10-Asynchronous Task

CENG427-Programming Mobile Devices

Asyancronous Tasks

- AsyncTasks should ideally be used for short operations (a few seconds at the most.)
- AsyncTasks enables proper and easy use of UI thread.

Asyancronous Tasks

- An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.
- An asynchronous task is defined by 3 generic types, called
  - Params
  - Progress
  - Result
- And 4 steps:
  - onPreExecute
  - doInBackground
  - onProgressUpdate
  - onPostExecute

AsyncTask Generic Types

- Params:
  - The type of the parameters sent to the task upon execution
- Progress:
  - The type of the progress units published during the bacground computation.
- Result:
  - The type of the result of the background computation.
- Example:
  - Private class DownloadFiles extend s AsyncTask<URL,Integer,Long>{….}

Asynchronous Methods

- AsyncTask must be subclassed to be used. You must implement at least one method (doInBackground(Params ….))

- 1) onPreExecute():
  - This method invoked on the UI thread before the task is executed.
  - This step is normally used to setup the task, for instance by showing a progress bar in the user interface.

Asynchronous Methods

- 2) **doInBackground(Params …):**
  - This method invoked on the background thread immediately after onPreExecute() finishes its execution.
  - This method is used to perform background computation that can take a long time
  - The result of the computation must be returned by this step and will be passed back to last step (**onPostExecute(Result)).**
  - From this method you can also call **publishProgress(Progress …)** method to publish current status at UI thread.
  - The progress publish on the UI thread from **onProgressUpdate(Progress..)** method.

Asynchronous Methods

- 3) **onPostProgressUpdate(Progress …)**
  - This method invoked on the main UI thread when there is a call to **publishProgess(Progress…)** from teh **doInBackground(Params…)** method.
  - This method is used to display any form of progress in the user interface while the background computation is still executing in **doInBackground(Params …).**

