When an app runs, get a single thread, the UI thread

- Long running code should not be run on this thread
  - If app does not return quickly, platform may kill the app

There is a way to get more threads running

- Only the main thread should update the UI
- All of your code now needs to be thread safe
- Note: the android UI toolkit is not thread safe!

AsyncTask class is one way to get do thread processing

- There also the Thread and Handler classes, underlying mechanisms
- AsyncTask helps use them with less "pain"

Required methods to overload:
  - doInBackground(...) -- does the actual work

Other methods to overload
  - onPreExecute() -- called by the UI thread, possibly sets up a progress bar ...
  - onPostExecute() -- called by the UI thread, clean up
  - onProgressUpdate(Progress ...) -- called by the UI thread
    - doInBackground() calls publishProgress(Progress...)
AsyncTask class -- can be cancelled
  □ Call the cancel() method -- returns boolean
    □ onCancelled() method will be called instead of onPostExecute(), on the UI thread
  □ doInBackground() should call isCancelled() if in a long loop

Issues for using the AsyncTask
  □ Task must be created on the UI thread
  □ UI threads calls execute() to run the task
  □ Task can be run only once. execute() called a second time => exception thrown
  □ getStatus() returns one value of AsyncTask.Status (PENDING, RUNNING, FINISHED)

AsyncTask <doInParam, onProgressParam, onPostParam>
  □ onPostParam is also the return type of doInBackground()

Book’s AsyncDemo program
Configuration Changes -- and AsyncTasks

- Activities are destroyed and re-created in many instances
  - orientation rotations
  - low memory situations ...
- What happens to AsyncTasks?
  - Continue running ...
- What happens to references to views? (think onPostExecute())
- Need a way to get new Activity to get access to "old" AsyncTask
- Activity
  - @Override Object onRetainNonConfigurationInstance()
    - Gets called just before onDestroy()
    - return the AsyncTask
  - getLastNonConfigurationInstance()
    - returns null or last object returned from OnRetain...
- With fragments ... setRetainInstance() ..
  - But still needs to get new activity
- Add a method to the class to set current context
  - onActivityCreated() can call this method
Chapter 47 Services

- Service Class -- to perform background tasks
  - started services, bound services and intent services
  - service tag in the manifest

- Started Services:
  - Started by an app and runs until stopped
  - Runs past the lifetime of the starting app
  - startService() method used to start them
  - Can stop self by calling stopSelf()
  - stopService() can be called by another app
  - Services have a high priority and are the last to be stopped

- Intent Services
  - Subclass of Service
  - Started by intents rather than startService()
  - onHandleIntent() queues requests to be handled
Bound Service

- Bound service allows interaction and results from starting app
  - Other services don’t
- Example: audio player, player is service, UI app controls the player via IPC
- "Client" starts and binds a service
- Service can be "unbound" when no longer needed

Service Methods

- onCreate() -- as expected
- onBind() or onStartCommand() or onHandleIntent() -- depends on kinds of service
- onDestroy()

onStartCommand returns one of:

- START_NOT_STICKY -- not restarted if destroyed by framework
- START_STICKY -- restart requested
- START_REDELIVER_INTEN -- service was destroyed after returning

Service can be started on system start with appropriate manifest data
Sample programs: Chapters 48, 49 and 50

48 -- Started Service (ServiceExample)
- Service is started by an intent
- Service code uses an AsyncTask
- This service doesn’t update the UI
- Watch this service via the Logcat

49 -- Bound Service (LocalBound)
- Activity can call methods in the service
- Service in this case has no AsyncTask
- Can the service update the UI?
  - Can give the activity a reference (setMethod)
  - Activity.runOnUiThread(Runnable action)
  - Activity needs to let service know if of changes

50 -- Remote Bound Service (RemoteBound)
- Sending messages rather than direct calls
- Sending a message may throw an exception
- This service does have access for toasts!

<< Not Done >>
Chapter 15 -- Services ...

Threading ....
- We have been working on the UI thread
- Need to do things quickly and return
- 5 second delay and you could possibly get a "force close" message
- How to do operations that take a longer time to do?
  - need to run on a non-UI thread
  - HTTP processing is an example ...

