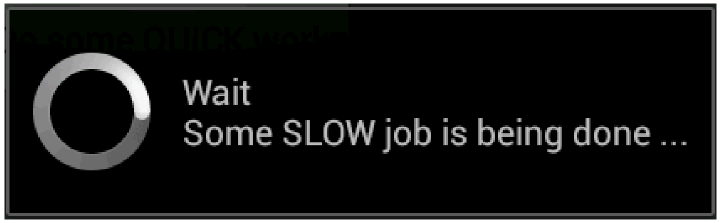
Lesson 9

AsyncTask
Http Connection
Parsing JSON

A. AsyncTask

Concurrency Control

Using the AsyncTask Class



Wait
Some SLOW job is being done ...

1. The **AsyncTask** class allows the execution of background operations and the publishing of results on the UI's thread without having to manipulate threads and/or handlers.

2. An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.

3. An asynchronous task class is defined by the following Types, States, and Method

Generic Types	Main States	Auxiliary Method
Params, Progress, Result	onPreExecute, doInBackground, onProgressUpdate onPostExecute.	publishProgress

Concurrency Control

Using the AsyncTask Class

```
AsyncTask <Params, Progress, Result>
```

AsyncTask's generic types
Params: the type of the input parameters sent to the task at execution.
Progress: the type of the progress units published during the background computation.
Result: the type of the result of the background computation.

To mark a type as unused, use the type **Void**

Note:
The Java notation "**String ...**" called **Varargs** indicates an array of String values. This syntax is somehow equivalent to "**String[]**" (see Appendix B).

Concurrency Control

Using the AsyncTask Class

```
private class VerySlowTask extends AsyncTask<String, Long, Void> {

       // Begin - can use UI thread here
1      protected void onPreExecute() {

       }

       // this is the SLOW background thread taking care of heavy  tasks
       // cannot directly change UI
2      protected Void doInBackground(final String... args) {
       ... publishProgress((Long) someLongValue);
       }

       // periodic updates - it is OK to change UI
       @Override
3      protected void onProgressUpdate(Long... value)  {

       }

       // End - can use UI thread here
4      protected void onPostExecute(final Void unused)  {

       }
}
```

Concurrency Control

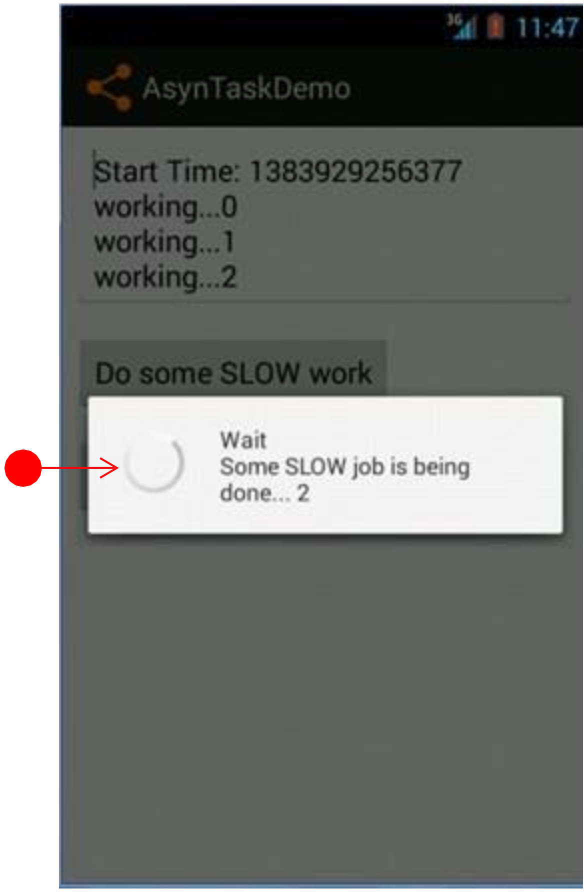
Using the AsyncTask Class

Methods
onPreExecute(), invoked on the UI thread immediately after the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
doInBackground(Params...), invoked on the background thread immediately after *onPreExecute*() finishes executing. This step is used to perform background computation that can take a long time. This step can also use *publishProgress(Progress...)* to publish one or more units of progress. These values are published on the UI thread, in the *onProgressUpdate(Progress...)* step.
onProgressUpdate(Progress...), invoked on the UI thread after a call to *publishProgress(Progress...)*. This method is used to inform of any form of progress in the user interface while the background computation is still executing.
onPostExecute(Result), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

