Programação Orientada e Objetos II

Anhanguera


AVALIE
SUA PROFISSÃO
QUANDO APARECER EM SEU PORTAL UMA AVALIAÇÃO SOBRE SEU CURSO, RESPONDA:
NOTAS
9 OU 10
SIGNIFICA QUE VOCÊ INDICA
NOTAS
7 OU 8
SIGNIFICA QUE VOCÊ NÃO INDICA
Anhanguera

Anhanguera

Programação em Java usando threads

Anhanguera

Para o desenvolvimento de um software, são necessários diversos conhecimentos sobre as maneiras de criação de interface gráfica, as diversas formas de conexão com o banco de dados e a utilização detalhada da orientação a objetos. É necessário usar corretamente a orientação a objetos para garantir a manutenção do código e a alta coesão e o baixo acoplamento dos métodos, classes e pacotes. Uma solução computacional tende a crescer e a se tornar mais complexa com o tempo, e todos os cuidados mencionados elevam as possibilidades de o sistema manter-se estável. Entretanto, para que novas demandas sejam atendidas, são necessárias mais técnicas e elementos de programação, a fim de propiciar o processamento e o acesso à informação de forma eficiente.

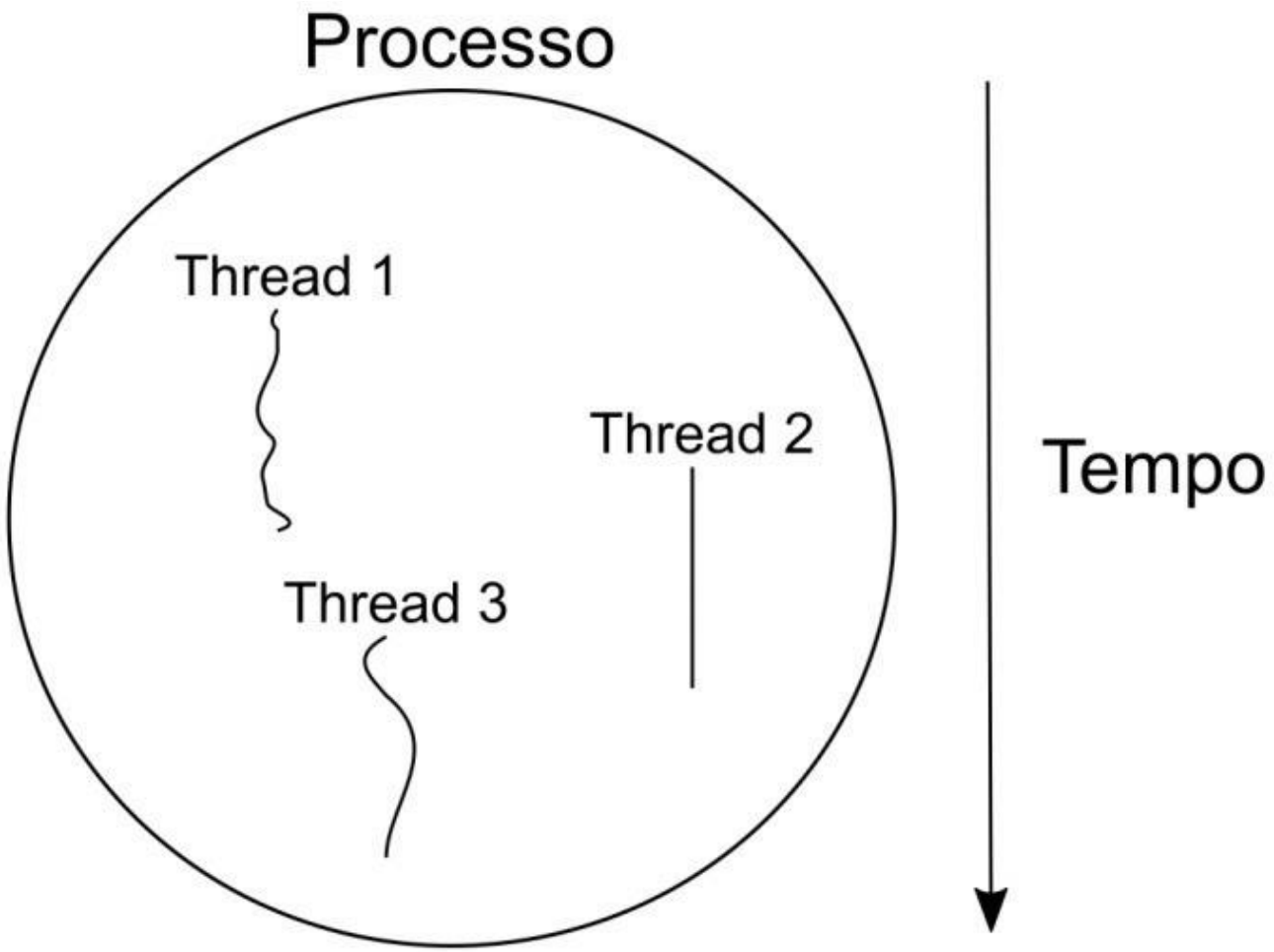
Com o avanço tecnológico, o hardware evoluiu para processadores que possuem diversos núcleos para a execução de instruções, levando novamente a necessidades mais específicas de programação. Esses cenários mais complexos precisam da execução de processos que tiram o máximo de rendimento do hardware, levando a um tempo menor de processamento ou ainda, à melhoria das formas de interação com o usuário.

