IT332: Mobile Application Development

Lecture # 16 : Using Dialogs & Fragments Communication

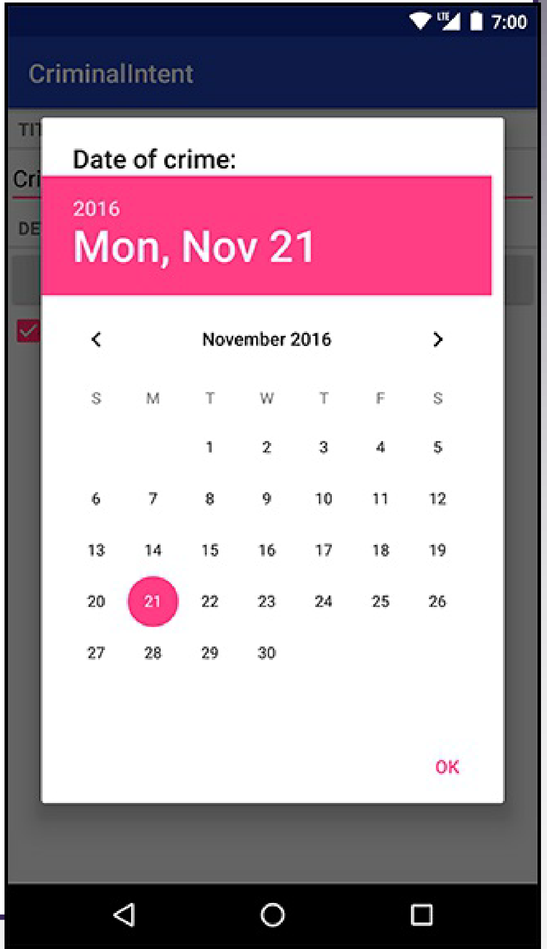
Muhammad Imran

Outline

- Final Objective Today
- Dialogs
- Creating a Dialog Fragment
- Showing a DialogFragment
- Setting a dialog’s contents
- Using DatePicker Widget
- Passing Data between Two Fragments
- Returning Data to a Fragment

Final Objective Today

- We will add a dialog in which users can change the date of a crime.
- Pressing the date button in CrimeFragment will present this dialog

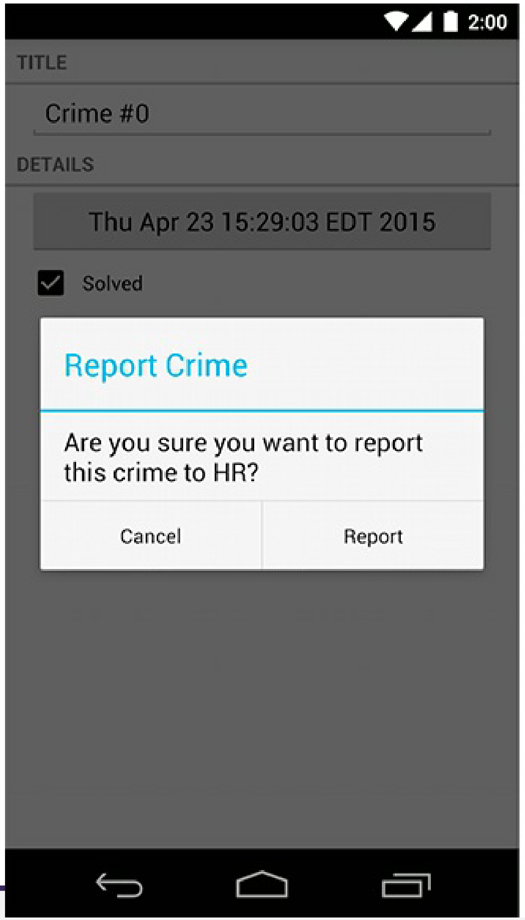


Dialogs

- Dialogs demand attention and input from the user. They are useful for presenting a **choice** or **important information**
- One possible dialog can be an instance of **AlertDialog**, a subclass of **Dialog**.
- AlertDialog is the **all-purpose** Dialog subclass that we use most often.
- When **Lollipop** was released, dialogs were given a **visual makeover**.

Old vs New

- AlertDialogs on Lollipop automatically use **new style.**
- On earlier versions of Android, AlertDialog will **fall back** to the **older style**, shown on the left


TITLE

Crime #0

DETAILS

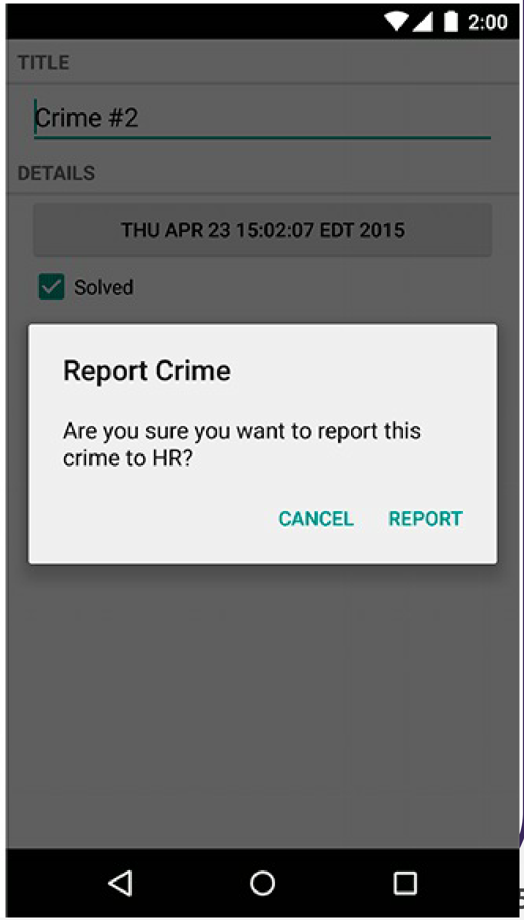
Thu Apr 23 15:29:03 EDT 2015

Solved

Report Crime

Are you sure you want to report this crime to HR?

Cancel Report


TITLE

Crime #2

DETAILS

THU APR 23 15:02:07 EDT 2015

Solved

Report Crime

Are you sure you want to report this crime to HR?

