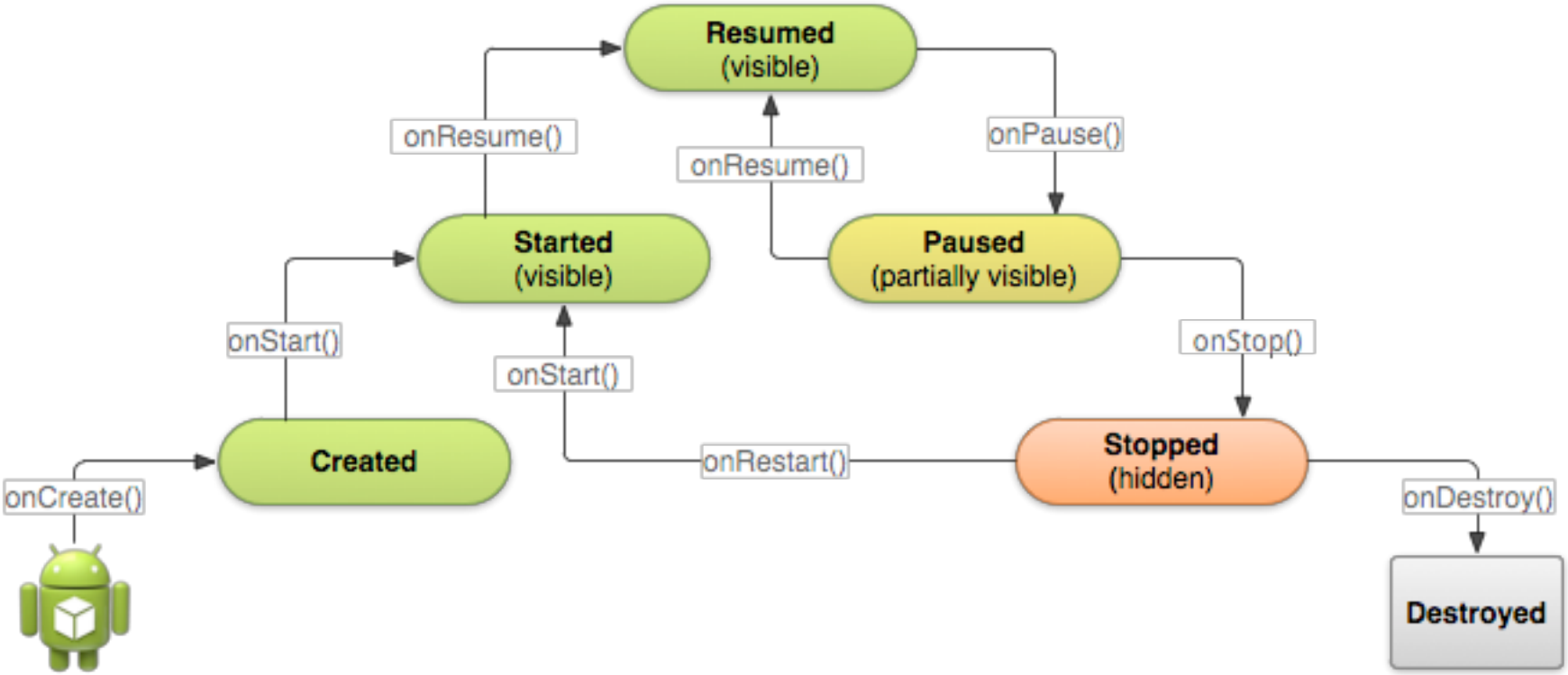
G53MDP
Mobile Device Programming

Time, Threads and Services


Resumed
(visible)
onResume()
onResume()
onPause()
Started
(visible)
Paused
(partially visible)
onStart()
onStart()
onStop()
Created
onRestart()
Stopped
(hidden)
onCreate()
onDestroy()
Destroyed

Saving State

- Shouldn't rely on an Activity storing UI state
  - E.g. rotating the device
    - Destroys and recreates the activity
  - Aggressive OS management
    - Application context handles various onTrimMemory calls
- Before onStop() is called, Android will call onSaveInstanceState()
  - To restore the **UI** to its previous state on restore
    - Cascading call into UI components
      - Everything needs an ID (why?)
  - This allows you to save any **UI state** into a Bundle object
    - When the Activity is recreated, the Bundle is passed to onCreate() and onRestoreInstanceState()
      - Where does this live?
    - Giving the Activity chance to restore its state
  - Save **other state** to more persistent storage
    - SQLite database / user preferences
    - More on this later

