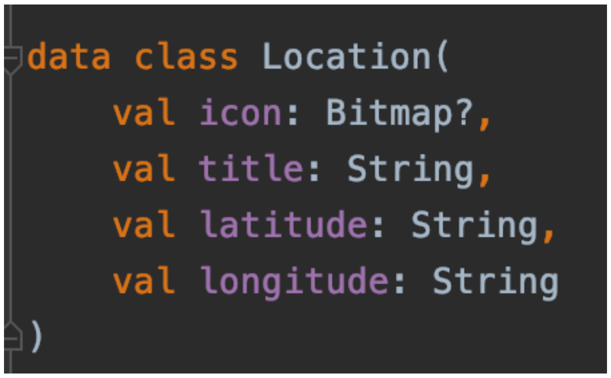
Directory name: layout

Add a couple widgets to this layout file such as TextView's and an ImageView (see example)
Note: After adding a couple views and arranging them, you should set the height of this view to "wrap_content" so each item will only take up as much space as it needs, and not a whole screen.



Create an data model for the item view

You will need to create a class that will store the properties that each item will need in the UI. (e.g., On instagram, a data model for a post might include properties such as the user, an image, how many likes a post has, a comment, etc)
Create a data class that store properties for the item.(Note: we use a data class because they provide the following for all properties declared in the primary constructor; equals()/hashcode() pair, toString(), copy())



```kotlin
data class Location(
    val icon: Bitmap?,
    val title: String,
    val latitude: String,
    val longitude: String
)
```

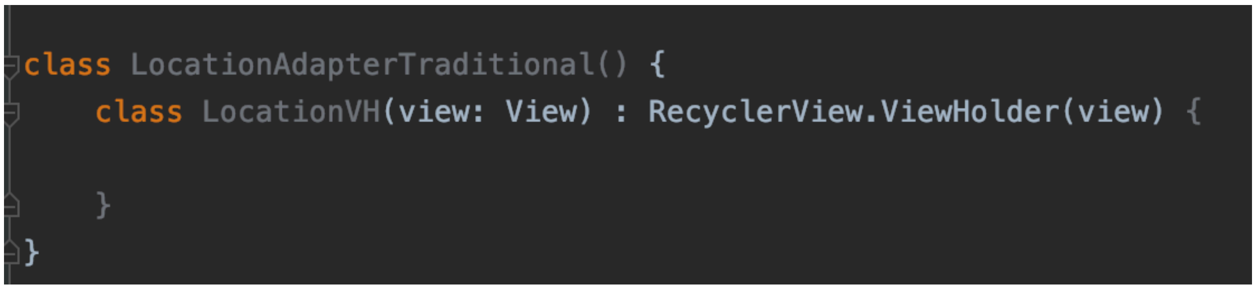
Create the recycler view adapter and view holder

To populate a recycler view with items, there are a few different methods you can use.
The two main classes you will deal with are generically known as the “Adapter” class, which is responsible for adapting one form of data into another (in our case, a list of locations into a View filled with some items), and the “ViewHolder” class, which is responsible for inflating the views, holding a reference to each item within that view, and binding data to that view.

You’ll notice in the code demo there are a few different implementations that are similar but unique. Adapter classes that have the word “Traditional” in them use the old school way to reference views (via findViewById()) which is fine, but can provide errors if the items are referenced incorrectly.

Method 1: Extend RecyclerView.Adapter, reference views the traditional way.

Create a class named “LocationAdapterTraditional”, then add a nested class named “LocationVH” which extends the RecyclerView.ViewHolder class. The LocationVH class is used to ‘hold’ a reference to the ‘views’ within each item layout file. It will be responsible for inflating the views and binding data to it.

