The Java ExecutorService

(Part 1)

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 powerful features provided by Java’s ExecutorService interface & related interfaces/classes

```
public interface ExecutorService extends Executor

An Executor that provides methods to manage termination and methods that can produce a Future for tracking progress of one or more asynchronous tasks.

An ExecutorService can be shut down, which will cause it to reject new tasks. Two different methods are provided for shutting down an ExecutorService. The shutdown() method will allow previously submitted tasks to execute before terminating, while the shutdownNow() method prevents waiting tasks from starting and attempts to stop currently executing tasks. Upon termination, an executor has no tasks actively executing, no tasks awaiting execution, and no new tasks can be submitted. An unused ExecutorService should be shut down to allow reclamation of its resources.
```

Interface ExecutorService

All Superinterfaces:
Executor

All Known Subinterfaces:
ScheduledExecutorService

All Known Implementing Classes:
AbstractExecutorService, ForkJoinPool, ScheduledThreadPoolExecutor, ThreadPoolExecutor
Overview of the ExecutorService Interface
Overview of the ExecutorService Interface

• Extends Executor to submit tasks, produce futures for asynchronous tasks, & manage termination of tasks & worker threads in pools

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces

```
<<Java Interface>>
Future<V>
  - cancel(boolean): boolean
  - isCancelled(): boolean
  - isDone(): boolean
  - get()
  - get(long, TimeUnit)

<<Java Interface>>
Callable<V>
  - call()

<<Java Interface>>
Runnable
  - run(): void
```
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - **Runnable**
    - A “one-way” task that does not return a result

See [docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - **Callable**
    - A “two-way” task that returns a result

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html)
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
    - A “two-way” task that returns a result
    - Typically used to run asynchronous tasks
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - **Callable**
    - A “two-way” task that returns a result
    - Typically used to run asynchronous tasks
  - Implements the *Active Object* pattern
    - Decouples method execution from method invocation for objects that each reside in their own thread(s) of control

See en.wikipedia.org/wiki/Active_object
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
- Future
  - Represents the result of an asynchronous two-way task

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html
Overview of the ExecutorService Interface

• The ExecutorService is used with other interfaces, e.g.
  • Runnable
  • Callable
  • Future
    • Represents the result of an asynchronous two-way task
    • Can be canceled & tested to see if task is done

<<Java Interface>>
Future<V>

- cancel(boolean):boolean
- isCancelled():boolean
- isDone():boolean
- get()
- get(long, TimeUnit)
Overview of the ExecutorService Interface

• The ExecutorService is used with other interfaces, e.g.
  • Runnable
  • Callable
  • Future
    • Represents the result of an asynchronous two-way task
      • Can be canceled & tested to see if task is done
    • Used to implement the *Active Object* pattern when used with a callable

See [en.wikipedia.org/wiki/Active_object](en.wikipedia.org/wiki/Active_object)
Overview of the ExecutorService Interface

• The ExecutorService is used with other interfaces, e.g.
  • Runnable
  • Callable
  • Future
    • Represents the result of an asynchronous two-way task
      • Can be canceled & tested to see if task is done
      • Used to implement the Active Object pattern when used with a callable
    • Other Future variants implement ExecutorService

<<Java Interface>>
RunnableFuture<V>
  run():void

<<Java Interface>>
Future<V>
  cancel(boolean):boolean
  isCancelled():boolean
  isDone():boolean
  get()
  get(long, TimeUnit)

<<Java Class>>
FutureTask<V>
  FutureTask(Callable<V>)
  FutureTask(Runnable,V)
  isCancelled():boolean
  isDone():boolean
  cancel(boolean):boolean
  get()
  get(long, TimeUnit)
  run():void
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
  - Future
    - Represents the result of an asynchronous two-way task
      - Can be canceled & tested to see if task is done
      - Used to implement the *Active Object* pattern when used with a callable
    - Other Future variants implement ExecutorService
      - FutureTask
        - Conveys result from thread executing a computation to thread(s) retrieving result

