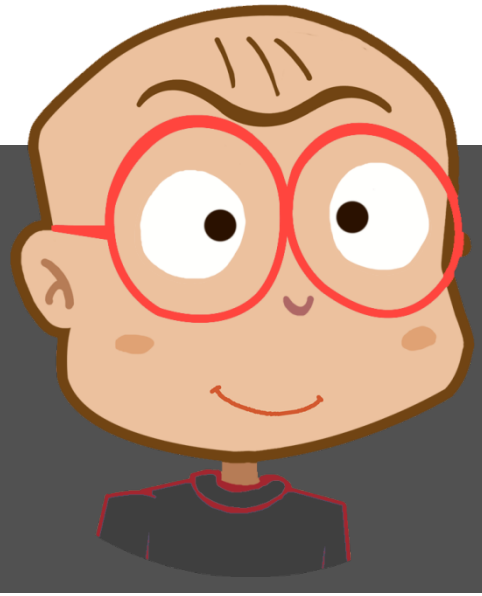


Multi-Threading

START

Learning Goals

After the course, attendees will be able to:

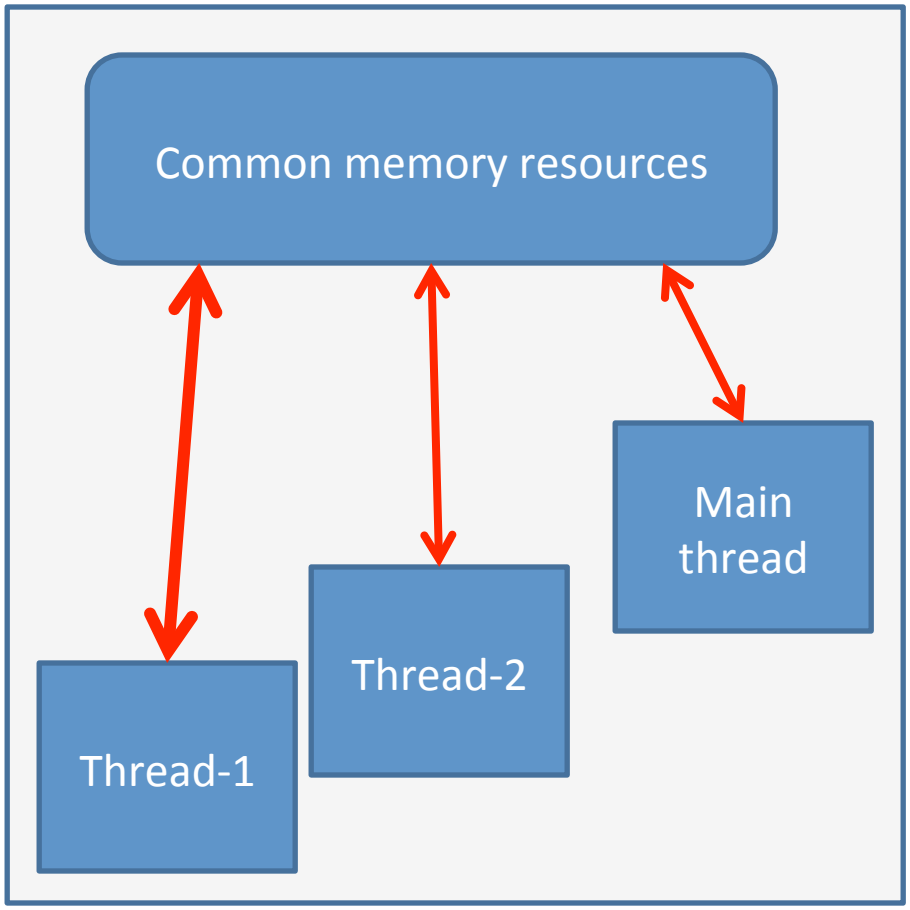
- Understanding about Thread and its mechanism in Android
- Understanding how to implement Thread in Android

Agenda

1. What is Thread?
2. When we need to use Multi-Threading?
3. How to work with Multi-Threading?
4. Practices

1. What is Thread?

- A Thread is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables.
- Each application has at least one thread running when it is started, the main thread, in the main ThreadGroup.
- The application can start the new Thread to handle particular purpose.



