Overview of the ImageTaskGang Application



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

Institute for Software Integrated Systems Vanderbilt University Nashville, Tennessee, USA



Learning Objectives in this Part of the Lesson

• Understand the ImageTaskGang application(s) & the patterns that are applied



See github.com/douglascraigschmidt/LiveLessons/tree/master/ImageTaskGang

 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently



See Android version at <u>github.com/douglascraigschmidt/</u> <u>LiveLessons/tree/master/ImageTaskGangApplication</u>

 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently



 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently



 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently



See en.wikipedia.org/wiki/NOP_(code)

 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently



See <u>en.wikipedia.org/wiki/Grayscale</u>

 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently



Various Java collections & thread pools are applied

 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently

| Strategy | Implementation |
|-------------|--------------------------|
| Executor | Fixed or variable-size |
| model | Thread pool |
| Unit of | Task per URL to download |
| concurrency | & image filter to apply |
| Results | Asynchronous Future |
| processing | model for immediate |
| model | concurrent processing |

Socket



Various concurrency strategies are also applied

 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently

| Strategy | Implementation |
|-------------|--------------------------|
| Executor | Fixed or variable-size |
| model | Thread pool |
| Unit of | Task per URL to download |
| concurrency | & image filter to apply |
| Results | Asynchronous Future |
| processing | model for immediate |
| model | concurrent processing |

Socket



 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently

| Strategy | Implementation |
|-------------|--------------------------|
| Executor | Fixed or variable-size |
| model | Thread pool |
| Unit of | Task per URL to download |
| concurrency | & image filter to apply |
| Results | Asynchronous Future |
| processing | model for immediate |
| model | concurrent processing |

Socket



 This app shows how the TaskGang framework can be customized to download, process, store, & display images concurrently

| Strategy | Implementation |
|-------------|--------------------------|
| Executor | Fixed or variable-size |
| model | Thread pool |
| Unit of | Task per URL to download |
| concurrency | & image filter to apply |
| Results | Asynchronous Future |
| processing | model for immediate |
| model | concurrent processing |

Socket



 We're going to focus largely on the command-line version of ImageTaskGang since it can use more modern Java features than the Android version

| Ima | geTaskGang $ angle$ src $ angle$ main $ angle$ java $ angle$ livelessons | s $ angle$ dimageTas | kGangTest | گ マ 🖌 📄 ImageTaskGang マ 🕨 🗯 🖏 マ ■ 🛛 Git: 🖌 🗸 🗡 🚫 | 5 Q 🖸 | |
|-------|---|----------------------|----------------------|---|---------------------------|---------|
| ect | 🔲 Project 🔻 😳 🔄 🗱 🖛 — | Ċ ImageTaskG | angTest.java $	imes$ | ImageTaskGang.java $	imes$ (C) TaskGang.java $	imes$ | | ())) |
| Proj | ImageTaskGang ~/Dropbox/Documer | 99 @ | priv | /ate static ImageTaskGang makeImageTaskGang 🛛 🔹 | x 1 <u>x</u> 1 ~ · | Data |
| | >gradle | 100 | | (Filten[] filtens | | base |
| mit | > build | TOO | | (IIIII) IIIIII, | | _ |
| Con | > DownloadImages | 101 | | List <list<url>> urlLists,</list<url> | | D |
| ¢ | > 🖿 gradle | 102 | | <pre>@NotNull TestsToRun choice) {</pre> | | evice I |
| lests | v src | 103 | | <pre>return switch (choice) {</pre> | | Manag |
| Requ | java | 100 | | | | ger |
| Pull | V Interview I | 104 | | case EXECUTUR_CUMPLETIUN_SERVICE_CACHED -> | | |
| 14 | > D filters | 105 | | new ImageTaskGang | | • Not |
| | ✓ Intasks | 104 | | (filtons | | ificat |
| | | 100 | | | | tions |
| | ✓ ➡ utils | 107 | | urlLists, | | ~ |
| | C ExceptionUtils | 108 | | <pre>testName: choice.toString(),</pre> | | Gra |
| | C Image | 100 | | // Create an Executor with a cached pool of Thread | | adle |
| | © Options | TOA | | // create an Executor with a cachea pool of intead | | |
| arks | C RunTimer | 110 | | <pre>// objects, which grow and shrink dynamically as new</pre> | | |
| Bookm | C StreamsUtils | 111 | | // tasks are executed. | | |
| | resources | 112 | | <pre>Executors.newCachedThreadPool()):</pre> | | |
| Ire | 🗬 build.gradle | 447 | | | | |
| ructu | gradlew | 112 | | case EXECUTOR_COMPLETION_SERVICE_FIXED -> | | |
| st St | i gradlew.bat | 114 | | new ImageTaskGang | | |
| æ | Settings.gradle | 115 | | (filters, | | i An |
| oolki | IIII External Libraries | 114 | | unllists | | droic |
| NS T | Scratches and Consoles | 110 | | UPTLISTS, | | Emu |
| (A) | | 117 | | <pre>testName: choice.toString(),</pre> | | ılator |
| | 🖡 Git 🕨 Run 🛛 🧖 CodeWhisperer Referenc | e Log 🖃 Log | jcat 🚱 Profiler | 🕏 Dependencies 🗮 TODO 🛛 🛛 Problems 🔯 Terminal 🖉 Services 🔮 App Inspection | | |
| | | | | 🛕 AWS: 2 Connections Expired 🗸 CodeWhisperer 99:35 CRLF UTF-8 4 spac | ces 🌵 master | |

