MODUL 6
Adding and Interacting with RecyclerView

THEME DESCRIPTION

In this module, students will learn how to add lists and grids of items to their apps and effectively leverage the recycling power of RecyclerView. Students will also learn how to handle user interaction with the item views on the screen and support different item view types.

WEEKLY LEARNING OUTCOME (SUB-LEARNING OUTCOME)

Students will have the skills required to present the users with interactive lists of rich items.

TOOLS/SOFTWARE USED

- Android Studio

CONCEPTS

Recycler View

RecyclerView orchestrates the **creation**, **population**, and **reuse** of views representing lists of items. To use RecyclerView, you need to familiarize yourself with two of its dependencies – the **adapter** (and through it, the **view holder**) and the **layout manager**. These dependencies provide our RecyclerView with the content to show, as well as tell it how to present that content and lay it out on the screen.

The **adapter** provides RecyclerView with child views (nested Android views within RecyclerView used to represent individual data items) to draw on the screen, binds those views to data (via **ViewHolder** instances), and reports user interaction with those views.

The **layout manager** tells RecyclerView how to lay its children out. We are provided with three layout types by default – linear, grid, and staggered grid – managed by LinearLayoutManager, GridLayoutManager, and StaggeredGridLayoutManager respectively.

PRACTICAL STEPS

Part 1 - Adding a RecyclerView and Populating it

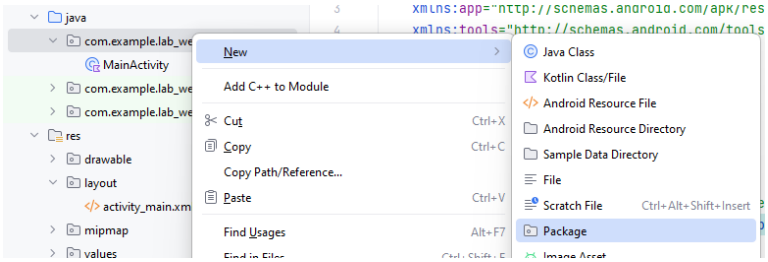
1. Open Android Studio and click **New Project.**

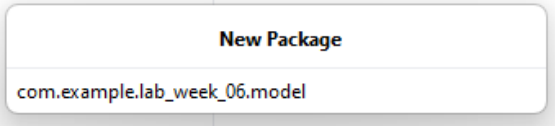
2. Choose the **Empty Views Activity** to start with.
3. Name your project **“LAB_WEEK_06”**.
4. Set the minimum SDK to **“API 24: Android 7.0 (Nougat)”**.
5. Click **Finish**, and let your android application build itself.
6. In this part, we will be focusing on how we can **create a RecyclerView** in Android. First, let’s create the layout for the **Recycler View.** Update your **activity_main.xml** to the code below.

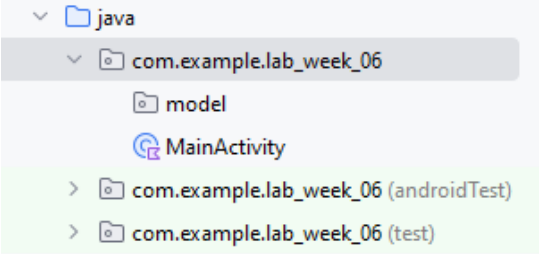
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recycler_view"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

7. If you **Run** your app now, it’ll only show a **Blank Page**. This is because the **RecyclerView** is not yet populated with the necessary data. How do we populate it? We use something that’s called an **Adapter**.

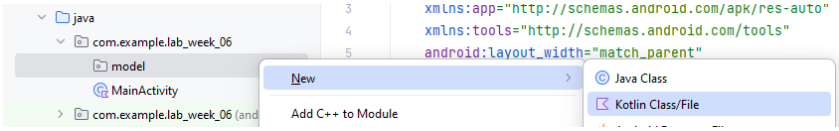
8. Now let's make the required **Adapter**. First, we need to create a **Model** for our **Data**. To keep things tidy, let's put the **Model** in a separate **Package**. Create a new **package** called **"com.example.lab_week_06.model"**.