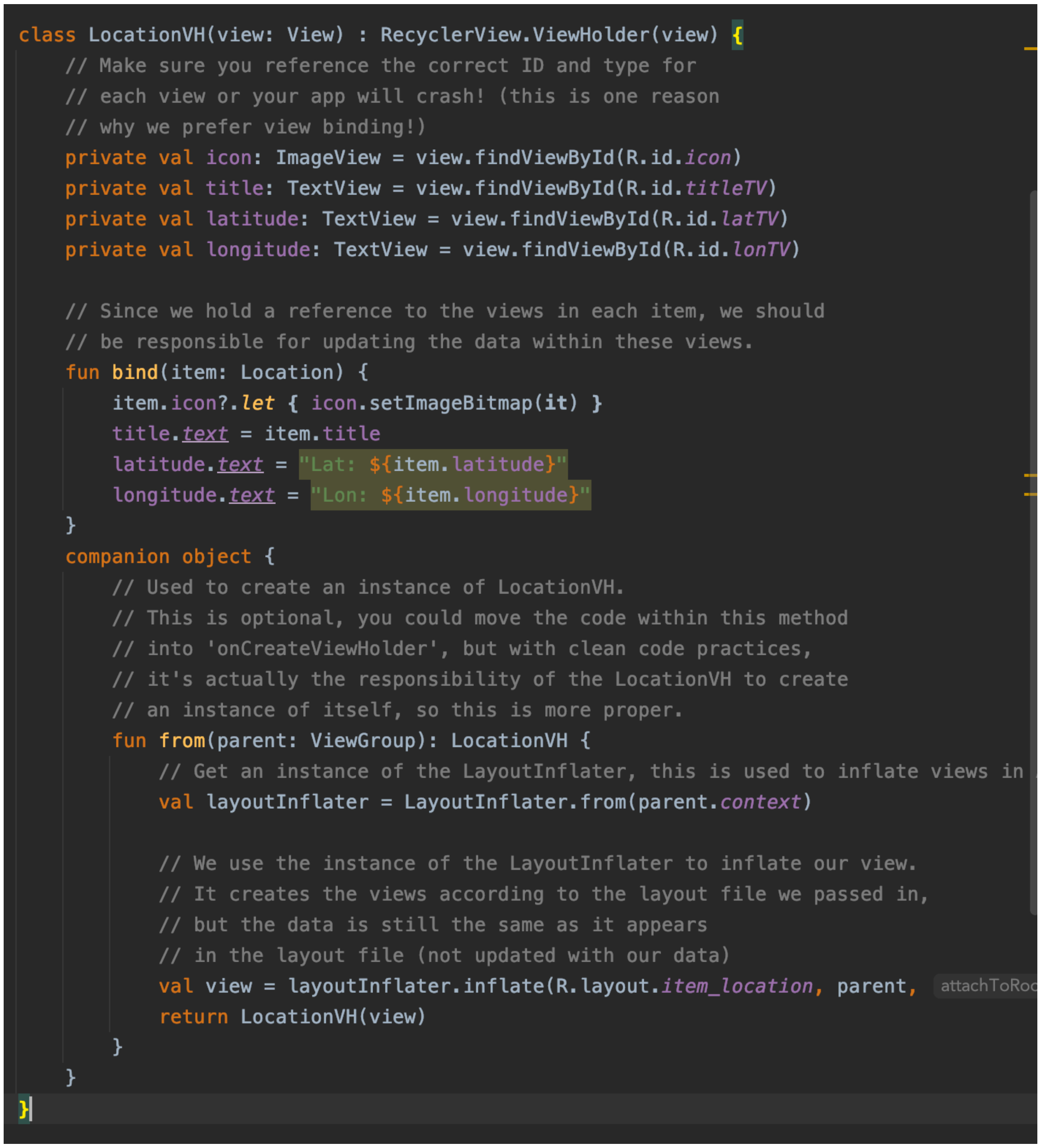


```
class LocationAdapterTraditional() {
    class LocationVH(view: View) : RecyclerView.ViewHolder(view) {

    }
}
```

Create references to each of the views for the item layout file you created. The traditional way is to use the 'findViewById()' method of the view that was passed in.

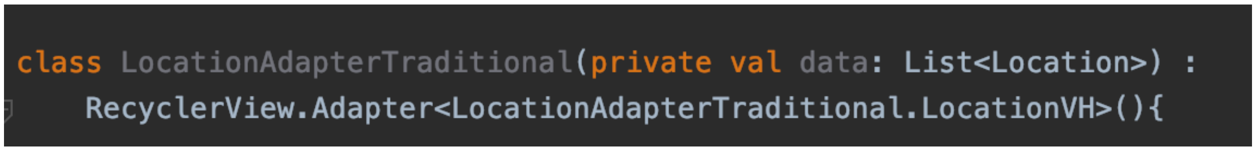
Add a function called "bind" that will be responsible for binding the data from a single item into our view. You will bind the data from each "item" into each item in the view.

Finally, add a static (companion object for Kotlin) function called "from" that is responsible for creating an instance of the ViewHolder class. (See code).



