Android GUI Renderer

David Lau • China

Email: zhiyong.liu@aliyun.com Latest Update：2013-09-08

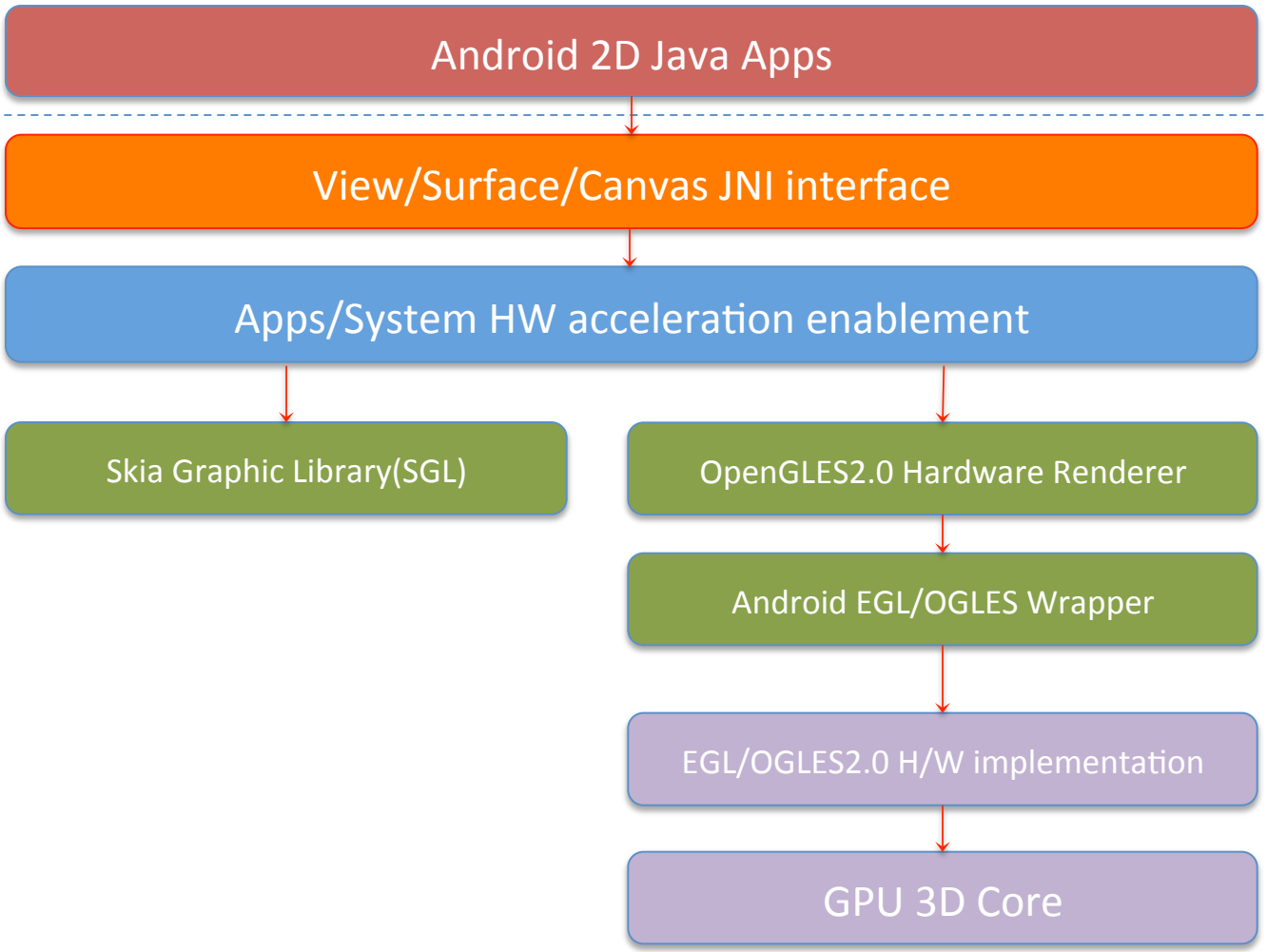
Render in software

- create a native window and then
- wrap a pointer to its buffer as an SkBitmap
- Initialize an SkCanvas with the bitmap

Render in hardware acceleration

- create a GLES2 window or framebuffer and
- create the appropriate GrContext, SkGpuDevice, and SkGpuCanvas

Android 2D Graphic Stack



Source From: https://community.freescale.com/people/XianzhongLi

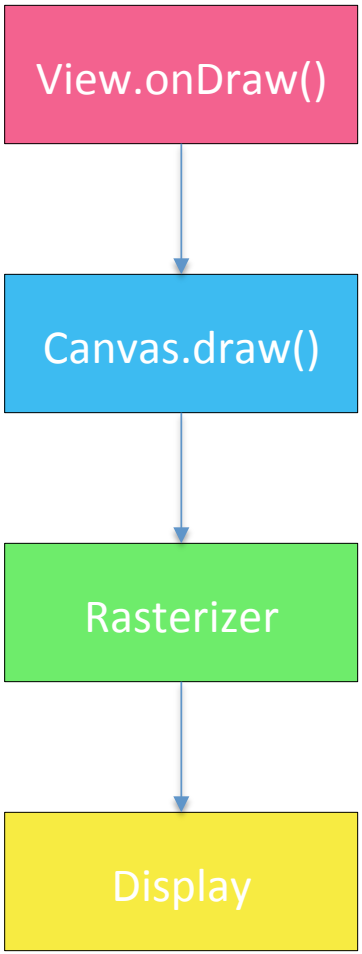
	Drawing a ListView		
	Hardware layer	DisplayList	Software
Time in ms	0.009	2.1	10.3

Measured when drawing a ListView with Android 3.0 on a Motorola XOOM

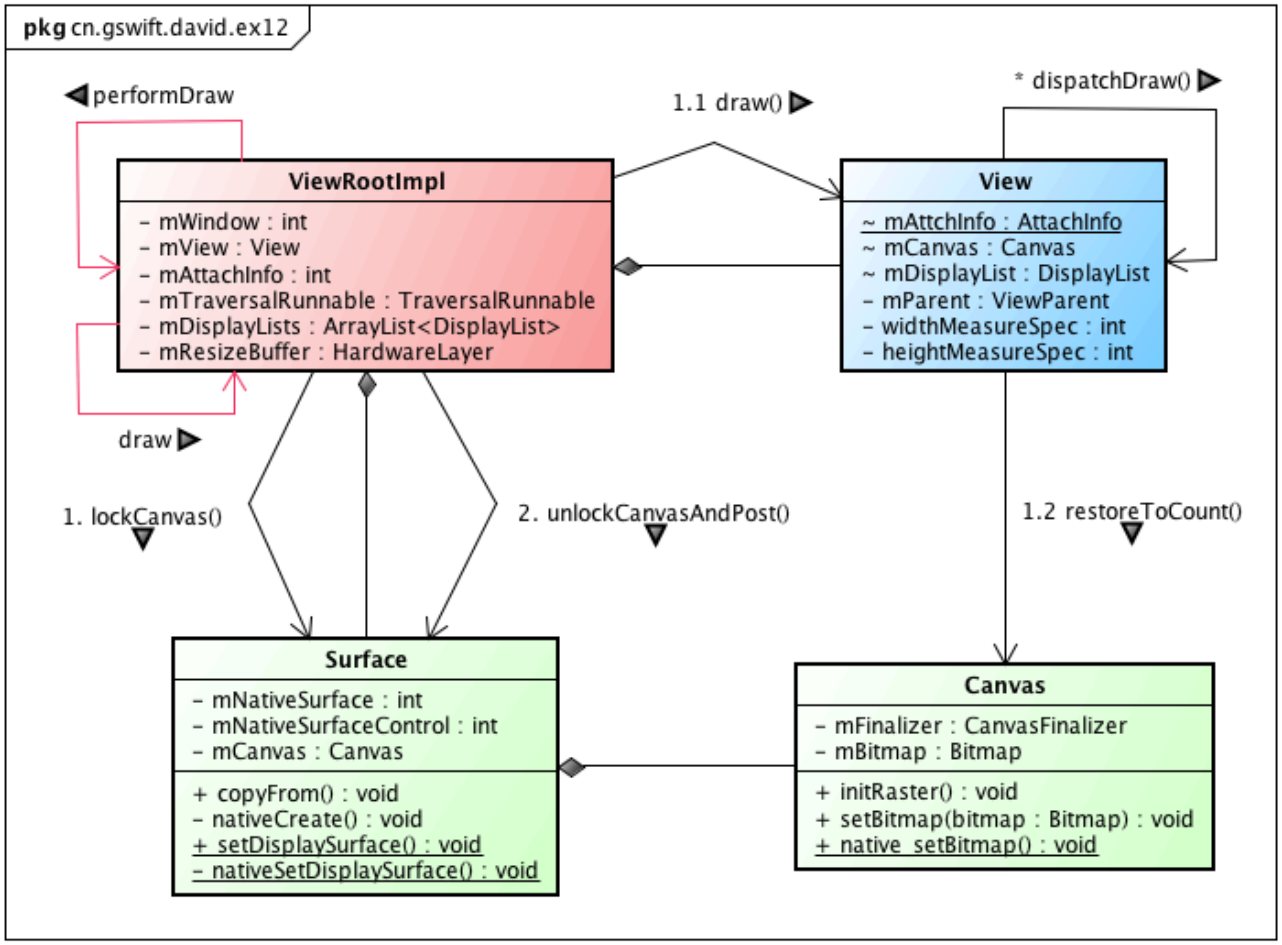
Render in software

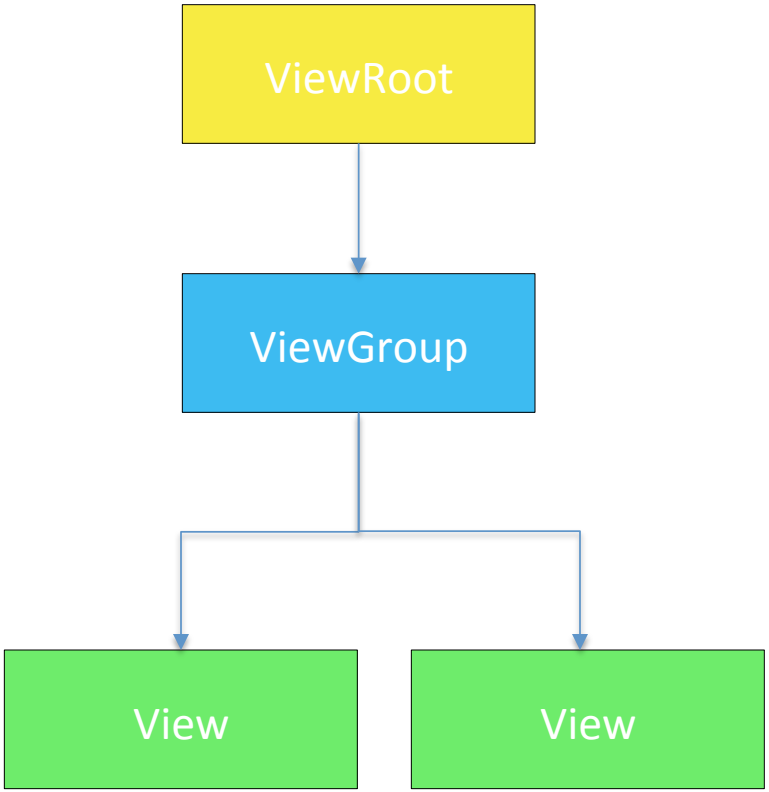
Android Render

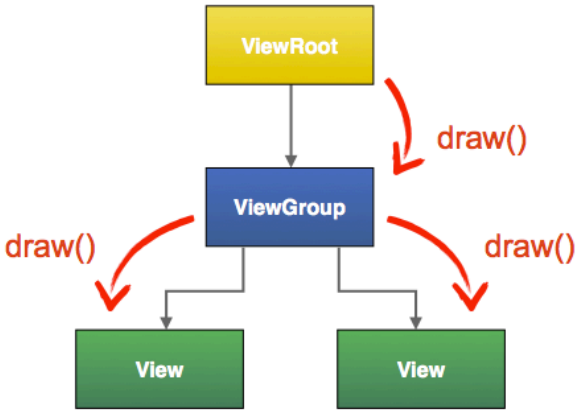
Software Rendering



Draw and *restoreToCount()



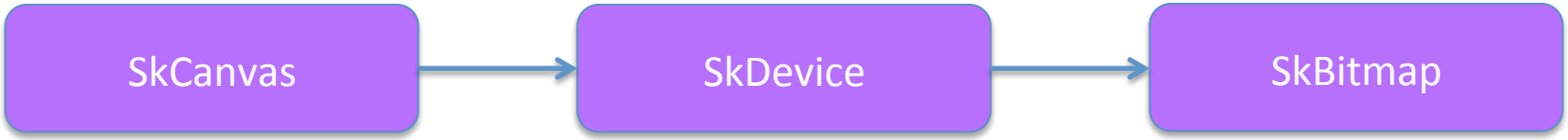

ViewRoot
ViewGroup
View
View


ViewRoot
draw()
ViewGroup
draw()
draw()
View
View

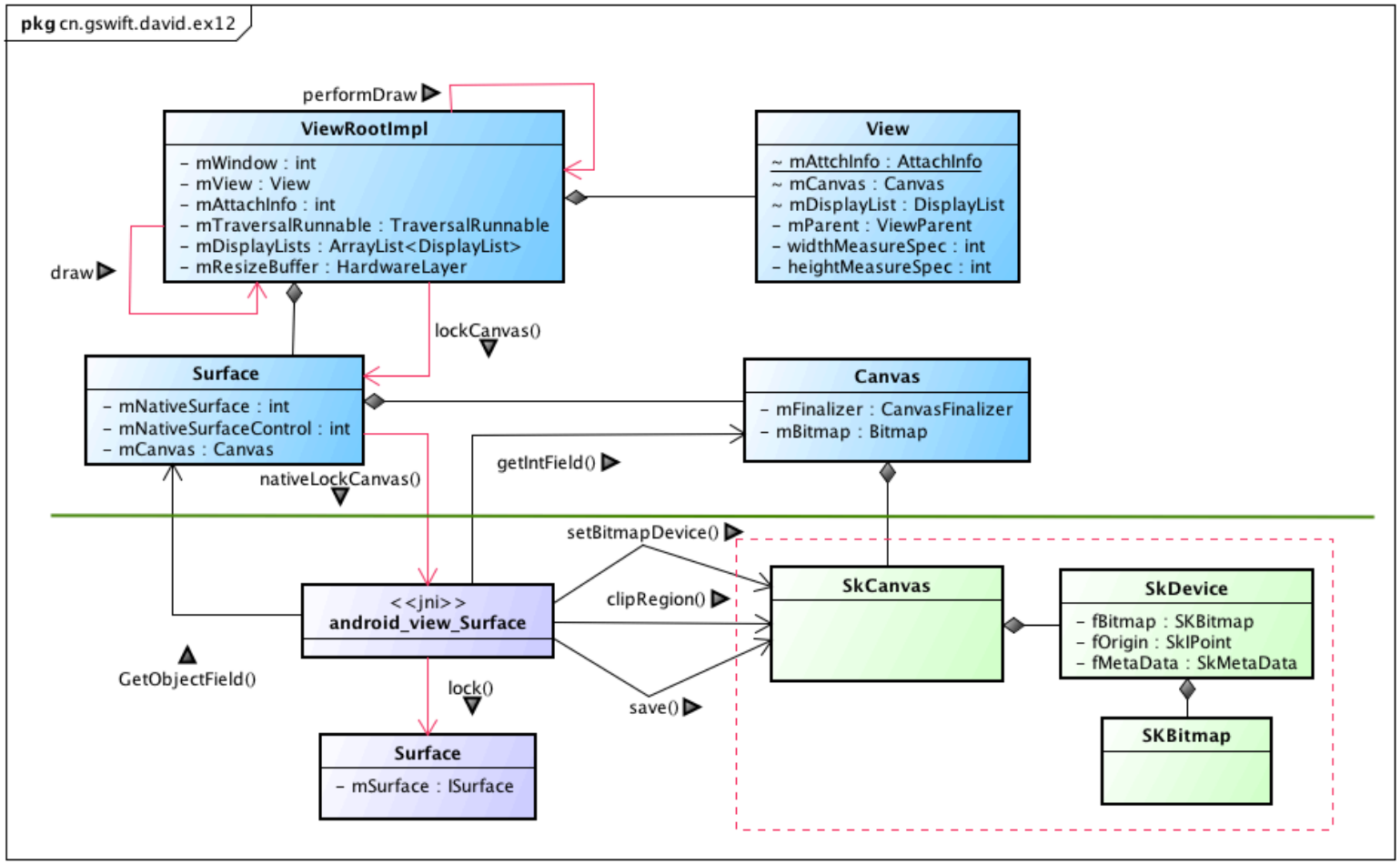
Old model

Render in software with 2D SKIA

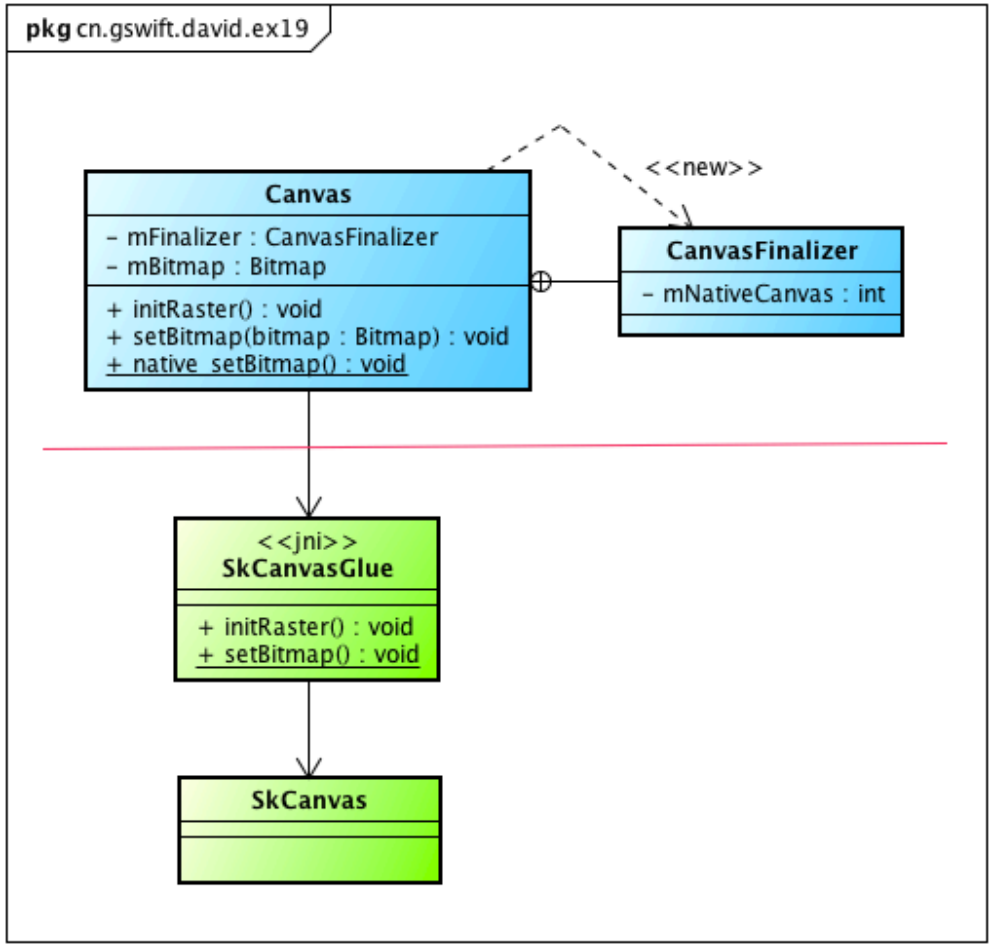
– create a native window and then
– wrap a pointer to its buffer as an SkBitmap
– Initialize an SkCanvas with the bitmap



Drawsoftware()

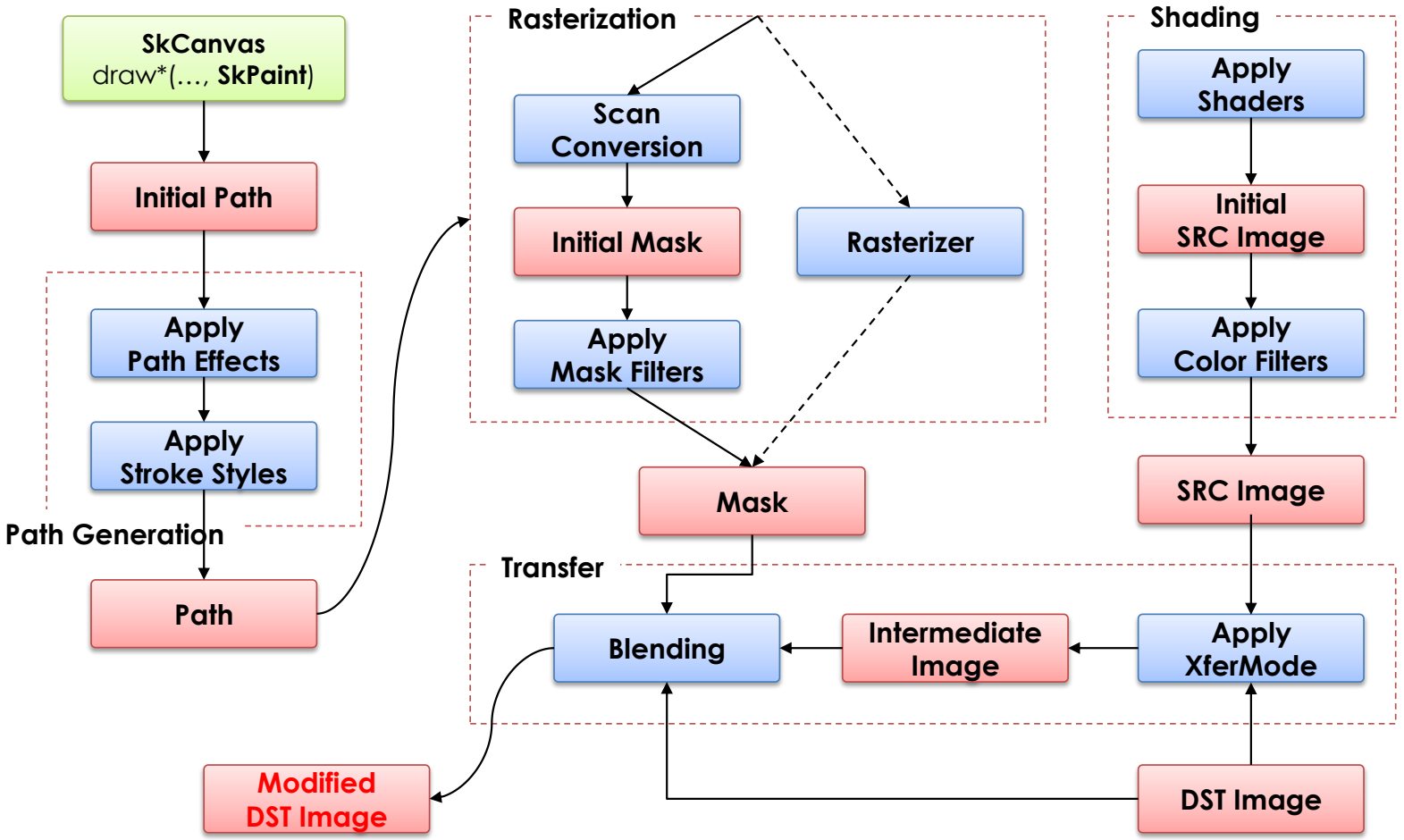


2D SKIA

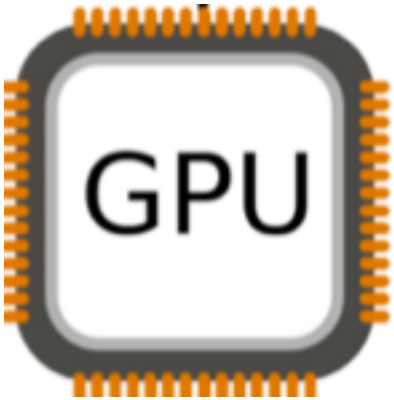


Drawing basic primitives include rectangles, rounded rectangles, ovals, circles, arcs, paths, lines, text, bitmaps and sprites. Paths allow for the creation of more advanced shapes.