Reference: http://developer.android.com/reference/android/os/AsyncTask.html

Concurrency Control

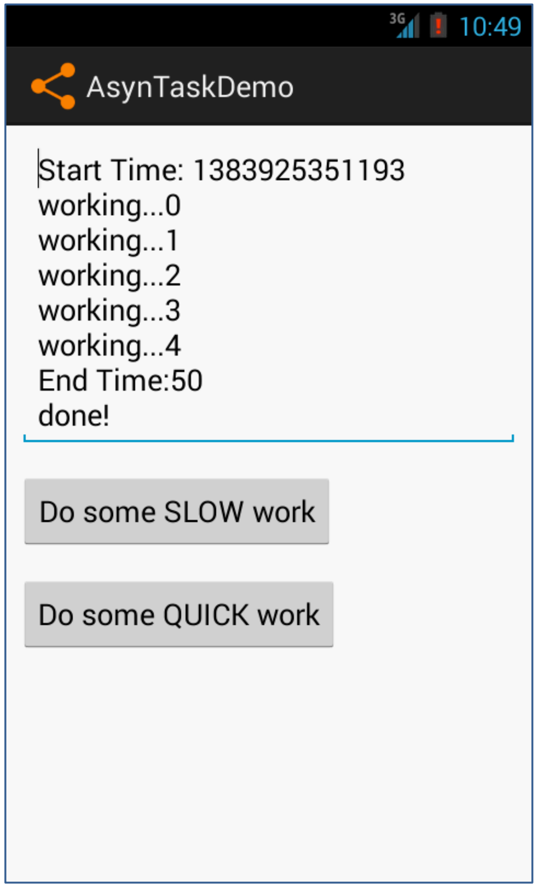
Example: Using the AsyncTask Class



AsynTaskDemo

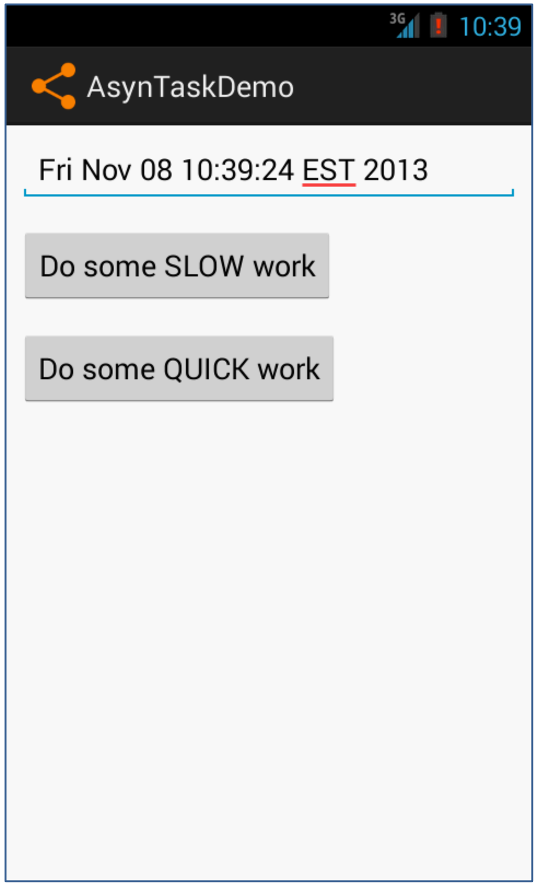
```
Start Time: 1383929256377
working...0
working...1
working...2
```

