RecyclerView-10

In this assignment, you will expand your CriminalIntent app to include a list of crimes. CriminalIntent's model layer currently consists of a single instance of Crime. In this assignment, you will update CriminalIntent to work with a list of crimes.

1. Open your CriminalIntent app in Android Studio. Ensure you have committed your previous lab and submitted the commit ID before moving forward. Also, ensure you create a new repository on GitHub for this lab and hook it up to your project (don't forget to add collaborators).

2. The first step is to upgrade CriminalIntent's model layer from a single Crime object to a List of Crime objects. We will create a singleton called CrimeLab to do this. The CrimeLab singleton is not a solution for long-term storage of data, but it does allow the app to have one owner of the crime data and provides a way to easily pass that data between controller classes. Right-click the com.bignerdranch.android.criminalintent package and choose New → Java Class. Name this class CrimeLab and click OK.

3. In CrimeLab.java, implement CrimeLab as a singleton with a private constructor and a get() method:

```
public class CrimeLab {
   private static CrimeLab sCrimeLab;

   public static CrimeLab get(Context context) {
      if (sCrimeLab == null) {
         sCrimeLab = new CrimeLab(context);
      }
      return sCrimeLab;
   }

   private CrimeLab(Context context) {

   }
}
```

4. Let’s give CrimeLab some Crime objects to store. In CrimeLab’s constructor, create an empty List of Crimes. Also, add two methods: a getCrimes() method that returns the List and a getCrime(UUID) that returns the Crime with the given ID.

```
public class CrimeLab {
   private static CrimeLab sCrimeLab;

   private List<Crime> mCrimes;

   public static CrimeLab get(Context context) {
      ...
   }

   private CrimeLab(Context context) {
      mCrimes = new ArrayList<>();
   }

   public List<Crime> getCrimes() {
      return mCrimes;
   }

   public Crime getCrime(UUID id) {
      for (Crime crime : mCrimes) {
         if (crime.getId().equals(id)) {
            return crime;
         }
      }

      return null;
   }
}
```

5. Eventually, the List will contain user-created Crimes that can be saved and reloaded. For now, populate the List with 100 boring Crime objects:

```
private CrimeLab(Context context) {
   mCrimes = new ArrayList<>();

   for (int i = 0; i < 100; i++) {
      Crime crime = new Crime();
      crime.setTitle("Crime #" + i);
      crime.setSolved(false);
      mCrimes.add(crime);
    }
}
```

6. Because activity_crime.xml does not name a particular fragment, we are going to use it for any activity we create that needs to host a single fragment. We are going to rename it to activity_fragment.xml to reflect its larger scope. In the project tool window, right-click res/layout/activity_crime.xml. (Be sure to right-click activity_crime.xml and not fragment_crime.xml). From the context menu, select Refactor → Rename.... Rename this layout activity_fragment.xml and click Refactor.

7. When you rename a resource, the references to it should be updated automatically. If you see an error in CrimeActivity.java, then you need to manually update the reference in CrimeActivity to reflect this name change.

8. Let's create our abstract class that we will reuse to save on coding. Right-click on the com.bignerdranch.android.criminalintent package, select New → Java Class, and name the new class SingleFragmentActivity. Make this class a subclass of AppCompatActivity and make it an abstract class. Your generated file should look like this:

```
public abstract class SingleFragmentActivity extends AppCompatActivity {

}
```

9. Now, add the following code to SingleFragmentActivity.java (notice it is almost identical to your CrimeActivity.java code, other than the createFragment() code.

```
public abstract class SingleFragmentActivity extends AppCompatActivity {
   protected abstract Fragment createFragment();

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

      FragmentManager fm = getSupportFragmentManager();
      Fragment fragment = fm.findFragmentById(R.id.fragment_container);

      if (fragment == null) {
         fragment = createFragment();
         fm.beginTransaction()
            .add(R.id.fragment_container, fragment)
            .commit();
      }
   }
}
```

