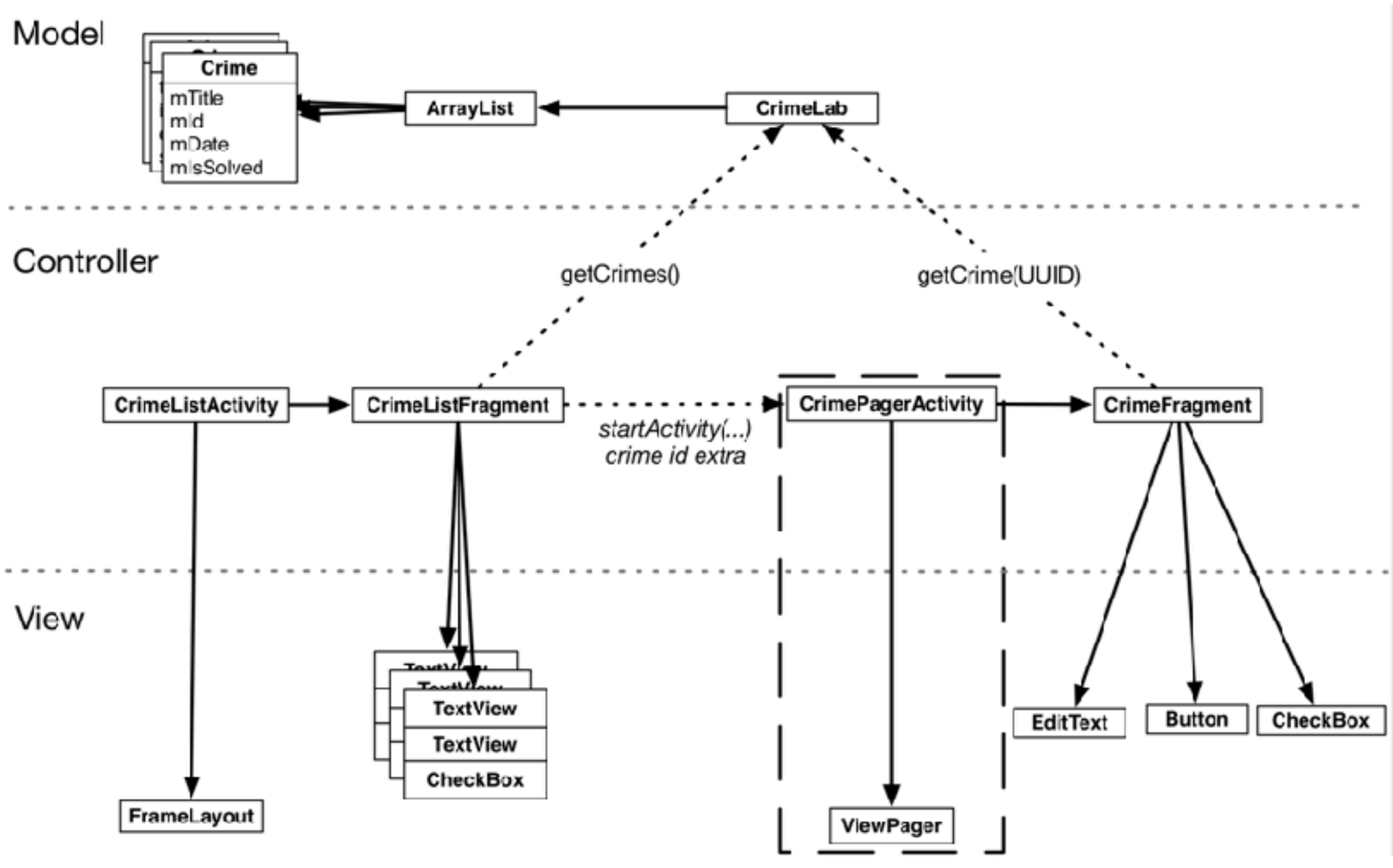
ANDROID PROGRAMMING

Course 11 -Using ViewPager

- In this chapter, you will create a new activity to host CrimeFragment. This activity's layout will consists of an instance of ViewPager. Adding a ViewPager to your UI lets users navigate between list items by swapping across the screen to "page" forward or backward through the crimes as seen on Figure.