Skia rendering pipeline



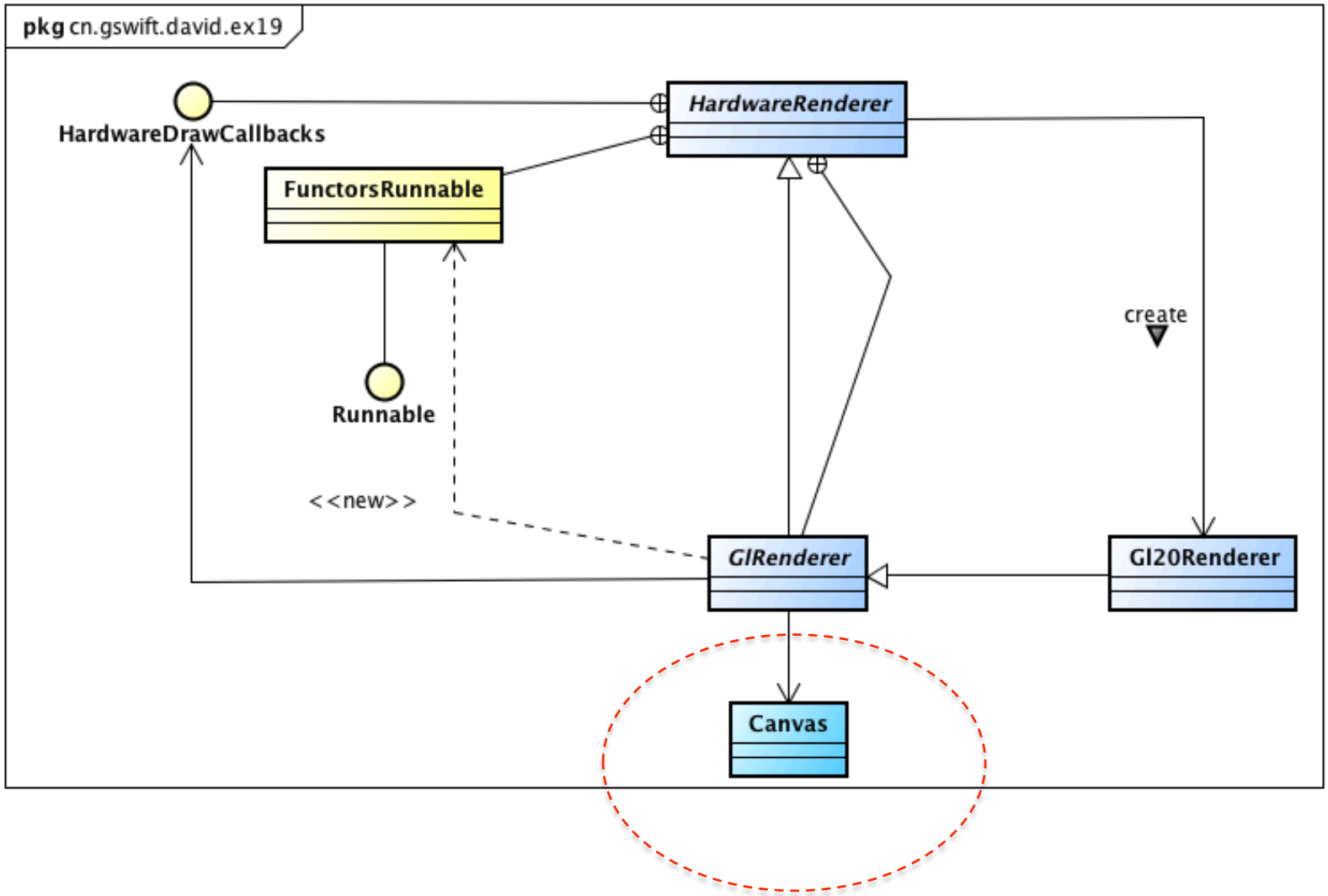
http://www.xenomachina.com/2011/05/androids-2d-canvas-rendering-pipeline.html



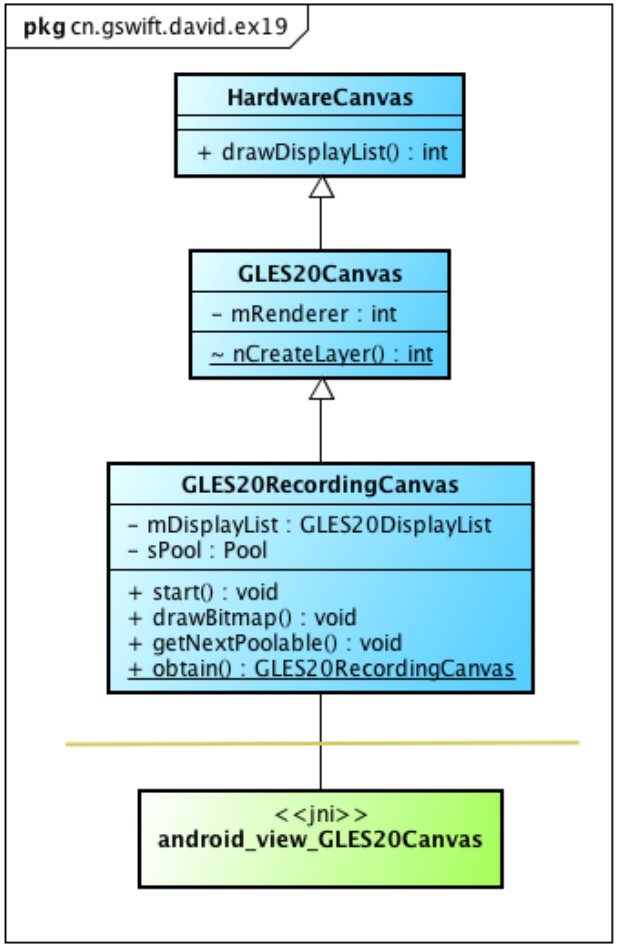
Render in hardware acceleration

ANDROID RENDER

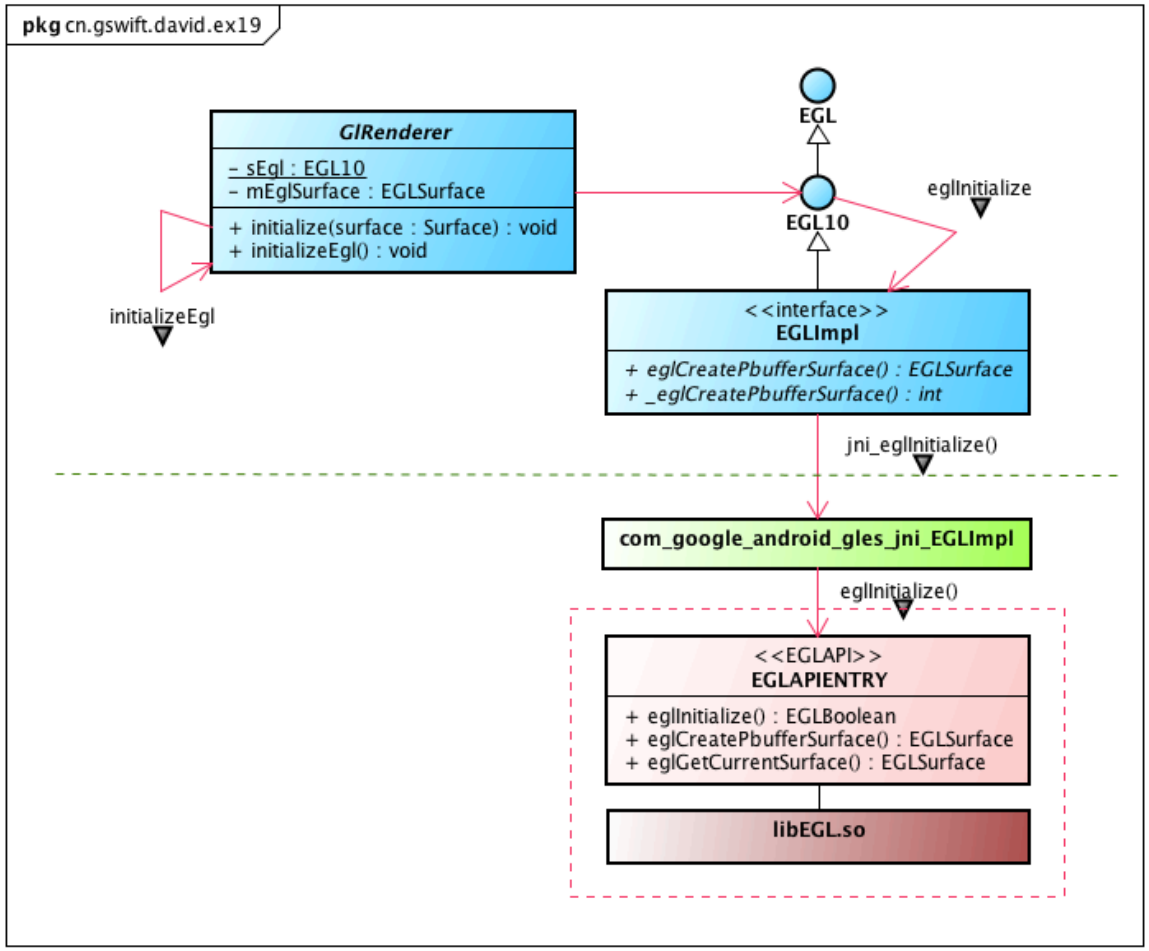
GL20Renderer



Canvas


pkg cn.gswift.david.ex19
HardwareCanvas
+ drawDisplayList() : int
GLES20Canvas
- mRenderer : int
~ nCreateLayer() : int
GLES20RecordingCanvas
- mDisplayList : GLES20DisplayList
- sPool : Pool
+ start() : void
+ drawBitmap() : void
+ getNextPoolable() : void
+ obtain() : GLES20RecordingCanvas
<<jni>>
android_view_GLES20Canvas

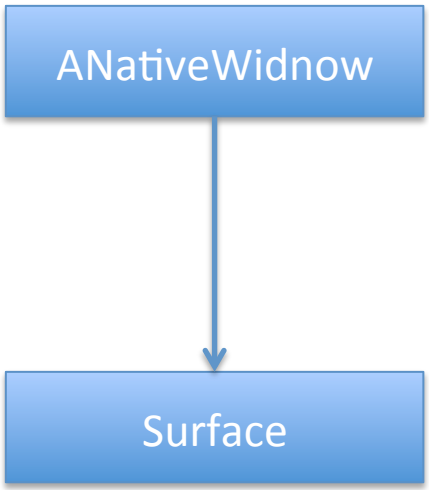
GLRenderer

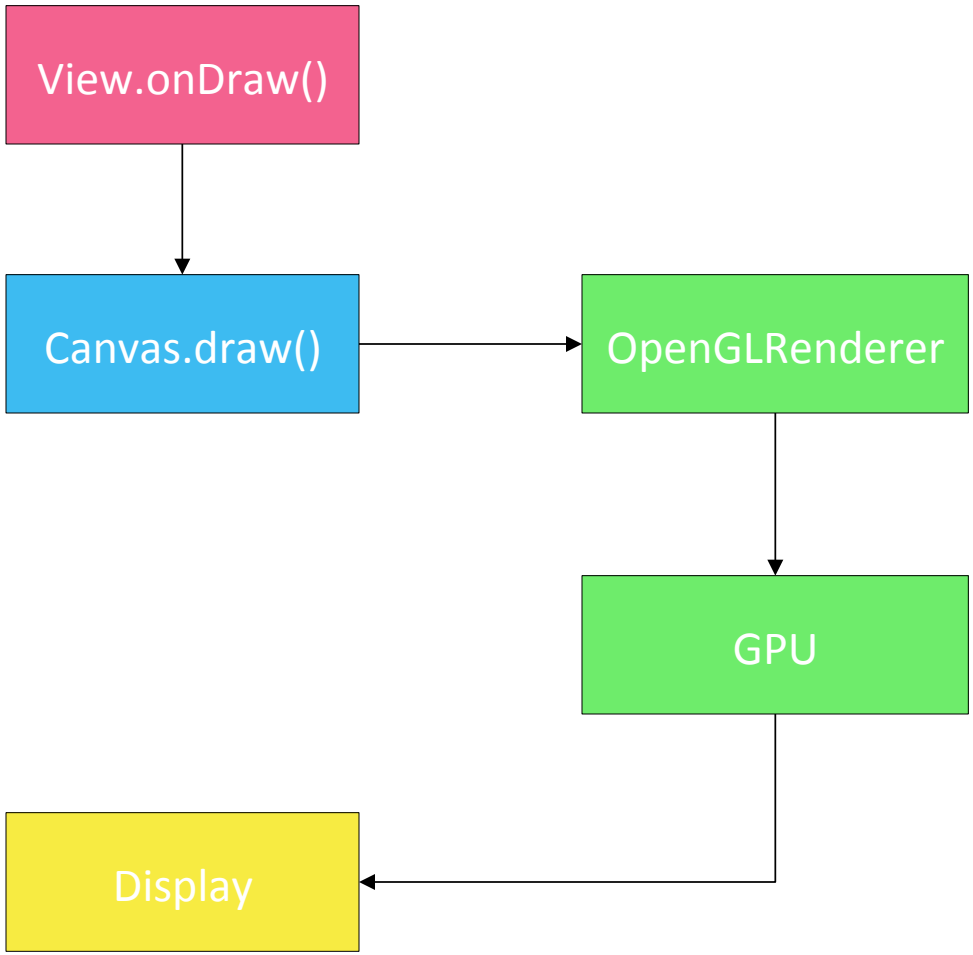

pkg cn.gswift.david.ex19
GlRenderer
- sEgl : EGL10
- mEglSurface : EGLSurface
+ initialize(surface : Surface) : void
+ initializeEgl() : void
initializeEgl
EGL
EGL10
eglInitialize
<<interface>>
EGLImpl
+ eglCreatePbufferSurface() : EGLSurface
+ _eglCreatePbufferSurface() : int
jni_eglInitialize()
com_google_android_gles_jni_EGLImpl
eglInitialize()
<<EGLAPI>>
EGLAPIENTRY
+ eglInitialize() : EGLBoolean
+ eglCreatePbufferSurface() : EGLSurface
+ eglGetCurrentSurface() : EGLSurface
libEGL.so

```java
@Override
boolean initialize(Surface surface) throws Surface.OutOfResourcesException {
    if (isRequested() && !isEnabled()) {
        initializeEgl();
        mGl = createEglSurface(surface);
        mDestroyed = false;

        if (mGl != null) {
            int err = sEgl.eglGetError();
            if (err != EGL_SUCCESS) {
                destroy(true);
                setRequested(false);
            } else {
                if (mCanvas == null) {
                    mCanvas = createCanvas();
                }
                if (mCanvas != null) {
                    setEnabled(true);
                } else {
                    Log.w(LOG_TAG, "Hardware accelerated Canvas could not be created");
                }
            }

            return mCanvas != null;
        }
    }
    return false;
}
```

```java
private boolean createSurface(Surface surface) {
    mEglSurface = sEgl.eglCreateWindowSurface(sEglDisplay, sEglConfig, surface, null);

    if (mEglSurface == null || mEglSurface == EGL_NO_SURFACE) {
        int error = sEgl.eglGetError();
        if (error == EGL_BAD_NATIVE_WINDOW) {
            Log.e(LOG_TAG, "createWindowSurface returned EGL_BAD_NATIVE_WINDOW.");
            return false;
        }
        throw new RuntimeException("createWindowSurface failed "
                + GLUtils.getEGLErrorString(error));
    }

    if (!sEgl.eglMakeCurrent(sEglDisplay, mEglSurface, mEglSurface, mEglContext)) {
        throw new IllegalStateException("eglMakeCurrent failed " +
                GLUtils.getEGLErrorString(sEgl.eglGetError()));
    }

    enableDirtyRegions();

    return true;
}
```

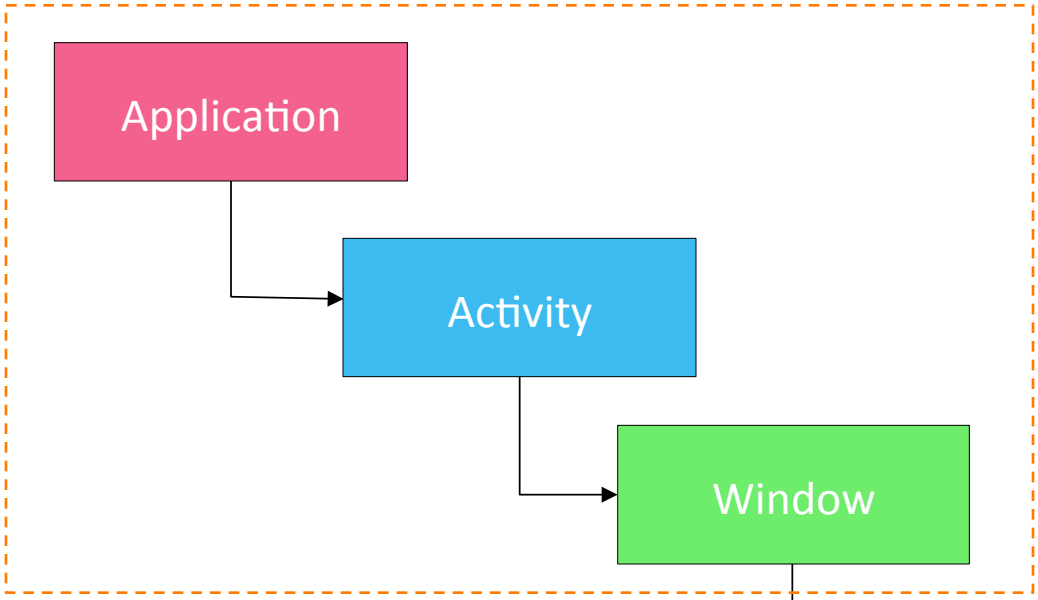
```
EGLSurface eglCreateWindowSurface(  EGLDisplay dpy, EGLConfig config,
                                    NativeWindowType window,
                                    const EGLint *attrib_list)
{
    clearError();

    egl_connection_t* cnx = NULL;
    egl_display_ptr dp = validate_display_connection(dpy, cnx);
    if (dp) {
        EGLDisplay iDpy = dp->disp.dpy;
        EGLint format;

        if (native_window_api_connect(window, NATIVE_WINDOW_API_EGL) != OK) {
            ALOGE("EGLNativeWindowType %p already connected to another API",
                    window);
            return setError(EGL_BAD_NATIVE_WINDOW, EGL_NO_SURFACE);
        }

        // set the native window's buffers format to match this config
        if (cnx->egl.eglGetConfigAttrib(iDpy,
                config, EGL_NATIVE_VISUAL_ID, &format)) {
            if (format != 0) {
                int err = native_window_set_buffers_format(window, format);
                if (err != 0) {
                    ALOGE("error setting native window pixel format: %s (%d)",
                            strerror(-err), err);
                    native_window_api_disconnect(window, NATIVE_WINDOW_API_EGL);
                    return setError(EGL_BAD_NATIVE_WINDOW, EGL_NO_SURFACE);
                }
            }
        }

        // the EGL spec requires that a new EGLSurface default to swap interval
        // 1, so explicitly set that on the window here.
        ANativeWindow* anw = reinterpret_cast<ANativeWindow*>(window);
        anw->setSwapInterval(anw, 1);

        EGLSurface surface = cnx->egl.eglCreateWindowSurface(
                iDpy, config, window, attrib_list);
        if (surface != EGL_NO_SURFACE) {
            egl_surface_t* s = new egl_surface_t(dp.get(), config, window,
                    surface, cnx);
            return s;
        }

        // EGLSurface creation failed
        native_window_set_buffers_format(window, 0);
        native_window_api_disconnect(window, NATIVE_WINDOW_API_EGL);
    }
    return EGL_NO_SURFACE;
}
```





* View is H/W-Accelerated since Android 3.x

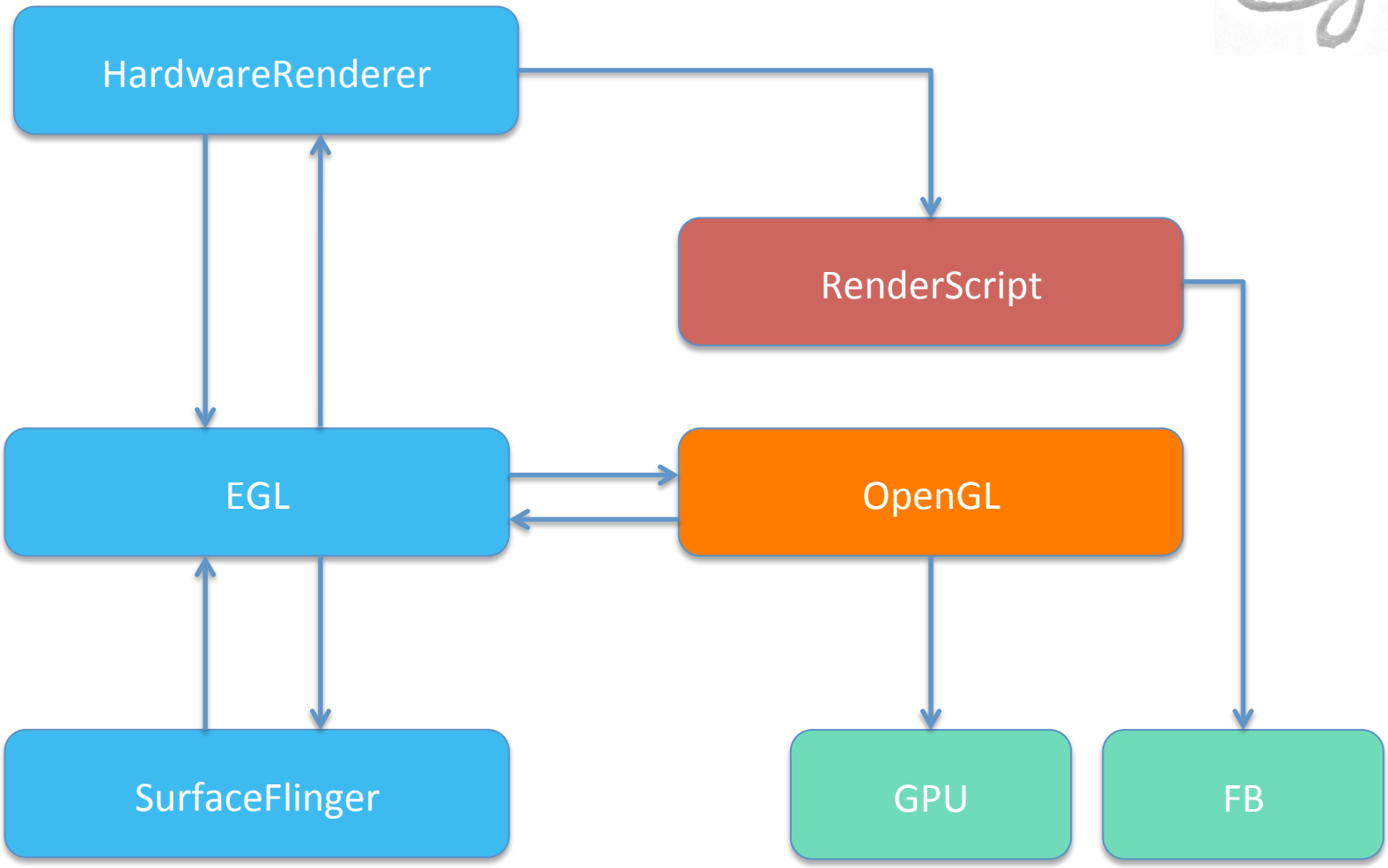
HW Acceleration Control

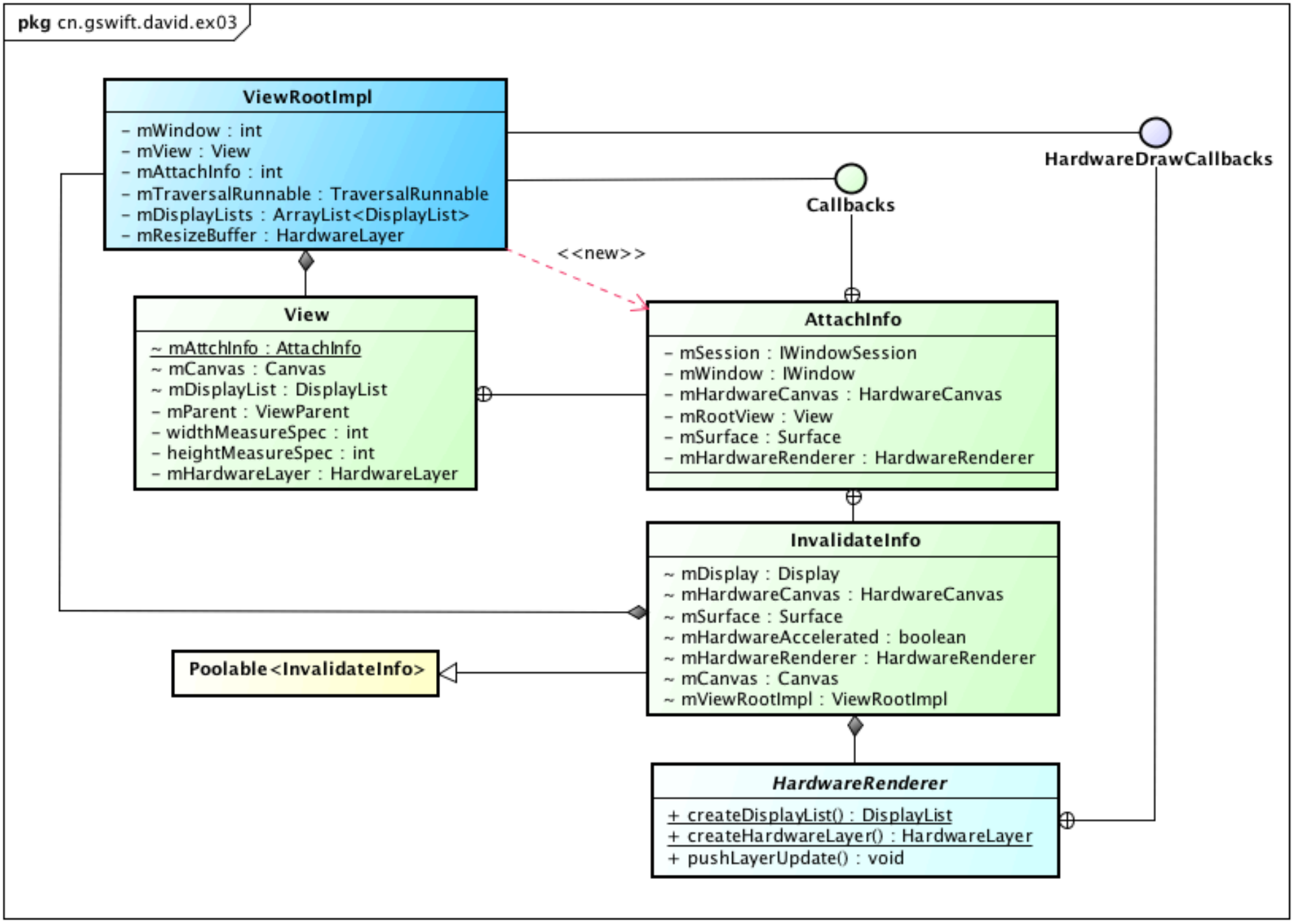


Applicationlevel
<application android:hardwareAccelerated="true">
– Default value
• False in Android 4.x
Activity
Window