HTTP operations:
- Simple one (using UI thread) and HttpClient (org.apache.http.client.HttpClient)
  1: create a HttpClient
  2: Instantiate a new HTTP method (PostMethod or GetMethod)
  3: Set HTTP parameter names/values.
  4: Execute HTTP call using the HttpClient
  5: Process the HTTP response
HttpClient -- Simple version

Book's SimpleHTTPGet demo program:

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); setContentView(R.layout.main);
    BufferedReader in = null;
    try {
        HttpClient client = new DefaultHttpClient();
       HttpGet request = new HttpGet("http://code.google.com/android/");
        HttpResponse response = client.execute(request);
        in = new BufferedReader(new InputStreamReader(
            response.getEntity().getContent()));
        StringBuffer sb = new StringBuffer(""");
        String line = "";
        String NL = System.getProperty("line.separator");
        while ((line = in.readLine()) != null) { sb.append(line + NL); }  
        in.close();
        String page = sb.toString();
        System.out.println(page);
    } catch (Exception e) { // TODO Auto-generated catch block
        e.printStackTrace();
    } finally {
        if (in != null) {
            try { in.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
        }
    }
}
HttpClient -- Simple Version (page 2)

- UI thread does the work -- not good
- Requires the try/catch/finally block
- Can use this from other places that are not on the UI thread!
- new HttpGet("http://path/to/document/name?name=value...")
  - Maximum length of string: 2048 characters
- new HttpPost() to get a post method

```java
    HttpClient client = new DefaultHttpClient();
    HttpPost request = new HttpPost("http://mysomewebsite.com/services/doSomething.do");
    List<NameValuePair> postParameters = new ArrayList<NameValuePair>();
    postParameters.add(new BasicNameValuePair("first", "param value one"));
    postParameters.add(new BasicNameValuePair("issuenum", "10317"));
    postParameters.add(new BasicNameValuePair("username", "dave"));
    UrlEncodedFormEntity formEntity = new UrlEncodedFormEntity(postParameters);

    request.setEntity(formEntity);
    HttpResponse response = client.execute(request);
```

- Book’s SimpleHTTPPost demo compiles but doesn’t run ...
- Can do multi-part requests -- see book and MultiPartPost demo program
What do you get with HTTP?

- Regular HTML Text
- Data (images, ...)

What about web services -- use HTTP to transport more than just web pages

- SOAP -- Simple Object Access Protocol
  - Uses XML in messages
  - Read about SOAP in many places
  - No direct support for SOAP in android

- JSON -- JavaScript Object Notation
  - Common use in Mobile applications/servers
  - Supported in several ways
  - org.json: JSONArray, JSONObject, JSONStringer, JSONTokener
    - Builds Objects you can use...
  - android.util.JsonReader/JsonWriter/JsonToken
    - Allows your code to more easily parse JSON.
Example JSON

[
    {
        "id": 912345678901,
        "text": "How do I read JSON on Android?",
        "geo": null,
        "user": {
            "name": "android_newb",
            "followers_count": 41
        }
    },
    {
        "id": 912345678902,
        "text": "@android_newb just use android.util.JsonReader!",
        "geo": [50.454722, -104.606667],
        "user": {
            "name": "jesse",
            "followers_count": 2
        }
    }
]
Other HTTP Issues

Timeouts -- a fact of life on the web ...

- How do you handle them?
  - Can set timeout values for HttpClient
  - Throws exceptions on timeout or connection errors
  - Can implement retries
    - while () { try { ... execute... } }
  - read book: Listing 15-5

How many HttpClient objects should you have for your app?

- Most likely one
- Need to deal with possible threading issues
  - Java -> synchronized methods

public class SynchronizedCounter {
    private int c = 0;
    public synchronized void increment() { c++; }
    public synchronized void decrement() { c--; }
    public synchronized int value() { return c; }
}

- Provides mutual exclusion for all synchronized methods in the class
  - Any thread in any synchronized method, another thread wait
public class CustomHttpClient { 
    private static HttpClient customHttpClient;

