Introduction to Android Programming

Xin Yang

2016-05-06

Outline

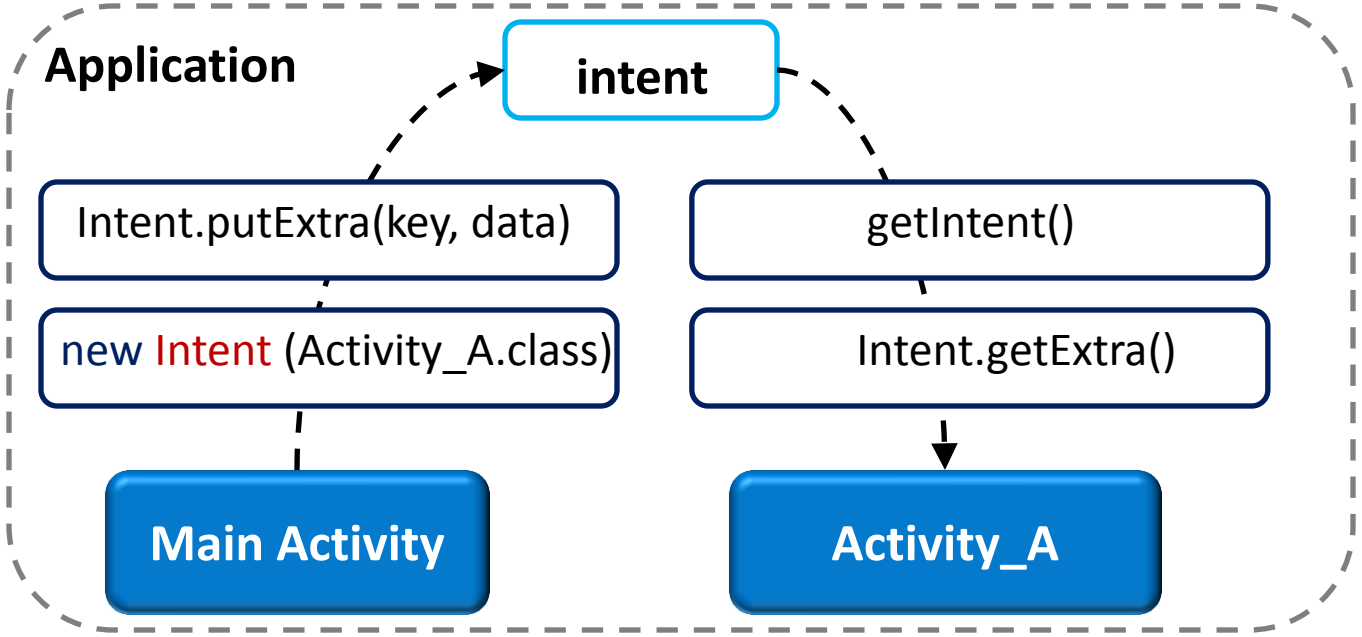
- Utilizing Camera Hardware in Your App
- Drawing 2D Graphics on Screen

Using Camera in Android App

- Two ways to invoke
  - Use existing camera app via **Intent**
    - ✓ Minimal coding, limited design flexibility
  - Implement your own using **Camera API**
    - ✓ More coding, customized interface and features

Use Existing Camera Apps via **Intent**

- Intent
  - A messaging object which facilitates communication between activities



http://developer.android.com/guide/components/intents-filters.html

Intent

- Intent Types
  - **Explicit intents**: specify component to start by name. It is used to start component in your own
  - **Implicit intents**: specify component neral action to perform.


```
intent.setAction(Intent.ACTION_VIEW);
```