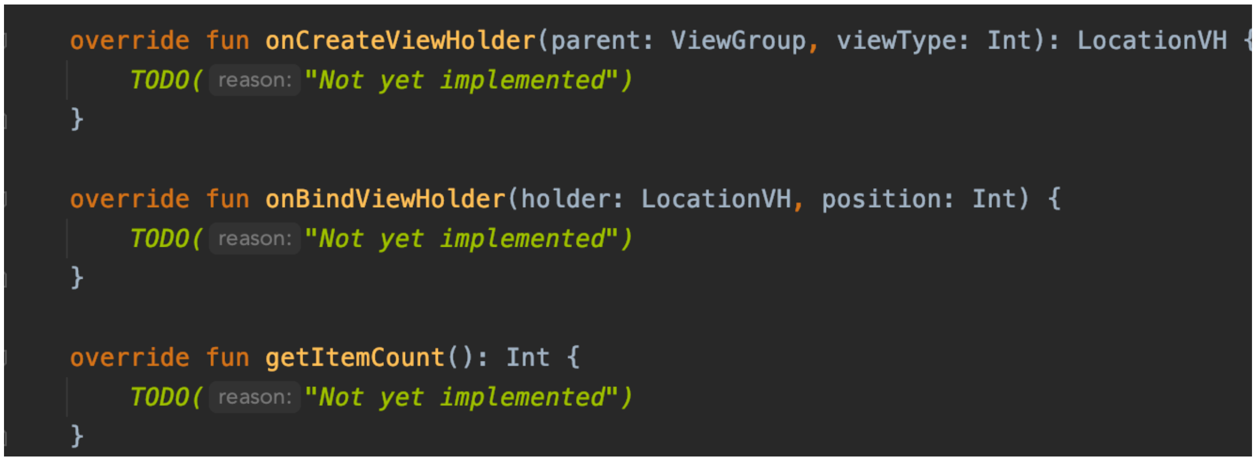
```kotlin
class LocationVH(view: View) : RecyclerView.ViewHolder(view) {
    // Make sure you reference the correct ID and type for
    // each view or your app will crash! (this is one reason
    // why we prefer view binding!)
    private val icon: ImageView = view.findViewById(R.id.icon)
    private val title: TextView = view.findViewById(R.id.titleTV)
    private val latitude: TextView = view.findViewById(R.id.latTV)
    private val longitude: TextView = view.findViewById(R.id.lonTV)

    // Since we hold a reference to the views in each item, we should
    // be responsible for updating the data within these views.
    fun bind(item: Location) {
        item.icon?.let { icon.setImageBitmap(it) }
        title.text = item.title
        latitude.text = "Lat: ${item.latitude}"
        longitude.text = "Lon: ${item.longitude}"
    }
    companion object {
        // Used to create an instance of LocationVH.
        // This is optional, you could move the code within this method
        // into 'onCreateViewHolder', but with clean code practices,
        // it's actually the responsibility of the LocationVH to create
        // an instance of itself, so this is more proper.
        fun from(parent: ViewGroup): LocationVH {
            // Get an instance of the LayoutInflater, this is used to inflate views in
            val layoutInflater = LayoutInflater.from(parent.context)

            // We use the instance of the LayoutInflater to inflate our view.
            // It creates the views according to the layout file we passed in,
            // but the data is still the same as it appears
            // in the layout file (not updated with our data)
            val view = layoutInflater.inflate(R.layout.item_location, parent, attachToRoo
            return LocationVH(view)
        }
    }
}
```

Next, make your LocationAdapterTraditional class extend the RecyclerView.Adapter class and add a parameter for a list of the items your adapter will create.



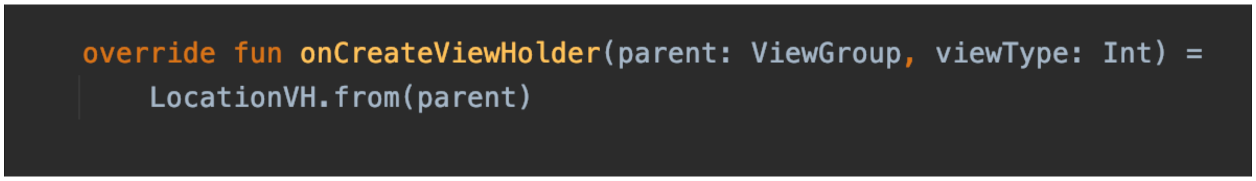
```
class LocationAdapterTraditional(private val data: List<Location>) :
    RecyclerView.Adapter<LocationAdapterTraditional.LocationVH>(){
```

You will need to implement the onCreateViewHolder(), onBindViewHolder(), and getItemCount() methods.



