RECYCLER VIEW

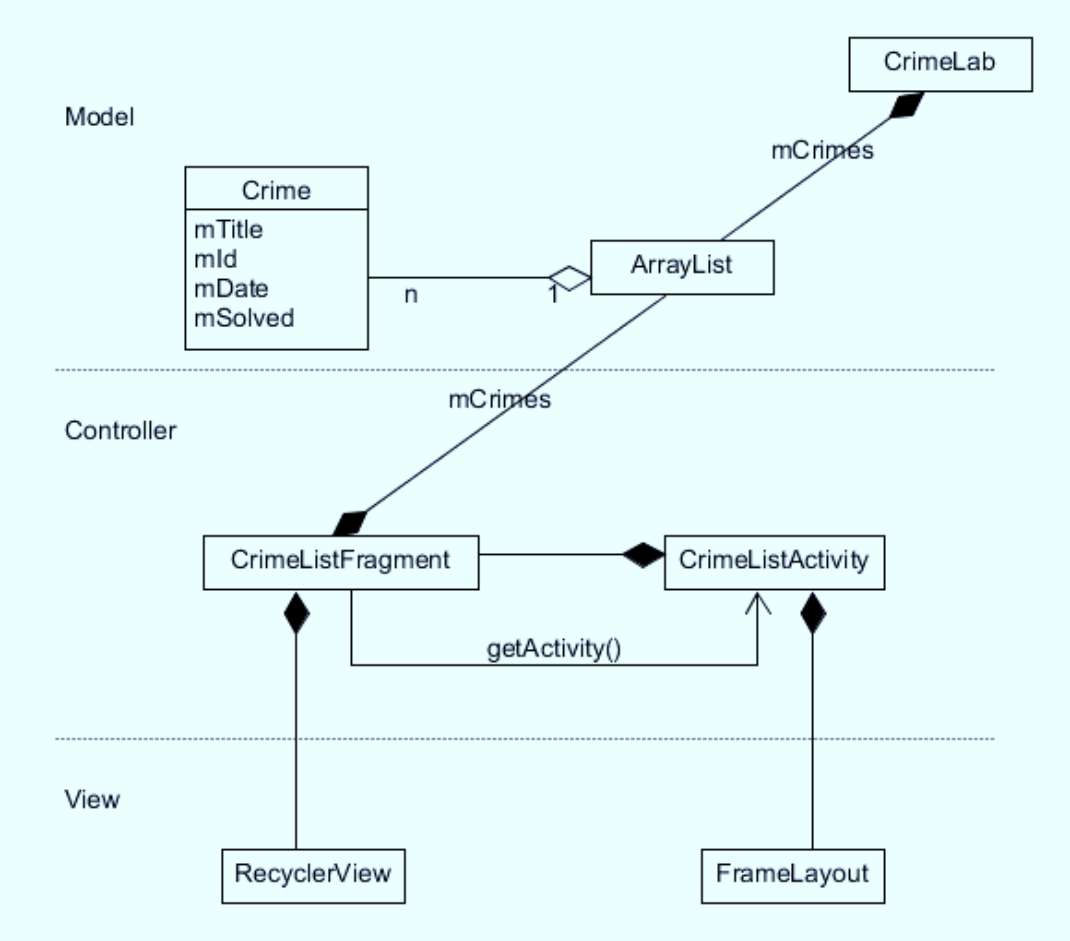
MOBILE APPLICATION DEVELOPMENT CS 347

INSTRUCTOR: PHILIP GAROFALO, M.S.

NEIU

THE CRIMINAL INTENT APP

CRIMINAL INTENT CLASS DIAGRAM



We're going to add these classes to the basic CriminalIntent app we built in Chapter 7.

CRIME LAB CLASS: A SINGLETON

- A singleton is a class that only permits a single object instance.
- It is implemented by making its constructor(s) private.
- It adds a static factory method that only instantiates the object once.
- Add this class.

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

    private CrimeLab(Context context) {

    }

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

LET'S FLESH OUT THE CRIME LAB CLASS

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

    private List<Crime> mCrimes;

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

    }

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

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

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

GENERATE DATA FOR THE ARRAY

In the constructor, add the following loop:

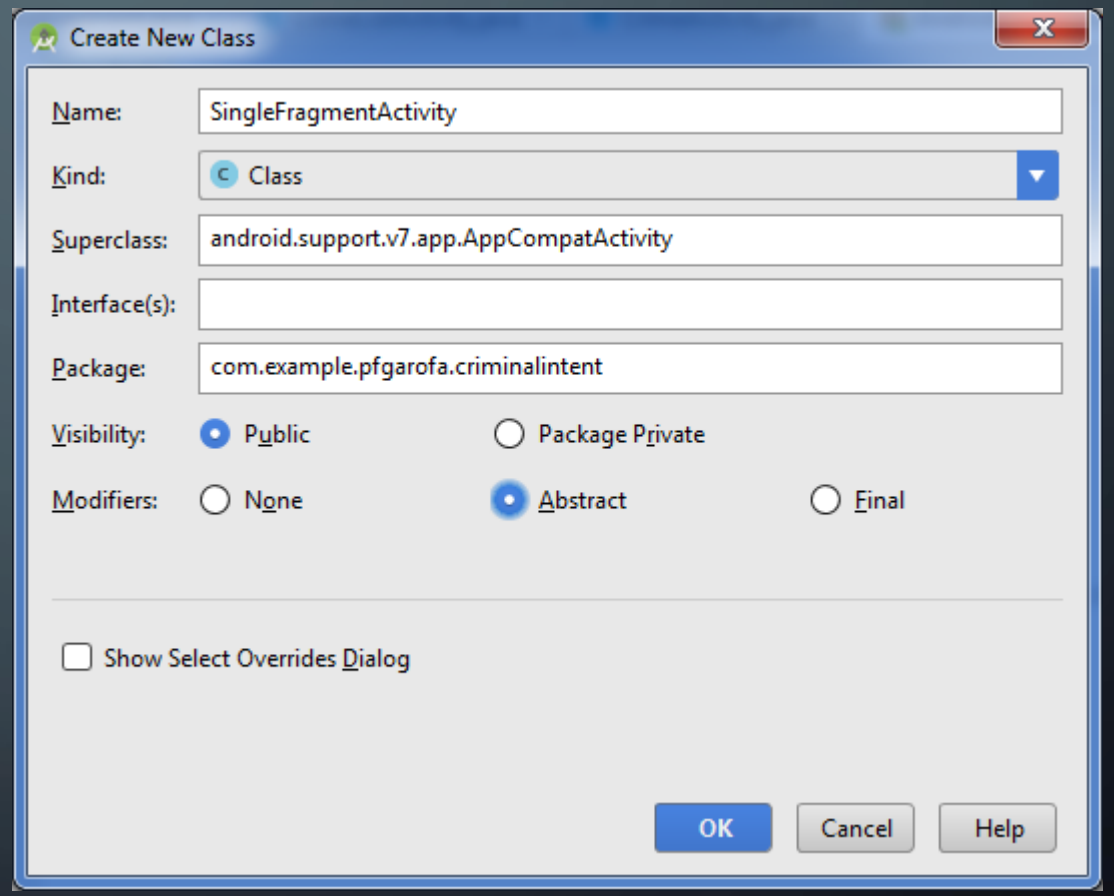
```
private CrimeLab(Context context) {
    mCrimes = new ArrayList<>();
    // generate 100 records
    for (int i = 0; i < 100; i++) {
        Crime crime = new Crime();
        crime.setTitle("Crime #" + i);
        crime.setSolved(i % 2 == 0); // every other record
        mCrimes.add(crime);
    }
}
```

USING AN ABSTRACT ACTIVITY CLASS

- We're going to create the CrimeListActivity class based on an ***abstract*** super class.
- Before we do, we're going to set up its view, then create the abstract class.
- Rename **activity_crime.xml** to **activity_fragment.xml**.
  - Right click on res/layout/activity_crime.xml
  - Select **Refactor>Rename** in the pop-up menus.
- The reference in **CrimeActivity.onCreate()** will automatically be updated.

ABSTRACT CLASS (PART 2)

- Create SingleFragmentActivity.
  - Right click on java/package.name
  - Select **New>Java Class**
  - Name is **SingleFragmentActivity.**
  - Superclass is **AppCompatActivity.**
  - Check the **Abstract** modifier.



ABSTRACT CLASS (PART 3)

```
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();
        }
    }
}
```

ABSTRACT CLASS (PART 4)

```
public class CrimeActivity extends SingleFragmentActivity {

    // DELETE onCreate()

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

ADDING TWO CONTROLLER CLASSES

- Now we're going to create the two controller classes, **CrimeListActivity** and **CrimeListFragment**.
- CrimeListActivity will ***subclass*** SingleFragmentActivity.
- Right click on java/package.name and select **New>Java Class**.
- Name the class **CrimeListActivity**.
- Modify the class as shown.

```
public class CrimeListActivity
  extends SingleFragmentActivity {

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

ADD THE SECOND CONTROLLER

- Right click on java/package.name and select **New>Java Class.**
- Name the class **CrimeListFragment.**
- Enter **Fragment** (support.v4 library) for its superclass.
- Click **OK.**

CHANGE THE LAUNCH ACTIVITY

- We’re going to change the launch activity to be the CrimeListActivity.
- It will be the first activity the user sees when they start the app.
- Modify AndroidManifest.xml as shown on the following slide.

