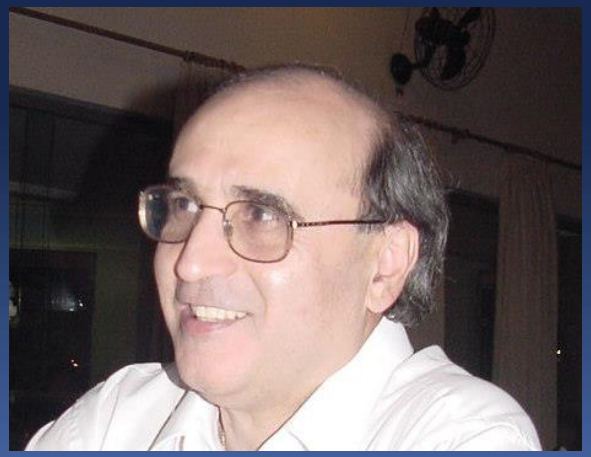
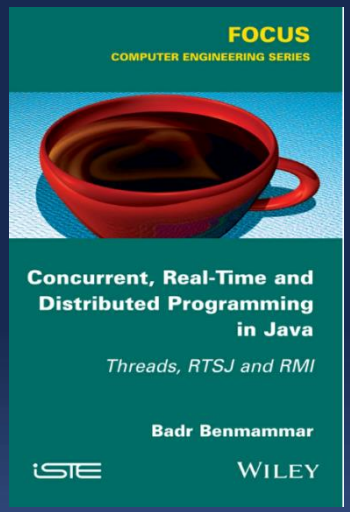
Programação Paralela e Concorrente

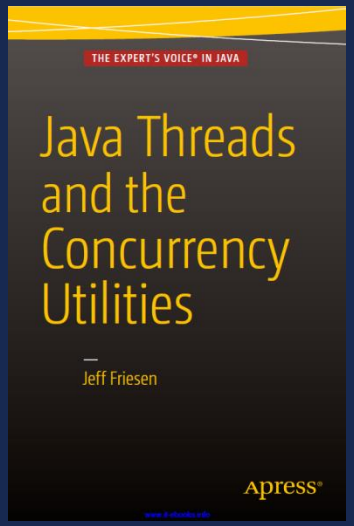
Unidade 8 – Sincronização de Threads
Framework Collections - ArrayList

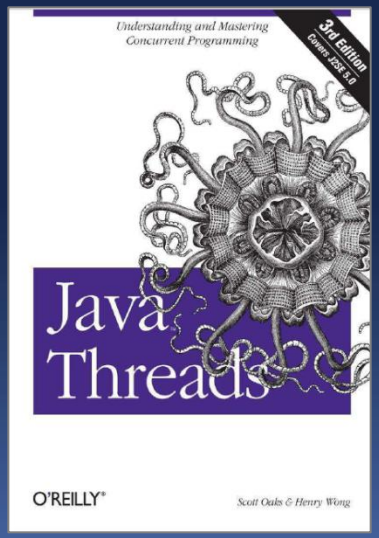


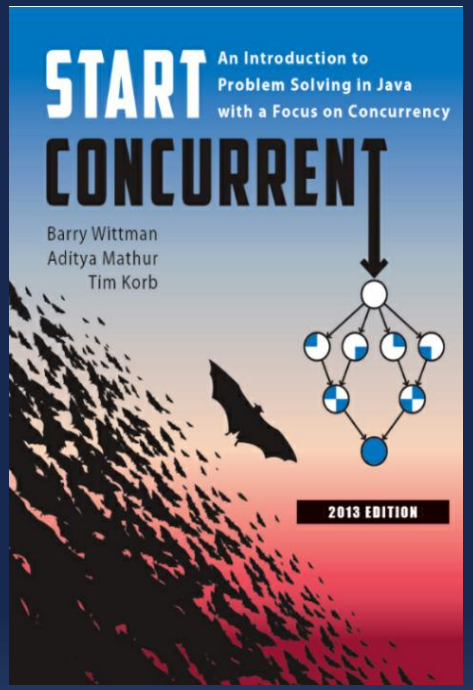
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

