http://www.android.com/

SMS, binary SMS and MMS

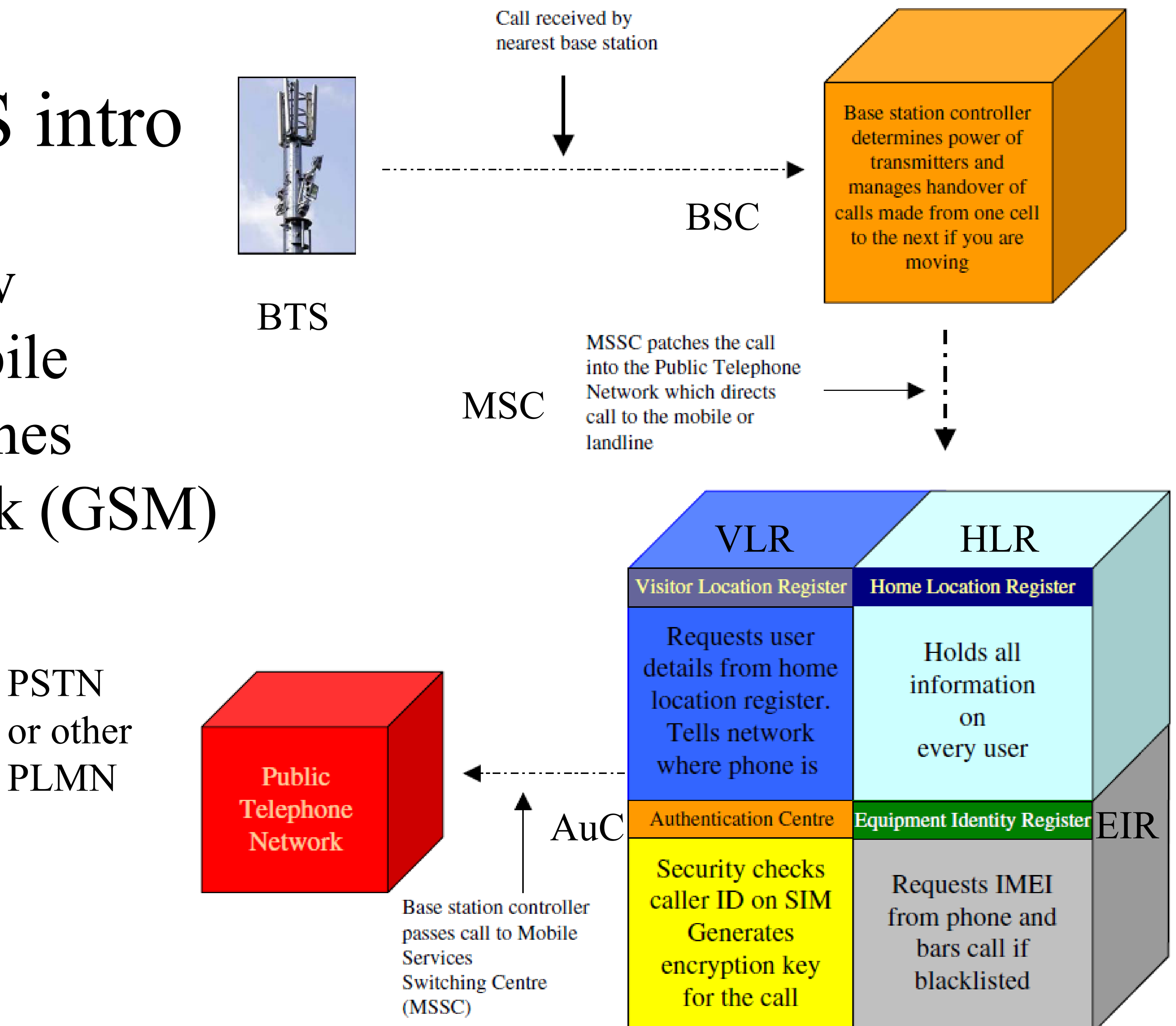
Concurrent programming (Threads, Handler and AsyncTask)

Timers and TimerTasks

Networking

SMS intro

- How mobile phones work (GSM)



Call received by nearest base station

BTS

BSC

Base station controller determines power of transmitters and manages handover of calls made from one cell to the next if you are moving

MSC

MSSC patches the call into the Public Telephone Network which directs call to the mobile or landline