Saving UI State

- Bundle
  - A collection of key/value pairs
  - Key
    - Unique String identifier
  - Value
    - A primitive value
    - A Serializable / Parcelable object
      - Writing and reading a complex class
      - More on this later on (IPC)
      - Limitations
  - i.e. myBundle.putInt(“myInteger”, 5);
  - … int i = myBundle.getInt(“myInteger”);

Let’s have a look…

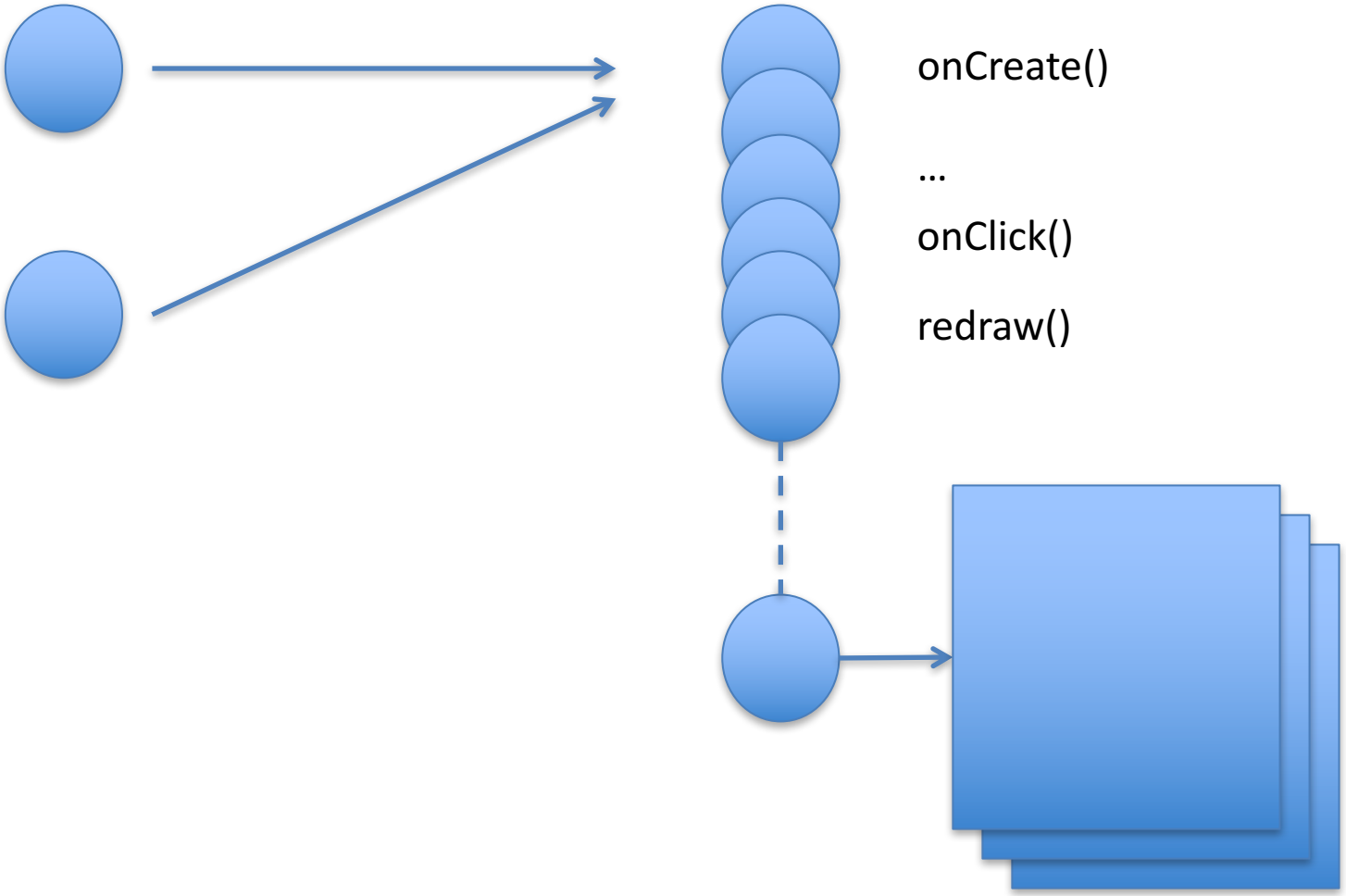
Threads and Services

- How long do things take?
  - Expected that Activities regularly transition to the ”background”
    - i.e. stopped
- Threads
  - Interacting with the UI thread
- Services
  - Application component #2
  - The Service lifecycle