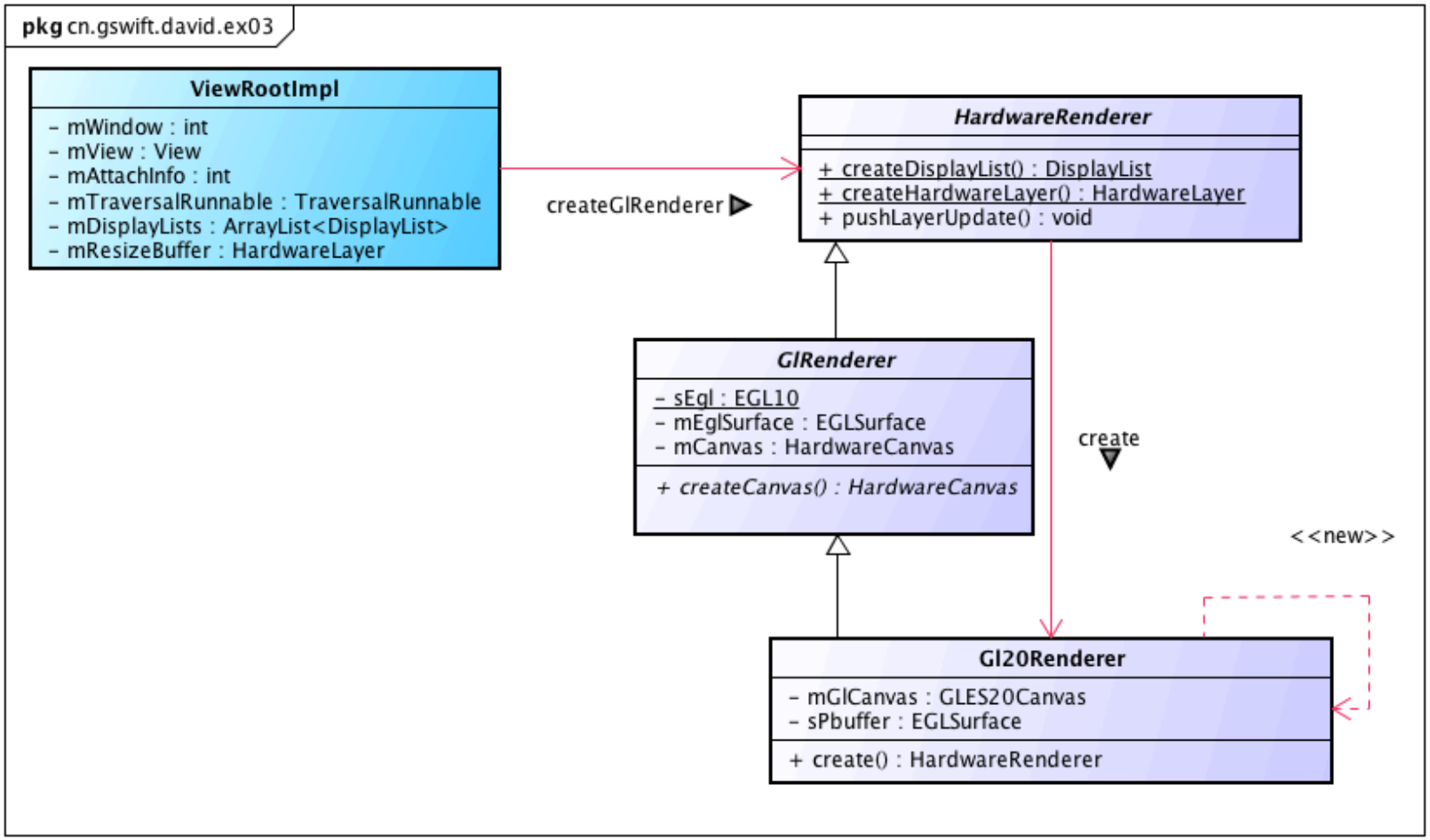
WindowManager.LayoutParams.FLAG_HARDWARE_ACCELERATED
View setLayerType(View.LAYER_TYPE_SOFTWARE,null)

HW Acceleration

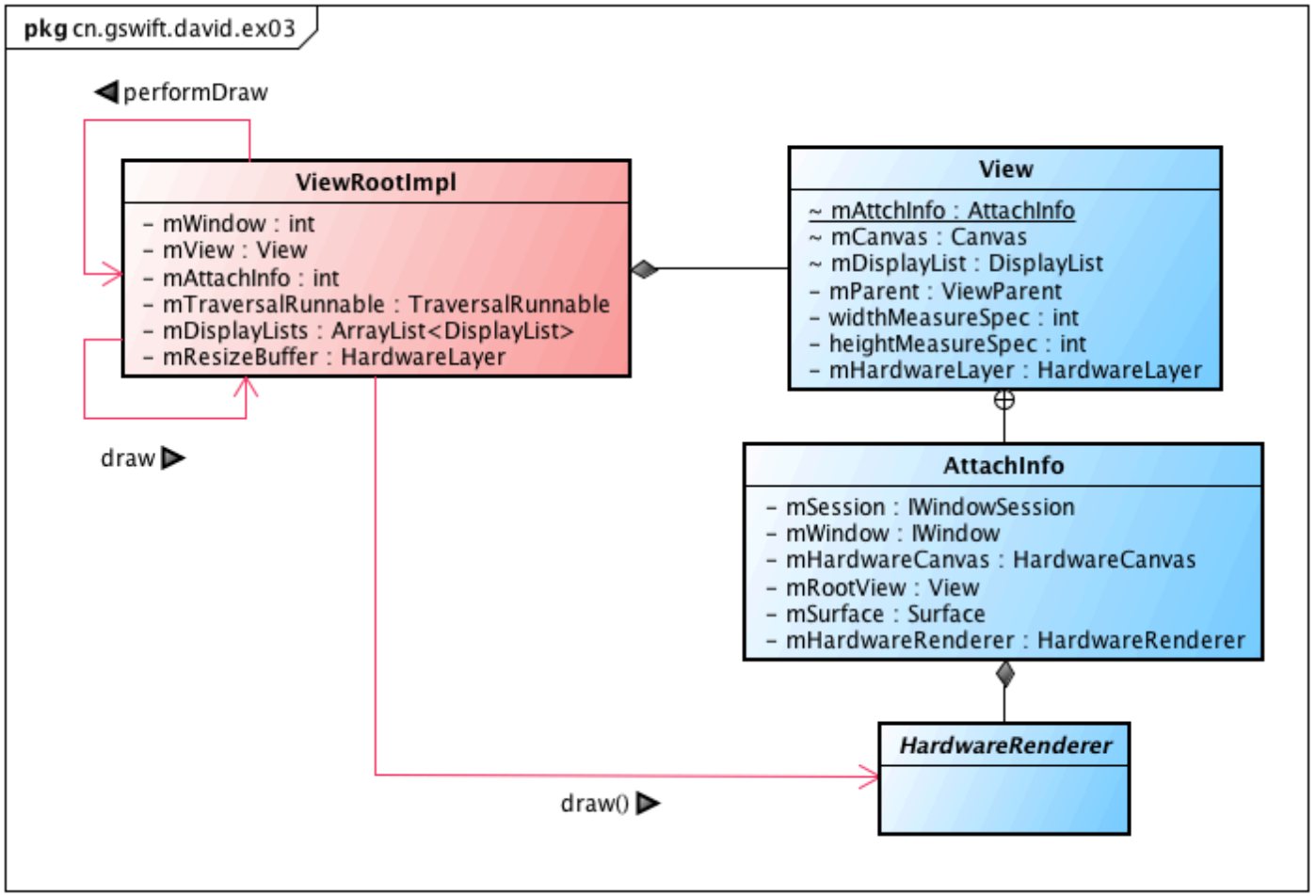



pkg cn.gswift.david.ex03
ViewRootImpl
- mWindow : int
- mView : View
- mAttachInfo : int
- mTraversalRunnable : TraversalRunnable
- mDisplayLists : ArrayList<DisplayList>
- mResizeBuffer : HardwareLayer
HardwareDrawCallbacks
Callbacks
<<new>>
View
~ mAttchInfo : AttachInfo
~ mCanvas : Canvas
~ mDisplayList : DisplayList
- mParent : ViewParent
- widthMeasureSpec : int
- heightMeasureSpec : int
- mHardwareLayer : HardwareLayer
AttachInfo
- mSession : IWindowSession
- mWindow : IWindow
- mHardwareCanvas : HardwareCanvas
- mRootView : View
- mSurface : Surface
- mHardwareRenderer : HardwareRenderer
InvalidateInfo
~ mDisplay : Display
~ mHardwareCanvas : HardwareCanvas
~ mSurface : Surface
~ mHardwareAccelerated : boolean
~ mHardwareRenderer : HardwareRenderer
~ mCanvas : Canvas
~ mViewRootImpl : ViewRootImpl
Poolable<InvalidateInfo>
HardwareRenderer
+ createDisplayList() : DisplayList
+ createHardwareLayer() : HardwareLayer
+ pushLayerUpdate() : void

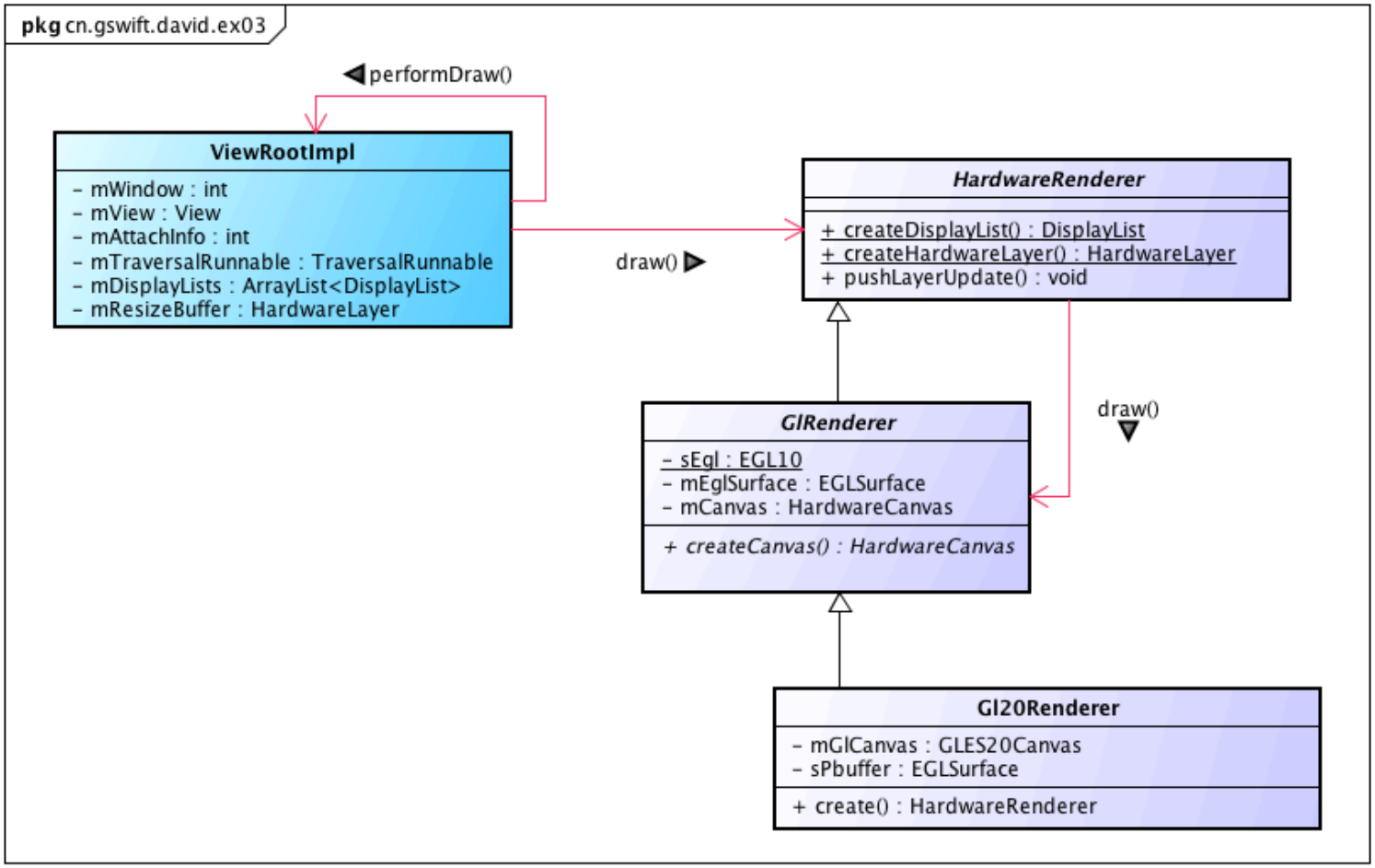
Create a GLRenderer



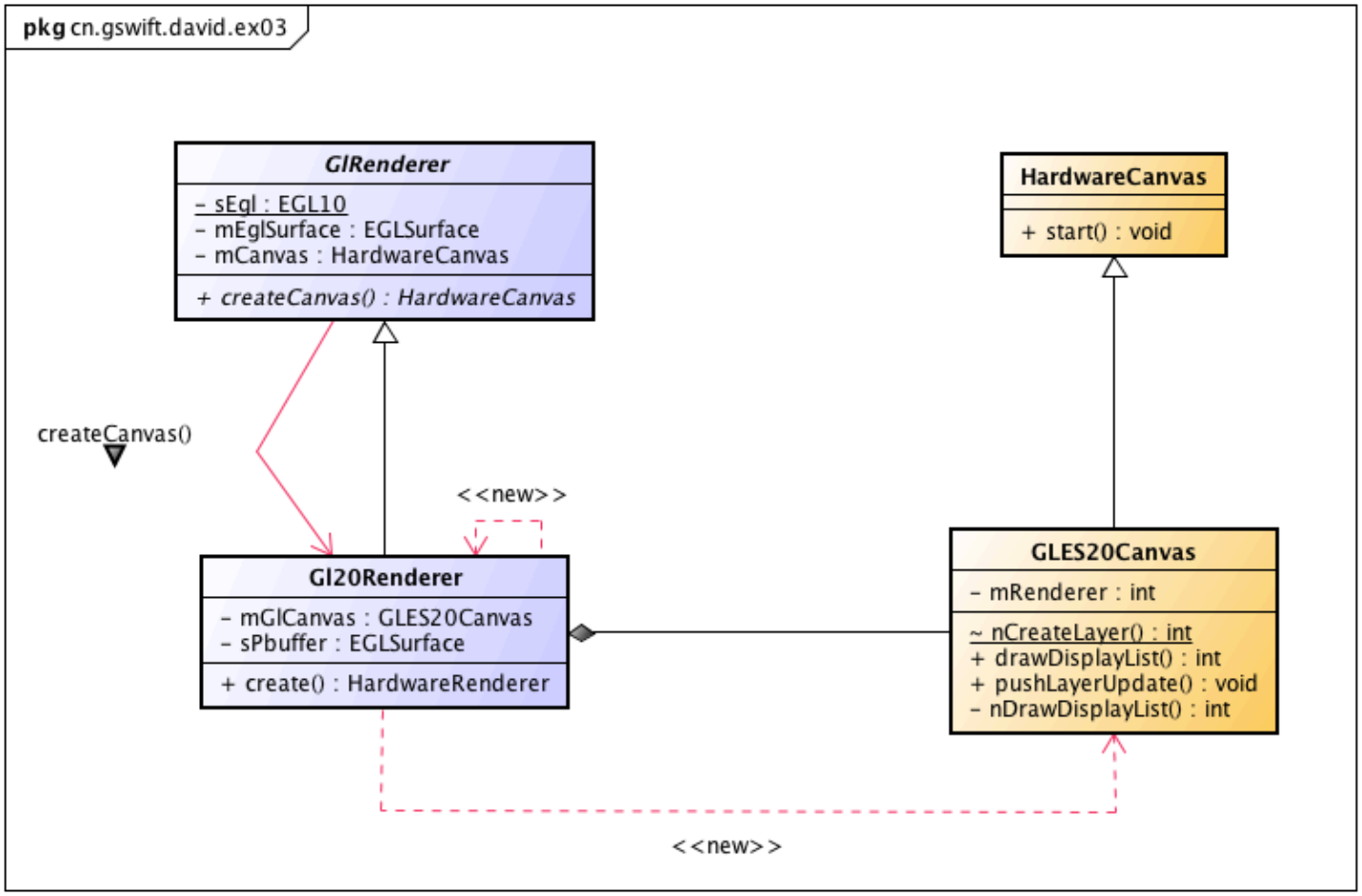
HardwareRenderer draw()



GLRenderer draw()



Create a GLES20Canvas



view/HardwareRenderer.java

```
GL20Renderer(boolean translucent) {
    super(2, translucent);
}

@Override
HardwareCanvas createCanvas() {
    return mGlCanvas = new GLES20Canvas(mTranslucent);
}

@Override
ManagedEGLContext createManagedContext(EGLContext eglContext) {
    return new GL20Renderer.GL20RendererEglContext(mEglContext);
}
```

1

```
/**
 * Creates a canvas to render directly on screen.
 */
GLES20Canvas(boolean translucent) {
    this(false, translucent);
}
```

2

```
/**
 * Creates a canvas to render into an FBO.
 */
GLES20Canvas(int layer, boolean translucent) {
    mOpaque = !translucent;
    mRenderer = nCreateLayerRenderer(layer);
    setupFinalizer();
}
```

View/GLES20RenderLayer.java

```
class GLES20RenderLayer extends GLES20Layer {
    private int mLayerWidth;
    private int mLayerHeight;

    private final GLES20Canvas mCanvas;

    GLES20RenderLayer(int width, int height, boolean isOpaque) {
        super(width, height, isOpaque);

        int[] layerInfo = new int[2];
        mLayer = GLES20Canvas.nCreateLayer(width, height, isOpaque, layerInfo);
        if (mLayer != 0) {
            mLayerWidth = layerInfo[0];
            mLayerHeight = layerInfo[1];

            mCanvas = new GLES20Canvas(mLayer, !isOpaque);
            mFinalizer = new Finalizer(mLayer);
        } else {
            mCanvas = null;
            mFinalizer = null;
        }
    }
```

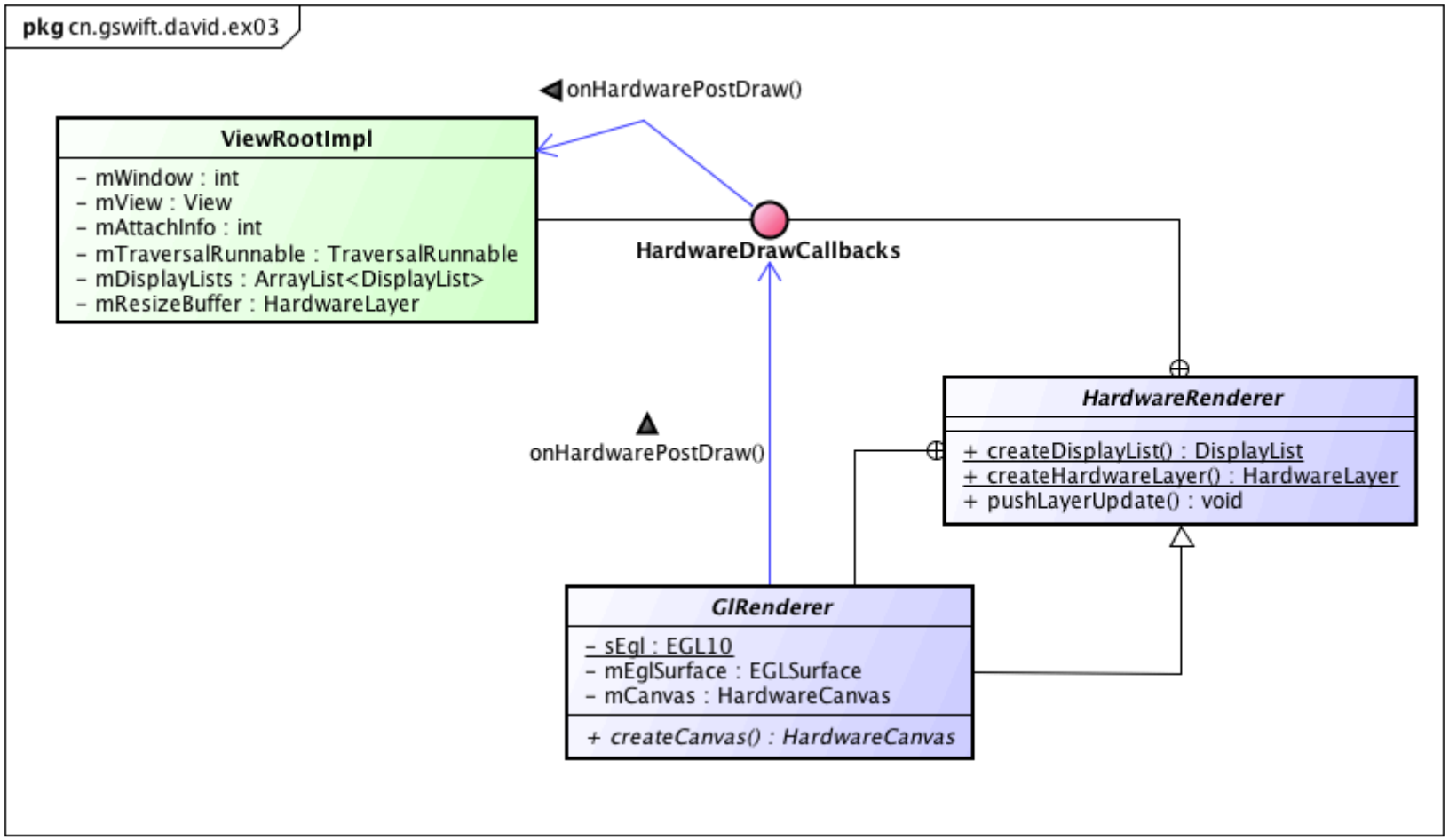
3

```
protected GLES20Canvas(boolean record, boolean translucent) {
    mOpaque = !translucent;

    if (record) {
        mRenderer = nCreateDisplayListRenderer();
    } else {
        mRenderer = nCreateRenderer();
    }

    setupFinalizer();
}
```