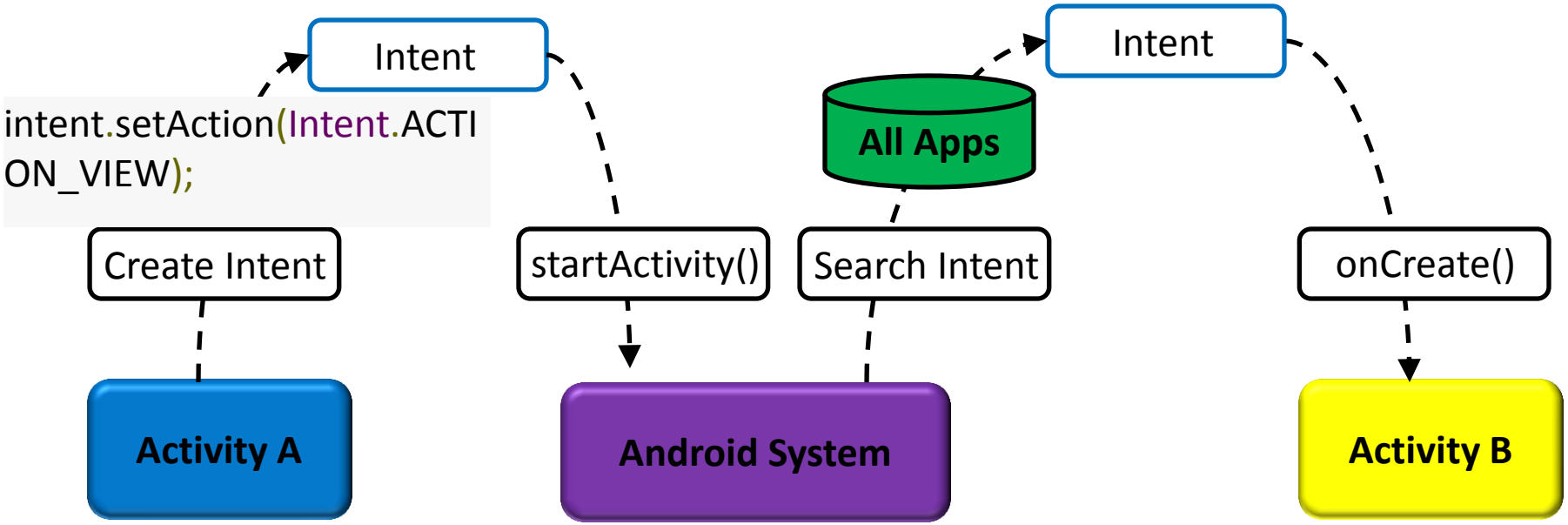


Fig. Illustration of how an implicit intent is delivered to start another activity

Using Existing Camera Apps

- Compose a Camera Intent
  - MediaStore.ACTION_IMAGE_CAPTURE
  - MediaStore.ACTION_VIDEO_CAPTURE
- Start the Camera Intent
  - StartActivityForResult()
- Receive the Intent Result
  - onActivityResult()

Example Code: Step 1 & 2

```
// create Intent t                    et
the calling applicati
Intent intent = new
                Intent(MediaStore.ACTION_IMAGE_CAPTURE);

// create a file to save the image
File tempFile = File.creatTempFile("cameraImg", ".jpg");
Uri fileUri = Uri.fromeFile(tempFile);

intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);


//start the image cap
startActivityForResult(intent,
                    CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE);
```

1. Compose a camera intent

Intent action type for requesting an image from an existing camera app

2. Start the camera intent and display camera app interface

Exampe Code: Step 3

3. Receive the intent result (i.e. callback and data)

```
@Override
protected void onActivityResult(int requestCode, int
                                resultCode, Intent data) {

    if (requestCode == CAPTURE_IMAGE_ACTIVITY_REQUEST_CODE) {
        if (resultCode == RESULT_OK) {

            //display image
            Bitmap img = (Bitmap) data.getExtras().get("data");
            ImageView imageTakePic =
                          (ImageView)findViewById(R.id.imageView);
            imageTakePic.setImageBitmap(img);
        }
        else if(resultCode == RESULT_CANCELLED){
            //User cancelled the image capture
        }
}
```

Build Your Own Camera App using **Camera API**

- General steps to build a camera app
  1. Detect and access camera
     - <uses-feature android:name="android.hardware.camera"
       android:required ="true"/>
     - <uses-permssion
       android:name="android.permission.CAMERA">
     - Check at runtime: **PackageManager**.hasSystemFeature (**PackageManager**.FEATURE_CAMERA))
     - **Camera**.open()

http://developer.android.com/guide/topics/media/camera.html#custom-camera

Detecting Camera Hardware

- If your app does not specifically require a camera using manifest declaration, you should check a camera is available at runtime

```
/** Check if this device has a camera */
private boolean checkCameraHardware(Context context) {
    if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)){
        // this device has a camera
        return true;
    } else {
        // no camera on this device
        return false;
    }
}
```