CANCEL REPORT

Old vs New

- Rather than displaying the crusty old dialog style, it would be nice to always show the **new dialog style**, no matter which version of Android the user's device is on.
- And you can do just that with the **AppCompat library's** AlertDialog class.
- This version of AlertDialog is very similar to the one included in the Android OS but, like other AppCompat classes, is **compatible** with earlier versions.
- To get the benefits of the AppCompat version, make sure you **import** android.support.v7.app.AlertDialog when prompted. (**or androidx**)

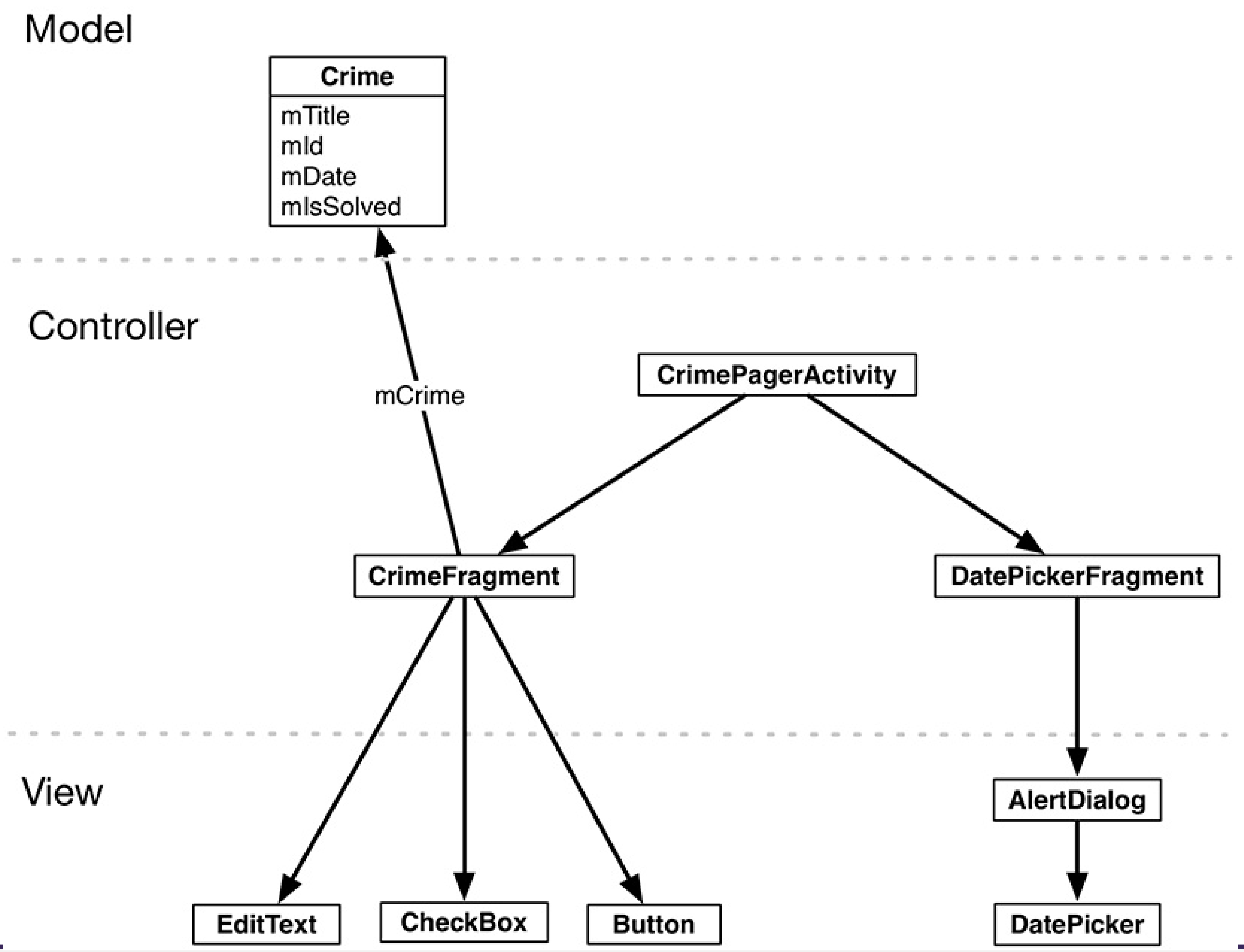
Creating a DialogFragment

- When using an **AlertDialog**, it is a good idea to **wrap** it in an instance of **DialogFragment**, a subclass of Fragment.
- It is possible to display an AlertDialog without a DialogFragment, but it is **not recommended**.
- Having the dialog managed by the FragmentManager gives you **more options** for presenting the dialog.
- In addition, a bare AlertDialog will **vanish** if the device is **rotated**.
- If the AlertDialog is **wrapped** in a fragment, then the dialog will be **re-created** and **put back** onscreen after rotation.

For the Crime Reporting Application

- For CrimeReporting App, we will create a DialogFragment subclass named **DatePickerFragment**.
- Within DatePickerFragment, we will **create and configure** an instance of AlertDialog that displays a DatePicker widget.
- DatePickerFragment will be **hosted by** CrimePagerActivity.

Object diagram for two fragments hosted by CrimePagerActivity



Tasks to be Completed

- creating the **DatePickerFragment** class
- building an **AlertDialog**
- getting the dialog **onscreen** via the FragmentManager
- wire up the DatePicker and pass the necessary data between **CrimeFragment** and **DatePickerFragment**.

Adding string for dialog title (values/strings.xml)

```
<resources>
    ...
    <string name="crime_solved_label">Solved</string>
    <string name="date_picker_title">Date of crime:</string>

</resources>
```

Creating a DialogFragment

- Create a new class named DatePickerFragment by extending the superclass DialogFragment.

- DialogFragment includes the following method:

```
public Dialog onCreateDialog(Bundle savedInstanceState)
```

- The **FragmentManager** of the hosting activity calls **onCreateDialog** as part of putting the DialogFragment onscreen.

Creating a DialogFragment

- The implementation of onCreateDialog(Bundle) builds an AlertDialog with a title and one OK button.
- We can use the AlertDialog.Builder class, which provides a fluent interface for constructing an AlertDialog instance

```
public class DatePickerFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
    }
}
```