Wait
Some SLOW job is being
done... 2



AsynTaskDemo

```
Start Time: 1383925351193
working...0
working...1
working...2
working...3
working...4
End Time:50
done!
```



AsynTaskDemo

```
Fri Nov 08 10:39:24 EST 2013
```

The main task invokes an **AsyncTask** to do some slow job. The AsyncTask method **doInBackgroud(…)** performs the required computation and periodically uses the **onProgressUpdate(…)** function to refresh the main's UI. In our the example, the AsyncTask manages the writing of progress lines in the UI's text box, and displays a **ProgressDialog** box.

Concurrency Control

Example : Using the AsyncTask Class - XML Layout

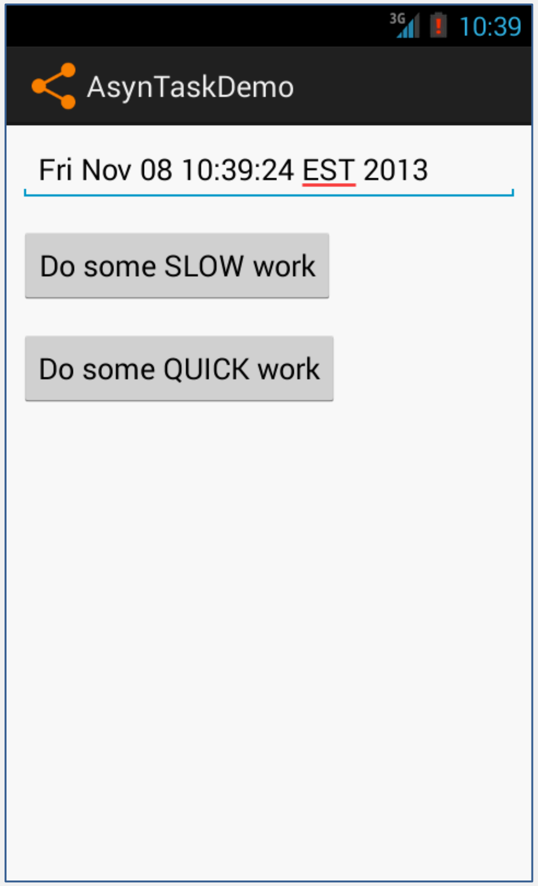
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <EditText
        android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="7dp" />

    <Button
        android:id="@+id/btnSlow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="7dp"
        android:text="Do some SLOW work" />

    <Button
        android:id="@+id/btnQuick"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="7dp"
        android:text="Do some QUICK work" />

</LinearLayout>
```



Concurrency Control

Example: Using the AsyncTask Class - XML Layout

```
public class MainActivity extends Activity {
   Button btnSlowWork;
   Button btnQuickWork;
   EditText txtMsg;
   Long startingMillis;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      txtMsg = (EditText) findViewById(R.id.txtMsg);

      // slow work...for example: delete databases: “dummy1” and “dummy2”
      btnSlowWork = (Button) findViewById(R.id.btnSlow);
      this.btnSlowWork.setOnClickListener(new OnClickListener() {
         public void onClick(final View v) {
1 →         new VerySlowTask().execute("dummy1", "dummy2");
         }
      });

      btnQuickWork = (Button) findViewById(R.id.btnQuick);
      this.btnQuickWork.setOnClickListener(new OnClickListener() {
         public void onClick(final View v) {
            txtMsg.setText((new Date()).toString()); // quickly show today’s date
         }
      });
   }// onCreate
```

Concurrency Control

Example: Using the AsyncTask Class - XML Layout

```
private class VerySlowTask extends AsyncTask<String, Long, Void> {
   private final ProgressDialog dialog = new ProgressDialog(MainActivity.this);
   String waitMsg = "Wait\nSome SLOW job is being done... ";