10. Try out your abstract class with CrimeActivity. Change CrimeActivity's superclass to SingleFragmentActivity, remove the implementation of onCreate(Bundle), and implement the createFragment() method:

```
public class CrimeActivity extends ~~AppCompatActivity~~ SingleFragmentActivity {
~~    /** Called when the activity is first created. */~~
~~    @Override~~
~~    protected void onCreate(Bundle savedInstanceState) {~~
~~        super.onCreate(savedInstanceState);~~
~~        setContentView(R.layout.activity_fragment);~~

~~        FragmentManager fm = getSupportFragmentManager();~~
~~        Fragment fragment = fm.findFragmentById(R.id.fragment_container);~~

~~        if (fragment == null) {~~
~~            fragment = new CrimeFragment();~~
~~            fm.beginTransaction()~~
~~                .add(R.id.fragment_container, fragment)~~
~~                .commit();~~
~~        }~~
~~    }~~

    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();
    }
}
```

11. Now, you will create the two new controller classes: CrimeListActivity and CrimeListFragment. Right-click on the com.bignerdranch.android.criminalintent package, select New → Java Class, and name the class CrimeListActivity.

12. Modify the new CrimeListActivity class to also subclass SingleFragmentActivity and implement the createFragment() method.

```
public class CrimeListActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }
}
```

13. If you have other methods in your CrimeListActivity, such as onCreate, remove them. Let SingleFragmentActivity do its job and keep CrimeListActivity simple.

14. The CrimeListFragment class has not yet been created. Let's do that now. Right-click on the com.bignerdranch.android.criminalintent package again, select New → Java Class, and name the class CrimeListFragment.

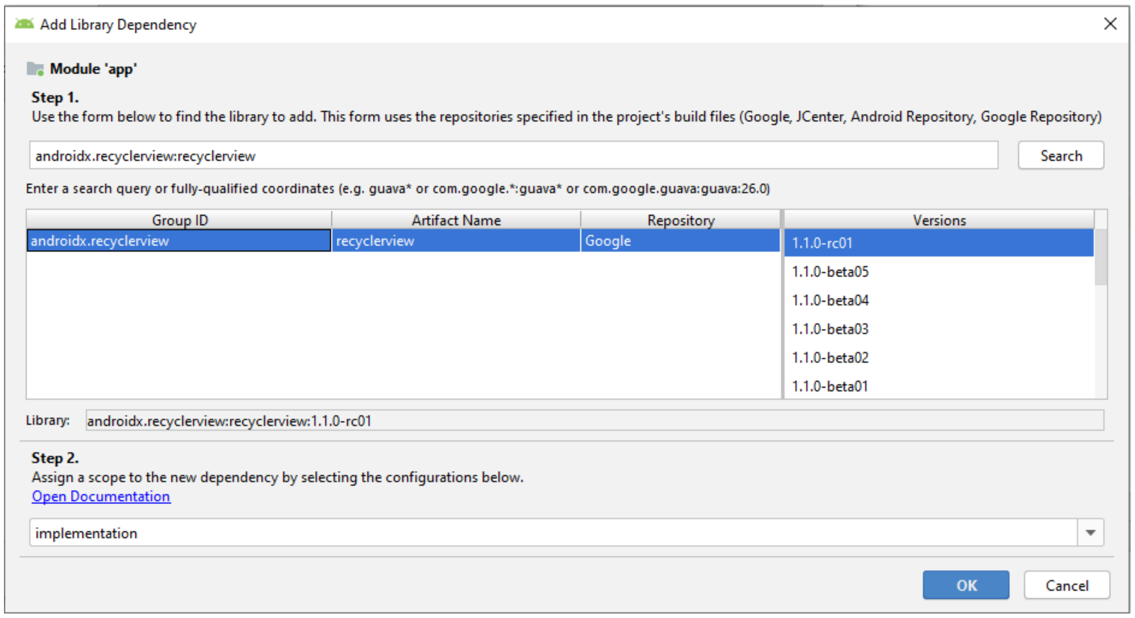
```
public class CrimeListFragment extends Fragment {
   // Nothing yet
}
```

