



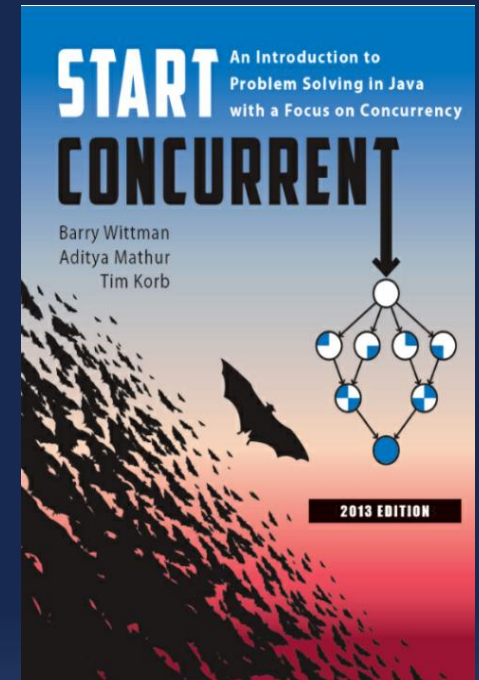
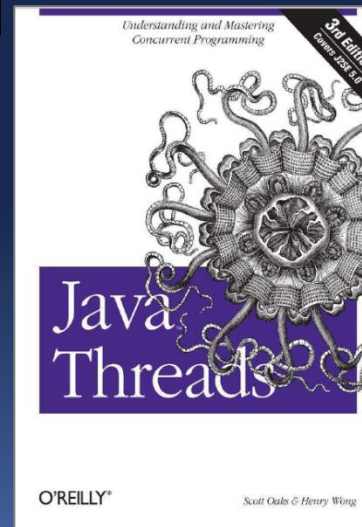
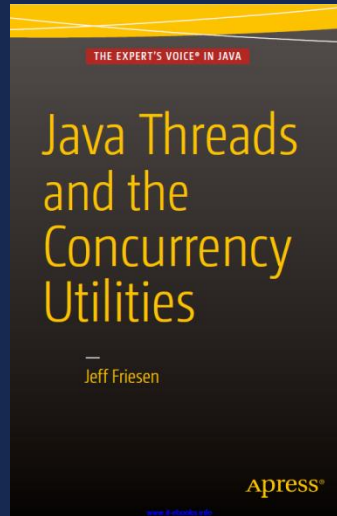
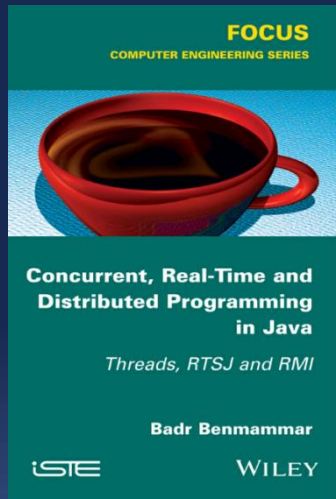
Programação Paralela e Concorrente

Unidade 9 – Comunicação entre Threads



Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUVSP
aparecidovfreitas@gmail.com