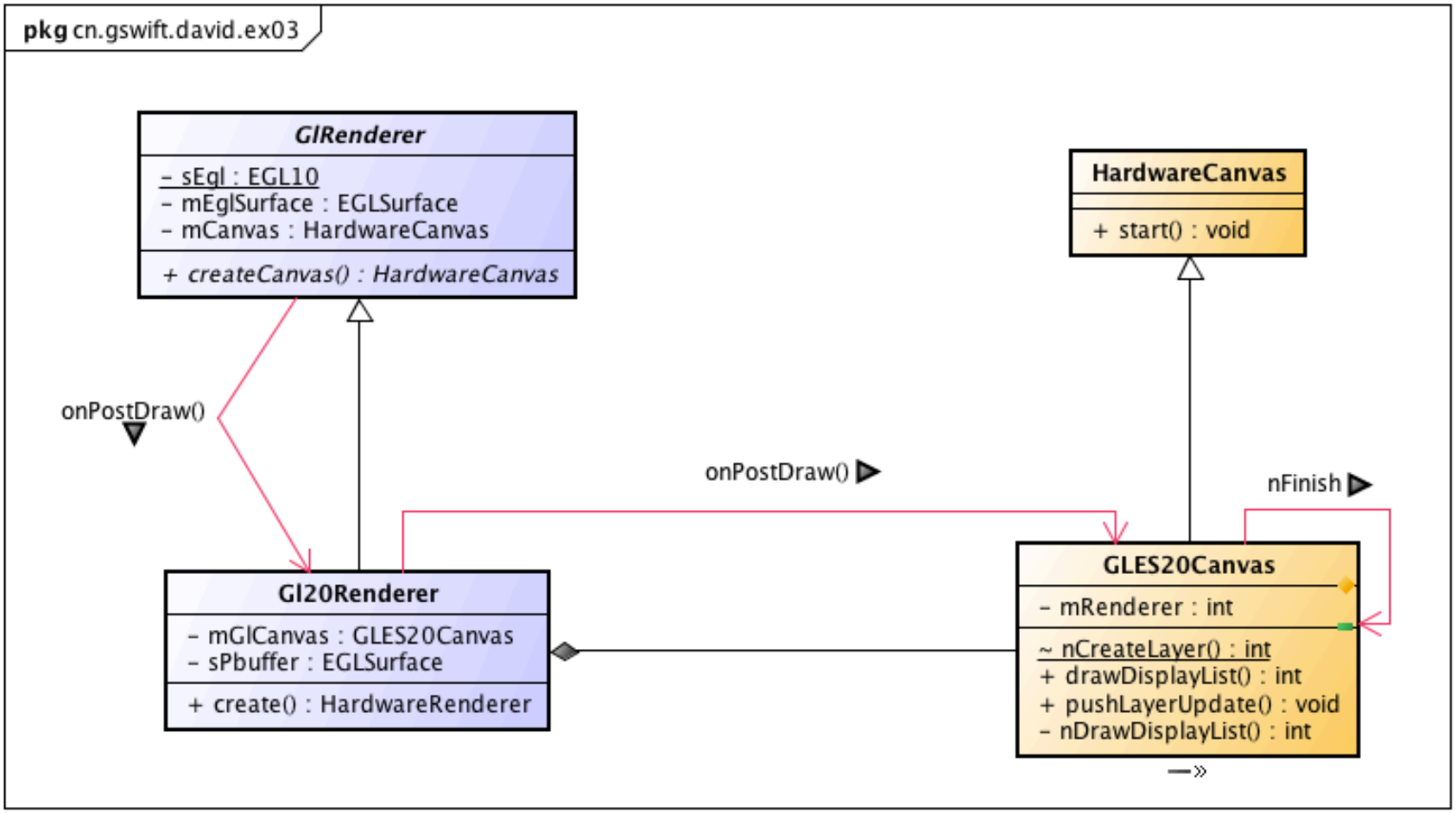
View/GLES20RecordingCanvas.java

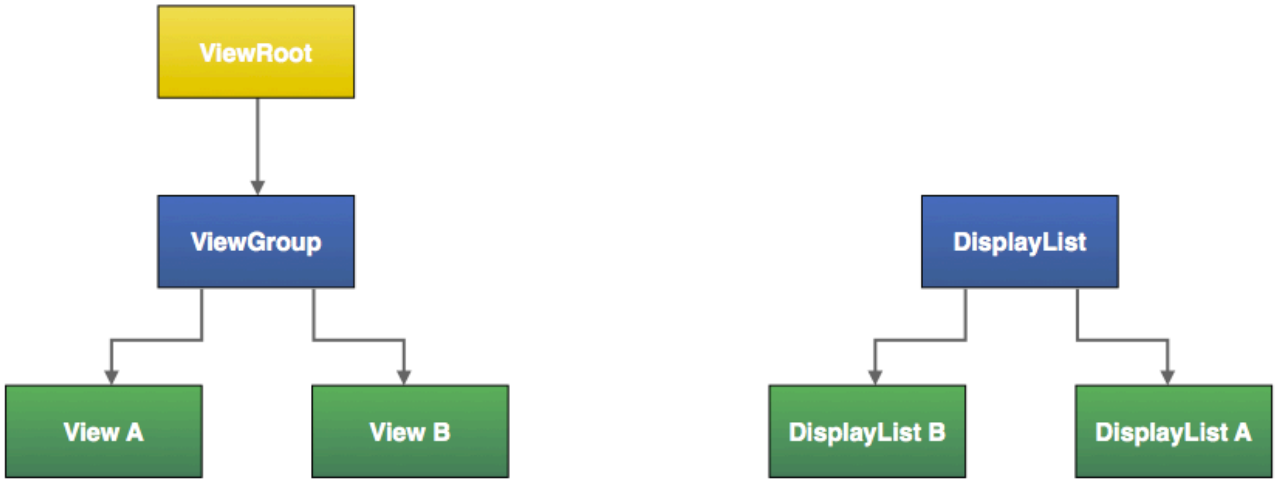
```
class GLES20RecordingCanvas extends GLES20Canvas implements Poolable<GLES20RecordingCanvas> {
    // The recording canvas pool should be large enough to handle a deeply nested
    // view hierarchy because display lists are generated recursively.
    private static final int POOL_LIMIT = 25;

    private static final Pool<GLES20RecordingCanvas> sPool = Pools.synchronizedPool(
            Pools.finitePool(new PoolableManager<GLES20RecordingCanvas>() {
                public GLES20RecordingCanvas newInstance() {
                    return new GLES20RecordingCanvas();
                }
                @Override
                public void onAcquired(GLES20RecordingCanvas element) {
                }
                @Override
                public void onReleased(GLES20RecordingCanvas element) {
                }
            }, POOL_LIMIT));

    private GLES20RecordingCanvas mNextPoolable;
    private boolean mIsPooled;

    private GLES20DisplayList mDisplayList;

    private GLES20RecordingCanvas() {
        super(true /*record*/, true /*translucent*/);
    }
```

onHardwarePostDraw()



onPostDraw()


pkg cn.gswift.david.ex03
GlRenderer
- sEgl : EGL10
- mEglSurface : EGLSurface
- mCanvas : HardwareCanvas
+ createCanvas() : HardwareCanvas
HardwareCanvas
+ start() : void
onPostDraw()
onPostDraw()
nFinish
Gl20Renderer
- mGlCanvas : GLES20Canvas
- sPbuffer : EGLSurface
+ create() : HardwareRenderer
GLES20Canvas
- mRenderer : int
~ nCreateLayer() : int
+ drawDisplayList() : int
+ pushLayerUpdate() : void
- nDrawDisplayList() : int

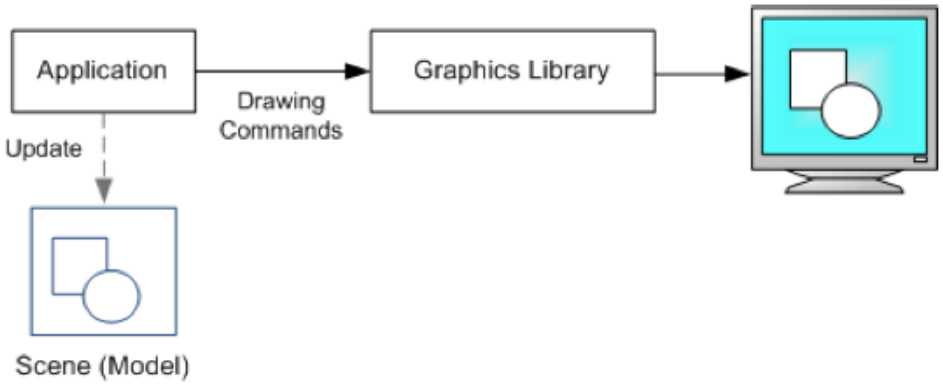

ViewRoot
ViewGroup
View A
View B
DisplayList
DisplayList B
DisplayList A

New model

What's DisplayList

- http://en.wikipedia.org/wiki/Display_list
- http://developer.android.com/guide/topics/graphics/hardware-accel.html

- A display list (or display file) is a series of graphics commands that define an output image. The image is created (rendered) by executing the commands.
- A display list can represent both two- and three-dimensional scenes.
- Systems that make use of a display list to store the scene are called retained mode systems as opposed to immediate mode systems.

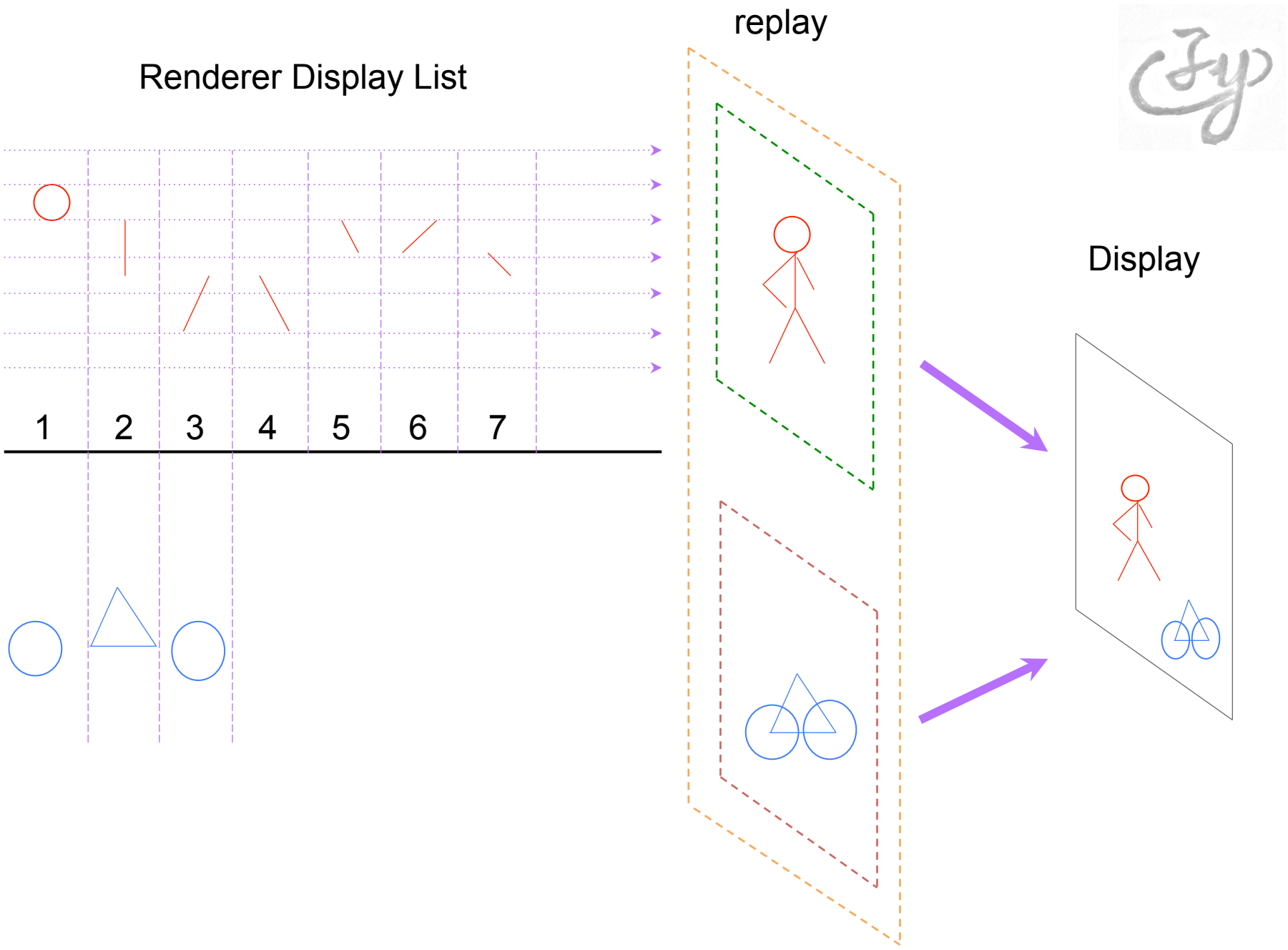


Android's DisplayList

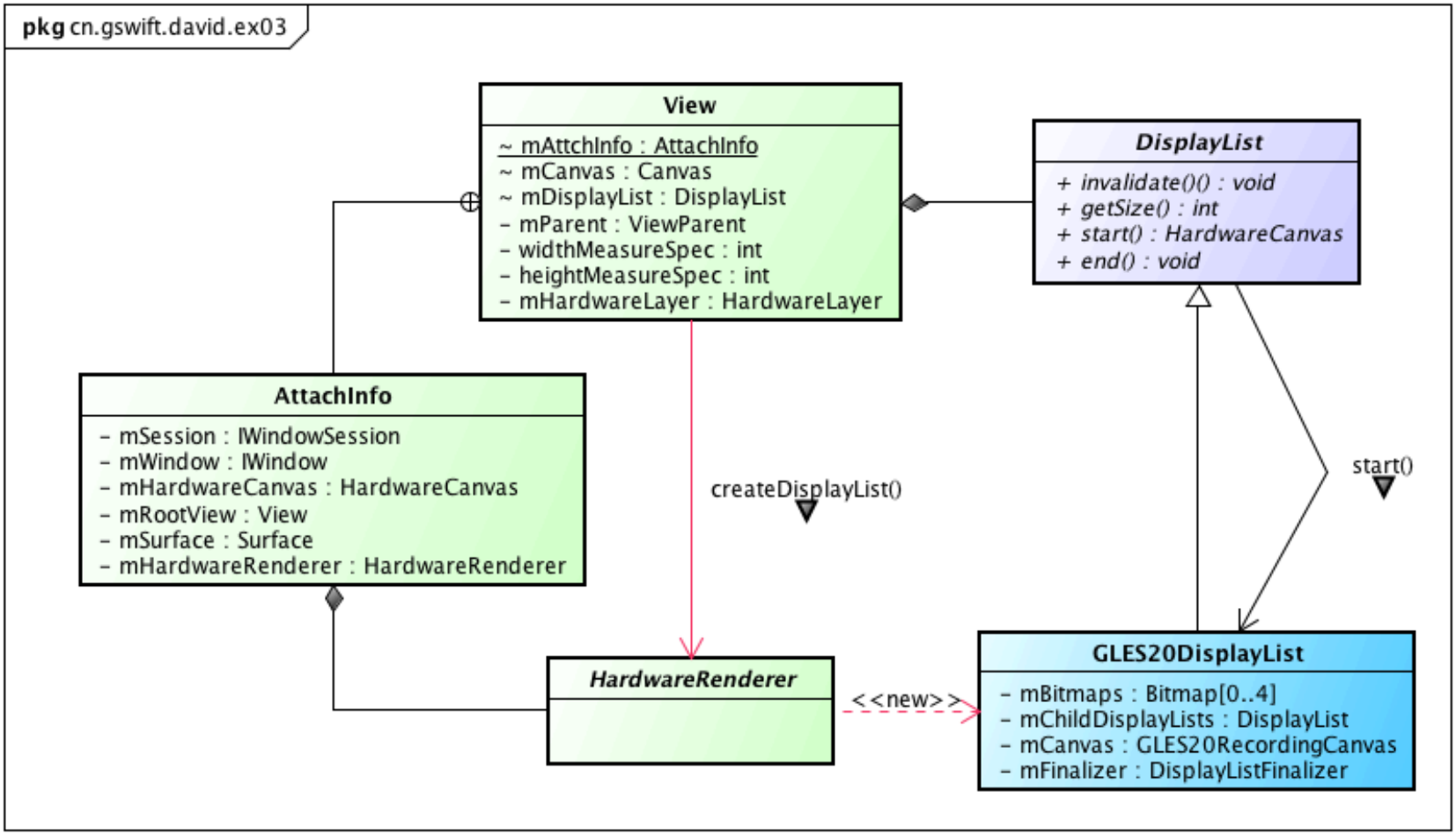
- A display list **records** a series of graphics related operation and can **replay** them later. Display lists are usually built by recording operations on a Canvas.
- Replaying the operations from a display list avoids executing views drawing code on every frame, and is thus much more efficient.

```
const char* DisplayList::OP_NAMES[] = {
    "Save",
    "Restore",
    "RestoreToCount",
    "SaveLayer",
    "SaveLayerAlpha",
    "Translate",
    "Rotate",
    "Scale",
    "Skew",
    "SetMatrix",
    "ConcatMatrix",
    "ClipRect",
    "DrawDisplayList",
    "DrawLayer",
    "DrawBitmap",
    "DrawBitmapMatrix",
    "DrawBitmapRect",
    "DrawBitmapData",
    "DrawBitmapMesh",
    "DrawPatch",
    "DrawColor",
    "DrawRect",
    "DrawRoundRect",
    "DrawCircle",
    "DrawOval",
    "DrawArc",
    "DrawPath",
    "DrawLines",
    "DrawPoints",
    "DrawTextOnPath",
    "DrawPosText",
    "DrawText",
                                    "ResetShader",
                                    "SetupShader",
                                    "ResetColorFilter",
                                    "SetupColorFilter",
                                    "ResetShadow",
                                    "SetupShadow",
                                    "ResetPaintFilter",
                                    "SetupPaintFilter",
                                    "DrawGLFunction"
                                };
```

graphics operations


Renderer Display List
replay
Display
1
2
3
4
5
6
7

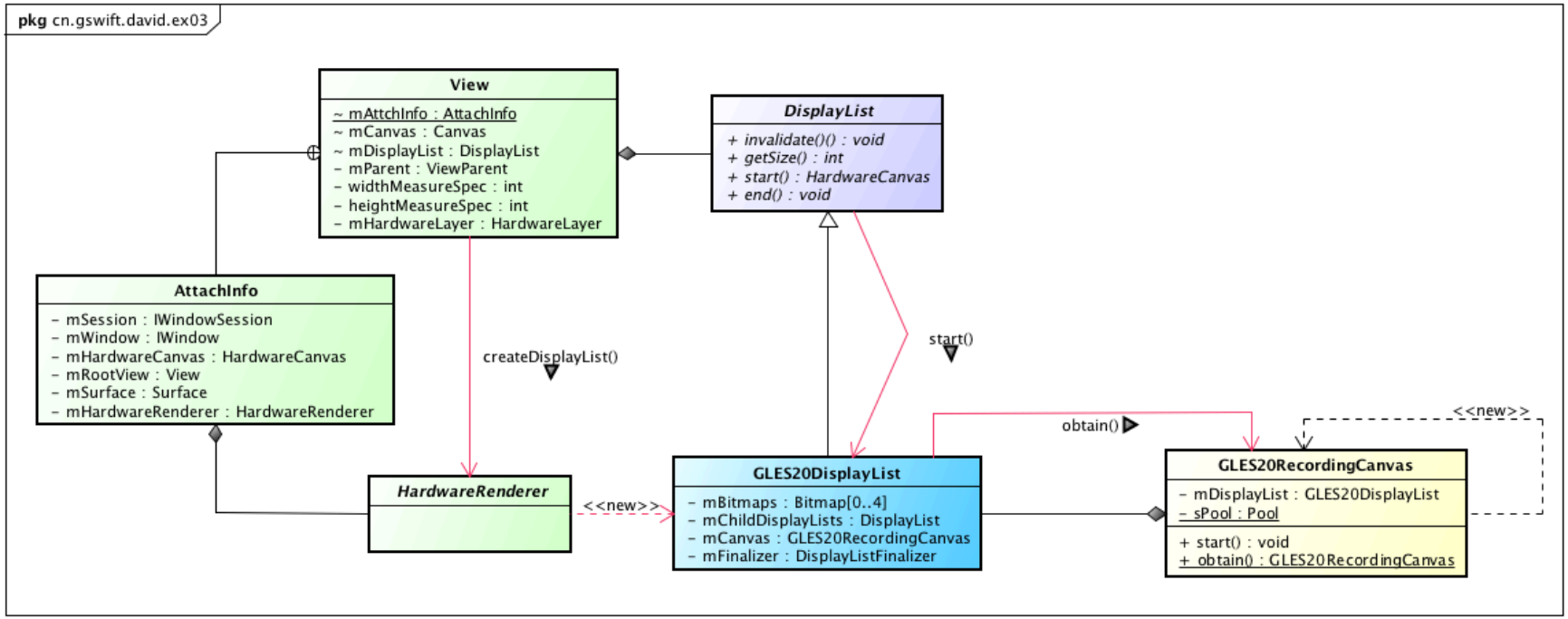
Create a GLES20DisplayList



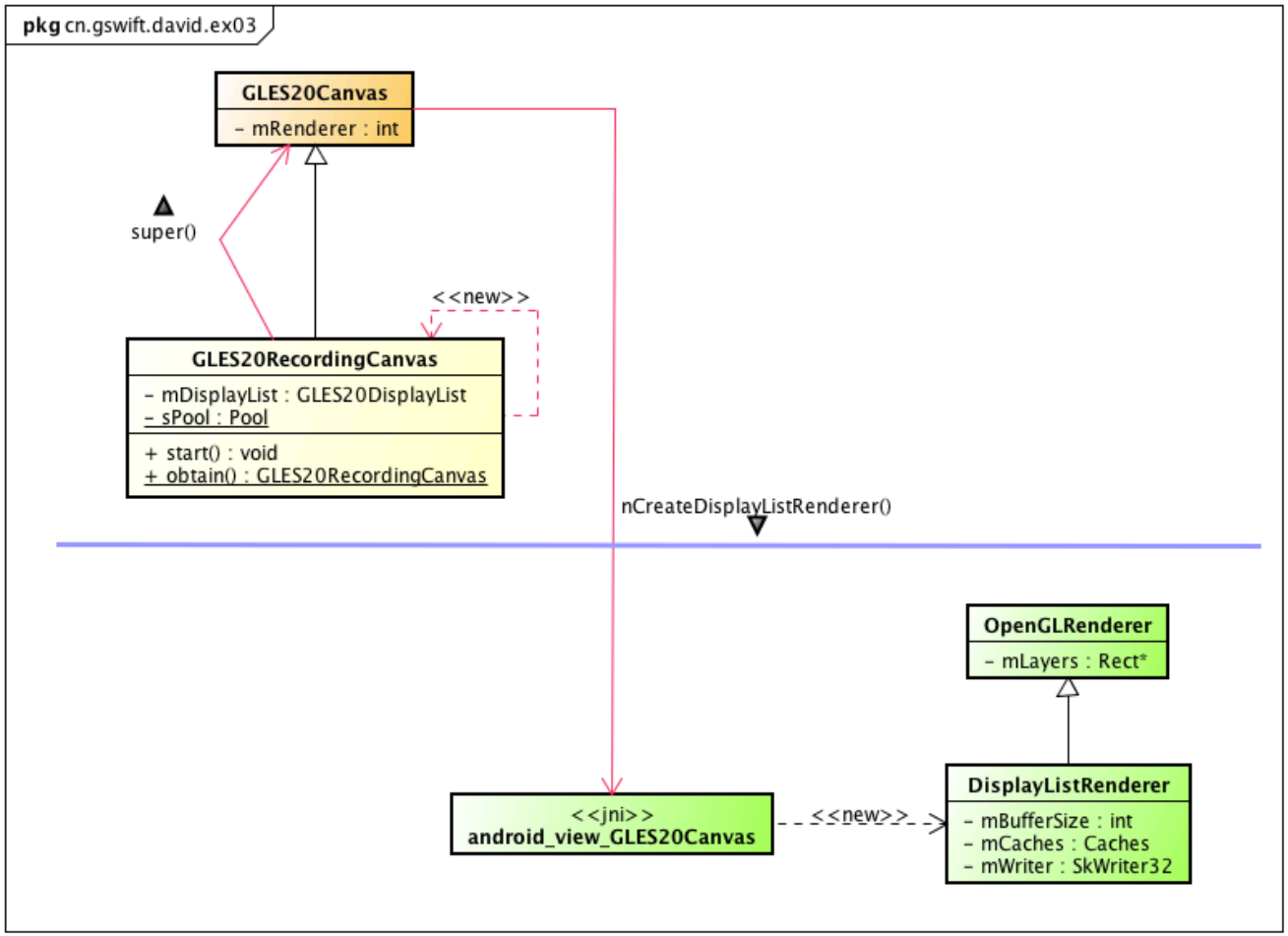
```
private DisplayList getDisplayList(DisplayList displayList, boolean isLayer) {
    if (!canHaveDisplayList()) {
        return null;
    }

    if (((mPrivateFlags & PFLAG_DRAWING_CACHE_VALID) == 0 ||
            displayList == null || !displayList.isValid() ||
            (!isLayer && mRecreateDisplayList))) {

        if (displayList == null) {
            final String name = getClass().getSimpleName();
            displayList = mAttachInfo.mHardwareRenderer.createDisplayList(name);
            // If we're creating a new display list, make sure our parent gets invalidated
            // since they will need to recreate their display list to account for this
            // new child display list.
            invalidateParentCaches();
        }
```