Bibliografia





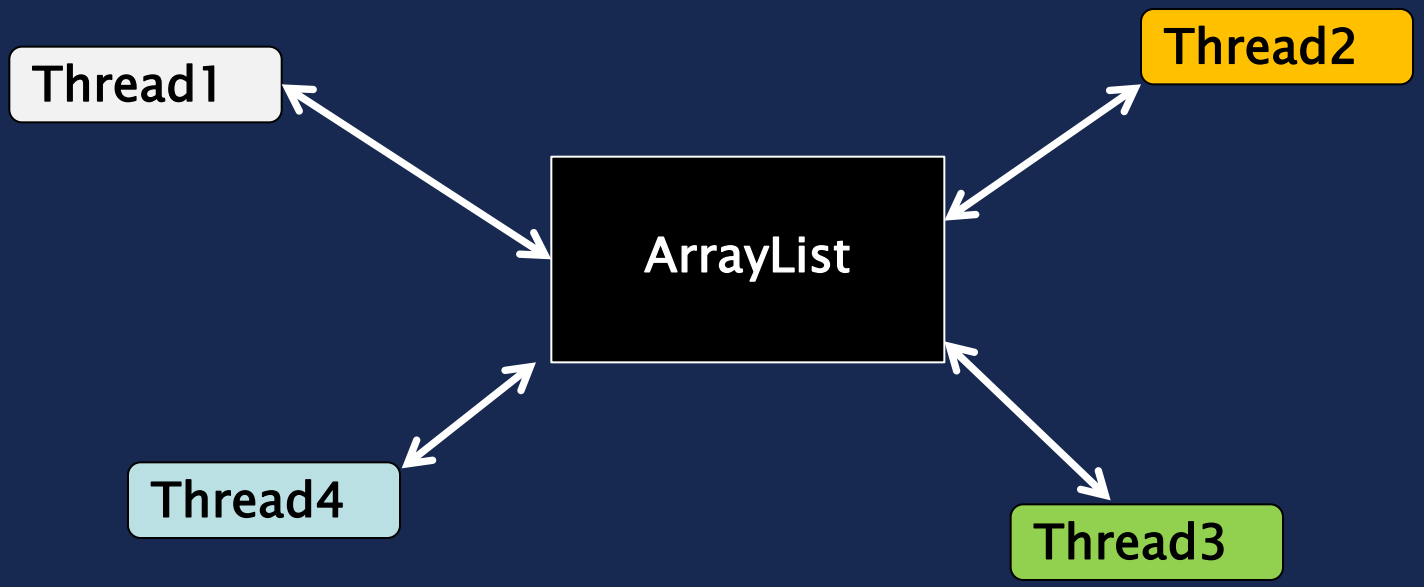




Introdução

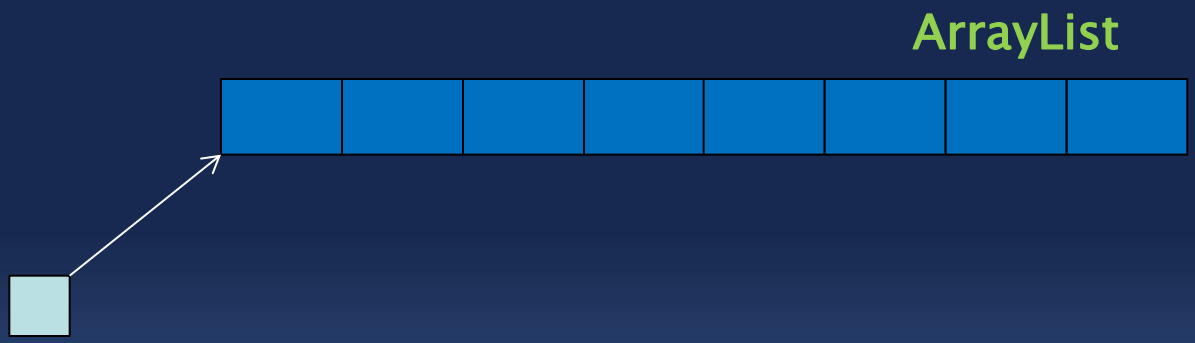
- Na última unidade, vimos um exemplo de uma aplicação no qual vários threads estão acessando um mesmo recurso (lista);
- Nesta unidade, veremos um exemplo adicional, no qual diversos threads estarão acessando uma lista de dados comum à todos eles, porém a lista será um objeto da classe **ArrayList** da Interface **Collections**.

