

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

	🖻 🖌 TaskGang <i< th=""><th>&gt;</th></i<>	>
<b>m</b> 🛍	TaskGang <b>()</b>	
<b>f</b>	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m 💡	advanceTaskToNextCyc	le() boolean
(m) 🤉	awaitTasksDone <b>()</b>	void
m 🤋	currentCycle()	long
m ę	getExecutor()	Executor
m 💡	getInput()	List <e></e>
(m) 🤉	getNextInput()	List <e></e>
m 💡	incrementCycle()	long
m 💡	initiateHook(int)	void
(m) 🤮	initiateTaskGang(int)	void
m 💡	makeTask <b>(</b> int <b>)</b>	Runnable
(m) 🤉	processInput <b>(</b> E)	boolean
<b>m</b> 🛍	run <b>()</b>	void
m ያ	setExecutor(Executor)	void
m 💡	setInput(List <e>)</e>	List <e></e>
m ?	taskDone(int)	void

See <u>SearchTaskGang/src/main/java/utils/TaskGang.java</u>

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

🕒 🖕 TaskGang <e></e>		
<b>m</b> 🔒	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
<b>m ?</b>	advanceTaskToNextCy	cle <b>()</b> boolean
(m) 🤉	awaitTasksDone <b>()</b>	void
<b>m ?</b>	currentCycle()	long
m 🤋	getExecutor()	Executor
<b>m ?</b>	getInput <b>()</b>	List <e></e>
(m) 🤉	getNextInput()	List <e></e>
<b>m ?</b>	incrementCycle()	long
<b>m ?</b>	initiateHook(int)	void
(m) 🤉	initiateTaskGang(int)	void
<b>m ?</b>	makeTask(int)	Runnable
(m) 🤉	processInput(E)	boolean
<b>m</b> 🛍	run <b>()</b>	void
<b>m ?</b>	setExecutor(Executor)	void
m 🤋	setInput(List <e>)</e>	List <e></e>
<b>m ?</b>	taskDone(int)	void

A "task" is a command that can execute in a background Thread

**Input Sources** 

- 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

(m)

m

m

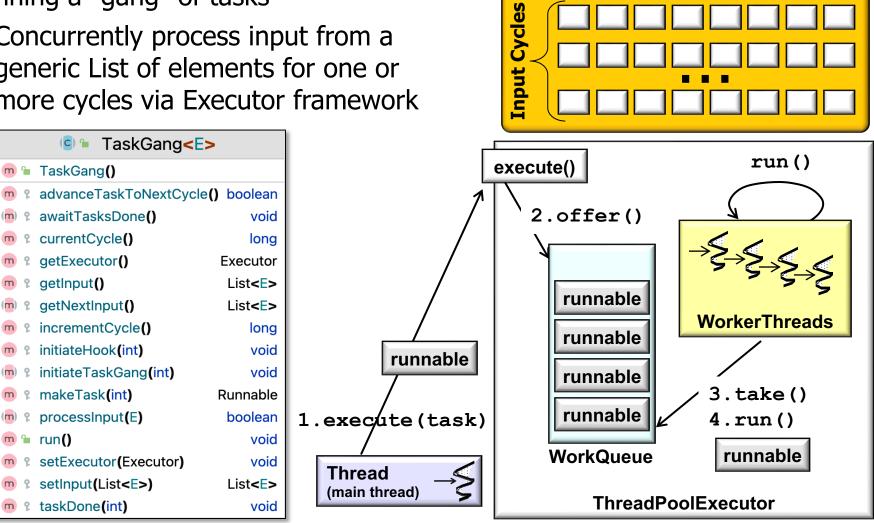
(m)

m

(m)

m

(m)



