Programação Concorrente em Java

Aula 04

Java Threads

Introdução

- Java é uma linguagem de alto nível com suporte nativo a threads.
- Threads são como qualquer outro objeto em Java.
- Toda thread é uma instância da classe java.lang.Thread, ou instâncias de suas subclasses.
- Além de serem objetos, Threads em Java também podem executar código.

Introdução

- Criando Threads

```
Thread thread = new Thread();
```

- Iniciando Threads

```
thread.start();
```

Introdução

- O código anterior não executa nenhuma computação aplicável. Apenas inicia uma thread que “morre” logo depois, sem ter feito nada.
- Existem duas formas de especificar o código que será executado:
  - Criar uma subclasse de Thread e reimplementar (sobrescrever) o método **run**.
  - Passar um objeto que implementa java.lang.Runnable para o construtor de uma Thread e depois executar essa thread.

Subclasse de Thread

- A forma mais tradicional.

```
public class MyThread extends Thread {
    public void run(){
       System.out.println("MyThread running");
    }
  }


 MyThread myThread = new MyThread();
   myTread.start();
```

Subclasse de Thread

- Com classe anônima

```
Thread thread = new Thread(){
    public void run(){
      System.out.println("Thread Running");
    }
  }

thread.start();
```

Implementar a interface Runnable

- Implemente a interface Runnable.
- Envia a sua classe para o construtor de uma Thread.
- A thread irá chamar o método run da sua classe.

```
public interface Runnable() {
    public void run();
}
```

Implementar a interface Runnable

- Existem 3 formas de implementar a interface Runnable:
  - Criar uma classe Java que simplemente implementa a interface;
  - Criar uma classe anônima que implementa a interface;
  - Criar um Java Lambda que implementa a interface.

Criar uma classe

```
public class MyRunnable implements Runnable {
    public void run(){
       System.out.println("MyRunnable running");
    }
  }
```

```
Thread myThread = new Thread(new MyRunnable());
myThread.start();
```

Criar uma classe anônima

```
  Runnable myRunnable =
      new Runnable(){
          public void run(){
              System.out.println("Runnable running");
          }
      }


Thread myThread = new Thread(myRunnable);
myThread.start();
```

Java Lambda

- Fazendo uso do paradigma funcional de Java.

```
Runnable runnable =
        () -> { System.out.println("Lambda Runnable running"); };
```

Subclasse ou Runnable?

- Não existe um método melhor.
- Implementar a interface Runnable é mais “elegante” pois o uso de interface aumenta a coesão de código e diminui o acoplamento.
- O programado deve escolher a abordagem que se adeque melhor ao seu problema.

Thread names

```
Thread thread = new Thread("New Thread") {
   public void run(){
     System.out.println("run by: " + getName());
   }
};
thread.start();
System.out.println(thread.getName())
```

```
Thread thread = new Thread("New Thread") {
   public void run(){
     System.out.println("run by: " + getName());
   }
};
thread.start();
System.out.println(thread.getName())
```

Thread.currentThread()

- Através deste método é possível conseguir uma referência para a Thread que está executando o código que fez a chamada.

```
Thread thread = Thread.currentThread();

String threadName = thread.currentThread().getName();
```

Exemplo

```
public class ThreadExample {
  public static void main(String[] args){
    System.out.println(Thread.currentThread().getName());
    for(int i=0; i<10; i++){
      new Thread("" + i){
        public void run(){
          System.out.println("Thread: " + getName() + " running");
        }//run
      }.start();//new Thread
    }//for
  }//main
}//class
```

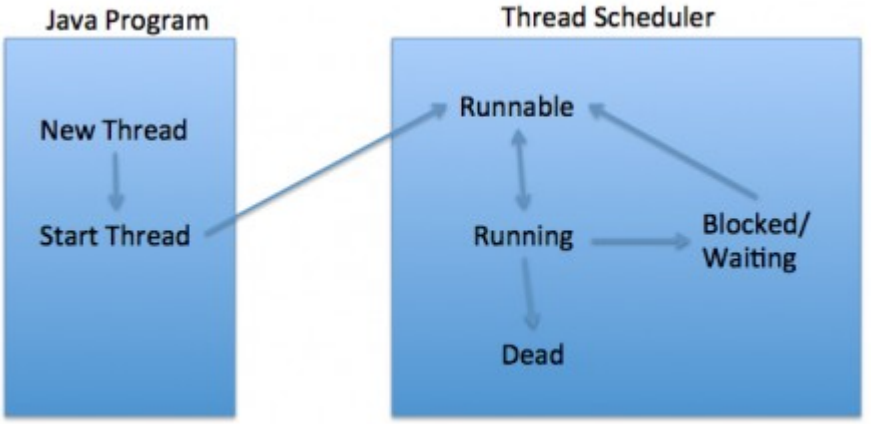
Pause a Thread

- Thread.sleep

```
try {
    Thread.sleep(10L * 1000L);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

Thread life-cycle

- O ciclo de vida de uma thread em java pode ser visto no seguinte diagrama:



Thread life-cycle

- **New**: quando a Thread é criada usando o operador new. A thread não está viva.
- **Runnable**: Quando chamamos método start, a Thread passa para o estado **runnable,** ou seja, a sua execução depende do scheduler do Sistema Operacional.
- **Running**: O scheduler pega uma das threads que está em estado runnable e a executa. Ou seja, a CPU passa de fato a executar o código **run**. Daí, ela pode partir para:
  - Dead
  - Blocked/Waiting

Thread life-cycle

- **Blocked/Waiting**: uma thread pode estar esperando outras acabarem de executar usando o método join() ou pode estar esperando algum recurso ficar disponível (produtor-consumidor). Uma vez terminado o bloqueio, a thread volta para o pool de threads com o estado runnable.
- **Dead**: a thread acabou sua execução. O objeto não está mais vivo e está pronto para ser recolhido pelo GC.

Stop a Thread

- Usando o método stop (deprecated)
- Não há garantia em qual estado a thread será parada.
- Todos os objetos que dependem da thread estarão em um estado impossível de se dizer.
- Se outras threads na sua aplicação usam os mesmos objetos compartilhados que a thread que parou, então a aplicação pode entrar em um estado inconsistente.
- **A solução é o próprio programador implementar a sua versão do stop.**

Stop a Thread - Exemplo

```
public class MyRunnable implements Runnable {
    private boolean doStop = false;
    public synchronized void doStop() {
        this.doStop = true;
    }
    private synchronized boolean keepRunning() {
        return this.doStop == false;
    }
    @Override
    public void run() {
        while(keepRunning()) {
            // keep doing what this thread should do.
            System.out.println("Running");
            try {
                Thread.sleep(3L * 1000L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Stop a Thread - Exemplo

```
public class MyRunnableMain {
    public static void main(String[] args) {
        MyRunnable myRunnable = new MyRunnable();
        Thread thread = new Thread(myRunnable);
        thread.start();
        try {
            Thread.sleep(10L * 1000L);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        myRunnable.doStop();
    }
}
```

Referências

- http://tutorials.jenkov.com/java-concurrency/creating-and-starting-threads.html