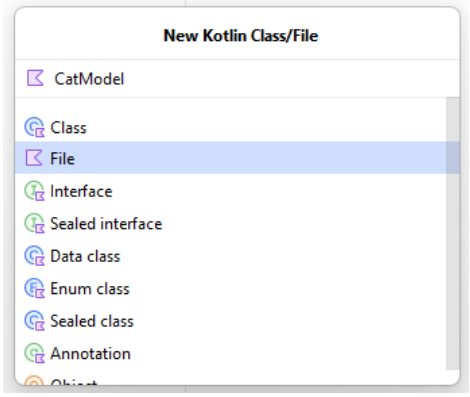






9. Create a new **Kotlin File/Class** inside the new **Package** and call it **"CatModel.kt"**. For the type, you can set it to **File**.

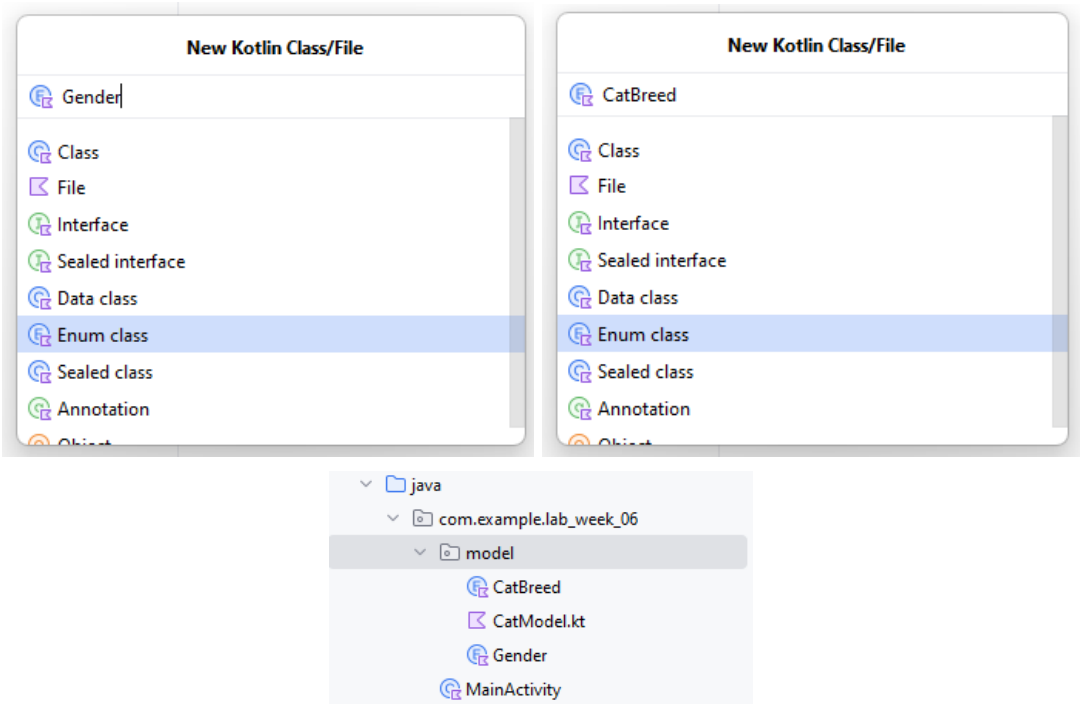




10. Add the code below to the newly created **CatModel.kt**.

```
data class CatModel(
    val gender: Gender,
    val breed: CatBreed,
    val name: String,
    val biography: String,
    val imageUrl: String
)
```

11. You may notice that **Gender** and **CatBreed** are not defined yet. Let's create 2 new Kotlin files called **Gender** and **CatBreed** inside the **Model Package**. For both files, set the type to **Enum Class**.



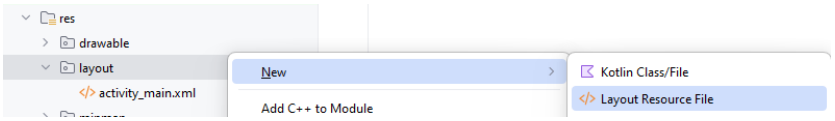
12. Add the code below to your newly created **Gender.kt**.

```
enum class Gender {
    Female, Male, Unknown
}
```

13. Add the code below to your newly created **CatBreed.kt.**

```
enum class CatBreed {
    AmericanCurl, BalineseJavanese, ExoticShorthair
}
```

14. We've created our **Data Model**. Now let's create the **Layout** for each **Item** that will be displayed in the **RecyclerView**. Create a new **Layout Resource File** and call it **"item_list.xml"**.