Accessing Cameras

- Access a camera by getting instance of **Camera** object

```
/** A safe way to get an instance of the Camera object. */
public static Camera getCameraInstance(){
    Camera c = null;
    try {
        c = Camera.open(); // attempt to get a Camera instance
    }
    catch (Exception e){
        // Camera is not available (in use or does not exist)
    }
    return c; // returns null if camera is unavailable
}
```

Customize Camera Interface using **Camera API**

- General steps to build a camera app
  1. Detect and access camera
  2. Create a camera preview class
     - **SurfaceView** : display the live image data
     - **SurfaceHolder.Callback**: capture the callback events for creating and destroying the view

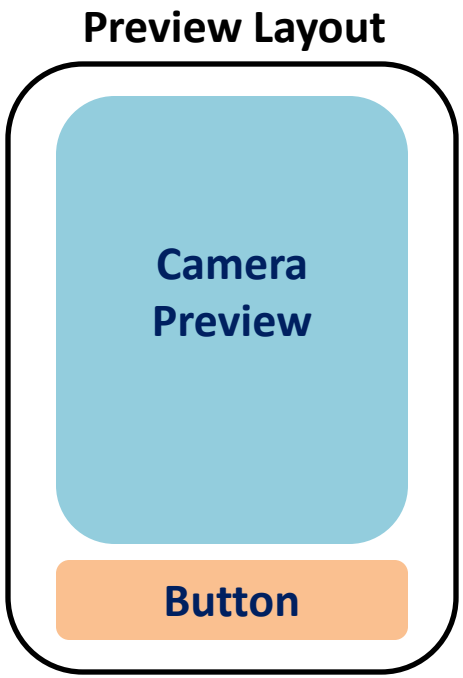
Creating a Preview Class

```
/** A basic Camera preview class */
public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback {
    private SurfaceHolder mHolder;
    private Camera mCamera;

    public CameraPreview(Context context, Camera camera) {
        super(context);
        mCamera = camera;

        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        // deprecated setting, but required on Android versions prior to 3.0
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    public void surfaceCreated(SurfaceHolder holder) {
        // The Surface has been created, now tell the camera where to draw the preview.
        try {
            mCamera.setPreviewDisplay(holder);
            mCamera.startPreview();
        } catch (IOException e) {
            Log.d(TAG, "Error setting camera preview: " + e.getMessage());
        }
    }
```

```
    public void surfaceDestroyed(SurfaceHolder holder) {
        // empty. Take care of releasing the Camera preview in your activity.
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        // If your preview can change or rotate, take care of those events here.
        // Make sure to stop the preview before resizing or reformatting it.

        if (mHolder.getSurface() == null){
          // preview surface does not exist
          return;
        }

        // stop preview before making changes
        try {
            mCamera.stopPreview();
        } catch (Exception e){
          // ignore: tried to stop a non-existent preview
        }

        // set preview size and make any resize, rotate or
        // reformatting changes here

        // start preview with new settings
        try {
            mCamera.setPreviewDisplay(mHolder);
            mCamera.startPreview();

        } catch (Exception e){
            Log.d(TAG, "Error starting camera preview: " + e.getMessage());
        }
    }
}
```

Customize Camera Interface using **Camera API**

- General steps to build a camera app
  1. Detect and access camera
  2. Create a camera preview class
  3. Build a preview layout



Placing Preview in a Layout

- Camera preview class must be placed in the layout of an activity along with UI controls for taking pictures

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
  <FrameLayout
    android:id="@+id/camera_preview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    />

  <Button
    android:id="@+id/button_capture"
    android:text="Capture"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    />
</LinearLayout>
```

FrameLayout element is a container for camera preview class

```
public class CameraActivity extends Activity {

    private Camera mCamera;
    private CameraPreview mPreview;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Create an instance of Camera
        mCamera = getCameraInstance();