Bibliografia

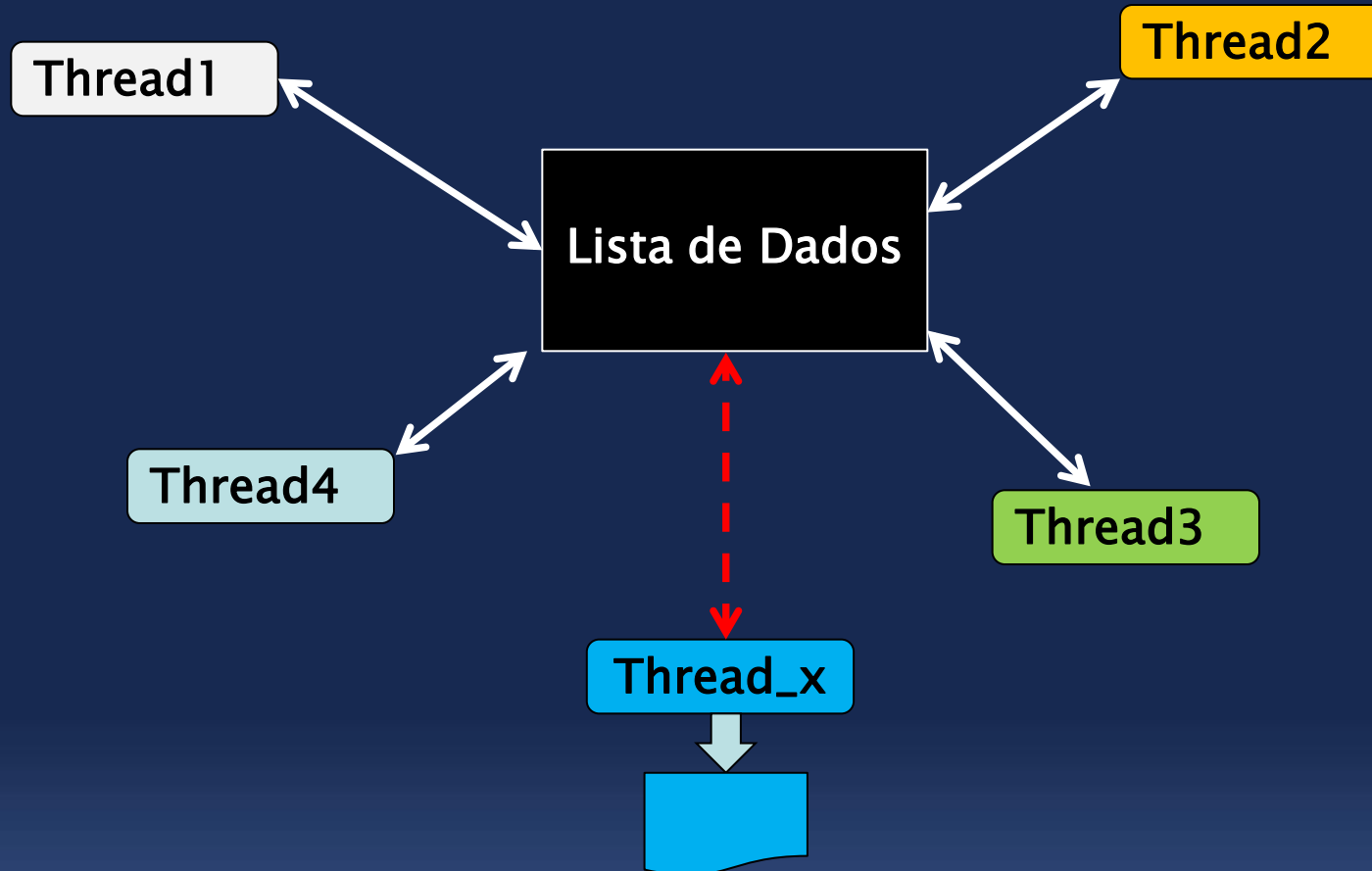


Introdução

- Veremos agora uma aplicação no qual um determinado thread se comunica com outro thread por meio de algum evento.
- Para exemplificar esse mecanismo de comunicação utilizaremos a mesma aplicação vista no capítulo 7, no qual alguns threads gravam em um array;
- O thread principal irá imprimir o array após a tarefa de preenchimento do array feita pelos outros threads.



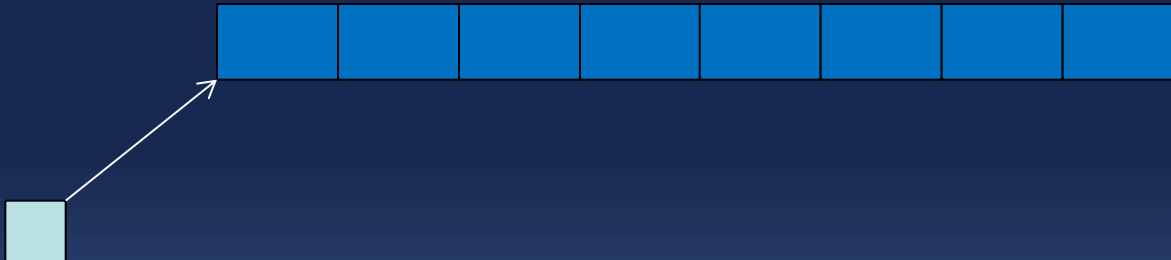
Aplicação



O **Thread_x** precisa esperar as tarefas de preenchimento do array feita pelos outros threads.