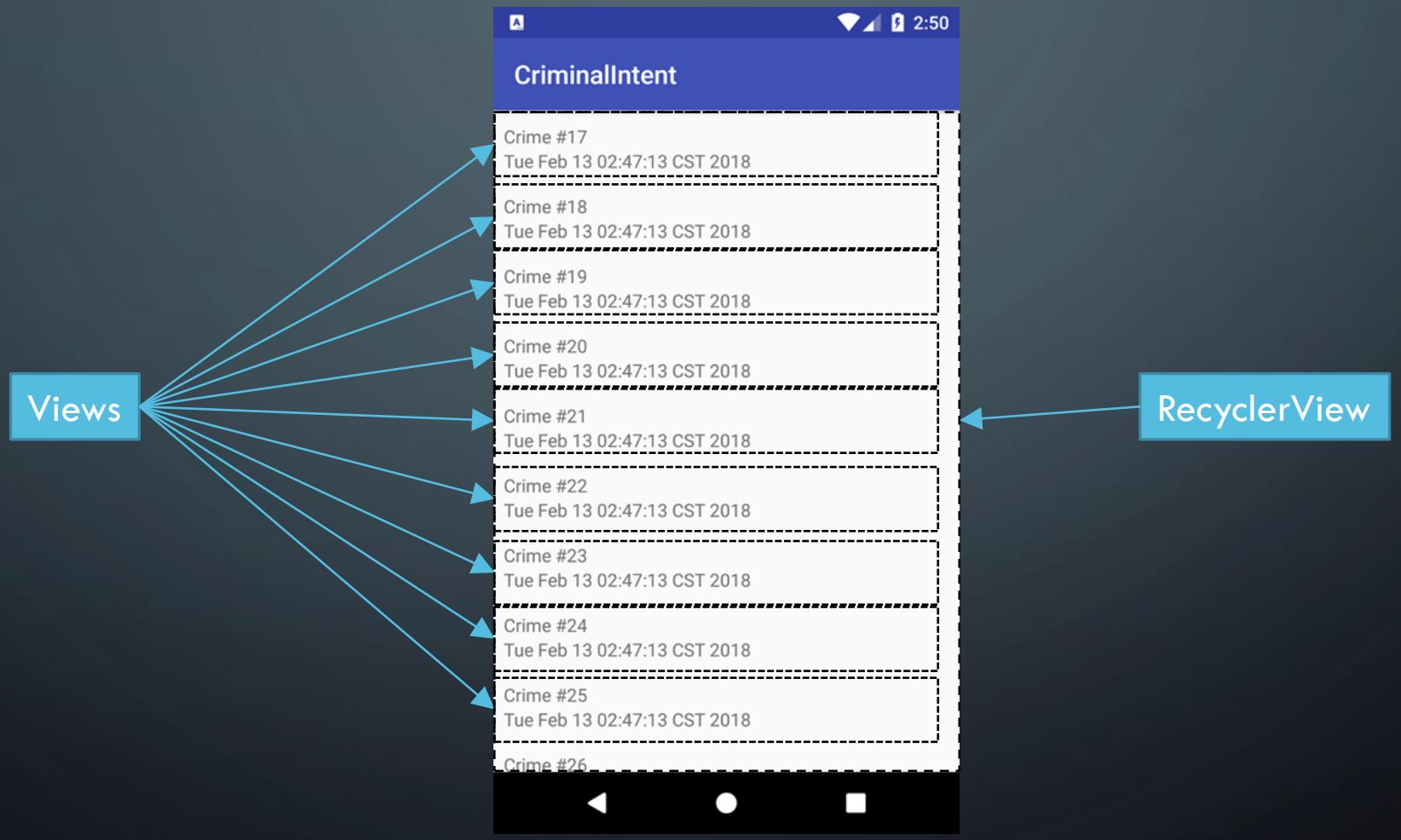
CHANGE THE LAUNCH ACTIVITY (PART 2)

```
<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <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">
             <!-- MOVE THE INTENT-FILTER ELEMENT TO THE CRIME LIST ACTIVITY -->
        </activity>
    </application>
```