    /** A private Constructor prevents any other class from instantiating. */
    private CustomHttpClient() {   }

    public static synchronized HttpClient getHttpClient() {
        if (customHttpClient == null) {
            HttpParams params = new BasicHttpParams();
            ... // next page
            customHttpClient = new DefaultHttpClient(conMgr, params);
        }
        return customHttpClient;
    }

    public Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException();
    }
}

Set up so it can’t be instantiated, cloned

Guards against creating a new one at the same time.
HttpParams params = new BasicHttpParams();
HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
HttpProtocolParams.setContentCharset(params, HTTP.DEFAULT_CONTENT_CHARSET);
HttpProtocolParams.setUseExpectContinue(params, true);

HttpProtocolParams.setUserAgent(params,
    "Mozilla/5.0 (Linux; U; Android 2.2.1; en-us; Nexus One Build/FRG83)" +
    " AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1"
);

ConnManagerParams.setTimeout(params, 1000);

HttpConnectionParams.setConnectionTimeout(params, 5000);
HttpConnectionParams.setSoTimeout(params, 10000);

SchemeRegistry schReg = new SchemeRegistry();
schReg.register(new Scheme("http", PlainSocketFactory.getSocketFactory(), 80));
schReg.register(new Scheme("https", SSLSocketFactory.getSocketFactory(), 443));

ClientConnectionManager conMgr = new ThreadSafeClientConnManager(params, schReg);

customHttpClient = new DefaultHttpClient(conMgr, params);
Simple use of CustomHttpClient

```java
httpClient = CustomHttpClient.getHttpClient();
try {
    HttpGet request = new HttpGet("http://www.google.com/");
    HttpParams params = request.getParams();
    HttpConnectionParams.setSoTimeout(params, 60000);  // 1 minute
    request.setParams(params);
    Log.v("connection timeout", String.valueOf(
            HttpConnectionParams.getConnectionTimeout(params)));
    Log.v("socket timeout", String.valueOf(HttpConnectionParams.getSoTimeout(params)));

    String page = httpClient.execute(request, new BasicResponseHandler());
    System.out.println(page);
} catch (IOException e) {
    // covers:
    //   ClientProtocolException
    //   ConnectTimeoutException
    //   ConnectionPoolTimeoutException
    //   SocketTimeoutException
    e.printStackTrace();
}

- Read "Fun With Timeouts" and "Using the AndroidHttpClient"
- There as also a HttpURLConnection class ... for more control
- Also, check out AndroidHttpClient ... read book
Using Background Threads

First simple method ... using android.os.AsyncTask
  □ public abstract class AsyncTask<Params, Progress, Result> { ... }
  □ onPreExecute() -- invoked on UI thread (UI thread must create it)
  □ doInBackground(Params...)
    □ Uses the Java varargs
    □ Runs the method on a different (non UI) thread
    □ Result must be returned by this method
    □ Can call PublishProgress(Progress...)
  □ onProgressUpdate(Progress...) called on UI thread in response to PublishProgress()
  □ onPostExecute(Result) -- called on UI thread after background job done
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));
        }
        return totalSize;
    }

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

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes"); // example call
        dismissProgressView(); // example call
    }
}

□ Not complete .... but gives you the idea.
□ could have onPreExecute() show the progress dialog.
AsyncTask -- other methods ..

- `AsyncTask.Status getStatus()` -- (.FINISHED, .PENDING, .RUNNING)
- `boolean cancel(mayInterruptIfRunning)`
- `boolean IsCancelled()`
- `Result get()` -- waits and gets result
- `Result get(timeout, unit)` -- waits at most the timeout time ...
- `AsyncTask<Params,Progress,Result> execute(Params... params)` -- (returns itself)

Typical use ...
- Create a class that extends it... why?
  - `doInBackground` is protected!
public class DownloadImageTask extends AsyncTask<String, Integer, Bitmap> {
    private Context mContext;

    DownloadImageTask(Context context) {
        mContext = context;
    }

    protected void onPreExecute() {
        // We could do some setup work here before doInBackground() runs
    }
}
protected Bitmap doInBackground(String... urls) {
    Log.v("doInBackground", "doing download of image");
    return downloadImage(urls);
}

protected void onProgressUpdate(Integer... progress) {
    TextView mText = (TextView) ((Activity) mContext).findViewByld(R.id.text);
    mText.setText("Progress so far: " + progress[0]);
}

protected void onPostExecute(Bitmap result) {
    if(result != null) {
        ImageView mImage = (ImageView) ((Activity) mContext).findViewByld(R.id.image);
        mImage.setImageBitmap(result);
    } else {
        TextView errorMsg = (TextView) ((Activity) mContext).findViewByld(R.id.errorMsg);
        errorMsg.setText("Problem downloading image. Please try again later.");
    }
}
```java
private Bitmap downloadImage(String... urls) {
    HttpClient httpClient = CustomHttpClient.getHttpClient();
    try {
        HttpClient request = new HttpClient(urls[0]);
        HttpParams params = new BasicHttpParams();
        HttpConnectionParams.setSoTimeout(params, 60000);  // 1 minute
        request.setParams(params);
        publishProgress(25);
        HttpResponse response = httpClient.execute(request);
        publishProgress(50);
        byte[] image = EntityUtils.toByteArray(response.getEntity());
        publishProgress(75);
        Bitmap mBitmap = BitmapFactory.decodeByteArray(image, 0, image.length);
        publishProgress(100);
        return mBitmap;
    } catch (IOException e) {
        // covers:
        //     ClientProtocolException
        //     ConnectTimeoutException
        //     ConnectionPoolTimeoutException
        //     SocketTimeoutException
        e.printStackTrace;
    }
    return null;
}
```
Configuration Changes -- and AsyncTasks

Activities are destroyed and re-created in many instances
- orientation rotations
- low memory situations ...

What happens to AsyncTasks?
- Continue running ...

What happens to references to views? (think onPostExecute())

Need a way to get new Activity to get access to "old" AsyncTask

Old Activity way
- @Override Object onRetainNonConfigurationInstance()
  - Gets called just before onDestroy()
- getLastNonConfigurationInstance()
  - returns null or last object returned from OnRetain...

With fragments ... setRetainInstance() ..
  - But still needs to get new activity

Add a method to the class to set current context
  - onActivityCreated() can call this method
Code for config changes with Async tasks