- In the following Figure, the updated CriminalIntent is shown. The new activity will be named CrimePagerActivity and will take the place of CrimeActivity. Its layout will consists of a ViewPager.



- The only new objects you need to create are within the dashed rectangle in the diagram. Nothing else CriminalIntent needs to change to implement paging between detail views. In particular, you will not have to touch the CrimeFragment class thanks to work you did in previous lesson to ensure CrimeFragment's independence.

- Here are the tasks ahead in this chapter:

  - create the **CrimePagerActivity** class
  - define a view hierarchy that consists of a **ViewPager**
  - wire up the **ViewPager** and its adapter in **CrimePagerActivity**
  - modify CrimeHolder.onClick(...) to start CrimePagerActivity instead of CrimeActivity.

CREATING CRIMEPAGERACTIVITY

- CrimePagerActivity will be a sublclass of FragmentActivity. It will create and manage the ViewPager.
- Create a new class names CrimePagerActivity. Make its superclass FragmentActivity and set up the view of the activity.

```
public class CrimePagerActivity extends FragmentActivity {

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

- The layout file does not yet exist. Create a new layout in res/layout/ and name it activity_crime_pager. Make its root view a **ViewPager** and give its attributes as shown in Figure. Notice that you must use **ViewPager**'s full package name **(android.support.v4.view.ViewPager)**

(activity_crime_pager.xml)

android.support.v4.view.ViewPager

```
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/activity_crime_pager_view_pager"
android:layout_width="match_parent"
android:layout_height="match_parent"
```

- You use ViewPager's full package name when adding it to the layout file because the ViewPager class is from the support library. Unlike Fragment,ViewPager is only available in the support library, there is not a standard ViewPager class in a later SDK.

VIEWPAGER AND PAGERADAPTER

- A ViewPager is like a RecyclerView in some ways. A RecyclerView requires an Adapter to provide views. A ViewPager requires a PagerAdapter.
- However, the conversation between ViewPager and PagerAdapter is much more involved than the conversation between RecyclerView and Adapter. Luckily, you can use FragmentStatePagerAdapter, a subclass of PageAdapter, to take care of many of the details.
- FragmentStatePagerAdapter will boil down the conversation to two simple methods : getCount() and getItem (int). When your getItem(int) method is called for a position in your array of crimes, it will return a CrimeFragment configured to display the crime at position.
- In CrimePagerActivity, set the ViewPager's page adapter and implement its getCount() and getItem(int) methods.

```
public class CrimePagerActivity extends FragmentActivity {

    private ViewPager mViewPager;
    private List<Crime> mCrimes;

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

        mViewPager = (ViewPager) findViewById(R.id.activity_crime_pager_view_pager);

        mCrimes = CrimeLab.get(this).getCrimes();
        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {

            @Override
            public Fragment getItem(int position) {
                Crime crime = mCrimes.get(position);
                return CrimeFragment.newInstance(crime.getId());
            }

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

- Let's go through this code. After finding the **ViewPager** in the activity's view you get your data set from **CrimeLab** - the **List** of crimes. Next, you get the activity's instance of **FragmentManager**.
- Then you set the adapter to be unnamed instance of FragmentStatePageAdapter. Creating the FragmentStatePageAdapter requires the **FragmentManager**. Remember that **FragmentStatePagerAdapter** is your agent managing the conversation with ViewPager. For your agent to do its job with the fragments that getItem (int) returns, it needs to be be able to add them to your activity. That is why it needs your **FragmentManager**.
- (What exactly is your agent doing? The short story is that it is adding the fragments you return to your activity and helping ViewPager identify the fragment's views so that they can placed correctly. More details are in the section For More Curious at the end of this chapter).
- The pager adapter's two methods are straightforward. The **getCount()** method returns the number of items in the array list. The **getItem(int)** method is where the magic is happen. It fetches the **Crime** instance for the given position in the dataset. It then users that **Crime**'s ID to create and return a properly configured **CrimeFragment**.