Aplicação



Classe ArrayList

- Vamos criar uma classe chamada Lista que representará um objeto da classe **ArrayList**, o qual estará sendo compartilhado por diversos threads.



Lembrando...

ArrayList é uma classe genérica !

Tipo Genérico

- Também chamado de **tipo parametrizado**, é uma definição de classe que tem um ou mais tipos de parâmetros.
- Por exemplo, considere uma lista de valores inteiros com um conjunto de operações definidas.
- Poderíamos necessitar da mesma lista para implementar Strings, e assim por diante.

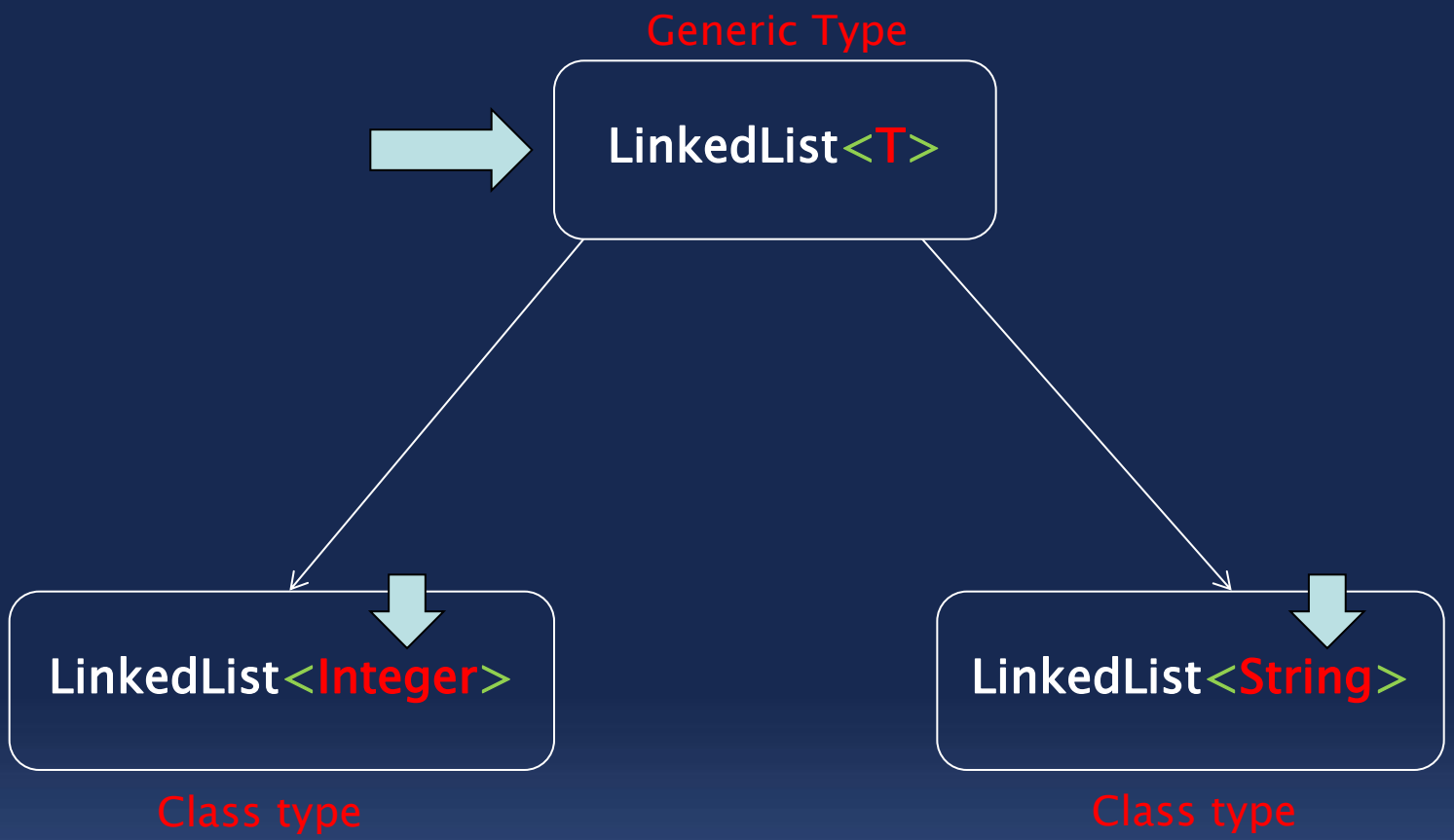
Como definir uma classe de tipo genérico ?

