SESSION #7

NETWORKING

OUTLINE

- Multitasking
  - Asynchronous tasks
- Networking
  - HTTP
  - URI
  - JSON
- Services
- Examples
- Sprint planning 3

APPLICATION UI THREAD

✧ **UI thread**: main application thread.

- When an application is launched, the system creates a thread of execution for the application, called "main thread or UI thread."
- The system does *not* create a separate thread for each instance of a component, all components are instantiated in the UI thread.
- It is responsible for:
  - Dispatch events to the appropriate user interface widgets
  - Interacts with components from the Android UI toolkit

✧ Single Thread model drawbacks:

- Long and intensive work, can yield poor performance unless you implement your application properly.
- When the thread is blocked, no events can be dispatched, including drawing events.
- Android UI toolkit **is *not* thread-safe**. You must do all manipulation to your user interface from the UI thread.

BACKGROUND PROCESSING

✧ Because of the single thread model, it's vital to the responsiveness of your application's UI that you do not block the UI thread.

✧ If you have operations to perform that are not instantaneous, you should make sure to do them in separate threads ("background" or "worker" threads)

✧ Multi-threading processing alternatives:

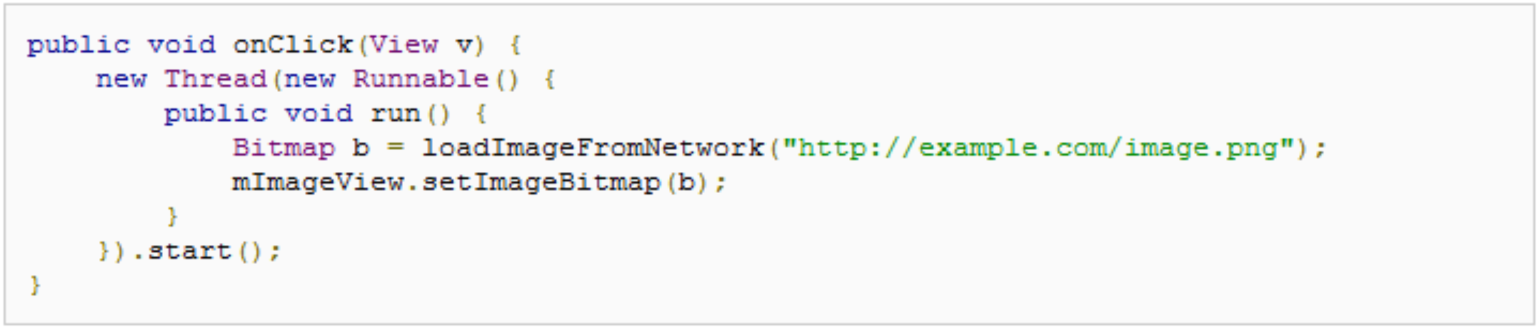
- **Java threads**
- **Handlers**
- **Asynchronous Tasks**

MULTITASKING

JAVA THREADS

✧ Android supports standard Java Threads .