INTEGRATING CRIMEPAGERACTIVITY

- Now, you can begin the process of decommissioning CrimeActivity and putting CrimePagerActivity in its place.
- First add a newIntent method to CrimePagerActivity along with an extra for the the crime ID.

```
public class CrimePagerActivity extends FragmentActivity {
    private static final String EXTRA_CRIME_ID =
            "com.bignerdranch.android.criminalintent.crime_id";

    private ViewPager mViewPager;
    private List<Crime> mCrimes;

    public static Intent newIntent(Context packageContext, UUID crimeId) {
        Intent intent = new Intent(packageContext, CrimePagerActivity.class);
        intent.putExtra(EXTRA_CRIME_ID, crimeId);
        return intent;
    }

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

        UUID crimeId = (UUID) getIntent()
                .getSerializableExtra(EXTRA_CRIME_ID);
        ...
    }
}
```

- Now, you want pressing a list item in CrimeListFragment to start an instance of CrimePagerActivity instead of CrimeActivity.
- Return to CrimeListFragment.java and modify CrimeHolder.OnClick(...) to start CrimePagerActivity.

```
private class CrimeHolder extends RecyclerView.ViewHolder
        implements View.OnClickListener {

    ...

    @Override
    public void onClick(View v) {
        Intent intent = CrimeActivity.newIntent(getActivity(), mCrime.getId());
        Intent intent = CrimePagerActivity.newIntent(getActivity(), mCrime.getId());
        startActivity(intent);
    }
}
```

- You also need to add CrimePagerActivity to the manifest so that the OS can start it. While you are in the manifest, remove CrimeActivity's declaration. To accomplish this, you can just rename the CrimeActivity to CrimePagerActivity in the manifest.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
  ...
  <application ...>
    ...
    <activity
      android:name=".CrimeActivity"
      android:name=".CrimePagerActivity"
      android:label="@string/app_name" >
    </activity>

    ...
  </application>

</manifest>
```

- Finally, to keep your project tidy, delete CrimeActivity.java from the project tool window.

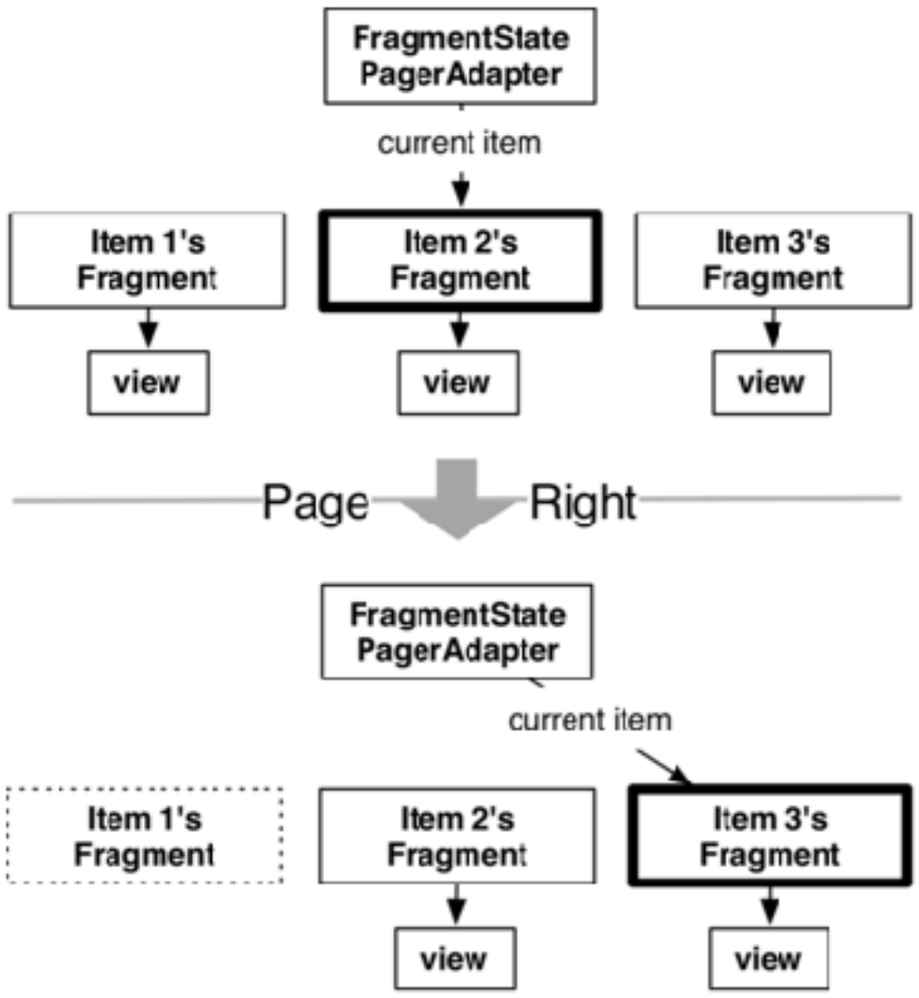
- Run CriminalIntent. Press Crime#0 to view its details. Then swipe left and right to browse more crimes. Notice that the paging is smooth and there is no delay in loading. By default, ViewPager loads the item currently on the screen plus one neighbouring page in each direction so that the response to a swipe is immediate.You can tweak how many neighbouring pages are loaded by loading **setOffscreenPageLimit(int)**.
- But all is not yet perfect with your pager. Press back to return to the list of crimes and press a different item. You will see the first crime displayed again instead of the crime that asked for.
- By default, the ViewPager shows the first item in its **PagerAdapter**. You can have it show the crime that was selected by setting the ViewPager's current item to the index of the selected item.
- At the end of **CrimePagerActivity.onCreate(...)**, find the index of crime to display by looping through and checking each crime's ID. When you find the **Crime** instance whose mID matches the crimeID in the intent extra, set the current item to the index of that **Crime**.