Classe Genérica – Definição

```
public     class    UserClass<T>  {

            //definicao de tipo genérico
}
```

- T entre < > é chamado tipo genérico (type parameter).
- Para se criar uma classe a partir da classe genérica, devemos simplesmente fornecer um apropriado argumento para o parâmetro entre < e >.
- Por exemplo: LinkedList<int> ou LinkedList<String> .

Classes Genéricas



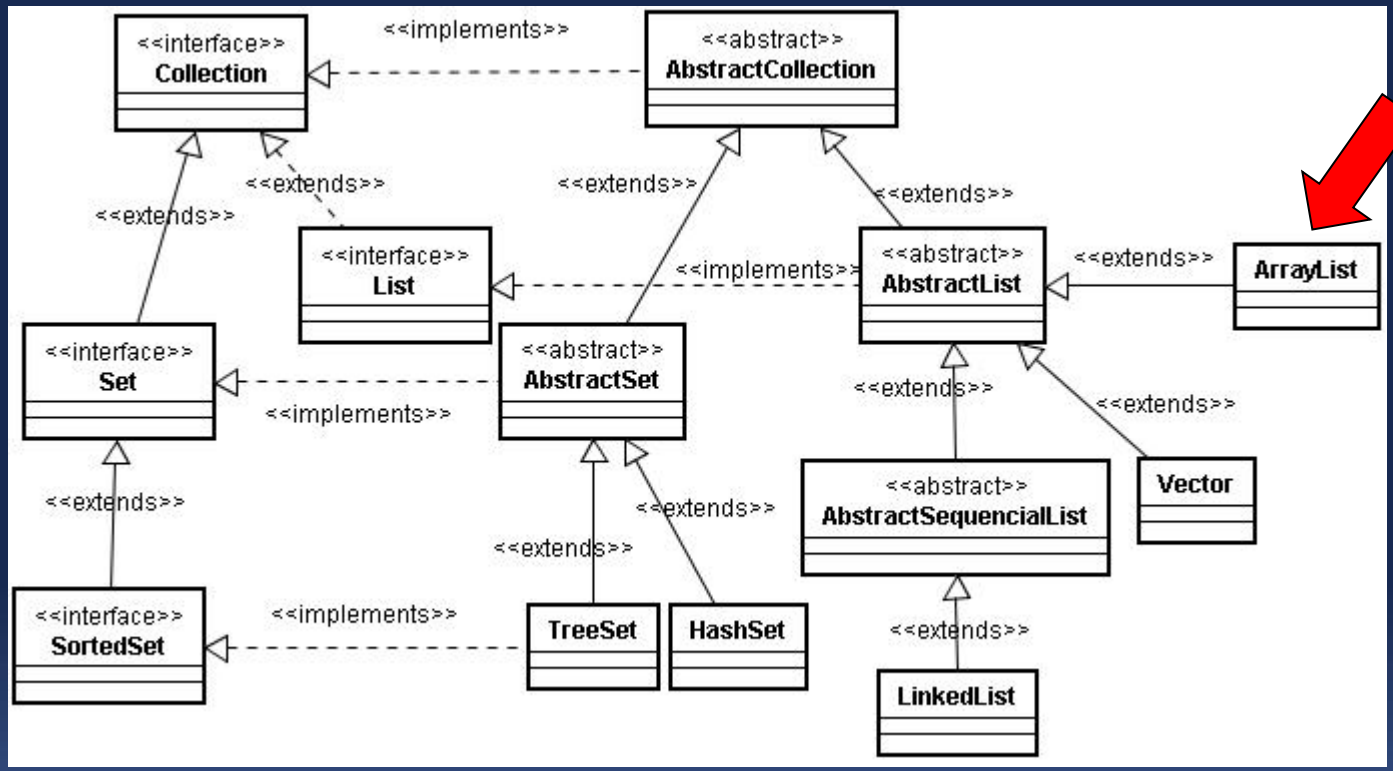
Classes Genéricas – Observações

- O argumento para o tipo genérico deve ser uma **classe** ou **interface**.
- Ou seja, não é permitido o uso de tipos primitivos, por exemplo, int ou double. Deve-se usar wrapper classes, tais como Integer, Double, etc.
- Ao se criar um tipo particular a partir de um genérico, o argumento é substituído em toda a ocorrência de **T** na especificação genérica de tipo.

API Collections Framework

- Exemplos de tipos genéricos são encontrados na **Collections Framework Java SE 5.0**.
- A necessidade de tipos genéricos surgiu na implementação e uso de **collections**.
- Uma coleção sempre contém elementos de um determinado tipo, tais como uma lista de inteiros, ou uma lista de Strings, etc.
- **ArrayList** é uma classe genérica com um type parameter que pertence ao Framework.
- **ArrayList** implementa uma estrutura de dados que modifica dinamicamente o seu tamanho em tempo de execução.

Classes e interfaces que estendem ou implementam a interface Collection



Classe ArrayList

- ArrayList é uma classe concreta da API Framework Collections.
- Diferentemente da estrutura Array que tem tamanho fixo, um ArrayList é um objeto que pode modificar seu tamanho e, portanto, é adequado para situações onde se necessita de comportamento dinâmico.
- Assim, um ArrayList tem o mesmo propósito de um Array, mas seu tamanho pode se modificar em tempo de execução.

ArrayList – método add

boolean	add(E e)Appends the specified element to the end of this list.

```
package oop;

import java.util.ArrayList;

public class ArrayList_01 {

        public static void main(String[] args) {

                ArrayList<String> x = new ArrayList<String>();
                x.add("USCS");
                x.add("Computacao");
                System.out.println(x.toString()) ;
        }
}
```

[USCS, Computacao]

ArrayList – método add

boolean	add(E e)Appends the specified element to the end of this list.

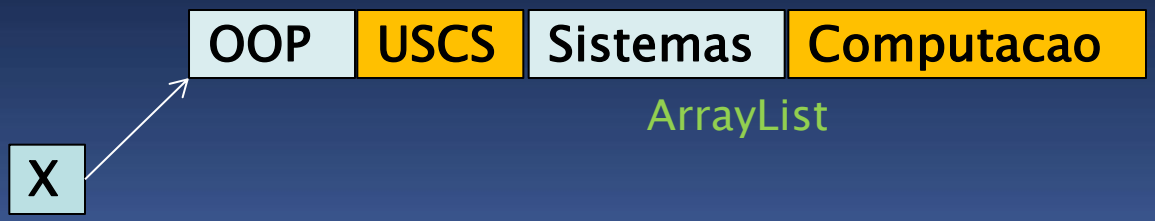
```
package oop;
import java.util.ArrayList;

public class ArrayList_02 {

        public static void main(String[] args) {

                ArrayList<String> x = new ArrayList<String>();
                x.add("USCS");
                x.add("Computacao");
                x.add(1,"Sistemas");
                System.out.println(x.toString()) ;
        }
}
```

[USCS, Sistemas, Computacao]

ArrayList – método add

void	add(int index, E element)Inserts the specified element at the specified position in this list.

