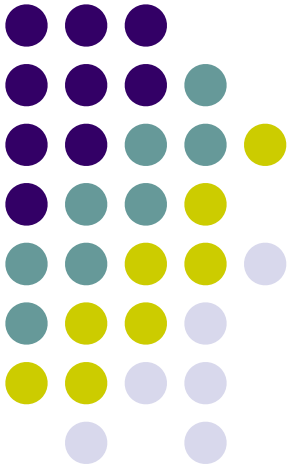
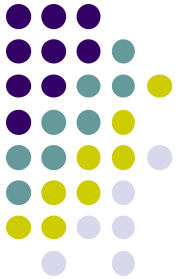
Android Programming

04 – User Interfaces

Part - 3

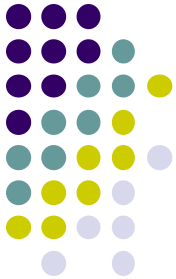


Dialogs



- A dialog is usually a small window that appears in front of the current Activity.
- Underlying activity looses focus when displayed
- Types:
  - AlertDialog
  - ProgressDialog
  - DatePickerDialog
  - TimePickerDialog

Dialogs (cont'd) - Deprecated



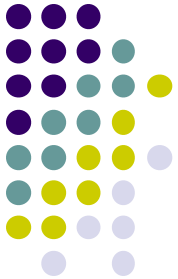
- To attach a dialog to an activity override onCreateDialog(int) method in the activity or call setOwnerActivity(Activity) to set the parent activity

```
static final int DIALOG_PAUSED_ID = 0;
static final int DIALOG_GAMEOVER_ID = 1;
```

```
protected Dialog onCreateDialog(int id) {
    Dialog dialog;
    switch(id) {
    case DIALOG_PAUSED_ID:
        // do the work to define the pause Dialog
        break;
    case DIALOG_GAMEOVER_ID:
        // do the work to define the game over Dialog
        break;
    default:
        dialog = null;
    }
    return dialog;
}
```

```
showDialog(DIALOG_PAUSED_ID);
```

ProgressDialog

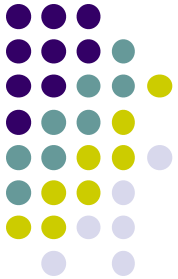


- Simple to create, has to versions: Wait style and progress style

```
ProgressDialog dialog = ProgressDialog.show(MyActivity.this, "",
                        "Loading. Please wait...", true);
```

```
ProgressDialog progressDialog;
progressDialog = new ProgressDialog(mContext);
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
progressDialog.setMessage("Loading...");
progressDialog.setCancelable(false);
```

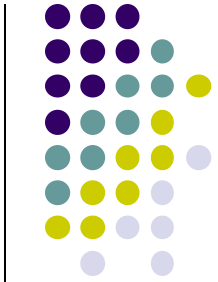
AlertDialog



- Use AlertDialogBuilder to get an instance int the onCreateDialog() method
- Use it in DialogFragment implementation class

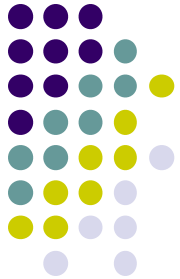
```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
       .setCancelable(false)
       .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                MyActivity.this.finish();
           }
       })
       .setNegativeButton("No", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                dialog.cancel();
           }
       });
AlertDialog alert = builder.create();
```

DialogFragment



- As of API level 11 DialogFragment must be used as a container for your dialog
- Ensures that it correctly handles lifecycle events such as when the user presses the Back button or rotates the screen
- Allows you to reuse the dialog's UI as an embeddable component in a larger UI, just like a traditional Fragment
- Import android.support.v4.app.DialogFragment