Showing a DialogFragment

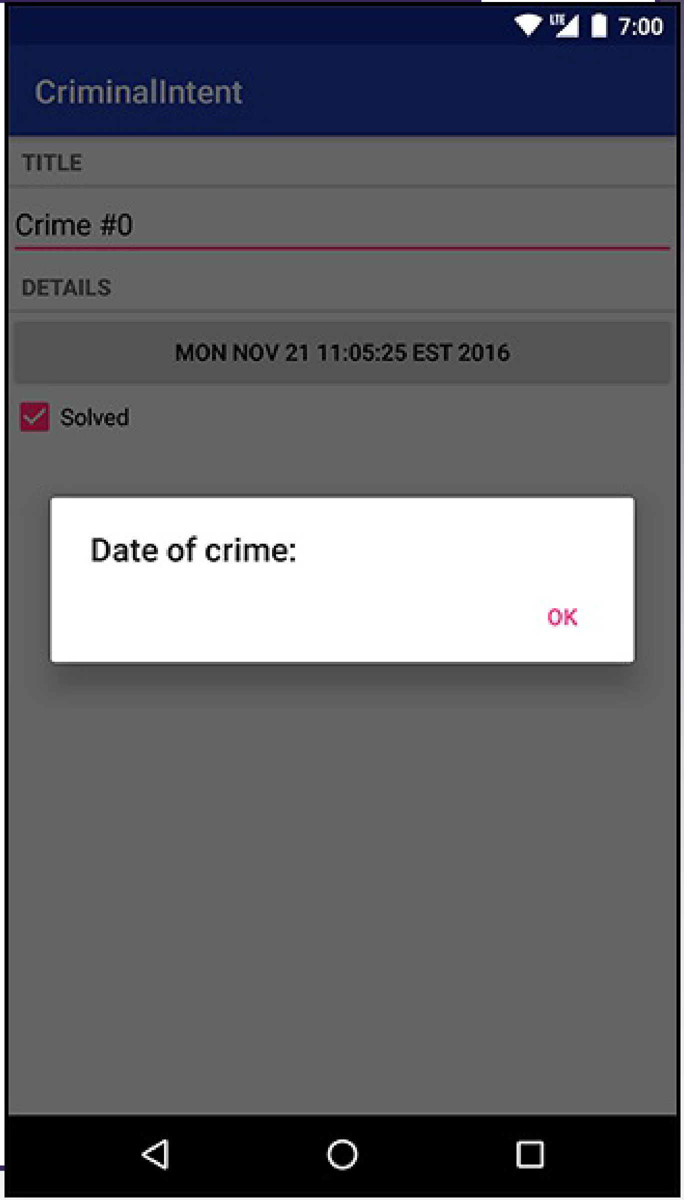
- Like all fragments, instances of DialogFragment are managed by the FragmentManager of the hosting activity.
- To get a DialogFragment **added** to the FragmentManager and put onscreen, following methods on the **fragment instance** can be called:

```
public void show(FragmentManager manager, String tag)
public void show(FragmentTransaction transaction, String tag)
```

- The **string** parameter uniquely **identifies** the DialogFragment in FragmentManager’s list.
- Use FragmentManager or FragmentTransaction is subjective.
- If we pass in a **FragmentTransaction**, we will manually create and commit that transaction.
- If we pass in a **FragmentManager**, a transaction will **automatically** be created and committed.

Showing a DialogFragment

```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        mDateButton = (Button) v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        ~~mDateButton.setEnabled(false);~~
        mDateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentManager manager = getFragmentManager();
                DatePickerFragment dialog = new DatePickerFragment();
                dialog.show(manager, DIALOG_DATE);
            }
        });

        mSolvedCheckBox = (CheckBox) v.findViewById(R.id.crime_solved);
        ...
        return v;
    }
}
```


7:00

CriminalIntent

TITLE

Crime #0

DETAILS

MON NOV 21 11:05:25 EST 2016

Solved

Date of crime:

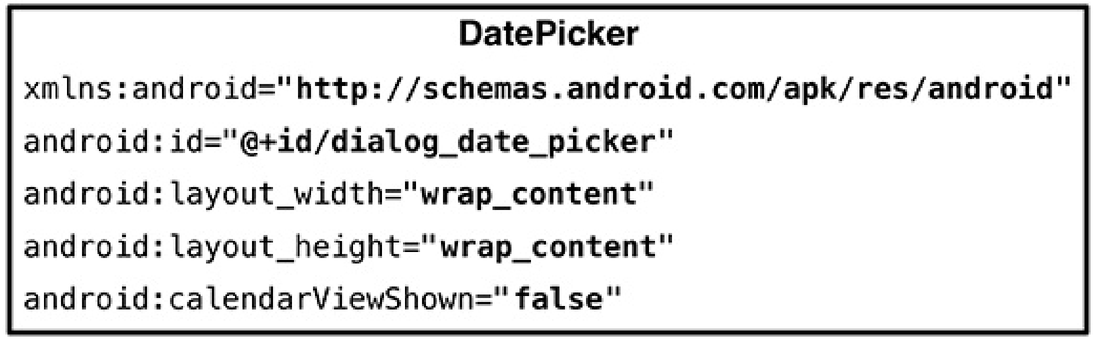
OK

Setting a dialog's contents

- We will be adding a DatePicker widget to AlertDialog using the following AlertDialog.Builder method:

```
public AlertDialog.Builder setView(View view)
```

- This method configures the dialog to display the passed-in View object between the dialog's title and its button(s).
- For using the DatePicker widget, we use a layout resource file and make its root element DatePicker.
- We use this layout consisting of a single View object – a DatePicker – and then inflate and pass into setView(…).



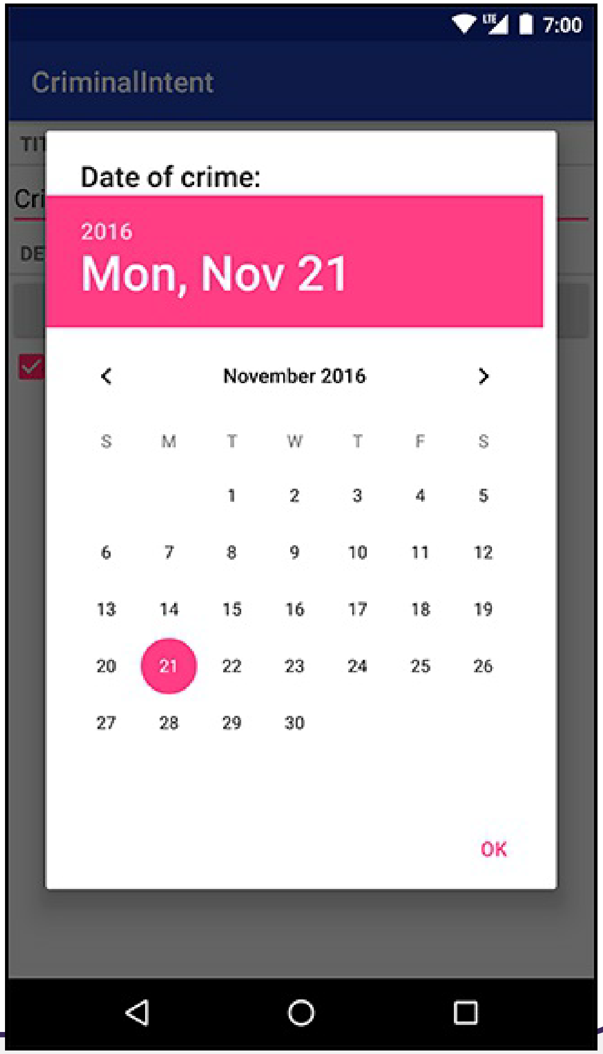
DatePicker

```
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/dialog_date_picker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:calendarViewShown="false"
```

