Call web services and asynch processing

HTTP requests
Background threads
The UI thread

Android web service clients

- ❖ Calling web services uses the HTTP protocol
- ❖ Android supports HTTP connections for REST
  - Usually is used the HttpURLConnection class
    - Supports all HTTP verbs, Http headers, Cookies, Timeouts, …
  - Payload requests (POST or PUT) are transmitted with the request, and before getting the input (the response)
  - Needs a separated thread to run (enforced after API 9), and a manifest permission
    - <uses-permission android:name="android.permission.INTERNET"/>
  - After API 27 only HTTPS is allowed by default
    - use android:usesCleartextTraffic="true" in the <application> manifest tag for allowing HTTP
- ❖ Android parsers for processing the response
  - Json and XML parsers - JSONObject, XMLPullParser
  - Google Gson external library allows
    - Translating between Json strings and Java objects

A HTTP request

❖ Create an explicit thread for an HTTP request

Example: AddUser using a POST and a userName String parameter

```
private class AddUser implements Runnable {
  String address = null;  String uname = null;

  AddUser(String baseAddress, String userName) {
    address = baseAddress;  uname = userName;
  }

  @Override public void run() {
    URL url;    HttpURLConnection urlConnection = null;
   try {
     url = new URL("http://" + address + ":8701/Rest/users");
     urlConnection = (HttpURLConnection) url.openConnection();
     urlConnection.setDoOutput(true);
     urlConnection.setDoInput(true);
     urlConnection.setRequestMethod("POST");
     urlConnection.setRequestProperty("Content-Type", "application/json");
     urlConnection.setUseCaches (false);

     DataOutputStream outputStream = new DataOutputStream(urlConnection.getOutputStream());
     String payload = "\"" + uname + "\"";
     outputStream.writeBytes(payload);
     outputStream.flush();
     outputStream.close();

     int responseCode = urlConnection.getResponseCode();
     if (responseCode == 200)
       String response = readStream(urlConnection.getInputStream());   // … and transmit to UI
  }
  catch (Exception e) {  . . .     // treat the exception
  }
  finally {
     if(urlConnection != null)    urlConnection.disconnect();
  }
} }
```

configure

payload

response

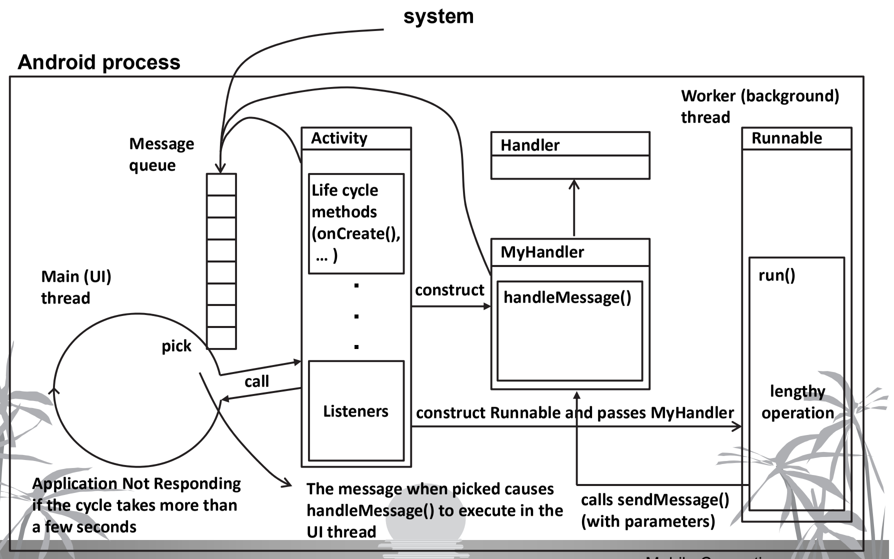
```
private String readStream(InputStream in) {
  BufferedReader reader = null;
  String line;
  StringBuilder response = new StringBuilder();

  try {
    reader = new BufferedReader(new InputStreamReader(in));
    while ((line = reader.readLine()) != null) {
      response.append(line);
    }
  }
  catch (IOException e) {
    return e.getMessage();
  }
  finally {
    if (reader != null) {
      try {
        reader.close();
      }
      catch (IOException e) {
        response = new StringBuilder(e.getMessage());
      }
    }
  }
  return response.toString();
}
```

invoking

```
AddUser addUser = new AddUser(address, name);
Thread thr = new Thread(addUser);
thr.start();
```