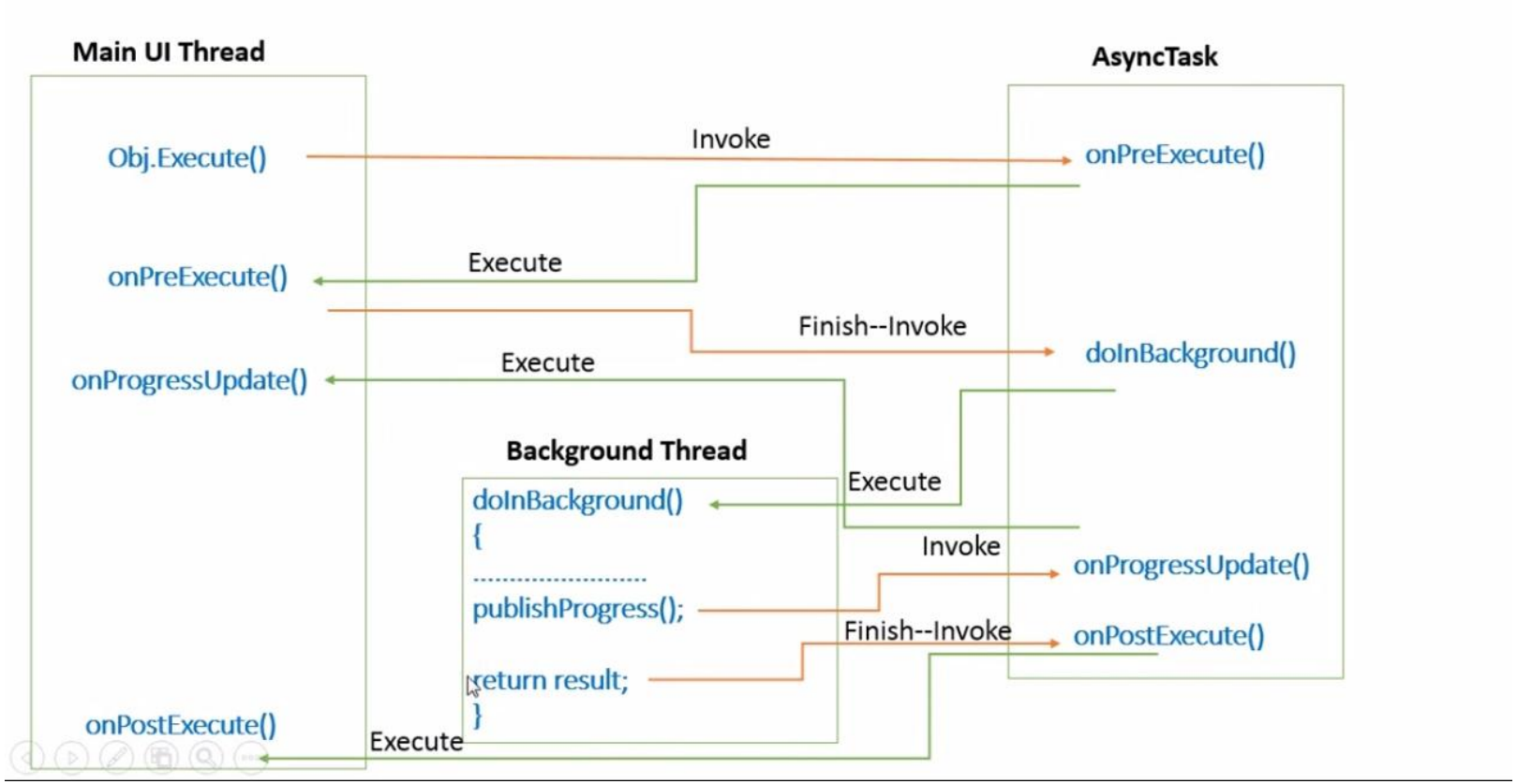
Asynchronous Methods

- 3) **onPostExecute(Results ….)**
  - This method invoked on the main UI thread after background task finişh its execution.
  - The result of the background computation is passed to this step as a parameter from **doInBackground(Params …)** method.

AsyncTask Rules

- The AsynchTask class must be loaded on UI thread.
- The task instance must be created on the UI thread.
- Execute(params…) must be in the UI thread.
- Do not call any of AsyncTask methods manually.
- The task cn be executed only once.
  - An exception will be thrown if a second execution is attempted.

AsyncTask Execution Flow



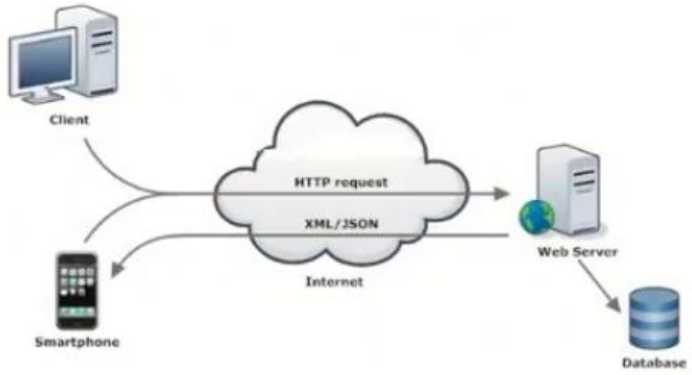
RESTFULL APIs

JSON, Retrofit, ION

Web Services

- **web service**: a set of functionality offered over a server using the web, but not web pages / HTML
  - Use the web's HTTP protocol to connect and transfer data.
  - Client connects to specific URLs to request specific data, which is then sent back in some documented format such as XML or JSON.
  - **REST**: Representational State Transfer. Common style of web services.
    - "RESTful web services" or "RESTful APIs"

- Web services are a bit like remote function calls where you can request data via URLs with parameters and get the data returned as a response.



How to find them & use web APIs

- **Locate** them online
  - Google for phrases like "<company> REST API" or "<service> free API"
- **Sign up** for an account
  - Many web APIs require a login or **API key**
  - Register to receive key or account
- Read the online **documentation** to find out how the API works
  - APIs are not standardized; each one is completely unique
  - Need documentation to learn the available services, parameters, etc.