15. Update your newly created **item_list.xml** to the code below. Your previously created **Data Model** will all be displayed in this **Layout**.

```
<?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="wrap_content"
    android:padding="10dp">
    <ImageView
        android:id="@+id/cat_photo"
        android:layout_width="60dp"
        android:layout_height="60dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:background="@color/material_dynamic_neutral50" />
    <TextView
        android:id="@+id/cat_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:textStyle="bold"
        app:layout_constraintStart_toEndOf="@+id/cat_photo"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="Cat Name" />
    <TextView
        android:id="@+id/cat_breed"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/black"
        app:layout_constraintStart_toStartOf="@+id/cat_name"
```

```
        app:layout_constraintTop_toBottomOf="@+id/cat_name"
        tools:text="Cat Breed" />
    <TextView
        android:id="@+id/cat_biography"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="@+id/cat_breed"
        app:layout_constraintTop_toBottomOf="@+id/cat_breed"
        tools:text="Cat Biography" />

    <TextView
        android:id="@+id/cat_gender"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:text="&#9794;" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

16. We've created the **Layout** for our **Data**, now let's start **Binding** our **Data** to the **Item Views**. For this, we can use the usual **FindViewById** approach. But remember that we have an **ImageView** inside our **Layout**, therefore we need **Glide** to load the required images. First, import the necessary **Glide Dependency** to your **build.gradle.kts (Module :app)** and don't forget to **Gradle Sync**.

```
implementation("com.github.bumptech.glide:glide:4.14.2")
```

17. Next, just like in **Week 5,** create an **Interface** in your **com.example.lab_week_06** package and call it **"ImageLoader.java"**. Update it to the code below.

```
interface ImageLoader {
    void loadImage(imageUrl: String, imageView: ImageView)
}
```

18. Next, still in the same package, create a new **Kotlin File** called **"GlideImageLoader.kt"** and update it to the code below.

```
class GlideImageLoader(private val context: Context) : ImageLoader{
    override fun loadImage(imageUrl: String, imageView: ImageView) {
        Glide.with(context)
```

```
            .load(imageUrl)
            .centerCrop()
            .into(imageView)
    }
}
```

19. Our **Glide** is ready, now let's make the **Binding** function for our **Data** and **Item Views**. For this, we can use what's called a **ViewHolder**. Create a new **Kotlin File** called **"CatViewHolder.kt"**. Update it to the code below.

```
private val FEMALE_SYMBOL = "\u2640"
private val MALE_SYMBOL = "\u2642"
private const val UNKNOWN_SYMBOL = "?"

class CatViewHolder(containerView: View, private val imageLoader:
ImageLoader) : RecyclerView.ViewHolder(containerView) {
    //containerView is the container layout of each item list
    //Here findViewById is used to get the reference of each views inside
the container
    private val catBiographyView: TextView by lazy {
        containerView.findViewById(R.id.cat_biography) }
    private val catBreedView: TextView by lazy {
        containerView.findViewById(R.id.cat_breed) }
    private val catGenderView: TextView by lazy {
        containerView.findViewById(R.id.cat_gender) }
    private val catNameView: TextView by lazy {
        containerView.findViewById(R.id.cat_name) }
    private val catPhotoView: ImageView by lazy {
        containerView.findViewById(R.id.cat_photo) }

    //This function is called in the adapter to provide the binding function
    fun bindData(cat: CatModel) {
        imageLoader.loadImage(cat.imageUrl, catPhotoView)
        catNameView.text = cat.name
        catBreedView.text = when (cat.breed) {
            CatBreed.AmericanCurl -> "American Curl"
            CatBreed.BalineseJavanese -> "Balinese-Javanese"
            CatBreed.ExoticShorthair -> "Exotic Shorthair"
            else -> "Unknown"
        }
        catBiographyView.text = cat.biography
        catGenderView.text = when (cat.gender) {
```

```
            Gender.Female -> FEMALE_SYMBOL
            Gender.Male -> MALE_SYMBOL
            else -> UNKNOWN_SYMBOL
        }
    }
}
```

20. We've created the **Binding Function (View Holder)**, next we need to populate and provide all the necessary data for the **ViewHolder** using an **Adapter**. **Recycler View Adapter** works similarly to **View Pager Adapter** from the previous **Week 5** Module. Create a new **Kotlin File** called **"CatAdapter.kt**" and update it to the code below.

