Android Service &Process &Thread

郑灵翔

lxzheng@xmu.edu.cn

Android Application Building Blocks

Activity	• UI Component Typically Corresponding to one screen.
IntentReceiver	• Responds to notifications or status changes. Can wake up your process.
Service	• Faceless task that runs in the background.
ContentProvider	• Enable applications to share data.

Android Application Anatomy

Activities

1. Provides User Interface
2. Usually represents a Single Screen
3. Can contain one/more Views
4. Extends the Activity Base

Services

1. No User Interface
2. Runs in Background
3. Extends the Service Base Class

Application= Set of Android Components

Intent/Broadcast Receiver

1. Receives and Reacts to broadcast **Intents**
2. No UI but **can start** an Activity
3. **Extends** the **BroadcastReceiver** Base Class

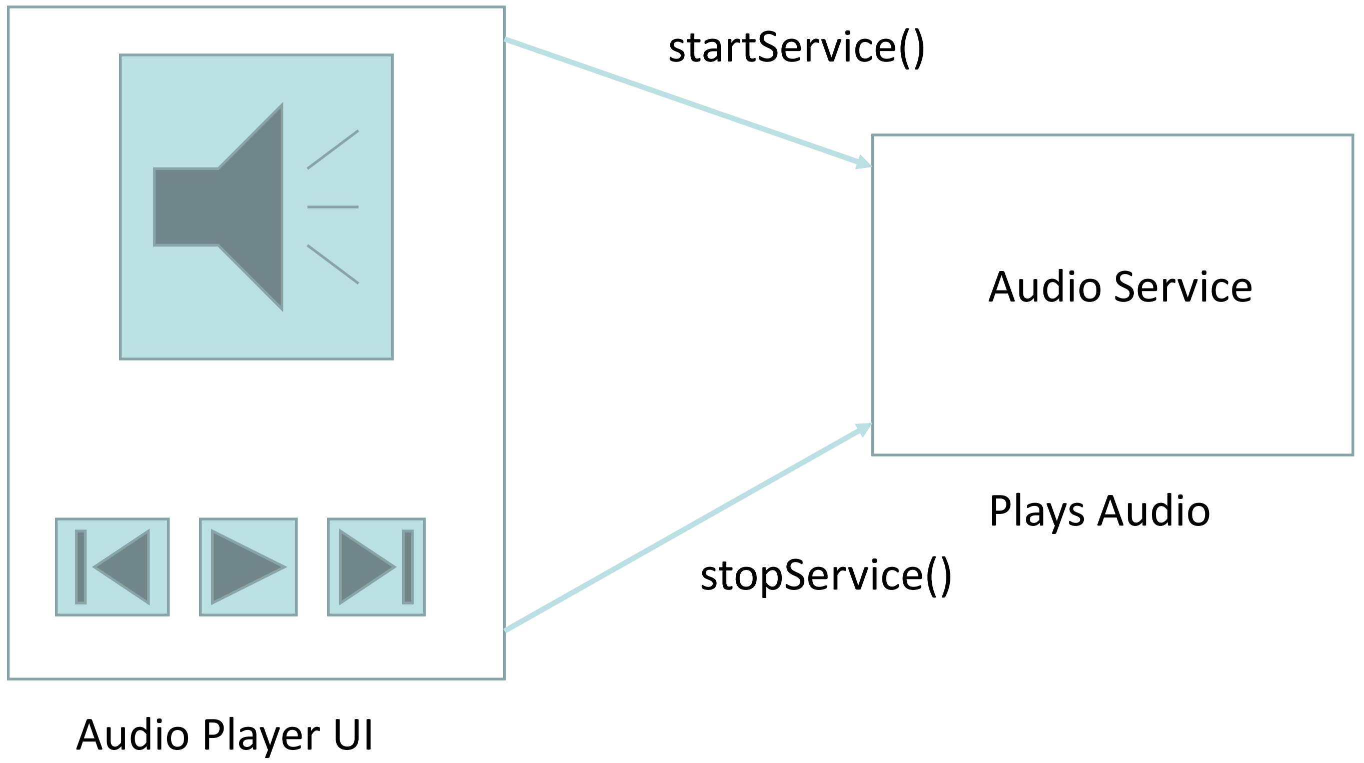
Content Provider

1. Makes application data available to other apps
2. Data stored in SQLite database
3. Extends the ContentProvider Base class