VLR

Visitor Location Register

Requests user details from home location register. Tells network where phone is

HLR

Home Location Register

Holds all information on every user

AuC

Authentication Centre

Security checks caller ID on SIM Generates encryption key for the call

EIR

Equipment Identity Register

Requests IMEI from phone and bars call if blacklisted

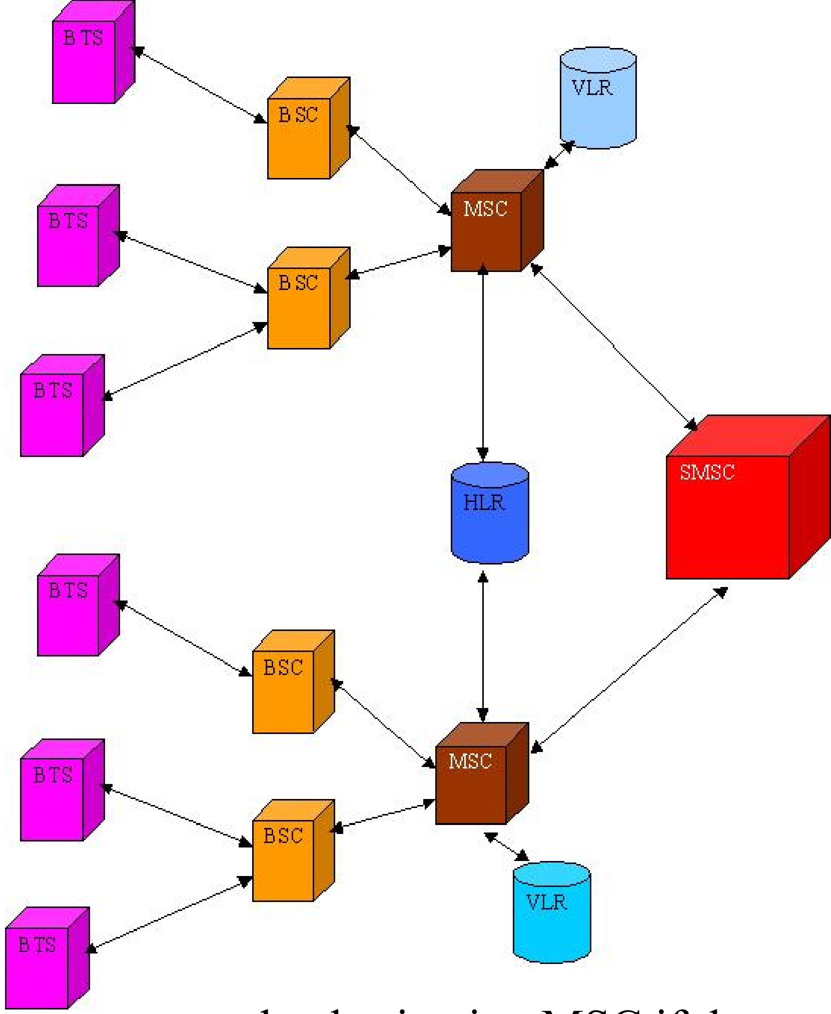
PSTN or other PLMN

Public Telephone Network

Base station controller passes call to Mobile Services Switching Centre (MSSC)

GSM

SMS intro



- BTS - Base Transceiver Station (antenna)
- BSC - Base Station Controller
- MSC - Mobile Switching Center
- HLR- Home Location Register
- VLR - Visitor Location Register
- SMSC - Short Message Service Center
- When a user sends an SMS, the request is placed via the MSC
- The MSC forwards the SMS to the SMSC where it gets stored
- The SMSC queries the HLR to find out where the destination mobile is and forwards the message to the destination MSC if the destination mobile is available
- If the mobile is not available the message gets stored in the current SMSC itself. In most installations If a mobile is not available for SMS delivery the SMSC will not retry. Instead the destination MSC will inform the SMSC when the mobile comes back in range

http://services.eng.uts.edu.au/userpages/kumbes/public_html/ra/sms/

Send a SMS

- A PendingIntent is a long lived (owner process can be killed) description of an Intent and target action to perform with it
  - By giving a PendingIntent to another application, you are granting it the right to perform the operation you have specified as if the other application was yourself (with the same permissions and identity)
  - https://stackoverflow.com/questions/2808796/what-is-an-android-pendingintent
- SmsManager manages SMS operations
  - Most SMSs are sent via the PDU (Protocol Description Unit) format: http://www.gsm-modem.de/sms-pdu-mode.html
- To receive SMS we must set up a receiver in AndroidManifest and create the BroadcastReceiver class which override the onReceive(Context context, Intent intent) method (next slide)

```
private void sendSMS(String phoneNumber, String message)
{
     PendingIntent pi = PendingIntent.getActivity(this, 0, new Intent(this, SMS.class), 0);
     // Get the default instance of the SmsManager
     SmsManager sms = SmsManager.getDefault();
     // sendTextMessage (String destinationAddress, String scAddress, String text,
     // PendingIntent sentIntent, PendingIntent deliveryIntent), beware of API changes!
     sms.sendTextMessage(phoneNumber, null, message, pi, null);
}
```

If we want to listen for the sent and delivery intents we need to set up receivers for these as well

Needed permissions in AndroidManifest
android.permission.SEND_SMS
android.permission.RECEIVE_SMS

Receive a SMS

- The incoming SMS broadcast receiver uses a bundle to retrieve the PDU (Protocol Description Unit), which contains the SMS text and any additional SMS meta-data, and parses it into an Object array