- You can use java Threads to perform operations that are not instantaneous, using separate threads ("background" or "worker" threads) to execute them in background:



```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

- This approach access the Android UI toolkit from outside the UI thread.
  - This can result in undefined and unexpected behavior, which can be difficult to track down.

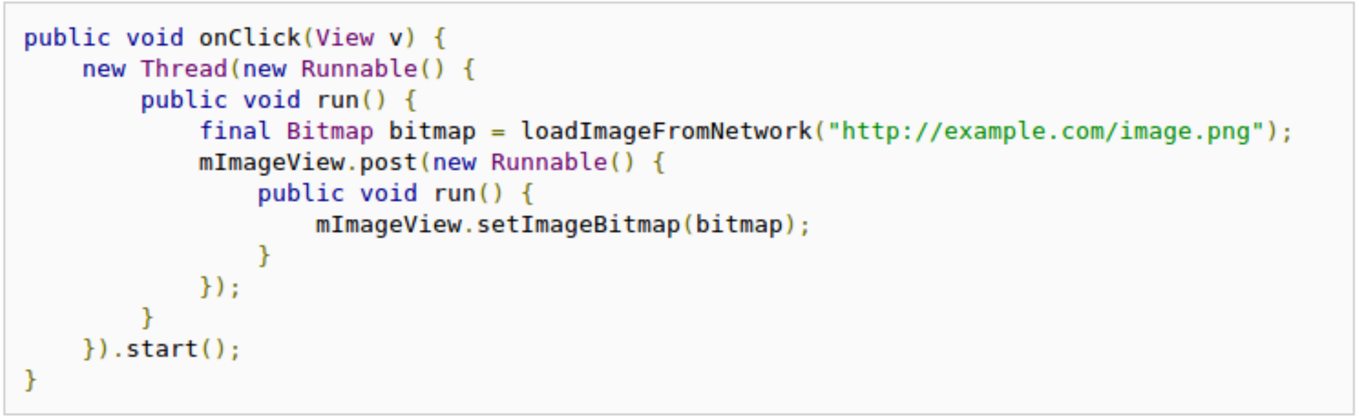
HANDLERS

✧ A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue.

- Each Handler instance is associated with a single thread and its message queue.
- There are two main uses for a Handler:
    1) To schedule messages and runnables to be executed as some point in the future.
    2) To enqueue an action to be performed on a different thread than your own.

✧ For example:

- It can be used the post(Runnable) method of a view object that causes the Runnable to be added to the view message queue. The runnable will be run on the user interface thread.



```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap = loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Now this implementation is thread-safe: the network operation is done from a separate thread while the ImageView is manipulated from the UI thread.

MULTITASKING

ASYNCHRONOUS TASKS

- ✧ AsyncTask allows to perform asynchronous work on your user interface.
  - It performs the blocking operations in a worker thread and then publish the results on the UI thread, without requiring you to handle threads and/or handlers yourself.

- ✧ To use it, you must:
  - Subclass **AsyncTask** and implement the doInBackground() callback method, which runs in a pool of background threads.
  - To update your UI, you should implement onPostExecute(), which delivers the result from doInBackground() and runs in the UI thread, so you can safely update your UI.
  - You can then run the task by calling execute() from the UI thread.

```java
public void onClick(View v) {
    new DownloadImageTask().execute("http://example.com/image.png");
}

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
      * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
      * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

- The type of the parameters, the progress values, and the final value of the task can be set using generics.
- doInBackground() it is executes automatically on a worker thread.
- onPreExecute(), onPostExecute(), and onProgressUpdate() are invoked on the UI thread
- The value returned by doInBackground() is sent to onPostExecute()
- You can call publishProgress() at anytime in doInBackground() to execute onProgressUpdate() on the UI thread

ASYNC TASK

```
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));
             // Escape early if cancel() is called
             if (isCancelled()) break;
         }
         return totalSize;
     }

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

     protected void onPostExecute(Long result) {
         showDialog("Downloaded " + result + " bytes");
     }
 }
```

Once created, a task is executed very simply:

```
new DownloadFilesTask().execute(url1, url2, url3);
```

✧ https://developer.android.com/reference/android/os/AsyncTask.html

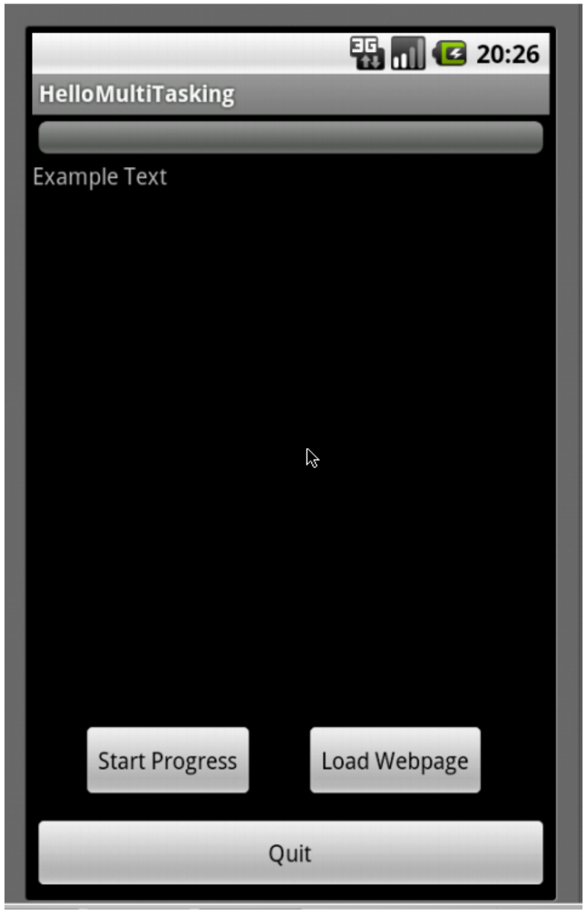
DOWNLOADING DATA

- Stream classes
  - Input/OutputStream
  - BufferedReader/Writer
- DownloadManager
  - The download manager is a system service that handles long-running HTTP downloads. Clients may request that a URI be downloaded to a particular destination file.
- Libraries
  - Images
    - Picasso
    - Glide

HELLO MULTITASKING

✧ We will create a HelloMultitasking application that uses 3 forms of multitasking:

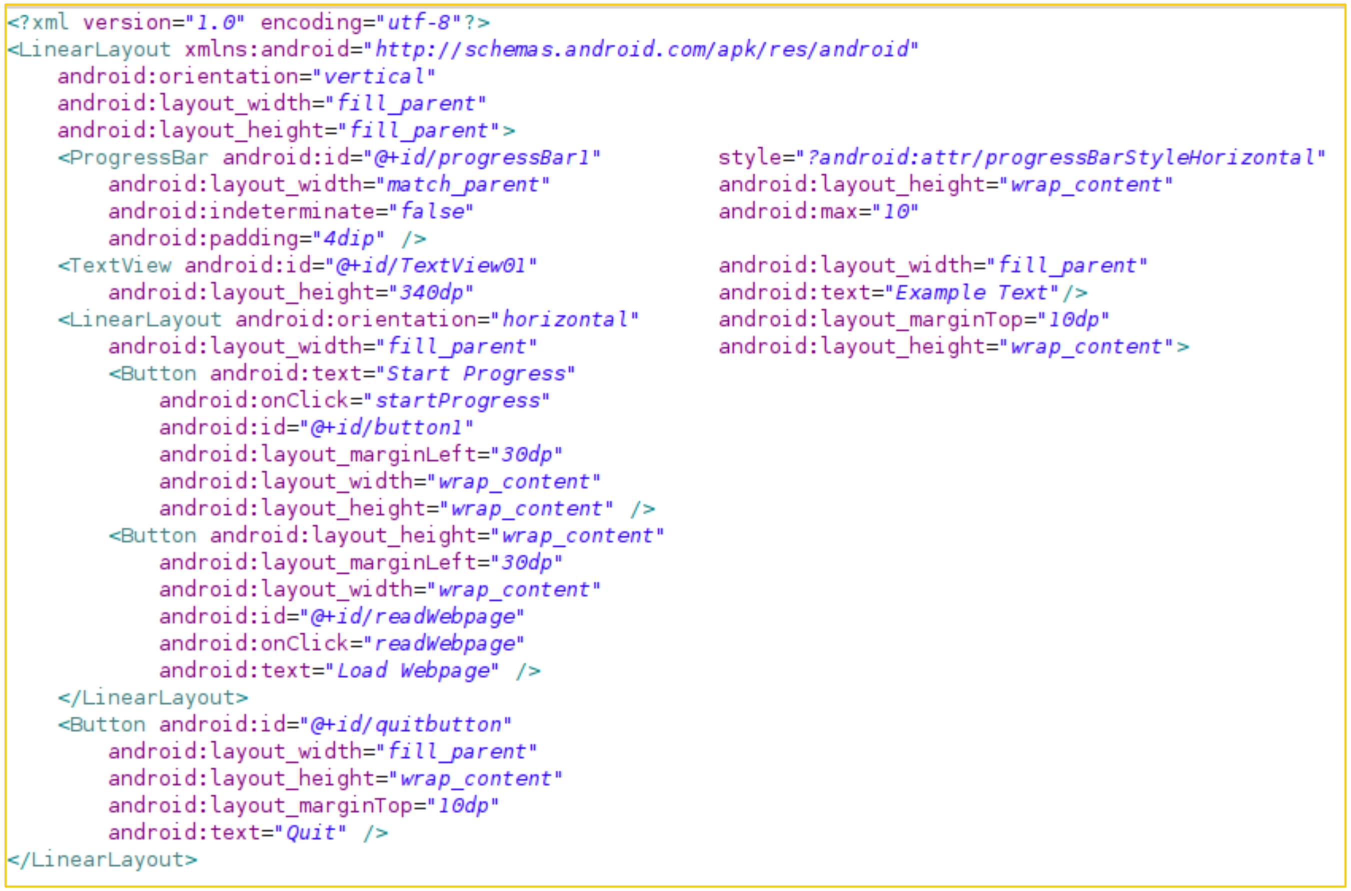
- A java thread to implement the progression in a progressDialog.
- A Handler to implement the progression in the UI progressBar view.
- A Asynchronous task to perform in background a longer operation, as download the code of a web page.



LAYOUT

✧ Start a new project and Activity called "HelloMultitasking“

1. Open the *res/layout/main.xml* file and replace it with the right code:



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ProgressBar android:id="@+id/progressBar1"          style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"              android:layout_height="wrap_content"
        android:indeterminate="false"                    android:max="10"
        android:padding="4dip" />
    <TextView android:id="@+id/TextView01"               android:layout_width="fill_parent"
        android:layout_height="340dp"                    android:text="Example Text"/>
    <LinearLayout android:orientation="horizontal"       android:layout_marginTop="10dp"
        android:layout_width="fill_parent"               android:layout_height="wrap_content">
        <Button android:text="Start Progress"
            android:onClick="startProgress"
            android:id="@+id/button1"
            android:layout_marginLeft="30dp"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <Button android:layout_height="wrap_content"
            android:layout_marginLeft="30dp"
            android:layout_width="wrap_content"
            android:id="@+id/readWebpage"
            android:onClick="readWebpage"
            android:text="Load Webpage" />
    </LinearLayout>
    <Button android:id="@+id/quitbutton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:text="Quit" />
</LinearLayout>
```

