Overview of the TaskGang Framework

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 the SearchTaskGang case study
- Recognize the methods that are defined by the TaskGang framework

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

See SearchTaskGang/src/main/java/utils/TaskGang.java

Overview of the TaskGang Framework

Overview of the TaskGang Framework

- Defines a framework for spawning & running a "gang" of tasks

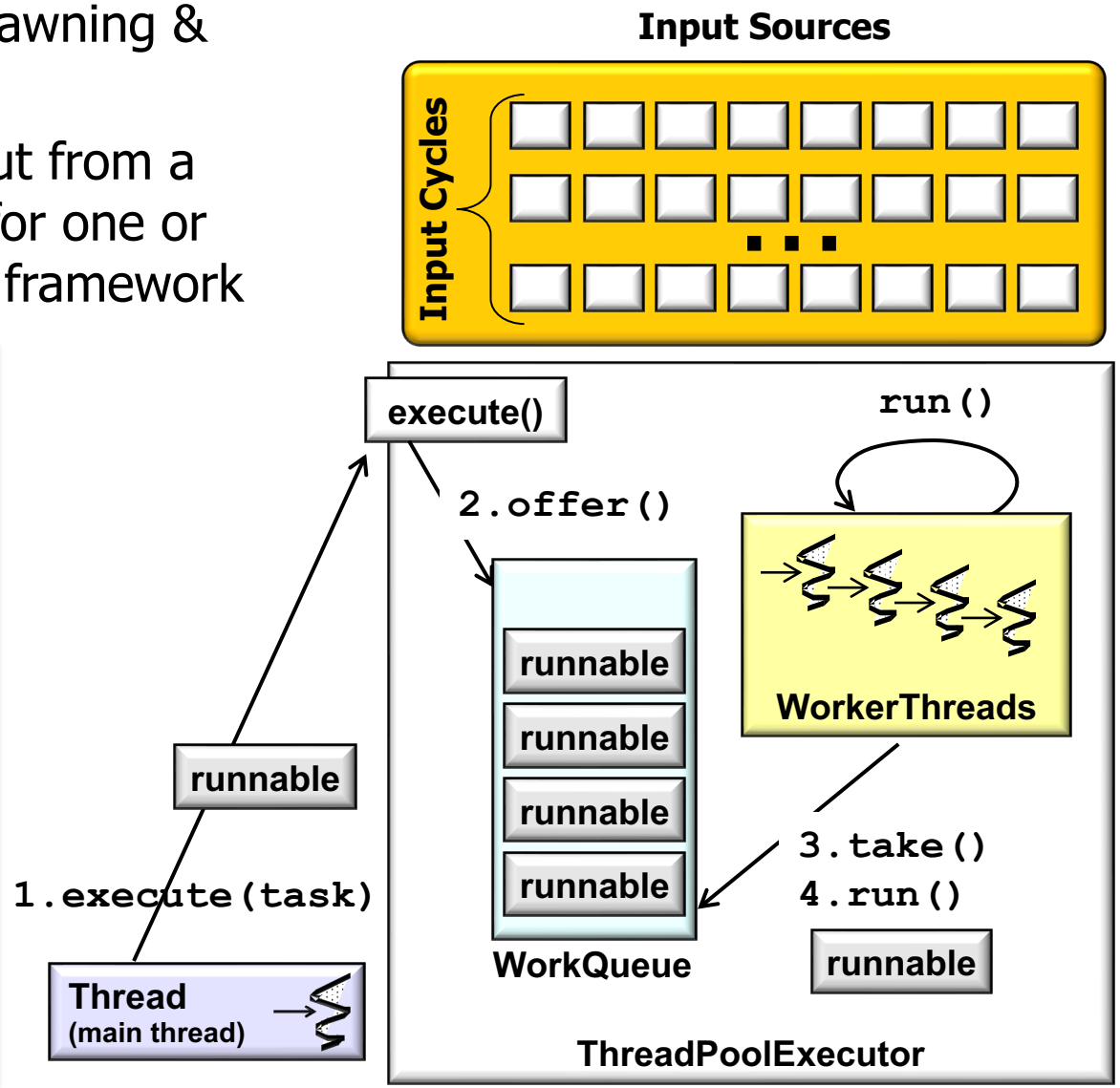
TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

A “task” is a command that can execute in a background Thread

Overview of the TaskGang Framework

- Defines a framework for spawning & running a "gang" of tasks
  - Concurrently process input from a generic List of elements for one or more cycles via Executor framework

TaskGang<E>	
TaskGang()	
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void



See SearchTaskGang/src/main/java/utils/TaskGang.java

Overview of the TaskGang Framework

- Defines a framework for spawning & running a "gang" of tasks
  - Concurrently process input from a generic List of elements for one or more cycles via Executor framework
  - Useful for "embarrassingly parallel" computations