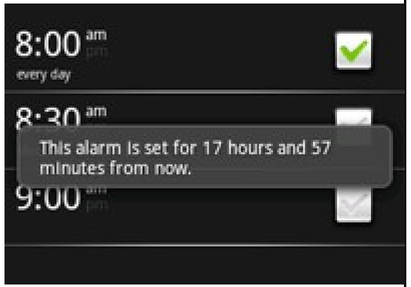
```
public class SmsReceiver extends BroadcastReceiver
{
      @Override
      public void onReceive(Context context, Intent intent)
      {
            Bundle bundle = intent.getExtras(); //---get the SMS message passed in---
            SmsMessage[] msgs = null;
            String str = "My SmsReceiver-> ";

            if (bundle != null)
            {
                  //---retrieve the SMS message received---
                  Object[] pdus = (Object[]) bundle.get("pdus");
                  msgs = new SmsMessage[pdus.length];
                  //---for every SMS message received---
                  for (int i=0; i<msgs.length; i++)
                  {
                        msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]); // convert Object array
                        str += "SMS from " + msgs[i].getOriginatingAddress(); //sender's phone number
                        str += " : ";
                        str += msgs[i].getMessageBody().toString(); // get the text message
                        str += "\n";
                  }//---display the new SMS message---
                  Toast.makeText(context, str, Toast.LENGTH_SHORT).show();
            }
      }
}
```

```
<receiver android:name=".SmsReceiver"> <!-- AndroidManifest -->
      <intent-filter>
            <action android:name="android.provider.Telephony.SMS_RECEIVED" />
      </intent-filter>
</receiver>
```

Dynamic BroadcastReceiver

- By registering a broadcast receiver in the AndroidManifest or dynamically in the source code, the application can listen and respond to broadcast Intents that match a specific filter criteria
- By calling batteryLevel() a Toast will show the battery level when onReceive() is called by the system
- When onRecive() is done the lifecycle has ended for a broadcast receiver

```
private void batteryLevel()
{
    BroadcastReceiver batteryLevelReceiver = new BroadcastReceiver()
    {
        @Override
        public void onReceive(Context context, Intent intent)
        {
             // unregistration of the reciever
            context.unregisterReceiver(this);
            int rawlevel = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
            int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
            int level = -1;
            if (rawlevel >= 0 && scale > 0) {
                level = (rawlevel * 100) / scale;
           }
           Toast.makeText(getApplicationContext(), "Battery Level Remaining: " + level + "%",
                 Toast.LENGTH_SHORT).show();
        }
    };

    // Intent and a dynamic registration of a receiver via registerReciver
    IntentFilter batteryLevelFilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
    registerReceiver(batteryLevelReceiver, batteryLevelFilter);
}
```



This alarm is set for 17 hours and 57 minutes from now.

Broadcast & Filter Intents

- We can create our own custom system wide broadcast intents with sendBroadcast()

```
public class Test extends Activity {
   public static final String CUST_EV1 = "myapp.CUSTOM_EVENT1";

   private void myReceiverMethod(Context context) {
      // create the receiver
      BroadcastReceiver myCustReceiver = new BroadcastReceiver(){
         @Override
         public void onReceive(Context c, Intent intent) {
            Bundle bundle = intent.getExtras();
            String message = bundle.getString(KEY);
            // TODO - take care of the actions
         }
      };
     // create filter and register the listener as usual
     // active when application is running or until unregistered
     // with unRegisterReceiver(myCustReceiver);
     registerReceiver(myCustReceiver, new IntentFilter(CUST_EV1));
   }

   private void mySendBroadcastMethod(Context context) {
      // create the custom broadcast intent
      Intent intent = new Intent(CUST_EV1);
      intent.putExtra(KEY, ...);
      context.sendBroadcast(intent);
   }
}
```

```
// there are many native broadcast
// intents we either can listen for
// or use, at least +100!
Intent it =
   new Intent(Intent.ACTION_VIEW, uri);
startActivity(it);

// examples
Intent.ACTION_DATE_CHANGED
Intent.ACTION_TIME_CHANGED
Intent.ACTION_MEDIA_BUTTON
Intent.ACTION_CAMERA_BUTTON
Intent.ACTION_NEW_OUTGOING_CALL
Intent.ACTION_SCREEN_ON
Intent.ACTION_SCREEN_OFF
Intent.ACTION_TIMEZONE_CHANGED
Intent.ACTION_PACKAGE_ADDED
Intent.ACTION_MEDIA_EJECT
Intent.ACTION_MEDIA_MOUNTED
Intent.ACTION_MEDIA_UNMOUNTED
Intent.ACTION_BATTERY_CHANGED
Intent.ACTION_POWER_CONNECTED
Intent.ACTION_POWER_DISCONNECTE
Intent.ACTION_BOOT_COMPLETED
Intent.ACTION_SHUTDOWN
Intent. ...
```