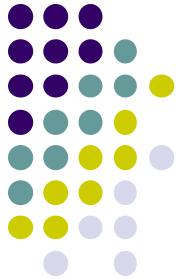
Using DialogFragment



- Implement the DialogFragment class and override onCreateDialog() which will return the desired dialog

```
public class MyDialogFragment extends DialogFragment{

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
      AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
      builder.setTitle("INFO")
            .setMessage("Are you sure?")
            .setPositiveButton("YES", new DialogInterface.OnClickListener()
            @Override
            public void onClick(DialogInterface dialog, int which) {
                  Toast.makeText(getActivity(), "YES", Toast.LENGTH_SHORT).show();
                  dismiss();

                  }
            }).setNegativeButton("No", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                  Toast.makeText(getActivity(), "NO", Toast.LENGTH_SHORT).show();
                  dismiss();
            }
            });
            return builder.create();
      }
}
```

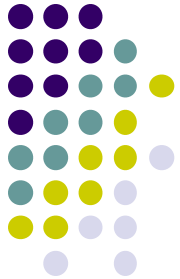
Using the DialogFragment (Cont'd)



- Extend your activity from FragmentActivity and get use of getSupportFragmentManager() method
- In the activity:

```
public void showtestdialog(View v){
    FragmentManager fm = getSupportFragmentManager();
    dialogFragment = new MyDialogFragment();
    dialogFragment.show(fm, "dial");
}
```

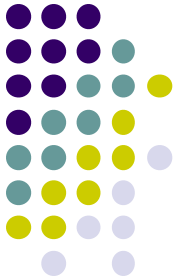
List Of Choices in Dialog



- setItems() used to add a list of choices to the dialog

```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        Toast.makeText(getApplicationContext(), items[item],
Toast.LENGTH_SHORT).show();
    }
});
AlertDialog alert = builder.create();
```

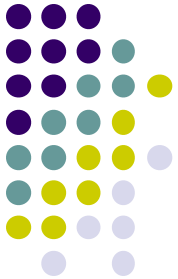
Multi-select in Dialogs



- use setMultiChoiceItems() to set items as an array and implement DialogInterface.OnMultiChoiceClickListener

```
builder.setTitle("Please select")
      .setMultiChoiceItems(values, null, new DialogInterface.OnMultiChoiceClickListener() {

      @Override
      public void onClick(DialogInterface dialog, int which, boolean isChecked) {
            if (isChecked) {
                        // If the user checked the item, add it to the selected items
            selectedItems.add(values[which]);
                  } else if (selectedItems.contains(which)) {
                        // Else, if the item is already in the array, remove it
                  selectedItems.remove(Integer.valueOf(which));
                  }

                  }
            }).setPositiveButton("OK", new DialogInterface.OnClickListener() {

            @Override
            public void onClick(DialogInterface dialog, int which) {
                  dismiss();
                  listener.onDialogDismissed(selectedItems);

                  }
            });
```

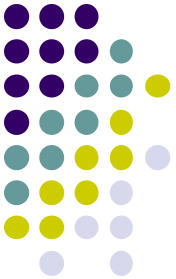
Passing Events Back to the Dialog's Host



- Create an interface in the DialogFragment, and implement the interface in the caller activity

```
public class NoticeDialogFragment extends DialogFragment {

    /* The activity that creates an instance of this dialog fragment must
     * implement this interface in order to receive event callbacks.
     * Each method passes the DialogFragment in case the host needs to query it. */
    public interface NoticeDialogListener {
        public void onDialogPositiveClick(DialogFragment dialog);
        public void onDialogNegativeClick(DialogFragment dialog);
    }

    // Use this instance of the interface to deliver action events
    NoticeDialogListener mListener;

    // Override the Fragment.onAttach() method to instantiate the NoticeDialogListener
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        // Verify that the host activity implements the callback interface
        try {
            // Instantiate the NoticeDialogListener so we can send events to the host
            mListener = (NoticeDialogListener) activity;
        } catch (ClassCastException e) {
            // The activity doesn't implement the interface, throw exception
            throw new ClassCastException(activity.toString()
                    + " must implement NoticeDialogListener");
        }
    }
    ...
}
```

