Call web services and async 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
  - It has the HttpURLConnection class
    - Supports all HTTP verbs, Http headers, Cookies, Timeouts, …
  - Payloads in requests (POST or PUT) are transmitted with the request, and before getting the result (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

```
private class AddUser(val address: String, val uname: String) : Runnable {

  override run() {
    val url: URL?  = null
    val urlConnection: HttpURLConnection? = null
    try {
      url = URL("http://" + address + ":8701/Rest/users")
      urlConnection = url.openConnection() as HttpURLConnection
      urlConnection!!.setDoOutput(true)
      urlConnection!!.setDoInput(true)
      urlConnection!!.setRequestMethod("POST")
      urlConnection.!!setRequestProperty("Content-Type", "application/json")
      urlConnection!!.setUseCaches (false)

      val outputStream =  DataOutputStream(urlConnection!!.getOutputStream())
      val payload = "\"" + uname + "\""
      outputStream.writeBytes(payload)
      outputStream.flush()
      outputStream.close()

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

configure

payload

response

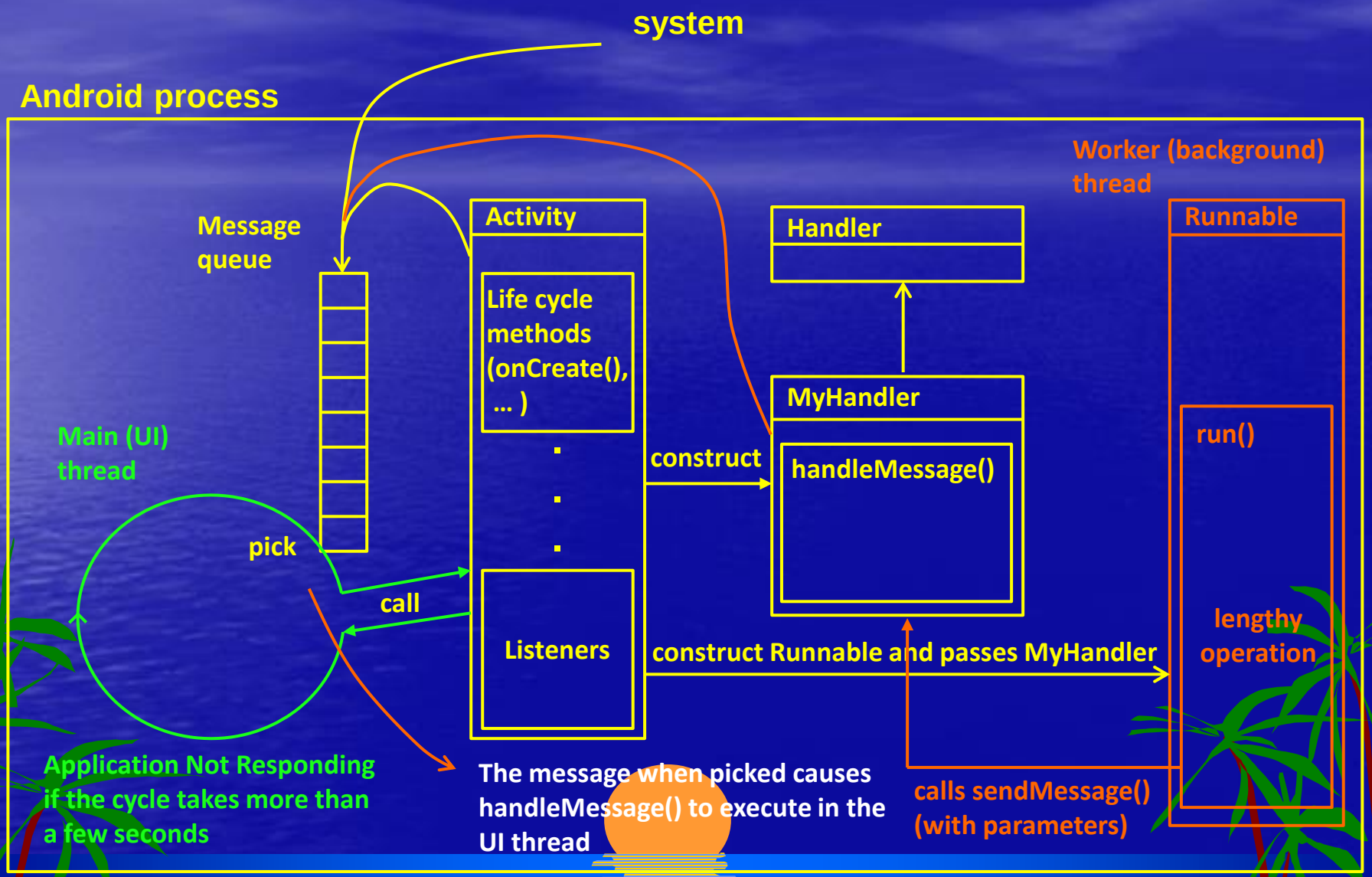
```
fun readStream(InputStream in): String {
    val reader: BufferedReader? = null
    var line: String? = null
    var response = StringBuilder()