❷ →  protected void onPreExecute() {
      startingMillis = System.currentTimeMillis();
      txtMsg.setText("Start Time: " + startingMillis);
      this.dialog.setMessage(waitMsg);
      this.dialog.setCancelable(false); //outside touch doesn't dismiss you
      this.dialog.show();
   }

❸ →  protected Void doInBackground(final String... args) {
      // show on Log.e the supplied dummy arguments
      Log.e("doInBackground>>", "Total args: " + args.length );
      Log.e("doInBackground>>", "args[0] = " + args[0] );

      try {
         for (Long i = 0L; i < 5L; i++) {
            Thread.sleep(10000); // simulate the slow job here . . .
            publishProgress((Long) i);
         }
      } catch (InterruptedException e) {
         Log.e("slow-job interrupted", e.getMessage());
      }
      return null;
   }
```

Concurrency Control

Example: Using the AsyncTask Class - XML Layout

```
      // periodic updates - it is OK to change UI
      @Override
4     protected void onProgressUpdate(Long... value) {
         super.onProgressUpdate(value);
         dialog.setMessage(waitMsg + value[0]);
         txtMsg.append("\nworking..." + value[0]);
      }

      // can use UI thread here
5     protected void onPostExecute(final Void unused) {

         if (this.dialog.isShowing()) {
            this.dialog.dismiss();
         }

         // cleaning-up, all done
         txtMsg.append("\nEnd Time:"
               + (System.currentTimeMillis() - startingMillis) / 1000);
         txtMsg.append("\ndone!");
      }