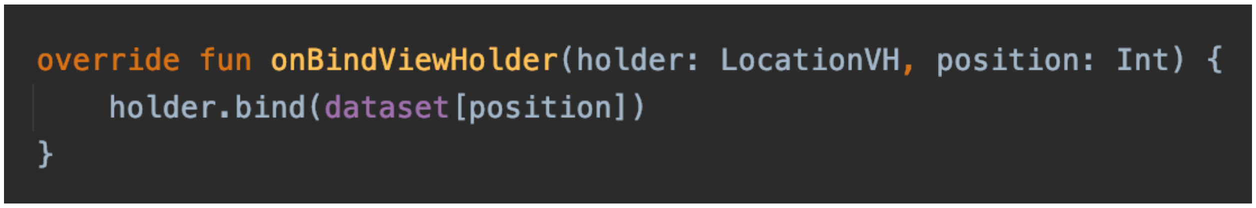
```
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): LocationVH {
        TODO( reason: "Not yet implemented")
    }

    override fun onBindViewHolder(holder: LocationVH, position: Int) {
        TODO( reason: "Not yet implemented")
    }

    override fun getItemCount(): Int {
        TODO( reason: "Not yet implemented")
    }
```

The ‘onCreateViewHolder’ method is responsible for creating an instance of the ViewHolder. We can use the function “from” we created within the companion object of our LocationVH class to simply create it from a parent ViewGroup.



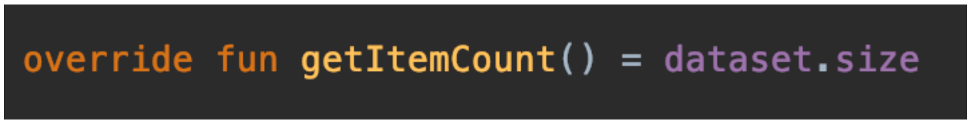
```
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int) =
        LocationVH.from(parent)
```

The ‘onBindViewHolder’ method is used to bind the data from our dataset into each inflated view. We can use the ‘bind’ method we created within the LocationVH to bind data to our views.



```
override fun onBindViewHolder(holder: LocationVH, position: Int) {
    holder.bind(dataset[position])
}
```

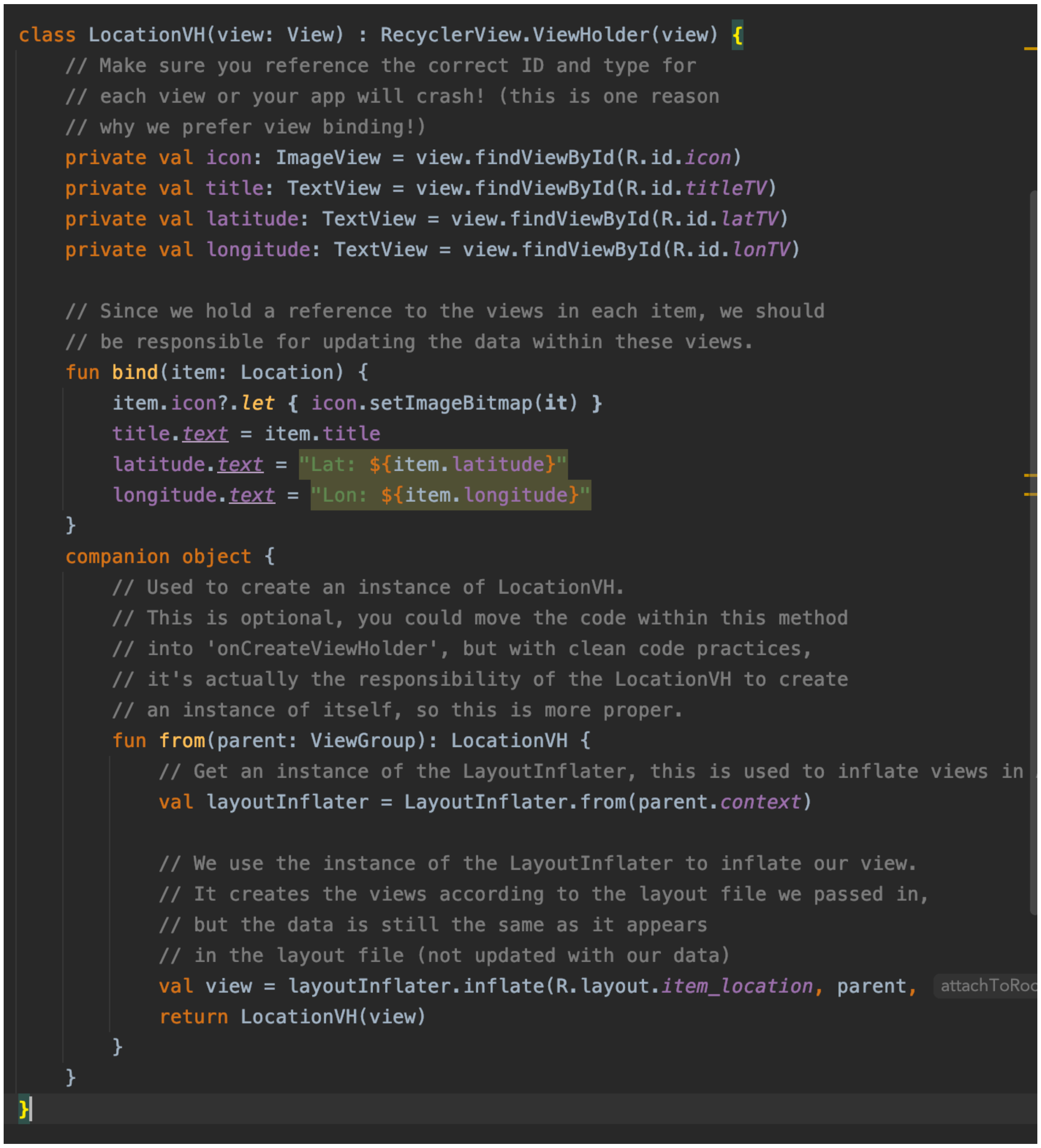
Finally, the ‘getItemCount’ method is used to get the number of items that will be displayed within this list. Simply return the number of items from our dataset.



```kotlin
override fun getItemCount() = dataset.size
```