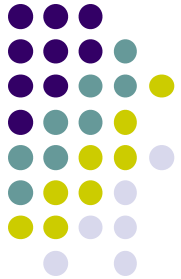
Creating Custom Dialogs



- Create an XML layout for your dialog
- Override onCreateView() in DialogFragment subclass

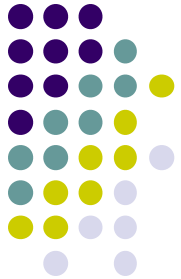
```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
           Bundle savedInstanceState) {

      View v=  inflater.inflate(R.layout.dialog_custom, container);
      txtName = (EditText) v.findViewById(R.id.txt_your_name);
      getDialog().setTitle("Enter your name:");
      btnOk = (Button)v.findViewById(R.id.btdialogok);
      btnOk.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View v) {
       EditCompleteListener activity =
            (EditCompleteListener) getActivity();
              activity.onEditComplete(txtName.getText().toString());
                  MyCustomDialogFragment.this.dismiss();
            }
      });
      return v;

}
@Override
public void onAttach(Activity activity) {
      listener = (EditCompleteListener)activity;
      super.onAttach(activity);
}
```

Binding Data with AdapterView



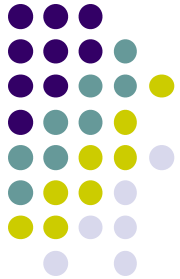
- The AdapterView is a ViewGroup subclass whose child Views are determined by an Adapter that binds to data of some type
- Some subclasses: Gallery, ListView, Spinner
- Responsible for:
  - Filling the layout with data
  - Handling user selections

Filling the Layout with Data



- Inserting data into the layout is typically done by binding the AdapterView class to an Adapter, which retrieves data from an external source
  - (perhaps a list that the code supplies or query results from the device's database).

Filling the Layout with Data (cont'd)



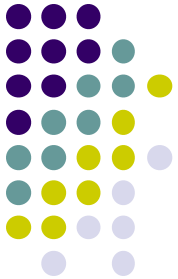
Creating a spinner with array data

```
 spplanets = (Spinner)findViewById(R.id.spplanets);
 ArrayAdapter<CharSequence> spinnerAdapter = ArrayAdapter.createFromResource(this,
R.array.planets_array, android.R.layout.simple_spinner_dropdown_item);

 spplanets.setAdapter(spinnerAdapter);
```

Handling User Selection

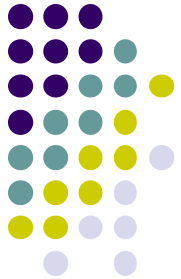
```
spplanets.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {

        @Override
        public void onItemSelected(AdapterView<?> adp, View v, int pos,
                  long id) {
        txtOUt.setText(spplanets.getAdapter().getItem(pos).toString());
        }
        @Override
        public void onNothingSelected(AdapterView<?> arg0) {
             // TODO Auto-generated method stub
        }
    });
```

ListView



- Android provides the view "ListView" which is capable of displaying a scrollable list of items
- "ListView" gets the data to display via an adapter. An adapter which must extend "BaseAdapter" and is responsible for providing the data model for the list and for converting the data into the fields of the list.

ListActivity



- In case your Activity is primary showing a list you can extend the activity "ListActivity" which simplifies the handling of a "ListView".
- ListActivity extends Activity
- onListItemClick() → Method to control item click event

Simple ListActivity Example

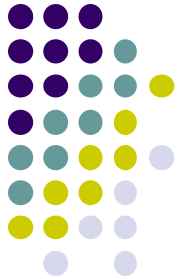
```
public class MainActivity extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        String[] cities = getResources().getStringArray(R.array.cities_array);

        ArrayAdapter<String> adp_cities = new
ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,cities);
        setListAdapter(adp_cities);
    }
    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {
     Toast.makeText(this, l.getAdapter().getItem(position).toString(),
Toast.LENGTH_SHORT).show();
    }
}
```

