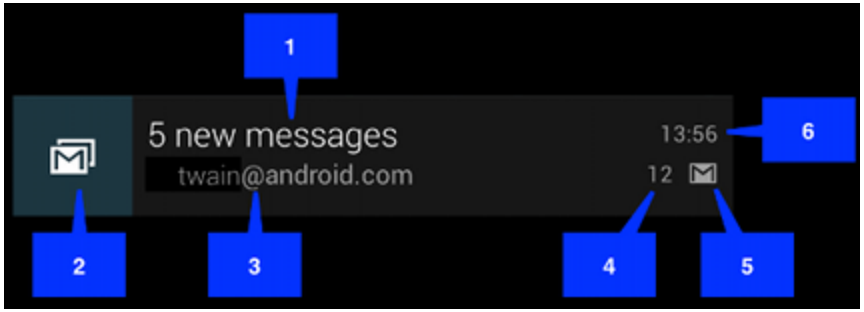
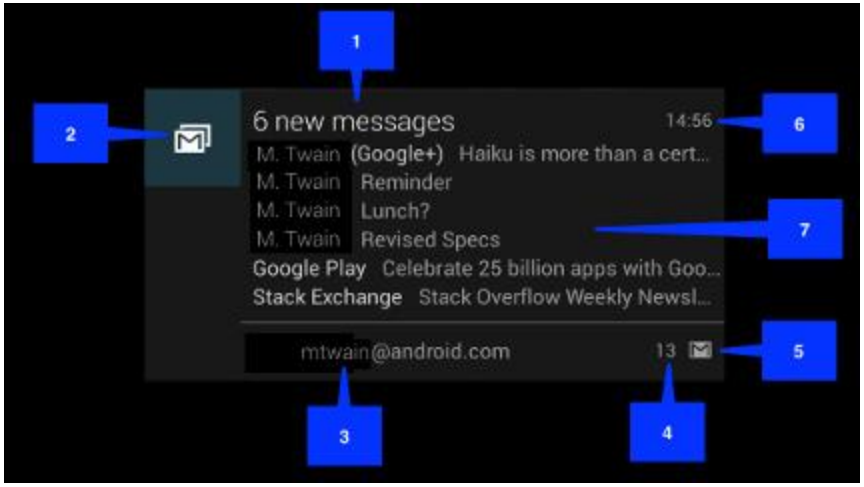
Android Programing