2. When we need to use Multi-Threading?

- Only the main thread should update the UI. All other threads in the application should return data back to the main thread to update the UI.

- When you want to parallelize tasks running longer than a few seconds, you should use thread. Android provides other, easier ways to parallelize, like AsyncTask and IntentService.

- Assuming we are doing network operation on a button click in our application. On button click a request would be made to the server and response will be awaited. Due to single thread model of android, till the time response is awaited our screen is non-responsive. So we should avoid performing long running operations on the UI thread. This includes file and network access.

- To overcome this we can create new thread and implement run method to perform this network call, so UI remains responsive.

3. How to work with Multi-Threading?

There are 2 ways to implement of Multi-Threading :

- **Handler**: used to register to a thread and provides a simple channel to send data to this thread.

- **AsyncTask:** encapsulates the creation of a background process and the synchronization with the main thread. It also supports reporting progress of the running tasks.

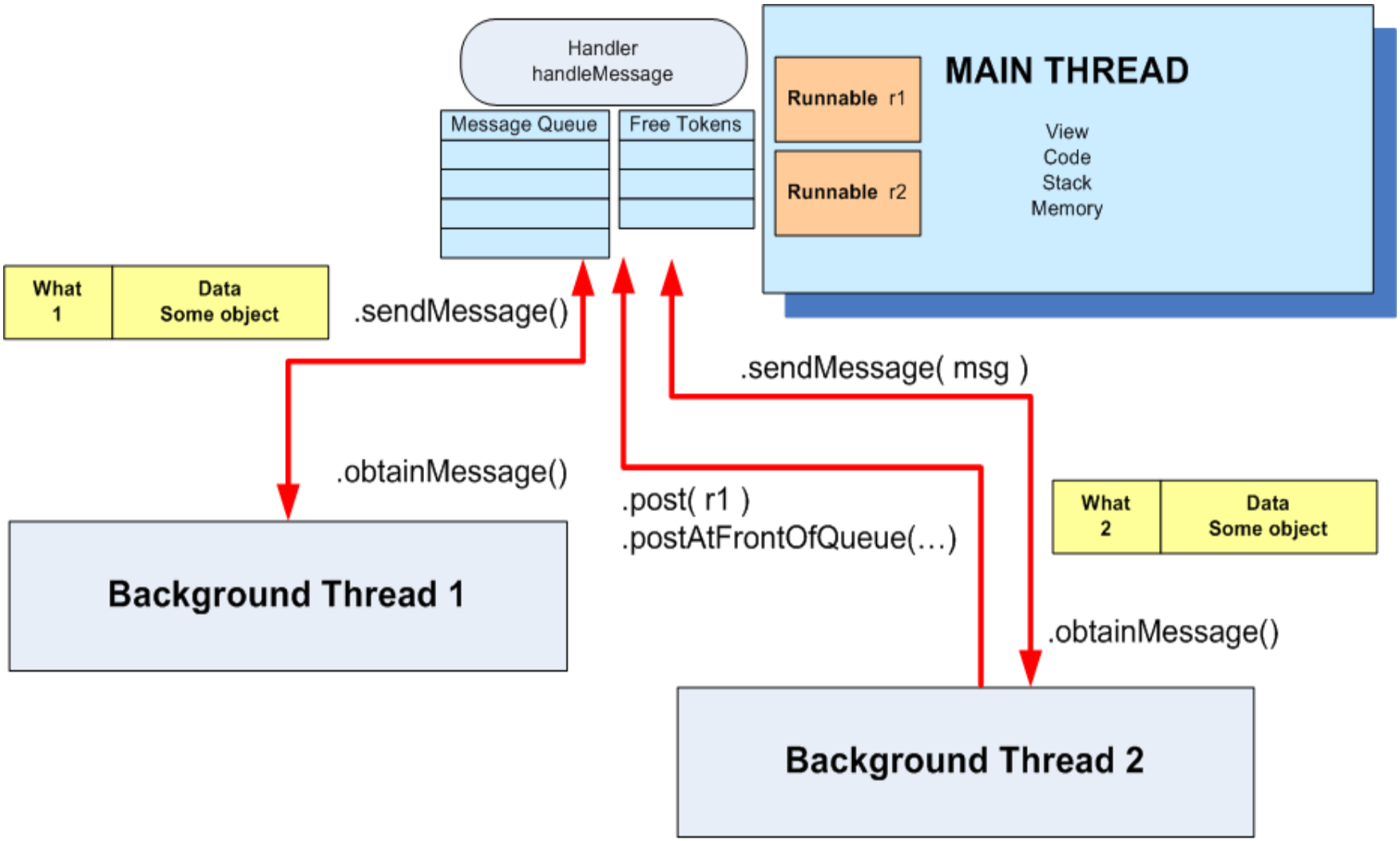
Handler

- We can create other thread, the interact with the UI thread of application through Handler
- When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

- There are two main uses for a Handler
  - Schedule messages and runnables to be executed as some point in the future
  - Enqueue an action to be performed on a different thread than your own.

Handler's MessageQueue

- If each thread (secondary thread) need interact the main thread, it must request a message token by using methods obtainMessage ().
- Once obtained message token, the secondary thread can write data to the message token and add it to the message queue of Handler by using the method SendMessage ().
- Handler uses handlerMessage() method to continuously handle the new message which sent to the UI thread.
- Each message got from the thread's queue can return data to the main thread or objects required to run the runnable via method post ().


Handler
handleMessage
Message Queue
Free Tokens
MAIN THREAD
Runnable r1
Runnable r2
View
Code
Stack
Memory
What
1
Data
Some object
.sendMessage()
.obtainMessage()
Background Thread 1
.post(r1)
.postAtFrontOfQueue(…)
.sendMessage(msg)
What
2
Data
Some object