JAVA THREADS (I)

✧ Add a java thread to increase a progress dialog when the user clicks on the Quit Button.

1. Insert the onCreateDialog callback to create the progress dialog:

```
protected Dialog onCreateDialog(int id)
{
      switch(id) {
        case DIALOG_EXIT_ID:
            // do the work to define the exit Dialog
          progressDialog = new ProgressDialog(HelloMultiTaskingActivity.this);
          progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
          progressDialog.setMessage("Closing Application...");
          progressDialog.setCancelable(false);
          return progressDialog;
        default:
            return null;
      }
}
```

2. Create the onClick listener for Quit button and show the progress dialog:

```
// OnClick Event Callback for QuitButton
final Button QuitButton = (Button) findViewById(R.id.quitbutton);
QuitButton.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        // Perform action on clicks
        showDialog(DIALOG_EXIT_ID);
```

MULTITASKING

JAVA THREADS (II)

3. Create a java thread to increase the progress dialog each 100 ms:

```
// Start progress dialog using a java thread (Not Correct).
new Thread(new Runnable() {
    public void run() {
        int x=0;
        while (x<100)
        {
            progressDialog.incrementProgressBy(1);
            x++;
            try {
                Thread.sleep(100);
            }
            catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        finish();
    }
}).start();
```

Increment progress Bar percentage

Sleep thread 100ms

Finish activity

4. Run the application

```
public class HelloMultiTaskingActivity extends Activity {

  static final int DIALOG_EXIT_ID = 0;

  private Handler handler;
  private ProgressBar progress;
  private ProgressDialog progressDialog;
  private TextView textView;

    /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    progress = (ProgressBar) findViewById(R.id.progressBar1);
    handler = new Handler();
    textView = (TextView) findViewById(R.id.TextView01);

    // OnClick Event Callback for QuitButton
        final Button QuitButton = (Button) findViewById(R.id.quitbutton);
        QuitButton.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                // Perform action on clicks
              showDialog(DIALOG_EXIT_ID);

              // Start progress dialog using a java thread (Not Correct).
              new Thread(new Runnable() {
                    public void run() {
                      int x=0;
                      while (x<100)
                      {
                        progressDialog.incrementProgressBy(1);
                        x++;
                        try {
                          Thread.sleep(100);
                        } catch (InterruptedException e) {
                  e.printStackTrace();
              }
                      }
                      finish();
                    }
                }).start();
            }
        });
  }
```

USING HANDLERS

✧ Add a Handler to increase the progressBar view when the user clicks on the Button1.