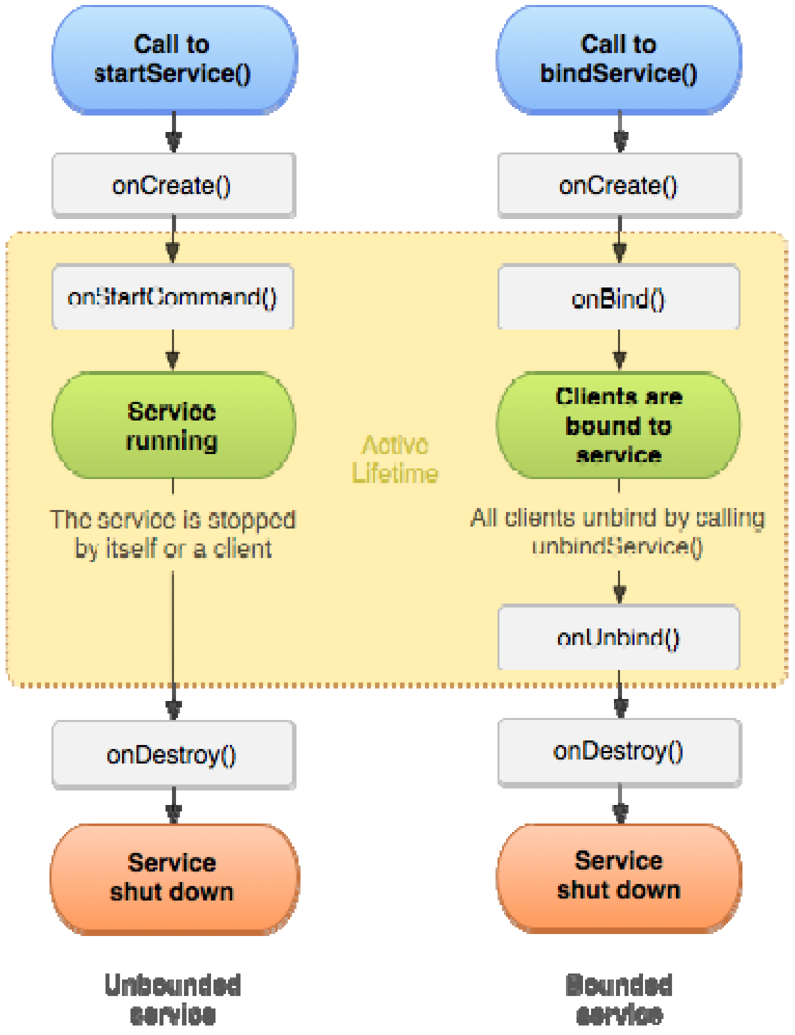
Service

What is Service?

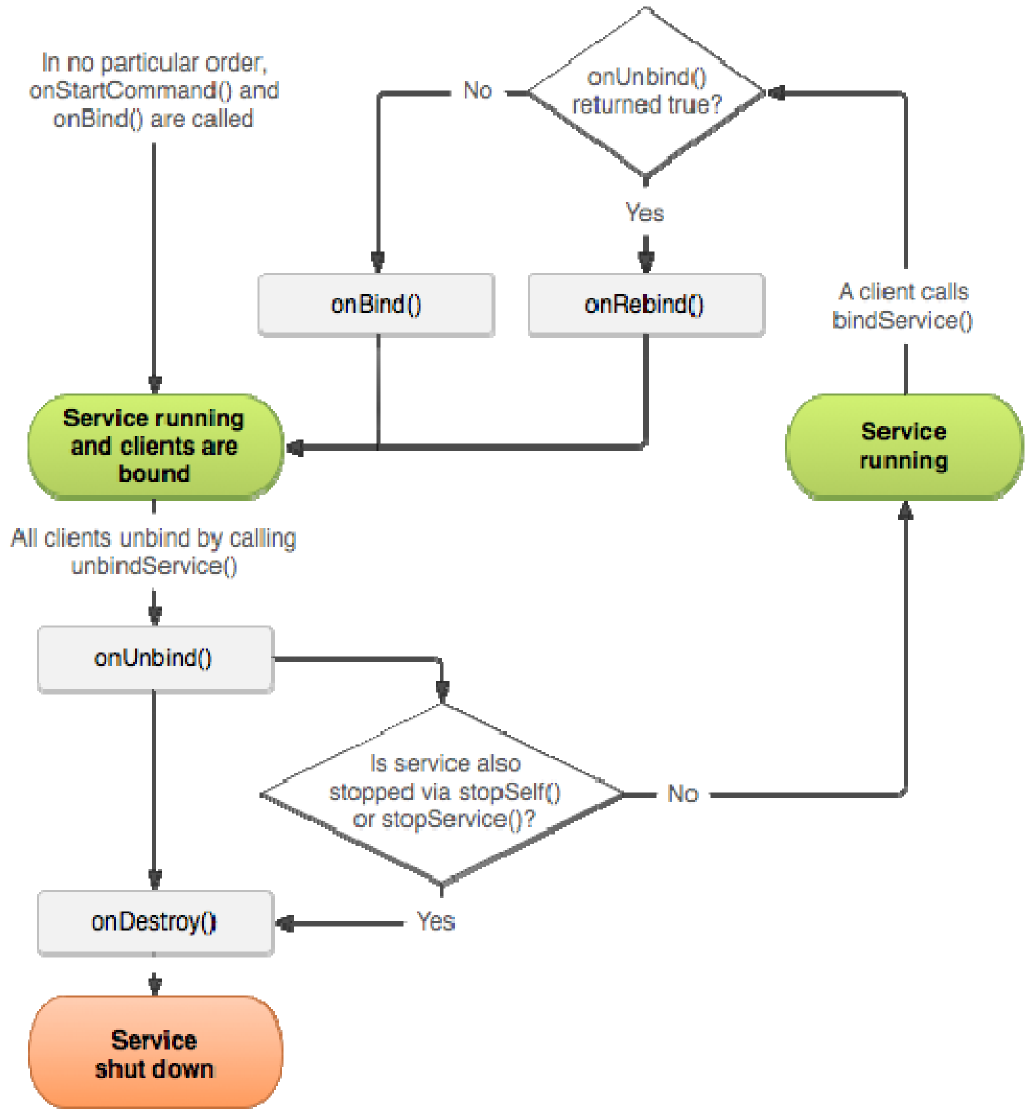
1. Services are codes that run **in the background**
2. They can be **started** and **stopped**
3. Services **doesn't have UI**