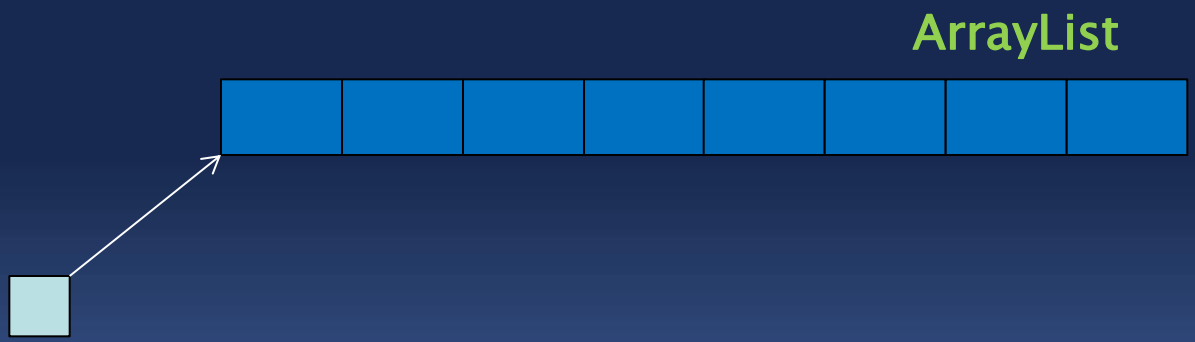
```
package oop;
import java.util.ArrayList;

public class ArrayList_03 {

        public static void main(String[] args) {

                ArrayList<String> x = new ArrayList<String>();
                x.add("USCS");
                x.add("Computacao");
                x.add(1,"Sistemas");
                x.add(0,"OOP");
                System.out.println(x.toString()) ;
        }
}
```



[OOP, USCS, Sistemas, Computacao]

Aplicação Threads com compartilhamento de Recursos

- Agora que já lembramos do conceito de Classe Genérica, do Framework **Collections** e de **ArrayList**, vamos implementar uma aplicação com o emprego de **threads** e que compartilha recursos, no caso um **ArrayList**.



Classe Principal

- Nesta classe criaremos threads que irão manipular um objeto da classe ArrayList (framework Collections);
- 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;

import java.util.ArrayList;

public class Principal {

    public static void main(String[] args) {

        List<String> lista =  new ArrayList<String>();

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

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

        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

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

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 no ArrayList;
- 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 .