1. Insert the startProgress method as onClick listener for button1, creating the handler thread:

```
// Handler for track the progression.
public void startProgress(View view) {
  // Do something long
  Runnable runnable = new Runnable() {
    @Override
    public void run() {
      for (int i = 0; i <= 10; i++) {
        final int value = i;
        try {
          Thread.sleep(200);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
        handler.post(new Runnable() {
          @Override
          public void run() {the
            progress.setProgress(value);
          }
        });
      }
    }
  };
  new Thread(runnable).start();
}
```

Schedule/Post a runnable in the message queue of UI Thread to update the UI.

2. Run the application.

MULTITASKING

USING ASYNCHRONOUS TASKS (I)

✧ Add an asynchronous task to download a web page in a textView.

1. Create DownloadWebPageTask class derived from AsyncTask.
   a) Inside the class implement the doInBackground callback.

```
private class DownloadWebPageTask extends AsyncTask<String, Void, String> {

  @Override
  protected String doInBackground(String... urls) {
    String response = "";
    for (String url : urls) {
      DefaultHttpClient client = new DefaultHttpClient();
      HttpGet httpGet = new HttpGet(url);
      try {
        HttpResponse execute = client.execute(httpGet);
        InputStream content = execute.getEntity().getContent();

        BufferedReader buffer = new BufferedReader(
            new InputStreamReader(content));
        String s = "";
        while ((s = buffer.readLine()) != null) {
          response += s;
        }

      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    return response;
  }
```

This method receives a url and download its html code and returns it in a string.

Connect to the url and read the web page content

Returns the web content string

   b) Inside the class implement the doInBackground callback.

```
@Override
protected void onPostExecute(String result) {
  textView.setText(result);
}
```

Assign the web content string to the textView

USING ASYNCHRONOUS TASKS (II)

3. Insert the readWebpage method as onClick listener for readWebPage button, creating and executing the AsyncTask:

```
public void readWebpage(View view) {
   DownloadWebPageTask task = new DownloadWebPageTask();
   task.execute(new String[] {"http://www.eps.udl.es"});
}
```

Create the Async Task

Execute the Async Task with the eps web as url

4. Activate the persmission to acess Internet to the application, adding the following line in the manifest.xml file:

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

5. Run the application

HTTP CLIENTS

- Most network-connected Android apps will use HTTP to send and receive data. Android includes two HTTP clients: HttpURLConnection and Apache HTTP Client. Both support HTTPS, streaming uploads and downloads, configurable timeouts, IPv6 and connection pooling.
- Apache HTTP Client
  - DefaultHttpClient and its sibling AndroidHttpClient are extensible HTTP clients suitable for web browsers. They have large and flexible APIs. Their implementation is stable and they have few bugs.
  - The large size of this API makes it difficult to improve it without breaking compatibility. The Android team is not actively working on Apache HTTP Client.
- HttpURLConnection
  - HttpURLConnection is a general-purpose, lightweight HTTP client suitable for most applications. This class has humble beginnings, but its focused API has made it easy to improve steadily.

- http://android-developers.blogspot.com.es/2011/09/androids-http-clients.html

HTTP CLIENTS

- Which client is best?
  - Apache HTTP client has fewer bugs on Eclair and Froyo. It is the best choice for these releases.
  - For Gingerbread and better, HttpURLConnection is the best choice. Its simple API and small size makes it great fit for Android. It is lighter and used for general purposes.
  - Transparent compression and response caching reduce network use, improve speed and save battery. New applications should use HttpURLConnection.

