**Formas de Escalonamento de Threads em Java**Exemplificando o uso das classes

**ExecutorService,**

**ScheduledExecutorService**

para diferentes formas de escalonamento.

O escalonamento de threads pode ser explicado em analogia ao número de pessoas que cabem num pedalinho num lago, em geral duas pessoas, (o pool de threads) que tem de atender a um número maior de pessoas, que podem estar numa fila e que disputam o pedalinho (equivalente ao processador), que processa um passeio pelas águas de um lago. Uma pessoa que administre os uso dos pedalinhos pelas pessoas, seria o ***Scheduler***. Veja o exemplo, que pode ser executado.

Neste exemplo, ***WorkThreads*** são threads que são executadas como escalonadas em um pool de threads, ou seja são as threads que estão no pool de threads.

O pool de threads define quantas threads são escalonadas pelo processador, de forma limitada ou ilimitada, podendo existir um número maior de *threads requisitando execução,* do que o tamanho do pool de threads definido para escalonar.

**public** **class** WorkerThread **implements** [Runnable](http://www.google.com/search?hl=en&q=allinurl%3Arunnable+java.sun.com&btnI=I%27m%20Feeling%20Lucky)
{
        **private** [String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky)  threadName   = **null**;

        **public** WorkerThread([String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky) threadName)
        {
                **this**.threadName = threadName;
        }

        **public** **void** run()
        {
                [System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println(**this**.threadName + " started...");
                **try**
                {
                        [Thread](http://www.google.com/search?hl=en&q=allinurl%3Athread+java.sun.com&btnI=I%27m%20Feeling%20Lucky).sleep(5000);
                }
                **catch** ([InterruptedException](http://www.google.com/search?hl=en&q=allinurl%3Ainterruptedexception+java.sun.com&btnI=I%27m%20Feeling%20Lucky) e)
                {
                        e.printStackTrace();
                }
                [System](http://www.google.com/search?hl=en&q=allinurl%3Asystem+java.sun.com&btnI=I%27m%20Feeling%20Lucky).out.println(**this**.threadName + " ended...");
        }
}
 **import** java.util.concurrent.ExecutorService;
**import** java.util.concurrent.Executors;
**import** java.util.concurrent.ScheduledExecutorService; **import** java.util.concurrent.ScheduledThreadPoolExecutor;
**import** java.util.concurrent.TimeUnit;

**public** **class** Main
{
        **public** **static** **void** main([String](http://www.google.com/search?hl=en&q=allinurl%3Astring+java.sun.com&btnI=I%27m%20Feeling%20Lucky) args[] )
        {
               /\* *define o tamanho do pool* **\*/** int PoolSize = 2;

 */\*\*
  \** ***Caso 1*** *- Esta classe mostra o uso de* ***ExecutorService*** *\*/*
 */\* Um pool de threads de tamanho PoolSize é criado com ExecutorService\*/* ExecuteService te = Executors.newFixedThreadPool(PoolSize);

 */ \* Threads são utilizadas pelo objeto te ( threadExecutor) para executar os
 \* Runnables (ou seja, o código no método run() das threads,
 \* que serão executadas nas threads (WorkThreads) criadas pelo ExecutorService).
 \* Se o método execute for chamado e todas as threads em ExecutorService estiverem
 \* em uso (caso em que existem mais threads requisitando execução do que threads
 \* no pool), a thread será colocada numa fila e atribuída no lugar da primeira thread
 \* que terminar.
 \*/* te.execute(new WorkThread( “WorkThread-executando-imediatamente-em-
 timesliced-default-do processador”) ); *// inicia a execução da WorkThread*

 /\* -------------------------------------------------------------------------------------------------------
 \*/
 */\*\** ***\* Caso 2*** *- Esta classe mostra o uso de* ***ScheduledExecutorService*** *\*/*

 /\* *Um pool de Threads de tamanho* PoolSize *é criado com* ScheduledExecutorService. *\*/* ScheduledExecutorService ste = Executors.newScheduledThreadPool(PoolSize);

 */\*
 \* Threads são os objeto* ste ( scheduledThreadExecutor )
 *\* para executar os Runnables (ou seja, os códigos nos métodos run() de classes
 \* que implementam a interface Runnable para implementação de threads em Java,
 \* Os códigos do método run() serão executados nas threads criadas pelo
 \* ScheduledExecutorService.
 \*/*

 */\*
                 \* Esta executará uma thread requerendo execução, continuamente de 5 em 5
 \* milisegundos, para a primeira WorkerThread iniciar o ciclo de execução.
 \* Neste caso, se a primeira WorkThread é
 \* completada ou não, a segunda WorkThread iniciará exatamente após 5
 segundos, portanto, chamada de escalonamento em taxa fixa (*schedule at
 FixedRate*).
 \* Isto continua até que 'n' threads, no todo, sejam executadas.
 \*/*

 ste.schedule (new WorkThread("WorkerThread-Executando-schedule",
 ou
 ste.schedule(new WorkThread(), 5, TimeUnit.MILLISECONDS);

 */\*
                 \* Esta executará uma thread requerendo execução, continuamente de 5 em 5
 \* milisegundos, com um atraso inicial de 10 milisegundos, para a primeira
 \* WorkerThread iniciar o ciclo de execução. Neste caso, se a primeira WorkThread é
 \* completada ou não, a segunda WorkThread iniciará exatamente após 5 segundos,
 \* portanto, chamada de escalonamento em taxa fixa (*schedule at FixedRate*).
 \* Isto continua até que 'n' threads, no todo, sejam executadas.*

 *\*Se o método* scheduleAtFixedRate *for chamado e todas as threads em
 \** ScheduledExecutorService *estiverem em uso(caso em que existem mais threads
 \* requisitando execução do que threads no pool),
 \* o Runnable será colocado numa fila e atribuido à primeira thread que terminar.
 \*/*
ste.scheduleAtFixedRate(**new** WorkerThread("WorkerThread-Executando-
 scheduled-At-Fixed-Rate"), 10, 5, TimeUnit.MILISECONDS);

 ste.shutdown();

 }

------------------------------------------------------------------------------------------------------------------------------

<http://tutorials.jenkov.com/java-util-concurrent/scheduledexecutorservice.html#scheduledexecutorservice-example>

-----------------------------------------------------------------------------------------------------------------------------