Send multipart SMS

- Most SMSes are restricted to 140 characters per text message. To make sure the message is within this limitation, use the **divideMessage()** method that divides the text into fragments in the maximum SMS message size. Then, the method **sendMultipartTextMessage()**

```
private void sendTextSMSMulti(String destination, String message) {
	SmsManager mySMS = SmsManager.getDefault();
	Intent sentIn = new Intent("SENT_SMS");
	PendingIntent sentPIn = PendingIntent.getBroadcast(this, 0, sentIn, 0);
	Intent deliverIn = new Intent("DELIVER_SMS");
	PendingIntent deliverPIn = PendingIntent.getBroadcast(this, 0, deliverIn, 0);
	ArrayList<String> multiSMS = mySMS.divideMessage(message);
	ArrayList<PendingIntent> sentIns = new ArrayList<PendingIntent>();
	ArrayList<PendingIntent> deliverIns = new ArrayList<PendingIntent>();
	for(int i=0; i < multiSMS.size(); i++){
		sentIns.add(sentPIn);
		deliverIns.add(deliverPIn);
	}
	mySMS.sendMultipartTextMessage(destination, null, multiSMS, sentIns, deliverIns);

	BroadcastReceiver sentReceiver = new BroadcastReceiver(){
		@Override public void onReceive(Context c, Intent in) {
			switch(getResultCode()){
			case Activity.RESULT_OK:
				Break; //sent SMS message successfully;
			default:
				Break; //sent SMS message failed
			}
		}
	};
	BroadcastReceiver deliverReceiver = new BroadcastReceiver(){
		@Override public void onReceive(Context c, Intent in) {
			//SMS delivered actions
		}
	};
	registerReceiver(sentReceiver, new IntentFilter("SENT_SMS"));
	registerReceiver(deliverReceiver, new IntentFilter("DELIVER_SMS"));
}
```

Retrieve a PendingIntent that will perform a broadcast, like calling Context.sendBroadcast()

Send binary SMS and MMS

- To send binary SMS with sendDataMessage we need a destination port
- Sending MMS using the built-in SMS/MMS manager (the ones who listen for ACTION_SEND)

```java
private void sendBinarySMS(String phoneNumber, byte[] data)
{
	short destinationPort = 2948;
	PendingIntent pi = PendingIntent.getActivity(this, 0, new Intent(this, SMS.class), 0);
	SmsManager sms = SmsManager.getDefault(); // Get the default instance of the SmsManager
	// Send a data based SMS to a specific application port.
	sms.sendDataMessage(phoneNumber, null, destinationPort, data, pi, null);
}
```

```java
// Send MMS via a broadcast intent to the UE built in action send components
private void sendMMS(String phoneNo, String subject, String message)
{
	String url = "file:////sdcard//DCIM//07.jpg"; // Environment.getExternalStorageDirectory();
	/* The url being passed to the Uri.parse method should be of the form used to access the media store
	 * such as content://media/external/images/media/23 or file://sdcard/dcim/Camera/off2.jpg
	*/
	Intent sendIntent = new Intent(Intent.ACTION_SEND);
	sendIntent.putExtra(Intent.EXTRA_PHONE_NUMBER, phoneNo);
	sendIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
	sendIntent.putExtra(Intent.EXTRA_TEXT, message);
	sendIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse(url));
	sendIntent.setType("image/jpg"); // specify explicit, normally type is set automatically from the data
	startActivity(sendIntent);    // broadcast intent for all apps listening to ACTION_SEND
}
```

Binary SMS receiver

```xml
<receiver android:name="BinarySmsReceiver">
     <intent-filter>
          <action android:name="android.intent.action.DATA_SMS_RECEIVED"
               android:scheme="sms" android:host="localhost" android:port="2948">
          </action>
     </intent-filter>
</receiver>
```

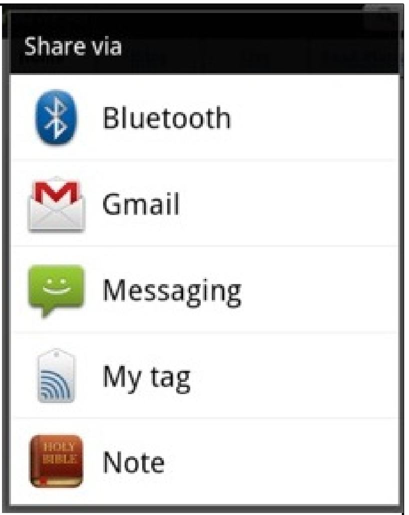
```java
public class BinarySmsReceiver extends BroadcastReceiver
{
     @Override
     public void onReceive(Context context, Intent intent)
     {
          Bundle bundle = intent.getExtras();
          SmsMessage[] msgs = null;
          String info = "Binary SMS from ";
          if (bundle != null){
               //---retrieve the binary SMS message received---
               Object[] pdus = (Object[]) bundle.get("pdus");
               msgs = new SmsMessage[pdus.length];
               byte[] data = null;
               for (int i=0; i<msgs.length; i++)
               {
                    msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
                    info += msgs[i].getOriginatingAddress();
                    info += "\n*****BINARY MESSAGE*****\n";
                    // returns the user data section minus the user data header if one was present.
                    data = msgs[i].getUserData();
                    for(int index=0; index<data.length; index++)
                         info += Byte.toString(data[index]);
               }
               //---display the new binary SMS message---
               Toast.makeText(context, info, Toast.LENGTH_LONG).show();
          }
     }
}
```