```java
DownloadImageTask(Context context) {
    // as before
    mContext = context;
}

// Called from main thread to re-attach
protected void setContext(Context context) {
    mContext = context;
    if (progress >= 0) {
        publishProgress(this.progress);
    }
}

// In the Activity
public class HttpActivity extends Activity {
    private DownloadImageTask diTask;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        if ((diTask = (DownloadImageTask) getLastNonConfigurationInstance()) != null) {
            diTask.setContext(this); // Give my AsyncTask the new Activity reference
            if (diTask.getStatus() == AsyncTask.Status.FINISHED)
                diTask.setImageInView();
        }
    }
}
```
DownloadManager -- an android class for downloading files

- Android 2.3 and later
- Uses a background thread to handle long-running HTTP downloads
- getSystemService(DOWNLOAD_SERVICE)
- Should read the code and the book ... 

Services .... 

- Like UNIX daemons, Windows services
- Different life cycle than activities
- No associated UI
- Is not a separate process
- Is not a thread
- Runs in the "background"
- Two types of services
  - Local -- only for the "hosting application"
  - Remote -- can be accessed by other applications (AIDL-supporting services)
- Context.startService() -- starts a service
- Context.stopService() -- stops a service
- stopSelf() -- called in the service stops the service
public class MyService extends Service { ..... }

public void onCreate() -- similar to an activity

public void onDestroy() -- similar to an activity

no onPause(), no onResume()

public int onStartCommand(Intent intent, int flags, int startId)

Service is started by an intent -- gives access to it

return START_STICKY -- can be scheduled for a restart

return START_NOT_STICKY -- don’t schedule for a restart

Flags -- 0 or START_FLAG_REDELIVERY or START_FLAG_RETRY

startId -- a unique id for this call to start

Can be called more than once during the lifetime of the service

public IBinder onBind(Intent intent)

For Remote services, local ones can return null

public void stopSelf(int startId) -- need to keep the startId for this

Many others ...

Often times services use the notification to give feedback to the user

Service may be killed if more memory is needed.

startForeground()/stopForeground() routines
public class BackgroundService extends Service {
    private static final String TAG = "BackgroundService";
    private NotificationManager notificationMgr;
    private ThreadGroup myThreads = new ThreadGroup("ServiceWorker");

    @Override
    public void onCreate() {
        super.onCreate();
        Log.v(TAG, "in onCreate()");
        notificationMgr = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        displayNotificationMessage("Background Service is running");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        super.onStartCommand(intent, flags, startId);
        int counter = intent.getExtras().getInt("counter");
        Log.v(TAG, "in onStartCommand(), counter = " + counter + ", startId = " + startId);
        new Thread(myThreads, new ServiceWorker(counter), "BackgroundService")
            .start();
        return START_STICKY;
    }
}
class ServiceWorker implements Runnable {

    private int counter = -1;
    public ServiceWorker(int counter) {
        this.counter = counter;
    }

    public void run() {
        final String TAG2 = "ServiceWorker:" + Thread.currentThread().getId();
        // do background processing here...
        try {
            Log.v(TAG2, "sleeping for 10 seconds. counter = " + counter);

            Thread.sleep(10000);
            Log.v(TAG2, "... waking up");
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            Log.v(TAG2, "... sleep interrupted");