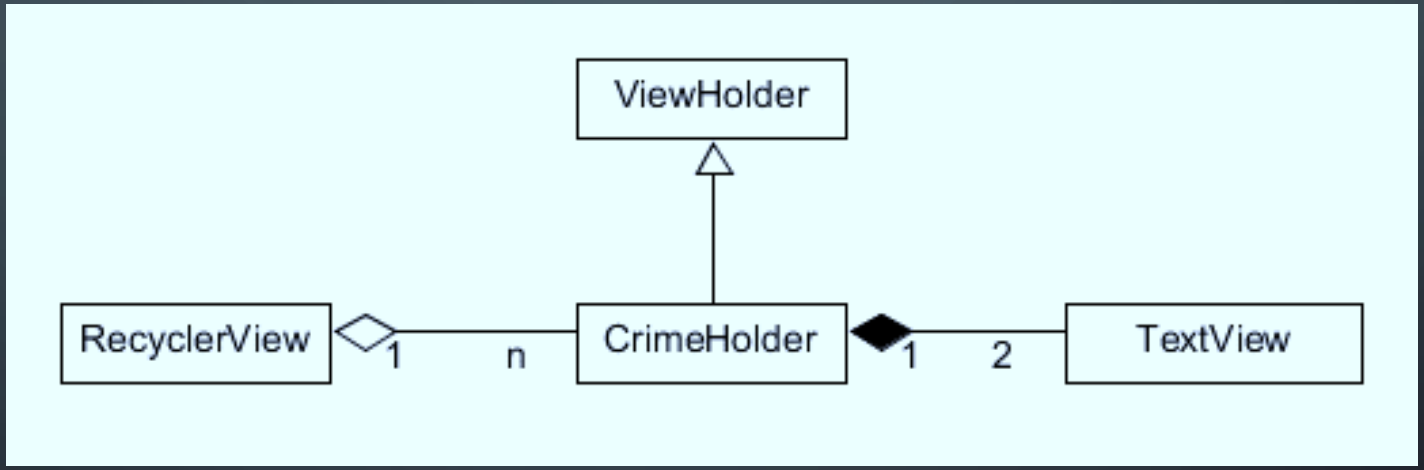
RUN IT

- You'll see a blank CrimeListFragment in a FrameLayout.
- That's to be expected because we need to add the list and adapter classes.