   }// AsyncTask

}// MainActivity
```

Concurrency Control

Example: Using the AsyncTask Class

Comments

1. The **MainActivity** instantiates our AsyncTask passing dummy parameters.
2. VerySlowTask sets a ProgressDialog box to keep the user aware of the slow job. The box is defined as *not cancellable*, so touches on the UI will not dismiss it (as it would do otherwise).
3. **doInBackground** accepts the parameters supplied by the **.execute(…)** method. It fakes slow progress by sleeping various cycles of 10 seconds each. After awaking it asks the onProgressUpdate() method to refresh the ProgressDialog box as well as the user's UI.
4. The **onProgressUpdate**() method receives one argument coming from the busy background method (observe it is defined to accept multiple input arguments). The arriving argument is reported in the UI's textbox and the dialog box.
5. The **OnPostExecute**() method performs house-cleaning, in our case it dismisses the dialog box and adds a "Done" message on the UI.

B. HTTP Connection

Android Networking

Network Prerequisites

- The **`<uses-permission>`** element must be included in the AndroidManifest.xml resource so as to allow the application to connect to the network
- Permissions are used to ask the operating system to access any privileged resource
- The **`<uses-permission>`** tag causes the application to request to use an Android resource that must be authorized
  - The tag must be an immediate child of **`<manifest>`**

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

Android Networking

Protocols

- Android supports several different network protocols.
  - TCP / IP (through the `Socket` class)
  - SMTP (through the `GMailSender` class)
  - HTTP
  - And others
- In this lesson, you will work with HTTP

Android Networking

HTTP

- Hyper Text Transfer Protocol
- In our case the transfer protocol is HTTP
  - We connect the client device to a server and get data
  - We then process that data somehow
    - We might render a Web page
    - We might parse and process XML
    - Or any other message

Android Networking

HTTP

- Two HTTP clients
  - `HttpClient`
  - `HttpURLConnection`
- Both support HTTPS and IPV6
- Use `HttpURLConnection` for post Lollipop devices (version 5.x)

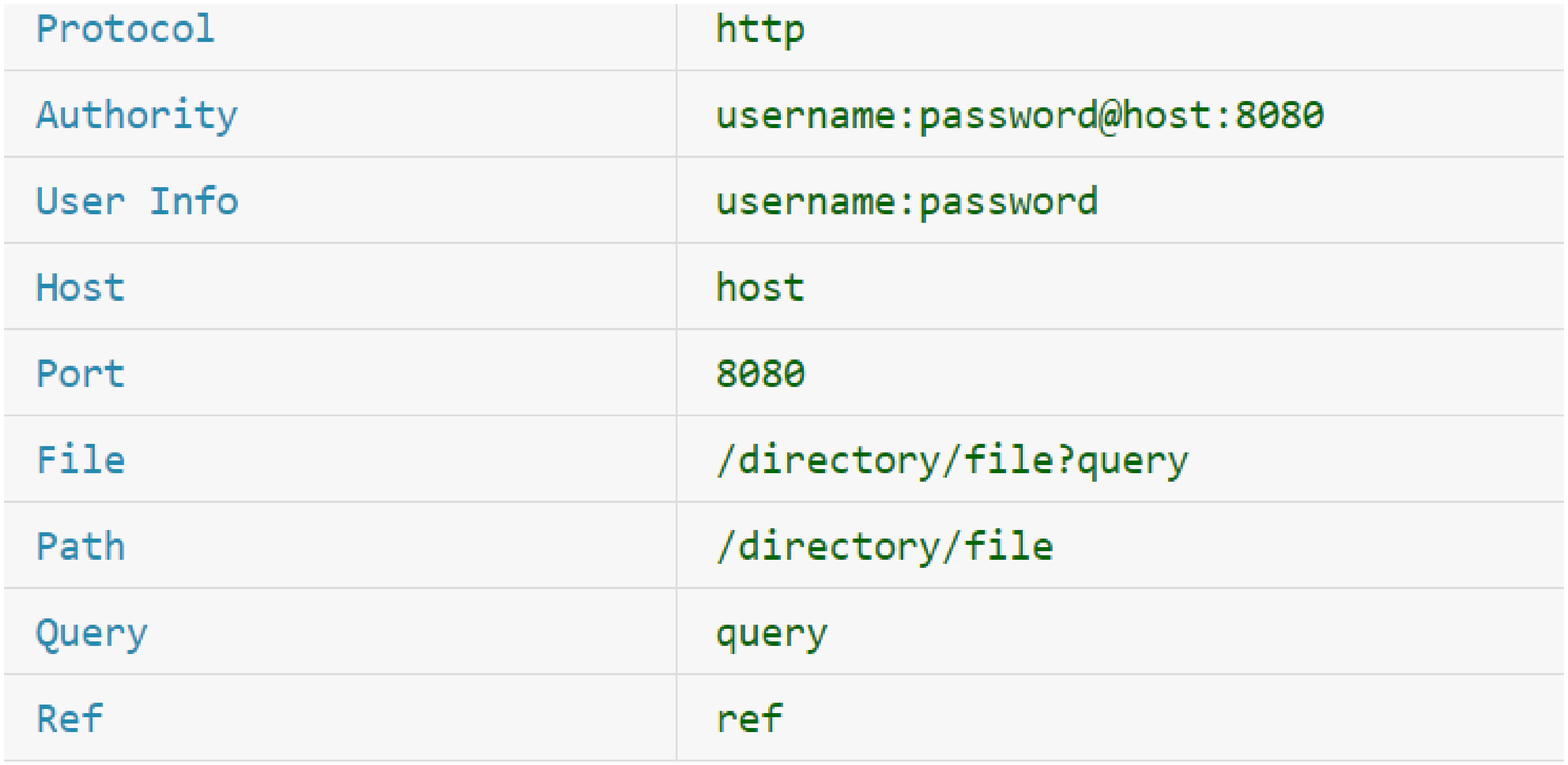
Android Networking

The URL class

- The `java.net.URL` class represents a url
  - Convert strings to URLs
  - Convert URLs to strings
- http://developer.android.com/reference/java/net/URL.html

Android Networking

The URL class



Protocol	http
Authority	username:password@host:8080
User Info	username:password
Host	host
Port	8080
File	/directory/file?query
Path	/directory/file
Query	query
Ref	ref

Android Networking

Opening a connection

- The `URL.openConnection()` method establishes a connection to a resource
- Over this connection, you make the request and get the response
- We will use HTTP here but other protocols are supported

Android Networking

Opening a connection

- The `setReadTimeout()` mutator defines the time to wait for data
- The `setConnectTimeout()` mutator the time to wait before establishing a connection
- The `setRequestMethod()` defines whether the request will be a GET or a POST

Android Networking

Opening a connection

- `getResponseCode()` gets the HTTP response code from the server
  - -1 if there is no response code.
  - Such as 404 not found?
- `getInputStream()` gets the input stream from which you can read data
  - Works just like an open file

Android Networking

Sending GET request