Send SMS and Email with built-in client apps (user interaction)

```
public void sendNativeSMS(String phoneNumber, String message)
{
      // sendNativeSMS("12345", "Hello my friends!");
      Intent i = new Intent(Intent.ACTION_VIEW);
      i.putExtra("address", phoneNumber);
      i.putExtra("sms_body", message);
      i.setType("vnd.android-dir/mms-sms");
      startActivity(i);
}
```

```
Convenience function for creating a ACTION_CHOOSER Intent
startActivity(Intent.createChooser(myIntent, myString));
```

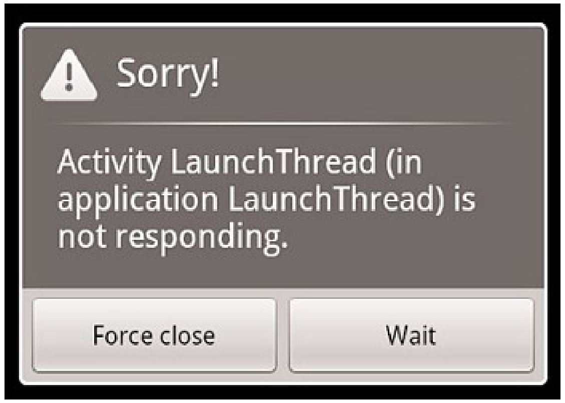
```
String[] to = {"hjo@du.se"};
String[] cc = {""};
sendEmail(to, cc, mSubject, mMessage, mFileUrl);

// check http://www.openintents.org/en/uris for MIME types
// http://developer.android.com/reference/android/content/Intent.html

//---sends an Email message to another device---
private void sendEmail(String[] emailAddresses, String[] carbonCopies,
       String subject, String message, String url)
{
    Intent emailIntent = new Intent(Intent.ACTION_SEND);
/* Intent emailIntent = new Intent(Intent.ACTION_SEND_MULTIPLE); */
    String[] to = emailAddresses;
    String[] cc = carbonCopies;
    emailIntent.putExtra(Intent.EXTRA_EMAIL, to);
    emailIntent.putExtra(Intent.EXTRA_CC, cc);
    emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
    emailIntent.putExtra(Intent.EXTRA_TEXT, message);
    emailIntent.putExtra(Intent.EXTRA_STREAM, Uri.parse("file:///" + mFileUrl));
    emailIntent.setType("text/plain");
    startActivity(Intent.createChooser(emailIntent, "Share via"));
}
```



Concurrent programming 1

- Android applications normally run entirely on a single thread (the “main thread” or “UI thread”)
- The main thread handles all user input, executing code in event listeners, rendering and life cycle call backs
- Code running in the main thread should do as **little work as possible** to keep the application and it’s UI responsive
- Howto handle time consuming tasks?
  - Spawn a new thread to do the work in the background
  - Examples: Long calculations; network, file and database operations; game animations; …

Concurrent programming 2


Sorry!

Activity LaunchThread (in application LaunchThread) is not responding.

- The Android system guards against non responsiveness
- The Application Not Responding (ANR) FC/Wait dialog
- No response to an input event (key press or screen touch) within 5 sec. or a BroadcastReceiver hasn't finished executing within 10 sec.
  - http://developer.android.com/training/articles/perf-anr.html
- Create new threads in two ways (see example next slide)
  - Extending Thread or implement Runnable interface

```
1. extend java.lang.Thread
2. override public void run()
3. call thread.start();
```

```
1. implement interface java.lang.Runnable
2. override public void run()
3. Thread t = new Thread(new MyRunnable());
4. t.start();
```

Java Threads Basics

Full examples in Communication and WaitNotify folders

```
class ThreadTest()
{
    // Threads using interface Runnable
    // you put in a ThreadDemo1 object as param when creating the thread
    Thread t1 = new Thread(new ThreadDemo1());
    // when we call thread start the run method is executed
    t1.start();
    // Threads inheriting (extends) from class Thread
    ThreadDemo2 t2 = new ThreadDemo2();
    // when we call thread start the run method is executed
    t2.start();
}
class ThreadDemo1 implements Runnable
{
   // the run method must be implemented when using threads,
   // here is where the actual thread execution is taking place,
   // the thread dies when run returns
   public void run(){
      // do the work
   }
}
class ThreadDemo2 extends Thread
{
   // identical code to ThreadDemo1 inside the class
   public void run() { // do the work }
}
```

```
// The Thread.stop() method is deprecated
// use this way to stop the thread myThread
if(myThread != null) {
     Thread dummy = myThread;
     myThread = null;
     dummy.interrupt();
}
```