Adding DatePicker to AlertDialog (DatePickerFragment.java)

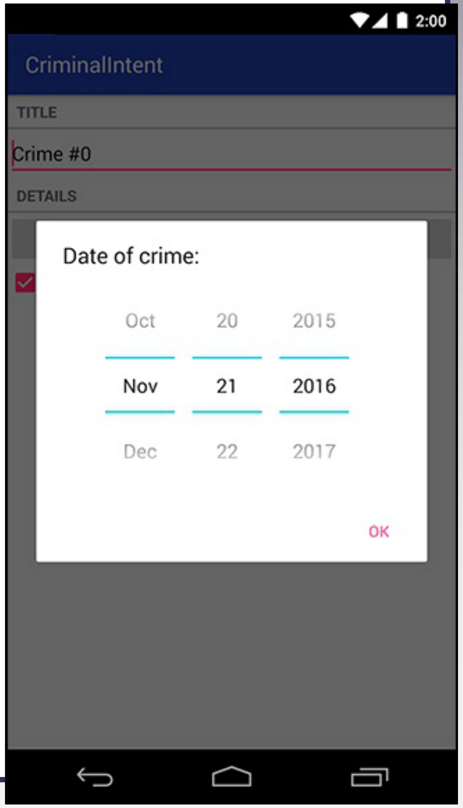
- In **DatePickerFragment.onCreateDialog**(Bundle), inflate this view and then set it on the dialog.

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    View v = LayoutInflater.from(getActivity())
        .inflate(R.layout.dialog_date, null);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}
```



calaendarViewShown attribute

- The calendar picker was introduced along with material design.
- This version of the DatePicker widget **ignores** the calendarViewShown attribute we set in your layout.
- In previous version of Android, however, we see the old spinner-based DatePicker version that respects that attribute



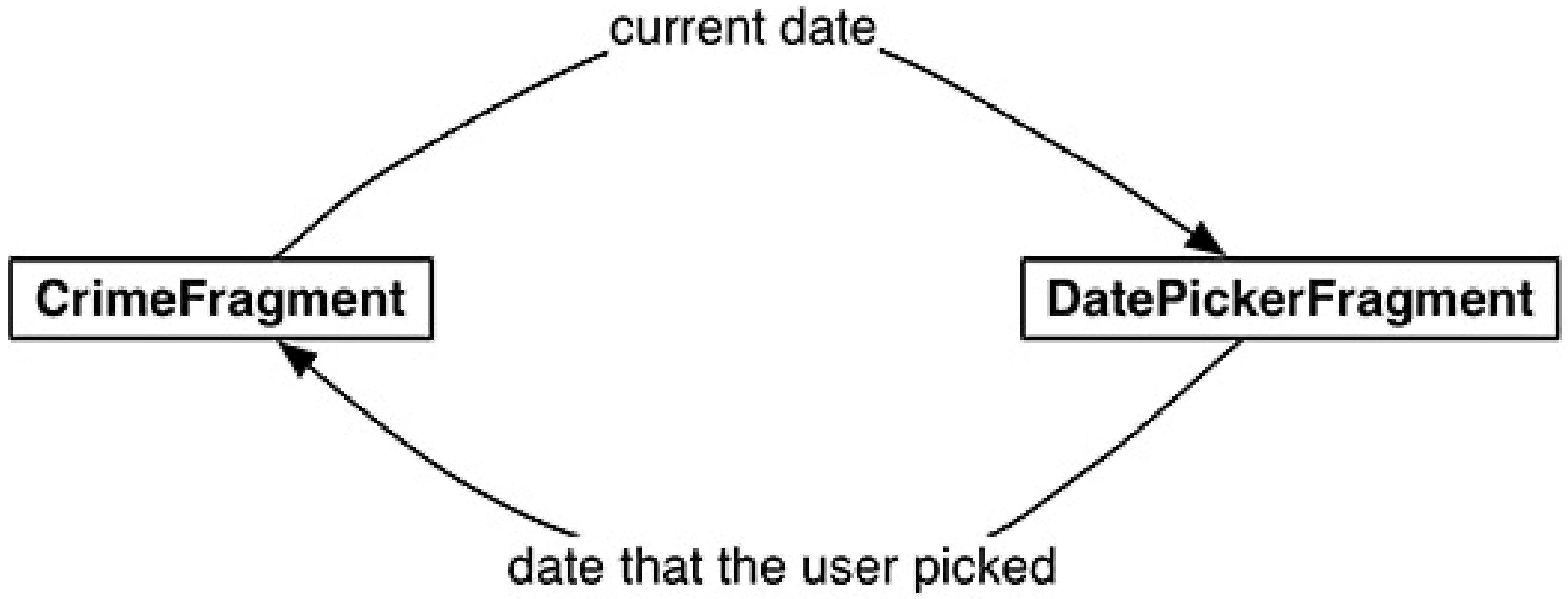
Why Use Layout for DatePicker Widget

- Instead of defining and inflating a layout, we can create the DatePicker object in code

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    DatePicker datePicker = new DatePicker(getActivity());

    return new AlertDialog.Builder(getActivity())
        .setView(datePicker)
        ...
        .create();
}
```