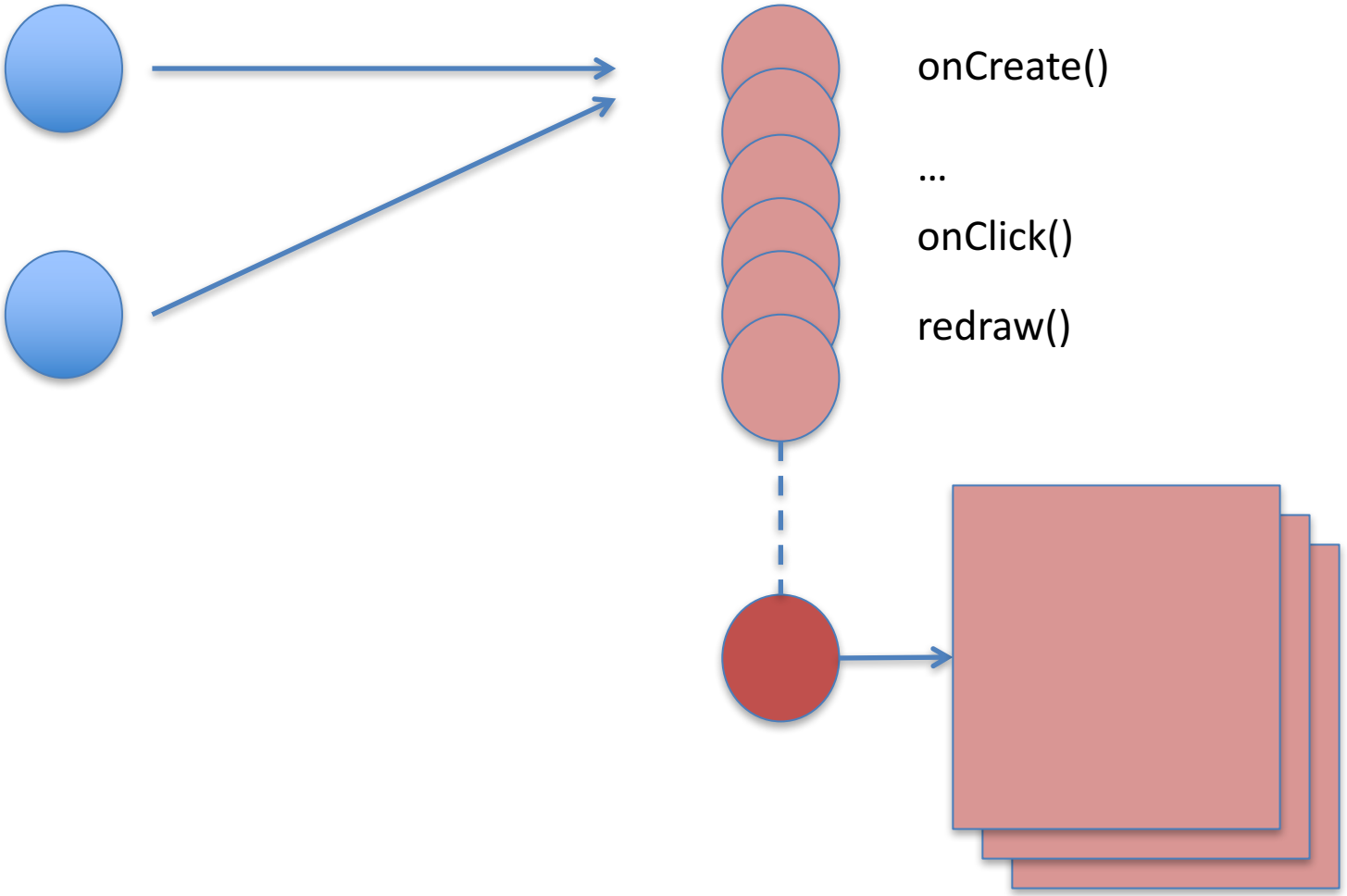
Threads

- Android applications use a **single thread model**
  - A single thread of execution called *main*
- Handles and dispatches user interface events
  - Drawing the interface
  - Responding to interactions
    - E.g. onClick()
- Handles activity lifecycle events
  - onCreate(), onDestroy…
- For **all** components in an application