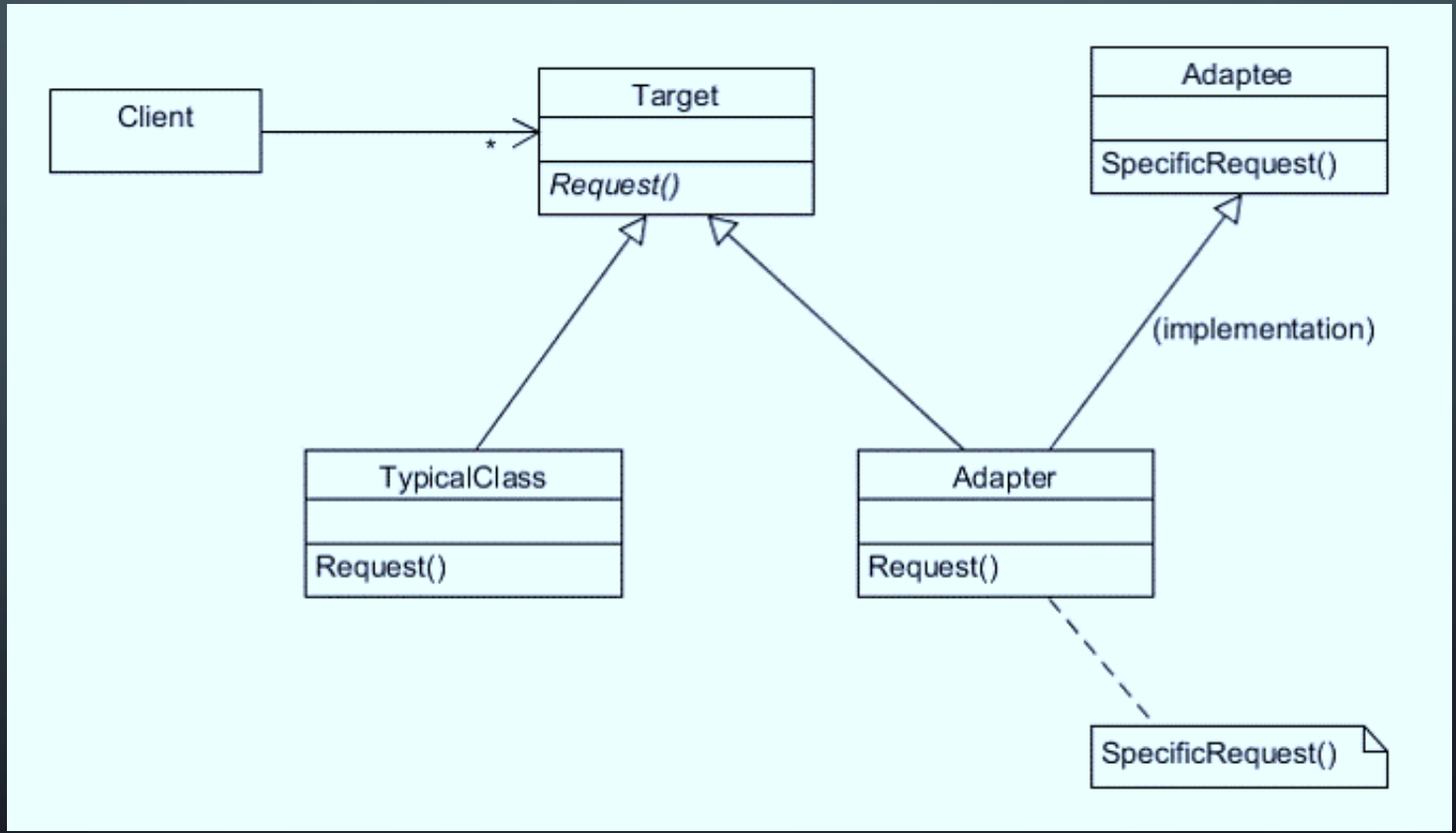
RECYCLER VIEW, ADAPTER, AND VIEW HOLDER



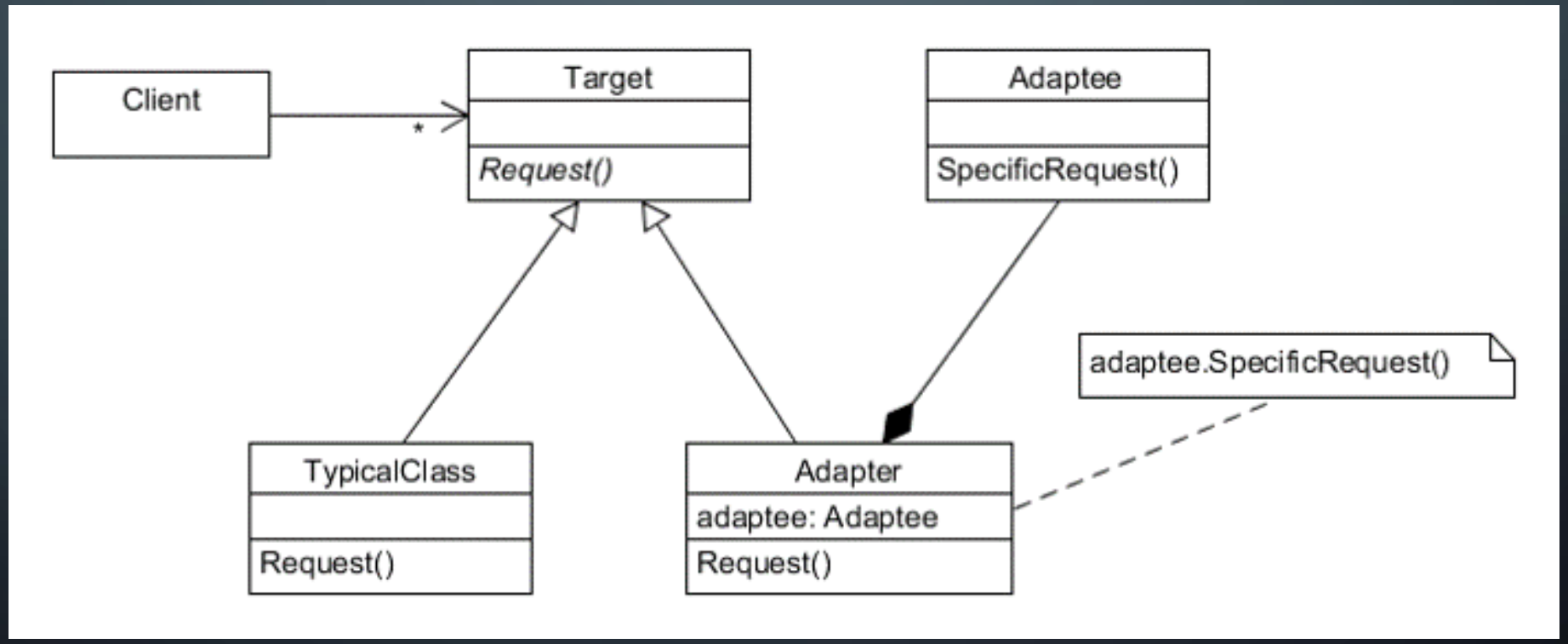
VIEW HOLDERS



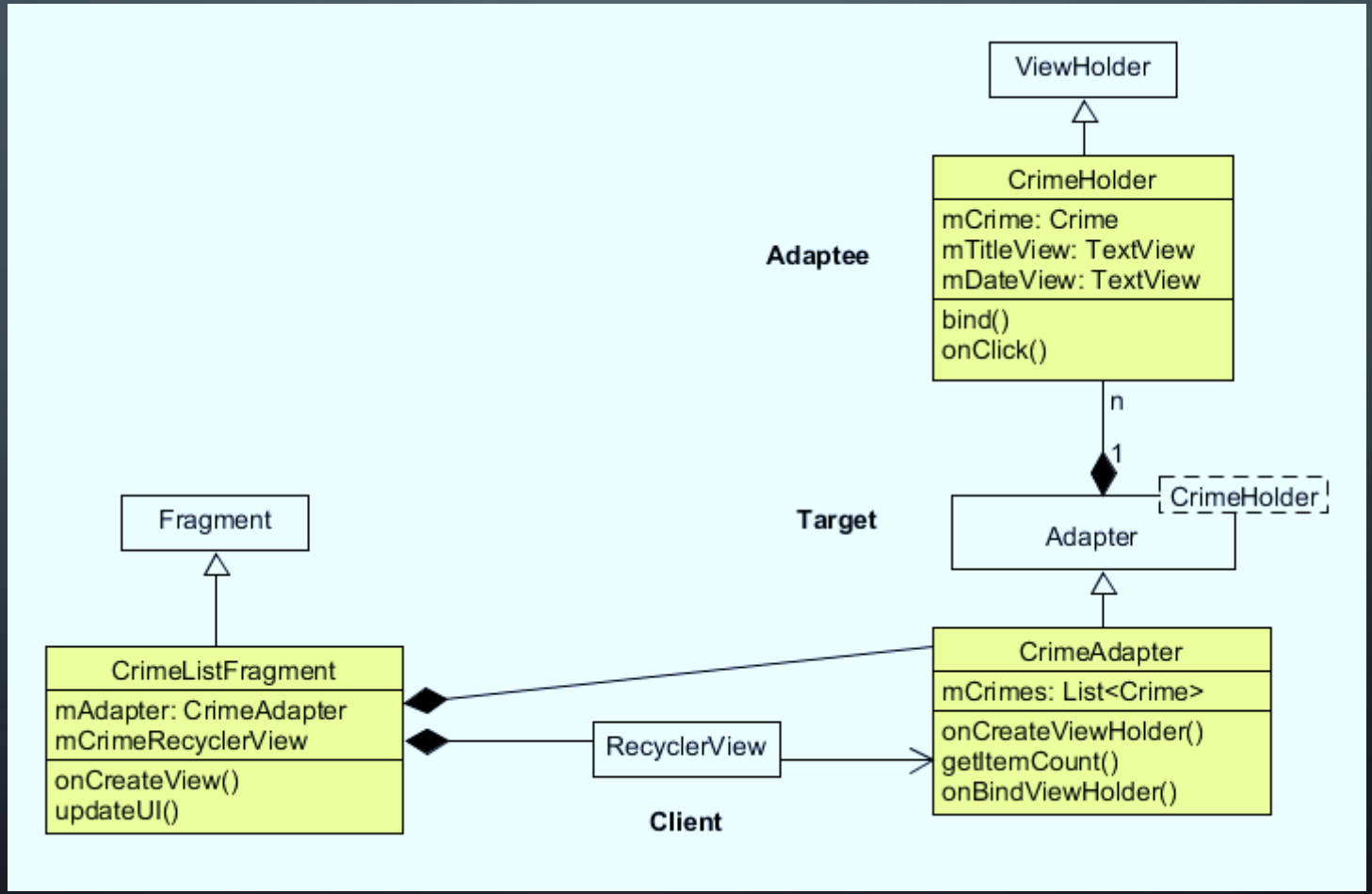
ADAPTER: CLASS ADAPTER PATTERN



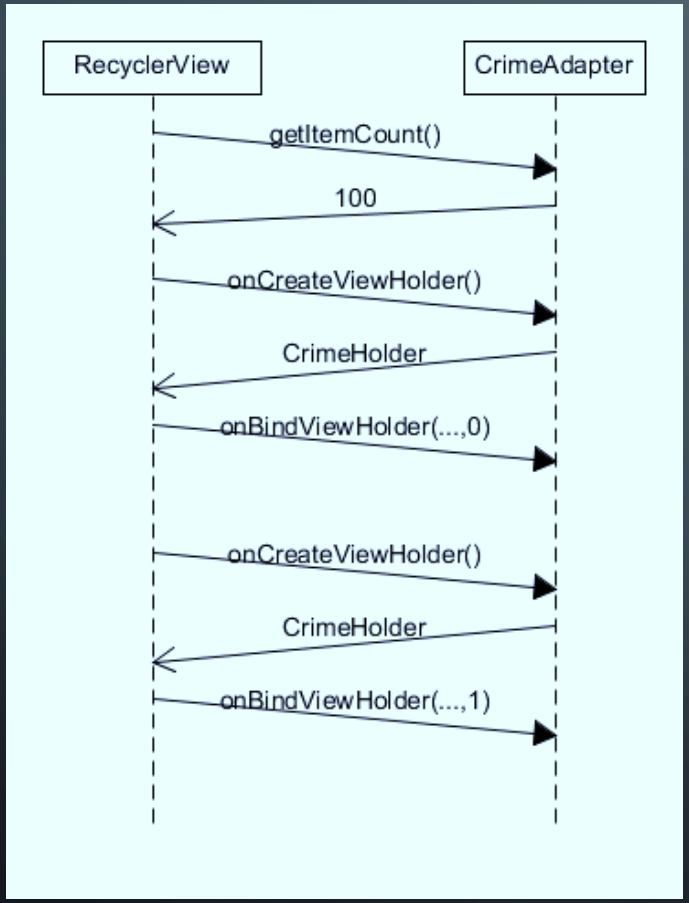
ADAPTERS: OBJECT ADAPTER PATTERN



ADAPTER FOR RECYCLER VIEW



ADAPTER CALLING SEQUENCE