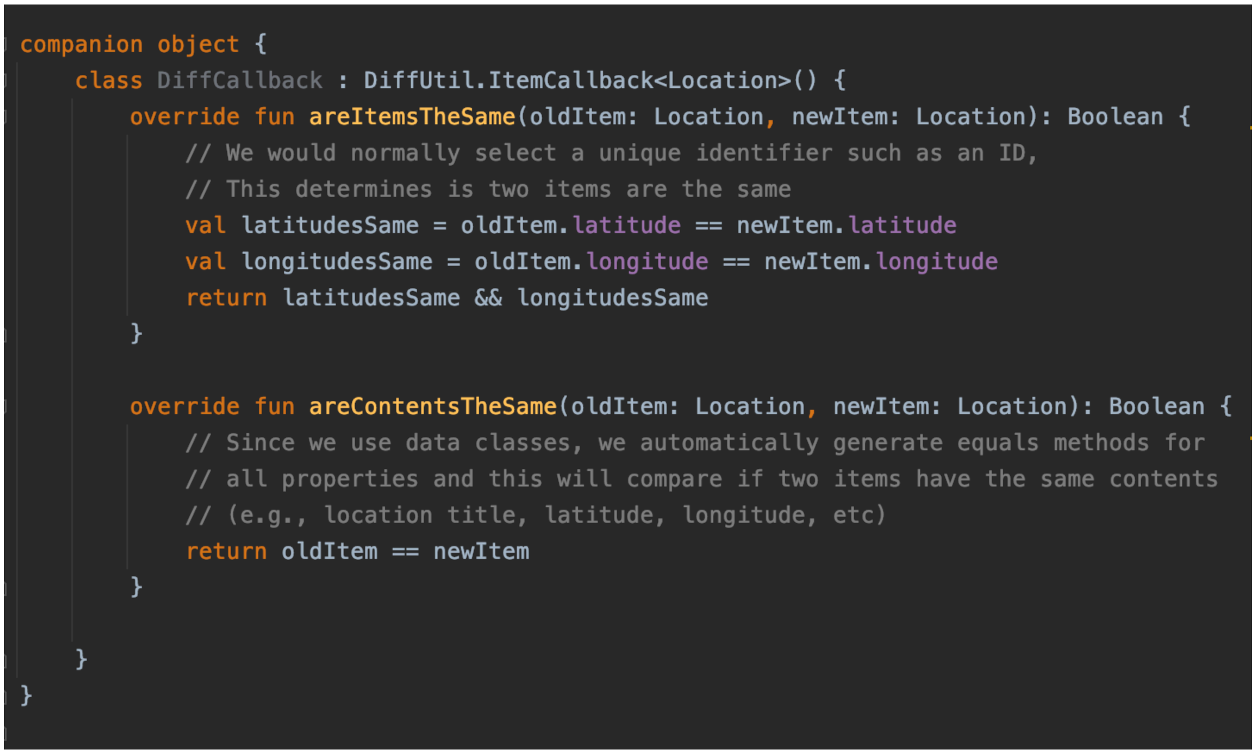
Method 2: Extend ListAdapter, reference views the traditional way.

