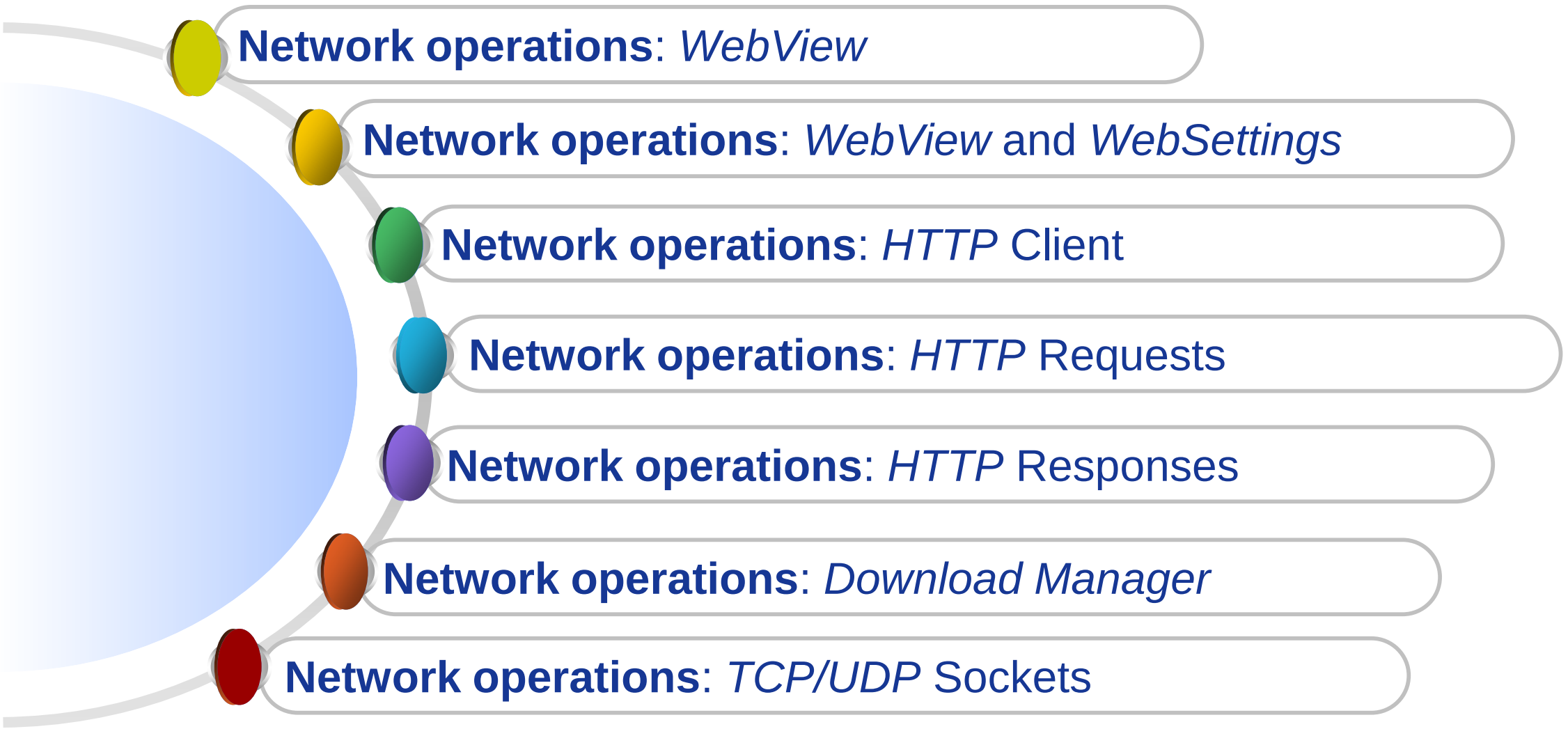




Chapter 12. Network Operations

Outline


Network operations: *WebView*
Network operations: *WebView* and *WebSettings*
Network operations: *HTTP* Client
Network operations: *HTTP* Requests
Network operations: *HTTP* Responses
Network operations: *Download Manager*
Network operations: *TCP/UDP* Sockets

Android: Network Operations

- In order to perform network operations (also the one described earlier), specific **permissions** must be set on the **AndroidManifest.xml**.

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

- Failure in setting the permissions will cause the system to throw a **run-time** exception …

Android: Network Operations

- Before the application attempts to connect to the network, it should check to see whether a network connection is available using **`getActiveNetworkInfo()`** and **`isConnected()`** …

```
ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnected()) {
        // fetch data
    } else {
        // display error
    }
```

Android: WebView Usage

WebView → A **View** that displays web pages, including simple browsing methods (history, zoom in/out/ search, etc).

Implemented by the WebView class

```
public WebView(Context contex)
```

Main methods:

- public void **loadUrl**(String url) → load the HTML page at url
- public void **loadData**(String data, String mimeType, string encoding) → load the HTML page contained in data

Android: WebView Usage

Android: WebView Usage

By default, the WebView UI does not include any navigation button …However, **callbacks** methods are defined:

- public void **goBack**()
- public void **goForward**()
- public void **reload**()
- public void **clearHistory**()

Android: WebView Usage

It is possible to modify the visualization options of a WebView through the **WebSettings** class.

```
public WebSettings getSettings()
```

Some options:

- void **setJavaScriptEnabled**(boolean)
- void **setBuildInZoomControls**(boolean)
- void **setDefaultFontSize**(int)

Android: Download Manager

DownloadManager → System service that handles long-run HTTP downloads.