Create a DisplayList

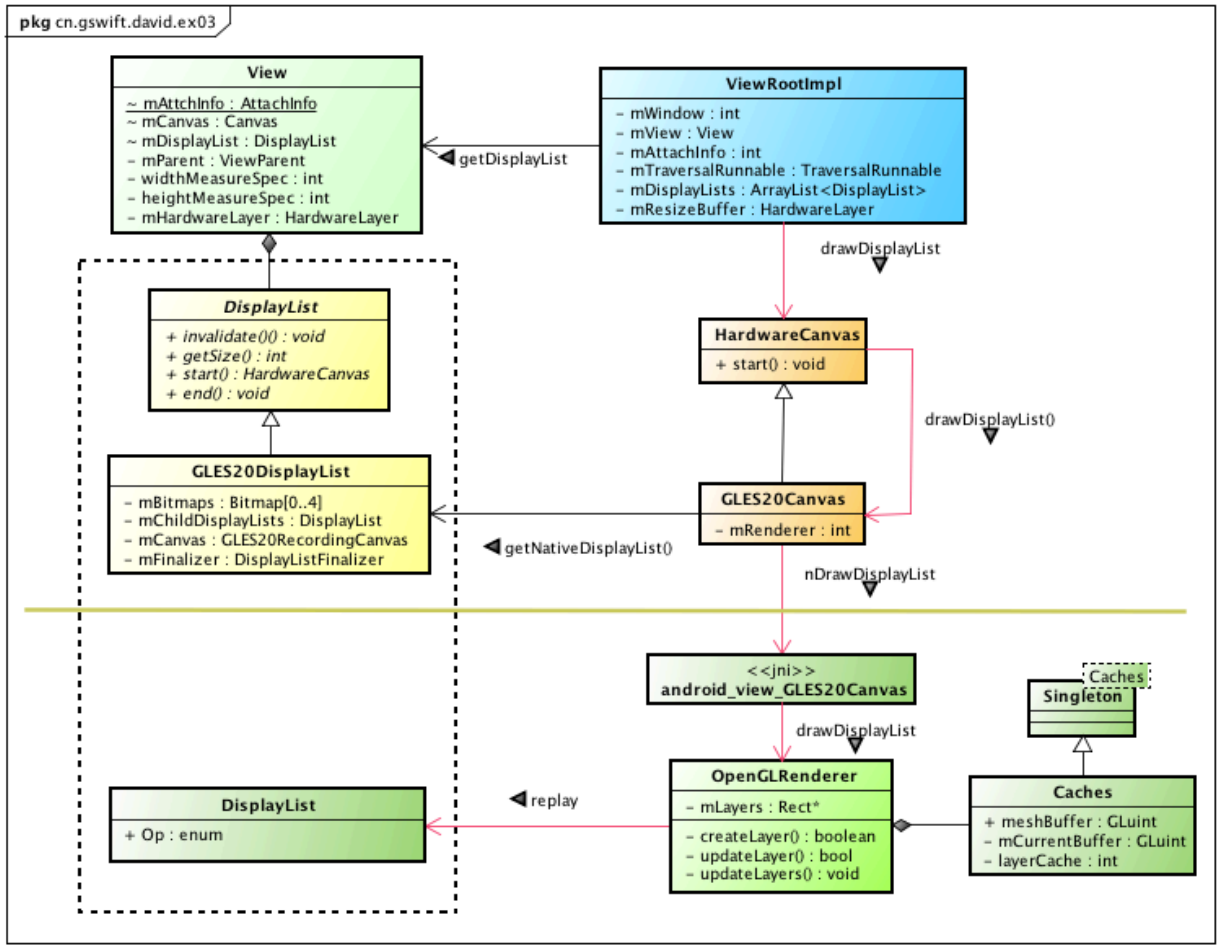
Create a GLESRecordingCanvas



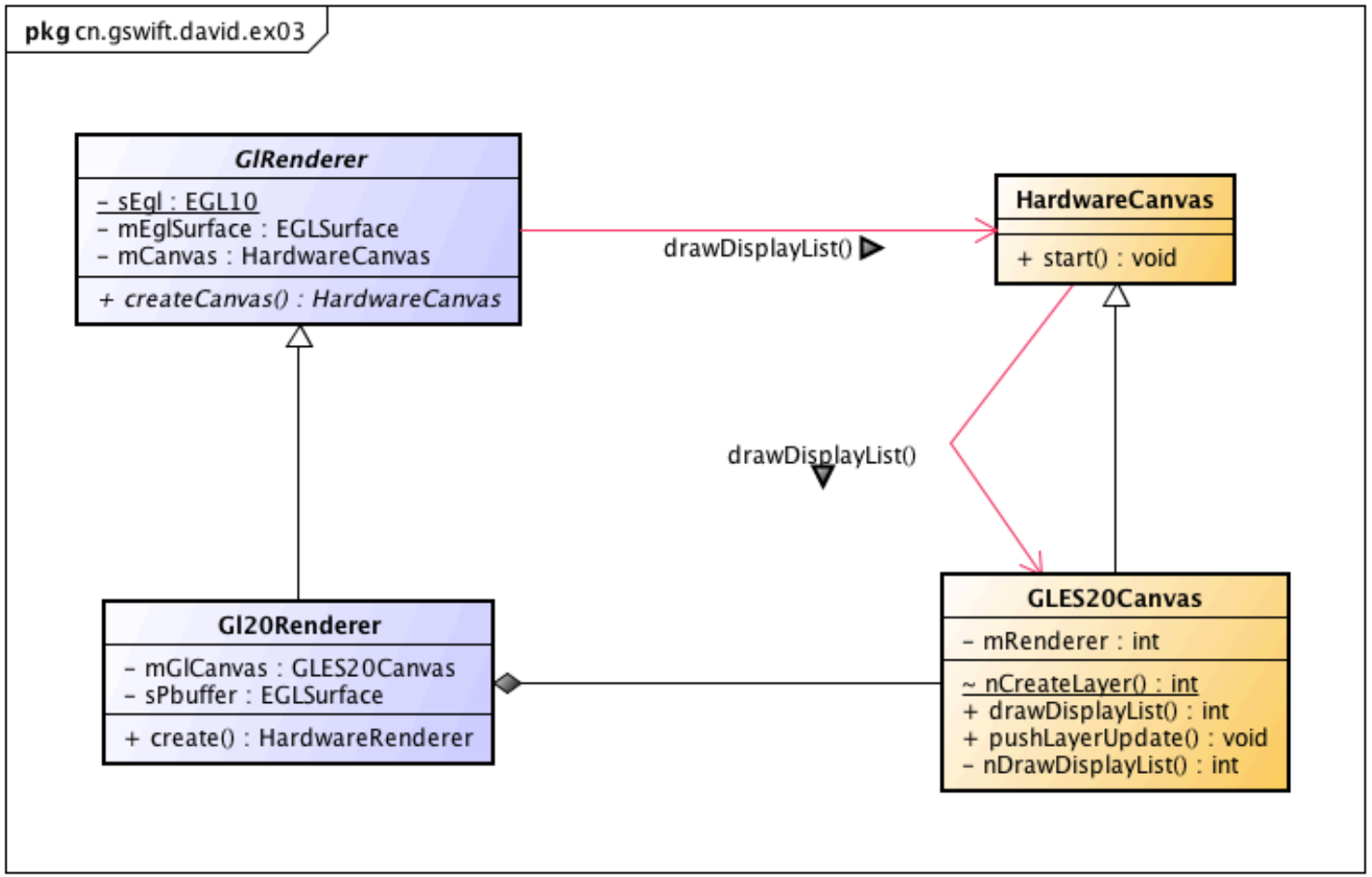
Create a native DisplayListRenderer



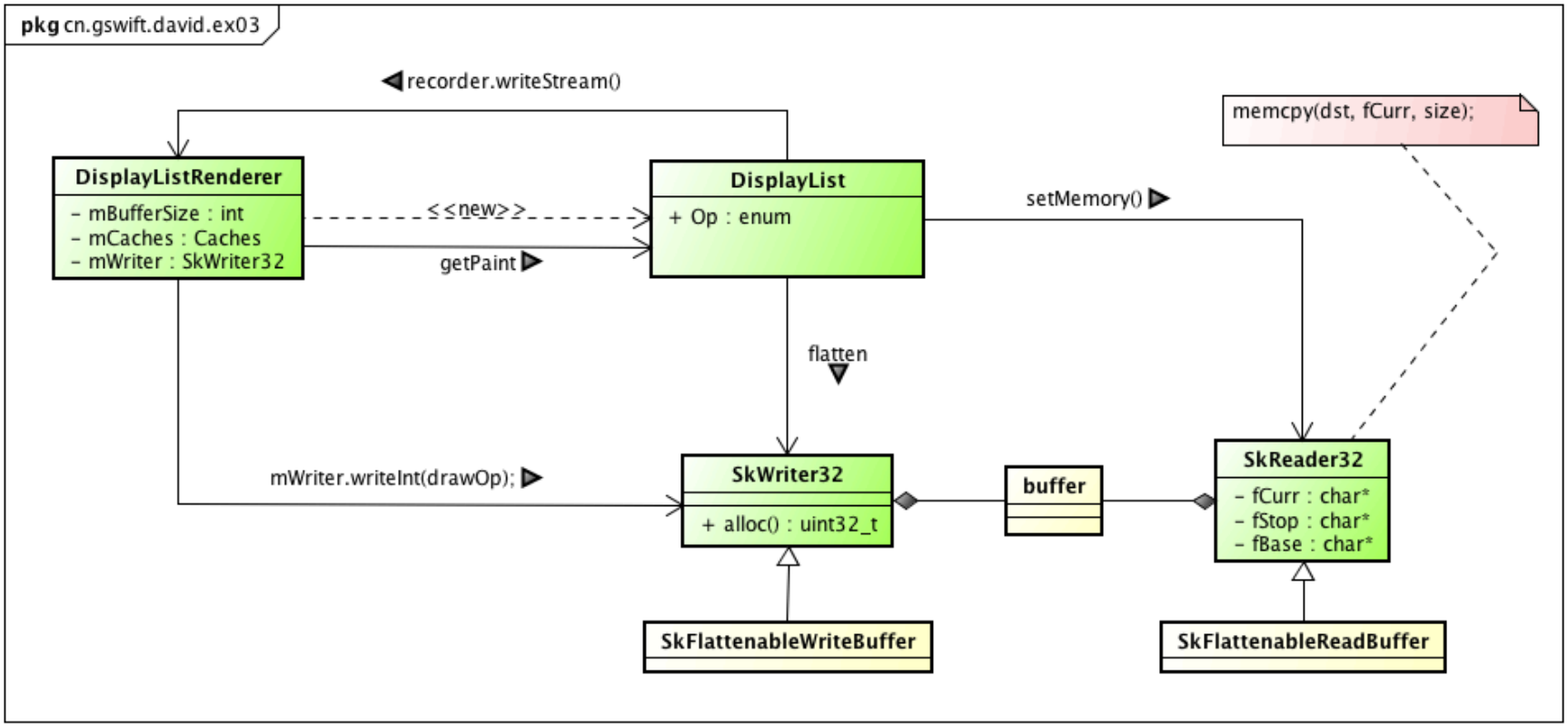
Native Draw display list



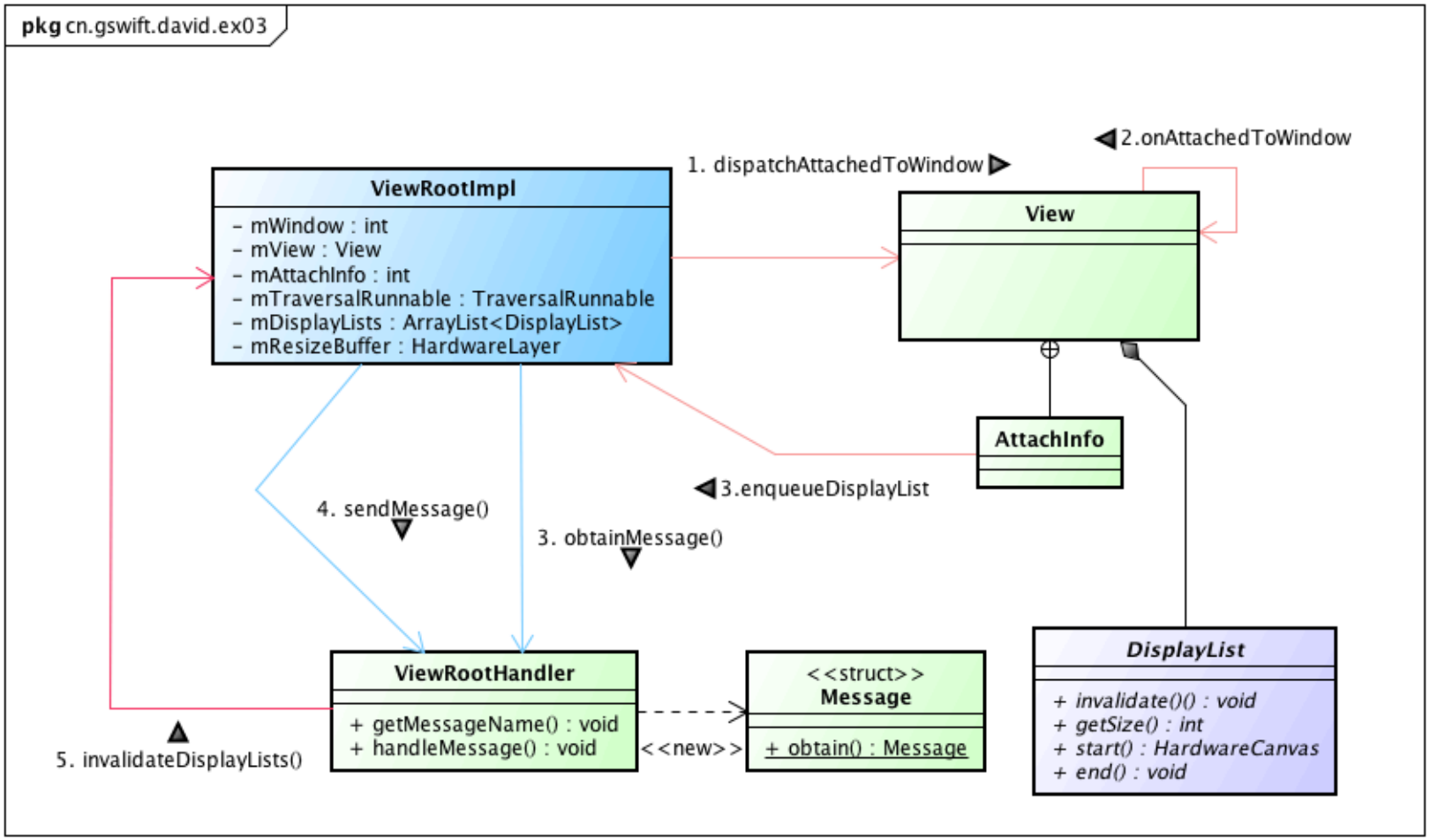
Draw display list with GLRenderer



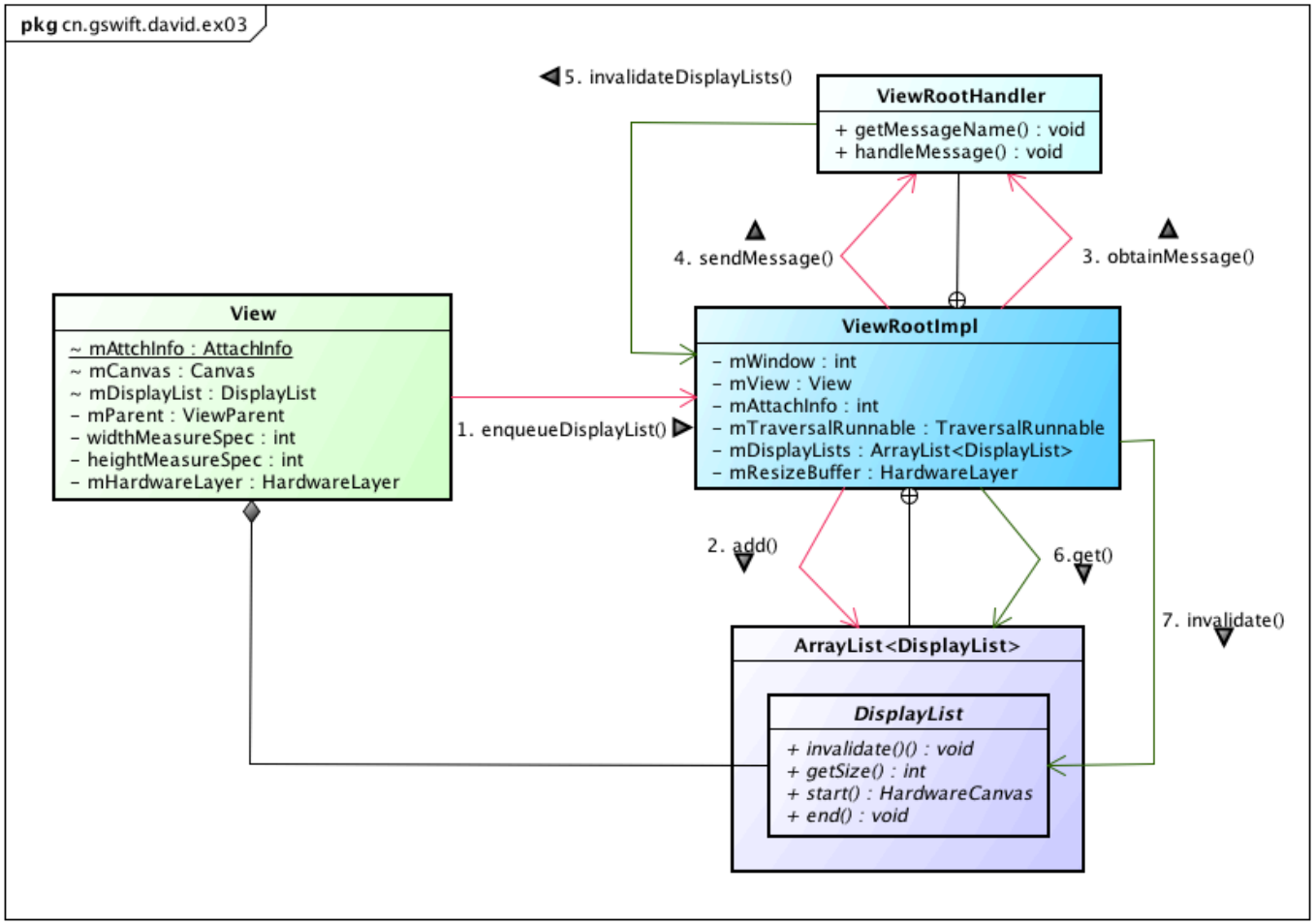
Native read and write display list



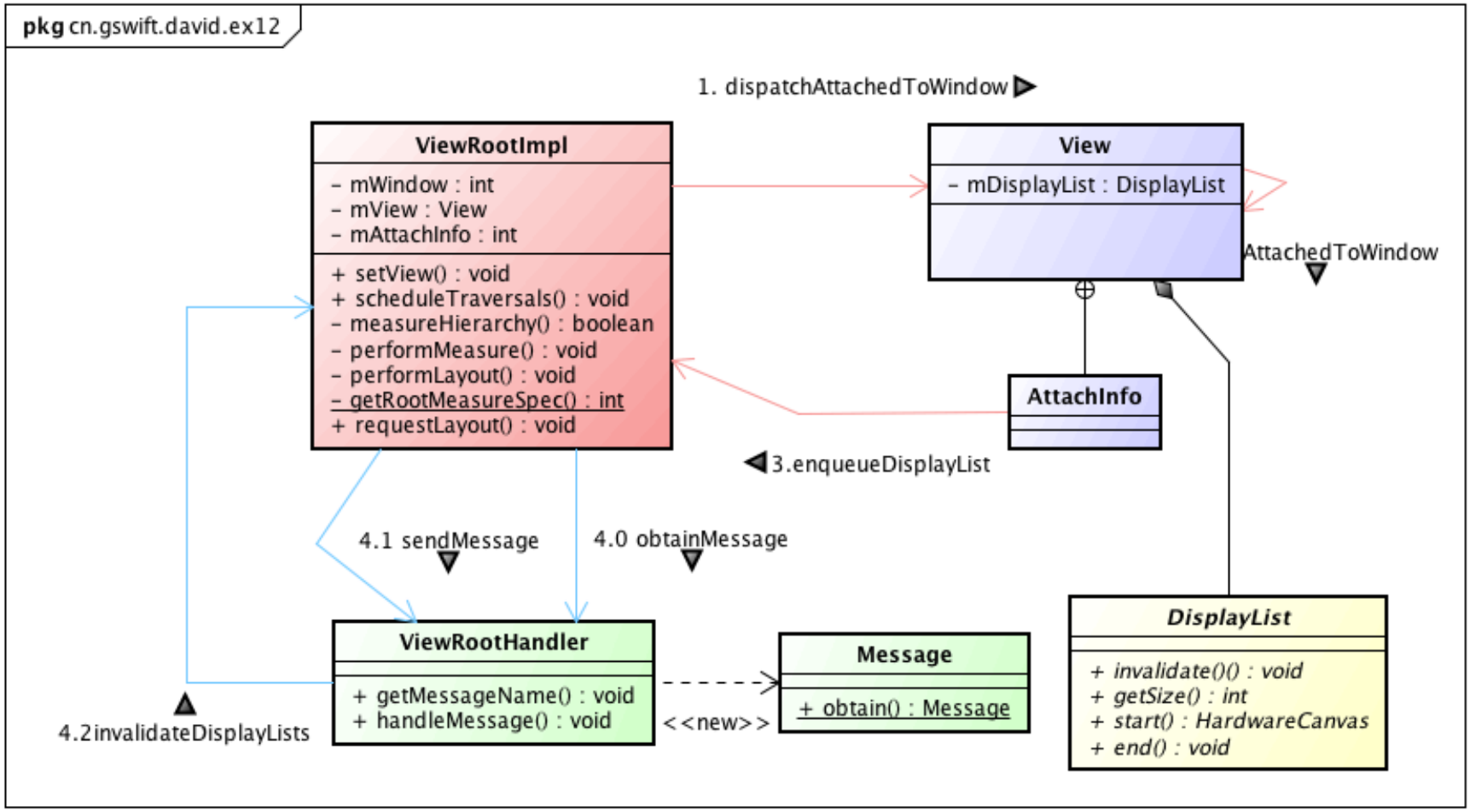
Attach the view to window

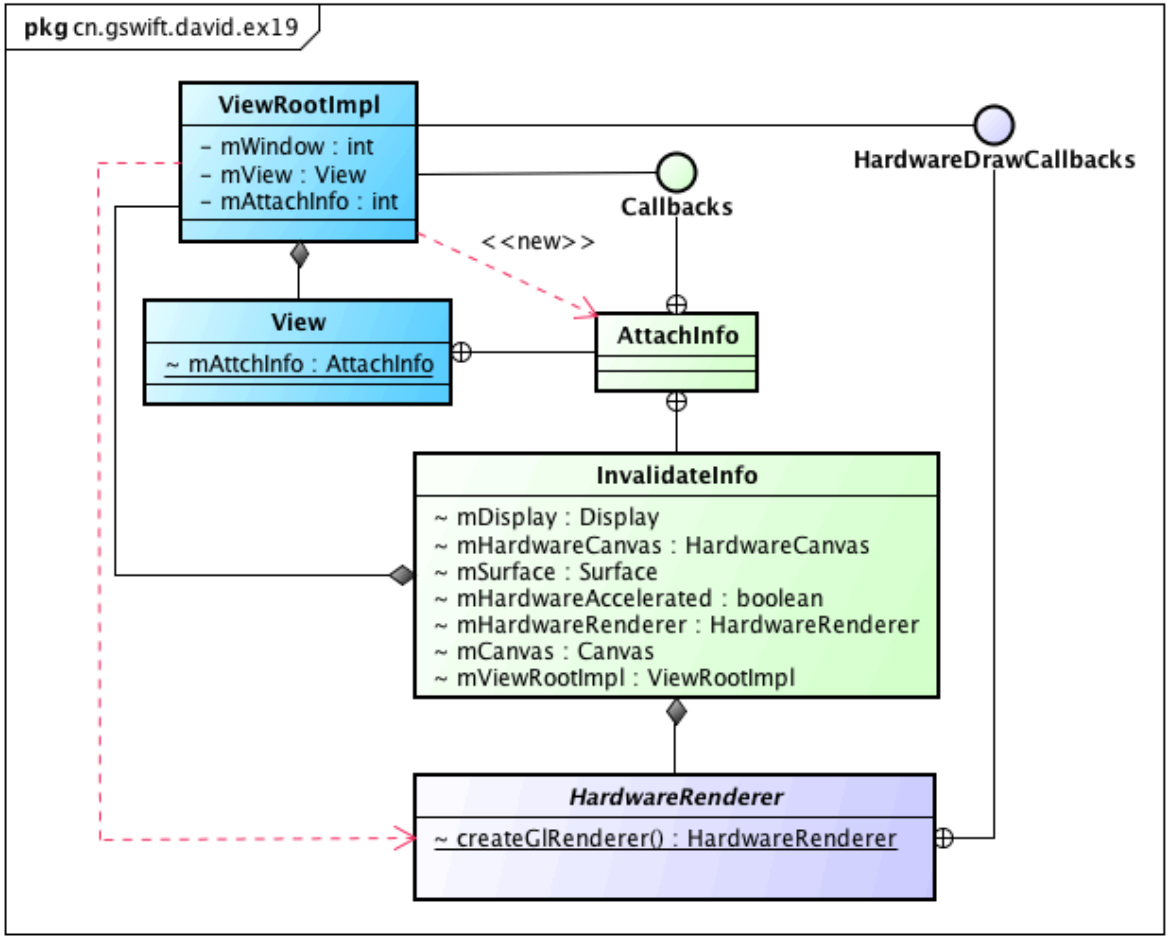


Enqueue Display list



Euqueue a DisplayList (1)




pkg cn.gswift.david.ex19
ViewRootImpl
- mWindow : int
- mView : View
- mAttachInfo : int
HardwareDrawCallbacks
Callbacks
<<new>>
View
~ mAttchInfo : AttachInfo
AttachInfo
InvalidateInfo
~ mDisplay : Display
~ mHardwareCanvas : HardwareCanvas
~ mSurface : Surface
~ mHardwareAccelerated : boolean
~ mHardwareRenderer : HardwareRenderer
~ mCanvas : Canvas
~ mViewRootImpl : ViewRootImpl
HardwareRenderer
~ createGlRenderer() : HardwareRenderer

View Layer

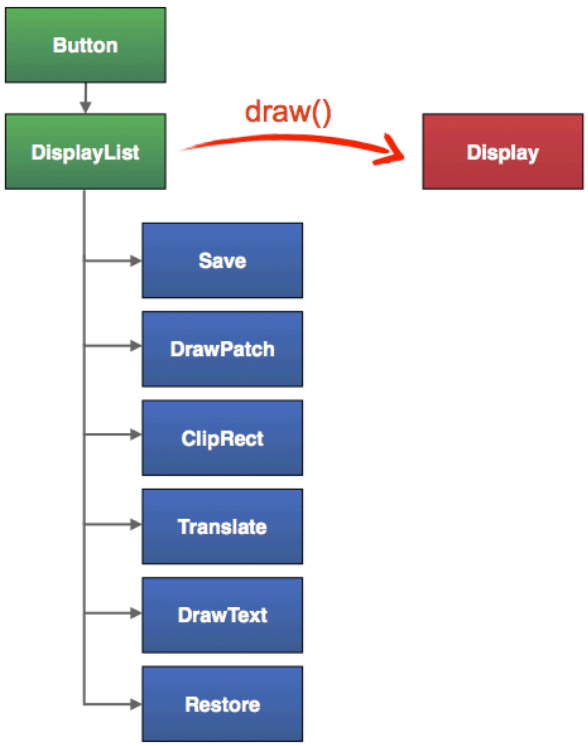
ANDROID RENDERER