URI

- In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations.
  - http://www.movieservice.com/movie/:id

    (URL is a good example of URI)
  - http://www.movieservice.com/movie/12345

    (will become your REST endpoint)
- An URI is a string of characters used to identify a resource over the Web. In simple words, the URI in a RESTful web service is a **hyperlink** to a resource, and it is the only means for clients and servers to exchange representations.
- In a RESTful system, the URI is not meant to change over time as it may break the contract between a client and a server. The URIs for resources are expected to remain the same as long as the web service is up and running.
- Uniform interface. Resources are manipulated using a fixed set of four create, read, update, delete operations (CRUD).
- We use URIs to connect clients and servers in order to exchange resources in the form of representations.

JSON

- ✧ JSON is a lightweight, text-based, platform neutral, data interchange format in which objects are represented in the attribute-value pair format.
- ✧ JSON sintax:
  - An unordered collection of name-value pairs (representing an object).

```
{"departmentId":10, "departmentName":"IT",
"manager":"John Chen"}
```

  - An ordered collection of values (representing an array)

```
{"departmentName":"IT",
"employees":[
{"firstName":"John", "lastName":"Chen"},
{"firstName":"Ameya", "lastName":"Job"},
{"firstName":"Pat", "lastName":"Fay"}
],
"location":["New York", "New Delhi"]
}
```

JSON

✧ Basic data types:

- Number: This type is used for storing a signed decimal number that may optionally contain a fractional part. {"totalWeight": 123.456}
- String: This type represents a sequence of zero or more characters. {"firstName": "Jobinesh"}
- Boolean: This type represents either a true or a false value. {"isValidEntry": true}
- Array: This type represents an ordered list of zero or more values, each of which can be of any type. {"fruits": ["apple", "banana", "orange"]}
- Object: This type is an unordered collection of comma-separated attributevalue pairs enclosed in curly braces.

  {"departmentId":10,

  "departmentName":"IT",

  "manager":"John Chen"}
- Null: This type indicates an empty value. {"error":null}

ORG.JSON

CLASS OR INTERFACE	DESCRIPTION
JSONArray	A dense indexed sequence of values. Values may be any mix of JSONObjects, other JSONArrays, Strings, Booleans, Integers, Longs, Doubles, null or NULL. Values may not be NaNs, infinities, or of any type not listed here.
JSONObject	A modifiable set of name/value mappings. Names are unique, non-null strings. Values may be any mix of JSONObjects, JSONArrays, Strings, Booleans, Integers, Longs, Doubles or NULL. Values may not be null, NaNs, infinities, or of any type not listed here.
JSONStringer	Implements toString() and toString(). Most application developers should use those methods directly and disregard this API.
JSONTokener	Parses a JSON (RFC 4627) encoded string into the corresponding object. Most clients of this class will use only need the constructor and nextValue() method.
JSONException	Thrown to indicate a problem with the JSON API.

✧ https://developer.android.com/reference/org/json/package-summary.html

SERVICES

- ✧ A Service is an application component that can perform long-running operations in the background and does not provide a user interface.
  - Services can run in their own process.
- ✧ These are the three different types of services::
  - **Started Service.**
    - A service is "started" when an application component starts it by calling startService().
    - Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
    - Usually, a started service performs a single operation and does not return a result to the caller.
  - **Bound Service.**
    - A service is "bound" when an application component binds to it by calling bindService().
    - A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with inter-process communication (IPC).
    - A bound service runs only as long as another application component is bound to it.
    - Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.
  - **Scheduled Service.**
    - A service is scheduled when an API such as the JobScheduler, introduced in Android 5.0 (API level 21), launches the service.
    - You can use the JobScheduler by registering jobs and specifying their requirements for network and timing.

EXAMPLES

- ✧ HelloMultitasking
  https://github.com/jordigervas/HelloMultiTasking
- ✧ Managing Network Usage
  https://developer.android.com/training/basics/network-ops/index.html
- ✧ Parsing XML Data
  https://developer.android.com/training/basics/network-ops/xml.html
- ✧ REST Example (An extract from https://github.com/udacity/Sunshine-Version-2)
  https://github.com/jordigervas/RESTExample