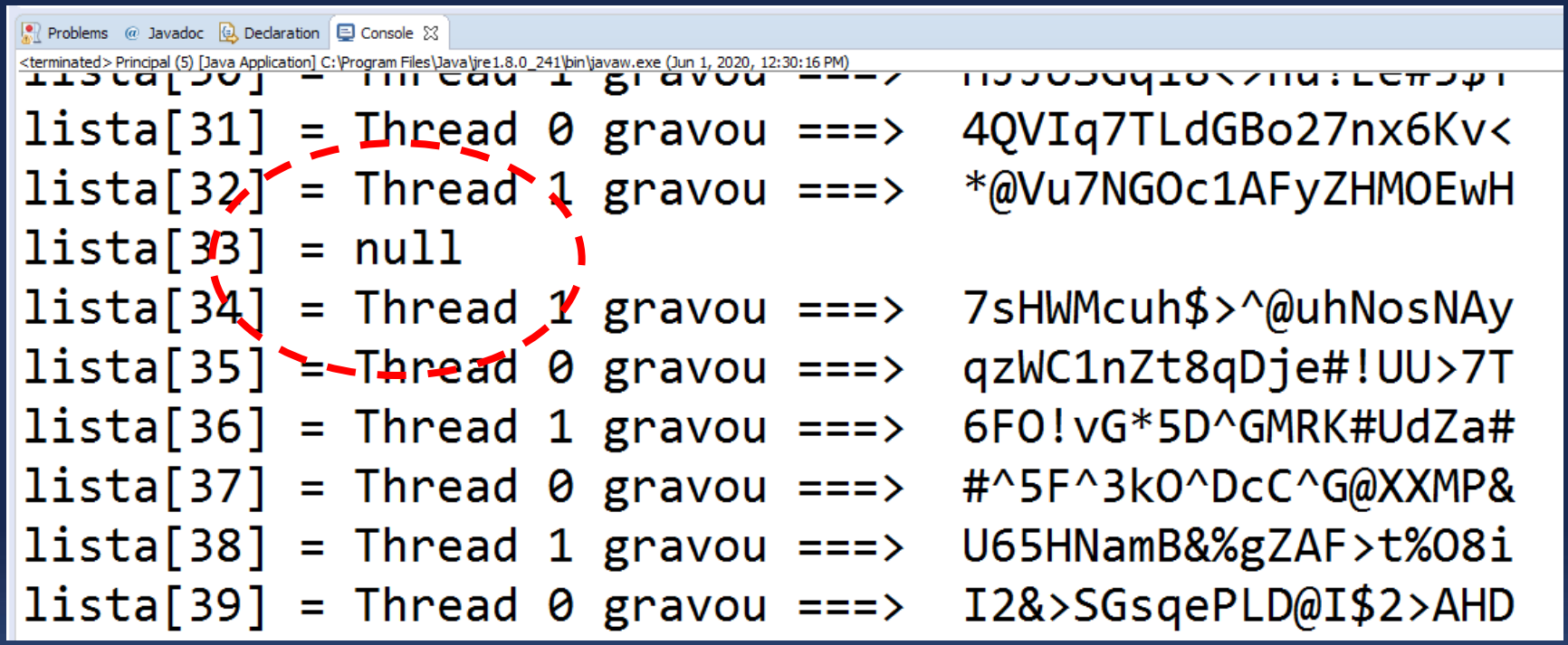
Classe TaskAdicionaString

```java
package br.uscs;

import java.util.List;

public class TaskAdicionaString implements Runnable{

    private List<String> lista;
    private int numeroDoThread;