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html)
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
  - Future
    - Represents the result of an asynchronous two-way task
      - Can be canceled & tested to see if task is done
      - Used to implement the Active Object pattern when used with a callable
    - Other Future variants implement ExecutorService
      - FutureTask
      - RunnableFuture
        - Successful execution of the run() method causes completion of the future & allows access to its results

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/RunnableFuture.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/RunnableFuture.html)
Overview of the ExecutorService Interface

- The ExecutorService is used with other interfaces, e.g.
  - Runnable
  - Callable
  - Future
  - CompletableFuture
- A Future variant that supports dependent functions & actions that trigger upon its completion

CompletableFuture isn’t part of the Java Executor framework, but they can be used together

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html)
Overview of the ExecutorService Interface

- ExecutorService also forms the basis for key Java Executor framework subclasses

See src/share/classes/java/util/concurrent
End of Overview of Java ExecutorService (Part 1)
The Java ExecutorService
(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

- Recognize the powerful features provided by Java’s ExecutorService interface & related interfaces/classes
- Know key methods provided by the Java ExecutorService

```
<<Java Interface>>

ExecutorService

- shutdown():void
- shutdownNow():List<Runnable>
- isShutdown():boolean
- isTerminated():boolean
- awaitTermination(long, TimeUnit):boolean
- submit(Callable<T>):Future<T>
- submit(Runnable,T):Future<T>
- submit(Runnable):Future<?>
- invokeAll(Collection<? extends Callable<T>>):List<Future<T>>
- invokeAny(Collection<? extends Callable<T>>)
- invokeAny(Collection<? extends Callable<T>>,long, TimeUnit)
```
Key Methods in the ExecutorService Interface (Part 1)
Key Methods in the ExecutorService Interface

• ExecutorService can execute individual tasks

```java
public interface ExecutorService
    extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    ...
```
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return `void`

```java
public interface ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);
...
```
Key Methods in the ExecutorService Interface

• ExecutorService can execute individual tasks
  • `execute()` runs one-way tasks that return `void`
  • `submit()` runs two-way asynchronous tasks

```
public interface ExecutorService
    extends Executor {

    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    ...
```

TWO WAY
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return `void`
  - `submit()` runs two-way asynchronous tasks
- Supports “synchronous future” processing model

```java
public interface ExecutorService
    extends Executor {
    // Inherited from Executor
    void execute(Runnable command);

    <T> Future<T> submit
        (Callable<T> task);

    ...
```
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
  - `submit()` runs two-way asynchronous tasks
  - Supports “synchronous future” processing model
  - `Future.get()` blocks until task completes successfully

```java
public interface ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);
    <T> Future<T> submit (Callable<T> task);
    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can execute individual tasks
  - `execute()` runs one-way tasks that return void
  - `submit()` runs two-way asynchronous tasks
  - Supports “synchronous future” processing model
  - `Future.get()` blocks until task completes successfully
  - After which point `get()` returns the task’s result

```java
class ExecutorService extends Executor {
    // Inherited from Executor
    void execute(Runnable command);
    <T> Future<T> submit (Callable<T> task);
    ...}
```
Key Methods in the ExecutorService Interface

• ExecutorService can also execute groups of tasks

```java
public interface ExecutorService
    extends Executor {
    ...
    <T> List<Future<T>> invokeAll(
        Collection<? extends Callable<T>> tasks) ...;
    
    <T> T invokeAny(
        Collection<? extends Callable<T>> tasks) ...;
    ...
```

These methods block the calling thread until they are finished, which may be non-intuitive...

Don’t modify collection while the invokeAll() or invokeAny() methods are in progress!!!
Key Methods in the ExecutorService Interface

• ExecutorService can also execute groups of tasks

• Returns a list of futures when tasks complete

```
public interface ExecutorService extends Executor {
    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends Callable<T>> tasks) ...;
    ...
```

All the futures returned in this list are “done”!

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAll](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#invokeAll)
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
- Returns a list of futures when tasks complete
- Return the result of one successful completion

```java
public interface ExecutorService
    extends Executor {
    ...