```
class CatAdapter(private val layoutInflater: LayoutInflater, private val
imageLoader: ImageLoader) : RecyclerView.Adapter<CatViewHolder>() {
    //Mutable list for storing all the list data
    private val cats = mutableListOf<CatModel>()

    //A function to set the mutable list
    fun setData(newCats: List<CatModel>) {
        cats.clear()
        cats.addAll(newCats)

        //This is used to tell the adapter that there's a data change in the
mutable list
        notifyDataSetChanged()
    }

    //A view holder is used to bind the data to the layout views
    //onCreateViewHolder is instantiating the view holder it self
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
CatViewHolder {
        val view = layoutInflater.inflate(R.layout.item_list, parent, false)
        return CatViewHolder(view, imageLoader)
    }

    //This is used to get the amount of data/item in the list
    override fun getItemCount() = cats.size

    //This is used to bind each data to each layout views
    override fun onBindViewHolder(holder: CatViewHolder, position: Int) {
        //The holder parameter stores our previously created ViewHolder
        //The holder.bindData function is declared in the CatViewHolder
```

```
        holder.bindData(cats[position])
    }
}
```

21. Lastly, let's update our **MainActivity.kt** to the code below.

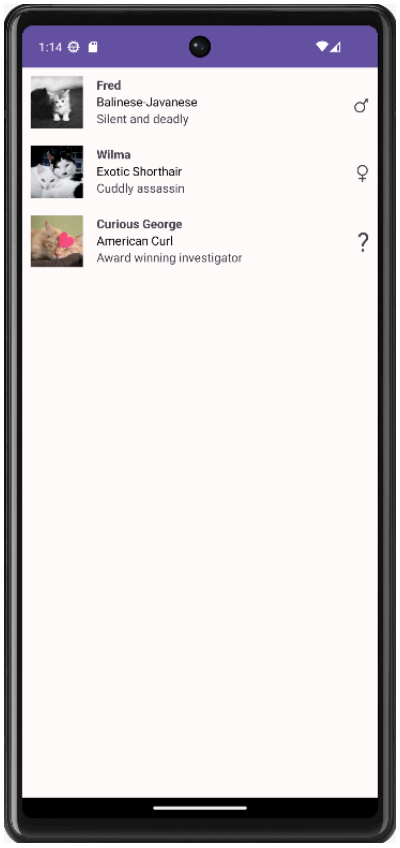
```
class MainActivity : AppCompatActivity() {
    private val recyclerView: RecyclerView by lazy {
        findViewById(R.id.recycler_view)
    }
    private val catAdapter by lazy {
        //Glide is used here to load the images
        CatAdapter(layoutInflater, GlideImageLoader(this))
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //Setup the adapter for the recycler view
        recyclerView.adapter = catAdapter

        //Setup the layout manager for the recycler view
        //A layout manager is used to set the structure of the item views
        //For this tutorial, we're using the vertical linear structure
        recyclerView.layoutManager = LinearLayoutManager(this,
LinearLayoutManager.VERTICAL, false)

        //Add data to the model list in the adapter
        catAdapter.setData(
            listOf(
                CatModel(
                    Gender.Male,
                    CatBreed.BalineseJavanese,
                    "Fred",
                    "Silent and deadly",
                    "https://cdn2.thecatapi.com/images/7dj.jpg"
                ),
                CatModel(
                    Gender.Female,
                    CatBreed.ExoticShorthair,
                    "Wilma",
                    "Cuddly assassin",
                    "https://cdn2.thecatapi.com/images/egv.jpg"
                ),
                CatModel(
```

```
                    Gender.Unknown,
                    CatBreed.AmericanCurl,
                    "Curious George",
                    "Award winning investigator",
                    "https://cdn2.thecatapi.com/images/bar.jpg"
                )
            )
        )
    }
}
```

22. You may notice, we're using an image from the internet, therefore we need to add the **INTERNET** permission into the **AndroidManifest.xml** file. Add the code below above the **Application Tag**.