Follow the same steps from method 1 to create a LocationVH, create references to your views, then add a bind and from method (see method 1).



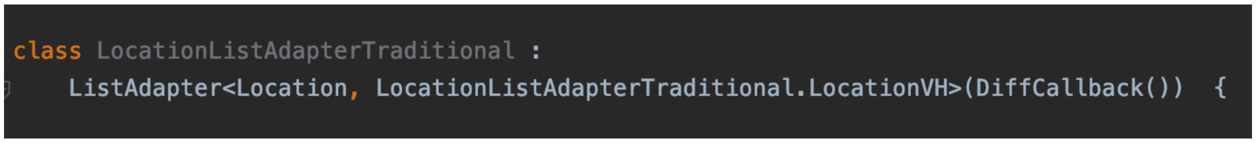
```kotlin
class LocationVH(view: View) : RecyclerView.ViewHolder(view) {
    // Make sure you reference the correct ID and type for
    // each view or your app will crash! (this is one reason
    // why we prefer view binding!)
    private val icon: ImageView = view.findViewById(R.id.icon)
    private val title: TextView = view.findViewById(R.id.titleTV)
    private val latitude: TextView = view.findViewById(R.id.latTV)
    private val longitude: TextView = view.findViewById(R.id.lonTV)

    // Since we hold a reference to the views in each item, we should
    // be responsible for updating the data within these views.
    fun bind(item: Location) {
        item.icon?.let { icon.setImageBitmap(it) }
        title.text = item.title
        latitude.text = "Lat: ${item.latitude}"
        longitude.text = "Lon: ${item.longitude}"
    }
    companion object {
        // Used to create an instance of LocationVH.
        // This is optional, you could move the code within this method
        // into 'onCreateViewHolder', but with clean code practices,
        // it's actually the responsibility of the LocationVH to create
        // an instance of itself, so this is more proper.
        fun from(parent: ViewGroup): LocationVH {
            // Get an instance of the LayoutInflater, this is used to inflate views in
            val layoutInflater = LayoutInflater.from(parent.context)

            // We use the instance of the LayoutInflater to inflate our view.
            // It creates the views according to the layout file we passed in,
            // but the data is still the same as it appears
            // in the layout file (not updated with our data)
            val view = layoutInflater.inflate(R.layout.item_location, parent, attachToRoo
            return LocationVH(view)
        }
    }
}
```

Create a companion object in LocationListAdapterTraditional called “DiffCallback” that extends “DiffUtil.ItemCallback<Location>”. This class is used to compare if two items have changed in the dataset. For example, if we had 3 items in our dataset, and we removed the middle item, the first and last items are the same so we don’t have to recreate them from scratch, just move the last item to the second. This provides a very efficient way to add/remove items without having to recreate every view.



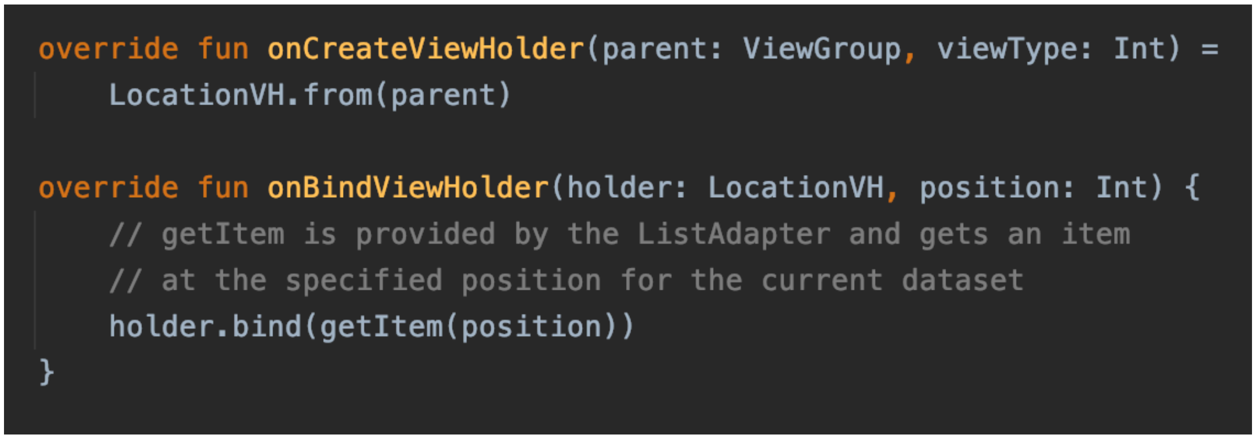
```kotlin
companion object {
    class DiffCallback : DiffUtil.ItemCallback<Location>() {
        override fun areItemsTheSame(oldItem: Location, newItem: Location): Boolean {
            // We would normally select a unique identifier such as an ID,
            // This determines is two items are the same
            val latitudesSame = oldItem.latitude == newItem.latitude
            val longitudesSame = oldItem.longitude == newItem.longitude
            return latitudesSame && longitudesSame
        }

        override fun areContentsTheSame(oldItem: Location, newItem: Location): Boolean {
            // Since we use data classes, we automatically generate equals methods for
            // all properties and this will compare if two items have the same contents
            // (e.g., location title, latitude, longitude, etc)
            return oldItem == newItem
        }

    }
}
```