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

- 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

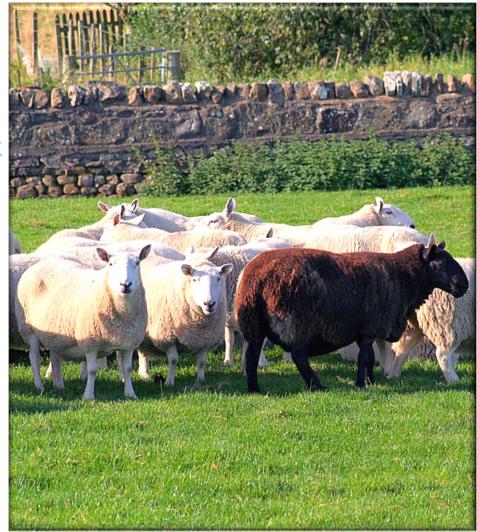


See <a href="mailto:en.wikipedia.org/wiki/Embarrassingly\_parallel">en.wikipedia.org/wiki/Embarrassingly\_parallel</a>

- 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



- 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 <a href="https://www.dre.vanderbilt.edu/~schmidt/PDF/Commonality\_Variability.pdf">www.dre.vanderbilt.edu/~schmidt/PDF/Commonality\_Variability.pdf</a>

• The framework itself supports "commonality"

	🖻 🛍 TaskGang <e></e>	
<b>m</b> 🔒	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m º	advanceTaskToNextCyc	le <b>()</b> boolean
(m) 🤉	awaitTasksDone <b>()</b>	void
<b>m ?</b>	currentCycle()	long
<b>m ?</b>	getExecutor()	Executor
m º	getInput()	List <e></e>
(m) 🤉	getNextInput()	List <e></e>
<b>m ?</b>	incrementCycle()	long
<b>m ?</b>	initiateHook(int)	void
(m) 🤉	initiateTaskGang(int)	void
m º	makeTask(int)	Runnable
(m) 🤉	processInput(E)	boolean
<b>m</b> 🛍	run <b>()</b>	void
m º	setExecutor(Executor)	void
m º	setInput(List <e>)</e>	List <e></e>
m º	taskDone(int)	void

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

🕒 ゛ TaskGang <e></e>		
m 🛍	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
<b>f</b> 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m ę	advanceTaskToNextCy	cle <b>()</b> boolean
(m) 🤉	awaitTasksDone <b>()</b>	void
m ę	currentCycle()	long
m 🤋	getExecutor()	Executor
m ę	getInput <b>()</b>	List <e></e>
(m) 🤉	getNextInput()	List <e></e>
m 🤋	incrementCycle()	long
m 🤋	initiateHook <b>(</b> int <b>)</b>	void
(m) 🧏	initiateTaskGang(int)	void
m 🤋	makeTask(int)	Runnable
(m) 🤉	processInput(E)	boolean
<b>m</b> 🛍	run <b>()</b>	void
m ę	setExecutor(Executor)	void
m ę	setInput(List <e>)</e>	List <e></e>
m १	taskDone(int)	void

#### e.g., BarrierTaskGang & ImageTaskGang

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

	🕒 🖕 TaskGang <e></e>		
<b>m</b> 🔒	TaskGang <b>()</b>		
🚹 🔒	mCurrentCycle	AtomicLong	
f 🔒	mExecutor	Executor	
f 🔒	mInput	List <e></e>	
m 🤋	advanceTaskToNextCycl	le() boolean	
(m) 🤋	awaitTasksDone <b>()</b>	void	
m 🔋	currentCycle()	long	
m 🤋	getExecutor()	Executor	
m 🤋	getInput()	List <e></e>	
(m) 🤮	getNextInput()	List <e></e>	
m º	incrementCycle()	long	
m º	initiateHook(int)	void	
(m) 🧏	initiateTaskGang(int)	void	
m º	makeTask(int)	Runnable	
(m) 🤉	processInput(E)	boolean	
<b>m</b> 🛍	run <b>()</b>	void	
m 🤋	setExecutor(Executor)	void	
m 🤋	setInput(List <e>)</e>	List <e></e>	
m 💡	taskDone(int)	void	

 The framework itself supports "commonality", e.g.