Concurrency issues and solutions

- Concurrency issues
  - Updating the UI or other components during the execution or after the worker thread have finished
  - Manipulating UI or other components from another thread than the main thread might cause corrupted state (race conditions)
  - Android UI is not thread-safe and must **always** be updated on the UI thread!
- Solutions to access the UI thread in order to update the UI from other threads
  - View.post(Runnable action)
  - View.postDelayed(Runnable action, long delayMillis)
  - Activity.runOnUiThread(Runnable action)
  - Handler – post (as View) and sendMessage
  - **AsyncTask** (this is the recommended solution)

View.post(), View.postDelayed()

- Further info about Android threading and a good read
  - http://android-developers.blogspot.com/2009/05/painless-threading.html

```
private ImageView mIVStatus = new ImageView();
@Override
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap b = loadImageFromNetwork();
            mIVStatus.post(new Runnable() {
                public void run() {
                    mIVStatus.setImageBitmap(b);
                }
            });
        }
    }).start();
}
```

```
// small thread function which can do some work
Thread bkgdThread = new Thread(new Runnable() {
        public void run() {
                doSomeWork();
        }
});
bkgdThread.start();
```

Handler - post()/postDelayed()

A Handler is used to send a **Message** or a **Runnable** object to a particular thread. The thing to remember is that a Handler is associated with the MessageQueue of the single thread which has created it. After creating a Handler, it can be used to post a **Message (next slide)** or **Runnable** to that particular thread (as View.post()).

There are two main uses for a Handler: (1) to schedule messages and runnables to be executed as some point in the future; and (2) to enqueue an action to be performed on a different thread than your own.

```
private TextView mTV; // A timer task that posts Cell ID messages to a textview in the activity
private final Handler mHandler = new Handler();
private final static int TIMER_DELAY = 3000;

private Runnable mUpdateTimeTask = new Runnable() {
      public void run() {
            Log.d(TAG, "Timer fired");
            getCID();
            // Causes the Runnable to be added to the message queue
            // The runnable will be run on the thread to which this handler is attached
            mTV.setText("CID: " + String.valueOf(mCellid) + " LAC: " + String.valueOf(mLac));
            mHandler.postDelayed(this, TIMER_DELAY);
      }
};
@Override
protected void onPause() {
      mHandler.removeCallbacks(mUpdateTimeTask);
      super.onPause();
}
@Override
protected void onResume() {
      super.onResume();
      mHandler.post(mUpdateTimeTask);
}
```

Threads in the same VM interact and synchronize by the use of shared objects and monitors associated with these objects.

Handler - sendMessage()

```
public class MyThreadActivity extends Activity {
       private int mResults;
       private Handler messageHandler;

       void startHeavyDutyStuff() { // Called from onCreate()
           mMessageHandler = getHandler();
          // Here is the heavy-duty anonymous class (no name) thread
          Thread t = new Thread() {
             public void run() {
                 while (true) {
                    mResults = doSomethingExpensive();
                     //obtain results and send update to the handler associated with main/UI thread
                    messageHandler.sendMessage(Message.obtain(messageHandler, mResults));
                 }
             }
          };
          t.start();
       }
       public Handler getHandler() { return new MyHandler(this); }
       private static class MyHandler extends Handler {
          // Using a weak reference means you won't prevent garbage collection
          private final WeakReference<MainActivity> myClassWeakReference;
          public MyHandler(MainActivity myClassInstance) {
             myClassWeakReference = new WeakReference<MyThreadActivity>(myClassInstance);
          }
          @Override
          public void handleMessage(Message msg) {
             if (myClassWeakReference.get()!= null) {
              switch(msg.what) {    // mResults == msg.what
                     Case 60:
                            // handle msg and update UI etc.
                            break;
                     default:
              }
              String str = msg.toString(); // toast msg.what and msg.getWhen()
              Toast.makeText(getApplicationContext(), str, Toast.LENGTH_SHORT).show();
             }
          }
       }
}// end of class MyThreadActivity
```

```
/* Alternative solution
Message msg =
    mMessageHandler.obtainMessage();
msg.arg1 = mResults;
mMessageHandler.sendMessage(msg);
*/
```

http://stackoverflow.com/questions/11407943/this-handler-class-should-be-static-or-leaks-might-occur-incominghandler

static inner class which doesn't hold an implicit reference to the outer class

Timer, TimerTask & runOnUiThread

- A Timer and a TimerTask can schedule one-shot or recurring tasks for execution
- If we do not want to use a handler we can use the function runOnUiThread(new Runnable() {…
  - http://writecodeeasy.blogspot.se/2012/08/androidtutorial-timer-p1.html

```
// even smaller thread function which can do some work
AsyncTask.execute(new Runnable() {
    @Override
    public void run() {
        // All your networking logic should be here
    }
});
```

```
private Timer timer = new Timer();
private TimerTask timerTask = null;

timerTask = new TimerTask() {
     @Override
     public void run() {
        Log.d("TIMER", "Timer fired");
        // Causes the Runnable to be added to the message queue
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                doSomethingWhichCanUpdateTheGUI();
            }
        });
     }
};
timer.schedule(timerTask, 3000); // delay 3 sec, after this run task once
timer.scheduleAtFixedRate(timerTask, 3000, 5000); // delay 3 sec, after this run every 5th sec
timer.cancel(); // Cancels the Timer and all scheduled tasks.
timertask.cancel();  // Cancels the TimerTask and removes it from the Timer's queue.
```