Next, have your LocationListAdapterTraditional class extend the ListAdapter class. This class is an extension to the traditional RecyclerView.Adapter class. You will need to pass in the Type of data the RecyclerView will display (Location for us) and the ViewHolder that will be used as Type Parameters, as well as the DiffCallback class we made as part of the constructor.



```kotlin
class LocationListAdapterTraditional :
    ListAdapter<Location, LocationListAdapterTraditional.LocationVH>(DiffCallback())  {
```

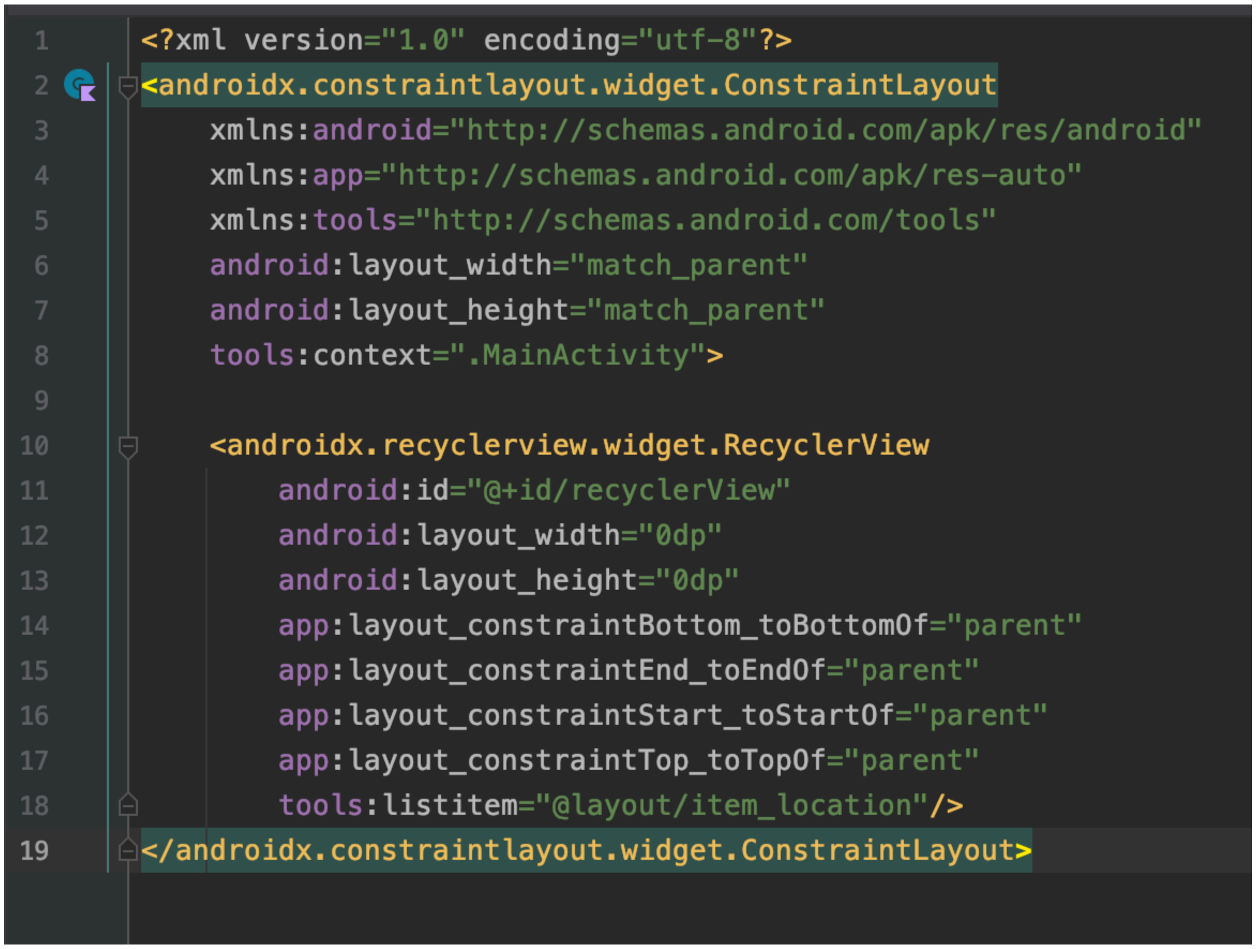
Implement the ‘onCreateViewHolder()’ and ‘onBindViewHolder’ similarly to how you did in Method 1 (notice we don’t need to set the size for our dataset)



```kotlin
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int) =
    LocationVH.from(parent)

override fun onBindViewHolder(holder: LocationVH, position: Int) {
    // getItem is provided by the ListAdapter and gets an item
    // at the specified position for the current dataset
    holder.bind(getItem(position))
}
```