**Design** Patterns Elements of Reusable Object-Oriented Software

Erich Gamma Richard Helm Ralph Johnson

- Common data members & m signatures
- Common algorithms & control
  - run() is a template method defines the entry into a tas

e framework itself supports	🧧 🖕 TaskGang	<e></e>
mmonality", e.g.	💼 🖆 TaskGang ()	
ommon data members & method	1 mCurrentCycle	AtomicLong
ignatures	f 🗎 mExecutor	Executor
	🔒 mInput	List <e></e>
ommon algorithms & control flow	m * advanceTaskToNextCy	/cle <b>()</b> boolean
run() is a template method that	💼 🔋 awaitTasksDone ()	void
defines the entry into a task gang	m 🔋 currentCycle()	long
	m 🔋 getExecutor()	Executor
n Patterns	💼 🖇 getInput()	List <e></e>
f Reusable ented Software	💼 🔋 getNextInput()	List <e></e>
n PROFESSI	m 🔋 incrementCycle()	long
	m 🔋 initiateHook (int)	void
	💼 🔋 initiateTaskGang(int)	void
dy Booch	m 🔋 makeTask(int)	Runnable
<pre>setInput(getNextInput());</pre>	m ProcessInput(E)	boolean
initiateTaskGang	<u>m 🔓</u> run ()	void
<pre>(getInput().size());</pre>	m s setExecutor(Executor)	void
<pre>awaitTasksDone();</pre>	m 🔋 setInput(List <e>)</e>	List <e></e>
	💼 🖇 taskDone(int)	void

TackCang

See en.wikipedia.org/wiki/Template\_method\_pattern

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



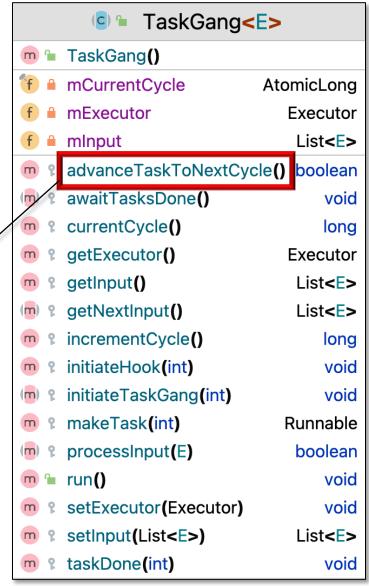
TaskGang<E> (C) 🔒 m 🖢 TaskGang() mCurrentCvcle AtomicLong mExecutor Executor mInput List<E> advanceTaskToNextCycle() boolean m % awaitTasksDone() void (m)currentCycle() lona m 2 getExecutor() Executor 9 m List<E> getNextInput() List<E> incrementCycle() long initiateHook(int) void initiateTaskGang(int) void makeTask(int) Runnable processInput(E) boolean void setExecutor(Executor) void List<E>

void

taskDone(int)

#### See en.wikipedia.org/wiki/Factory\_method\_pattern

- 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



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

- 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></e>		
m 🔒	TaskGang <b>()</b>		
🚹 🔒	mCurrentCycle	AtomicLong	
f 🔒	mExecutor	Executor	
f 🔒	mInput	List <e></e>	
m ?	advanceTaskToNext	Cycle <b>()</b> boolean	
(m) <b>°</b>	awaitTasksDone()	void	
m 🤋	currentCycle()	long	
m ę	getExecutor()	Executor	
m 🤋	getInput()	List <e></e>	
(m) 🤉	getNextInput()	List <e></e>	
m 9	incrementCycle()	long	
m 9	initiateHook(int)	void	
(m) º	initiateTaskGang(int)	void	
m 🤋	makeTask(int)	Runnable	
(m) º	processInput(E)	boolean	
m 🛍	run <b>()</b>	void	
m ę	setExecutor(Executor	r) void	
m º	setInput(List <e>)</e>	List <e></e>	
m 🤋	taskDone(int)	void	

#### Keeps track of the cycle count

 Framework must be customized to support "variability"