```
URL url = new URL("http://httpbin.org/get");
HttpURLConnection con = (HttpURLConnection) url.openConnection();

// optional default is GET
con.setRequestMethod("GET");

int responseCode = con.getResponseCode();
Log.v("TAG", "Sending 'GET' request to URL : " + url.toString());
Log.v("TAG", "Response Code : " + responseCode);

BufferedReader in = new BufferedReader(new
InputStreamReader(con.getInputStream()));

String inputLine;
StringBuilder response = new StringBuilder();
while ((inputLine = in.readLine()) != null) {
       response.append(inputLine);
}

in.close();
//print result
Log.v("TAG", response.toString());
```

Android Networking

Download file

```
URL url = new
URL("https://data.chiasenhac.com/dataxx/16/downloads/1021/2/1020691-
46ae8508/128/Happy%20New%20Year%20-%20ABBA.mp3");
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");

int responseCode = conn.getResponseCode();
Log.v("TAG", "Response Code = " + responseCode);

InputStream inputStream = conn.getInputStream();
File file = new File(MainActivity.this.getFilesDir(), "test.mp3");
FileOutputStream outputStream = openFileOutput("test.mp3", MODE_PRIVATE);

byte[] buffer = new byte[1024];
int len;

while ((len = inputStream.read(buffer)) != -1)
    outputStream.write(buffer, 0, len);

outputStream.close();
inputStream.close();

Log.v("TAG", "Download complete");
```

Android Networking

Sending POST request

```java
URL url = new URL("http://httpbin.org/post");
HttpURLConnection con = (HttpURLConnection) url.openConnection();

//add reuqest header
con.setRequestMethod("POST");

String params = "user=admin&pass=123456";

// Send post request
con.setDoOutput(true);
DataOutputStream wr = new DataOutputStream(con.getOutputStream());
wr.writeBytes(params);
wr.flush();
wr.close();

int responseCode = con.getResponseCode();
Log.v("TAG", "Sending 'POST' request to URL : " + url.toString());
Log.v("TAG", "Response Code : " + responseCode);

BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
String inputLine;
StringBuilder response = new StringBuilder();
while ((inputLine = in.readLine()) != null) {
           response.append(inputLine);
}
in.close();

//print result
Log.v("TAG", response.toString());
```

Android Networking

3rd party libraries

```
Http Client:
- OKHttp http://square.github.io/okhttp/
- Retrofit http://square.github.io/retrofit/

Image Loader:
- Picasso http://square.github.io/picasso/
- Universal Image Loader
https://github.com/nostra13/Android-Universal-
Image-Loader
```

C. Parsing JSON

Parsing JSON

What is JSON

- **J**ava**S**cript **O**bject **N**otation
- JSON is a syntax for storing and exchanging data
- JSON is text, written with JavaScript object notation
- Completely language independent
- Easy to understand, manipulate and generate

Parsing JSON

JSON syntax

- Data is in name/value pairs