    public TaskAdicionaString(List<String> lista, int numeroDoThread) {
        this.lista = lista;
        this.numeroDoThread = numeroDoThread;
    }

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

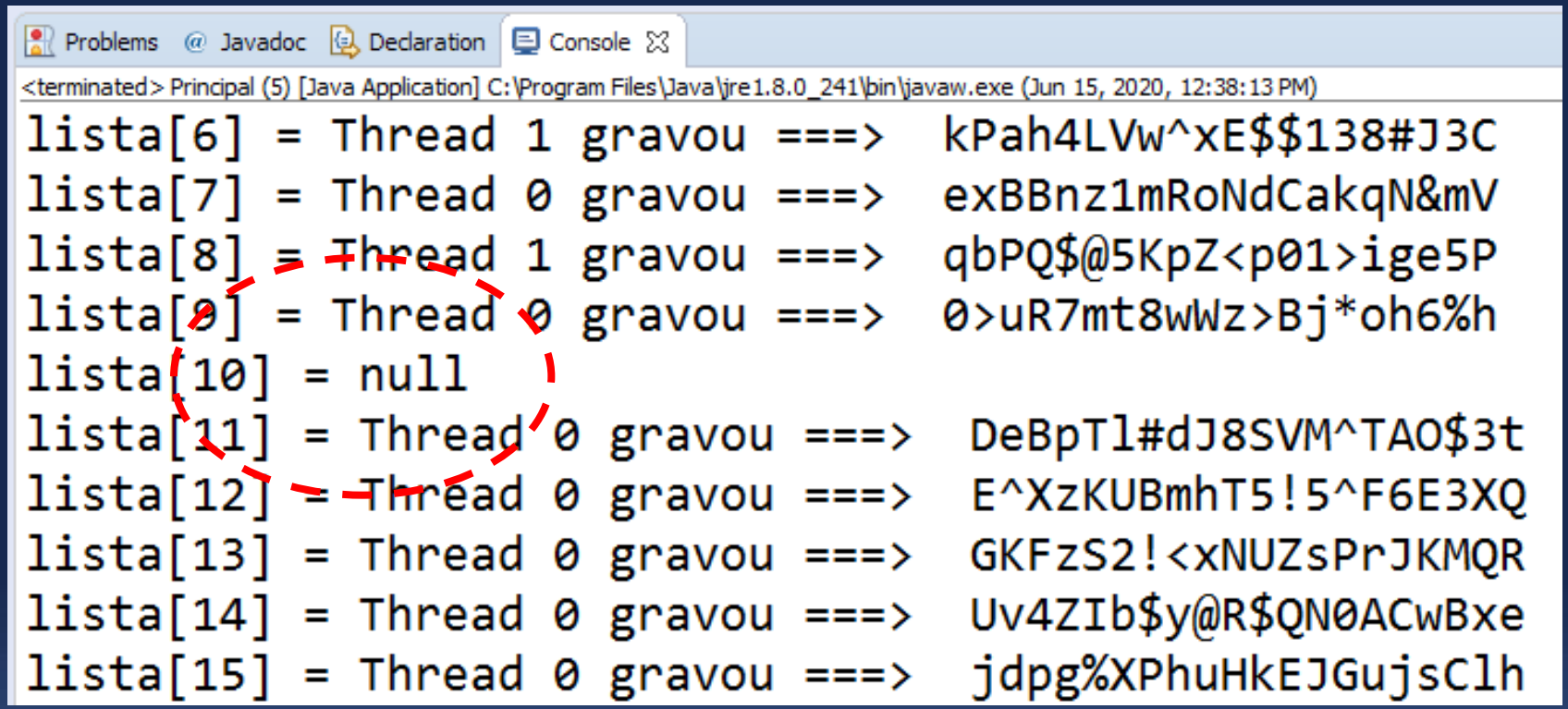
Classe TaskAdicionaString