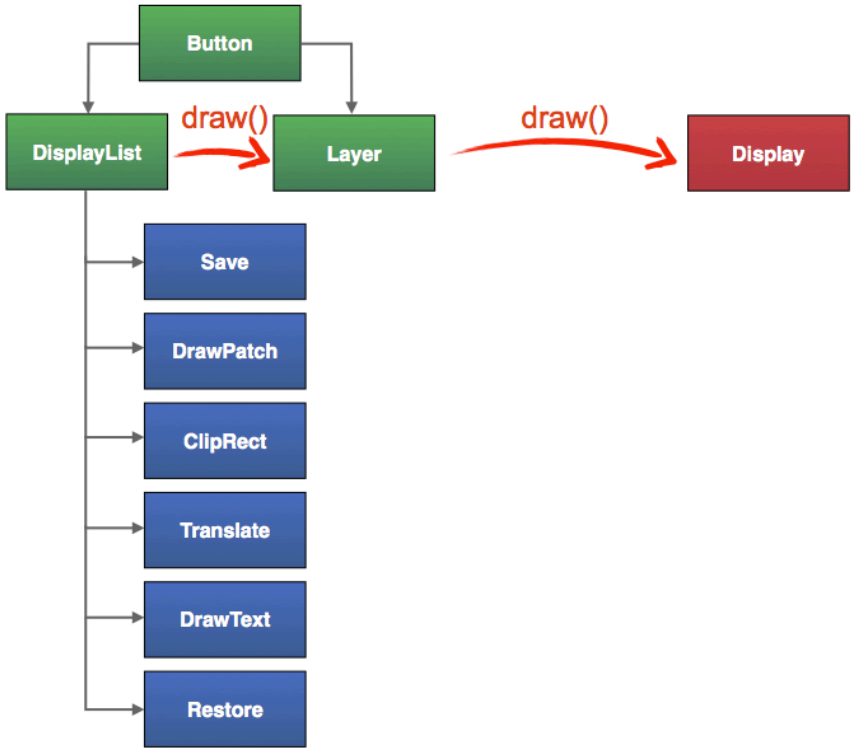
Layer Type

type	View is H/W-Accelerated	View is NOT H/W Accelerated
LAYER_TYPE_NONE	• Rendered in **H/W** • Directly into **surface**	• Rendered in **S/W** • Directly into **surface**
LAYER_TYPE_HARDWARE	• Rendered in **H/W** • Into **hardware texture**	• Rendered in **S/W** • Into **bitmap**
LAYER_TYPE_SOFTWARE	• Rendered in **S/W** • Into **bitmap**	• Rendered in **S/W** • Into **bitmap**

Hardware Layer

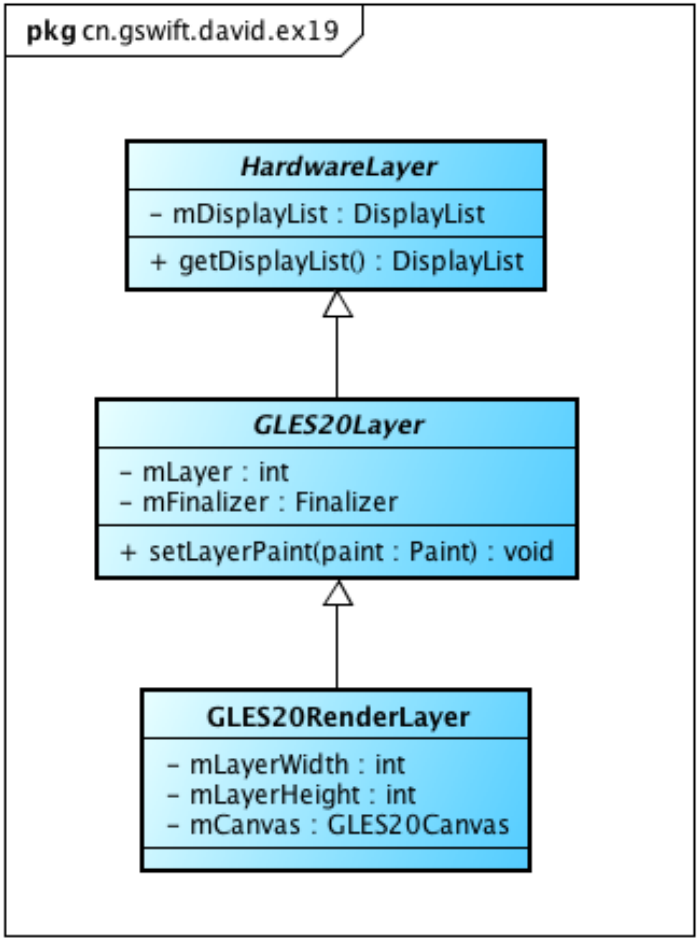
- A hardware layer can be used to render graphics operations into a hardware friendly buffer.
- For instance, with an OpenGL backend, a hardware layer would use a Frame Buffer Object (FBO.) The hardware layer can be used as a drawing cache when a complex set of graphics operations needs to be drawn several times.



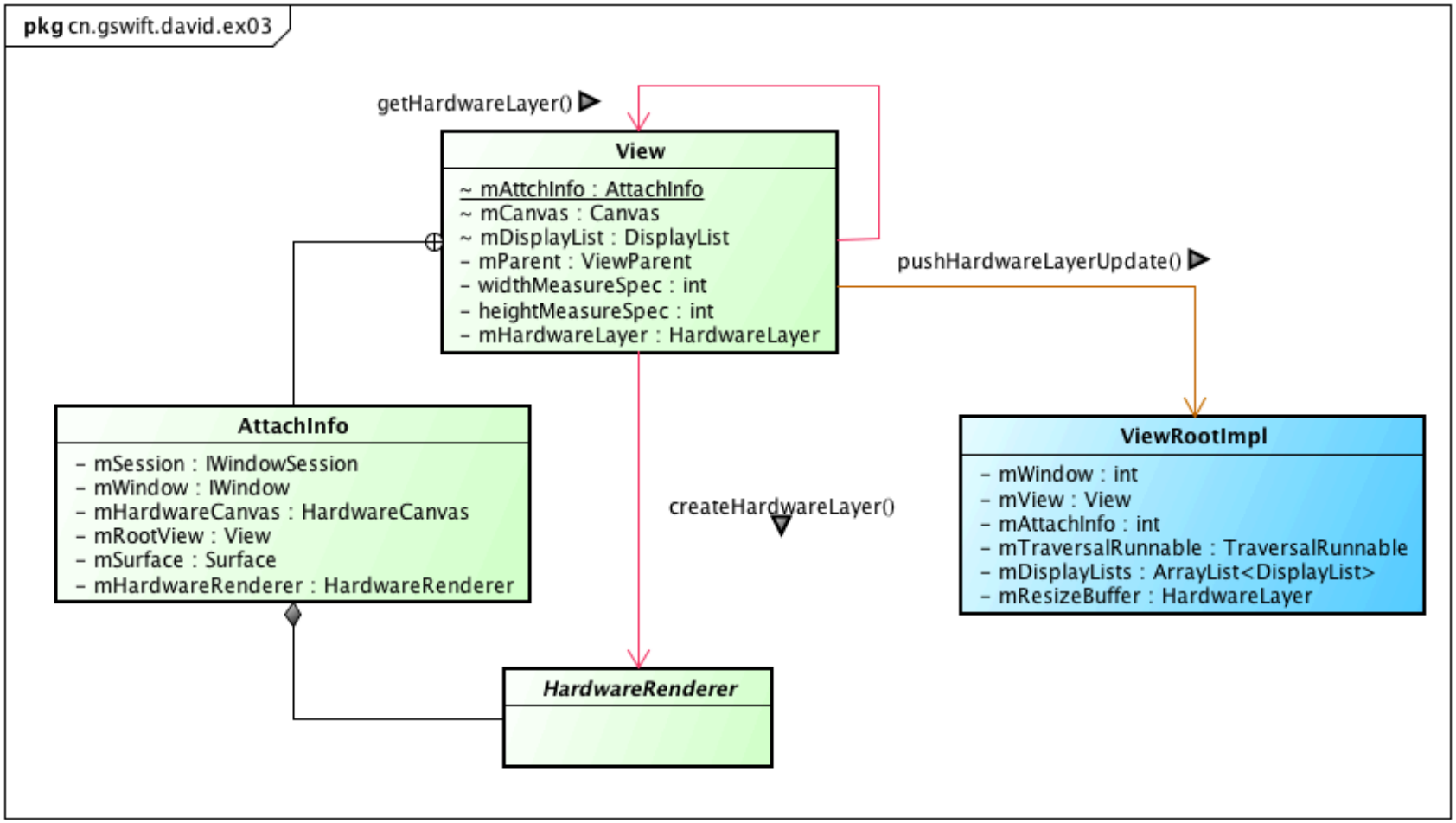


From:Google i/o

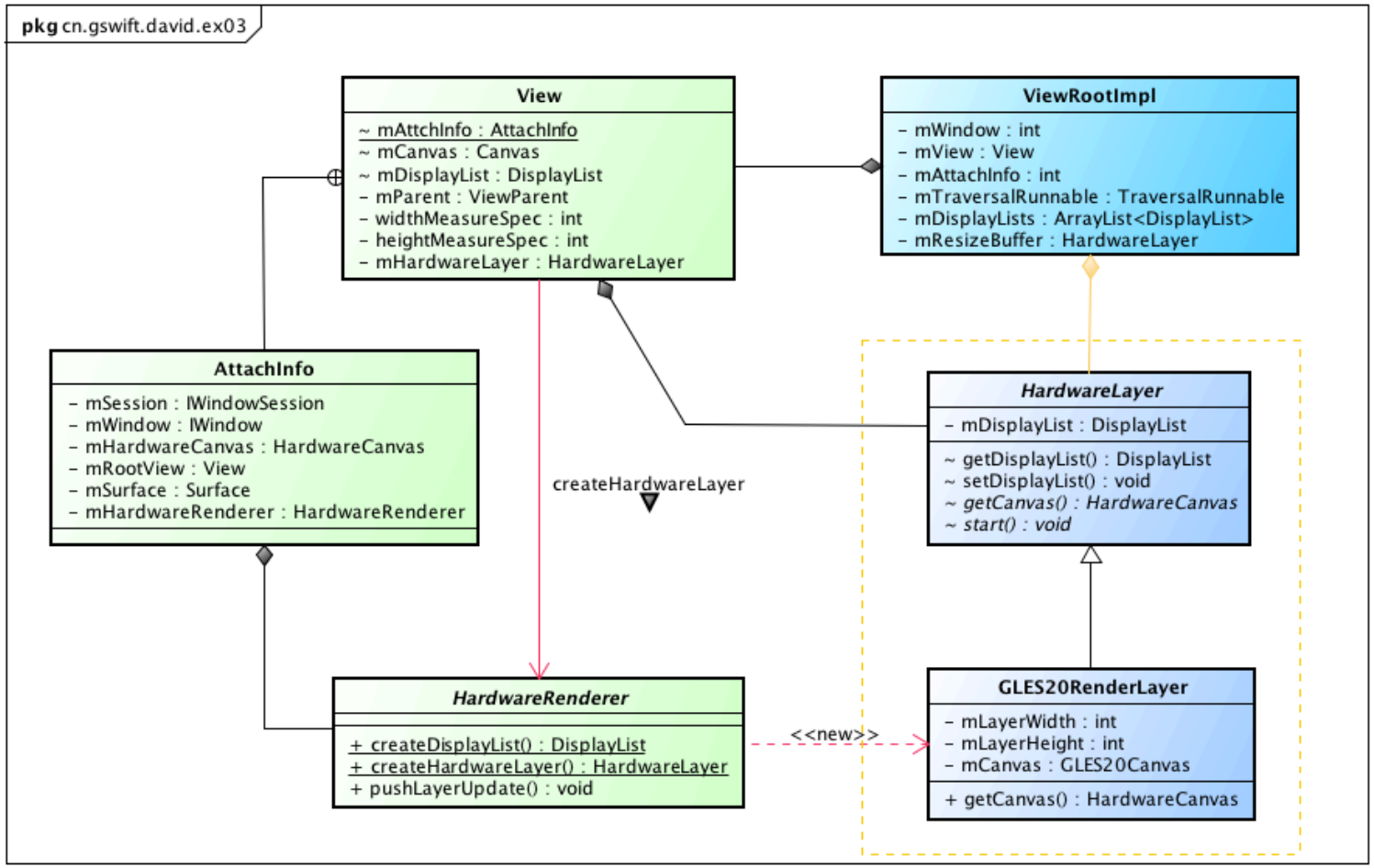
Java level: Layer



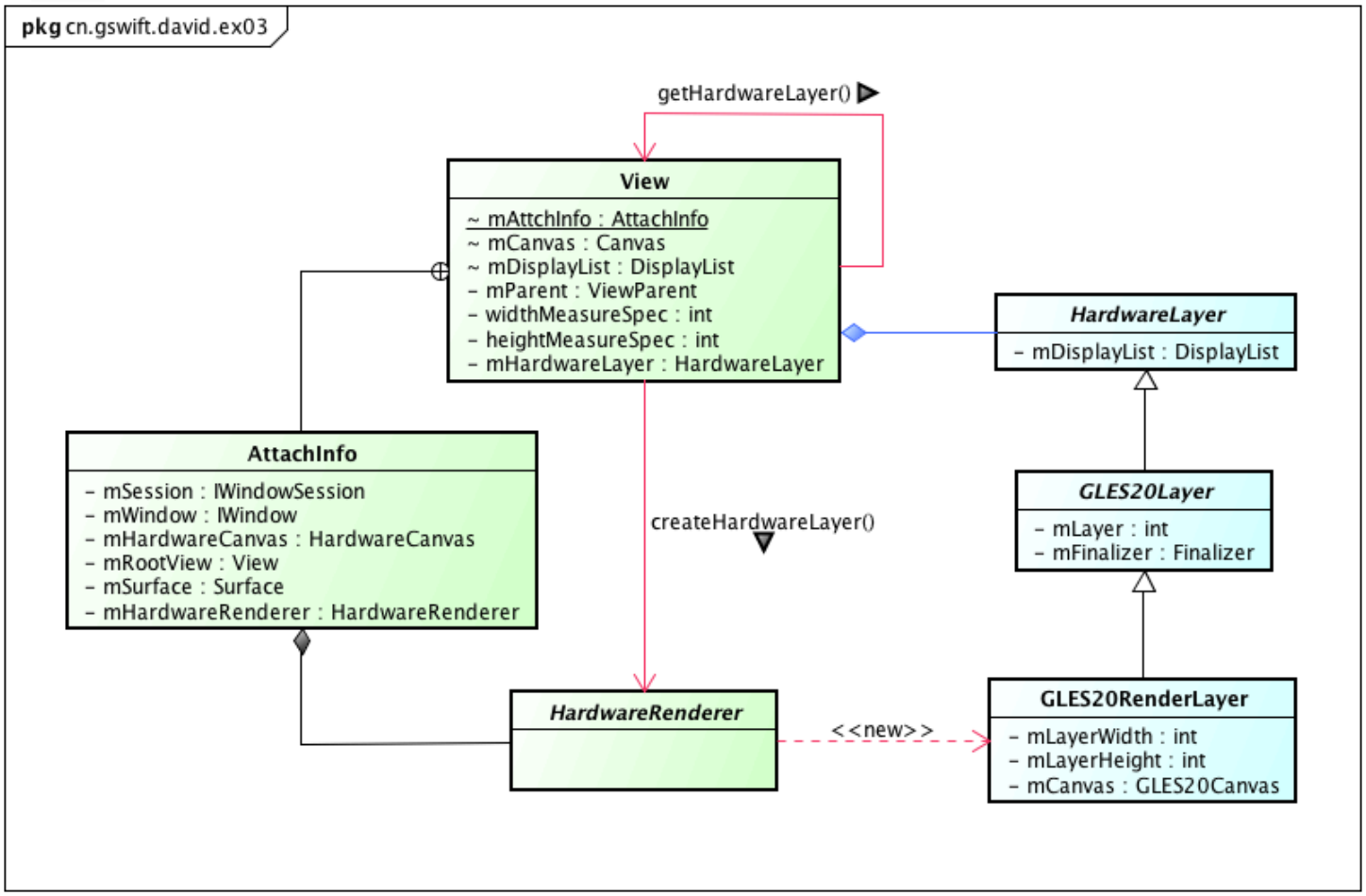
Hardware Layer



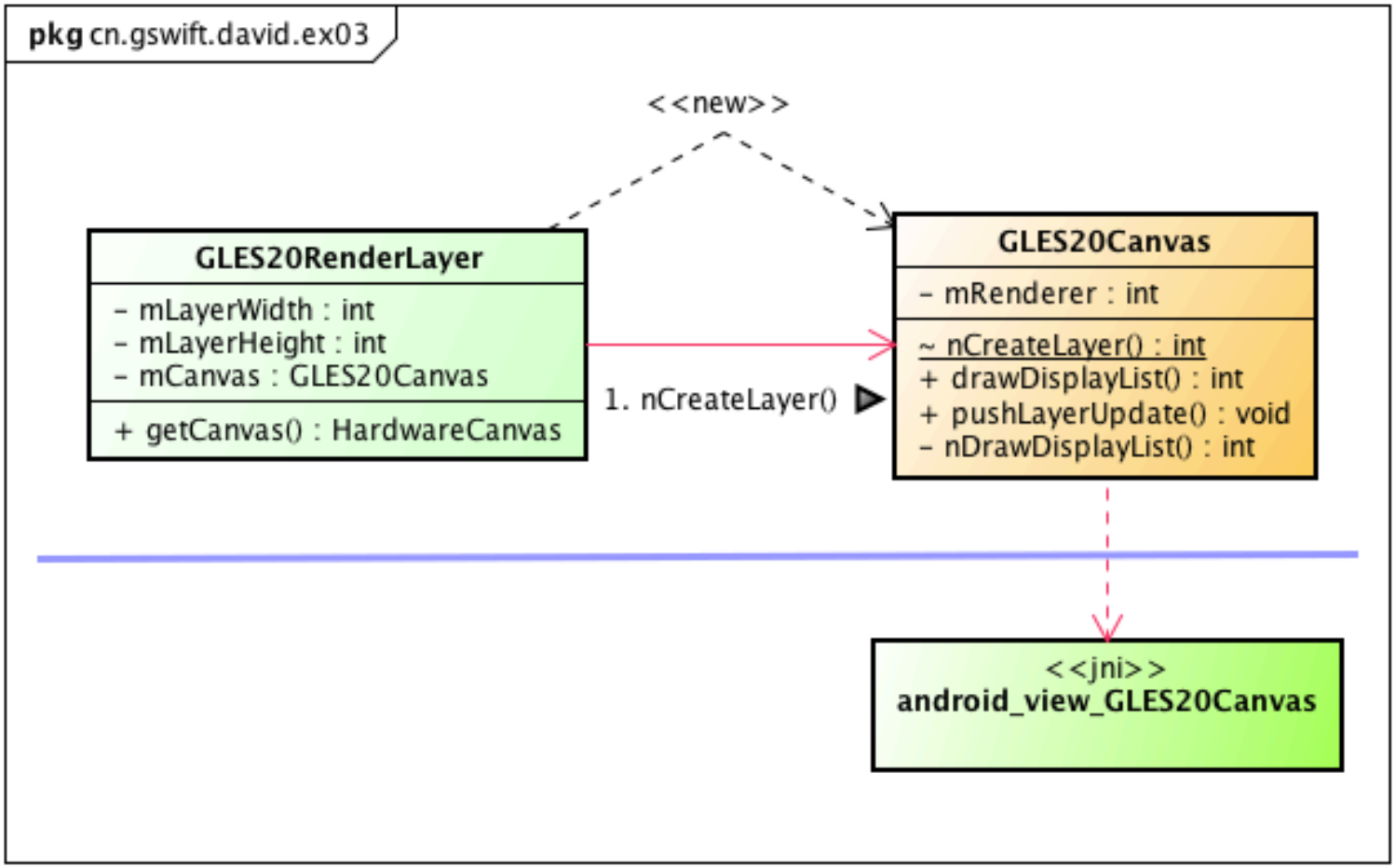
Create a GLES20RenderLayer(1)