```
public class CrimePagerActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...

        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {
            ...
        });

        for (int i = 0; i < mCrimes.size(); i++) {
            if (mCrimes.get(i).getId().equals(crimeId)) {
                mViewPager.setCurrentItem(i);
                break;
            }
        }
    }
```

- Run CriminalIntent. Selecting any list item should display the details of the current Crime. And that is it. Your ViewPager is now fully armed and operational.

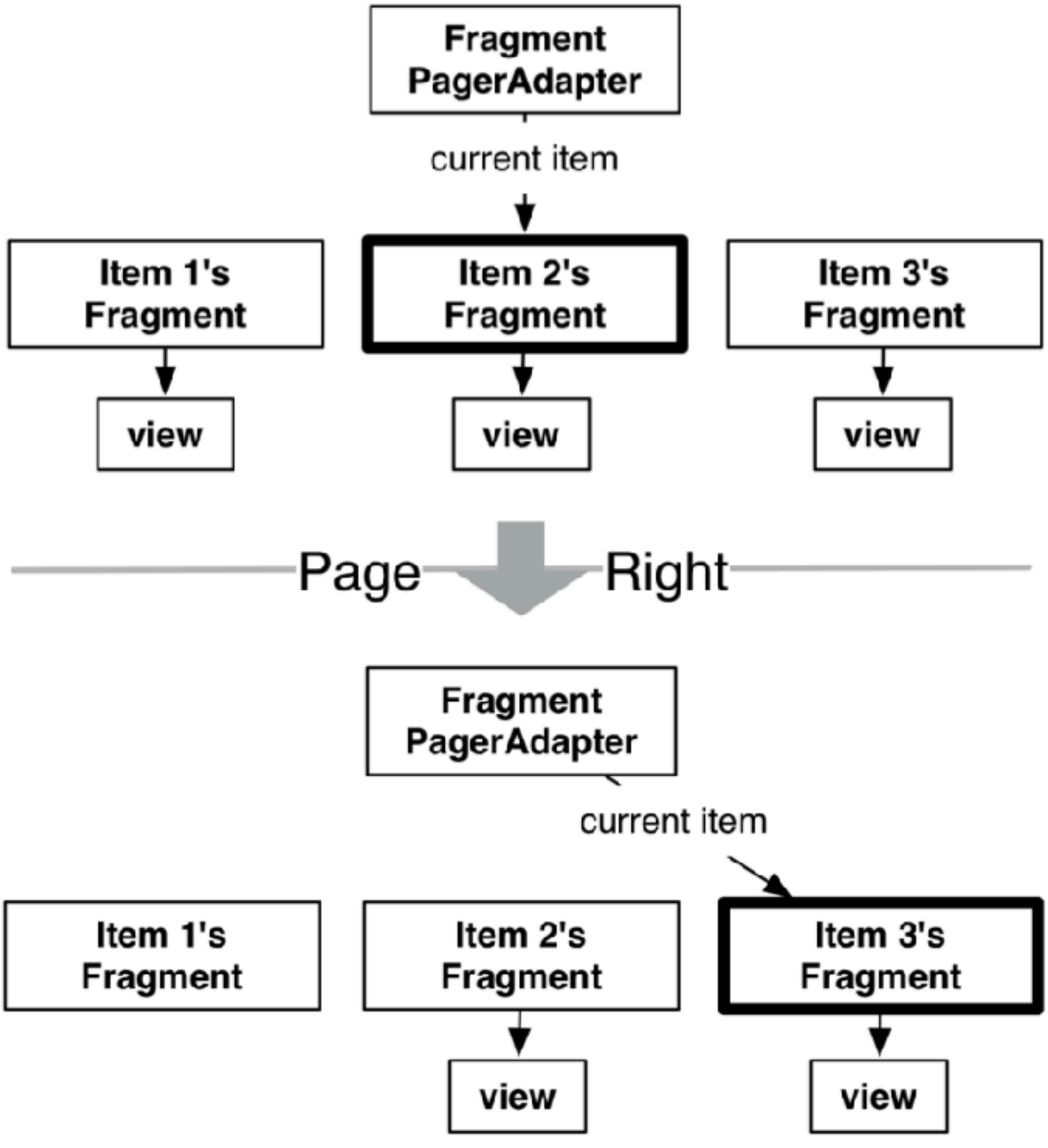
FRAGMENTSTATEPAGER VS FRAGMENTPAGERADAPTER

- There is another PagerAdapter type that you can use called FragmentPagerAdapter. FragmentPagerAdapter is used exactly like FragmentStatePagerAdapter. It only differs in how it unload your fragments when the are no longer needed.



- With **FragmentStatePagerAdapter**, your unneeded fragment is destroyed. A transaction is committed to completely remove the fragment from your activity's **FragmentManager**. The "state" in the **FragmentStatePagerAdapter** comes from the fact that it will save your fragment's **Bundle** from **onSaveInstanceState (Bundle)** when it is destroyed. When the user comes back, the new fragment will be restored using that instance state.