Pensando nesses cenários, podemos imaginar diversos casos em que é necessário usar mais de um núcleo ou executar duas tarefas diferentes ao mesmo tempo. Como exemplo, podemos pensar em um sistema que gere relatórios de um grande número de entradas ou ainda que demande um processamento estatístico com uma quantidade elevada de informações. A interface gráfica inicial de um sistema que gera esses relatórios pode ter apenas os campos com as informações que estarão no relatório, então, ao selecionar o item que inicia o processamento, o sistema faz a busca no sistema de gerenciamento de banco de dados e começa a processar a informação. É necessário informar o usuário sobre o progresso desse processamento. Porém, se o software está executando esse processo e precisa terminar o mais rápido possível, não deve parar exibir informações ao usuário.

Observando as necessidades de aproveitar diversos núcleos de processamento, desenvolvendo softwares que usam o hardware de maneira mais eficiente, podemos pensar em duas abordagens. A primeira consiste em gerar diversos executáveis de um programa para processar as informações, porém, essa abordagem gera formas mais complexas e lentas de comunicação, como arquivos de texto que são lidos por diversas aplicações, comunicação pelas informações escritas no banco de dados ou comunicação pela rede. Uma forma mais eficaz de se tirar proveito de um processador com diversos núcleos ou ainda prover maneiras para que um software processe as informações e ainda informar o progresso para o usuário são as threads.