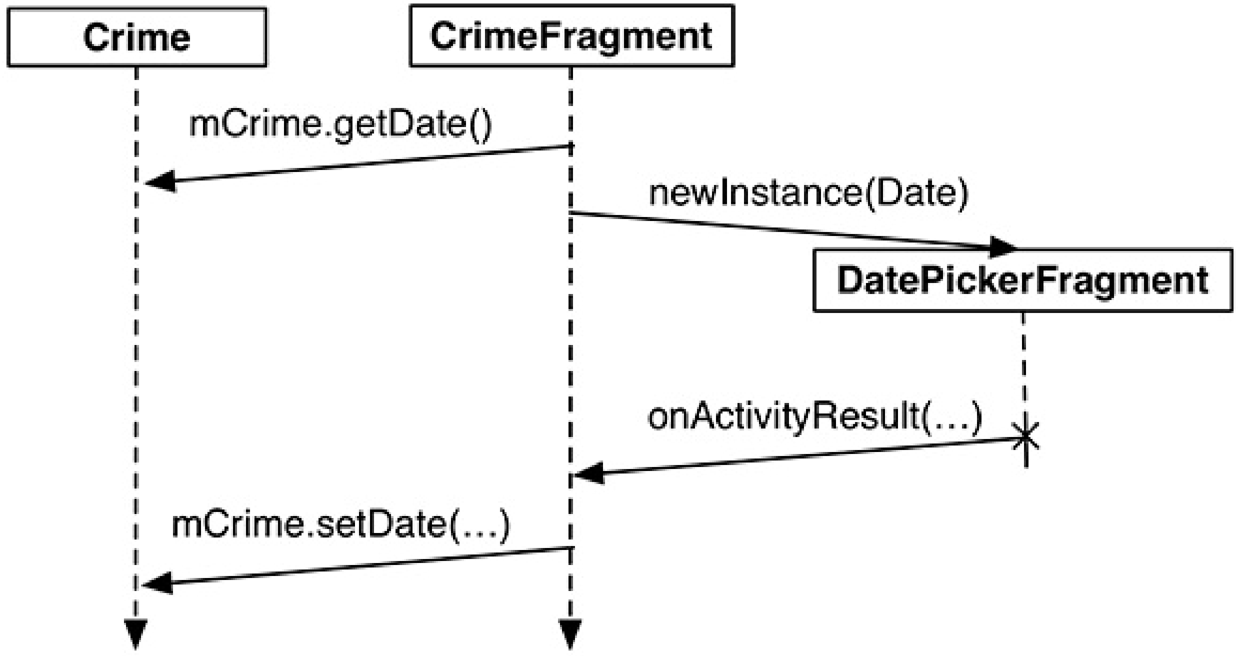
- **But** using a layout makes modifications easy. For example: if we want a TimePicker next to the DatePicker in this dialog?
- By defining and inflating a layout, we can simply update the layout file and the new view will appear.
- Also, the selected date in the DatePicker is automatically preserved across rotation because the Views can save state across configuration changes, but only if they have an ID attribute.

Conversation between CrimeFragment and DatePickerFragment



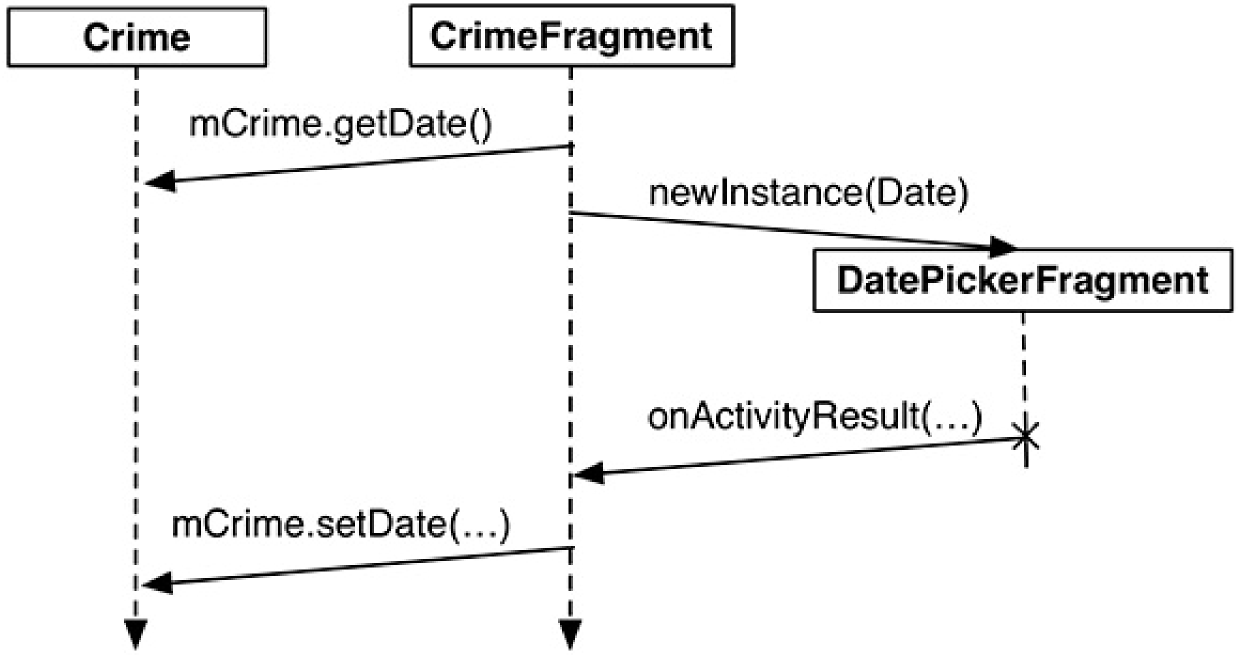
Passing Data Between Two Fragments

- To get the Crime's date to DatePickerFragment, we will write a newInstance(Date) method and make the Date an argument on the fragment.
- To get the new date back to the CrimeFragment so that it can update the model layer and its own view, we will package up the date as an extra on an Intent and pass this Intent in a call to CrimeFragment.onActivityResult(…)



Passing Data Between Two Fragments

- To get the **Crime's** date to **DatePickerFragment**, we will write a **newInstance(Date)** method and make the **Date** an argument on the fragment.
- To get the new date back to the **CrimeFragment** so that it can update the model layer and its own view, we will package up the date as an extra on an **Intent** and pass this **Intent** in a call to **CrimeFragment.onActivityResult(…)**