Gavrilut Dragos

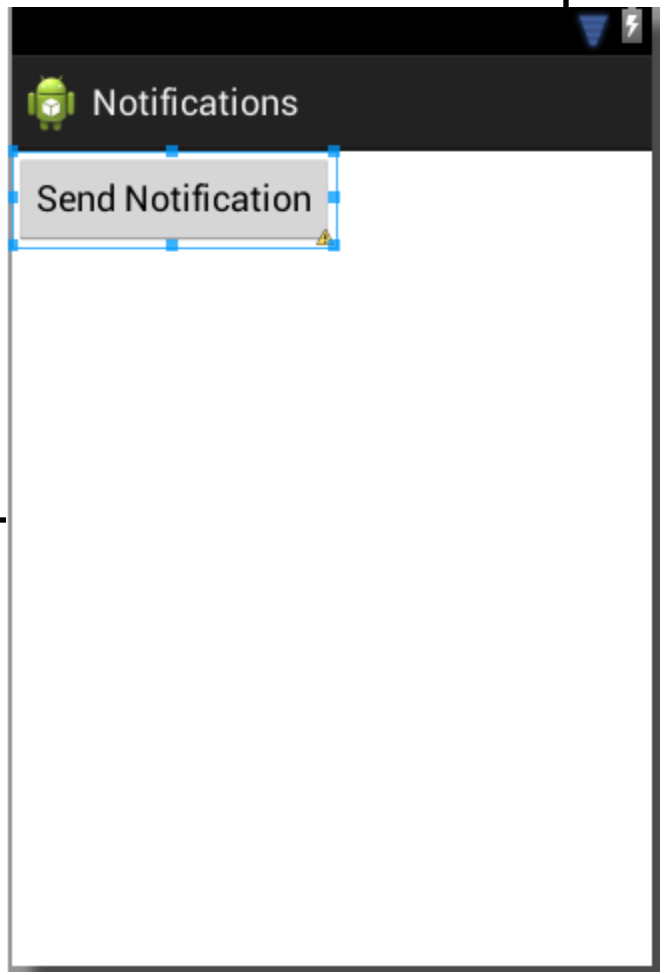
Notifications





Notifications

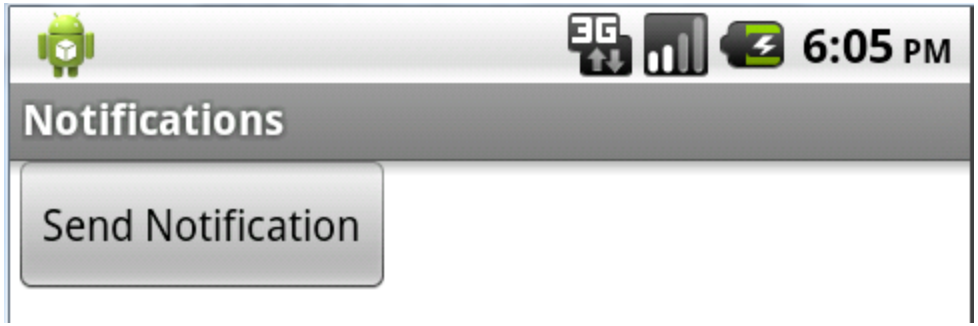
```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Send Notification"
        android:onClick="sendNotification" />

</RelativeLayout>
```

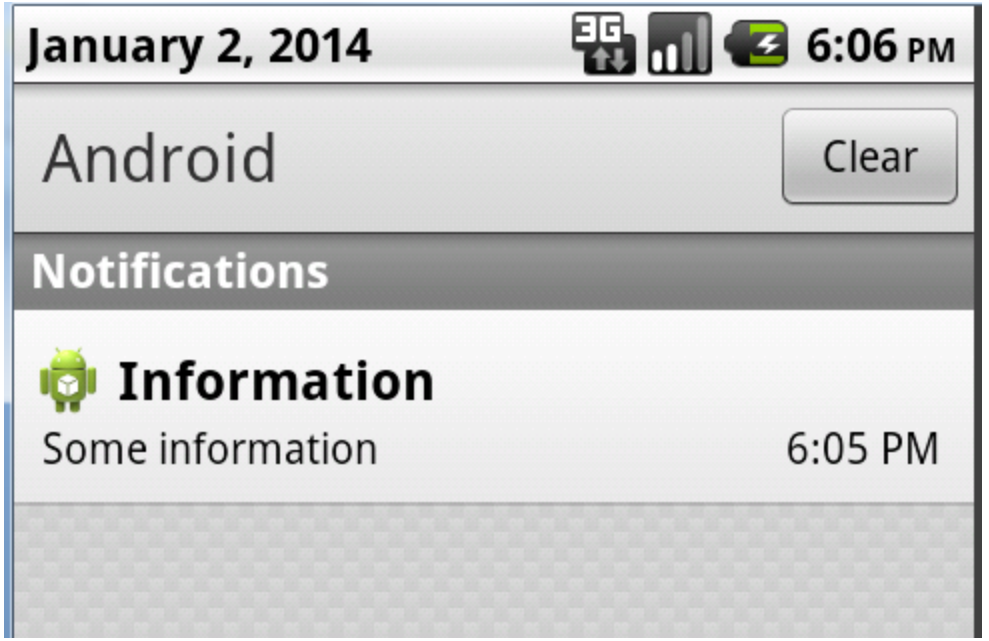


Notifications

```
public class MainActivity extends Activity {

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

    public void sendNotification(View view) {
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.infoiasi.ro"));
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);

        NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
        builder.setSmallIcon(R.drawable.ic_launcher);
        builder.setContentIntent(pendingIntent);

        builder.setAutoCancel(true);

        builder.setLargeIcon(BitmapFactory.decodeResource(getResources(), R.drawable.ic_launcher));
        builder.setContentTitle("Information");
        builder.setContentText("Some information");
        builder.setSubText("Touch to see infoiasi.ro site");

        NotificationManager notificationManager;
        notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        notificationManager.notify(1234, builder.build());

    }
}
```