15. Now that you have created CrimeListActivity, you must declare it in the manifest. In addition, you want the list of crimes to be the first screen that the user sees when CriminalIntent is launched, so CrimeListActivity should be the launcher activity. In the manifest, declare CrimeListActivity and move the launcher intent filter from CrimeActivity's declaration to CrimeListActivity's declaration.

```
<application … >
  <activity android:name=".CrimeListActivity">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
  <activity android:name=".CrimeActivity">
    ~~<intent-filter>~~
      ~~<action android:name="android.intent.action.MAIN" />~~
      ~~<category android:name="android.intent.category.LAUNCHER" />~~
    ~~</intent-filter>~~
  </activity>

</application>
```

16. CrimeListActivity is now the launcher activity. Run CriminalIntent and you will see CrimeListActivity's FrameLayout hosting an empty CrimeListFragment.

17. Let's start work on our RecyclerView. The RecyclerView class lives in one of Google's many support libraries. The first step to using a RecyclerView is to add the RecyclerView library as a dependency. Navigate to your project structure window with File → Project Structure.... Select Dependencies on the left, then the app module. Use the + button and choose Library dependency to add a dependency. Find and select the androidx.recyclerview:recyclerview library and click OK to add the library as a dependency:



Step 1.
Use the form below to find the library to add. This form uses the repositories specified in the project's build files (Google, JCenter, Android Repository, Google Repository)

androidx.recyclerview:recyclerview

Enter a search query or fully-qualified coordinates (e.g. guava* or com.google.*:guava* or com.google.guava:guava:26.0)

Group ID	Artifact Name	Repository
androidx.recyclerview	recyclerview	Google

Versions
1.1.0-rc01
1.1.0-beta05
1.1.0-beta04
1.1.0-beta03
1.1.0-beta02
1.1.0-beta01

Library: androidx.recyclerview:recyclerview:1.1.0-rc01

Step 2.
Assign a scope to the new dependency by selecting the configurations below.
Open Documentation

implementation

18. Your RecyclerView will live in CrimeListFragment's layout file. First, you must create the layout file. Right-click on the res/layout directory and select New → Layout resource file. Name the file fragment_crime_list and click OK to create the file.

19. Open the new fragment_crime_list file and modify the root view to be a RecyclerView and to give it an ID attribute:

```
~~<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"~~
~~        android:orientation="vertical"~~
~~        android:layout_width="match_parent"~~
~~        android:layout_height="match_parent">~~

~~</LinearLayout>~~
<androidx.recyclerview.widget.RecyclerView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/crime_recycler_view"
  android:layout_width="match_parent"
  android:layout_height="match_parent"/>
```

