Overview of Concurrency Patterns in Android & Java Frameworks (Part 2)

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

• Understand how *patterns* improve the structure & functionality of Java & Android concurrency frameworks used by apps & services

• Recognize key POSA patterns used in these concurrency frameworks
POSA Patterns in Android & Java Concurrency Frameworks
POSA Patterns in Android & Java Concurrency Frameworks

- Android & Java’s concurrency frameworks are designed, implemented, & integrated in accordance with many POSA patterns

See [www.dre.vanderbilt.edu/~schmidt/POSA](http://www.dre.vanderbilt.edu/~schmidt/POSA)
POSA Patterns in Android & Java Concurrency Frameworks

- Android & Java’s concurrency frameworks are designed, implemented, & integrated in accordance with many POSA patterns

<table>
<thead>
<tr>
<th>Android/Java Mechanism</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Looper</td>
<td>Thread-Specific Storage</td>
</tr>
<tr>
<td>HaMeR framework &amp; Java Executor framework</td>
<td>Command Processor &amp; Active Object</td>
</tr>
<tr>
<td>AsyncTask framework</td>
<td>Half-Sync/Half-Async</td>
</tr>
<tr>
<td>Java LinkedBlockingQueue</td>
<td>Monitor Object</td>
</tr>
<tr>
<td>Java Streams framework</td>
<td>Pipes &amp; Filters</td>
</tr>
</tbody>
</table>
Thread-Specific Storage – Allows multiple threads to use one ‘logically global’ access point to retrieve an object that is local to a thread, without incurring locking overhead on each object access.

See [www.dre.vanderbilt.edu/~schmidt/PDF/TSS-pattern.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/TSS-pattern.pdf)
• **Thread-Specific Storage** – Allows multiple threads to use one ‘logically global’ access point to retrieve an object that is local to a thread, without incurring locking overhead on each object access

• e.g., the Android Looper applies this pattern to ensure there’s only one Looper per thread

See upcoming discussions on “The **Thread-Specific Storage** Pattern”
POSA Patterns in Android & Java Concurrency Frameworks

• **Command Processor** – Package a piece of application functionality—as well as its parameterization in an object—to execute it in another context, such as a later point in time, in a different thread or process, etc.

See www.dre.vanderbilt.edu/~schmidt/PDF/CommandRevisited.pdf
• **Command Processor** – Package a piece of application functionality—as well as its parameterization in an object—to execute it in another context, such as a later point in time, in a different thread or process, etc.

  • e.g., Android’s HaMeR framework applies this pattern to process runnable commands in different threads

**Runnable**

```java
Runnable command = new Runnable() {
    @Override
    public void run() {
        // Execute command
    }
};
```

**Handler & Looper**

```java
Handler handler = new Handler(Looper.getMainLooper());

// Post to UI thread
handler.post(command);
```

**Background Thread**

```java
Runnable command = new Runnable() {
    @Override
    public void run() {
        // Execute command
    }
};

// Post to background thread
Runnable runnable = new Runnable() {
    @Override
    public void run() {
        // Execute command
    }
};

// Dispatch message to background thread
handler.post(runnable);
```

**Execution**

```java
// Execute command
execute(command);
```

See upcoming discussions on the “The **Command Processor Pattern**”
• **Active Object** – Define service requests on components as the units of concurrency & run service requests on a component in different thread(s) from the requesting client thread
POSAs Patterns in Android & Java Concurrency Frameworks

- **Active Object** – Define service requests on components as the units of concurrency & run service requests on a component in different thread(s) from the requesting client thread.
  - e.g., Android’s HaMeR framework applies this pattern to enable one thread to send messages to a handler in another thread.

See upcoming discussion on “The Active Object Pattern”
POSA Patterns in Android & Java Concurrency Frameworks

- **Active Object** – Define service requests on components as the units of concurrency & run service requests on a component in different thread(s) from the requesting client thread
  - e.g., Java’s ExecutorService framework applies this pattern to enable two-way callables to execute in a thread pool
POSA Patterns in Android & Java Concurrency Frameworks

- **Half-Sync/Half-Async** – Decouple async & sync processing via two layers that simplify programming without degrading performance

See [www.dre.vanderbilt.edu/~schmidt/PDF/HS-HA.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/HS-HA.pdf)
• **Half-Sync/Half-Async** – Decouple async & sync processing via two layers that simplify programming without degrading performance.
  
  e.g., Android’s AsyncTask framework applies this pattern to process tasks in background threads & publish results in UI thread.

---

**Diagram:**

**Synchronous Service Layer**

- **Background Thread$_1$**
  - <<take>>
  - **BlockingQueue**
    - <<take>>
    - <<take>>
  - **AsyncTask framework**
    - <<execute>>
    - **MyActivity**
      - <<next>>
    - **UI Thread Looper**
      - <<onPostExecute>>

**Queueing Layer**

- **Background Thread$_2$**
- **Background Thread$_3$**
**Monitor Object** – Synchronizes concurrent method execution to ensure only one method at a time runs within an object & allows an object’s methods to cooperatively schedule their execution sequences.

See [www.dre.vanderbilt.edu/~schmidt/PDF/monitor.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/monitor.pdf)
Monitor Object – Synchronizes concurrent method execution to ensure only one method at a time runs within an object & allows an object’s methods to cooperatively schedule their execution sequences

- e.g., Java’s LinkedBlockingQueue applies this pattern to implement a synchronized blocking queue

See upcoming discussions on "The Monitor Object Pattern"
POSAs Patterns in Android & Java Concurrency Frameworks

- **Pipes & Filters** – Divide an app’s tasks into multiple self-contained data processing steps & connect these steps via intermediate data buffers to form a data processing pipeline.

See hillside.net/plop/2011/papers/B-10-Hanmer.pdf
POSA Patterns in Android & Java Concurrency Frameworks

- **Pipes & Filters** – Divide an app’s tasks into multiple self-contained data processing steps & connect these steps via intermediate data buffers to form a data processing pipeline.
- e.g., Java streams framework applies this pattern to implement sequential & parallel streams.

See [hillside.net/plop/2011/papers/B-10-Hanmer.pdf](hillside.net/plop/2011/papers/B-10-Hanmer.pdf)
End of Overview of Concurrency Patterns in Android & Java Frameworks (Part 2)