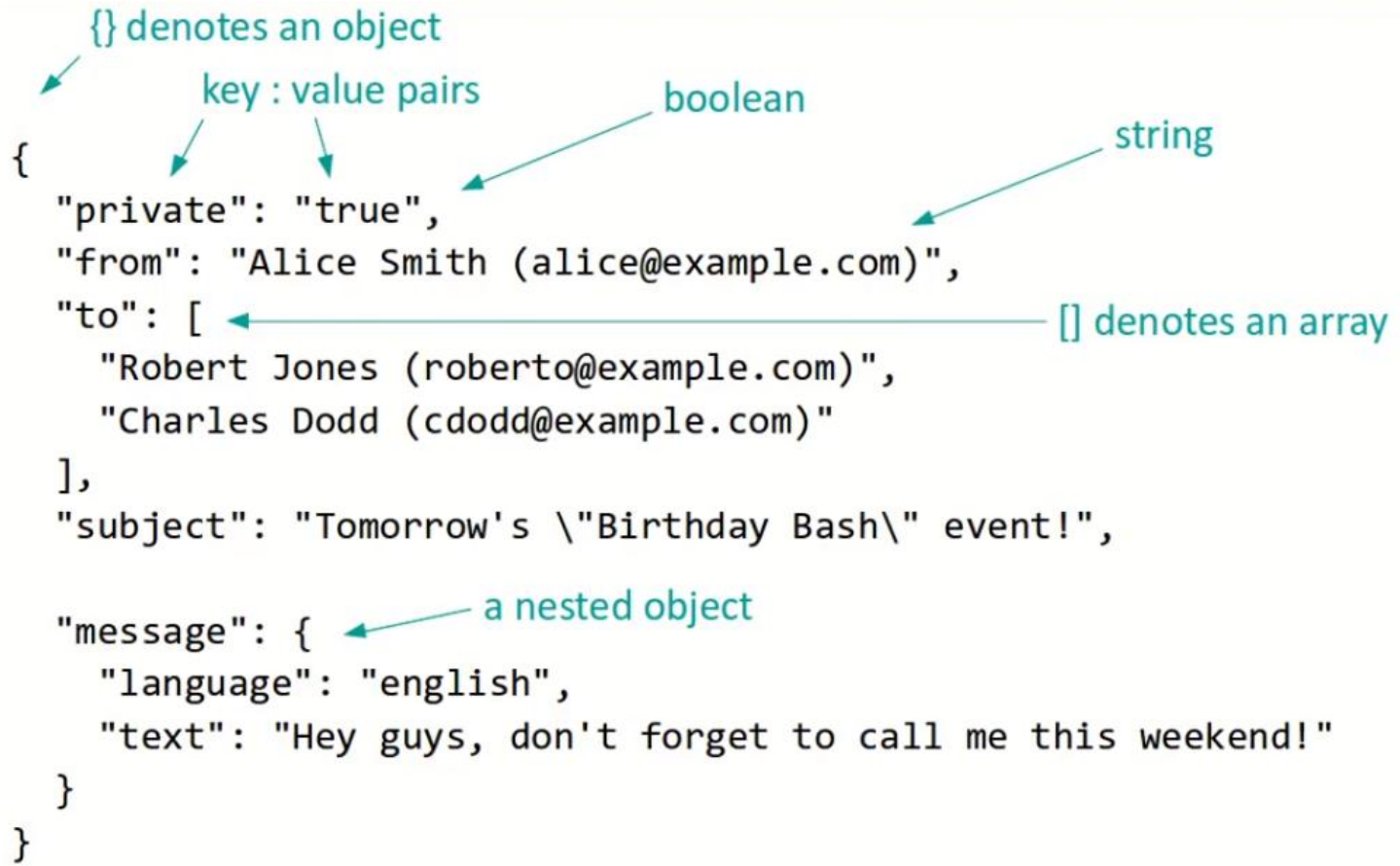
Data formats: JSON, XML

- Most web APIs return their data in one of these formats:
  - **JSON**: JavaScript Object Notation
    - Data is represented as a JavaScript object literal.
      - JavaScript objects are basically maps from keys to values.
    - All values in the data are the fields of the object.
    - Object can contain sub-objects (maps), lists, strings, numbers, etc. inside it.
    - Slightly less capable than XML, but much simpler to read, write, and parse in app code.
    - Currently the most popular web data interchange format for most apps.

  - **XML**: Extensible Markup Language
    - Data is represented as a nested tree of tags and attributes.
    - Better structure than JSON, but bulkier syntax and harder to parse in app code.
    - Very popular 5-10 years ago but being superseded by JSON in many cases.