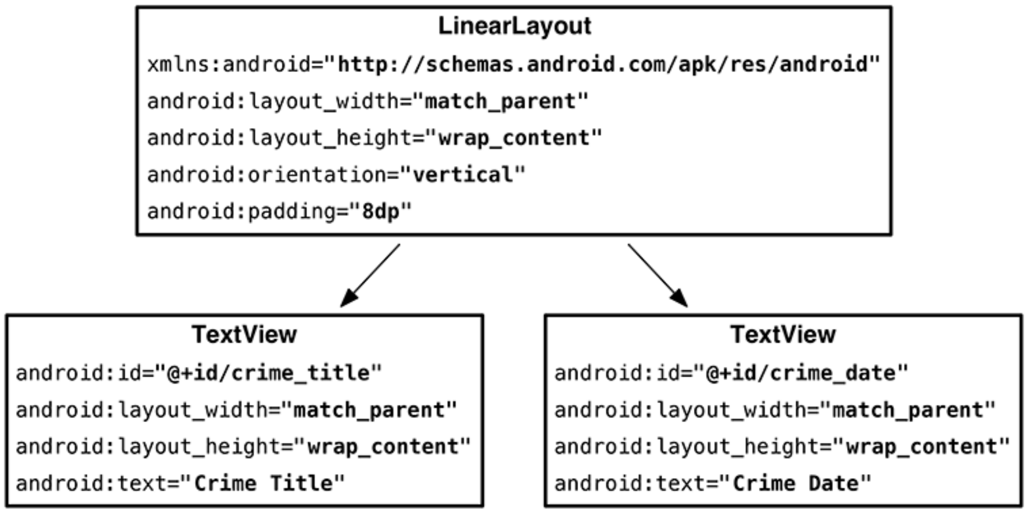
20. Now that CrimeListFragment’s view is set up, hook up the view to the fragment. Modify CrimeListFragment to use this layout file and to find the RecyclerView in the layout file:

```
public class CrimeListFragment extends Fragment {
    ~~// Nothing yet~~
    private RecyclerView mCrimeRecyclerView;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_crime_list, container, false);

        mCrimeRecyclerView = (RecyclerView) view
                .findViewById(R.id.crime_recycler_view);
        mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

        return view;
    }
}
```

21. Run the app. You should again see a blank screen, but now you are looking at an empty RecyclerView. You will not see any Crimes represented on the screen until the Adapter and ViewHolder implementations are defined.

22. Each item displayed on the RecyclerView will have its own view hierarchy, exactly the way CrimeFragment has a view hierarchy for the entire screen. You create a new layout for a list item view the same way you do for the view of an activity or a fragment. In the project tool window, right-click the res/layout directory and choose New → Layout resource file. In the dialog that appears, name the file list_item_crime and click OK.

23. Update your layout file to add the two TextViews:



```
LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:orientation="vertical"
android:padding="8dp"

TextView
android:id="@+id/crime_title"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Crime Title"

TextView
android:id="@+id/crime_date"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Crime Date"
```

24. The next job is to define the ViewHolder that will inflate and own your layout. Define it as an inner class in CrimeListFragment:

```
public class CrimeListFragment extends Fragment {
    ...
    private class CrimeHolder extends RecyclerView.ViewHolder {
        public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
            super(inflater.inflate(R.layout.list_item_crime, parent, false));
        }
    }
}
```

25. Create the adapter:

```
public class CrimeListFragment extends Fragment {
    ...
    private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {

        private List<Crime> mCrimes;

        public CrimeAdapter(List<Crime> crimes) {
            mCrimes = crimes;
        }
    }
}
```

26. Next, implement three method overrides in CrimeAdapter. (You can automatically generate these overrides by putting your cursor on top of extends and pressing Alt+Enter, selecting Implement methods, and clicking OK. Then you only need to fill in the bodies.)

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
        ...
        @Override
        public CrimeHolder onCreateViewHolder(ViewGroup parent, int viewType) {
            LayoutInflater layoutInflater = LayoutInflater.from(getActivity());

            return new CrimeHolder(layoutInflater, parent);
        }
        @Override
        public void onBindViewHolder(CrimeHolder holder, int position) {

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

27. Now that you have an Adapter, connect it to your RecyclerView. Implement a method called updateUI that sets up CrimeListFragment's UI. For now it will create a CrimeAdapter and set it on the RecyclerView.

```
public class CrimeListFragment extends Fragment {

  private RecyclerView mCrimeRecyclerView;
  private CrimeAdapter mAdapter;

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
                    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_crime_list, container, false);

    mCrimeRecyclerView = (RecyclerView) view
          .findViewById(R.id.crime_recycler_view);
    mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

    updateUI();

    return view;
  }

  private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    mAdapter = new CrimeAdapter(crimes);
    mCrimeRecyclerView.setAdapter(mAdapter);
  }

  ...
}
```

