Emerging Technologies: Mobile Development for Android Devices

Custom Views

Introduction

- Why custom views?
- Approaches to creating custom views.
- Subclassing an existing view.
- Compose a view from several existing views.
- Create a full custom implementation.
- Constructors used in a full custom implementation.
- The init( ) method.
- The onDraw( ) method.
- The matrix stack.
- Managing the view.
- User interaction.
- Integration into an application.

Why Custom Views?

- The Android toolkit contains a large range of ready made components.
- However, it will not always contain every component that a developer might require.
- If the toolkit tried to provide too many variations of UI components, then the toolkit would be unreasonably large.
- In addition, some types of applications require components that are very specialised and would be unlikely to be anticipated by a UI toolkit.
- For example game apps etc.
- However, developers may also create their own custom components.

Approaches to Creating Custom Views

- There are 3 approaches:
- Subclass an existing view (rare).
- Compose a view out of several existing views (rare).
- Create a full custom implementation (most likely).

Subclassing an Existing View

- This is the best approach if only small changes are required to an already existing view.
- The behaviour required can not be added by changing attributes.
- So subclass to create a new component.
- Example, extend the NumberPicker class to add a maximum and minimum value.
- Often requires very little code and reasonably easy to do if it satisfies the requirements for the component.

Compose a View from Several Existing Views

- Combine multiple existing views.
- For example, create an EditText that also has a clear button.
- When using this approach, the layout must be subclassed rather than the view.
- Rarely used but little code required and can be easy to do if it satisfies requirements for a specialised component.

Full Custom Implementation

- This is the most frequently used approach.
- Take a bare view class and implement it.
- The view needs to scale to different devices and screen sizes. These can not be fixed.
- Requires engagement with mathematics.
- Some trial and error/experimentation.
- Verify the view visually to ensure correctness.

Full Custom Implementation

- There are a number of tasks that must be undertaken by the developer.
- Provide all drawing operations in onDraw( )
- Handle user touches.
- Provide a set of attributes so that the view can be further adjusted without direct access to the source code.

Constructors

- 3 Constructors to be implemented:

- CustomView(Context c)
- CustomView(Context c, AttributeSet as)
- CustomView(Context c, AttributeSet as, int def_style)

Constructors

3 Constructors to be implemented:

- **CustomView(Context c)**

Used when the view is directly initialised from Java code.

- **CustomView(Context c, AttributeSet as)**

Allows a list of xml attributes to be applied to the view.

- **CustomView(Context c, AttributeSet as, int def_style)**

Supports a list of attributes as well as a style to be applied to the application.

The init( ) Method

- Can be called by any of the three constructors.
- Colours, shapes, data structures and other parameters required by the custom view.
- Try to initialise objects in init( ) rather than in onDraw( ) as init( ) gets called once whereas onDraw( ) is called in accordance with the refresh rate (eg. 60Hz).
- Objects created in onDraw( ) could be allocated/deallocated up to 60 times per second.
- Performance. A further penalty if garbage collection is called to deal with these out of reach objects.

The onDraw( ) Method

- Every custom view must implement onDraw( ).
- Draws the entire view.
- Drawing can be effected through primitives or bitmaps.
- Primitives are often preferred as they are quick to render and scale easily.
- Algebra/Linear algebra is useful.
- The developer needs to create mathematical expressions for positioning and scaling components.
- Items may also move around within the view, eg. a game.

The Matrix Stack

- This is a stack for storing 4 x 4 matrices which determine the position and orientation of the drawing origin in the current view.
- Eliminates rendering glitches that are caused by rounding errors in floating point numbers.
- This is necessary because floating point numbers have limited precision such that they can be thought of as approximations.
- If multiple translations are made to and from the origin, the origin may ‘drift’.
- The matrix stack addresses this problem.

The Matrix Stack

- The canvas methods save( ) and restore( ) provide a means of saving the drawing origin onto the matrix stack and then recovering it.
- Similar to push and pop operations.
- saves and restores must match due to the stack structure (LIFO).
- translate( ) allows the origin to be moved by an offset.
- scale( ) allows an object to be drawn larger or smaller without dealing with its co-ordinates.
- rotate( ) allows the rotation of an object without having to change its co-ordinates.

Managing the View

- The invalidate( ) method allows the view to be re-drawn when something has been updated internally by the application and needs to display.
- Requests the OS to schedule a re-draw.
- Never call onDraw( ) directly as this creates a feedback loop that locks up the application.
- Sometimes it is required to restrict the size and shape of the view. onMeasure( ) can be used for this.

User Interaction

- User interaction can be handled by implementing the onTouchEvent( ) method.
- Single touch (simple model).
- Multiple touches (complex model).
- Single touch is the easiest case:
- ACTION_DOWN, ACTION_MOVE, ACTION_MOVE, ACTION_UP
- Multi touch:
- ACTION_POINTER_DOWN, ACTION_POINTER_UP, ACTION_MOVE (indicates movement)

Integration into an Application

- Use a custom view in an application by using its fully qualified name in the xml file:

```
<com.example.customview3.SpecialView3
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

- Consider providing an attribute set for greater flexibility.
- Consider also providing event handlers so that it is possible to use the custom view without requiring access to the source code to handle events.