- FragmentPagerAdapter handles things differently. When your fragment is no longer needed, FragmentPagerAdapter calls detach(Fragment) on the transaction, instead of remove (Fragment). This destroy's the fragment's view, but leaves the fragment instance alive in the FragmentManager. So the fragments created by FragmentPagerAdapter are never destroyed as shown in next Figure.


Fragment PagerAdapter
current item
Item 1's Fragment
Item 2's Fragment
Item 3's Fragment
view
view
view
Page
Right
Fragment PagerAdapter
current item
Item 1's Fragment
Item 2's Fragment
Item 3's Fragment
view
view

- Which kind of adapter you want to use depends on your application. In general, FragmentStatePagerAdapter is more figural with memory. CriminalIntent is displaying what could be a long list of crimes, each of which will eventually include a photo. You want to keep all that information in memory, so you user FragmentStatePagerAdapter.
- On the other hand, if your interface has a small, fixed number of fragments, FragmentPagerAdapter is safe and appropriate. The most common example of this scenario is a tabbed interface. Some detail views have enough details to require two screens, so the details are split across multiple tabs. Adding a swappable ViewPager to this interface makes the app tactile. Keeping these fragments in memory can make your controller code easier to manage, because this style of interface usually has only two or three fragments per activity there is a little danger of running low on memory.