	🖻 🛍 TaskGang <e></e>	
<b>m</b> 🔒	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m 🤋	advanceTaskToNextCyc	cle() boolean
(m) º	awaitTasksDone <b>()</b>	void
m 🤋	currentCycle()	long
m 🤋	getExecutor()	Executor
m 🤋	getInput <b>()</b>	List <e></e>
(m) 🤉	getNextInput()	List <e></e>
m 🤋	incrementCycle()	long
m 🤋	initiateHook(int)	void
(m) º	initiateTaskGang <b>(</b> int)	void
m 🤋	makeTask(int)	Runnable
(m) 🤉	processInput(E)	boolean
<b>m</b> 🛍	run <b>()</b>	void
m 🤋	setExecutor(Executor)	void
m 🤋	setInput(List <e>)</e>	List <e></e>
m 🤋	taskDone(int)	void

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

	🕒 ゛ TaskGang <e></e>		
m 🛍	TaskGang <b>()</b>		
🚹 🔒	mCurrentCycle	AtomicLong	
f 🔒	mExecutor	Executor	
<b>f</b>	mInput	List <e></e>	
m 🤋	advanceTaskToNextCycle	e <b>() boolean</b>	
(m) 🤉	awaitTasksDone <b>()</b>	void	
m º	currentCycle()	long	
m 💡	getExecutor()	Executor	
m 🤋	getInput <b>()</b>	List <e></e>	
(m) 🤉	getNextInput()	List <e></e>	
m ያ	incrementCycle()	long	
<b>m ?</b>	initiateHook(int)	void	
(m) 🧏	initiateTaskGang(int)	void	
m 🤋	makeTask(int)	Runnable	
(m) 🤉	processInput(E)	boolean	
<b>m</b> 🛍	run <b>()</b>	void	
m 🤋	setExecutor(Executor)	void	
m 🤋	setInput(List <e>)</e>	List <e></e>	
m º	taskDone(int)	void	

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

	🕒 🛍 TaskGang <e></e>		
<b>m</b> 🐿	TaskGang <b>()</b>		
🚹 🔒	mCurrentCycle	AtomicLong	
f 🔒	mExecutor	Executor	
<b>f</b>	mInput	List <e></e>	
m 🤋	advanceTaskToNextCycl	e <b>()</b> boolean	
(m) 🤉	awaitTasksDone <b>()</b>	void	
m 💡	currentCycle()	long	
m 🤋	getExecutor()	Executor	
m 🤋	getInput <b>()</b>	List <e></e>	
( <mark>m</mark> ) ያ	getNextInput()	List <e></e>	
m 🤋	incrementCycle()	long	
m 🤋	initiateHook(int)	void	
(m) 🧣	initiateTaskGang(int)	void	
m 🤋	makeTask(int)	Runnable	
(m) 🧏	processInput(E)	boolean	
<b>m</b> 🛍	run <b>()</b>	void	
m 🔋	setExecutor(Executor)	void	
m 🤋	setInput(List <e>)</e>	List <e></e>	
m ?	taskDone(int)	void	

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

🕒 🖕 TaskGang <e></e>		
m 🔒	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
۳ ۴	advanceTaskToNextCycl	e() boolean
(m) °	awaitTasksDone()	void
m 🖁	currentCycle()	long
m ያ	getExecutor()	Executor
۳۶	getInput()	List <e></e>
(m) 🤮	getNextInput()	List <e></e>
۳۶	incrementCycle()	long
m ያ	initiateHook(int)	void
(m) 🧣	initiateTaskGang(int)	void
m 💡	makeTask(int)	Runnable
(m) 🧣	processInput(E)	boolean
<b>m</b> 🛍	run <b>()</b>	void
m 🤋	setExecutor(Executor)	void
m 🤋	setInput(List <e>)</e>	List <e></e>
m १	taskDone(int)	void

- 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>
m 🔒	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m ę	advanceTaskToNextCy	cle <b>()</b> boolean
(m) <b>°</b>	awaitTasksDone()	void
m P	currentCycle()	long
m ę	getExecutor()	Executor
m १	getInput()	List <e></e>
(m) 🤉	getNextInput()	List <e></e>
m ?	incrementCycle()	long
m P	initiateHook(int)	void
(m) 🤉	initiateTaskGang(int)	void
m P	makeTask(int)	Runnable
(m) 🤉	processInput(E)	boolean
<b>m</b> 🔒	run()	void
m ę	setExecutor(Executor)	void
m ę	setInput(List <e>)</e>	List <e></e>
<b>m</b> ?	taskDone(int)	void