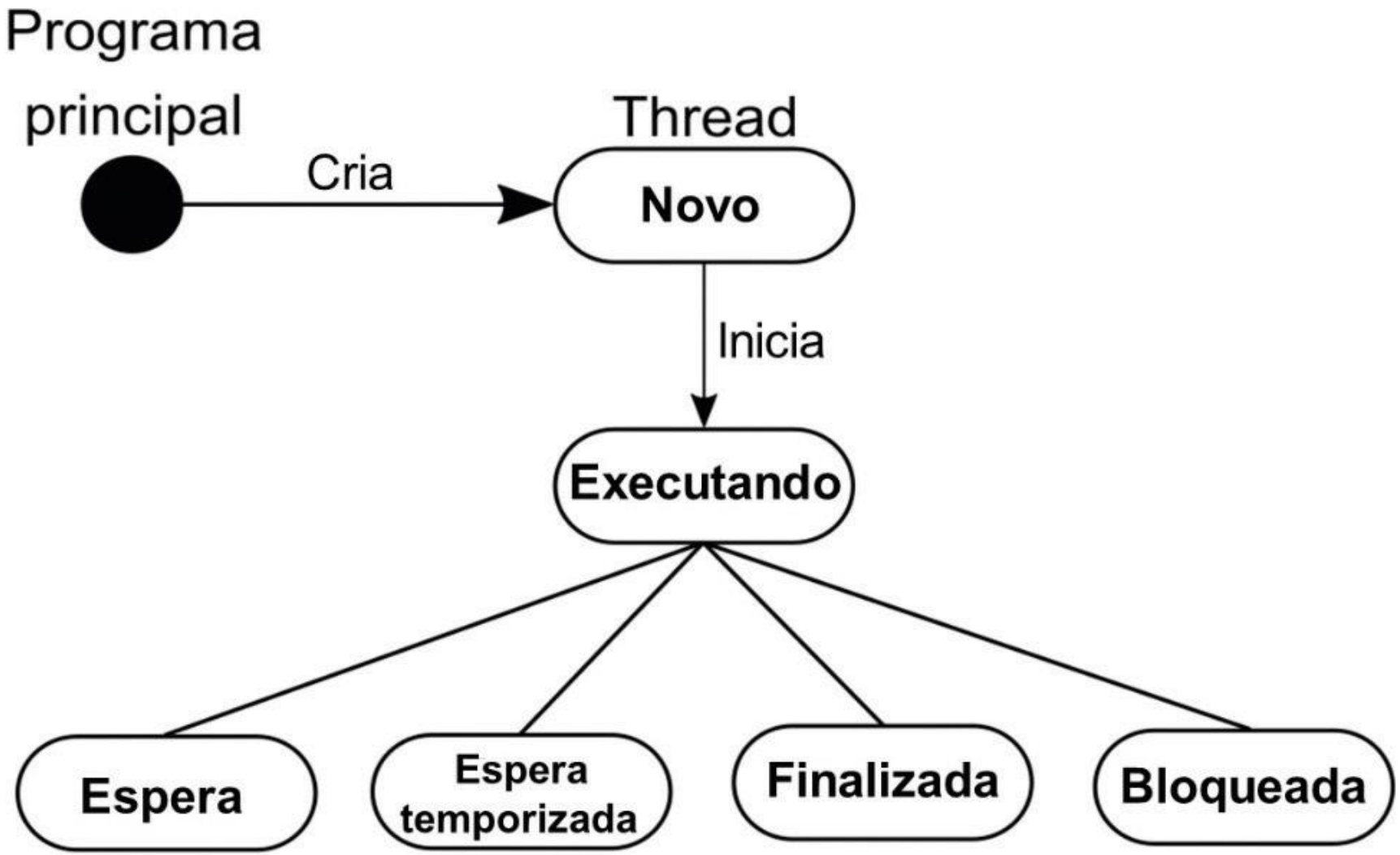
Cada programa em execução possui uma sequência de passos a serem realizados, ou seja, um fluxo de execução. Uma thread nada mais é do que um fluxo de execução ou, como muitas vezes é chamada, uma linha de execução dentro de um processo. Para um software efetuar diversas tarefas ao mesmo tempo, é preciso utilizar os diversos núcleos do processador de forma paralela, ou seja, é preciso criar um programa que possua várias threads ou simplesmente multithread (DEITEL e DEITEL, 2016).

A Figura 2.1 representa um processo (unidade do sistema operacional que possui os dados de um software em execução) em que há diversas linhas de execução que podem processar funções diferentes para aproveitar hardware com diversos núcleos ou ainda utilizar uma linha de execução para processar informações e outra para informar o usuário sobre o progresso dessa tarefa.


Processo
Thread 1
Thread 2
Thread 3
Tempo

Utilização de threads na linguagem Java

Na linguagem Java, para que seja possível gerar essas linhas de execução, é necessário utilizar uma Application Programming Interface (API) que interaja com o sistema operacional, determine a criação das threads e faça as configurações necessárias (FURGERI, 2015). Além da criação, é preciso observar que as threads possuem estados de execução relativos ao tempo de processamento concedido pelo sistema operacional. A Figura 2.2 apresenta os passos de criação e os possíveis estados que uma thread pode assumir. O programa principal cria a thread e essa nova linha de execução fica em estado “Novo”. Quando o sistema operacional escalona essa thread para execução, ela entra no estado “Executando”, etapa em que as tarefas definidas para thread são efetuadas. A partir desse ponto é possível que ela passe para quatro outros estados: “Esperando”, “Espera temporizada”, “Finalizada” ou “Bloqueada” (DEITEL e DEITEL, 2016).


Programa
principal
Cria
Thread
Novo
Inicia
Executando
Espera
Espera
temporizada
Finalizada
Bloqueada

A thread entra estado de “Espera” quando, após executar por um certo tempo, o sistema operacional interrompe o processamento para permitir que as outras threads ou processos possam realizar suas tarefas. O estado “Espera temporizada” é acionado quando a própria thread pede para interromper a execução por certo tempo, e essa interrupção é importante para garantir que uma thread nãotente ocupar muito tempo de processamento (HORSTMANN, 2016).

Certos momentos, uma thread pode requisitar alguma operação que necessita de ações relacionadas à entrada e saída do sistema. Com isso, é feita uma chamada ao sistema operacional para disponibilizar esse recurso (TANENBAUM e BOS, 2016) que, enquanto não está pronto e disponibilizado pelo sistema operacional, deixa a thread no estado “Bloqueada”, somente voltando a executar quando o recurso está liberado. Por fim, o estado “Finalizada” se refere a quando o processo que controla a thread determina seu fim e pede para o sistema operacional terminar a execução.

