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

**ExecutorService,**

**ScheduledExecutorService**

**ScheduledThreadPoolExecutor**

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.

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);

*/\* Um pool de Threads de tamanho PoolSize fixado é criado com ExecutorService.   
 \* 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*** *\*/*

ScheduledExecutorService ste = Executors.newScheduledThreadPool(PoolSize);

*/\* Um pool de Threads de tamanho* PoolSize *é criado com* ScheduledExecutorService. *\* Threads são utilizadas pelo 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. 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.   
 \*/  
  
 /\*  
                 \* 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.   
 \*/*  
ste.scheduleAtFixedRate(**new** WorkerThread("WorkerThread-Executando-  
 scheduled-At-Fixed-Rate"), 10, 5, TimeUnit.MILISECONDS);  
  
 /\* -------------------------------------------------------------------------------------------------------------   
 \*/

*/\*\*  
  \** ***Caso 3*** *- Esta classe mostra o uso de* ***ScheduledThreadPoolExecutor*** *\*/*  
 ScheduledThreadPoolExecutor stpe = **new** ScheduledThreadPoolExecutor(PoolSize);  
  
                */\*  
                 \* Esta instrução executará uma* ***WorkerThread*** *imediatamente.  
                 \*/*  
                stpe.execute(**new** WorkerThread("WorkerThread-executando-Imediatamente"));  
  
                */\*  
                 \* Esta instrução executará uma* ***WorkerThread*** *somente uma vez, após 10 segundos.  
                 \*/*  
                stpe.schedule( **new** WorkerThread("WorkerThread-Escalonada-Após-10-segundos"),   
 10, TimeUnit.SECONDS );  
   
                */\*  
                 \* Esta executará uma WorkerThread continuamente de 5 em 5 segundos, com um   
 \* atraso inicial de 10 segundos 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 fixed rate****). Isto continua até que 'n'   
 \* threads sejam executadas.  
                 \*/*  
                stpe.scheduleAtFixedRate(**new** WorkerThread("WorkerThread-Executando-At-Fixed-  
 Rate"), 10, 5, TimeUnit.SECONDS);  
  
                */\*  
                 \* Esta executará uma WorkerThread continuamente com um atraso inicial de 10   
 \* segundos para a primeira WorkerThread iniciar o ciclo de execução.Uma vez a   
 \* primeira execução de thread complete, então um atraso de 5 segundos é   
 \* introduzido de modo que a próxima WorkerThread inicie o ciclo de execução.   
 \* Isto continua até que as 'n' threads sejam executadas. Isto é chamado escalonar   
 \* cada thread com um atraso fixo.                   
 \*/*  
                stpe.scheduleWithFixedDelay(**new** WorkerThread("WorkerThread-executando-com-  
 Fixed-Delay"), 10, 5, TimeUnit.SECONDS);  
        }  
}  
  
Executando teremos a saída para o Caso 3:   
 (ScheduledThreadPoolExecutor)

WorkerThread-Running-Immediately started...  
WorkerThread-Running-Immediately ended...  
WorkerThread-Scheduled-After-10-seconds started...  
WorkerThread-Running-At-Fixed-Rate started...  
WorkerThread-Running-At-Fixed-Rate ended...  
WorkerThread-Scheduled-After-10-seconds ended...  
WorkerThread-Running-With-Fixed-Delay started...  
WorkerThread-Running-At-Fixed-Rate started...  
WorkerThread-Running-At-Fixed-Rate ended...  
WorkerThread-Running-With-Fixed-Delay ended...  
WorkerThread-Running-At-Fixed-Rate started...  
WorkerThread-Running-At-Fixed-Rate ended...  
WorkerThread-Running-At-Fixed-Rate started...  
WorkerThread-Running-With-Fixed-Delay started...  
------------------------------------------------------------------------------------------------------------------------------