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

	🕒 🛍 TaskGan	g <e></e>
<b>m</b> 🔒	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m ę	advanceTaskToNext	Cycle() boolean
(m) 🤉	awaitTasksDone <b>()</b>	void
m ę	currentCycle()	long
m 🤋	getExecutor()	Executor
m ę	getInput()	List <e></e>
(m) 🤮	getNextInput()	List <e></e>
m ę	incrementCycle()	long
m ę	initiateHook(int)	void
(m) 🧏	initiateTaskGang <b>(</b> int)	void
m ę	makeTask(int)	Runnable
(m) 🤉	processInput(E)	boolean
m 🛍	run <b>()</b>	void
m ę	setExecutor(Executo	or) void
m ę	setInput(List <e>)</e>	List <e></e>
m ?	taskDone <b>(</b> int <b>)</b>	void

- 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>
<b>m</b> 🔒	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m º	advanceTaskToNextCyc	le() boolean
(m) 🤉	awaitTasksDone <b>()</b>	void
۳۶	currentCycle()	long
m 🤋	getExecutor()	Executor
m 🤋	getInput <b>()</b>	List <e></e>
(m) 🤉	getNextInput()	List <e></e>
m 🤋	incrementCycle()	long
m 🤋	initiateHook(int)	void
(m) 🤉	initiateTaskGang(int)	void
m 🤋	makeTask(int)	Runnable
(m) 🤉	processInput(E)	boolean
<b>m</b> 🛍	run()	void
m 🤋	setExecutor(Executor)	void
m 🤋	setInput(List <e>)</e>	List <e></e>
m ?	taskDone(int)	void

- 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>
<b>m</b> 🛍	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
<b>m ?</b>	advanceTaskToNextCy	cle <b>()</b> boolean
(m) <b>°</b>	awaitTasksDone <b>()</b>	void
<b>m ?</b>	currentCycle()	long
m ę	getExecutor()	Executor
m ę	getInput()	List <e></e>
(m) º	getNextInput()	List <e></e>
<b>m ?</b>	incrementCycle()	long
<b>m ?</b>	initiateHook(int)	void
(m) <b>°</b>	initiateTaskGang(int)	void
<b>m ?</b>	makeTask(int)	Runnable
(m) <b>°</b>	processInput(E)	boolean
<b>m</b> 🔒	run <b>()</b>	void
<b>m ?</b>	setExecutor(Executor)	void
<b>m ?</b>	setInput(List <e>)</e>	List <e></e>
m 🤋	taskDone(int)	void

- 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></e>
m 🔒	TaskGang <b>()</b>	
🚹 🔒	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m º	advanceTaskToNextCy	/cle <b>()</b> boolean
(m) 🤉	awaitTasksDone <b>()</b>	void
m ę	currentCycle()	long
m º	getExecutor()	Executor
m 💡	getInput()	List <e></e>
(m) <b>°</b>	getNextInput()	List <e></e>
m º	incrementCycle()	long
m º	initiateHook(int)	void
(m) 🧣	initiateTaskGang(int)	void
m º	makeTask(int)	Runnable
(m) <b>°</b>	processInput(E)	boolean
<b>m</b> 🛍	run()	void
m º	setExecutor(Executor)	void
۳۶	setInput(List <e>)</e>	List <e></e>
<b>m</b> ?	taskDone(int)	void

CyclicBarrier must be used with Thread-per-Input model

- 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>
<b>m</b> 🔒	TaskGang <b>()</b>	
<b>f</b>	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m º	advanceTaskToNextCy	cle <b>()</b> boolean
(m) 🤉	awaitTasksDone <b>()</b>	void
<b>m ?</b>	currentCycle()	long
<b>m ?</b>	getExecutor()	Executor
m ?	getInput()	List <e></e>
(m) <b>°</b>	getNextInput()	List <e></e>
<b>m ?</b>	incrementCycle()	long
<b>m ?</b>	initiateHook(int)	void
(m) <b>°</b>	initiateTaskGang(int)	void
m የ	makeTask(int)	Runnable
(m) <b>°</b>	processInput(E)	boolean
m 🔒	run <b>()</b>	void
<b>m ?</b>	setExecutor(Executor)	void
<b>m ?</b>	setInput(List <e>)</e>	List <e></e>
m ?	taskDone(int)	void