.obtainMessage()
Background Thread 2

AsyncTask

- AsyncTask allows use UI thread easily and properly.

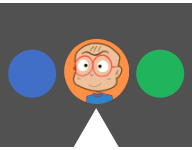
- AsyncTask allows implementation of background activity and send the results to the UI thread without having to manipulate the thread and / or handler.

- A task asynchronously (asynchronous task) is a computational task in a background thread running and the results will be sent to the UI thread.
- There are four main state of asynchronous task
  - **onPreExecute**
  - **doInBackground**
  - **onProgressUpdate**
  - **onPostExecute**.

AsyncTask's params

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

- **Params**: The data type of parameter will be sent to the task when it is executed.
- **Progress**: The type of process announced in the calculation process in background (How long shall notify the results of progress once)
- **Result**: The type of calculation results of background.



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

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

    }

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

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

    }

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

    }
}
```

AsyncTask's methods

- **onPreExecute ()**, called in the UI thread before the task is executed. This step is often used to setup tasks, such as to show a progress bar in the user interface..
- **doInBackground (Params ...)**, called for the background thread after onPreExecute () completes. This step performs calculations background that take time. The parameters of asynchronous task passed to this step. Calculation results must be returned by this step and will be passed on to the final step. This step can also be used publishProgress (Progress ...) to report one or several units progress. These values are published in the UI thread, in step onProgressUpdate (Progress ...).
- **onProgressUpdate (Progress ...),** called for the UI thread after each call to publishProgress (Progress ...). The time of execution is not determined. This method used to display an arbitrary form of progress in the user interface. For example, it could be a progress bar or out logs (log) in a text field.
- **onPostExecute (Result),** called for the UI thread after the calculation has been done in the background. Calculation results of the background for this step is passed through parameters (Result).

Exit Course

THANK YOU