ELEMENTOS PARA SINCRONIZAÇÃO

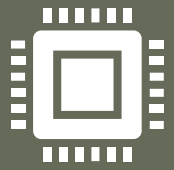
PROF. HÉLIO ESPERIDIÃO

INTRODUÇÃO

s recursos computacionais podem ser comparados a produtos de supermercados: existem diversos itens a serem comprados.

Ee dois clientes diferentes forem comprar o mesmo item, ao mesmo tempo, será necessário algum tipo de controle para evitar conflitos.

THREADS



As threads em um sistema computacional podem ser comparadas aos clientes, e os itens do mercado são relacionados à memória, aos dispositivos de entrada e de saída ou a outros recursos do computador.

Como forma de utilizar o hardware de maneira mais eficiente ou fazer com que uma aplicação seja capaz de executar diversas tarefas ao mesmo tempo, utilizamos as threads.

EXCEÇÕES EM UM CÓDIGO ORIENTADO A OBJETOS E PARALELO

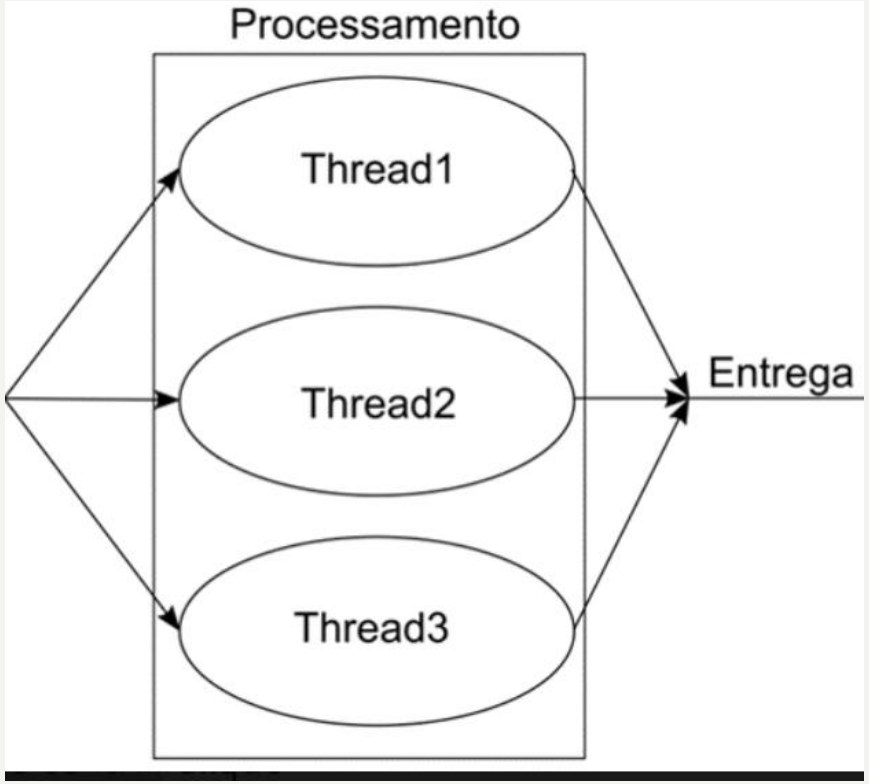
- Existem diversas formas de tratar as exceções em um código orientado a objetos e paralelo, todavia existem problemas que podem ocorrer em sistemas concorrentes que não geram erros que são tratados pelos mecanismos do próprio Java

- A utilização de threads em um código é essencial para que um software execute mais de uma tarefa ao mesmo tempo e, com isso, propicie mais interação com o usuário e use o hardware de maneira mais completa, entre outros cenários (MANZANO, 2014). Todavia, o uso da programação paralela pode gerar problemas decorrentes da execução de duas ou mais tarefas simultâneas.

Pense em uma classe básica em que quantidades de elementos são somadas ou subtraídas.

Nessa situação, imagine que diversas threads utilizam a mesma instância dessa classe para centralizar essas estatísticas.

Esse tipo de cenário em programação é bem comum, e a imagem apresenta um caso em que um programa principal cria três threads para fazer um processamento e, ao final, todas elas escrevem seus dados em apenas um elemento central



O MÉTODO SINCRONIZADO

É importante pontuar que quando duas threads fazem o acesso ao mesmo tempo, são necessárias estratégias da própria linguagem de programação, juntamente ao sistema operacional, para garantir que certos métodos sejam acessados por apenas uma thread por vez.

Para isso existe a palavra reservada **synchronized.**

CONTAGEM COM MÉTODOS SINCRONIZADOS

Imagine que diversas threads devem ao final de sua execução somar mais um em um determinado ponto da memória

Quando diversas threads acessarem esses métodos, a própria JVM, junto ao sistema operacional, consegue garantir que o acesso seja individual. Com os métodos **synchronized**, é possível produzir códigos paralelos confiáveis.

AS CLASSES *TIMER* E *TIMERTAK*

java.util.TimerTask: irá representar a tarefa a ser agendada. É uma classe abstrata que implementa a interface Runnable, assim, devemos criar uma sub-classe que implemente o método run().

java.util.Timer: irá representar o agendado de tarefas através de uma Thread.

AS CLASSES *TIMER* E *TIMERTAK*

```
public class Cronometro {
    Timer timer;

    public Cronometro(int seconds) {
        timer = new Timer();
        timer.schedule(new Tarefa(), seconds*1000);
    }

    class Tarefa extends TimerTask {
        public void run() {
            System.out.println("Execução da Tarefa Agendada");
            timer.cancel(); //Terminate the timer thread
        }
    }

    public static void main(String args[]) {
        new Cronometro(5);
        System.out.println("Tarefa Agendada");
    }
}
```

AGENDANDO TAREFAS DIÁRIAS

```
public class Cronometro {
    public Timer timer;
    private int primeiraVez=0;
    private int intervalo=0;
    public Cronometro(int primeiraVez,int intervalo) {
        this.primeiraVez=primeiraVez*1000;
        this.intervalo=intervalo*1000;
        this.timer = new Timer();
        this.timer.scheduleAtFixedRate(new Tarefa(), this.primeiraVez, this.intervalo);
         System.out.println("Tarefa Agendada ");

    }
    private class Tarefa extends TimerTask {
        public void run() {
            System.out.println("Execução da Tarefa Agendada");
        }
    }
    public static void main(String args[]) {
       Cronometro c= new Cronometro(5,1);
       int i=0;
       while(i!=1){
           System.out.println("1-Digite um para parar a execução da tarefa");
           Scanner scan = new Scanner(System.in);
           i=scan.nextInt();
           if(i==1)
               c.timer.cancel();
        }
        System.out.println("Tarefa Cancelada");

    }
}
```