Service Lifecycle

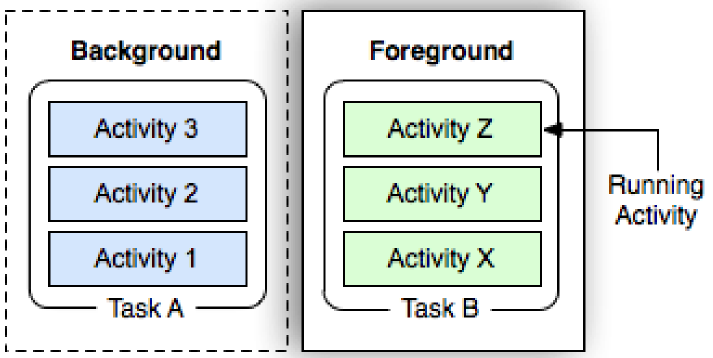


Bound Service Lifecycle

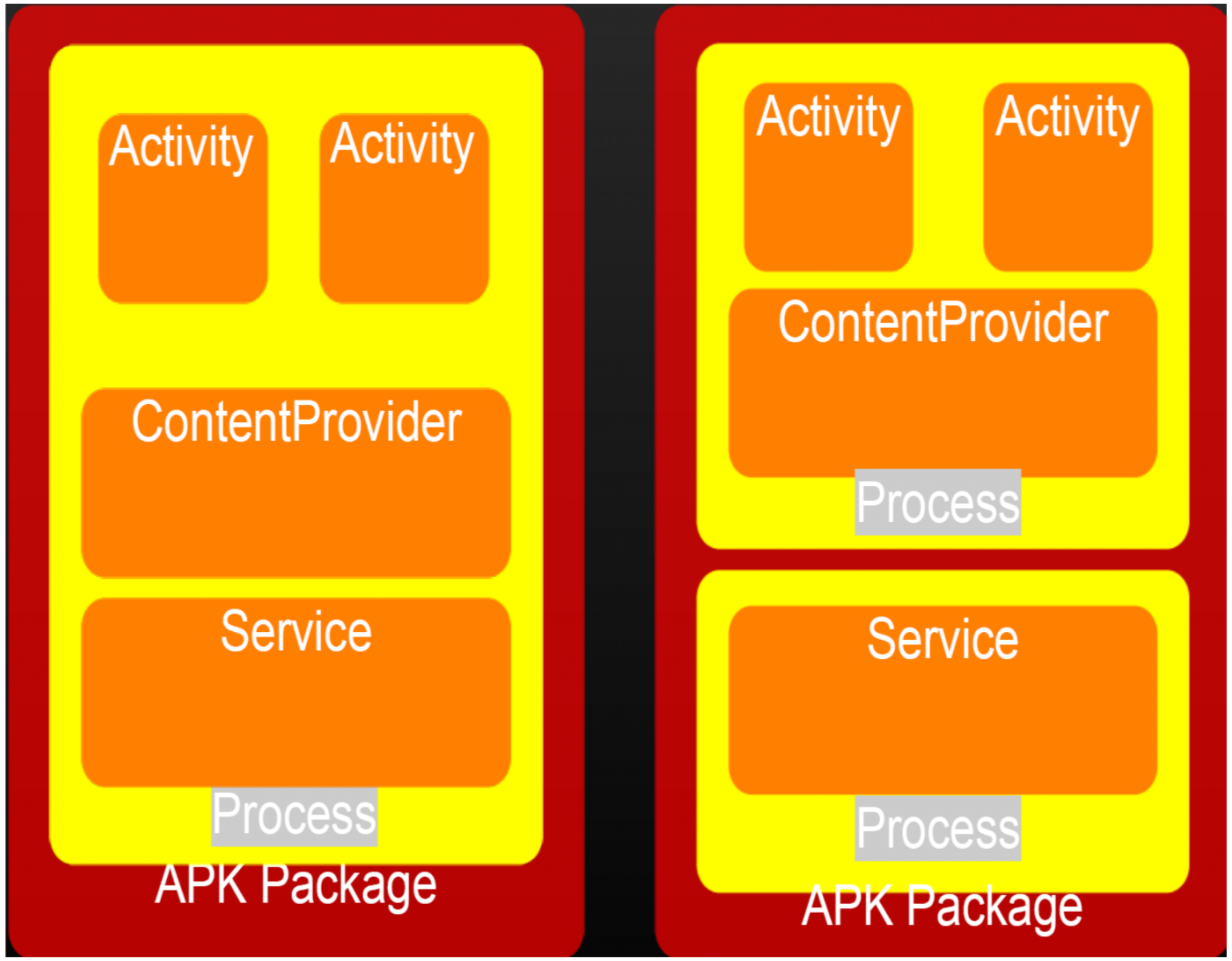

In no particular order, onStartCommand() and onBind() are called
No
onUnbind() returned true?
Yes
onBind()
onRebind()
A client calls bindService()
Service running and clients are bound
Service running
All clients unbind by calling unbindService()
onUnbind()
Is service also stopped via stopSelf() or stopService()?
No
onDestroy()
Yes
Service shut down