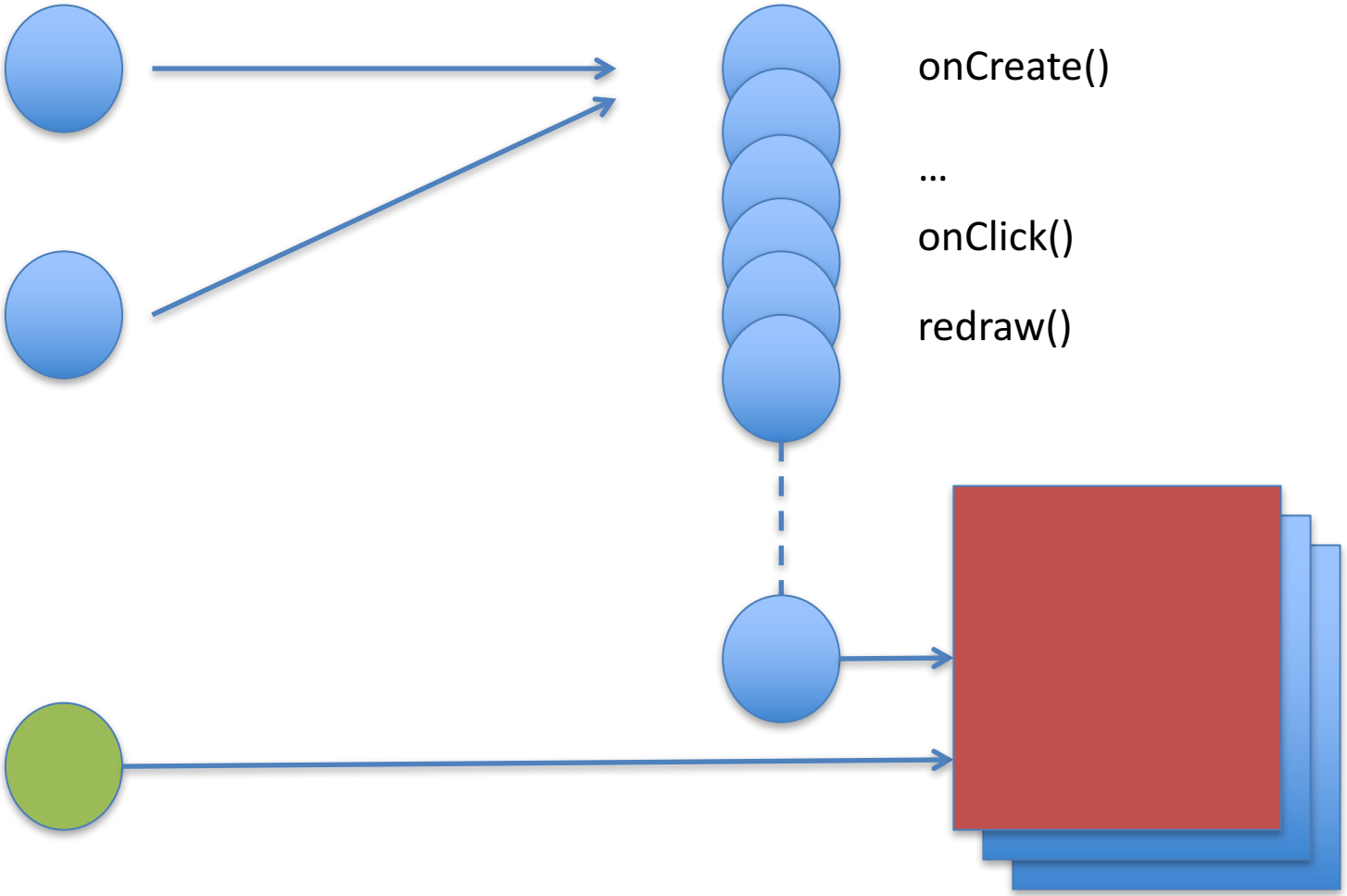
Threads / Looper



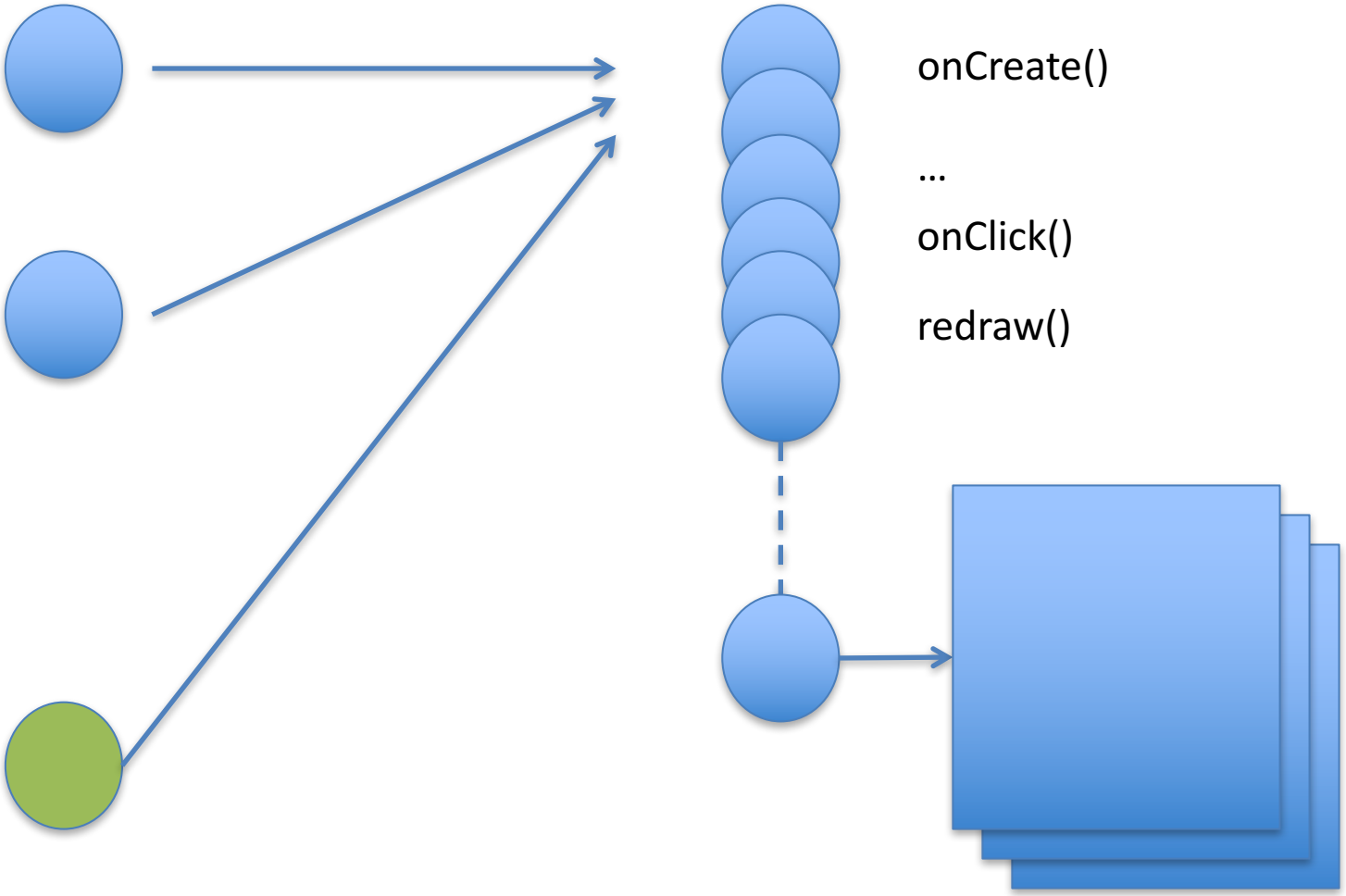
Threads / Looper



Threads / Looper



Threads / Looper



Threads

- How do we then execute code that may take a long time?
  - A long time > 1s
  - The application will appear to hang
  - “Application not responding” after 5s
- Put longer-running code / not instantaneous code in a separate thread of execution
  - Network access, file access
- Two golden rules
  - Do not block the UI thread
  - Do not access the UI thread from outside the UI thread
    - Concurrency!

Runnable

- We can programmatically interact with UI components
  - myTextField.setText(result);
  - Cannot call this method from outside the UI thread
    - Rule number 2
- Instead, split code into two parts
  - Long (ish) running code that does not involve the UI
    - E.g. an image download
      - Occurs in a separate thread of execution
      - Still tightly coupled to an activity
  - Instantaneous code that does involve the UI
    - E.g. drawing the image that has been downloaded
    - **posted** to the UI thread responsible for a particular View to execute, logically parceled up as a **Runnable** object

Handlers

- Provide a thread-safe way of talking to a specific thread of execution
  - Schedule messages and runnables to be executed at some point in the future
    - Runnable – a class to be run on a particular thread
    - Message – a package of data
  - Enqueue an action to be performed on a different thread than your own
    - UI thread -> worker thread
    - Worker thread -> UI thread
- Activity.runOnUiThread(Runnable …)

AsyncTask

- A convenience class for making complex asynchronous worker tasks easier
- Worker / blocking tasks
  - Executed in a background thread
- Results callback
  - Executed in the UI thread

Let’s have a look…

References

- http://developer.android.com/guide/components/processes-and-threads.html