Para criar threads na linguagem Java, é possível utilizar diversas classes que fazem a abstração das linhas de execução de dentro de um processo. O Quadro 2.1 apresenta as duas mais comuns.

Classe	Descrição
java.lang.Thread	Classe inicial que fornece a utilização básica de threads.
java.util.Timer	Classe que fornece a utilização de thread com elementos de repetição e controle encapsulados.

Para se criar uma thread, são necessários 3 passos:

1. Criar uma classe que implementa a interface Runnable e o método run().
2. Criar uma instância tipo Thread e indicar a classe que implementa a interface Runnable.
3. Iniciar a thread com o método start().

Para visualizar essas implementações a melhor forma é um exemplo de código, vamos ver como criar essas threads? No Quadro 2.2 a classe Processador implementa a interface Runnable, que determina a criação do método run(), que será uma nova linha de execução dentro desse processo. A linha 2 declara um objeto da classe Thread chamado th que faz a abstração para se criar as threads. Na linha 4 esse objeto é instanciado, informando que a própria classe Processador possui o método run() (th = new Thread(this);) e, por fim, a thread é iniciada (th.start();). A linha 6 é o método que será executado quando a thread for iniciada e, finalmente, a linha 7 cria uma instância da classe Processador no método main, dando início ao software.

```
    package U2S1;
1.  public class Processador implements Runnable{
2.      private Thread th;
        public Processador()
3.      {
            th = new Thread(this);
4.          th.start();
5.      }
        public void run() {
6.          for (int i = 0; i < 1000; i++) {
                System.out.println("Processando
dados "+i);
            }
        }
            public static void main(String[] args)
7.          Processador p = new Processador();
        }
}
```

O Quadro 2.3 apresenta o resultado da execução do código da thread que acabamos de verificar. Ainda no código, repare que em nenhum momento é chamado o método run(), ele apenas é indicado na criação da thread (th = new Thread(this);). Com isso, foi criada uma linha de execução separada da linha principal.

```
Processando dados 0
Processando dados 1
Processando dados 2
Processando dados 3
Processando dados 4
Processando dados 5
...
Processando dados 999
```

Aplicação usando thread na linguagem Java

Um cenário para aplicação de threads são os watchdogs, softwares que têm o objetivo de monitorar recursos e aplicações. Como exemplo, podemos citar o monitoramento de uma conexão com o sistema de gerenciamento de banco de dados ou ainda a execução de um software de controle de câmeras de segurança. Os watchdogs ficam monitorando as aplicações e, em caso de problemas, podem gerar um alerta ou forçar a reinicialização do software ou do computador. Vamos ver no Quadro 2.4 um código completo que utiliza thread para monitorar um recurso.

As linhas 1 e 2 definem os arquivos a serem incluídos e a qual pacote a classe pertence. A configuração da thread é feita das linhas 3 a 8, onde criam-se os elementos básicos da thread, que consistem em implementar a interface Runnable (linha 3), declarar o objeto da classe Thread (linha 4), declarar e atribuir estado da variável booleana para monitoramento (linhas 5 e 7), instanciar o objeto do tipo Thread e indicar qual classe possui o método run() (linha 8). O processo de controle de uma thread consiste em formas de iniciar e interromper essa linha de execução. O método da linha 9 public void iniciar() inicia a thread chamando o método th.start(); e o método da linha 11 public void parar() força a interrupção da thread. O processo de interrupção necessita de dois passos: interrupt() e join(). Na linha 12 a variável booleana de controle é definida como false e na linha 13 é chamado o método th.interrupt();, que declara que essa thread deve ser parada, mas isso deve ser feito pela própria implementação no método que a thread executa (repare que o método run() na linha 20 verifica se a thread for interrompida e, nesse caso, o método run() para de executar). O próximo passo consiste em chamar o método th.join(2000); na linha 15, que espera 2000 milissegundos até que a thread finalize sua execução. Repare que ele utiliza o sistema de tratamento de exceção nas linhas 14, 16 e 17. Utilizar o join é importante para garantir que a thread termine seu processamento e possa reportar sua finalização ao processo principal.