Thread communication


system
Android process
Worker (background) thread
Message queue
Activity
Handler
Runnable
Life cycle methods (onCreate(), …)
MyHandler
Main (UI) thread
construct
handleMessage()
run()
pick
call
Listeners
construct Runnable and passes MyHandler
lengthy operation
Application Not Responding if the cycle takes more than a few seconds
The message when picked causes handleMessage() to execute in the UI thread
calls sendMessage() (with parameters)

Example

```
… Activity …
… e.g. in a Button or Menu listener …

myHandler = new MyHandler(this);
worker = new Thread(new MyRunnable(myHandler));
worker.start();
…
```

```
… MyHandler class …
MyActivity uiActivity;

public MyHandler(MyActivity context) {
   uiActivity = context;
}

@Override
public void handleMessage(Message m) {
   String s = m.getData().getString("msgstr");
   // … uiActivity.doSomething(s);
}
```

```
… MyRunnable class …
Handler uiHandler;

public MyRunnable(Handler handler) {
   uiHandler = handler;
}

public void run() {
   . . .
   interact();
   . . .
}

public void interact() {
   Message m = uiHandler.obtainMessage();
   m.setData( createBundleFromStr("something") );
   uiHandler.sendMessage(m);
}
```

The message linked to the handleMessage() method must be obtained with obtainMessage() from the Handler object. Other data can be transported by this message as a Bundle. We can use the setData() and getData() methods of the message object to attach and extract the Bundle.

```
public Bundle createBundleFromStr(String val) {
   Bundle b = new Bundle();
   b.putString("msgstr", val);
   return b;
}
```

Another simpler example with a Handler

```
// Initialize a handler on the main thread.
private Handler handler = new Handler();

private void mainProcessing()  {
   Thread thread = new Thread(doBackgroundThreadProcessing,  "Background");
    thread.start();
}

private Runnable doBackgroundThreadProcessing = new Runnable() {
      public void run() {
            [ ... Time consuming operations ... ]
            handler.post(doUpdateGUI);
      }
};

// Runnable that executes the update GUI method on the UI thread.
private Runnable doUpdateGUI = new Runnable() {
      public void run() {
         [ ... Open a dialog or modify a GUI element ... ]
      }
};
```

With this simpler approach there is no parameters transported between threads.

The post() method just creates a message linked with a Runnable.

Asynchronous tasks

❖ **Convenience class to a background thread**

```
AsyncTask<[Input Parameter Type], [Progress Report Type], [Result Type]>
```

```
private class MyAsyncTask extends AsyncTask<String, Integer,  Integer>  {
   @Override
   protected void onProgressUpdate(Integer progress) {
         // [... Update progress bar, Notification, or other UI element ...]
   }
   @Override
   protected void onPostExecute(Integer result) {
         // [... Report results via UI update, Dialog, or notification ...]
   }
   @Override
   protected Integer doInBackground(String parameter) {
      int myProgress = 0;
            // [... Perform background processing task, update myProgress ...]
      PublishProgress(myProgress);
            // [... Continue performing background processing task ...]
            // Return the value to be passed to onPostExecute
      return result;
   }
}
```

Creating and running the task

```
new MyAsyncTask()
            .execute("inputString");
```

doInBackground() is executed by a background thread when AsyncTask is executed.
onPostExecute() is executed by the UI thread when doInBackground() finishes.
onProgressUpdate() is also executed by the UI thread when the background thread calls PublishProgress().
There is a parameter passing between these methods.