        // Create our Preview view and set it as the content of our activity.
        mPreview = new CameraPreview(this, mCamera);
        FrameLayout preview = (FrameLayout) findViewById(R.id.camera_preview);
        preview.addView(mPreview);
    }
}
```

Customize Camera Interface using **Camera API**

- General steps to build a camera app
  1. Detect and access camera
  2. Create a camera preview class
  3. Build a preview layout
  4. Setup listeners for capture
     - Call **Camera**.takePicture()

Capturing Pictures

- To retrieve a picture, use **Camera.takePicture()** method
  - Implements **Camera.PictureCallback** interface to receive data

```
private PictureCallback mPicture = new PictureCallback() {

    @Override
    public void onPictureTaken(byte[] data, Camera camera) {

        File pictureFile = getOutputMediaFile(MEDIA_TYPE_IMAGE);
        if (pictureFile == null){
            Log.d(TAG, "Error creating media file, check storage permissions: " +
                e.getMessage());
            return;
        }

        try {
            FileOutputStream fos = new FileOutputStream(pictureFile);
            fos.write(data);
            fos.close();
        } catch (FileNotFoundException e) {
            Log.d(TAG, "File not found: " + e.getMessage());
        } catch (IOException e) {
            Log.d(TAG, "Error accessing file: " + e.getMessage());
        }
    }
};
```

Adding a Listener to the Capture Button