    <T> List<Future<T>> invokeAll
        (Collection<? extends Callable<T>> tasks) ...
;

    <T> T invokeAny
        (Collection<? extends Callable<T>> tasks) ...
;

    <T> T invokeAny
        (Collection<? extends Callable<T>> tasks,
            long timeout,
            TimeUnit unit) ...
;
    ...
```

Useful for concurrent algorithms that just want the result that completes first
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
- Returns a list of futures when tasks complete
- Return the result of *one* successful completion
- Cancel uncompleted tasks

```java
class ExecutorService extends Executor {
    ...
    <T> List<Future<T>> invokeAll(
        Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny(
        Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny(
        Collection<? extends Callable<T>> tasks,
        long timeout,
        TimeUnit unit) ...;
    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can also execute groups of tasks
  - Returns a list of futures when tasks complete
  - Return the result of *one* successful completion
  - Cancel uncompleted tasks
  - Ignore other completed task results

```java
public interface ExecutorService extends Executor {
    ...
    <T> List<Future<T>> invokeAll
        (Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends Callable<T>> tasks) ...;

    <T> T invokeAny
        (Collection<? extends Callable<T>> tasks, long timeout,
         TimeUnit unit) ...;
    ...
```

Key Methods in the ExecutorService Interface (Part 2)
Key Methods in the ExecutorService Interface

- ExecutorService can initiate shutdown operations

```java
public interface ExecutorService extends Executor {
    ...
    void shutdown();
    List<Runnable> shutdownNow();
    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can initiate shutdown operations
- Perform “orderly shutdown” that completes existing tasks

```java
public interface ExecutorService extends Executor {

    ... 

    void shutdown();

    List<Runnable> shutdownNow();

    ...

```
Key Methods in the ExecutorService Interface

- ExecutorService can initiate shutdown operations
- Perform “orderly shutdown” that completes existing tasks
- But ignores new ones

```java
public interface ExecutorService extends Executor {
    ...
    void shutdown();
    List<Runnable> shutdownNow();
    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can initiate shutdown operations
- Perform “orderly shutdown” that completes existing tasks
- Attempt to stop active tasks & don’t process waiting tasks

```java
public interface ExecutorService extends Executor {
    ...
    void shutdown();
    List<Runnable> shutdownNow();
    ...
}
```
Key Methods in the ExecutorService Interface

- ExecutorService can initiate shutdown operations
  - Perform “orderly shutdown” that completes existing tasks
  - Attempt to stop active tasks & don’t process waiting tasks
  - Returns waiting tasks

```java
public interface ExecutorService extends Executor {
    ... 
    void shutdown();
    List<Runnable> shutdownNow();
    ... 
```
Key Methods in the ExecutorService Interface

• ExecutorService can query shutdown status & wait for termination to finish

```java
public interface ExecutorService extends Executor {
    ...
    boolean isShutdown();
    boolean isTerminated();
    boolean awaitTermination(long timeout, TimeUnit unit) ...;
```
Key Methods in the ExecutorService Interface

- ExecutorService can query shutdown status & wait for termination to finish
- True if Executor shut down

```java
public interface ExecutorService
    extends Executor {

    ...;

    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;

```
Key Methods in the ExecutorService Interface

- ExecutorService can query shutdown status & wait for termination to finish
  - True if Executor shut down
  - True if all tasks completed after shut down

```java
public interface ExecutorService
    extends Executor {

    ... // other methods

    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
        (long timeout,
         TimeUnit unit) ...;
```
Key Methods in the ExecutorService Interface