Add a RecyclerView to the main activity

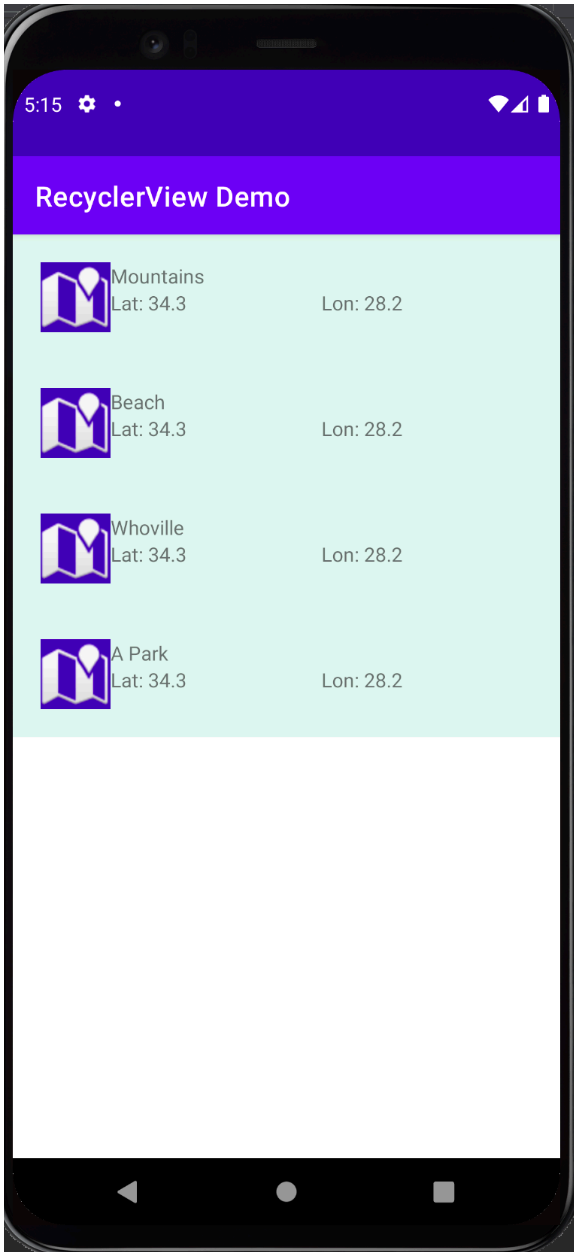
Add a RecyclerView widget to the main activity, make sure it's constrained to the edges of the screen. (Tip: to preview what your items will look like, specify the item layout in the 'tools:listitem' property)



```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    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">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        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"
        tools:listitem="@layout/item_location"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Initialize the recycler view in the main activity

Create a reference to your RecyclerView (via findViewById() or viewBinding), initialize your Adapter (see code) then initialize your RecyclerView. (See code)



Mountains
Lat: 34.3 Lon: 28.2

Beach
Lat: 34.3 Lon: 28.2

Whoville
Lat: 34.3 Lon: 28.2

A Park
Lat: 34.3 Lon: 28.2