Classe Lista

- Vamos criar uma classe chamada **Lista** que representará o objeto que estará sendo compartilhado por diversos threads.



Classe Lista



- Na classe **Lista** iremos definir um array de **5000** elementos do tipo String;
- O método **insereString()** recebe um objeto do tipo String, adiciona-o na lista numa posição i e também incrementa essa posição i;
- O método **tamanho()** retorna o tamanho total da Lista;
- O método **recuperaString()** recebe uma posição i e retorna o String armazenado nessa posição.



Classe Lista

```
package br.uscs;

public class Lista {

    //objeto lista com 5000 elementos será compartilhado entre 10 threads
    private String[] listaString = new String[5000];
    private int indice = 0;

    public void insereString(String elemento) {
        this.listaString[indice] = elemento;
        this.indice++;
    }

    public int tamanho() {
        return this.listaString.length;
    }

    public String recuperaString(int posicao) {
        return this.listaString[posicao];
    }
}
```

Definição da Tarefa a ser executada

- Definiremos uma tarefa chamada **TaskAdicionaString** que será executada pelos **threads**, caracterizando dessa forma uma **Programação Concorrente**.
- A tarefa **TaskAdicionaString** será portanto do tipo **Runnable** e será responsável por inserir Strings na lista;
- Criaremos assim uma classe chamada **TaskAdicionaString** que deverá portanto implementar a interface **Runnable**;
- Nessa aplicação, iremos considerar que **10 threads** irão executar essa task em paralelo .

Classe TaskAdicionaString

```
package br.uscs;

public class TaskAdicionaString implements Runnable{

    private Lista lista;
    private int numeroDoThread;

    // Construtor
    public TaskAdicionaString(Lista lista, int numeroDoThread) {
        this.lista = lista;
        this.numeroDoThread = numeroDoThread;
    }
}
```

Classe TaskAdicionaString

```
@Override
public void run() {
    // Temos 10 threads para gravar 5000 elementos no array
    // cada thread irá, portanto, gravar 500 elementos no array
    // teremos ao final da aplicação todo o array preenchido com 5000 elementos

    for(int i=0; i < 500; i++) {
        String textoGerado = geraString();
        lista.insereString("Thread " + numeroDoThread + " gravou ==> " + textoGerado);
    }
}
```

Classe TaskAdicionaString

```
public static String geraString() {  
  
    String textoGerado = " ";  
    int indice;  
    String alfabeto = new String("!@#$%^&*<>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789");  
  
    for (int i=0 ; i<20; i++) {  
        indice = (int) (70.0 * Math.random());  
        String sub = alfabeto.substring(indice, indice+1);  
        textoGerado = textoGerado + sub;  
    }  
    return textoGerado;  
}
```

Classe Principal

- Nesta classe criaremos threads que irão manipular o objeto Lista da classe Lista definida anteriormente;
- Vamos criar nessa classe **10 threads** por meio de um laço **for**;
- Dentro desse laço, serão criados os threads, cada qual processando a tarefa **TaskAdicionaString()**, recebendo a lista compartilhada como argumento.

Classe Principal



```
package br.uscs;

public class Principal {

    public static void main(String[] args) {

        Lista lista = new Lista(); //objeto lista será compartilhado

        // A aplicação irá operar com 10 threads

        for (int i=0; i<10; i++) {

            TaskAdicionaString task1 = new TaskAdicionaString(lista,i);
            Thread thread1 = new Thread(task1, "Thread T" + i);
            thread1.start();

        }

        TaskImprimeLista task2 = new TaskImprimeLista (lista);
        Thread thread2 = new Thread(task2, "Thread I");
        thread2.start();

    }

}
```