For single selection list → android.R.layout.simple_list_item_single_choice

For multi selection list → android.R.layout.simple_list_item_multiple_choice

To determine checked items → getCheckedItemPositions() of ListView

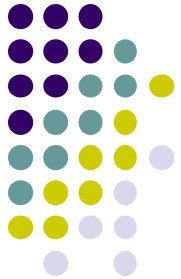
Layout for ListActivity



- ListActivity does not require that you assign a layout to it via the setContentView() method
- In case you need to include more Views than a ListView in your ListActivity you can still assign a layout to your activity.
- In this case your layout must contain a ListView with the android:id attribute set to @android:id/list.

```
<ListView
  android:id="@android:id/list"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" >
</ListView>
```

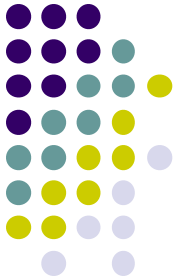
Custom Layout For Lists



- You can also define your own layout for the rows and assign this layout to your row adapter.
- This way only dynamic part will be the item label

```
ArrayAdapter<String> myAdapter = new
ArrayAdapter<String>(LayoutListActivity.this,R.layout.layoutlistrow,R.id.label,names);
this.setListAdapter(myAdapter);
```

Determining the Rows



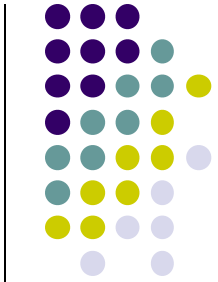
- We can define a layout for each row in a ListView using a layout resource

The Layout File

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <ImageView android:src="@drawable/ic_launcher" android:layout_width="30dip"
android:layout_height="wrap_content"
        android:id="@+id/icon" android:padding="3dip" android:contentDescription="img"/>
    <TextView android:id="@+id/label" android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
</LinearLayout>
```

Activity Code

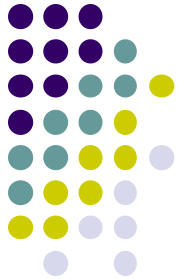
```
@Override
protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     String[] cities = getResources().getStringArray(R.array.cities_array);
     ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
R.layout.listwithrowactivity,R.id.label,cities);
     setListAdapter(adapter);
     getListView().setTextFilterEnabled(true);
     }
```



Complete Custom Lists

- In order to create complete custom Lists you should extend the BaseAdapter or ArrayAdapter class to add extra dynamic fields to your list
- getView() method should be overridden in order to fill rows from dynamic data and layout
- must check whether a row is already drawn or not using convertView
- Cache the views in a row using the holder pattern

Custom List Example

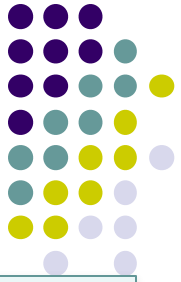


- Suppose we are listing user data consisting of user names and rating. We want to display different icons according to the rating of the user

Holder Class (example cont'd)

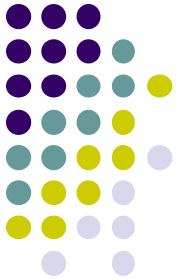
```
class ViewHolder {
      private View base; private ImageView icon; private TextView label; private TextView rating;
            public ViewHolder(View base) {
                  this.base = base;
            }
            public ImageView getIcon() {
                  if(icon==null){
                        icon = (ImageView)base.findViewById(R.id.icon);
                  }

                  return icon;
            }
            public void setIcon(ImageView icon) {
                  this.icon = icon;
            }
            public TextView getLabel() {
                  if(label==null){
                        label = (TextView)base.findViewById(R.id.label);
                  }
                  return label;
            }
            public void setLabel(TextView label) {
                  this.label = label;
            }
            public TextView getRating() {
                  if(rating==null){
                        rating = (TextView)base.findViewById(R.id.rating);
                  }
                  return rating;
            }
            public void setRating(TextView rating) {
                  this.rating = rating;
            }