```
<uses-permission android:name="android.permission.INTERNET" />
```

23. **Run** your app, and your **RecyclerView** should be working as intended.

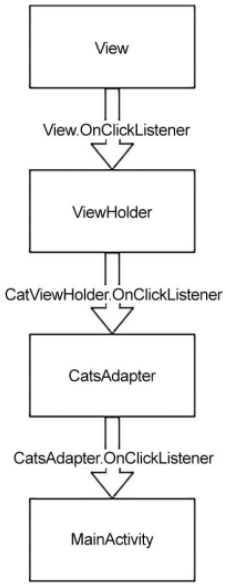


COMMIT to Github at this point

COMMIT Message: Commit No.1 - Adding a RecyclerView and Populating it

Part 2 - Clickable RecyclerView

1. Continue your "**LAB_WEEK_06"** project.
2. In this part, we will be focusing on how you can **set an OnClickListener for your RecyclerView** in Android. In order to set the necessary **OnClickListener**, we need to follow the below pattern.



3. Basically, the **Click Events** are delegated from **View** to **View Holder**, then delegated again from **View Holder** to **Adapter**, and lastly delegated from **Adapter** to **Main Activity**.
4. First, let's create and add the **OnClickListener** for our **ViewHolder**. Update your **CatViewHolder.kt** to the code below. Notice the **Highlighted** part of the code snippet is what is **changed** from the **previous** version.

```
class CatViewHolder(private val containerView: View, private val
imageLoader: ImageLoader, private val onClickListener: OnClickListener) :
RecyclerView.ViewHolder(containerView) {
   //containerView is the container layout of each item list
   //Here findViewById is used to get the reference of each views inside
the container
   private val catBiographyView: TextView by lazy {
       containerView.findViewById(R.id.cat_biography) }
   private val catBreedView: TextView by lazy {
```

```
        containerView.findViewById(R.id.cat_breed) }
    private val catGenderView: TextView by lazy {
        containerView.findViewById(R.id.cat_gender) }
    private val catNameView: TextView by lazy {
        containerView.findViewById(R.id.cat_name) }
    private val catPhotoView: ImageView by lazy {
        containerView.findViewById(R.id.cat_photo) }

    //This function is called in the adapter to provide the binding function
    fun bindData(cat: CatModel) {
        //Override the onClickListener function
        containerView.setOnClickListener {
            //Here we are using the onClickListener passed from the Adapter
            onClickListener.onItemClick(cat)
        }

        imageLoader.loadImage(cat.imageUrl, catPhotoView)
        catNameView.text = cat.name
        catBreedView.text = when (cat.breed) {
            CatBreed.AmericanCurl -> "American Curl"
            CatBreed.BalineseJavanese -> "Balinese-Javanese"
            CatBreed.ExoticShorthair -> "Exotic Shorthair"
            else -> "Unknown"
        }
        catBiographyView.text = cat.biography
        catGenderView.text = when (cat.gender) {
            Gender.Female -> FEMALE_SYMBOL
            Gender.Male -> MALE_SYMBOL
            else -> UNKNOWN_SYMBOL
        }
    }

    //Declare an onClickListener interface
    interface OnClickListener {
        fun onClick(cat: CatModel)
    }
}
```

5. Next, let's create and add the **OnClickListener** for our **Adapter.** Update your **CatAdapter.kt** to the code below.

```
class CatAdapter(private val layoutInflater: LayoutInflater, private val
```

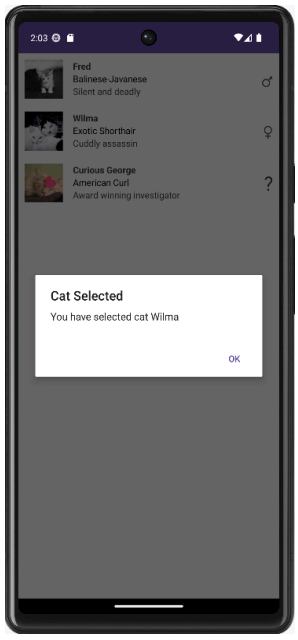
```
imageLoader: ImageLoader, private val onClickListener:
OnClickListener) : RecyclerView.Adapter<CatViewHolder>() {
    //Mutable list for storing all the list data
    private val cats = mutableListOf<CatModel>()

    //A function to set the mutable list
    fun setData(newCats: List<CatModel>) {
        cats.clear()
        cats.addAll(newCats)

        //This is used to tell the adapter that there's a data change in the
mutable list
        notifyDataSetChanged()
    }

    //A view holder is used to bind the data to the layout views
    //onCreateViewHolder is instantiating the view holder it self
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
CatViewHolder {
        val view = layoutInflater.inflate(R.layout.item_list, parent, false)
        return CatViewHolder(view, imageLoader, onClickListener)
    }

    //This is used to get the amount of data/item in the list
    override fun getItemCount() = cats.size

    //This is used to bind each data to each layout views
    override fun onBindViewHolder(holder: CatViewHolder, position: Int) {
        //The holder parameter stores our previously created ViewHolder
        //The holder.bindData function is declared in the CatViewHolder
        holder.bindData(cats[position])
    }

    //Declare an onClickListener interface
    interface OnClickListener {
        fun onItemClick(cat: CatModel)
    }
}
```

6. Lastly, create and add the **OnClickListener** for our **Main Activity.** Update your **MainActivity.kt** to the code below.