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

invoking

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

Thread communication



Example

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

val myHandler = MyHandler(this)
val worker = Thread(MyRunnable(myHandler))
    worker.start()
...
```

```
... MyRunnable class ...
MyRunnable(val uiHandler: Handler) {

  fun run() {
    . . .
    interact()
    . . .
  }

  fun interact() {
    val m = uiHandler.obtainMessage()   // a message
    m.setData( createBundleFromStr("something") )
    uiHandler.sendMessage(m)
  }
}
```

```
... MyHandler class ...
MyHandler(val uiActivity: MyActivity) {

   override handleMessage(m: Message) {
      val s = m.getData().getString("msgstr")
      // ... uiActivity.doSomething(s)
   }
}
```

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.

```
fun createBundleFromStr(value: String): Bundle {
   val b = Bundle()
   b.putString("msgstr", value)
   return b
}
```

Another simple example with a Handler

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

private fun mainProcessing()  {
   val thread = Thread(doBackgroundThreadProcessing,  "Background")
    thread.start()
}

private val doBackgroundThreadProcessing = Runnable() {
      override run() {
            [ ... Time consuming operations ... ]
            handler.post(doUpdateGUI)
      }
};

// Runnable that executes the update GUI method on the UI thread.
val doUpdateGUI = Runnable() {
      override 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.

This mechanism is implented In the Activity method:
runOnUiThread(Runnable)

Asynchronous tasks

Convenience class to a background thread

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

```
private class MyAsyncTask() : AsyncTask<String, Int,  Int>  {
   override onProgressUpdate(progress: Int) {
         // [... Update progress bar, Notification, or another UI element ...]
   }

   override onPostExecute(result: Int) {
         // [... Report results via UI update, Dialog, or notification ...]
   }

   override doInBackground(parameter: String ): Int {
      val 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
   }
}
```

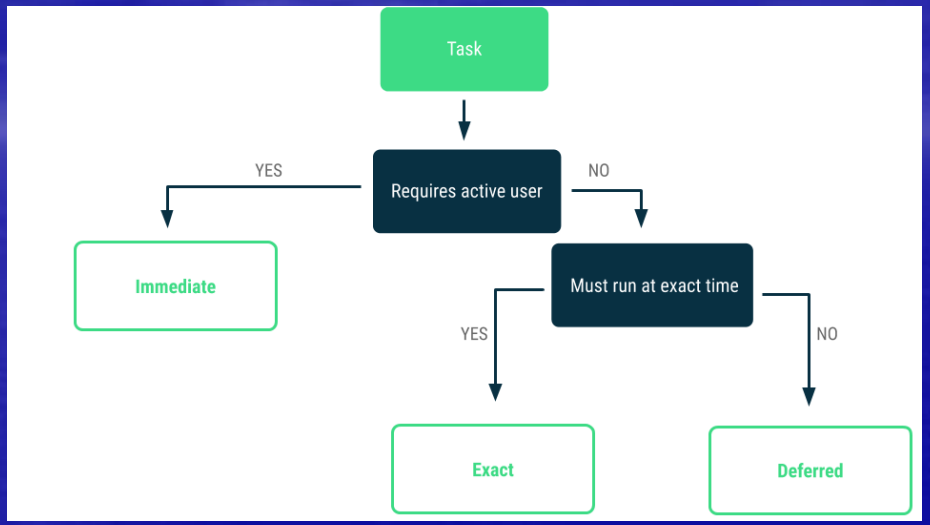
Creating and running the task

```
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 parameter passing between these methods.

Background tasks

- Many apps need background processing



  - Immediate:
    - Thread, AsyncTask, ThreadPoolExecutor
  - Without the user (Exact time or Deferred):
    - JobScheduler
    - AlarmManager
    - WorkManager (JetPack)
  - HTTP requests: External libraries: Retrofit, Volley