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

- 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	g <e></e>
<b>m</b> 🛍	TaskGang <b>()</b>	
<b>f</b>	mCurrentCycle	AtomicLong
f 🔒	mExecutor	Executor
f 🔒	mInput	List <e></e>
m ę	advanceTaskToNextC	Cycle <b>()</b> boolean
(m) 🤉	awaitTasksDone <b>()</b>	void
<b>m ?</b>	currentCycle()	long
m P	getExecutor()	Executor
m ?	getInput()	List <e></e>
(m) <b>°</b>	getNextInput()	List <e></e>
m ?	incrementCycle()	long
<b>m ?</b>	initiateHook(int)	void
(m) <b>°</b>	initiateTaskGang(int)	void
m P	makeTask(int)	Runnable
(m) <b>°</b>	processInput(E)	boolean
<b>m</b> 🔒	run()	void
m 🤋	setExecutor(Executor	) void
m ę	setInput(List <e>)</e>	List <e></e>
<b>m ?</b>	taskDone(int)	void

### Walkthrough of the TaskGang Class

•••				SearchTaskGang – TaskGang.java [SearchTaskGang.main]	
SearchTaskGang	g $ angle$ src $ angle$ main $ angle$ java $ angle$ utils $ angle$	C TaskGang		🚨 🗸 📄 SearchTaskGangTest 🔻 🕨 🗯 🖏 🗸 🖉 Git: 🖌 🗸 🗡 🔘	ଚ 🔾 📀 🔈
ਰੂ 📄 Project 🗖	• 🕀 🗄 🛨 📥 🗕 -	C TaskGang	.java ×		: ()))
	TaskGang ~/Dropbox/Docu	9			A 2 ^ V
■ > ■ .grac		10	<b>-/*</b> *		base
idea		11	*	Defines a framework for spawning and running a "gang" of tasks that	~
-0-					Gradle
> grad	le	12	*	concurrently process input from a generic { <u>@link</u> <b>List</b> } of elements	dle
rests	nain	13	*	{ <u>@code</u> E} for one or more cycles.	6
Pull Requests	java	14	- */		Devic
	🖿 tasks	15 🔍		lic abstract class TaskGang <e></e>	Device Manager
PI /	C SearchTaskGangTest		pop	_	nager
🗬 build	•	16		<pre>implements Runnable {</pre>	
≥ grad grad		17		/**	No.
	ngs.gradle	18		* The input { <u>@link</u> <b>List</b> } that's processed, which can be	Notifications
> IIII External		19		* initialized via the { <u>@code</u> setInput()} method.	tions
> 🔨 Scratch	es and Consoles				
		20		*/	
		21		<pre>private volatile List<e> mInput;</e></pre>	
		22			
arks		23		/**	
Bookmarks		24			
Ä				* Executes submitted Runnable tasks in a {@link Thread} pool.	
ø		25		*/	
Structure		26		<pre>private Executor mExecutor;</pre>	
Str		27			_
		28		/**	⊆⊓ Ar
oolkii					ndroid
🖒 AWS Toolkit		29		* Keeps track of which cycle is currently active.	Android Emulato
Ø		30		*/	lator
🎙 Git 🔾	CodeWhisperer Reference Log	≡ Logcat	I Profiler	📚 Dependencies 🖽 TODO 🛛 Problems 🖾 Terminal 🜑 Services 🔮 App Inspection	
				AWS: No credentials selected 🗸 CodeWhisperer 15:23 CRLF UTF-8 4 spac	es 🥇 master 🍗

#### See <u>SearchTaskGang/src/main/java/utils/TaskGang.java</u>

# End of Overview of the TaskGang Framework