        }
    }
}
@Override
public void onDestroy()
{
    Log.v(TAG, "in onDestroy(). Interrupting threads and cancelling notifications");
    myThreads.interrupt();
    notificationMgr.cancelAll();
    super.onDestroy();
}

@Override
public IBinder onBind(Intent intent) {
    Log.v(TAG, "in onBind()");
    return null;
}

private void displayNotificationMessage(String message)
{
    Notification notification = new Notification(R.drawable.emo_im_winking,
        message, System.currentTimeMillis());
    notification.flags = Notification.FLAG_NO_CLEAR;
    PendingIntent contentIntent =
        PendingIntent.getActivity(this, 0, new Intent(this, MainActivity.class), 0);
    notification.setLatestEventInfo(this, TAG, message, contentIntent);
    notificationMgr.notify(0, notification);
}
Remote services need a way to talk with clients...

- AIDL is short for Android Interface Definition Language
- AIDL is a kind of Remote Procedure Call (RPC)
- Appears in Java as an interface definition
- Book has examples in Chapter 15 of defining and using AIDL services
- Will return when needed later...

Chapter 16 ... exploring packages

- A package -> process
- Read more about package signing
- Packages can share data by using the same user ID
- Need to be careful ... should be a related group of apps
- This chapter also talks about library projects ... good stuff
- Libraries can have test apps (as can other packages)
- No time to go into detail here .... read it
Chapter 17 -- Handlers

Handlers are an important concept in Android
- Threading issues arise again
- Main objects: Activity, service, ContentProvider, BroadcastReceiver
- These all get run on the "main thread"
- These must specifically request new threads if needed.
  - AsyncTask did get a new thread.
- Services and Content Providers most often want other threads.
- java.lang.Thread class is the primary method of threading in Java
  - There are thread groups as we saw before
  - Objects with synchronized methods can help synchronize threads
  - Not going to talk much more about thread details here
- How does one "communicate" between threads?
  - Can use those synchronized objects ...
    - Can cause threads to be blocked for a long time.
- Android provides a Handler class that provides thread communication ...
Handler Basics

The basic thing a handler does is allow one to send and receive messages

- A handler is associated with the starting thread (handleMessage() method)
- A handler may send "itself" a delayed message
- A handler may start a "worker thread"
  - The worker thread can then send a message to the starting thread

Book has demo app. A highly modified version is on-line: PhilsTestHandlers.tgz

Book’s first example just sends a delayed message:

- sendMessageDelayed(m,timeinmillisseconds)

This demo has three kinds of handlers: (code shown in Eclipse)

- deferWorkHandler.java -- works by itself, needs access to activity
- workerThreadRunnable and ReportStatusHandler -- working together
- PhilsHandler -- requires user to override handleMessage() method
- SimpleService -- sends messages to main thread via a Handler

Other notes:

- Main TextView inside a ScrollView (scrolling to bottom not perfect)
- workerThreadRunnable has been changed from the book (book’s didn’t work)
- menu processing is different, removed a menu item
- Handlers can be used to talk with local services, notice synchronized methods
android.os.Message

Fields:
- public int arg1
- public int arg2
- public Object obj
- public Messenger replyTo
- public int what

Methods:
- Bundle getData()
- void setData(Bundle)

Static Factory Methods
- obtain(Handler, what, arg1, arg2, obj)
- obtain(Handler, what, obj)
- ....