- ExecutorService can query shutdown status & wait for termination to finish
  - True if Executor shut down
  - True if all tasks completed after shut down
  - Blocks until all tasks complete

```java
public interface ExecutorService extends Executor {
  ...
  boolean isShutdown();
  boolean isTerminated();
  boolean awaitTermination
      (long timeout,
       TimeUnit unit) ...;
```
**Key Methods in the ExecutorService Interface**

- ExecutorService can query shutdown status & wait for termination to finish
  - True if Executor shut down
  - True if all tasks completed after shut down
  - Blocks until all tasks complete

```java
public interface ExecutorService extends Executor {

    boolean isShutdown();

    boolean isTerminated();

    boolean awaitTermination
            (long timeout,
             TimeUnit unit) ...;
```

shutdown() & awaitTermination() can be used together to provide “exit barrier” synchronization.
End of Overview of Java ExecutorService (Part 2)
The Java ExecutorService
(Part 3)

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 Lesson

- Recognize the powerful features provided by Java’s ExecutorService interface & related interfaces/classes
- Know key methods provided by the Java ExecutorService
- Understand how ThreadPoolExecutor implements the ExecutorService

<<Java Class>>

- ThreadPoolExecutor
- Worker

<<Java Method>>

- execute(Runnable): void
- shutdown(): void
- shutdownNow(): void
- isShutdown(): boolean
- isTerminating(): boolean
- isTerminated(): boolean
- awaitTermination(): void
- setThreadFactory(ExecutorService): void
- getThreadFactory(): void
- setRejectedExecutionHandler(RejectedExecutionHandler): void
- getRejectedExecutionHandler(): void
- setCorePoolSize(int): void
- getCorePoolSize(): int
- prestartCoreThread(): void
- prestartAllCoreThreads(): void
- allowsCoreThreadTimeOut(): boolean
- allowCoreThreadTimeOut(): void
- setMaximumPoolSize(int): void
- getMaximumPoolSize(): int
- setKeepAliveTime(TimeUnit): void
- getKeepAliveTime(TimeUnit): long
- getQueue(): void
- remove(Runnable): void
- purge(): void
- getPoolSize(): int
- getActiveCount(): int
- getLargestPoolSize(): int
- getTaskCount(): long
- getCompletedTaskCount(): long
- toString(): void
Overview of the Java ThreadPoolExecutor
Overview of the Java ThreadPoolExecutor

- ThreadPoolExecutor implements the ExecutorService interface
- Indirectly via the AbstractExecutorService super class
Overview of the Java ThreadPoolExecutor

• Executes each submitted task using a worker thread provided by a pool

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html
Overview of the Java ThreadPoolExecutor

- Executes each submitted task using a worker thread provided by a pool.
- Blocking queue passes tasks from clients to worker threads.

```
<<Java Class>>

@ThreadPoolExecutor

- threadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>)
- threadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory)
  - execute(Runnable): void
  - shutdown(): void
  - shutdownNow():
  - isShutdown(): boolean
  - isTerminating(): boolean
  - isTerminated(): boolean
  - awaitTermination(long, TimeUnit): boolean
  - setThreadFactory(ThreadFactory): void
  - getThreadFactory():
  - setRejectedExecutionHandler(RejectedExecutionHandler): void
  - getRejectedExecutionHandler():
  - setCorePoolSize(int): void
  - getCorePoolSize(): int
  - prestartCoreThread(): boolean
  - prestartAllCoreThreads(): int
  - allowCoreThreadTimeout(): boolean
  - allowCoreThreadTimeout(boolean): void
  - setMaximumPoolSize(int): void
  - getMaximumPoolSize(): int
  - setKeepAliveTime(long, TimeUnit): void
  - getKeepAliveTime(TimeUnit): long
  - getQueue():
  - remove(Runnable): boolean
  - purge(): void
  - getPoolSize(): int
  - getActiveCount(): int
  - getLargestPoolSize(): int
  - getTaskCount(): long
  - getCompletedTaskCount(): long
  - toString()
```