Classe TaskImprimeLista

- Definiremos adicionalmente uma tarefa chamada **TaskImprimeLista** que será imprimir os valores da lista após o preenchimento dela pelos outros threads.

```
package br.uscs;

public class TaskImprimeLista implements Runnable{

    private Lista lista;
    private int numeroDoThread;

    // Construtor
    public TaskImprimeLista(Lista lista) {
        this.lista = lista;
        this.numeroDoThread = numeroDoThread;
    }


    @Override
    public void run() {
        for (int i=0; i<5000; i++) {
            System.out.println("lista[" + i + "] = " + lista.recuperaString(i));
        }
    }
}
```

Executando...

```

Problems @ Javadoc Declaration Console
<terminated> Principal (6) [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Jun 15, 2020, 4:52:44 PM)
--- lista[4986] = null ----
--- lista[4987] = null ----
--- lista[4988] = null ----
--- lista[4989] = null ----
--- lista[4990] = null ----
--- lista[4991] = null ----
--- lista[4992] = null ----
--- lista[4993] = null ----
--- lista[4994] = null ----
--- lista[4995] = null ----
--- lista[4996] = null ----
--- lista[4997] = null ----
--- lista[4998] = null ----
--- lista[4999] = null ----

```

-  A execução está mostrando que em algumas situações o thread que efetua a listagem da lista pode ter sido executado antes do preenchimento dela pelos outros threads.

Trata-se aqui de um problema de Sincronização de Threads

Reescrevendo a aplicação

```
package br.uscs;

public class Lista {

    //objeto lista com 1000 elementos será compartilhado entre 10 threads
    private String[] listaString = new String[1000];
    private int indice = 0;

    //synchronized ==> essa função está sendo chamada simultaneamente pelos threads
    public synchronized void insereString(String elemento) {

        this.listaString[indice] = elemento;
        this.indice++;

        if (this.indice == this.tamanho()) {
            System.out.println("Notificando que a lista foi preenchida pelos threads.....");
            this.notify();
        }
    }

    public int tamanho() {
        return this.listaString.length;
    }

    public String recuperaString(int posicao) {
        return this.listaString[posicao];
    }
}
```

Reescrevendo a aplicação

```
package br.uscs;

public class Principal {

    public static void main(String[] args) {

        Lista lista = new Lista(); //objeto lista será compartilhado

        // A aplicação irá operar com 10 threads

        for (int i=0; i<10; i++) {

            TaskAdicionaString task1 = new TaskAdicionaString(lista,i);
            Thread thread1 = new Thread(task1, "Thread T" + i);
            thread1.start();
        }

        TaskImprimeLista task2 = new TaskImprimeLista (lista);
        Thread thread2 = new Thread(task2, "Thread I");
        thread2.start();

    }
}
```

Reescrevendo a aplicação

```
package br.uscs;

public class TaskAdicionaString implements Runnable{

    private Lista lista;
    private int numeroDoThread;

    public TaskAdicionaString(Lista lista, int numeroDoThread) {
        this.lista = lista;
        this.numeroDoThread = numeroDoThread;
    }

    @Override
    public void run() {
        for(int i=0; i < 100; i++) {
            String textoGerado = geraString();
            lista.inserirString("Thread " + numeroDoThread + " gravou ==> " + textoGerado);
        }
    }

    public static String geraString() {

        String textoGerado = " ";
        int indice;
        String alfabeto = new String("!@#$%^*<>abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789");

        for (int i=0 ; i<20; i++) {
            indice = (int) (70.0 * Math.random());
            String sub = alfabeto.substring(indice, indice+1);
            textoGerado = textoGerado + sub;
        }
        return textoGerado;
    }
}
```

Reescrevendo a aplicação



```
package br.uscs;

public class TaskImprimeLista implements Runnable{

    private Lista lista;
    private int numeroDoThread;

    // Construtor
    public TaskImprimeLista(Lista lista) {
        this.lista = lista;
        this.numeroDoThread = numeroDoThread;
    }

    @Override
    public void run() {
        synchronized(lista) {
            try {
                System.out.println("Estou esperando que os outros threads terminem a gravação na Lista....");
                System.out.println("You dormir e aguardar a notificação ....");
                lista.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            for (int i=0; i<1000; i++) {
                System.out.println("--- lista[" + i + "] = " + lista.recuperaString(i) + " ----");
            }
        }
    }
}
```