See en.wikipedia.org/wiki/Embarrassingly_parallel

Overview of the TaskGang Framework

- Defines a framework for spawning & running a "gang" of tasks
  - Concurrently process input from a generic List of elements for one or more cycles via Executor framework
  - Useful for "embarrassingly parallel" computations
    - e.g., little or no effort required to separate the problem into a number of parallel tasks

Overview of the TaskGang Framework

- Defines a framework for spawning & running a "gang" of tasks
  - Concurrently process input from a generic List of elements for one or more cycles via Executor framework
  - Useful for "embarrassingly parallel" computations
  - Representative case study for "commonality" & "variability" in framework design

See www.dre.vanderbilt.edu/~schmidt/PDF/Commonality_Variability.pdf

Overview of the TaskGang Framework

- The framework itself supports "commonality"

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

Overview of the TaskGang Framework

- The framework itself supports "commonality", e.g.
  - Common data members & method signatures reused by TaskGang framework & applications that customize the framework

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

e.g., BarrierTaskGang & ImageTaskGang

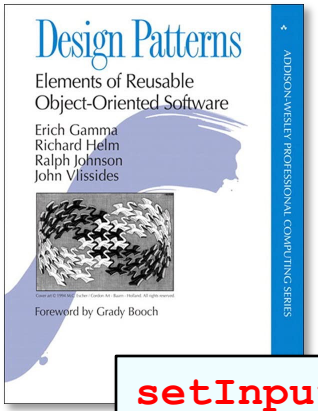
Overview of the TaskGang Framework

- The framework itself supports "commonality", e.g.
  - Common data members & method signatures
  - Common algorithms & control flow

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

Overview of the TaskGang Framework

- The framework itself supports "commonality", e.g.
  - Common data members & method signatures
  - Common algorithms & control flow
    - run() is a template method that defines the entry into a task gang



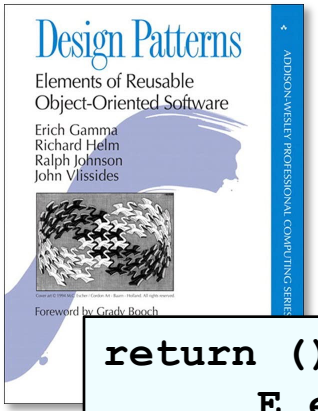

```
setInput(getNextInput());
initiateTaskGang
    (getInput().size());
awaitTasksDone();
```

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

See en.wikipedia.org/wiki/Template_method_pattern

Overview of the TaskGang Framework

- The framework itself supports "commonality", e.g.
  - Common data members & method signatures
  - Common algorithms & control flow
    - makeTask() factory method creates Runnable (often run concurrently)




```
return () -> { ...
      E e =
        getInput().get(index);
      if (processInput(e)))
        taskDone(index);
      ... };
```

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

See en.wikipedia.org/wiki/Factory_method_pattern

Overview of the TaskGang Framework

- The framework itself supports "commonality", e.g.
  - Common data members & method signatures
  - Common algorithms & control flow
    - advanceTaskToNextCycle() controls whether "one-shot" or "cyclic" processing occurs

```
return false;
```

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

Defaults to just running once (i.e., a "one-shot")

Overview of the TaskGang Framework

- The framework itself supports "commonality", e.g.
  - Common data members & method signatures
  - Common algorithms & control flow
    - AtomicLong is used to increment & read the current cycle

```
TaskGang<E>
TaskGang()
mCurrentCycle                    AtomicLong
mExecutor                        Executor
mInput                           List<E>
advanceTaskToNextCycle()         boolean
awaitTasksDone()                 void
currentCycle()                   long
getExecutor()                    Executor
getInput()                       List<E>
getNextInput()                   List<E>
incrementCycle()                 long
initiateHook(int)                void
initiateTaskGang(int)            void
makeTask(int)                    Runnable
processInput(E)                  boolean
run()                            void
setExecutor(Executor)            void
setInput(List<E>)                List<E>
taskDone(int)                    void
```

Keeps track of the cycle count

Overview of the TaskGang Framework

- Framework must be customized to support "variability"

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from
    - e.g., from files, strings, network connection, etc.