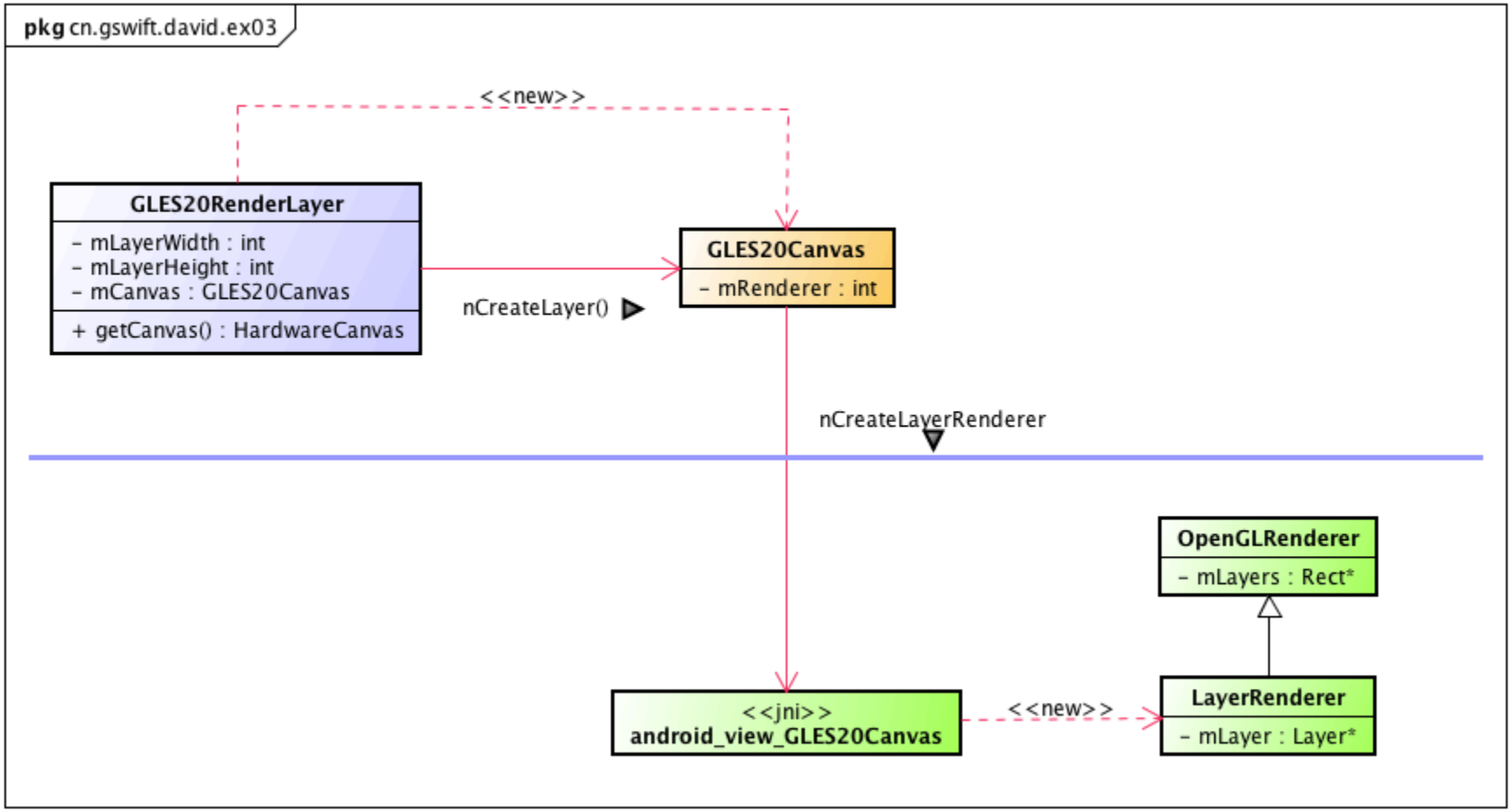
Create a GLES20RenderLayer(2)



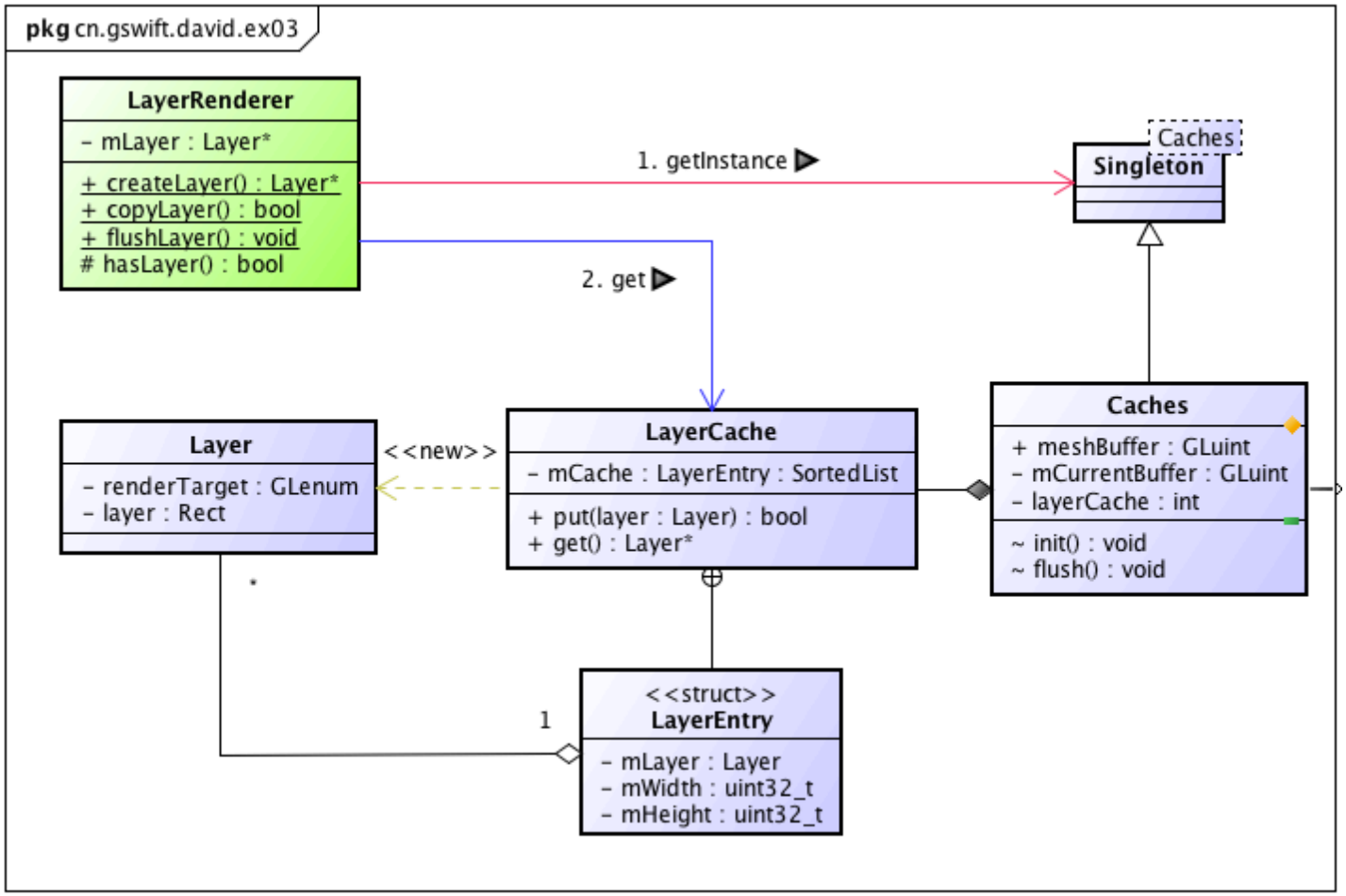
Create a native Layer



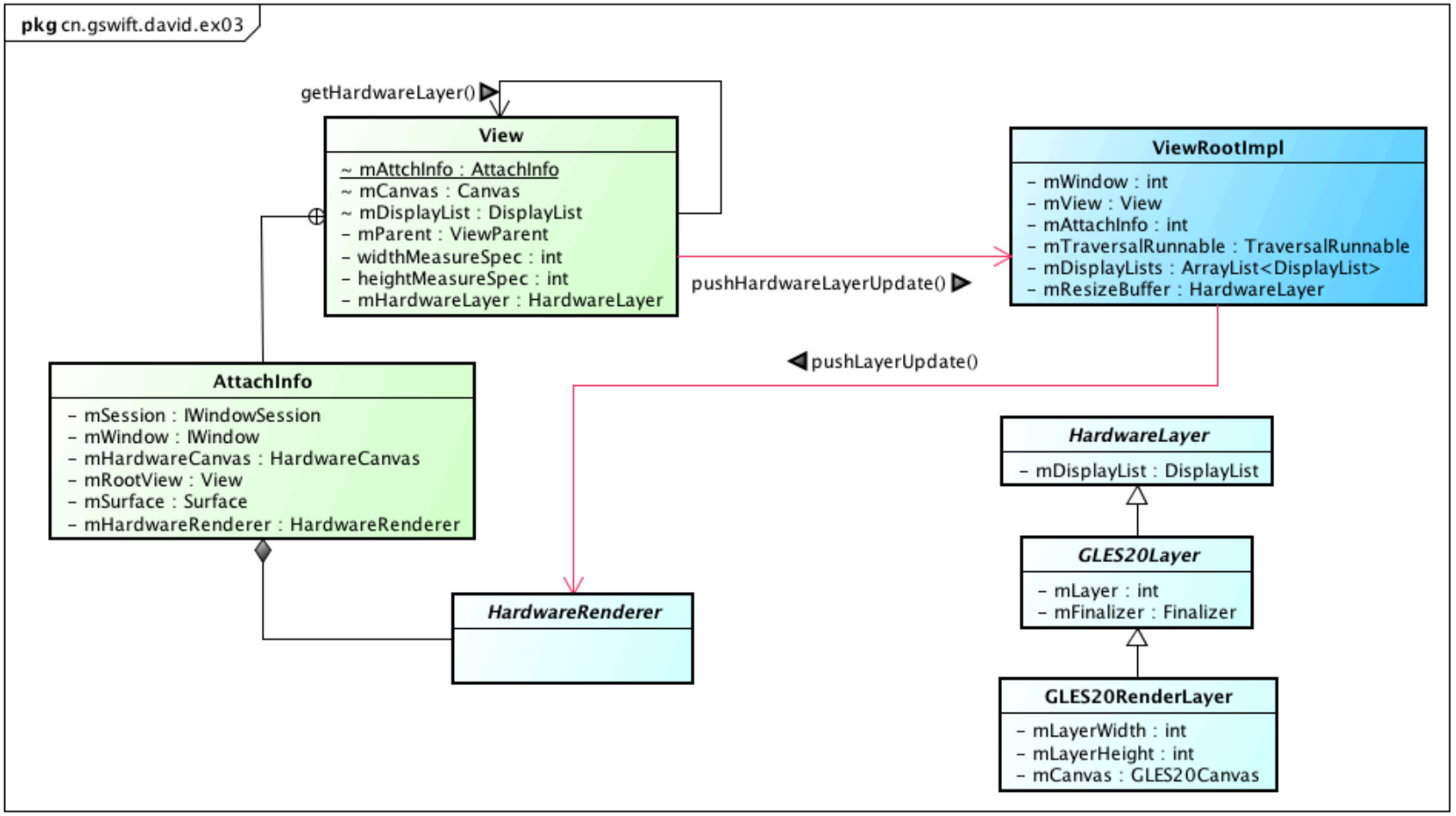
Create a native LayerRenderer



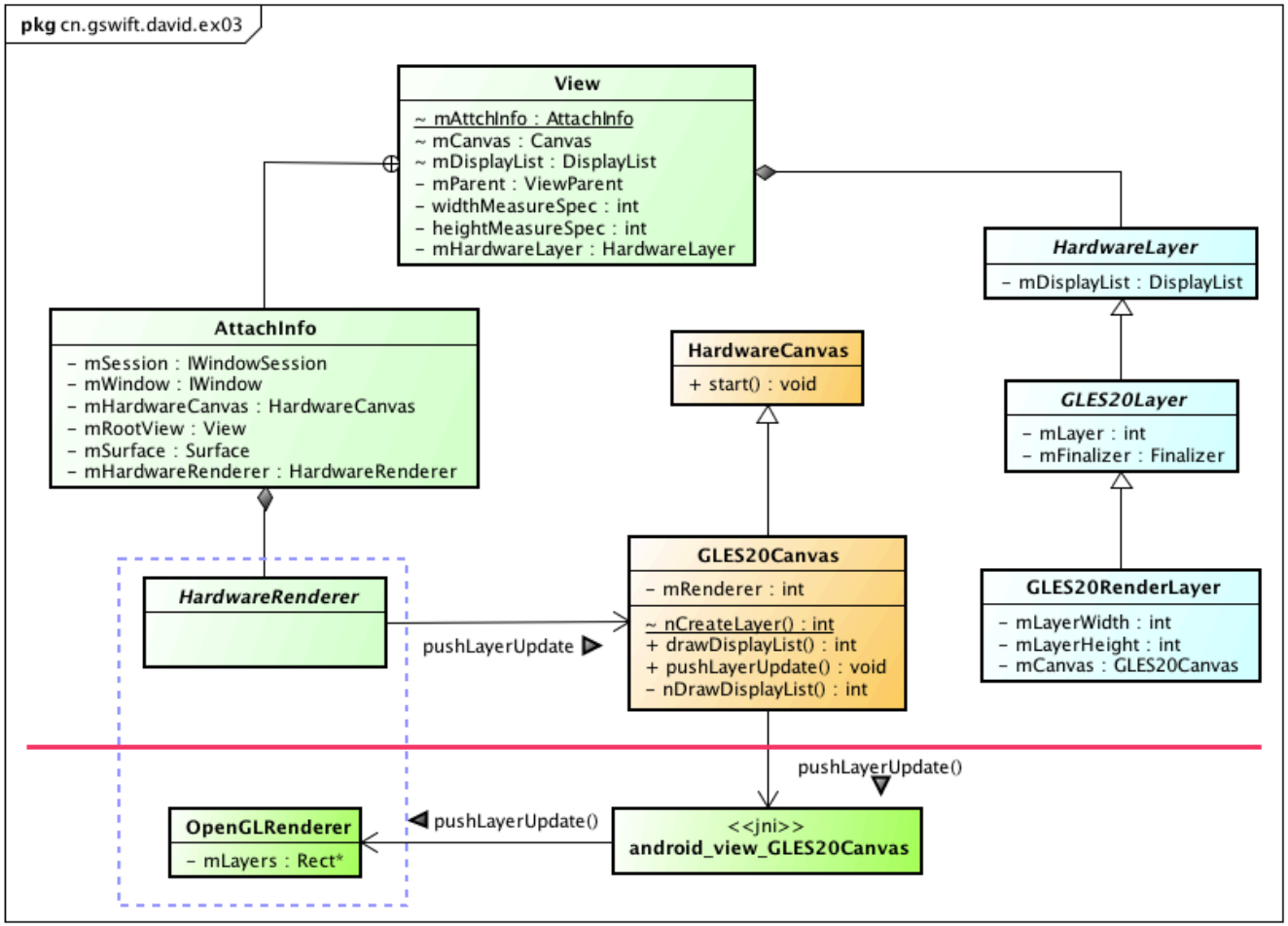
Create and get a native layer



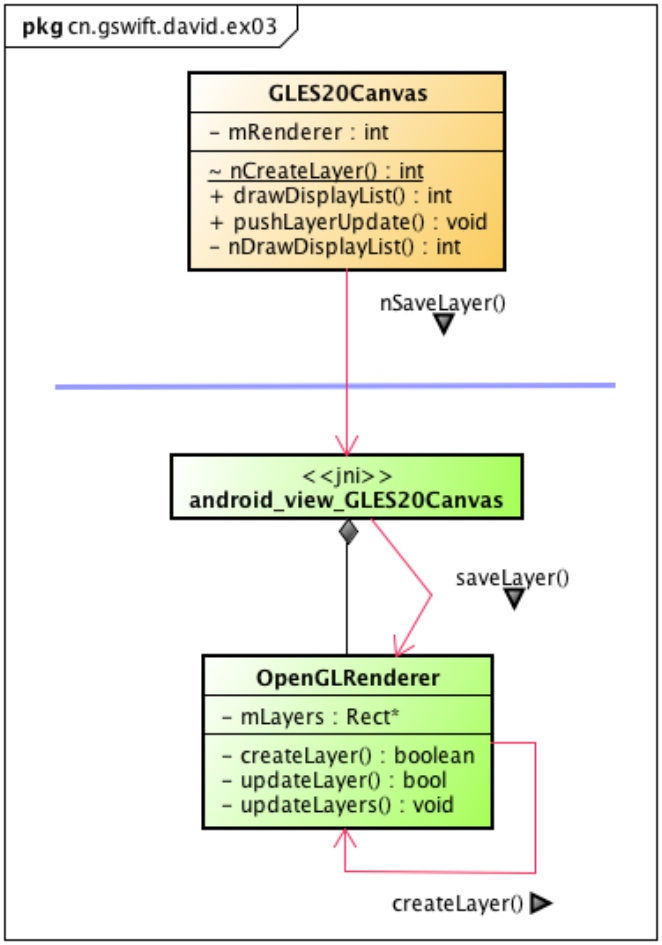
pushLayerUpdate



Native pushLayerUpdate



nSaveLayer


pkg cn.gswift.david.ex03
GLES20Canvas
- mRenderer : int
~ nCreateLayer() : int
+ drawDisplayList() : int
+ pushLayerUpdate() : void
- nDrawDisplayList() : int
nSaveLayer()
<<jni>>
android_view_GLES20Canvas
saveLayer()
OpenGLRenderer
- mLayers : Rect*
- createLayer() : boolean
- updateLayer() : bool
- updateLayers() : void
createLayer()

View.getDisplayList()

Prepare a Layer

No Layer,Prepare a bitmap

```
boolean caching = false;
final HardwareCanvas canvas = displayList.start();
int width = mRight - mLeft;
int height = mBottom - mTop;

try {
    canvas.setViewport(width, height);
    // The dirty rect should always be null for a display list
    canvas.onPreDraw(null);
    int layerType = getLayerType();
    if (!isLayer && layerType != LAYER_TYPE_NONE) {
        if (layerType == LAYER_TYPE_HARDWARE) {
            final HardwareLayer layer = getHardwareLayer();
            if (layer != null && layer.isValid()) {
                canvas.drawHardwareLayer(layer, 0, 0, mLayerPaint);
            } else {
                canvas.saveLayer(0, 0, mRight - mLeft, mBottom - mTop, mLayerPaint,
                        Canvas.HAS_ALPHA_LAYER_SAVE_FLAG |
                                Canvas.CLIP_TO_LAYER_SAVE_FLAG);
            }
            caching = true;
        } else {
            buildDrawingCache(true);
            Bitmap cache = getDrawingCache(true);
            if (cache != null) {
                canvas.drawBitmap(cache, 0, 0, mLayerPaint);
                caching = true;
            }
        }
    } else {

        computeScroll();

        canvas.translate(-mScrollX, -mScrollY);
        if (!isLayer) {
            mPrivateFlags |= PFLAG_DRAWN | PFLAG_DRAWING_CACHE_VALID;
            mPrivateFlags &= ~PFLAG_DIRTY_MASK;
        }

        // Fast path for layouts with no backgrounds
        if ((mPrivateFlags & PFLAG_SKIP_DRAW) == PFLAG_SKIP_DRAW) {
            dispatchDraw(canvas);
        } else {
            draw(canvas);
        }
    }
} finally {
    canvas.onPostDraw();

    displayList.end();
    displayList.setCaching(caching);
    if (isLayer) {
        displayList.setLeftTopRightBottom(0, 0, width, height);
    } else {
        setDisplayListProperties(displayList);
    }
}
```

This is called when the view is attached to a window. At this point it has a Surface and will start drawing. Note that this function is guaranteed to be called before onDraw(android.graphics.Canvas), however it may be called any time before the first onDraw – including before or after onMeasure(int, int). See onDetachedFromWindow()

```
protected void onAttachedToWindow() {
    if ((mPrivateFlags & PFLAG_REQUEST_TRANSPARENT_REGIONS) != 0) {
        mParent.requestTransparentRegion(this);
    }

    if ((mPrivateFlags & PFLAG_AWAKEN_SCROLL_BARS_ON_ATTACH) != 0) {
        initialAwakenScrollBars();
        mPrivateFlags &= ~PFLAG_AWAKEN_SCROLL_BARS_ON_ATTACH;
    }

    jumpDrawablesToCurrentState();

    clearAccessibilityFocus();
    if (isFocused()) {
        InputMethodManager imm = InputMethodManager.peekInstance();
        imm.focusIn(this);
    }

    if (mAttachInfo != null && mDisplayList != null) {
        mAttachInfo.mViewRootImpl.dequeueDisplayList(mDisplayList);
    }
}
```

```java
/**
 * This is called when the view is detached from a window.  At this point it
 * no longer has a surface for drawing.
 *
 * @see #onAttachedToWindow()
 */
protected void onDetachedFromWindow() {
    mPrivateFlags &= ~PFLAG_CANCEL_NEXT_UP_EVENT;

    removeUnsetPressCallback();
    removeLongPressCallback();
    removePerformClickCallback();
    removeSendViewScrolledAccessibilityEventCallback();

    destroyDrawingCache();

    destroyLayer(false);

    if (mAttachInfo != null) {
        if (mDisplayList != null) {
            mAttachInfo.mViewRootImpl.enqueueDisplayList(mDisplayList);
        }
        mAttachInfo.mViewRootImpl.cancelInvalidate(this);
    } else {
        // Should never happen
        clearDisplayList();
    }

    mCurrentAnimation = null;

    resetRtlProperties();
    onRtlPropertiesChanged(LAYOUT_DIRECTION_DEFAULT);
    resetAccessibilityStateChanged();
}
```