28. Run CriminalIntent and scroll through your new RecyclerView.

29. Add code to your CrimeHolder class to bind the list items. This starts by pulling out all the widgets you are interested in. This only needs to happen one time, so write this code in your constructor:

```
private class CrimeHolder extends RecyclerView.ViewHolder {

   private TextView mTitleTextView;
   private TextView mDateTextView;

   public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
      super(inflater.inflate(R.layout.list_item_crime, parent, false));

      mTitleTextView = (TextView) itemView.findViewById(R.id.crime_title);
      mDateTextView = (TextView) itemView.findViewById(R.id.crime_date);
   }
}
```

30. Your CrimeHolder will also need a bind(Crime) method. This will be called each time a new Crime should be displayed in your CrimeHolder. First, add bind(Crime).

```
private class CrimeHolder extends RecyclerView.ViewHolder {

   private Crime mCrime;
   ...
   public void bind(Crime crime) {
      mCrime = crime;
      mTitleTextView.setText(mCrime.getTitle());
      mDateTextView.setText(mCrime.getDate().toString());
   }
}
```

31. When given a Crime, CrimeHolder will now update the title TextView and date TextView to reflect the state of the Crime. Next, call your newly minted bind(Crime) method each time the RecyclerView requests that a given CrimeHolder be bound to a particular crime.

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
   ...
   @Override
   public void onBindViewHolder(CrimeHolder holder, int position) {
      Crime crime = mCrimes.get(position);
      holder.bind(crime);
   }
   ...
}
```

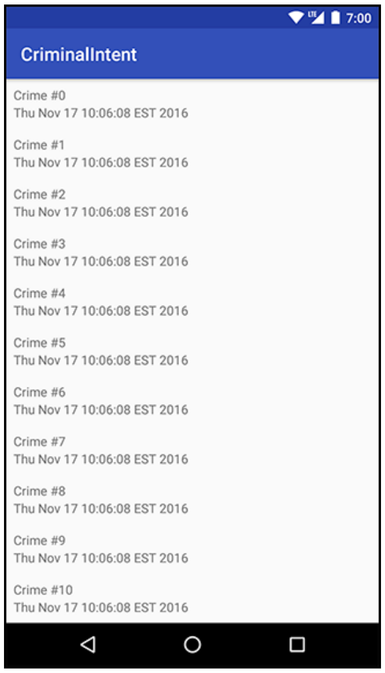
32. Run CriminalIntent one more time, and every visible CrimeHolder should now display a distinct Crime.

33. Modify the CrimeHolder to handle presses for the entire row.

```
private class CrimeHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
   ...
   public CrimeHolder(LayoutInflater inflater, ViewGroup parent) {
      super(inflater.inflate(R.layout.list_item_crime, parent, false));
      itemView.setOnClickListener(this);
      ...
   }
   ...
   @Override
   public void onClick(View view) {
      Toast.makeText(getActivity(),
            mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
            .show();
   }
}
```

34. Run CriminalIntent and press on an item in the list. You should see a Toast indicating that the item was clicked.

Final App Example



Crime #0
Thu Nov 17 10:06:08 EST 2016

Crime #1
Thu Nov 17 10:06:08 EST 2016

Crime #2
Thu Nov 17 10:06:08 EST 2016

Crime #3
Thu Nov 17 10:06:08 EST 2016

Crime #4
Thu Nov 17 10:06:08 EST 2016

Crime #5
Thu Nov 17 10:06:08 EST 2016

Crime #6
Thu Nov 17 10:06:08 EST 2016

Crime #7
Thu Nov 17 10:06:08 EST 2016

Crime #8
Thu Nov 17 10:06:08 EST 2016

Crime #9
Thu Nov 17 10:06:08 EST 2016

Crime #10
Thu Nov 17 10:06:08 EST 2016