Application, Activity, Stack, Task, Process

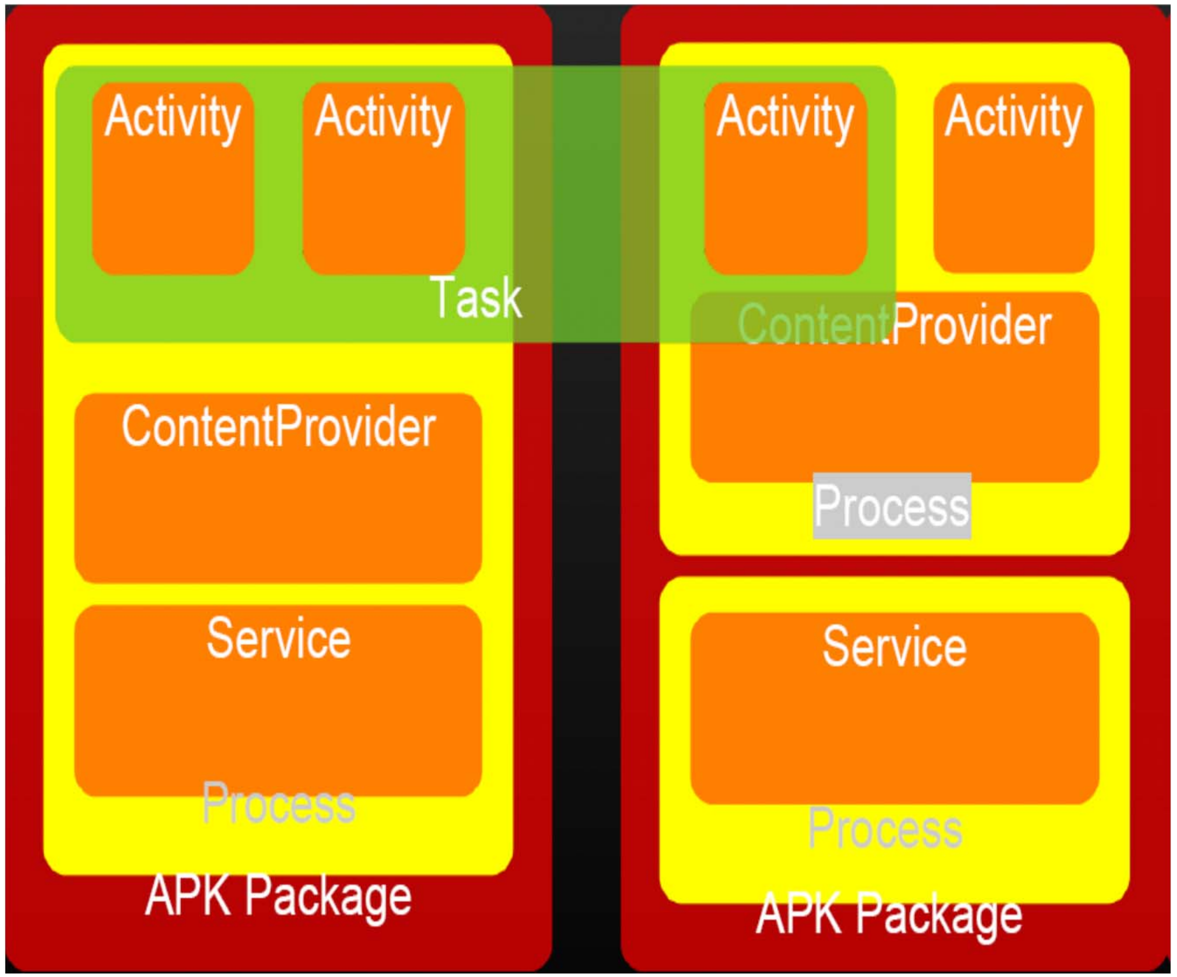
- Application
  - one or more related, loosely bound activities
  - bundled up in a single apk file
- Activity
  - the main building blocks of Android applications
- Stack
  - a linear navigation history of activities the user has visited
- Task
  - A task is the sequence of activities the user follows to accomplish an objective
- Process
  - A “process” is a standard Linux process
  - every application runs in its own Linux process