- The **RecyclerView** first calls **getItemCount()** for the number of items in the list.
- It then loops or each item, as needed:
  1. If a new **ViewHolder** is needed, it calls **onBindViewHolder()** otherwise it recycles an existing one.
  2. It then calls **onBinViewHolder** with the item number.
  3. The adapter looks up the item number in the list and populates its fields with the item data.

ADAPTER IMPLEMENTATION

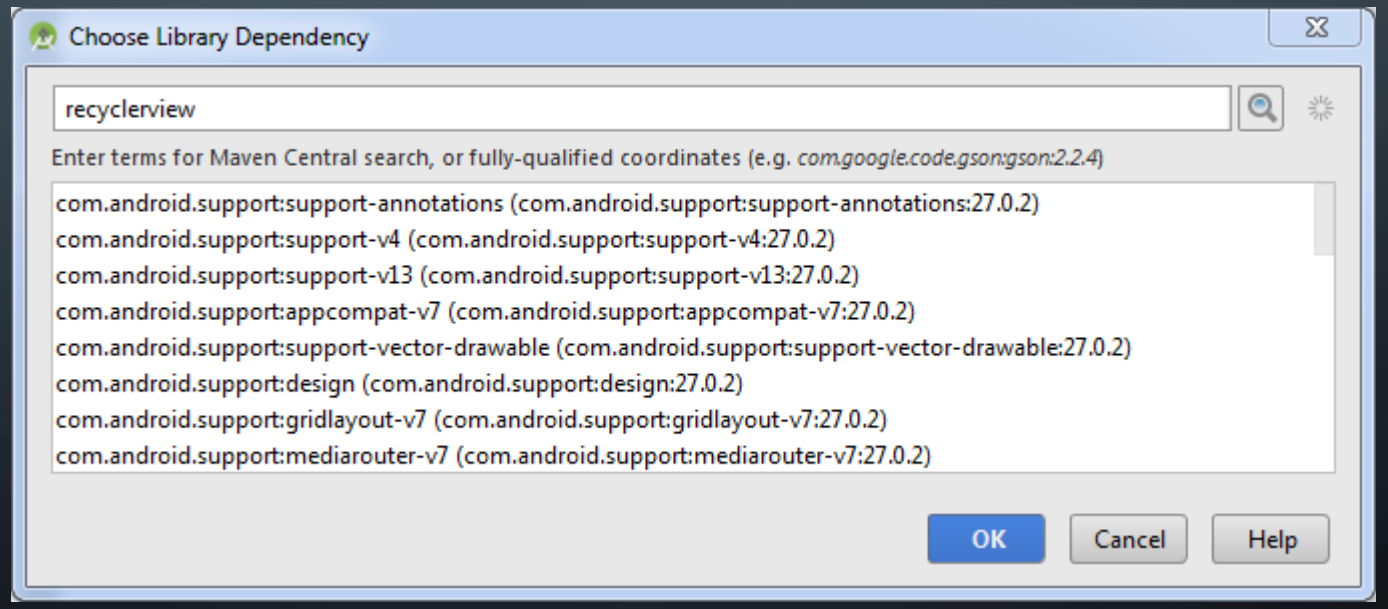
We're going to implement **three classes:**

1. A fragment, **CrimeListFragment**, derived from Fragment.
2. A view holder, **CrimeHolder**, derived from `RecycleView.ViewHolder`.
3. An adapter, **CrimeAdapter**, derived from `RecycleView.Adapter<CrimeHolder>`.