See github.com/douglascraigschmidt/LiveLessons/tree/master/ImageTaskGang

 Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation



- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - POSA: Proactor, Future, & Pooling



See <u>en.wikipedia.org/wiki/Proactor_pattern</u>, <u>en.wikipedia.org/wiki/Futures_</u> and_promises, & <u>kircher-schwanninger.de/michael/publications/Pooling.pdf</u>

- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - POSA: Proactor, Future, & Pooling

The Proactor pattern separates the initiation of asynchronous operations from their handling, enabling efficient & scalable event-driven processing



See en.wikipedia.org/wiki/Proactor_pattern

- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - POSA: Proactor, Future, & Pooling

The Future pattern provides a proxy for a result that may not be available yet, allowing a program to continue execution without blocking until the result is needed



See en.wikipedia.org/wiki/Futures_and_promises

- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - POSA: Proactor, Future, & Pooling

The Pooling pattern involves creating & managing a pool of reusable resources to improve performance & efficiency by reducing overhead associated with creating/destroying resources repeatedly



See kircher-schwanninger.de/michael/publications/Pooling.pdf

- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - POSA: Proactor, Future, & Pooling
 - GoF: *Decorator, Factory Method,* & *Template Method*



See <u>en.wikipedia.org/wiki/Decorator_pattern</u>, <u>en.wikipedia.org/wiki/Factory</u> __method_pattern, & <u>en.wikipedia.org/wiki/Template_method_pattern</u>

- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - POSA: Proactor, Future, & Pooling
 - GoF: Decorator, Factory Method, & Template Method

The Decorator pattern allows behavior to be added to an object dynamically, providing flexible & alternative combinations of features without modifying the underlying object



See en.wikipedia.org/wiki/Decorator_pattern

- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - POSA: Proactor, Future, & Pooling
 - GoF: *Decorator, Factory Method,* & *Template Method*

The Factory Method pattern defines an interface for creating objects but allows implementations to decide which concrete class to instantiate, promoting loose coupling & extensibility



See en.wikipedia.org/wiki/Factory_method_pattern

- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - POSA: Proactor, Future, & Pooling
 - GoF: *Decorator, Factory Method,* & *Template Method*

The Template Method pattern defines the skeleton of an algorithm in a base class, allowing subclasses to override certain steps while keeping the overall structure of the code intact, enabling more reuse & customization



See en.wikipedia.org/wiki/Template_method_pattern

- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - The Singleton & Command patterns are also used in its implementation



- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - The Singleton & Command patterns PATTERN-ORIENTED SOFTWARE are also used in its implementation ARCHITECTURE A Pattern Language for Distributed Object Computing Volume 4 Frank Buschman Ceulia Henney Douglas C. Schmidt Michael Kirche The Singleton pattern Prashant Jain PATTERN-ORIENTED ensures that only one Douglas Schmid Michael Stal SOFTWARE Hans Rohnert Frank Buschma ARCHITECTURE instance of a class is Volume 3 Patterns for **Resource Management** created & provides a **Design** Patterns PATTERN-ORIENTED global point of access SOFTWARE Elements of Reusable ARCHITECTURE to it, commonly used Object-Oriented Software **Patterns for Concurrent** Erich Gamma Richard Helm and Networked Objects for scenarios where a Ralph Johnson TLEY WARE DESIGN PA single, shared resource needs to be managed

See en.wikipedia.org/wiki/Singleton_pattern

- Several "Gang-of-Four" & POSA patterns are applied to enhance the framework-based ImageTaskGang implementation
 - These patterns most essential to its design
 - The *Singleton* & *Command* patterns are also used in its implementation

The Command pattern encapsulates a request as an object, allowing the parameterization of clients with different requests, queuing or logging requests, & supporting operations like undo & redo



See en.wikipedia.org/wiki/Command_pattern

End of the Overview of the ImageTaskGang Application