![Work Queue Diagram]
Overview of the Java ThreadPoolExecutor

- Blocking queue can be strategized
- Direct handoff

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/SynchronousQueue.html
Overview of the Java ThreadPoolExecutor

- Blocking queue can be strategized
- Direct handoff
  - Pros – Avoids deadlock when internal dependencies

```
<<Java Class>>
@ThreadPoolExecutor

- ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>)
- ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory)
- execute(Runnable): void
- shutdown(): void
- shutdownNow()
- isShutdown(): boolean
- isTerminating(): boolean
- isTerminated(): boolean
- awaitTermination(long, TimeUnit): boolean
- setThreadFactory(ThreadFactory): void
- getThreadFactory()
- setRejectedExecutionHandler(RejectedExecutionHandler): void
- getRejectedExecutionHandler()
- setCorePoolSize(int): void
- getCorePoolSize(): int
- prestartCoreThread(): boolean
- prestartAllCoreThreads(): int
- allowsCoreThreadTimeOut(): boolean
- allowCoreThreadTimeOut(boolean): void
- setMaximumPoolSize(int): void
- getMaximumPoolSize(): int
- setKeepAliveTime(long, TimeUnit): void
- getKeepAliveTime(TimeUnit): long
- getQueue()
- remove(Runnable): boolean
- purge(): void
- getPoolSize(): int
- getActiveCount(): int
- getLargestPoolSize(): int
- getTaskCount(): long
- getCompletedTaskCount(): long
- toString()
```
Overview of the Java ThreadPoolExecutor

• Blocking queue can be strategized
• Direct handoff
  • Pros – Avoids deadlock when internal dependencies
  • Cons – Unlimited threads

```java
<<Java Class>>
 ThreadPoolExecutor

public ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQueue, ThreadFactory threadFactory, RejectedExecutionHandler handler)

execute(Runnable): void
shutdown(): void
shutdownNow()
isShutdown(): boolean
isTerminating(): boolean
isTerminated(): boolean
awaitTermination(long, TimeUnit): boolean
setThreadFactory(ThreadFactory): void
getThreadPool()
setRejectedExecutionHandler(RejectedExecutionHandler): void
getRejectedExecutionHandler()
setCorePoolSize(int): void
getCorePoolSize(): int
prestartCoreThread(): boolean
prestartAllCoreThreads(): int
allowsCoreThreadTimeOut(): boolean
allowCoreThreadTimeOut(boolean): void
setMaximumPoolSize(int): void
getMaximumPoolSize(): int
setKeepAliveTime(long, TimeUnit): void
getKeepAliveTime(TimeUnit): long
getQueue()
remove(Runnable): boolean
purge(): void
getPoolSize(): int
getActiveCount(): int
getLargestPoolSize(): int
getTaskCount(): long
getCompletedTaskCount(): long
toString()
```
Overview of the Java ThreadPoolExecutor

- Blocking queue can be strategized
- Direct handoff
- Unbounded queues

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingQueue.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/LinkedBlockingQueue.html)
Overview of the Java ThreadPoolExecutor

• Blocking queue can be strategized
  • Direct handoff
• Unbounded queues
  • Pros – Smooths bursty requests

<<Java Class>>

ThreadExecutor

execute(Runnable): void
shutdown(): void
shutdownNow()
isShutdown(): boolean
isTerminating(): boolean
isTerminated(): boolean
awaitTermination(long, TimeUnit): boolean
setThreadFactory(ThreadFactory): void
getThreadFactory()
setRejectedExecutionHandler(RejectedExecutionHandler): void
getRejectedExecutionHandler()
setCorePoolSize(int): void
getCorePoolSize(): int
prestartCoreThread(): boolean
prestartAllCoreThreads(): int
allowsCoreThreadTimeOut(): boolean
allowCoreThreadTimeOut(boolean): void
setMaximumPoolSize(int): void
getMaximumPoolSize(): int
setKeepAliveTime(long, TimeUnit): void
getKeepAliveTime(TimeUnit): long
getQueue()
remove(Runnable): boolean
purge(): void
getPoolSize(): int
getActiveCount(): int
getLargestPoolSize(): int
getTaskCount(): long
getCompletedTaskCount(): long
toString()
Overview of the Java ThreadPoolExecutor