Countdown task

- Good to have as a watchdog etc.

```
// Declare and start the count down timer task
private CountDownTimer mUploadMediaCountDownTimerTask;
myCountDownTimerTask();
/* http://developer.android.com/reference/android/os/CountDownTimer.html
Parameters
MillisInFuture - The number of millis in the future from the call to start() until the countdown
is done and onFinish() is called.
CountDownInterval - The interval along the way to receive onTick(long) callbacks. */
private void myCountDownTimerTask() {
    // 15 minutes with minute updates
    final int millisInFuture = 1000 * 60 * 15;
    final int countDownInterval = 1000 * 60;
    final Activity activity = getActivity();
    // first cancel ongoing task if available
    if(mUploadMediaCountDownTimerTask != null)
        mUploadMediaCountDownTimerTask.cancel();
    mUploadMediaCountDownTimerTask = new CountDownTimer(millisInFuture, countDownInterval)
    {
        public void onTick(long millisUntilFinished) {
            //Log.d(TAG, "seconds remaining until finish: " + millisUntilFinished / 1000);
        }
        public void onFinish() {
             // check if we should do something
             boolean finishWatchDog = startMyUploadMediaTask();

            // start countdown task again if we need to
            If(!finishWatchDog)
                mUploadMediaCountDownTimerTask.start();
        }
    };
    mUploadMediaCountDownTimerTask.start();
}
```

AsyncTask 1

- AsyncTask allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers
- AsyncTask must be subclassed in order to be used!

```
new DownloadFilesTask().execute(urlArr1, intArr2, longArr3); // executes the task

private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
     protected Long doInBackground(URL... urls) {
          int count = urls.length;
          long totalSize = 0;
          for (int i = 0; i < count; i++) {
               totalSize += Downloader.downloadFile(urls[i]);
               publishProgress((int) ((i / (float) count) * 100));
               // Escape early if cancel() is called
               if (isCancelled()) break;
          }
          return totalSize;
     }

     protected void onProgressUpdate(Integer... progress) {
          setProgressPercent(progress[0]);
     }

     protected void onPostExecute(Long result) {
          showDialog("Downloaded " + result + " bytes");
     }
}
```

Note the data types!
To mark a type as unused, simply use the type Void

The task can at any time be canceled with cancel(boolean). If calling isCancelled() returns true, then onCancelled() will be called instead of onPostExecute().

AsyncTask 2

- The three types used by an asynchronous task are the following
  - **Params**, the type of the parameters sent to the task upon execution.
  - **Progress**, the type of the progress units published during the background computation.
  - **Result**, the type of the result of the background computation.
- When an asynchronous task is executed, the task goes through 4 steps
  - **onPreExecute()**, invoked on the **<u>UI thread</u>** before the task is executed. This step is normally used to setup the task.
  - **doInBackground(Params...)**, invoked on the **<u>background thread</u>** immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time.
  - **onProgressUpdate(Progress...)**, invoked on the **<u>UI thread</u>** after a call to publishProgress(Progress...). The timing of the execution is undefined.
  - **onPostExecute(Result)**, invoked on the **<u>UI thread</u>** after the background computation finishes. The result of the background computation is passed to this step as a parameter.
- Canceling an AsyncTask is not easy!
  - http://vikaskanani.wordpress.com/2011/08/03/android-proper-way-to-cancel-asynctask/

AsyncTask 3

- AsyncTask scheduling varies between Android versions
  - Before 1.6, they run in sequence on a single thread.
  - From 1.6 to 2.3, they run in parallel on a thread pool.
  - Since 3.0, back to the old behaviour by default! They run in sequence
  - No parallelization by default on modern phones!
- Unless you execute them with
  - executeOnExecutor()with a ThreadPoolExecutor

```
public class ConcurrentAsyncTask {
     public static voidexecute(AsyncTask as) {
          if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
               as.execute();
          } else {
               as.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR
                    /* AsyncTask.SERIAL_EXECUTOR */);
          }
     }
}
```

Networking 1

- In the Android emulator the IP-address 10.0.2.2 provides access to your development machines localhost adress
  - May be useful if your development machine act as a server
- Android allows to access the network via the standard ”java.net.*” package. Therefore you can perform network operations via the standard Java Networking API
  - URLConnection, HTTPURLConnection, Socket, DatagramSocket, ...
- java.io provide classes for system I/O
  - InputStream, OutputStream (and many wrapper streams), IOException, …
- java.nio (New I/O)
  - low-level intense I/O in modern OS:s
  - http://en.wikipedia.org/wiki/New_I/O