```
class MainActivity : AppCompatActivity() {
    private val recyclerView: RecyclerView by lazy {
        findViewById(R.id.recycler_view)
    }
    private val catAdapter by lazy {
        //Glide is used here to load the images
        //Here we are passing the onClickListener function to the Adapter
        CatAdapter(layoutInflater, GlideImageLoader(this), object:
CatAdapter.OnClickListener {
            //When this is triggered, the pop up dialog will be shown
            override fun onItemClick(cat: CatModel) = showSelectionDialog(cat)
        })
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //Setup the adapter for the recycler view
        recyclerView.adapter = catAdapter

        //Setup the layout manager for the recycler view
        //A layout manager is used to set the structure of the item views
        //For this tutorial, we're using the vertical linear structure
        recyclerView.layoutManager = LinearLayoutManager(this,
LinearLayoutManager.VERTICAL, false)

        //Add data to the model list in the adapter
        catAdapter.setData(
            listOf(
                CatModel(
                    Gender.Male,
                    CatBreed.BalineseJavanese,
                    "Fred",
                    "Silent and deadly",
                    "https://cdn2.thecatapi.com/images/7dj.jpg"
                ),
                CatModel(
                    Gender.Female,
                    CatBreed.ExoticShorthair,
                    "Wilma",
                    "Cuddly assassin",
                    "https://cdn2.thecatapi.com/images/egv.jpg"
                ),
                CatModel(
                    Gender.Unknown,
                    CatBreed.AmericanCurl,
```

```
                    "Curious George",
                    "Award winning investigator",
                    "https://cdn2.thecatapi.com/images/bar.jpg"
                )
            )
        )
    }

    //This will create a pop up dialog when one of the items from the recycler view
is clicked.
    private fun showSelectionDialog(cat: CatModel) {
        AlertDialog.Builder(this)
            //Set the title for the dialog
            .setTitle("Cat Selected")
            //Set the message for the dialog
            .setMessage("You have selected cat ${cat.name}")
            //Set if the OK button should be enabled
            .setPositiveButton("OK") { _, _ -> }.show()
    }
}
```

7. You may notice the `showSelectionDialog(cat: CatModel)` which is used to display a **Pop Up Dialog** when triggered. This will be triggered every time one of the item lists in the **Recycler View** is clicked.
8. **Run** your app and your **RecyclerView** should now be clickable.



COMMIT to Github at this point

Part 3 - Deleting an Item from RecyclerView by Swiping

1. Continue your "**LAB_WEEK_06"** project.
2. In this part, we will be focusing on how you can **delete an item from your RecyclerView** in Android. First, let's make a function to remove the item from our list. Add this function below your **setData function** in **CatAdapter.kt**.

```
fun removeItem(position: Int) {
    cats.removeAt(position)
    notifyItemRemoved(position)
}
```

3. Next, let's make the swiping functionality to our **RecyclerView**. Add the function below your **onClickListener Interface** in **CatAdapter.kt**.

```
//You can declare a class inside a class using the inner keyword
//Declare a class for the swipe functionality
inner class SwipeToDeleteCallback : ItemTouchHelper.SimpleCallback(0,
ItemTouchHelper.LEFT or ItemTouchHelper.RIGHT) {
    //This is used if item lists can be moved
    //Since we don't need that, we can set to false
    override fun onMove(
        recyclerView: RecyclerView,
        viewHolder: RecyclerView.ViewHolder,
        target: RecyclerView.ViewHolder
    ): Boolean = false

    //This is used to determine which directions are allowed
    override fun getMovementFlags(
        recyclerView: RecyclerView,
        viewHolder: RecyclerView.ViewHolder
    ) = if (viewHolder is CatViewHolder) {
        //Here, if we're not touching our phone, left and right are allowed
        makeMovementFlags(
            ItemTouchHelper.ACTION_STATE_IDLE,
            ItemTouchHelper.LEFT or ItemTouchHelper.RIGHT
        //Here, if we're swiping our phone, left and right are allowed
        ) or makeMovementFlags(
            ItemTouchHelper.ACTION_STATE_SWIPE,
            ItemTouchHelper.LEFT or ItemTouchHelper.RIGHT
        )
    //Other gestures are not allowed (Drag, etc.)
    } else {
```