Activities and Tasks



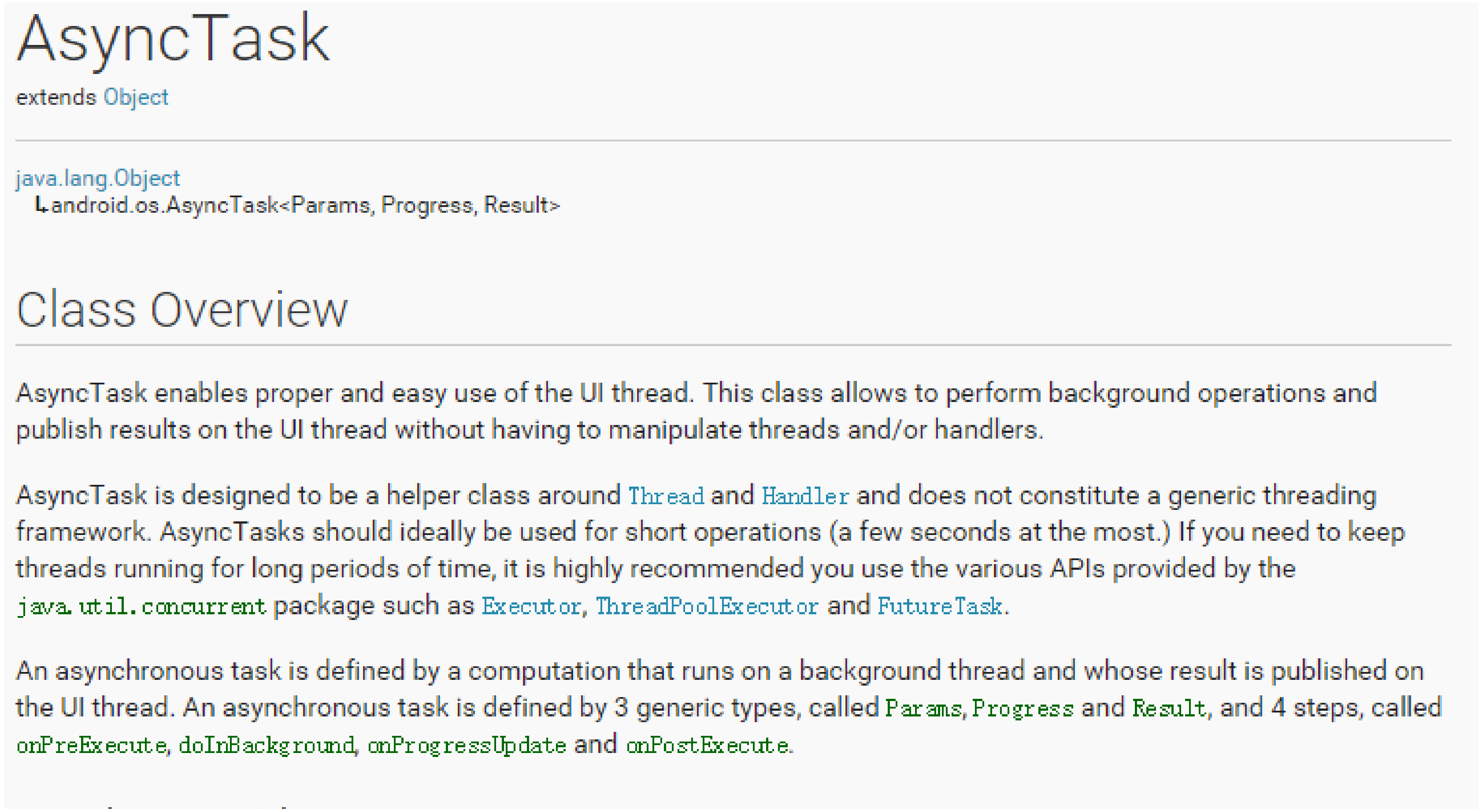
Activities and Tasks



Process & Thread

- 进程
  - 前台进程
  - 可见进程
  - 服务进程
  - 后台进程
  - 空进程
- 线程
  - UI 线程
  - 工作线程
- 线程安全方法

What class does Android use to Simplify background thread creation and UI thread synchronization?