- The client can specify the file to be downloaded through an **URI** (path).
- Download is conducted in **background** (with retries)
- Broadcast Intent action is sent to notify when the download completes.

```
DownloadManager dm=(DownloadManager)
getSystemService(DOWNLOAD_SERVICE);
```

Android: Download Manager

- The Request class is used to specify a download request to the Download Manager.

```
Request request=new DownloadManager.Request(Uri.parse(address));
```

Main methods of the **DownloadManager**

- long **enqueue**(DownloadManager.Request)
- Cursor **query**(DownloadManager.Query)
- ParcelFileDescriptor **openDownloadedFile**(long)

Android: HTTP Classes

HTTP (HyperText Tranfer Protocol): Network protocol for exchange/transfer data (hypertext)

Request/Resonse Communication Model

MAIN COMMANDS

- HEAD
- GET
- POST
- PUT
- DELETE
- TRACE
- CONNECT

Android: HTTP Classes

HTTP (HyperText Tranfer Protocol): Network protocol for exchange/transfer data (hypertext)

Two implementations of HTTP Clients for Android:

- **`HTTPClient`** → Complete extendable HTTP Client suitable for web browser (not supported anymore?)
- **`HTTPUrlConnection`** → Light-weight implementation, suitable for client-server networking applications (recommended by Google)

In both cases, HTTP connections must be managed on a separate thread, e.g. using **AsynchTask** (not the UI thread!).

Android: HTTP (Abstract) Classes

- **HttpClient** → Interface for an HTTP client
- **HttpRequest** → Interface for an HTTP request
- **HttpResponse** → Interface for an HTTP response
- **ResponseHandler<T>** → Handler that creates an object <T> from an HTTP Response
- **HttpContext** → Context of the HTTP Request (request+response+data)

Android: HTTP Classes

- **HttpClient** → Interface for an HTTP client
  (DefaultHttpClient → implementation of an HttpClient)

```
HttpClient client=new DefaultHttpClient();
```

Main method:

The public method **execute**(…) performs an HTTP request, and allows to process an HTTP reply from the HTTP server.

One of the signature of **execute**()

```
abstract<T> T execute(HttpUriRequest request,
ResponseHandler <T> responseHandler)
```

Android: HTTP Classes

- **HttpRequest** → Interface for an HTTP request

Two implementations:

HttpGet → implements the **GET** HTTP method

```
HttpGet request=new HttpGet(String address);
HttpGet request=new HttpGet(URI address);
```

HttpPost → Implements the **POST** HTTP method

Android: HTTP Classes

- **ResponseHandler <T>** → Interface for creating an object <T> from an HttpResponse, obtained after having executed an HttpRequest.

Method to override

```
public abstract T handleResponse (HttpResponse res)
```

Generally, <T> is a String (HTML code) …

Android: HTTP Classes

- **HttpPost** → Implements the **POST** HTTP method

```
HttpPost request=new HttpPost(String address);
```

```
HttpPost request=new HttpPost(URI address);
```

Encapsulating a parameter …

```
List<NomeValuePair> par=new ArrayList<NomeValuePair>()
par.add(new BasicNameValuePair("name","Marco");
HttpEntity postEntity=new UrlEncodedFormEntity(par);
request.setEntity(postEntity);
```

Android: HTTP Classes

Basic HTTPClient Request-Response Application …

```
HttpClient client=new DefaultHttpClient();
HttpGet request=new HttpGet();
request.setURI("http//www.cs.unibo.it");
try {
  client.execute(request, responseHandler);
} catch (ClientProtocolException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Android: HTTP Classes

Basic HTTPClient Request-Response Application …

```
class MyResponseHandler implements ResponseHandler<String> {
       @Override
       public String handleResponse(HttpResponse response) {
          InputStream content=response.getEntity().getContent();
          byte[] buffer=new byte[1024];
          int numRead=0;
          ByteArrayOutputStream stream=new
ByteArrayOutputStream();
          while ((numRead=content.read(buffer))!=-1)
              stream.write(buffer, 0, numRead);
          content.close();
          String result=new String(stream.toByteArray());
          return result;
       }
    }
```

Android: HTTP Classes

HTTPUrlConnection → HTTP component to send and receive streaming data over the web.

1. Obtain a new **HttpURLConnection** by calling the **URL.openConnection()**

```
URL url = new URL("http://www.android.com/");
HttpURLConnection urlConnection = (HttpURLConnection)
                                  url.openConnection();
```

2. Prepare the request, set the options (e.g. session cookies)

3. For **POST** commands, invoke **`setDoOutput(true).`** Transmit data by writing to the stream returned by **`getOutputStream().`**

Android: HTTP Classes

HTTPUrlConnection → HTTP component to send and receive streaming data over the web.

4. Read the response (data+header). The response body may be read from the stream returned by **`getInputStream()`**.

```
InputStream in = new
BufferedInputStream(urlConnection.getInputStream());
```

5. Close the session when ending reading the stream through **`disconnect`**().

```
urlConnection.disconnect();
```

Android: TCP/IP Communication

TCP/UDP Communication → Android applications can use java.net.Socket facilities.

- Use socket-based programming like in Java …

Class **DatagramSocket** → UDP Socket

Classes **Socket/ServerSocket** → TCP socket

Read/Write on Sockets through **InputStream/OutputStream**