```
"name":"John"
```

- Data is separated by commas

```
"name":"John", "age":20, "gender":"male"
```

- Curly braces hold objects

```
{"name":"John", "age":20, "gender":"male"}
```

- Square braces hold arrays

```
    [{"name":"John", "age":20,
"gender":"male"}, {"name":"Peter", "age":21,
"gender":"male"}, {"name":"July", "age":19,
"gender":"female"}]
```

Parsing JSON

JSON values

- In JSON, values must be one of the following data types:
  - A string

    ```
    { "name":"John" }
    ```
  - A number

    ```
    { "age":30 }
    ```
  - An object

    ```
    {"employee":{ "name":"John", "age":30, "city":"New
    York" }}
    ```
  - An array

    ```
    {"employees":[ "John", "Anna", "Peter" ]}
    ```
  - A Boolean

    ```
    { "sale":true }
    ```
  - Null

    ```
    { "middlename":null }
    ```

Parsing JSON

Parsing JSON string

- Basic types such as *strings*, *numbers*, and *Boolean values*, are represented in Java as their corresponding types (String, int/double, boolean, respectively).
- Compound types are represented using classes in the org.json package.
  - JSON arrays are represented by the class org.json.JSONArray;
  - JSON objects are represented by the class org.json.JSONObject.
- *Null values* are represented by the instance JSONObject.NULL.

Parsing JSON

Parsing JSON string

- To parse compound JSON data from a String, create a new Java object of the appropriate type, passing the String as the only argument to the constructor.

```
// Parsing JSON object
JSONObject jsonData = new JSONObject(jsonString);

// Parsing JSON array
JSONArray jsonData = new JSONArray(jsonString);
```

Parsing JSON

Parsing JSON string – Retrieving data

- Values for the keys in a JSONObject may be obtained using ***get*(String key)*** methods
  - ***getBoolean(key)*** will get a boolean value
  - ***getInt(key)*** will get an int value
  - ***getString(key)*** will get a String value
  - ***getJSONObject(key)*** will get a JSONObject value
  - ***isNull(String key)*** may be used to test if the value of a key is null. It will also return true if key does not exist in the JSONObject
- ***keys()*** will return an iterator Java object (java.util.Iterator) which you can use to iterate through the keys in a JSONObject.
- Values in a JSONArray may be obtained using ***get*(int index)***
- Both JSONObject and JSONArray provide the ***count()*** method to return the number of items in the *object/array*

Parsing JSON

Example: Parsing JSON data from URL

- Sample URL contains JSON data
  https://jsonplaceholder.typicode.com/users

- Steps:
  - Implement a class extended from AsyncTask to get data in background
  - Implement GET request by using URLConnection
  - Parse JSON by using JSONArray and JSONObject classes

Serialize object to JSON

Convert JAVA object to JSON string

- Use GSON library https://github.com/google/gson
- Main functions:
  - toJson() serialize object to Json
  - fromJson() deserialize json to object
- Example:

```
Gson gson = new Gson(); // Or use new GsonBuilder().create();
MyType obj1 = new MyType();
String json = gson.toJson(obj1); // serializes obj1 to Json
MyType obj2 = gson.fromJson(json, MyType.class); // deserializes json into obj2
```

D. Working with Socket class

Working with Socket class

Setup a client socket

```
String serverIP = "192.168.0.167";
int serverPort = 9000;
int timeOut = 5000;

InetAddress serverAddress = InetAddress.getByName(serverIP);
client = new Socket();
client.connect(new InetSocketAddress(serverAddress, serverPort), timeOut);
```

Working with Socket class

Send data to server

```
String data = "Hello server";
OutputStreamWriter writer = new OutputStreamWriter(client.getOutputStream());
writer.write(data);
writer.flush();
```

Working with Socket class

Receive data from server

```
String result = "";
byte buffer[] = new byte[1024];
while (client.isConnected() && !client.isClosed()) {

    int res = client.getInputStream().read(buffer);
    if (res > 0) {
        result = new String(buffer, 0, res);
        publishProgress(result);
        Log.v("TAG", "Received: " + result);
    }
    else
        Log.v("TAG", "Received: ERROR" + result);
}
```