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from
  - How many cycles to run

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from
  - How many cycles to run
    - e.g., one-shot or cyclic

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from
  - How many cycles to run
  - How to structure threading & synchronization

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from
  - How many cycles to run
  - How to structure threading & synchronization, e.g.
    - Which type of Executor
      - e.g., fixed vs. cached

```
TaskGang<E>
TaskGang()
mCurrentCycle                    AtomicLong
mExecutor                        Executor
mInput                           List<E>
advanceTaskToNextCycle()         boolean
awaitTasksDone()                 void
currentCycle()                   long
getExecutor()                    Executor
getInput()                       List<E>
getNextInput()                   List<E>
incrementCycle()                 long
initiateHook(int)                void
initiateTaskGang(int)            void
makeTask(int)                    Runnable
processInput(E)                  boolean
run()                            void
setExecutor(Executor)            void
setInput(List<E>)                List<E>
taskDone(int)                    void
```

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from
  - How many cycles to run
  - How to structure threading & synchronization, e.g.
    - Which type of Executor
    - Which type of concurrency model
      - e.g., Thread pool vs. Thread -per-input element

```
TaskGang<E>
TaskGang()
mCurrentCycle                    AtomicLong
mExecutor                        Executor
mInput                           List<E>
advanceTaskToNextCycle()         boolean
awaitTasksDone()                 void
currentCycle()                   long
getExecutor()                    Executor
getInput()                       List<E>
getNextInput()                   List<E>
incrementCycle()                 long
initiateHook(int)                void
initiateTaskGang(int)            void
makeTask(int)                    Runnable
processInput(E)                  boolean
run()                            void
setExecutor(Executor)            void
setInput(List<E>)                List<E>
taskDone(int)                    void
```

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from
  - How many cycles to run
  - How to structure threading & synchronization, e.g.
    - Which type of Executor
    - Which type of concurrency model
    - What type of synchronizer
      - e.g., CyclicBarrier, Phaser, or CountDownLatch

TaskGang<E>	
TaskGang()	
mCurrentCycle	AtomicLong
mExecutor	Executor
mInput	List<E>
advanceTaskToNextCycle()	boolean
awaitTasksDone()	void
currentCycle()	long
getExecutor()	Executor
getInput()	List<E>
getNextInput()	List<E>
incrementCycle()	long
initiateHook(int)	void
initiateTaskGang(int)	void
makeTask(int)	Runnable
processInput(E)	boolean
run()	void
setExecutor(Executor)	void
setInput(List<E>)	List<E>
taskDone(int)	void

CyclicBarrier must be used with Thread-per-Input model

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from
  - How many cycles to run
  - How to structure threading & synchronization
  - What processing to perform on each input element
    - e.g., synchronous vs. asynchronously

```
TaskGang<E>
TaskGang()
mCurrentCycle                    AtomicLong
mExecutor                        Executor
mInput                           List<E>
advanceTaskToNextCycle()         boolean
awaitTasksDone()                 void
currentCycle()                   long
getExecutor()                    Executor
getInput()                       List<E>
getNextInput()                   List<E>
incrementCycle()                 long
initiateHook(int)                void
initiateTaskGang(int)            void
makeTask(int)                    Runnable
processInput(E)                  boolean
run()                            void
setExecutor(Executor)            void
setInput(List<E>)                List<E>
taskDone(int)                    void
```

Can support highly concurrent processing via thread pools or virtual threads!

Overview of the TaskGang Framework

- Framework must be customized to support "variability", e.g.
  - Where the data comes from
  - How many cycles to run
  - How to structure threading & synchronization
  - What processing to perform on each input element
  - How to wait for all the tasks in the gang to complete
    - e.g., join(), CountDownLatch, etc.

```
TaskGang<E>
TaskGang()
mCurrentCycle                    AtomicLong
mExecutor                        Executor
mInput                           List<E>
advanceTaskToNextCycle()         boolean
awaitTasksDone()                 void
currentCycle()                   long
getExecutor()                    Executor
getInput()                       List<E>
getNextInput()                   List<E>
incrementCycle()                 long
initiateHook(int)                void
initiateTaskGang(int)            void
makeTask(int)                    Runnable
processInput(E)                  boolean
run()                            void
setExecutor(Executor)            void
setInput(List<E>)                List<E>
taskDone(int)                    void
```

Walkthrough of the TaskGang Class

SearchTaskGang – TaskGang.java [SearchTaskGang.main]

SearchTaskGang 〉 src 〉 main 〉 java 〉 utils 〉 TaskGang

```
/**
 * Defines a framework for spawning and running a "gang" of tasks that
 * concurrently process input from a generic {@link List} of elements
 * {@code E} for one or more cycles.
 */
public abstract class TaskGang<E>
        implements Runnable {
    /**
     * The input {@link List} that's processed, which can be
     * initialized via the {@code setInput()} method.
     */
    private volatile List<E> mInput;

    /**
     * Executes submitted Runnable tasks in a {@link Thread} pool.
     */
    private Executor mExecutor;

    /**
     * Keeps track of which cycle is currently active.
     */
```

See SearchTaskGang/src/main/java/utils/TaskGang.java

End of Overview of the TaskGang Framework