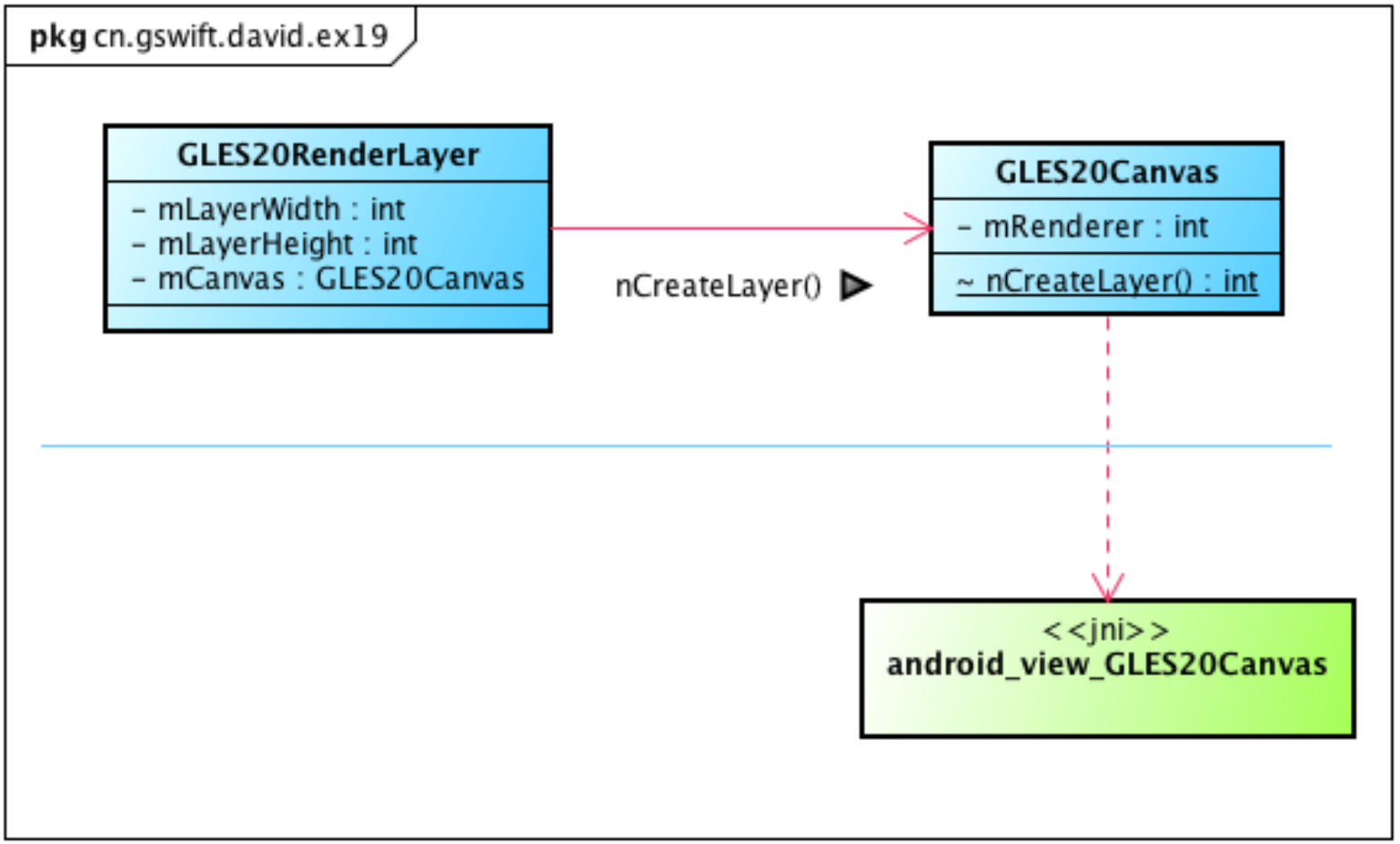
View.draw()

```
DisplayList displayList = null;
Bitmap cache = null;
boolean hasDisplayList = false;
if (caching) {
    if (!hardwareAccelerated) {
        if (layerType != LAYER_TYPE_NONE) {
            layerType = LAYER_TYPE_SOFTWARE;
            buildDrawingCache(true);
        }
        cache = getDrawingCache(true);
    } else {
        switch (layerType) {
            case LAYER_TYPE_SOFTWARE:
                if (useDisplayListProperties) {
                    hasDisplayList = canHaveDisplayList();
                } else {
                    buildDrawingCache(true);
                    cache = getDrawingCache(true);
                }
                break;
            case LAYER_TYPE_HARDWARE:
                if (useDisplayListProperties) {
                    hasDisplayList = canHaveDisplayList();
                }
                break;
            case LAYER_TYPE_NONE:
                // Delay getting the display list until animation-driven alpha values are
                // set up and possibly passed on to the view
                hasDisplayList = canHaveDisplayList();
                break;
        }
    }
}
```

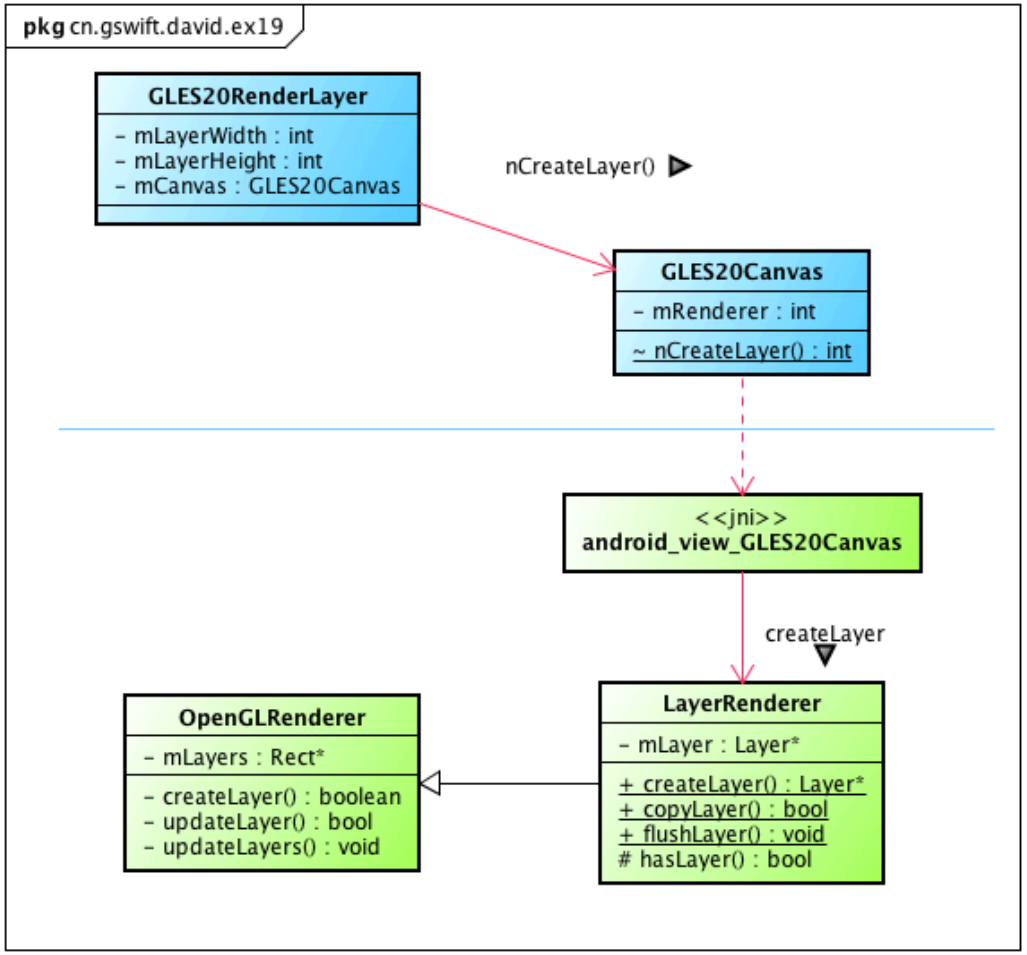
```
if (hasNoCache) {
    boolean layerRendered = false;
    if (layerType == LAYER_TYPE_HARDWARE && !useDisplayListProperties) {
        final HardwareLayer layer = getHardwareLayer();
        if (layer != null && layer.isValid()) {
            mLayerPaint.setAlpha((int) (alpha * 255));
            ((HardwareCanvas) canvas).drawHardwareLayer(layer, 0, 0, mLayerPaint);
            layerRendered = true;
        } else {
            final int scrollX = hasDisplayList ? 0 : sx;
            final int scrollY = hasDisplayList ? 0 : sy;
            canvas.saveLayer(scrollX, scrollY,
                    scrollX + mRight - mLeft, scrollY + mBottom - mTop, mLayerPaint,
                    Canvas.HAS_ALPHA_LAYER_SAVE_FLAG | Canvas.CLIP_TO_LAYER_SAVE_FLAG);
        }
    }

    if (!layerRendered) {
        if (!hasDisplayList) {
            // Fast path for layouts with no backgrounds
            if ((mPrivateFlags & PFLAG_SKIP_DRAW) == PFLAG_SKIP_DRAW) {
                mPrivateFlags &= ~PFLAG_DIRTY_MASK;
                dispatchDraw(canvas);
            } else {
                draw(canvas);
            }
        } else {
            mPrivateFlags &= ~PFLAG_DIRTY_MASK;
            ((HardwareCanvas) canvas).drawDisplayList(displayList, null, flags);
        }
    }
} else if (cache != null) {
```

```
// TODO: should handle create/resize failure
layerCanvas = mResizeBuffer.start(hwRendererCanvas);
layerCanvas.setViewport(mWidth, mHeight);
layerCanvas.onPreDraw(null);
final int restoreCount = layerCanvas.save();

int yoff;
final boolean scrolling = mScroller != null
        && mScroller.computeScrollOffset();
if (scrolling) {
    yoff = mScroller.getCurrY();
    mScroller.abortAnimation();
} else {
    yoff = mScrollY;
}

layerCanvas.translate(0, -yoff);
if (mTranslator != null) {
    mTranslator.translateCanvas(layerCanvas);
}

DisplayList displayList = mView.mDisplayList;
if (displayList != null) {
    layerCanvas.drawDisplayList(displayList, null,
            DisplayList.FLAG_CLIP_CHILDREN);
} else {
    mView.draw(layerCanvas);
}

drawAccessibilityFocusedDrawableIfNeeded(layerCanvas);

mResizeBufferStartTime = SystemClock.uptimeMillis();
mResizeBufferDuration = mView.getResources().getInteger(
        com.android.internal.R.integer.config_mediumAnimTime);
completed = true;

layerCanvas.restoreToCount(restoreCount);
```

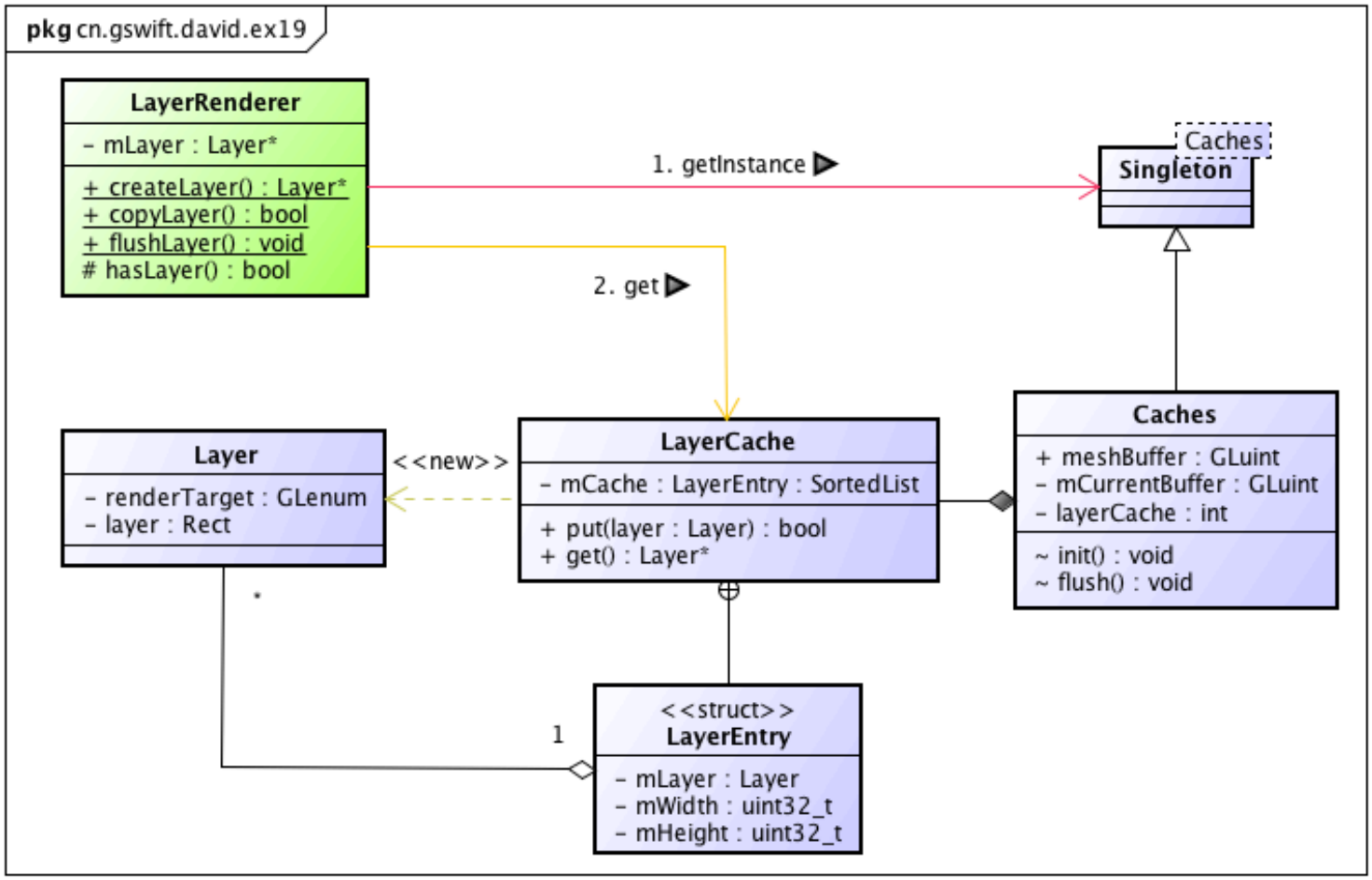
nCreatelayer



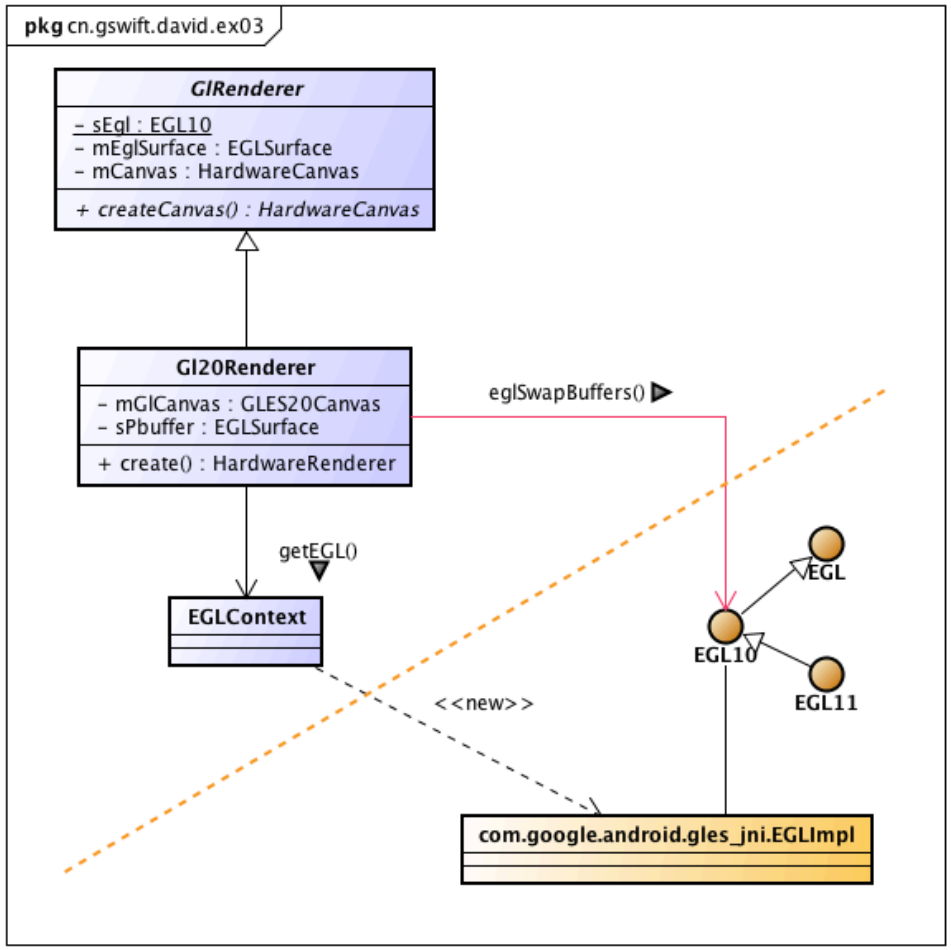
Create a native Layer(1)



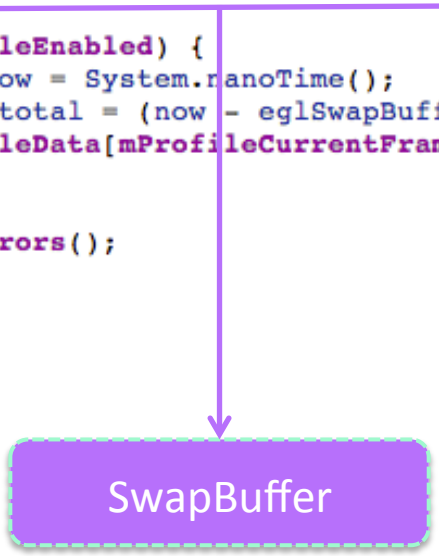
Create a native Layer (2)



Swap Buffers



```
onPostDraw();

attachInfo.mIgnoreDirtyState = false;

if ((status & DisplayList.STATUS_DREW) == DisplayList.STATUS_DREW) {
    long eglSwapBuffersStartTime = 0;
    if (mProfileEnabled) {
        eglSwapBuffersStartTime = System.nanoTime();
    }

    sEgl.eglSwapBuffers(sEglDisplay, mEglSurface);

    if (mProfileEnabled) {
        long now = System.nanoTime();
        float total = (now - eglSwapBuffersStartTime) * 0.000001f;
        mProfileData[mProfileCurrentFrame + 2] = total;
    }

    checkEglErrors();
}
```



```
            void* prevBits;
            if (lock(previousBuffer,
                    GRALLOC_USAGE_SW_READ_OFTEN, &prevBits) == NO_ERROR) {
                // copy from previousBuffer to buffer
                copyBlt(buffer, bits, previousBuffer, prevBits, copyBack);
                unlock(previousBuffer);
            }
        }
    }
    oldDirtyRegion = dirtyRegion;
}

if (previousBuffer) {
    previousBuffer->common.decRef(&previousBuffer->common);
    previousBuffer = 0;
}
```

```
unlock(buffer);
previousBuffer = buffer;
nativeWindow->queueBuffer(nativeWindow, buffer, -1);
buffer = 0;
```

queueBuffer

```
// dequeue a new buffer
int fenceFd = -1;
if (nativeWindow->dequeueBuffer(nativeWindow, &buffer, &fenceFd) == NO_ERROR) {
    sp<Fence> fence(new Fence(fenceFd));
    if (fence->wait(Fence::TIMEOUT_NEVER)) {
        nativeWindow->cancelBuffer(nativeWindow, buffer, fenceFd);
        return setError(EGL_BAD_ALLOC, EGL_FALSE);
    }

    // reallocate the depth-buffer if needed
    if ((width != buffer->width) || (height != buffer->height)) {
        // TODO: we probably should reset the swap rect here
        // if the window size has changed
        width = buffer->width;
        height = buffer->height;
        if (depth.data) {
            free(depth.data);
            depth.width   = width;
            depth.height  = height;
            depth.stride  = buffer->stride;
            depth.data    = (GGLubyte*)malloc(depth.stride*depth.height*2);
            if (depth.data == 0) {
                setError(EGL_BAD_ALLOC, EGL_FALSE);
                return EGL_FALSE;
            }
        }
    }

    // keep a reference on the buffer
    buffer->common.incRef(&buffer->common);
```

dequeueBuffer

```
status_t OpenGLRenderer::prepareDirty(float left, float top, float right, float bottom,
        bool opaque) {
    mCaches.clearGarbage();

    mSnapshot = new Snapshot(mFirstSnapshot,
            SkCanvas::kMatrix_SaveFlag | SkCanvas::kClip_SaveFlag);
    mSnapshot->fbo = getTargetFbo();
    mSaveCount = 1;

    mSnapshot->setClip(left, top, right, bottom);
    mDirtyClip = true;

    updateLayers();

    // If we know that we are going to redraw the entire framebuffer,
    // perform a discard to let the driver know we don't need to preserve
    // the back buffer for this frame.
    if (mCaches.extensions.hasDiscardFramebuffer() &&
            left <= 0.0f && top <= 0.0f && right >= mWidth && bottom >= mHeight) {
        const GLenum attachments[] = { getTargetFbo() == 0 ? GL_COLOR_EXT : GL_COLOR_ATTACHMENT0 };
        glDiscardFramebufferEXT(GL_FRAMEBUFFER, 1, attachments);
    }

    syncState();

    // Functors break the tiling extension in pretty spectacular ways
    // This ensures we don't use tiling when a functor is going to be
    // invoked during the frame
    mSuppressTiling = mCaches.hasRegisteredFunctors();

    mTilingSnapshot = mSnapshot;
    startTiling(mTilingSnapshot, true);

    debugOverdraw(true, true);

    return clear(left, top, right, bottom, opaque);
}

void OpenGLRenderer::startTiling(const sp<Snapshot>& s, bool opaque) {
    if (!mSuppressTiling) {
        Rect* clip = mTilingSnapshot->clipRect;
        if (s->flags & Snapshot::kFlagIsFboLayer) {
            clip = s->clipRect;
        }

        mCaches.startTiling(clip->left, s->height - clip->bottom,
                clip->right - clip->left, clip->bottom - clip->top, opaque);
    }
}
```

Next->
Go to SurfaceFlinger

参考资料：

Android4.2 http://code.metager.de/source/xref/android/4.2/

Android Accelerated Rendering, Romain Guy Chet Haase, May 11, 2011, http://www.google.com/events/io/2011/sessions/accelerated-android-rendering.html

Android Graphics Architecture I ,himmele,http://himmele.googlecode.com

Android's 2D Canvas Rendering Pipeline, Laurence Gonsalves, May 2011, http://www.xenomachina.com/2011/05/androids-2d-canvas-rendering-pipeline.html

Skia & FreeType Android 2D Graphics Essentials , Kyungmin Lee , Software Platform Lab., LG Electronics ,2012, http://www.kandroid.org

http://en.wikipedia.org/wiki/Display_list

http://developer.android.com/guide/topics/graphics/hardware-accel.html

About Me

- I have been working as a product-designer specializing in software/Web application design and development. I am passionate about mobile application development and became interested in Android programming when the platform was launched by Google. Thus I was not programming on Android projects, I spent spare time reading technical blogs, researching, analyzing, and testing mobile applications, as a software consultancy specialized in android technologies.
- In my product-design time, in the developing, I've encountered too many program manage troubles that suffer due to poor communication and code design, I know that help them to understand the system framework is very important. I amd experienced in system and application layers, my goal is simple: help someone who wishes to better understand the **Android framework** in java、JNI and C/C++ libraries.

- Please also check my article and slides on this http://blog.sina.com.cn/gswift

Contact: Zhiyong.liu@aliyun.com

http://weibo.com/gswift