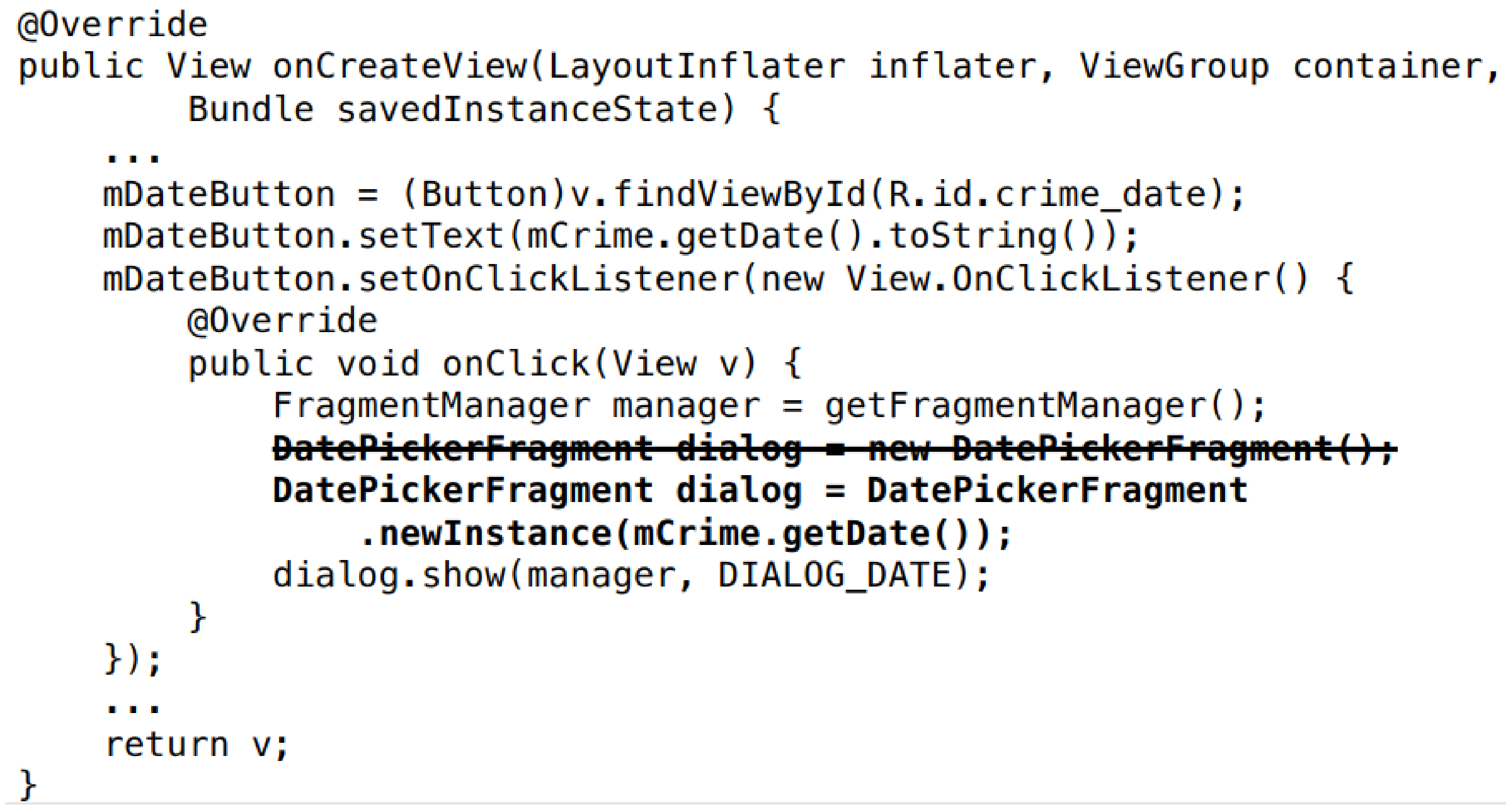
Passing data to DatePickerFragment

- To get data into DatePickerFragment, we supply the date in DatePickerFragment’s arguments bundle, where the DatePickerFragment can access it.
- Creating and setting fragment arguments is typically done in a **newInstance()** method that **replaces** the fragment **constructor**.

```
public class DatePickerFragment extends DialogFragment {

    private static final String ARG_DATE = "date";

    private DatePicker mDatePicker;

    public static DatePickerFragment newInstance(Date date) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_DATE, date);

        DatePickerFragment fragment = new DatePickerFragment();
        fragment.setArguments(args);
        return fragment;
    }
    ...
}
```

Passing data to DatePickerFragment

- In CrimeFragment, remove the call to the DatePickerFragment constructor and replace it with a call to **DatePickerFragment.newInstance(Date).**



```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    ...
    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setText(mCrime.getDate().toString());
    mDateButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            FragmentManager manager = getFragmentManager();
            ~~DatePickerFragment dialog = new DatePickerFragment();~~
            DatePickerFragment dialog = DatePickerFragment
                .newInstance(mCrime.getDate());
            dialog.show(manager, DIALOG_DATE);
        }
    });
    ...
    return v;
}
```

Extracting the date and initializing DatePicker (DatePickerFragment.java)

- DatePickerFragment needs to initialize the DatePicker using the information held in the Date.
- However, initializing the DatePicker requires integers for the month, day, and year.
- Date is more of a timestamp and cannot provide integers like this directly.
- To get the integers, we create a Calendar object and use the Date to configure the Calendar.
- Then we can retrieve the required information from the Calendar.
- In onCreateDialog(Bundle), get the Date from the arguments to initialize the DatePicker.

Extracting the date and initializing DatePicker (DatePickerFragment.java)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    Date date = (Date) getArguments().getSerializable(ARG_DATE);

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    View v = LayoutInflater.from(getActivity())
            .inflate(R.layout.dialog_date, null);

    mDatePicker = (DatePicker) v.findViewById(R.id.dialog_date_picker);
    mDatePicker.init(year, month, day, null);