Notifications





Notifications

```
public class MainActivity extends Activity {

@Override
protected void onCreate(Bundle savedInstanceState) {
        // …..
}

    public void sendNotification(View view) {
        NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
        builder.setContentTitle("Percentage");
        builder.setContentTitle("Percentage for download");
        builder.setSmallIcon(R.drawable.ic_launcher);
        builder.setProgress(100, 10, false);
        // set the intent and other options

        NotificationManager notificationManager;
        notificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        notificationManager.notify(1234, builder.build());

    }


}
```

JNI (C/C++ in Android System)

- Works as a library (for linux system *.so) that is loaded and called directly from dalvik virtual machine
- To compile the library the appropiate NDK package from google must be downloaded and installed. The package contains all the tools necessary to create a .so library that can be used by the Android system
  http://developer.android.com/tools/sdk/ndk/index.html

JNI (C/C++ in Android System)

- To use such a library the following steps must be performed:
  a) Create a wrapper class in Java (a wrapper class is a class that provides the Java prototype functions for the functions that are exported from the C/C++ library)
  b) Create the C/C++ code. Make sure that the function that should be exported are renamed so that they match the funtion rom the wrapper class
  c) Compile the library. The newly created library will be added in the **\libs** folder (under the appropriate architecture type). Currently – android system supports tha following architecture: ARM, x86, MIPS
  d) Compile the whole project

JNI (C/C++ in Android System)

- Create a wrapper class in Java :

```
public class MyWrapper {

     static {
         System.loadLibrary("my_library");
     }

     public static native void function_1(int param1, int param2);
     public static native void function_2();
     public static native char function_3(int param1);
}
```

JNI (C/C++ in Android System)

- Create a folder "jni" in your Android project
- Create "Android.mk" file
- Create several C/C++ files (*.h, *.cpp, *.c) that will be compiled in a single library. For this example we create "test.c" file

```
LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := my_library
LOCAL_CFLAGS    := -Werror
LOCAL_SRC_FILES := test_c
LOCAL_LDLIBS    := -llog

include $(BUILD_SHARED_LIBRARY)
```

JNI (C/C++ in Android System)

- Create “test.c” file

```
#include <jni.h>
#include <android/log.h>
#include <stdio.h>
#include <stdlib.h>
#define  LOG(...)  __android_log_print(ANDROID_LOG_INFO, “TAG",__VA_ARGS__)

extern "C" {
    JNIEXPORT void JNICALL Java_com_teststudenti_MyWrapper_function_1(JNIEnv * env, jobject obj,
                                                                      Jint param1, jint param2);
    JNIEXPORT void JNICALL Java_com_teststudenti_MyWrapper_function_2(JNIEnv * env, jobject obj);
    JNIEXPORT void JNICALL Java_com_teststudenti_MyWrapper_function_3(JNIEnv * env, jobject obj,
                                                                      jint param1);
};

JNIEXPORT void JNICALL Java_com_teststudenti_MyWrapper_function_1(JNIEnv * env, jobject obj,  jint
param1, jint param2)
{
    LOG(“Apel functie 1 (Param1=%d,Param2=%d)“,param1,param2);
}
JNIEXPORT void JNICALL Java_com_teststudenti_MyWrapper_function_2(JNIEnv * env, jobject obj)
{
    LOG(“Apel functie 2");
}
JNIEXPORT void JNICALL Java_com_teststudenti_MyWrapper_function_3(JNIEnv * env, jobject obj, jint
param1)
{
    LOG(“Apel functie 3");
}
```

JNI (C/C++ in Android System)

- Using Java functions from JNI

```
#include <jni.h>
#include "ArrayHandler.h"

JNIEXPORT jobjectArray JNICALL <path+name> (JNIEnv *env, jobject jobj)
{
    jobjectArray ret;
    char *message[3]= {"text-1", "text-2","text-3"};
    ret= (jobjectArray)env->NewObjectArray(3,
                                    env->FindClass("java/lang/String"),
                                    env->NewStringUTF(""));

    for(int tr=0;tr<3;tr++)
        env->SetObjectArrayElement(ret,i,env->NewStringUTF(message[i]));

    return(ret);
}
```