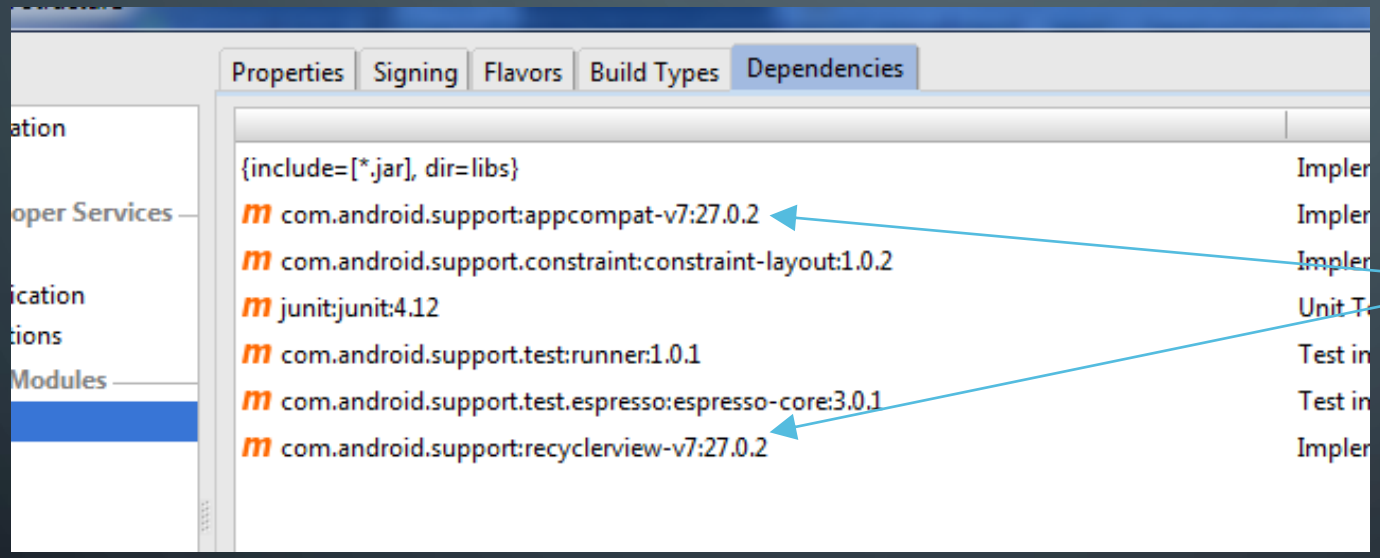
The view holder and adapter will be **top-level nested classes** of CrimeListFragment.

RECYCLER VIEW

- You have to manually add the RecyclerView support library.
  - Select **File, Project Structure..., App, Dependencies, +, Library dependency**
  - Search for "recyclerview"



RECYCLER VIEW (PART 2)



Support library version numbers have to match so you might need to update the other support library dependencies. (Except for test.)

RECYCLER VIEW (PART 3)

Replace the LinearLayout in fragment_crime_list.xml with a RecyclerView.

```
<android.support.v7.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">
</android.support.v7.widget.RecyclerView>
```

RECYCLER VIEW (PART 2)

Add to the CrimeListFragment

```
public class CrimeListFragment extends Fragment {
    private RecyclerView mCrimeRecyclerView;

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

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

        return v;
    }
}
```

ADD A LIST ITEM LAYOUT

- Right click on **res/layout** select **New>Layout.**
- Name it **list_item_crime.xml.**

```
<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="@string/crime_title" />

    <TextView
        android:id="@+id/crime_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_date" />

</LinearLayout>
```

START TO IMPLEMENT THE VIEW HOLDER

```
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));
    }
  }
}
```

START TO IMPLEMENT 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;
    }
  }
}
```

FILLING OUT CRIME ADAPTER

```
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();
        }
}
```

Automatically generate these by right-clicking on "extends" and selecting "Implement methods".

SETTING THE ADAPTER

```
public class CrimeListFragment extends Fragment {
    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime_list, container, false);
        mCrimeRecyclerView = v.findViewById(R.id.crime_recycler_view);
        mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager(getActivity()));

        CrimeLab crimeLab = CrimeLab.get(getActivity());
        List<Crime> crimes = crimeLab.getCrimes();
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);

        return v;
    }
}
```

RUN IT!

- You'll repeated series of tool text.
- Now we'll **bind** the actual list values with the view holder views on the RecyclerView.

BINDING LIST ITEMS

Not done with the view holder yet...

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

BINDING LIST ITEMS (PART 2)

```
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());
        }
}
```

BINDING LIST ITEMS (PART 3)

Call CrimeHolder.bind() from CrimeAdapter.onBindViewHolder().

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

RUN IT AGAIN!

Now you'll see actual data on the screen.

RESPONDING TO TAPS

To respond to taps on individual list items, we make the view holder implement the OnClickListener interface.

```
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);
      ...
```

RESPONDING TO TAPS (PART 2)

...and then add the following override. Right click on the CrimeHolder class declaration and select **Implement Overrides**.

```
    ...
    @Override
    public void onClick(View v) {
        Toast.makeText(getActivity(),
                mCrime.getTitle() + " clicked!",
                Toast.LENGTH_SHORT).show();
    }
}
```

NOW RUN IT!

You should see a scrollable list of items, that when clicked on, display a toast.

WHAT WE LEARNED

- On the path to understanding **RecyclerView,** we also learned about:
- **Singletons**
- **Abstract classes**
- **Adapter pattern**