      }
```

Custom Adapter Using the Holder (example cont'd)



```
public class CustomListAdapter extends ArrayAdapter<User>{
      public CustomListAdapter(Context context) {
            super(context, R.layout.customlistactivityrow, User.getAllUsers());
      }
      @Override
      public View getView(int position, View convertView, ViewGroup parent) {

            View row = convertView;
            ViewHolder holder = null;

            if(row ==null){
                  LayoutInflater inflater = ((ListActivity)getContext()).getLayoutInflater();
                  row = inflater.inflate(R.layout.customlistactivityrow, parent,false);
                  holder = new ViewHolder(row);
                  row.setTag(holder);

            }else{
                  holder = (ViewHolder)row.getTag();
            }

            holder.getLabel().setText(getItem(position).getName());
            int rating = getItem(position).getRating();
            holder.getRating().setText(String.valueOf(rating));

            if(rating<3){
                  holder.getIcon().setImageResource(R.drawable.negative);

            }else if(rating==3){
                  holder.getIcon().setImageResource(R.drawable.neutral);
            }else{
                  holder.getIcon().setImageResource(R.drawable.positive);
            }

            return row;
      }
      }
```

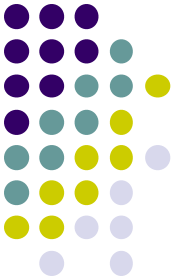
More UI: ScrollView



- Used to make screen scrollable
- Should be the root container

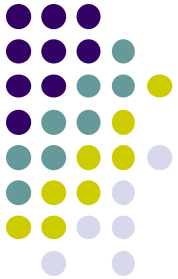
```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"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

    <!- View ITEMS -->

    </LinearLayout>
</ScrollView>
```

More UI: DatePickerDialog and TimePickerDialog



- Used for selecting date and time in dialog style
- Callbacks managed by
  - DatePickerDialog.OnDateSetListener
  - TimePickerDialog.OnTimeSetListener
- Helper classes:
  - Calender (abstract)
  - DateFormat (abstract)
- For displaying a clock use: AnalogClock or DigitalClock

Flipping with Animation



- ViewFlipper container used like TabHost
- It has FrameLayout by default

```
Layout:
<ViewFlipper …>
     View Items to overlap go here!
</ViewFlipper>
```

Setting Animation of Flip

```
flipper.setInAnimation(AnimationUtils.loadAnimation(this,
android.R.anim.slide_out_right));
```

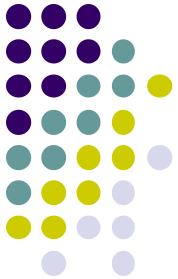
Actual Flipping

```
flipper.showNext();
flipper.showPrevious();
```

Automatic Flipping

```
flipper.setFlipInterval(2000);
flipper.startFlipping();
```

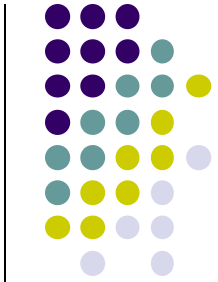
Sliding Drawer



- Should contain
  - A handle, frequently an ImageView
  - The contents of the drawer itself, usually some sort of container

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <SlidingDrawer android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:handle="@+id/handle" android:content="@+id/content">
        <ImageView android:id="@id/handle" android:layout_width="wrap_content"
android:layout_height="wrap_content" android:src="@drawable/ic_launcher"/>
        <Button android:id="@id/content" android:layout_width="fill_parent"
android:layout_height="fill_parent" android:text="I am visible"/>

    </SlidingDrawer>
</FrameLayout>
```

Examples



- Simple Notes app (later to use with SQLLite)