  - Some web APIs use other data formats:
    - YAML: Yet Another Markup Language. Popular in Ruby/Rails community.
    - plain text

JSON Data Example




```
{
  "private": "true",
  "from": "Alice Smith (alice@example.com)",
  "to": [
    "Robert Jones (roberto@example.com)",
    "Charles Dodd (cdodd@example.com)"
  ],
  "subject": "Tomorrow's \"Birthday Bash\" event!",

  "message": {
    "language": "english",
    "text": "Hey guys, don't forget to call me this weekend!"
  }
}
```

Parsing JSON

```
{
  "private": "true",

  "from": "Alice Smith (alice@ex.com)",

  "to": [
    "Robert Jones (roberto@ex.com)",
    "Charles Dodd (cdodd@ex.com)"
  ],

  "subject": "Today's \"Bash\" event!",


  "message": {
    "lang": "english",
    "text": "Call me this weekend!"
  }
}
```

```
private void processData(String data) {
try {
  // extract the information from JSON data
  JSONObject json = new JSONObject(data);
  boolean private = json.getBoolean("private");

  String from = json.getString("from");

  JSONArray a = json.getJSONArray("to");
  String to1 = a.getString(0);
  String to2 = a.getString(1);


  String subject = json.getString("subject");

  JSONObject msg =
            json.getJSONObject("message");
  String language = msg.getString("lang");
  String text = msg.getString("text");

} catch (JSONException e) {
    Log.wtf("json", e);
}
}
```

JSON Methods

Method	Description

`j.get("key")`
`j.getBoolean("key")`
`j.getDouble("key")`
`j.getInt("key")`
`j.getJSONArray("key")`
`j.getJSONObject("key")`
`j.getLong("key")`
`j.getString("key")`
	retrieve value of the given key, or throw a `JSONException` if key is not found

`j.has("key")`	return true if given key maps to a value
`j.isNull("key")`	return true if given key maps to null
`j.keys()`	return iterator of all keys in object

`j.opt("key", fallback)`
`j.optBoolean("key", fallback)`
`...`
	retrieve value of the given key, or return *fallback* if key is not found

`j.put("key", value)`	sort/filter results by their key
`j.remove("key")`	removes key/value mapping if it exists
`j.toString()` `j.toString(spaces)`	convert JSON object to a string, with optional indentation and spacing
`JSONObject.quote(string)`	encodes data as a JSON string

Fetching Data, No Library Required

```
// how to fetch REST API data in a background thread without any libraries (ick!)
public void fetchData(String urlString) {
    Thread thread = new Thread(new Runnable() {
        public void run() {
            try {
                URL url = new URL(urlString);                                   // connect to the site
                HttpURLConnection conn = (HttpURLConnection) url.openConnection();
                conn.setConnectTimeout(30000);                                  // milliseconds
                conn.setReadTimeout(10000);
                conn.setRequestMethod("GET");
                conn.connect();
                int responseCode = conn.getResponseCode();                      // HTTP result codes; 200=success
                if (responseCode == HttpURLConnection.HTTP_OK) {
                    InputStream input = conn.getInputStream();                  // read data from URL to string
                    StringBuilder sb = new StringBuilder();
                    while (true) {
                        int ch = input.read();
                        if (ch == -1) break;
                        sb.append((char) ch);
                    }
                    String text = sb.toString();
                    processData(text);                                          // you write this!
                } else {
                    Log.d("url", "HTTP fail, code " + responseCode);            // request failed
                }
            } catch (IOException ioe) {
                Log.wtf("url", ioe);
            }
        }
    });
    thread.start();
}
```

Fetch Data with ION Library

```
// how to fetch REST API data in a background thread
// using Ion web library (yay!)
public void fetchData(String urlString) {
    Ion.with(context)
        .load(urlString)
        .asJsonObject()
        .setCallback(new FutureCallback<JsonObject>() {
            public void onCompleted(Exception e,
                                    JsonObject json) {
                // process the data or error
                processData(json);   // you write this!
            }
        });
}
```

Examples in class

- More with Class examples
  - http://api.icndb.com/jokes/random
  - http://thecatapi.com/?id=5dc

References

- http://developer.android.com/reference/android/os