```
    public static String geraString() {

        String textoGerado = " ";
        int indice;
        String alfabeto = new String("!@#$%^&*<>abcdefghijklmnopqrstuvxyzwABCDEFGHIJKLMNOPQRSTUVXZW0123456789");

        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;
    }
}
```

Executando a aplicação

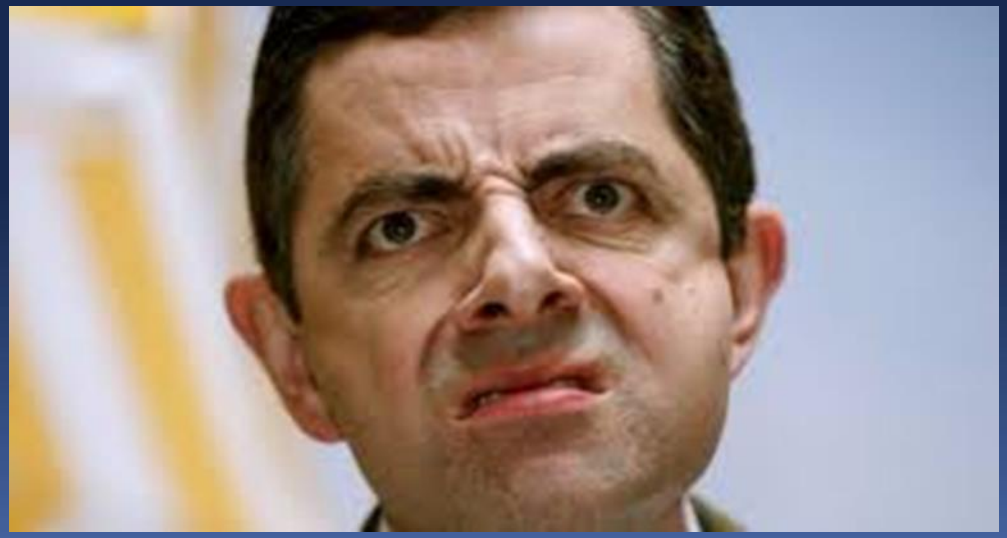


Executando a aplicação



Vé ... Mesmo com o uso de ArrayList (Collections), algumas posições do array estão com valores nulos ????

Por que?



Documentação ArrayList

compact1, compact2, compact3

java.util

Class ArrayList<E>

java.lang.Object
    java.util.AbstractCollection<E>
        java.util.AbstractList<E>
            java.util.ArrayList<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

Documentação ArrayList

Note that this implementation is not synchronized. If multiple threads access an `ArrayList` instance concurrently, and at least one of the threads modifies the list structurally, it *must* be synchronized externally. (A structural modification is any operation that adds or deletes one or more elements, or explicitly resizes the backing array; merely setting the value of an element is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the list. If no such object exists, the list should be "wrapped" using the

ArrayList não é thread safe!

Documentação ArrayList

`Collections.synchronizedList` method. This is best done at creation time, to prevent accidental unsynchronized access to the list:

```
List list = Collections.synchronizedList(new ArrayList(...));
```

Reescrevendo a aplicação

- Para trabalharmos de forma thread safe com array lists devemos utilizar o método **synchronizedList( )** da classe **Collections**;

- Iremos então reescrever a criação do ArrayList com a chamada deste método.

Reescrevendo a aplicação

```
package br.uscs;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Principal {

    public static void main(String[] args) {

        List<String> lista =  Collections.synchronizedList(new ArrayList<String>());

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

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

        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

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

Reexecutando a aplicação