```
// Add a listener to the Capture button
Button captureButton = (Button) findViewById(id.button_capture);
captureButton.setOnClickListener(
    new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // get an image from the camera
            mCamera.takePicture(null, null, mPicture);
        }
    }
);
```

public final void **takePicture** (Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg)

Added in API level 1

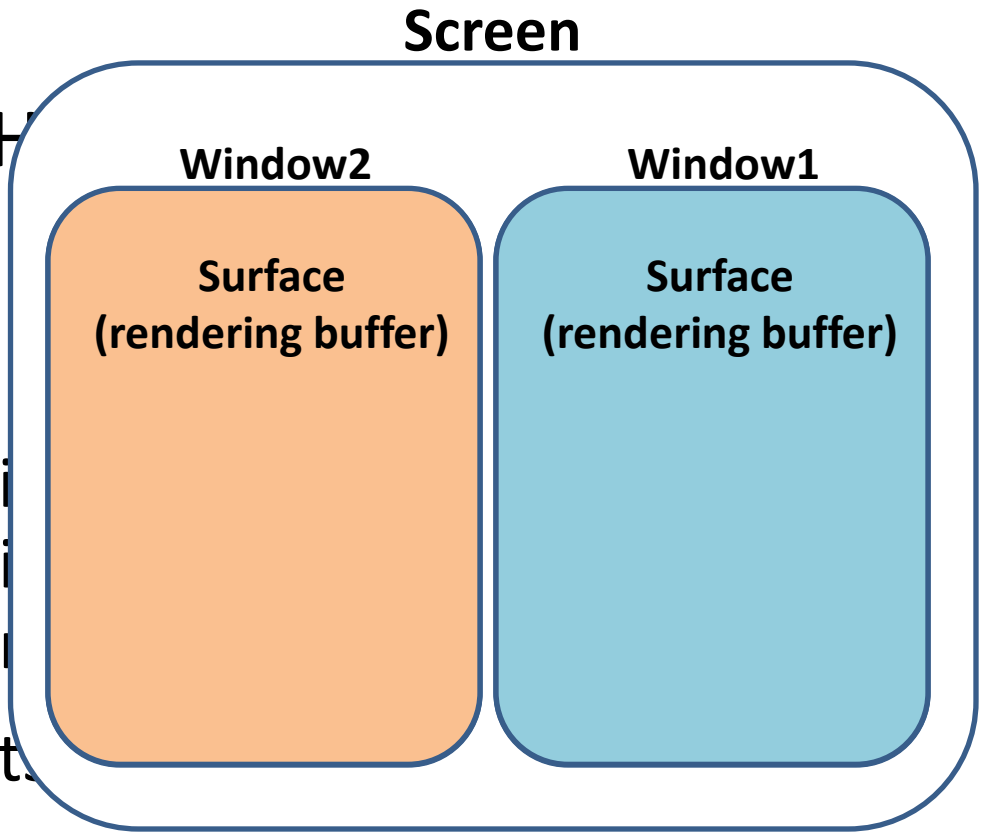
Customize Camera Interface using **Camera API**

- General steps to build a camera app
  1. Detect and access camera
  2. Create a camera preview class
  3. Build a preview layout
  4. Setup listeners for capture
  5. Capture and save file
  6. Release the camera

http://developer.android.com/guide/topics/media/camera.html#custom-camera

2D Graphics Drawing

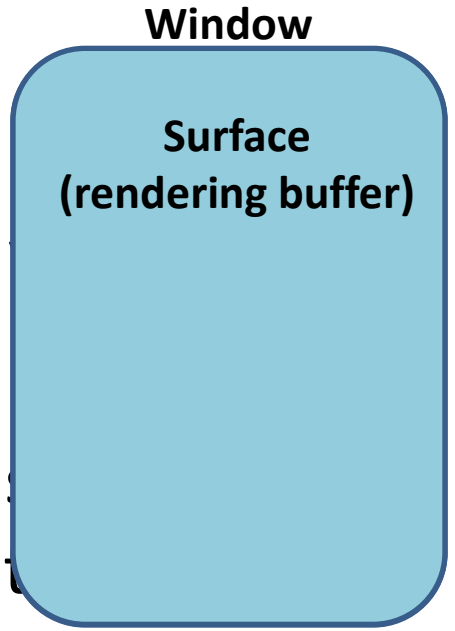
- Where to draw and H
- Terminologies
  - **Surface**
    - An object (i.e. rendi
      of a window (e.g. di
      status bar) is render
    - Every window has it
    - It has more than one buffer for double-buffered rendering



http://source.android.com/devices/graphics.html

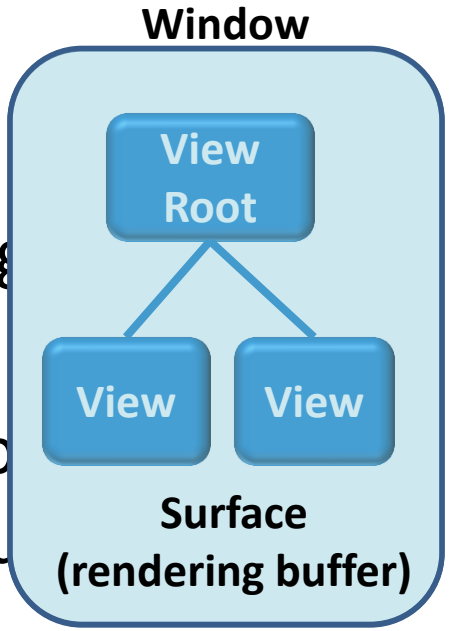
2D Graphics Drawing

- Terminologies
  - **Canvas**
    - An interface to Surface upon which graphics will be drawn
    - It provides a set of drawing methods drawBitmap(), drawCircle(), drawPat
    - Each Canvas maps to a Bitmap to store the content on the surface



2D Graphics Drawing

- Terminologies
  - **View**
    - An interactive UI element (e.g. Imag
      inside of window
    - View objects within a window are o
      view hierarchy and share a single su
  - **SurfaceView**
    - A special implementation of View that creates its own dedicated Surface to directly draw into

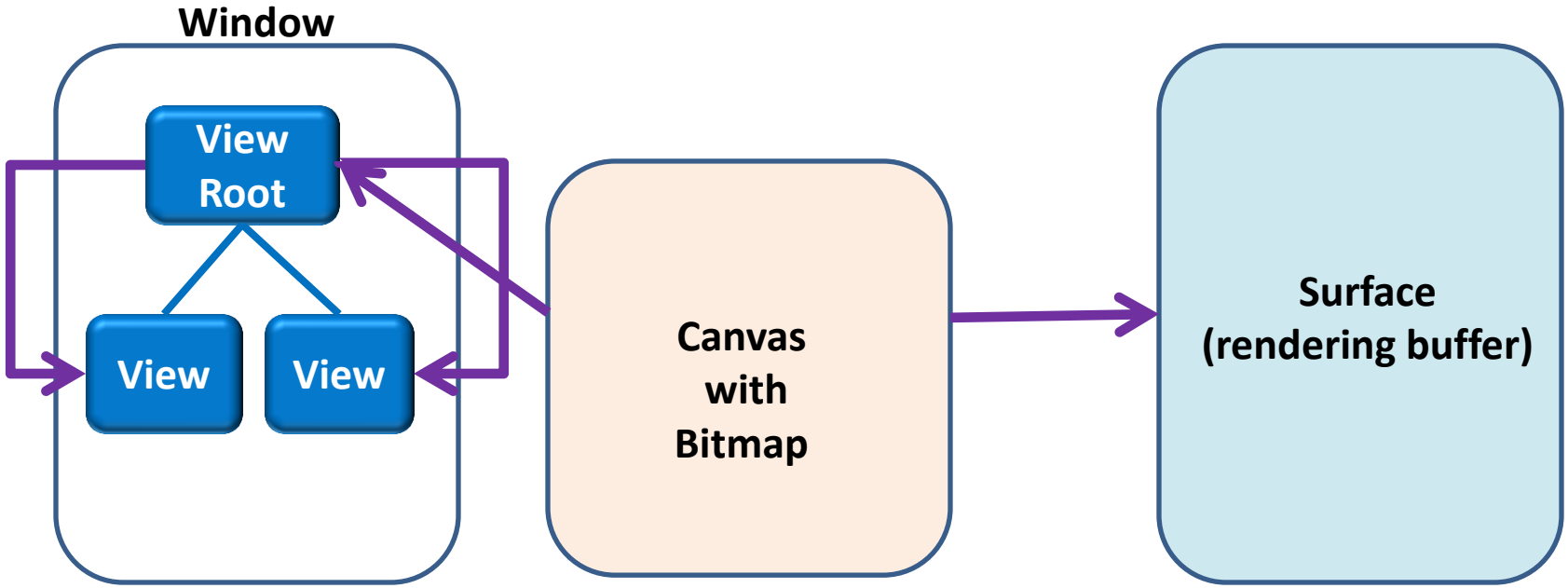


2D Graphics Drawing

- Two ways to draw 2D graphics
  - Draw with a canvas on a View
  - Draw with a canvas on a SurfaceView

http://developer.android.com/guide/topics/graphics/2d-graphics.html

Draw on a View



- Go through view hierarchy drawing process
- For apps which do not require frequent redraw