    return new AlertDialog.Builder(getActivity())
            .setView(v)
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
}
```

- Now **CrimeFragment** is successfully telling **DatePickerFragment** what date to show.

Returning data to CrimeFragment

- To have **CrimeFragment** receive the date back from **DatePickerFragment**, you need a way to keep track of the relationship between the two fragments.
- With activities, you call **startActivityForResult(…),** and the **ActivityManager** keeps track of the parent-child activity relationship.
- When the child activity dies, the **ActivityManager** knows which activity should receive the result.

Setting a target fragment

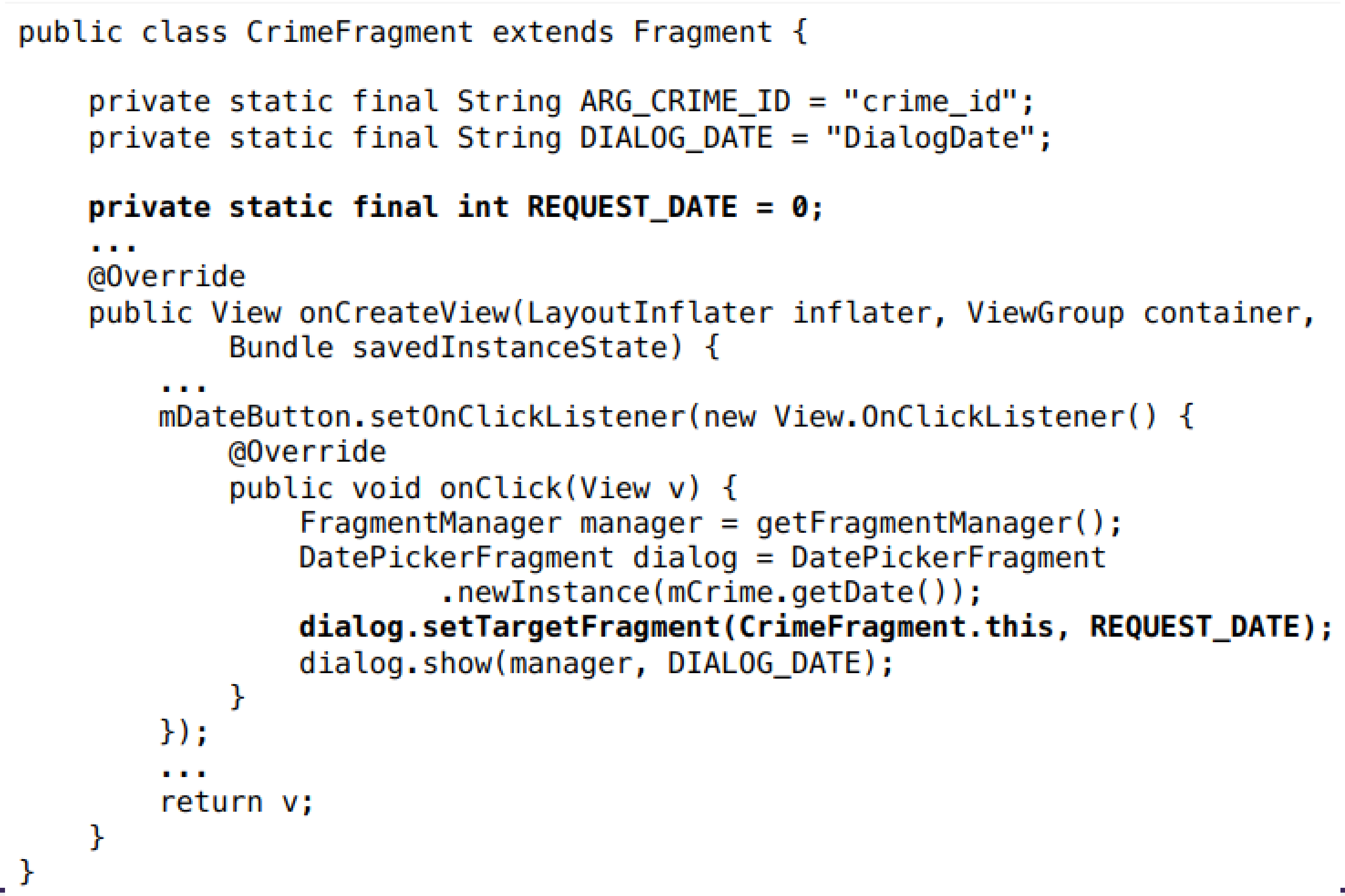
- You can create a similar connection by making CrimeFragment the target fragment of DatePickerFragment.
- This connection is automatically reestablished after both CrimeFragment and DatePickerFragment are destroyed and re-created by the OS.
- To create this relationship, you call the following Fragment method:

```
public void setTargetFragment(Fragment fragment, int requestCode)
```

- This method accepts the fragment that will be the target and a request code just like the one you send in startActivityForResult(…). The target fragment can use the request code later to identify which fragment is reporting back.
- The FragmentManager keeps track of the target fragment and request code. You can retrieve them by calling getTargetFragment() and getTargetRequestCode() on the fragment that has set the target.

Setting a target fragment

- In CrimeFragment.java, create a constant for the request code and then make CrimeFragment the target fragment of the DatePickerFragment instance.



```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";

    private static final int REQUEST_DATE = 0;
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
        mDateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentManager manager = getFragmentManager();
                DatePickerFragment dialog = DatePickerFragment
                        .newInstance(mCrime.getDate());
                dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);
                dialog.show(manager, DIALOG_DATE);
            }
        });
        ...
        return v;
    }
}
```

Sending data to the target fragment

- When we have **connection** between two fragments (CrimeFragment and DatePickerFragment), we need to send the **data back** to **Target Fragment** (CrimeFragment)
- We can put the respective data on an **Intent** as an extra.
- We will have **DatePickerFragment** pass it into **CrimeFragment.onActivityResult(int, int, Intent)**.
- Activity.onActivityResult(…) is the method that the **ActivityManager** calls on the parent activity after the **child activity dies**.
- When dealing with activities, we **do not** call Activity.onActivityResult(…) **ourself**; that is the ActivityManager’s job.
- After the activity has received the call, the activity’s FragmentManager then calls Fragment.onActivityResult(…) on the appropriate fragment.

Sending data to the target fragment

- When dealing with two fragments hosted by the same activity, we can borrow Fragment.onActivityResult(…) and call it directly on the target fragment to pass back data.
- It has provides us the following:
  - a **request code** that matches the code passed into setTargetFragment(…) to tell the target what is returning the result
  - a **result code** to determine what action to take
  - an **Intent** that can have **extra data**

Calling back to your target (DatePickerFragment.java)

- In **DatePickerFragment**, we will create a private method that creates an intent, puts the date on it as an extra, and then calls CrimeFragment.onActivityResult(…).

```
public class DatePickerFragment extends DialogFragment {

    public static final String EXTRA_DATE =
            "com.bignerdranch.android.criminalintent.date";

    private static final String ARG_DATE = "date";
    ...
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        ...
    }

    private void sendResult(int resultCode, Date date) {
        if (getTargetFragment() == null) {
            return;
        }

        Intent intent = new Intent();
        intent.putExtra(EXTRA_DATE, date);

        getTargetFragment()
                .onActivityResult(getTargetRequestCode(), resultCode, intent);
    }
}
```

Calling back to your target (DatePickerFragment.java)

- When the user presses the positive button in the dialog, you want to retrieve the date from the **DatePicker** and send the result back to **CrimeFragment** by retrieving the selected date and calling **sendResult(…)**.

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    ...
    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        ~~.setPositiveButton(android.R.string.ok, null);~~
        .setPositiveButton(android.R.string.ok,
                new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        int year = mDatePicker.getYear();
                        int month = mDatePicker.getMonth();
                        int day = mDatePicker.getDayOfMonth();
                        Date date = new GregorianCalendar(year, month, day).getTime();
                        sendResult(Activity.RESULT_OK, date);
                    }
        })
        .create();
}
```

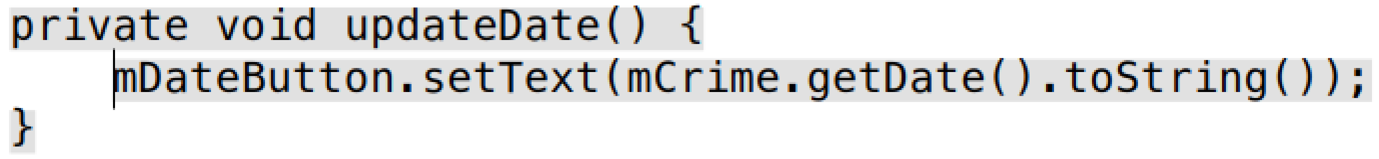
Responding to the dialog (CrimeFragment.java)

