Threads, Gerenciamento de Threads

Pool de Threads, Grupo de Threads
Variáveis Locais à Threads

Tarefa para Adormecimento e Despertar de Threads

```
public class ThreadSleep extends Thread {
    private long tempo = 0;
    public ThreadSleep(long tempo) { this.tempo = tempo; } // Construtor
    public void run() { // Código da Thread
        System.out.println(getName() +" vai dormir por "+ tempo +" ms.");
        try {
                sleep(tempo);
                 System.out.println(getName() + " acordou.");
         } catch (InterruptedException e) { e.printStackTrace(); }
    }
    public static void main(String args[]) {
          for (int i=0; i<10; i++)
          // Cria e executa as Threads
          new ThreadSleep((long)(Math.random()*10000)).start();
    }
}
```

Gerenciamento de Threads

- Executores gerenciam pools de threads.
  - Interfaces **Executor** e **ExecutorService** (que estende a primeira; é mais completa)
- São criados através da classe **Executors**:
  - **SingleThreadExecutor** usa uma thread para:
    executar atividades (tarefas) seqüencialmente.
  - **FixedThreadPool** usa um pool de threads
    de tamanho fixado para executar atividades (tarefas).
  - **CachedThreadPool** cria threads sob demanda, para a execução de atividades (tarefas).

Método Execute

- `Execute` é um método herdado da interface `Executor`.

- `java.util.concurrent.Executor`

Exemplo de uso de Executor

```
import java.util.concurrent.*;
public class ExecutorTest {
   public static void main(String args[]) {
      ExecutorService exec = Executors.newSingleThreadExecutor();
      // ExecutorService exec = Executors.newFixedThreadPool(5);
      // ExecutorService exec = Executors.newCachedThreadPool();
      for (int i=0; i<10; i++) {
           // Cria e executa as threads
            exec.execute(new ThreadSleep((long)(Math.random()*10000)));
       }
       // Encerra o executor assim que as threads terminarem
       exec.shutdown();
   }
}
```

Escalonamento de atividades

- Uma interface **ScheduledExecutorService** permite escalonar a execução de atividades (tarefas), definindo um atraso para início da atividade e/ou um período de repetição entre execuções.
  - **newSingleThreadScheduledExecutor** usa uma thread para todas as atividades (tarefas) escalonadas.
  - **ScheduledThreadPool** cria um pool de threads para executar as atividades (tarefas).

A Interface ScheduledExecutorService

- `java.util.concurrent`
- **Interface** `ScheduledExecutorService`
- **Subinterfaces:**
  `Executor,ExecutorService`

  Implementando Classes:
  `ScheduledThreadPoolExecutor`

A Interface ScheduledExecutorService

- `public interface` **ScheduledExecutorService** `extends ExecutorService`

- Um `ExecutorService` que pode escalonar comandos para executarem após um dado atraso, ou executarem periodicamente.

O Método Schedule

- O método `schedule` aceita atrasos relativos e períodos como argumentos, não tempos absolutos ou datas.

O Método Schedule

- O método `schedule` cria tarefas com vários atrasos e retorna um objeto-tarefa que pode ser usado para cancelar ou verificar execução.

Escalonamento com ScheduledExecutor

```
import java.util.concurrent.*;
public class ScheduleThread extends Thread {
        public void run () { System.out.println(getName() + " executada.");
        }
        public static void main(String args[]) {
                ScheduledExecutorService exec = Executors.newSingleThreadScheduledExecutor();
                for (int i=0; i<10; i++) {
                        long tempo = (long) (Math.random()*10000);
                        System.out.println("Thread-" + i + " será executada em "+ tempo + "ms.");
                        // Escalona a execução da thread
                        exec.schedule(new ScheduleThread(), tempo, TimeUnit.MILLISECONDS);
                }
                exec.shutdown();
        }
}
```

O Método `scheduleAtFixedRate`

- Os métodos `scheduleAtFixedRate` e `scheduleWithFixedDelay` criam e executam tarefas que rodam periodicamente, até serem canceladas.

- `scheduleWithFixedDelay` (não usado aqui)

Exemplo com await.Termination

- http://www.java2s.com/Code/JavaAPI/java.util.concurrent/ExecutorServiceawaitTerminationlongtimeoutTimeUnitunit.htm

Execução periódica com ScheduledExecutor

```
import java.util.concurrent.*;
public class CountdownThread extends Thread {
    private static long tempo = 0, cont = 10, espera = 5, intervalo = 1;
    public CountdownThread(long tempo) { this.tempo = tempo; }
    public void run () {
        System.out.println(tempo +"segundos para o encerramento.");
         tempo--; }
    public static void main(String args[]) {
         ScheduledExecutorService exec =
                Executors.newSingleThreadScheduledExecutor();
         exec.scheduleAtFixedRate(new CountdownThread(cont),
                 espera, intervalo, TimeUnit.SECONDS);
         try { exec.awaitTermination(cont+espera, TimeUnit.SECONDS);
         } catch (InterruptedException ie) { ie.printStackTrace(); }
         exec.shutdown();
    }
}
```

Grupos de Threads

- A classe `ThreadGroup` define um **grupo de threads que são controladas conjuntamente**.
- **O grupo de thread é definido usando o construtor** `Thread(ThreadGroup, ... ).`
- // Cria grupo de Threads

```
ThreadGroup myGroup = new ThreadGroup("MyThreads");
...
// Cria Threads e as insere no grupo
      Thread myThread1 = new MyThread(myGroup,"MyThread"+i);
      Thread myThread2 = new MyThread(myGroup,"MyThread"+i);
 ...
// Interrompe todas as Threads do grupo
      group.interrupt();
      ...
```

Variáveis locais em Threads

- A classe `ThreadLocal` permite **criar variáveis locais para threads** (ou seja, variáveis que têm valores distintos para cada thread).

- // Cria variável local

```
static ThreadLocal valorLocal = new ThreadLocal();
 …
// Define o valor da variável local para uma thread
valorLocal.set(new Integer(10));
// Obtém o valor de valor correspondente a essa thread
int valor = ((Integer) valorLocal.get()).intValue();
// valor = 10
…
// Outra thread pode acessar a mesma variável e obter outro valor
int valor = ((Integer) valorLocal.get()).intValue();
// valor = 0
```

Exemplo usando ThreadLocal

- http://www.antoniorafael.eti.br/index.php?option=com_content&view=article&id=58:usando-threadlocal&catid=37:conceitos-avancados&Itemid=56