```
        0
    }

    //This is used for swipe detection
    //If a swipe is detected, then remove item
    override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
        val position = viewHolder.adapterPosition
        removeItem(position)
    }
}
```

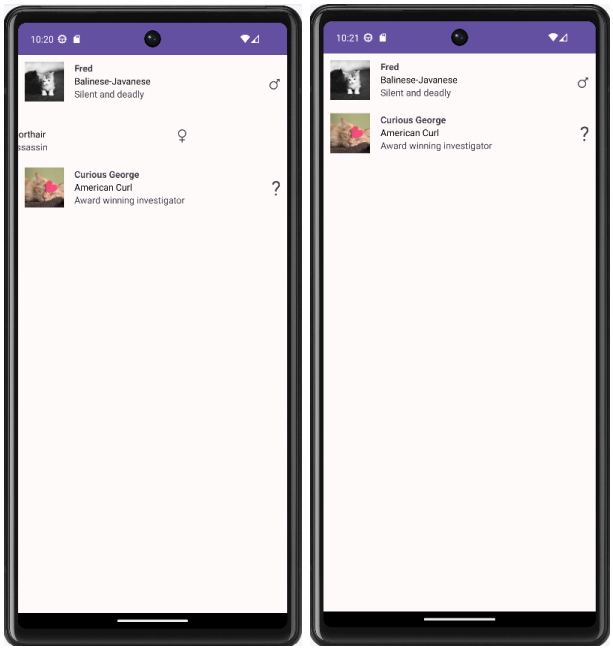
4. Now that you've made an **Inner Class**, time to instantiate that class in **CatAdapter.** On the very top before the **Cat Data List** declaration, add the code below.

```
//Delete Callback Instantiation
val swipeToDeleteCallback = SwipeToDeleteCallback()
```

5. Lastly, update your **MainActivity.kt** to the code below to attach the swipe functionality to our **RecyclerView**. Add the code below before the **setData function** and after the **layoutManager attachment**.

```
//Instantiate ItemTouchHelper for the swipe to delete callback and
//attach it to the recycler view
val itemTouchHelper = ItemTouchHelper(catAdapter.swipeToDeleteCallback)
itemTouchHelper.attachToRecyclerView(recyclerView)
```

6. **Run** your app, and everything should be working as intended.

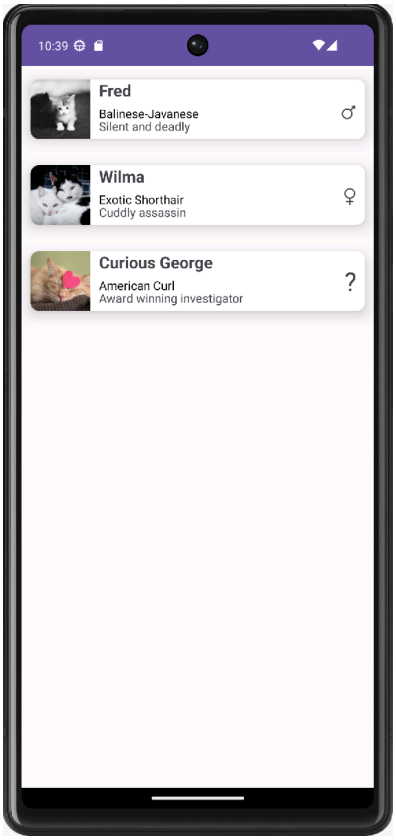


COMMIT to Github at this point
COMMIT Message: Commit No.3 - Deleting an Item from RecyclerView by Swiping

ASSIGNMENT

Continue your **LAB_WEEK_06** project, and:

1. Add more items until you have **10** lists at **Minimum**.
2. Your current view in **item_list.xml** doesn't really have anything interesting. Use **CardView** for your **item_list.xml** so your **RecyclerView** can look like this. Which is a lot cleaner for a list.


10:39
Fred
Balinese-Javanese
Silent and deadly
Wilma
Exotic Shorthair
Cuddly assassin
Curious George
American Curl
Award winning investigator