The AsyncTask Framework: Usage Considerations

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Recognize the capabilities provided by the Android AsyncTask framework
- Know which methods are provided by AsyncTask class
- Understand what black-box & white-box framework are... & how AsyncTask implements both types of frameworks
- Learn how the AsyncTaskInterrupted program works
- Appreciate AsyncTask usage considerations
AsyncTask Usage Considerations
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate

1. `execute(url)`
2. `onPreExecute()`
3. `execute(future)`
4. `doInBackGround()`
5. `onProgressUpdate()`
6. `onPostExecute()`

Its `onPreExecute()`, `onProgressUpdate()`, & `onPostExecute()` methods *always* run in the context of the UI thread!
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate

These methods are strongly connected via AsyncTask framework classes
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate

1. execute(url)
2. onPreExecute()
3. execute(future)
4. doInBackground()
5. onProgressUpdate()
6. onPostExecute()
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate

1. execute(url)
2. onPreExecute()
3. execute(future)
4. doInBackground()
5. onProgressUpdate()
6. onPostExecute()
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate

1. `execute(url)`
2. `onPreExecute()`
3. `execute(future)`
4. `doInBackground()`
5. `onProgressUpdate()`
6. `onPostExecute()`
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate

1. execute(url)
2. onPreExecute()
3. execute(future)
4. doInBackground()
5. onProgressUpdate()
6. onPostExecute()
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate
- Unlike HaMeR framework, no direct manipulation of handlers, messages, runnables, or threads
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework

See [www.dre.vanderbilt.edu/~schmidt/PDF/Queue-04.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/Queue-04.pdf)
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework, e.g.
  - Inversion of control

Diagram:
- UI Thread (main thread)
  - Looper
  - Message Queue
    - Message
    - Message
  - Handler
  - FutureTask
    - Executor

1. `execute(url)`
2. `onPreExecute()`
3. `execute(future)`
4. `doInBackground()`
5. `onProgressUpdate()`
6. `onPostExecute()`
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework, e.g.
  - Inversion of control
  - Domain-specific structure & functionality

```java
1. execute(url)
2. onPreExecute()
3. execute(future)
4. doInBackground()
5. onProgressUpdate()
6. onPostExecute()
```
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework, e.g.
  - Inversion of control
  - Domain-specific structure & functionality
  - Semi-complete portions of apps

```java
AsyncTask
execute()
cancel()
onPreExecute()
 doInBackground()
onProgressUpdate()
onPostExecute()
onCancelled()
```

```java
ImageDownloadTask
onPreExecute()
 doInBackground()
onProgressUpdate()
onPostExecute()
onCancelled()
```
AsyncTask Usage Considerations

• AsyncTask embodies key characteristics of a framework, e.g.
  • Inversion of control
  • Domain-specific structure & functionality
• Semi-complete portions of apps

**AsyncTask**

- execute()
- cancel()
- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()

**ImageDownloadTask**

- onPreExecute()
- doInBackground()
- onProgressUpdate()
- onPostExecute()
- onCancelled()
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework, e.g.
  - Inversion of control
  - Domain-specific structure & functionality
  - Semi-complete portions of apps

![AsyncTask Diagram](image)

```java
AsyncTask

execute()
cancel()
onPreExecute()
`doInBackground()`
onProgressUpdate()
onPostExecute()
onCancelled()

ImageDownloadTask

onPreExecute()
doInBackground()
onProgressUpdate()
onPostExecute()
onCancelled()
```
AsyncTask Usage Considerations

- AsyncTask has elements of both black-box & white-box frameworks

```java
// Black-box
: Threaded
: Download

Black-box

: Download
: Task

executeOnExecutor(executorStrategy, params)

onPreExecute()

onPostExecute()

execute()

call()

doInBackground()

postResult()

White-box

UI ➔

Thread

Background Threads

: Executor
: Worker
: Runnable
```
AsyncTask Usage Considerations

- AsyncTask has elements of both black-box & white-box frameworks, e.g.
- Its hook methods are elements of a white-box framework
AsyncTask Usage Considerations

- AsyncTask has elements of both black-box & white-box frameworks, e.g.
  - Its hook methods are elements of a white-box framework
  - Its executor strategy is an element of a black-box framework

```
UI thread
|               |
|               |
|               |
| Thread        |
|              |
| Return Result |
|              |
| Background   |
| Threads      |
```

```
DownloadTask

- doInBackground()
- onPostExecute()
- execute() (call())
- onPreExecute()
- executeOnExecutor() (executorStrategy, params)
- onPostExecute()
- doInBackground() (call())
- onPreExecute()
```

Black-box

White-box
AsyncTask Usage Considerations

• There are trade-offs between each approach
• White-box frameworks are generally easier to develop...
AsyncTask Usage Considerations

• There are trade-offs between each approach
  • White-box frameworks are generally easier to develop...
  • ... but harder to use
AsyncTask Usage Considerations