AsyncTask

extends Object

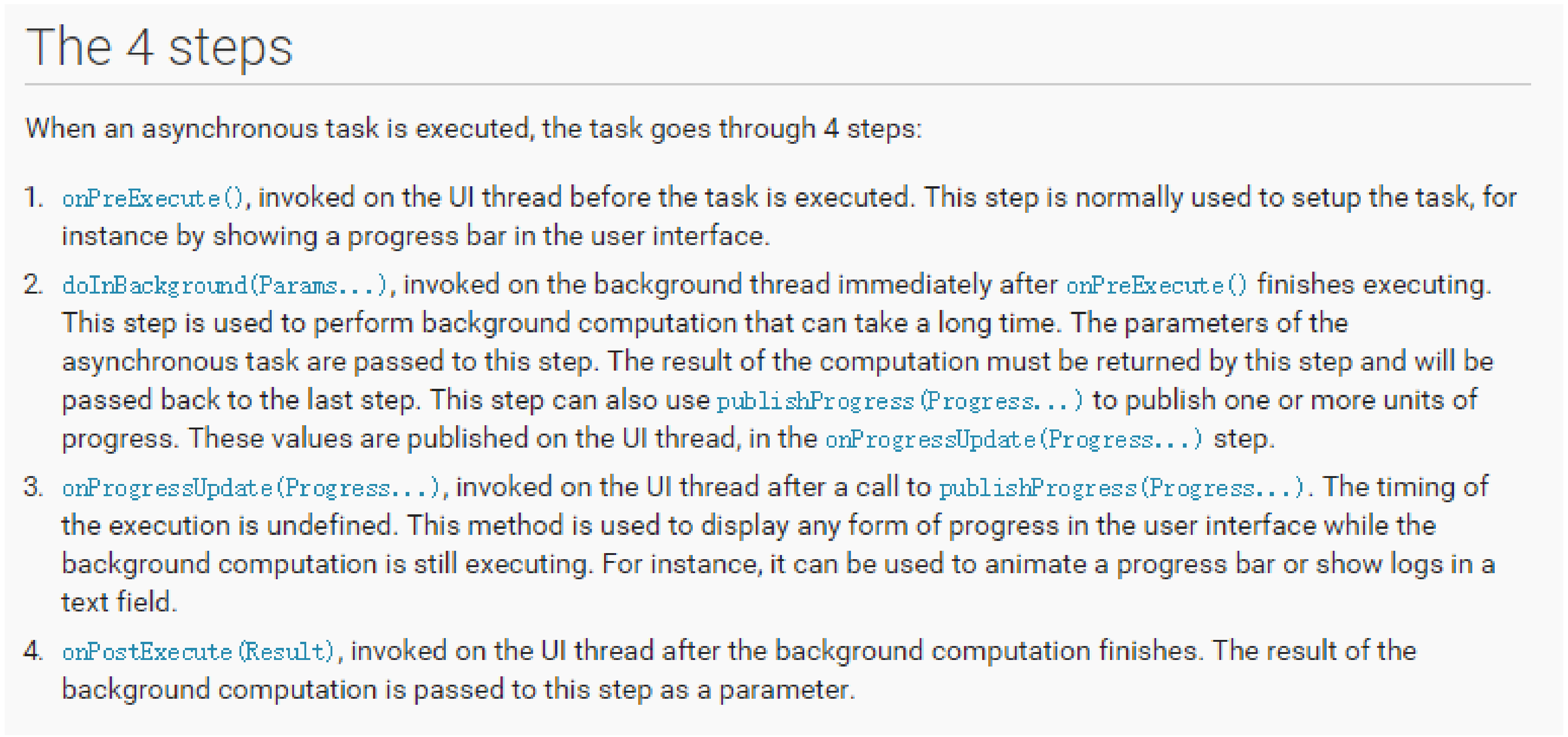
java.lang.Object
 ↳android.os.AsyncTask<Params, Progress, Result>

Class Overview

AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask is designed to be a helper class around `Thread` and `Handler` and does not constitute a generic threading framework. AsyncTasks should ideally be used for short operations (a few seconds at the most.) If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the `java.util.concurrent` package such as `Executor`, `ThreadPoolExecutor` and `FutureTask`.

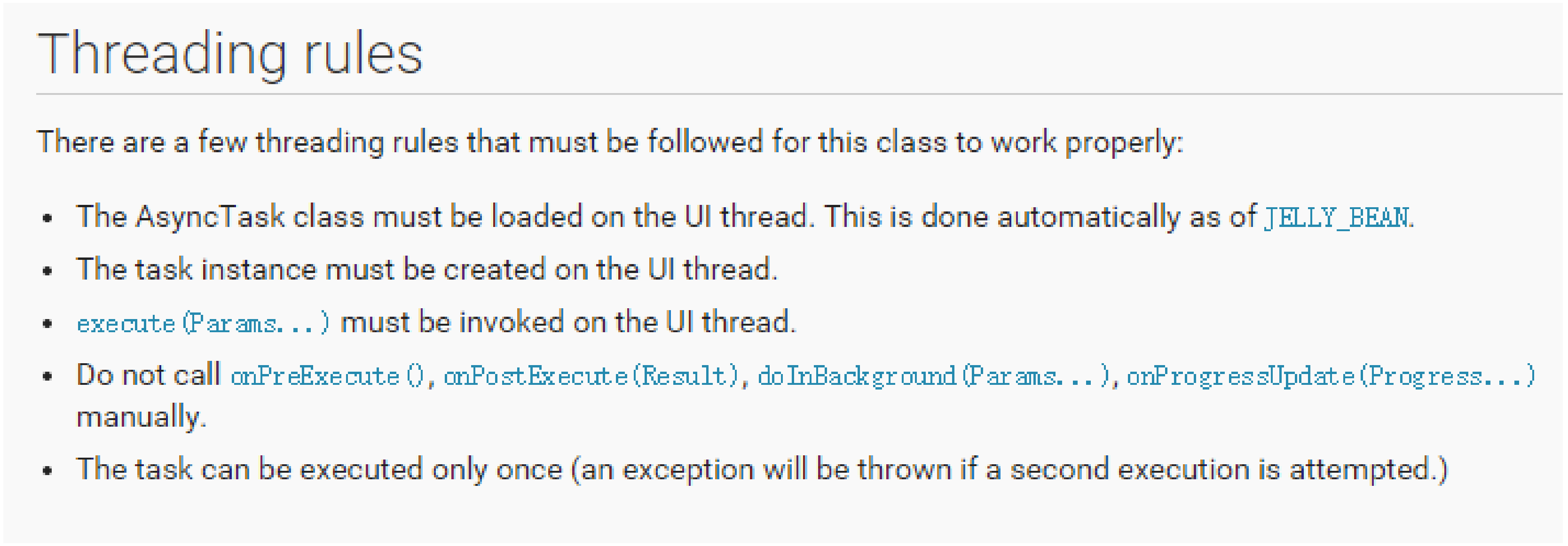
An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called `Params`, `Progress` and `Result`, and 4 steps, called `onPreExecute`, `doInBackground`, `onProgressUpdate` and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