- Blocking queue can be strategized
- Direct handoff
- Unbounded queues
- Pros – Smooths bursty requests
- Cons – Unlimited resources

<<Java Class>>

```java
public class ThreadPoolExecutor {
    // Constructor
    public ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue<Runnable> workQueue)

    // Methods
    public void execute(Runnable command)
    public void shutdown()
    public void shutdownNow()
    public boolean isShutdown()
    public boolean isTerminating()
    public boolean isTerminated()
    public boolean awaitTermination(long timeout, TimeUnit unit)
    public void setThreadFactory(ThreadFactory threadFactory)
    public ThreadFactory getThreadFactory()
    public void setRejectedExecutionHandler(RejectedExecutionHandler handler)
    public RejectedExecutionHandler getRejectedExecutionHandler()
    public int getCorePoolSize()
    public void setCorePoolSize(int corePoolSize)
    public boolean isPrestartCoreThreadEnabled()
    public int prestartAllCoreThreads()
    public boolean allowsCoreThreadTimeOut()
    public void setMaximumPoolSize(int maximumPoolSize)
    public int getMaximumPoolSize()
    public void setKeepAliveTime(long keepAliveTime, TimeUnit unit)
    public long getKeepAliveTime(TimeUnit unit)
    public BlockingQueue<Runnable> getQueue()
    public boolean remove(Runnable command)
    public void purge()
    public int getPoolSize()
    public int getActiveCount()
    public int getLargestPoolSize()
    public long getTaskCount()
    public long getCompletedTaskCount()
    public String toString()
}
```
Overview of the Java ThreadPoolExecutor

- Blocking queue can be strategized
- Direct handoff
- Unbounded queues
- Bounded queues

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ArrayBlockingQueue.html
Overview of the Java ThreadPoolExecutor

- Blocking queue can be strategized
  - Direct handoff
  - Unbounded queues
  - Bounded queues
- Pros – Limits resource utilization
Overview of the Java ThreadPoolExecutor

- Blocking queue can be strategized
  - Direct handoff
  - Unbounded queues
  - Bounded queues
    - Pros – Limits resource utilization
    - Cons – Hard to tune & may incur deadlock

```java
<<Java Class>>
@ ThreadPoolExecutor

- ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>)
- ThreadPoolExecutor(int, int, long, TimeUnit, BlockingQueue<Runnable>, ThreadFactory)
  - execute(Runnable): void
  - shutdown(): void
  - shutdownNow()
  - isShutdown(): boolean
  - isTerminating(): boolean
  - isTerminated(): boolean
  - awaitTermination(long, TimeUnit): boolean
  - setThreadFactory(ThreadFactory): void
  - getThreadFactory()
  - setRejectedExecutionHandler(RejectedExecutionHandler): void
  - getRejectedExecutionHandler()
  - setCorePoolSize(int): void
  - getCorePoolSize(): int
  - prestartCoreThread(): boolean
  - prestartAllCoreThreads(): int
  - allowsCoreThreadTimeOut(): boolean
  - allowCoreThreadTimeOut(boolean): void
  - setMaximumPoolSize(int): void
  - getMaximumPoolSize(): int
  - setKeepAliveTime(long, TimeUnit): void
  - getKeepAliveTime(TimeUnit): long
  - getQueue()
  - remove(Runnable): boolean
  - purge(): void
  - getPoolSize(): int
  - getActiveCount(): int
  - getLargestPoolSize(): int
  - getTaskCount(): long
  - getCompletedTaskCount(): long
  - toString()
```
End of The JavaExecutor Service (Part 3)