- When the user presses the positive button in the dialog, you want to retrieve the date from the **DatePicker** and send the result back to **CrimeFragment** by retrieving the selected date and calling **sendResult(…)**.

```
public class CrimeFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }

        if (requestCode == REQUEST_DATE) {
            Date date = (Date) data
                    .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            mDateButton.setText(mCrime.getDate().toString());
        }
    }
}
```

Responding to the dialog (CrimeFragment.java)

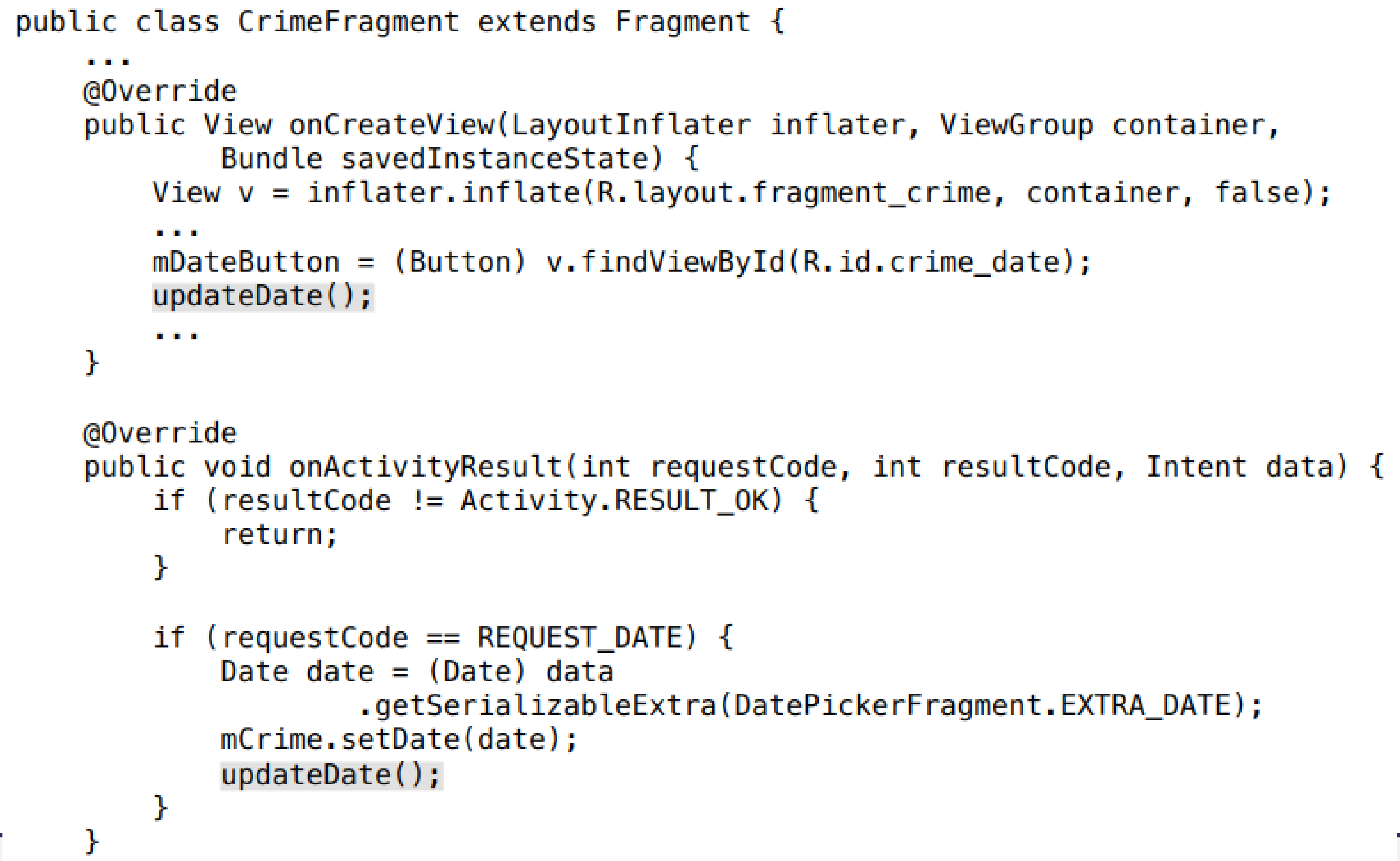
- To avoid setting the text in multiple places, encapsulate repective code in a private updateDate() method and then call it in onCreateView(…) and onActivityResult(…).



```
private void updateDate() {
    mDateButton.setText(mCrime.getDate().toString());
}
```

Responding to the dialog (CrimeFragment.java)

- The update code may look like this



```
public class CrimeFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        ...
        mDateButton = (Button) v.findViewById(R.id.crime_date);
        updateDate();
        ...
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }

        if (requestCode == REQUEST_DATE) {
            Date date = (Date) data
                    .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
            mCrime.setDate(date);
            updateDate();
        }
    }
```

InClass Task 10 (More Dialogs)

- Write another dialog fragment named TimePickerFragment that allows the user to select what time of day the crime occurred using a TimePicker widget.
- Add another button to CrimeFragment to display a TimePickerFragment.

InClass Task 11 (A Responsive DialogFragment)

- For a more involved challenge, modify the presentation of the DatePickerFragment.
- The first stage of this challenge is to supply the DatePickerFragment's view by overriding onCreateView(…) instead of onCreateDialog(Bundle). When setting up a DialogFragment in this way, your dialog will not be presented with the built-in title area and button area on the top and bottom of the dialog. You will need to create your own OK button in dialog_date.xml.
- Once DatePickerFragment's view is created in onCreateView(…), you can present DatePickerFragment as a dialog or embedded in an activity. For the second stage of this challenge, create a new subclass of SingleFragmentActivity and host DatePickerFragment in that activity.
- When presenting DatePickerFragment in this way, you will use the startActivityForResult(…) mechanism to pass the date back to CrimeFragment. In DatePickerFragment, if the target fragment does not exist, use the setResult(int, intent) method on the hosting activity to send the date back to the fragment.
- For the final step of this challenge, modify CriminalIntent to present the DatePickerFragment as a full-screen activity when running on a phone. When running on a tablet, present the DatePickerFragment as a dialog.

Recommended Readings

- Page # 227 to 245, Chapter 12: Dialogs from Android Programming: The Big Nerd Ranch Guide, 3rd Edition by Bill Phillips, 2017