Thread

Para entender o funcionamento de uma thread é necessário analisar, inicialmente, um processo. A maioria dos sistemas de hoje são baseados em computadores com apenas um processador que executam várias tarefas simultâneas. Ou seja, vários processos que compartilham do uso da CPU tomando certas fatias de tempo.

- O inicio de cada processo é bastante custoso, em termos de uso de memória e desempenho, e o mecanismo de troca de mensagens entre os processos é mais complexo e mais lento, se comparado a um único programa acessando seus próprios dados. Uma solução encontrada foi o uso de threads, (também conhecidas por linhas de execução).

Como a thread simula o multiprocessamento, temos a impressão que, se criarmos 10 threads, por exemplo, é como se tivéssemos 10 processadores dada a velocidade com que ela é executada.
Quando queremos executar várias coisas ao mesmo tempo, entra em cena o **escalonador de thread**.

Thread

O escalonador (scheduler), sabendo que apenas uma coisa pode ser executada de cada vez, pega todas as threads que precisam ser executadas e faz o processador ficar alternando a execução de cada uma delas. A idéia é executar um pouco de cada thread e fazer essa troca tão rapidamente que a impressão que fica é que as coisas estão sendo feitas ao mesmo tempo.

Thread

Há duas possibilidades de criar a classe que você deseja que seja “um segmento separado de execução”:

Na primeira forma a sua classe deve implementar a interface Runnable.
Na segunda forma, você pode estender diretamente a classe Thread.

Vamos analisar a primeira possibilidade, a de implementar a interface Runnable. Precisamos implementar o método run() obrigatório para classes que implementam a interface Runnable. Essa interface possui esse único método. É nele que codificamos o que a Thread irá fazer.

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

Agora que já implementamos o método run() da interface Runnable, agora temos que passá-la para a classe que irá executá-la.

Thread

Agora que já criamos nossa classe que implementa Runnable ("MyRunnable"), precisamos passar um objeto dessa classe como argumento para o construtor da classe Thread.

```
class ThreadTester {

    public static void main(String[] args) {
            Runnable threadJob = new MyRunnable();
            Thread myThread = new Thread(threadJob);
            Thread myThread2 = new Thread(threadJob);
            myThread.start();
            myThread2.start();
        }
    }
```

Acima, passamos um objeto de MyRunnable representado pela variável threadJob para o construtor de um Thread. (new Thread (threadJob)). Criamos a segunda Thread e executamos as duas com o método start(). Os dois processos rodam quase que simultaneamente.

Thread

A segunda forma de criar uma Thread é estender a classe Thread e reescrever o método run().
A vantagem é que fica um pouco mais fácil no momento da criação do objeto, como mostrado abaixo:

```
public class MyThread extends Thread {

    public MyThread(String name) {
         super(name);
    }

    public void run() {
        for (int i= 0; i<1000; i++) {
            System.out.println(this.getName()+" i: "+i);
        }
    }
}
```

Na classe acima, criamos um construtor onde recebemos o nome da Thread, só para identificar quando cada classe receberá uma “fatia”de tempo. Todo o processamento ficará dentro do método run().

Thread

Abaixo, um exemplo de execução de cinco Threads da classe acima:

```
public class NamedThreadsTest {
    public static void main(String[] args) {

        MyThread primeira   = new MyThread("primeira ...");
        MyThread segunda  = new MyThread("segunda ...");
        MyThread terceira   = new MyThread("terceira ...");
        MyThread quarta   = new MyThread("quarta ...");
        MyThread quinta   = new MyThread("quinta ...");
        primeira.start();
        segunda.start();
        terceira.start();
        quarta.start();
        quinta.start();
    }
}
```

Ao executarmos a classe acima, perceberemos que todas as Threads irão imprimir até o número 999, mas não podemos garantir quem irá terminar primeiro.