The 4 steps

When an asynchronous task is executed, the task goes through 4 steps:

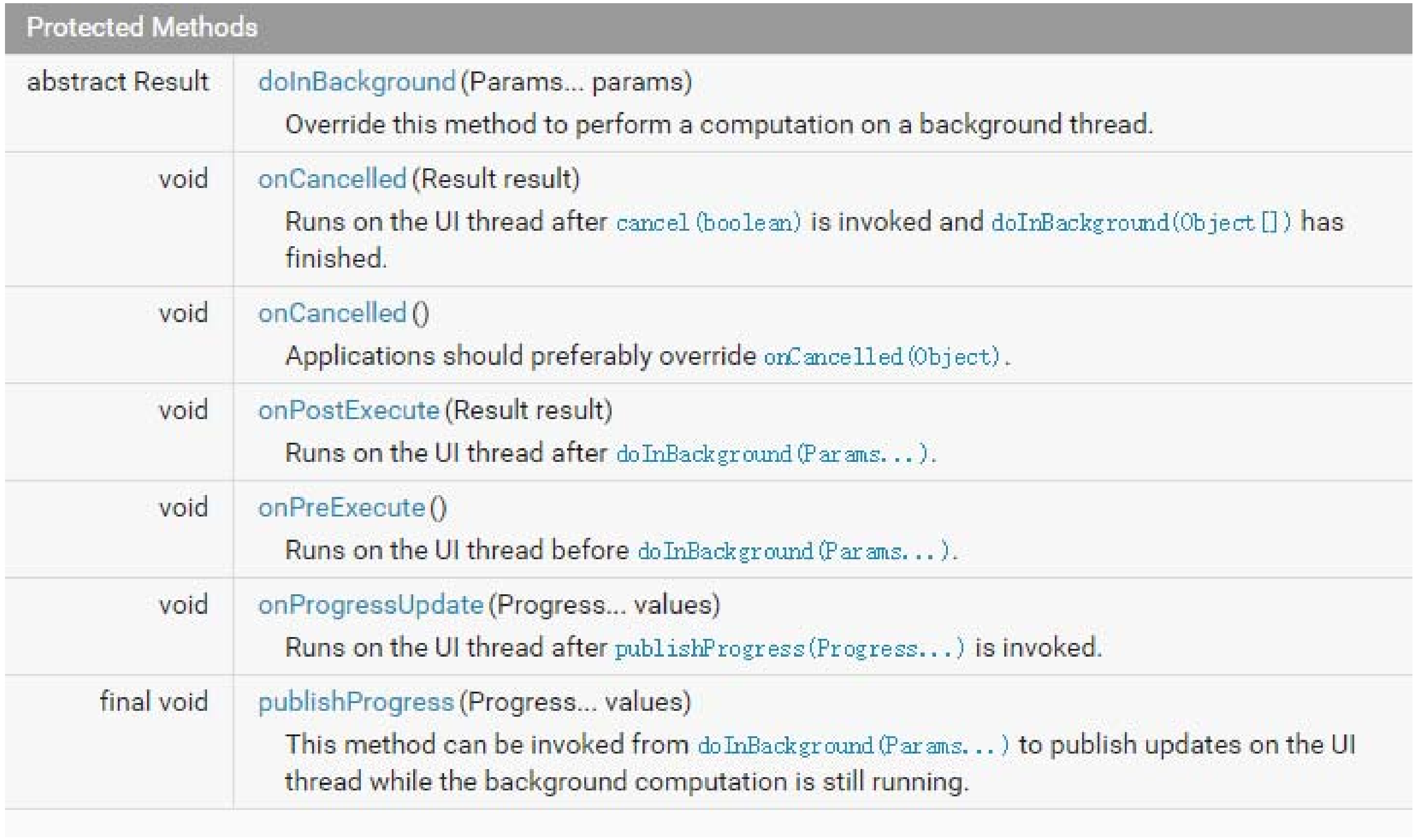
1. `onPreExecute()`, invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
2. `doInBackground(Params...)`, invoked on the background thread immediately after `onPreExecute()` finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. 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.
3. `onProgressUpdate(Progress...)`, invoked on the UI thread after a call to `publishProgress(Progress...)`. The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.
4. `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.