O método run() é aquele que a thread utiliza para executar suas tarefas e inicia exibindo uma mensagem, em seguida inicia-se o processo de monitoramento através da estrutura while. Quando a thread é interrompida pelo método parar(), a execução do run() entra na linha 22, definindo a interrupção do processamento. Em alguns casos, a thread pode não mostrar essa mensagem, pois é possível que esteja em sleep. Repare que na linha 26 é utilizado o Thread.sleep(1000);, forçando a thread a entrar em estado de espera temporizada, evitando usar muitos recursos e permitindo que outros processos possam ser utilizados. Nesse método run() devem ser implementadas as regras e o código que serão executados pela thread. Repare que o sleep está utilizando o tratamento de exceção (try e catch), para que, caso esse método seja interrompido, a thread também seja.

No método main, nas linhas 30 e 31 é inicializado o monitoramento e como exemplo foi utilizado a classe Scanner (linha 32) para ler os dados do teclado para perguntar ao usuário (linha 36). Na linha 37 há a reposta do usuário comparada, ignorando se o texto é maiúsculoou minúsculo, e verificando se deve continuar o monitoramento. Caso seja interrompido, é chamado o método monitor.parar();.

```
1.  package U2S1;

2.  import java.util.Scanner;

3.  public class Monitor implements Runnable{

4.      private Thread th;
5.      private boolean monitorando;

6.      public Monitor()
        {
7.          monitorando = true;
8.          th = new Thread(this);
        }
9.      public void inciar()
        {
10.         th.start();
        }
11.     public void parar()
        {
12.                 monitorando = false
13.         th.interrupt();
14.         try {
15.             th.join(2000);
16.         } catch (InterruptedException e) {
17.             e.printStackTrace();
            }
        }
18.     public void run() {
19.         System.out.println("Iniciando
monitoramento.");
20.         while(monitorando == true)
21.         {
```

```
22.             // verifica se sistema alvo ainda está em execução
23.             System.out.println("Monitorando.");
24.             if(th.isInterrupted() == true)
                {
25.                 System.out.println("Parando monitoramento.");
27.                 return;
28.             }
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    monitorando = false;
29.         }
30.       }
31.
32.     }
33.     public static void main(String[] args) {
```

```
34.         Monitor monitor = new Monitor();
35.         monitor.inciar();
36.         Scanner sc = new Scanner(System.in);
37.         boolean monitorar = true;
            do {
38.             System.out.println("Continuar monitoramento S/N?");
                String resp = sc.next();
40.             if(resp.equalsIgnoreCase("N") == true)
41.             {
                    monitorar = false;
                    monitor.parar();
                }
            }while(monitorar == true);
            sc.close();
    }
}
```

O resultado da execução está no Quadro 2.5. Repare que, após o início do monitoramento, as respostas para "*Continuar monitoramento S/N?*" ficam fora de ordem em relação às mensagens "Monitorando". Isso ocorre porque a thread que faz o monitoramento possui uma linha e execução separada da parte que faz a leitura do teclado.

```
Iniciando monitoramento.
Monitorando.
Continuar monitoramento S/N?
Monitorando.
Monitorando.
S
Continuar monitoramento S/N?
Monitorando.
Monitorando.
Monitorando.
N
```

O uso de threads é essencial para seja possível produzir códigos de qualidade que utilizam os recursos do processador de forma otimizada. A implementação desses recursos cria diversas possibilidades para interações, desempenho e monitoramento. Todavia, como é uma execução paralela, deve-se planejar e manter a organização do código para evitar problemas de alto uso de recursos e problemas de sincronismo.

O desenvolvimento de aplicações paralelas requer um planejamento preciso, pois diversas tarefas ocorrerão ao mesmo tempo. Dessa forma, para seguir a correta implementação, é necessário executar três passos para se criar uma thread.

Quais são os 3 passos para se criar uma thread utilizando Java?

a) Implementar a interface Runnable, indicar essa implementação na instância da classe Thread e iniciar a thread com o método start().

b) Implementar a interface Clonnable, indicar essa implementação na instância da classe Thread e iniciar a thread com o método join().

c) Instanciar a classe Thread, pois o método de execução não depende de interfaces.

d) Implementar a interface Runnable e indicar essa implementação na instância da classe Thread, pois a thread tem o início automático.

e) Instanciar a classe Runnable, pois o método de execução não depende de interfaces.

As threads são iniciadas por um processo principal que já possui a sua própria linha de execução. Quando a thread termina a execução dos comandos do seu corpo, é possível estabelecer uma espera (sleep) para que esta seja finalizada.

Na application programming interface (API) do Java, qual comando garante que só terá retorno quando a thread terminar sua tarefa?

a) sleep.

b) cancel.

c) start.

d) join.

e) kill.