JNI (C/C++ in Android System)

- Using Java functions from JNI (2)

```
#include <jni.h>
#include <android/bitmap.h>

JNIEXPORT void JNICALL <package_fnc>(JNIEnv * env,
                                     jobject  obj,
                                     jobject bitmap)
{
    AndroidBitmapInfo  imageInfo;
    void*              pixelsMatrix;


    if (AndroidBitmap_getInfo(env, bitmap, &imageInfo) < 0)
          return;
    if (AndroidBitmap_lockPixels(env, bitmap, &pixelsMatrix) < 0)
          return;

    // do some work with the image
    AndroidBitmap_unlockPixels(env, bitmap);
}
```

JNI (C/C++ in Android System)

- Using Java functions from JNI(3)

```
#include <jni.h>

void Java_the_package_MainActivity_getJniString( JNIEnv* env, jobject obj){

    jstring txt = (*env)->NewStringUTF(env, “<some text>”)
    jclass classObject = (*env)->FindClass(env,
                                            “<package>\MainActivity");
    jmethodID classMethod = (*env)->GetMethodID(env,
                    classObject,
                    “<function name>",
                    "(Ljava/lang/String;)Ljava/lang/String;");
    jobject result = (*env)->CallObjectMethod(env, obj, classMethod, txt);
}
```

JNI (C/C++ in Android System)

- Naming conventions / Signatures
  - B = byte
  - S = short
  - I = int
  - J = long
  - F = float
  - D = double
  - C = char
  - Z = boolean
  - V = void
  - L = prefix for a specific type
  - [→ prefix for an array
- Example
  - [B → byte[ ]
  - [[D → double[ ][ ]
  - [Ljava/lang/String; → String [ ]

Open GL in Android

- Android supports OpenGL ES (embedded system) specifications for versions 1.0, 1.1, 2.0 and 3.0
- OpenGL 1.x = fix pipeline
- OpenGL 2.x+ = programable pipeline

OpenGL ES Version	Percentage
1.1	0.1%
2.0	96.3%
3.0	3.6%

- Devices that support a version of OpenGL also suports the previous verions. That means tha 99.9% of the currently available Android devices suports OpenGL 2.0

OpenGL in Android

- There are two methods that can be used in Android to implement OpenGL code
  - Natively through JNI code (complicated but more compatible with other OS that supports OpenGL ES)
  - Through GLSurfaceView class that exista in Android. While the code is similar to the one from JNI, there are still some differences that makes porting a GLSurfaceView class inneficient.

OpenGL in Android

- GLSurfaceView uses a renderer (a class that will be called to create a list of commands for the OpenGL pipe line)
- In case of Android devices OpenGL ES can produce up to 60 FPS
- While the GLSurfaceView is slower than a code written in JNI it has the advantage of beeing able to work directly with resources (and especially with Bitmap object or PNG files). These can also be achieve using JNI with:
  - Loading the Bitmap class from android system and using it internally
  - Including a PNG C/C++ library directly in your JNI code

OpenGL in Android

- Activity Class

```
public class OpenGLActivity extends Activity {
    OpenGLSurfaceView glView;

    @Override
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        glView = new OpenGLSurfaceView(getApplication());
        setContentView(glView);
    }

    @Override
    protected void onPause() {
        super.onPause();
        glView.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
        glView.onResume();
    }
}
```

OpenGL in Android

- Surface View Class

```
class OpenGLSurfaceView extends GLSurfaceView {

    public OpenGLSurfaceView(Context context) {
        super(context);
        setEGLContextClientVersion(2);
        setRenderer(new MyRenderer());
    }
}
```

```
class MyRenderer implements GLSurfaceView.Renderer {
    public void onDrawFrame(GL10 gl) {
        …
    }

    public void onSurfaceChanged(GL10 gl, int width, int height) {
        …
    }

    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        …
    }
}
```