Networking 2

- Android also contains the ”android.net.*” package beyond the java.net and the Apache HttpComponents (org.apache.http.*) classes (deprecated since API-23) which may be easier to use under some situations
  - http://hc.apache.org/
- The simplest way to use web content is with a WebView
- For apps using web services use the OKHttp or Volley or libraries instead
- android.bluetooth
  - BluetoothSocket, BluetoothServerSocket, …
- Permission to set in AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```

URL and HTTP Connection

- Some simple networking examples
- Rember that all code must run in a thread!

```
// a connection to a URL for reading or writing
BufferedInputStream in = null;
try {
        URL url = new URL("ftp://ftp://ftp.sunet.se/ls-lR.gz");
        URLConnection urlConnection = url.openConnection();
        in = new BufferedInputStream(urlConnection.getInputStream());
        readStream(in);
} catch (Exception e){
finally {
        // try catch here as well
        in.close();
}
```

```
// Used to send and receive data over the web
// Data may be of any type and length (here: downloading an image from internet
HttpURLConnection http = null;
InputStream istream = null;
try {
        URL text = new URL(urlStr);
        http = (HttpURLConnection) text.openConnection();
        istream = http.getInputStream();
        Bitmap bmImg = BitmapFactory.decodeStream(istream);
        imageView.setImageBitmap(bmImg);
} catch (Exception e){
} finally {
        // try catch here as well
        if(istream != null) istream.close();
        if(http != null) http.disconnect();
}
```

HttpDownload example

```
// Reading from a HttpURLConnection, saving to file
try {
        URL text = new URL(urlStr);
        http = (HttpURLConnection) text.openConnection();
        istream = http.getInputStream();
        byte[] buffer = new byte[1024];
        fos = this.openFileOutput (fileName, Activity.MODE_PRIVATE);
        int readSize = 0;
        while (readSize != -1) {
                readSize = istream.read(buffer);
                if (readSize > 0) {
                        fos.write(buffer, 0, readSize);
                }
        }
```

URI (Uniform Resource Identifier) vs. URL (Uniform Resource Locator) http://ajaxian.com/archives/uri-vs-url-whats-the-difference

Networking with AsyncTask 1

- There are a few threading rules that must be followed for the AsyncTask class to work properly, read more here:
- http://developer.android.com/reference/android/os/AsyncTask.html

```
// call the AsyncTask with: new mygoogleSearchTask().execute(String[], int[], String[]);

private class mygoogleSearchTask extends AsyncTask<String, Integer, String> {
      protected String doInBackground(String... searchKey) {
            String key = searchKey[0];
            try {
                  return SearchRequest(key);
            } catch(Exception e) {
                  Log.v("Exception google search", "Exception: " + e.getMessage());
                  return "";
            }
      }
      // This method runs on the UI thread, it receives progress updates
      // doInBackground worker method needs to call publishProgress(percent);
      protected void onProgressUpdate(Integer... progress) {
            mViewObjectThatCanDisplay.setProgressPercent(progress[0]);
      }
      // what to do when the doInBackground job is done
      protected void onPostExecute(String result) {
            try {
                  mObjectThatCanManage.ProcessResponse(result);
            } catch(Exception e) {
                  Log.v("Exception google search", "Exception: " + e.getMessage());
            }
      }
}
```

Networking with AsyncTask 2

- Using HTTP GET to retrive JSON or XML data from the Google search Representational State Transfer (REST) API

```
// url = http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=android
// searchString = android
public String SearchRequest(String searchString) throws MalformedURLException, IOException
{
     String newFeed = url + searchString;
     StringBuilder response = new StringBuilder();
     Log.v("gsearch", "gsearch url: " + newFeed);
     URL url = new URL(newFeed);
     HttpURLConnection httpconn = (HttpURLConnection) url.openConnection();
//   Further adjustments of the connection can be made as: httpconn.setXYZ();
//   Ex: httpconn.setRequestMethod("POST"); // POST = params are passed in the body of the request
//   Article: Upload Files from Android to a Website/Http Server using Post
//   http://www.codicode.com/art/upload_files_from_android_to_a_w.aspx
     if(httpconn.getResponseCode() == HttpURLConnection.HTTP_OK)
     {
          BufferedReader input = new BufferedReader(
                    new InputStreamReader(httpconn.getInputStream()), 8192);
          String strLine = null;
          while ((strLine = input.readLine()) != null) {
               response.append(strLine);
          }
          input.close();
     }
     return response.toString();
}
```

Remember that the cell phone got a false IP-address and cannot act as a server. The phone must always init the network session! If not using GCM (Google Cloud Messaging) which is a sync protocol

Lab review - Android Lab4

- List with topics you need to understand before next laboration
- You must be able or know how to
  - understand all the previous points from former labs
  - send and receive SMS
  - manage the new permission model in API-23 and above
  - use concurrent programming methods
  - use networking
  - use files (next presentation)
  - use telephony manager (next presentation)