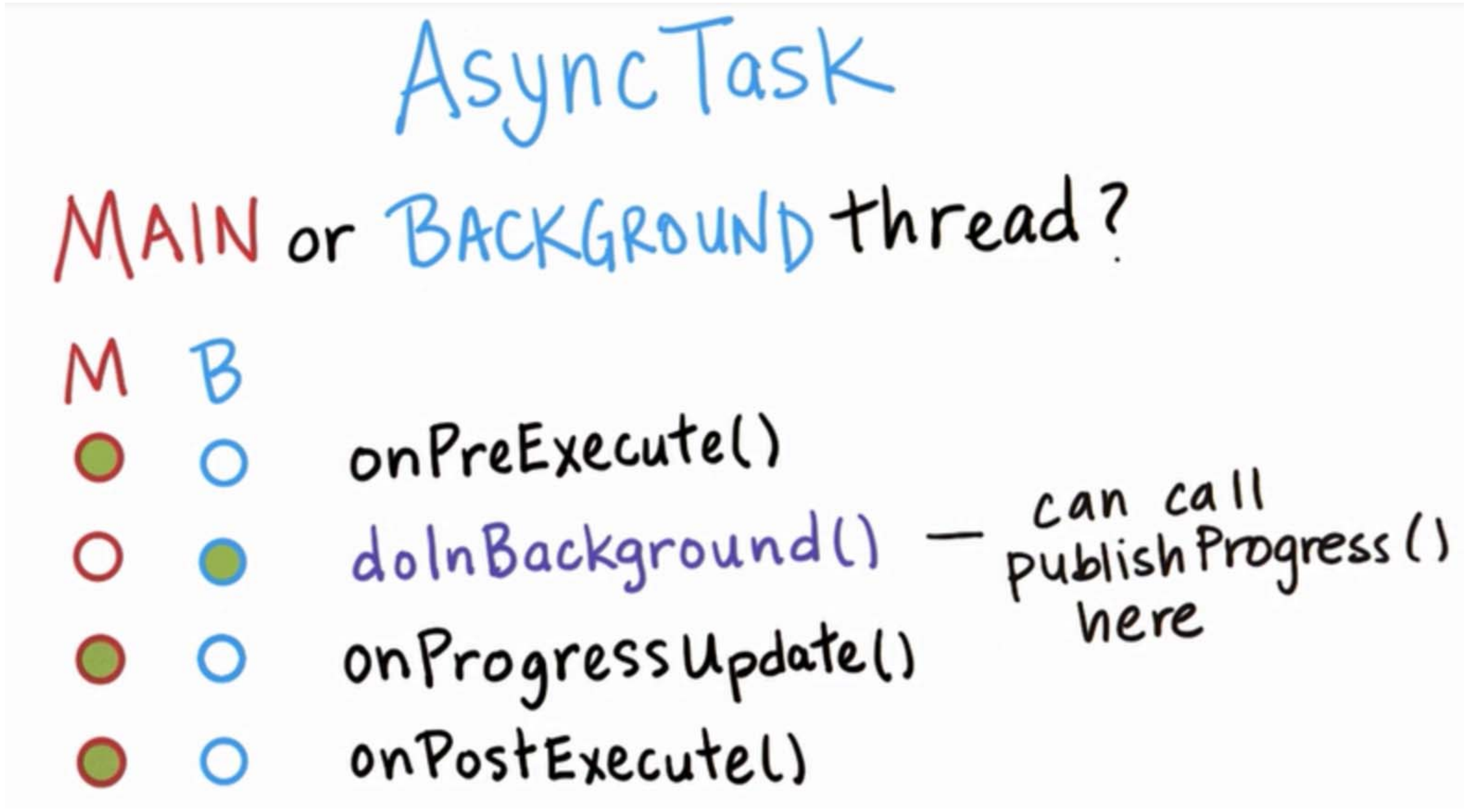
Threading rules

There are a few threading rules that must be followed for this class to work properly:

- The AsyncTask class must be loaded on the UI thread. This is done automatically as of `JELLY_BEAN`.
- The task instance must be created on the UI thread.
- `execute(Params...)` must be invoked on the UI thread.
- Do not call `onPreExecute()`, `onPostExecute(Result)`, `doInBackground(Params...)`, `onProgressUpdate(Progress...)` manually.
- The task can be executed only once (an exception will be thrown if a second execution is attempted.)



Protected Methods	
abstract Result	doInBackground(Params... params) Override this method to perform a computation on a background thread.
void	onCancelled(Result result) Runs on the UI thread after `cancel(boolean)` is invoked and `doInBackground(Object[])` has finished.
void	onCancelled() Applications should preferably override `onCancelled(Object)`.
void	onPostExecute(Result result) Runs on the UI thread after `doInBackground(Params...)`.
void	onPreExecute() Runs on the UI thread before `doInBackground(Params...)`.
void	onProgressUpdate(Progress... values) Runs on the UI thread after `publishProgress(Progress...)` is invoked.
final void	publishProgress(Progress... values) This method can be invoked from `doInBackground(Params...)` to publish updates on the UI thread while the background computation is still running.



AsyncTask

MAIN or BACKGROUND thread?

M	B	
●	○	onPreExecute()
○	●	doInBackground() — can call publishProgress() here
●	○	onProgressUpdate()
●	○	onPostExecute()