```
Class  DrawView  extends View {

        Paint paint = new Paint();
        public DrawView(Context context) {
                super(context);
                paint.setColor(Color.BLUE);
        }

        @Override
        public void onDraw(Canvas canvas) {
                super.onDraw(canvas);
                canvas.drawLine(10, 10, 90, 10, paint);
        }
}
```

Will be called when we call DrawView.invalidate().

Layout file include DrawView (activity_draw_view.xml)

```
<View
    class="DrawView"
    android:id ="@+id/drawing_area"
    android:layout_width ="match_parent"
    android:layout_height ="match_parent" />

<Button
    android:onClick="redraw"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

DrawActivity

```
Class DrawActivity extends Activity {

    @Override
     public void onCreate(Bundle savedInstanceSate) {
          super.onCreate(savedInstanceState);

          setContentView(R.layout.activity_draw_view);
          mDrawingArea =
               (DrawView) findViewById(R.id.drawing_area);

          //handle events for button
          void redraw(){
               mDrawingArea.invalidate();
           }
      }
}
```

Main Activity

```
Class MainActivity extends Activity {

    @Override
     public void onCreate(Bundle savedInstanceSate) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);

          //handle events for button
          {
               Intent  intent = new Intent(this, DrawActivity.class);
               startActivity(intent);
           }
     }
}
```

Draw on a SurfaceView

- Has dedicated surface, does not need to go through View hierarchy drawing process
- For apps which require frequent redraw

http://developer.android.com/guide/topics/graphics/2d-graphics.html

```
Class DrawSurfaceView extends SurfaceView
                              implements SurfaceHolder.Callback{

    SurfaceHolder mHolder;
    DrawSuraceView(Context context){
        mHolder = getHolder;
        mHolder.addCallback(this);
    }

    void surfaceCreated(SurfaceHolder holder) {
        Canvas canvas = mHolder.lockCanvas();
        canvas.drawCircle(100, 200, 50, paint);
        mHolder.unlockCanvasAndPost(canvas); }
}
```

Questions???