- There are trade-offs between each approach
  - White-box frameworks are generally easier to develop...
    - ... but harder to use
  - Black-box frameworks are generally harder to develop...
AsyncTask Usage Considerations

• There are trade-offs between each approach
  • White-box frameworks are generally easier to develop...
    • ... but harder to use
  • Black-box frameworks are generally harder to develop...
    • ... but easier to use

See en.wikipedia.org/wiki/Plug-in_(computing)
AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
AsyncTask Usage Considerations

• AsyncTask applies several GoF patterns
  • *Template Method* is used for its white-box capabilities

<table>
<thead>
<tr>
<th>AsyncTask</th>
</tr>
</thead>
<tbody>
<tr>
<td>execute()</td>
</tr>
<tr>
<td>cancel()</td>
</tr>
<tr>
<td>onPreExecute()</td>
</tr>
<tr>
<td>doInBackground()</td>
</tr>
<tr>
<td>onProgressUpdate()</td>
</tr>
<tr>
<td>onPostExecute()</td>
</tr>
<tr>
<td>onCancelled()</td>
</tr>
</tbody>
</table>

See [en.wikipedia.org/wiki/Template_method_pattern](en.wikipedia.org/wiki/Template_method_pattern)
AsyncTask Usage Considerations

• AsyncTask applies several GoF patterns
• *Template Method* is used for its white-box capabilities
AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
- *Template Method* is used for its white-box capabilities
AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
- *Template Method* is used for its white-box capabilities
AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
- *Template Method* is used for its white-box capabilities

```
AsyncTask usage:
  doInBackground() -> execute() -> onPostExecute()
  onPreExecute() -> execute() -> onPostExecute()
```

Diagram:
- : Threaded Download
- : Download Task
- : Executor
- : Worker Runnable

UI -> Background Thread

Execute() -> call() -> doInBackground() -> UI
AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
  - *Template Method* is used for its white-box capabilities
  - *Strategy* is used for its black-box capabilities

AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
  - *Template Method* is used for its white-box capabilities
  - *Strategy* is used for its black-box capabilities
  - *Facade* is used to simplify access to the Java Executor framework

See [en.wikipedia.org/wiki/Facade_pattern](en.wikipedia.org/wiki/Facade_pattern)
AsyncTask Usage Considerations

• AsyncTask also applies several POSA patterns
AsyncTask Usage Considerations

- AsyncTask also applies several POSA patterns
- *Half-Sync/Half-Async* is used to coordinate between the UI thread & background thread(s)

See [www.dre.vanderbilt.edu/~schmidt/PDF/HS-HA.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/HS-HA.pdf)
AsyncTask Usage Considerations

- AsyncTask also applies several POSA patterns
  - *Half-Sync/Half-Async* is used to coordinate between the UI thread & background thread(s)
  - *Pooling* is used to manage multiple instances of threads, which allows for reuse when AsyncTasks release threads they no longer need

See [www.kircher-schwanninger.de/michael/publications/Pooling.pdf](http://www.kircher-schwanninger.de/michael/publications/Pooling.pdf)
AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls

See bon-app-etit.blogspot.com/2013/04/the-dark-side-of-asynctask.html
AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls
- Cancellation
  - Cancellation is voluntary, just like Thread.interrupt()
AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
    - Memory leaks occur if there’s a strong references to enclosing Activity

See medium.com/@zhangqichuan/memory-leak-in-android-4a6a7e8d7780
AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
  - Losing results if/when runtime configurations change
    - e.g., Activity associated with an AsyncTask may be destroyed

See commonsware.com/blog/2010/09/10/asynctask-screen-rotation.html
AsyncTask Usage Considerations

• AsyncTask has traps & pitfalls
  • Cancellation
  • Dependency on Activity
  • Losing results if/when runtime configurations change
• Portability
  • Concurrency semantics of AsyncTask execute() have changed over time

Before API 1.6 (Donut):
• In the first version of AsyncTask, the tasks were executed serially, so a task won't start before a previous task is finished. This caused quite some performance problems. One task had to wait on another one to finish.

API 1.6 to API 2.3 (Gingerbread):
• The Android developers team decided to change this so that AsyncTasks could run parallel on a separate worker thread. There was one problem. Many developers relied on the sequential behavior and suddenly they were having a lot of concurrency issues.

API 3.0 (Honeycomb) until now
• "Hmmm, developers don't seem to get it? Let's just switch it back." The AsyncTasks where executed serially again. However, they can run parallel via executeOnExecutor(Executor).
AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
  - Losing results if/when runtime configurations change
  - Portability

See en.wikipedia.org/wiki/Model-view-presenter
AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
  - Losing results if/when runtime configurations change
  - Portability

Other issues can be addressed only by understanding Android patterns & APIs

See [developer.android.com/training/articles/perf-anr.html#Avoiding](https://developer.android.com/training/articles/perf-anr.html#Avoiding)
End of the AsyncTask Framework: Usage Considerations