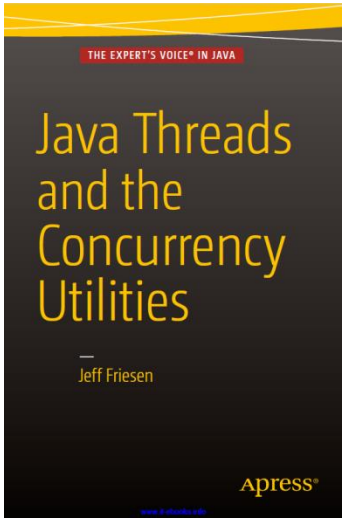
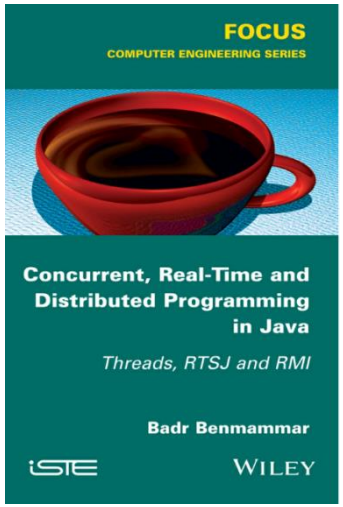
# Programação Paralela e Concorrente

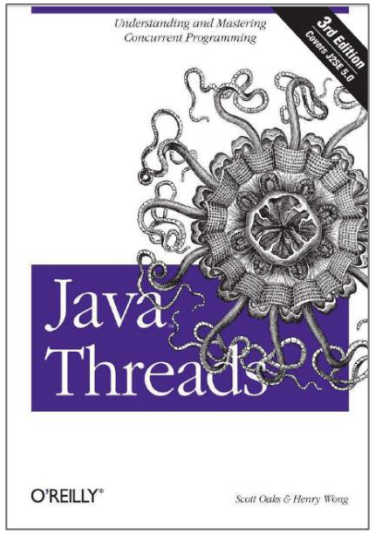
## Unidade 3 – Criação de Threads com a Linguagem Java

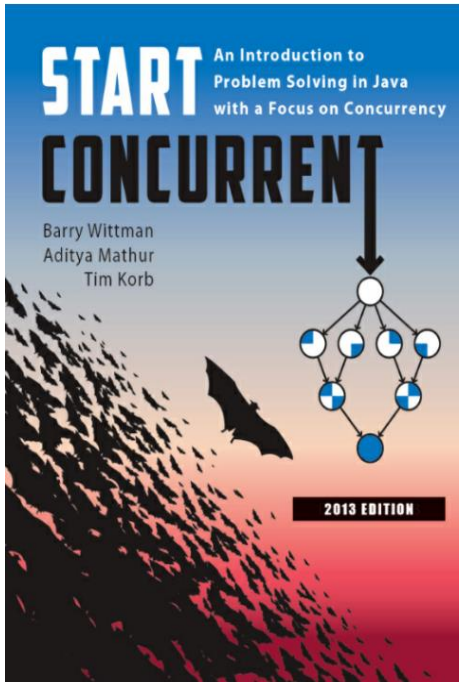

Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

# Bibliografia




# Como se cria threads em Java ?

# Criação de Threads em Java - Método 1

- Criar nova classe derivada de **java.lang.Thread**;

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

**PREV CLASS** **NEXT CLASS** FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.lang

## Class Thread

java.lang.Object
    java.lang.Thread

**All Implemented Interfaces:**

Runnable

**Direct Known Subclasses:**

ForkJoinWorkerThread

---

```
public class Thread
extends Object
implements Runnable
```

# Criação de Threads em Java - Método 1

- A classe **Thread** e suas subclasses fornecem mecanismos para a criação e controle de threads;
- Conforme **API** abaixo, a classe **Thread** implementa a interface **Runnable**;

```
public class Thread
extends Object
implements Runnable
```

# Criação de Threads em Java - Método 1

```
public class Thread
extends Object
implements Runnable
```

- Isto significa que a classe que cria **threads** deve estender a classe **Thread** e, portanto, deve implementar a interface **Runnable**.

# Mas, quais são os métodos que estão presentes na Interface Runnable ?

# Interface Runnable


OVERVIEW PACKAGE CLASS USE TREE DEPRECATED

**PREV CLASS** **NEXT CLASS** FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL:

compact1, compact2, compact3

java.lang

## Interface Runnable

```
public interface Runnable
```

The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called `run`.

# Interface Runnable – Método run()

***Method Summary***

**All Methods** | **Instance Methods** | **Abstract Methods**

| Modifier and Type | Method and Description |
|---|---|
| `void` | **`run`**`()`<br>When an object implementing interface `Runnable` is used to create a thread, starting the thread causes the object's `run` method to be called in that separately executing thread. |

# API – Java Threading

- No código abaixo, estamos definindo a classe **PrintUSCS** a qual tem apenas um método público chamado **imprimeUSCS();**

- Esse método simplesmente grava o String *"USCS... "* 20 vezes para a console.

```
package br.uscs;

public class PrintUSCS {

        public void imprimeUSCS() {

                for(int i = 0; i < 20; i++)
                        System.out.println(i + ": USCS....");
        }
}
```

# API – Java Threading

- Para se executar o código, pode-se criar uma classe chamada **TestPrintUSCS** a qual chamará o método **imprimeUSCS()** a partir de **main()**.

```
package br.uscs;
public class TestePrintUSCS {

        public static void main(String[] args) {

                PrintUSCS printUSCS = new PrintUSCS();

                printUSCS.imprimeUSCS();

                for (int i = 0; i < 10; i++)
                        System.out.println(i + ". Msg - main()...");
        }
}
```

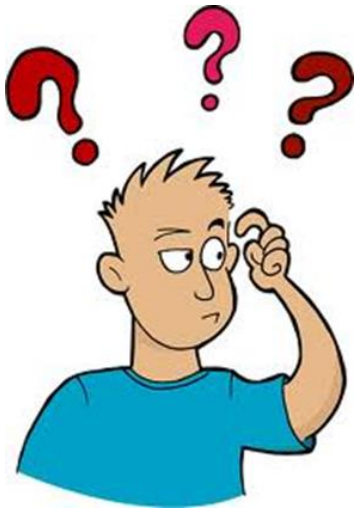
# Execução

Problems @ Javadoc Declaration Console

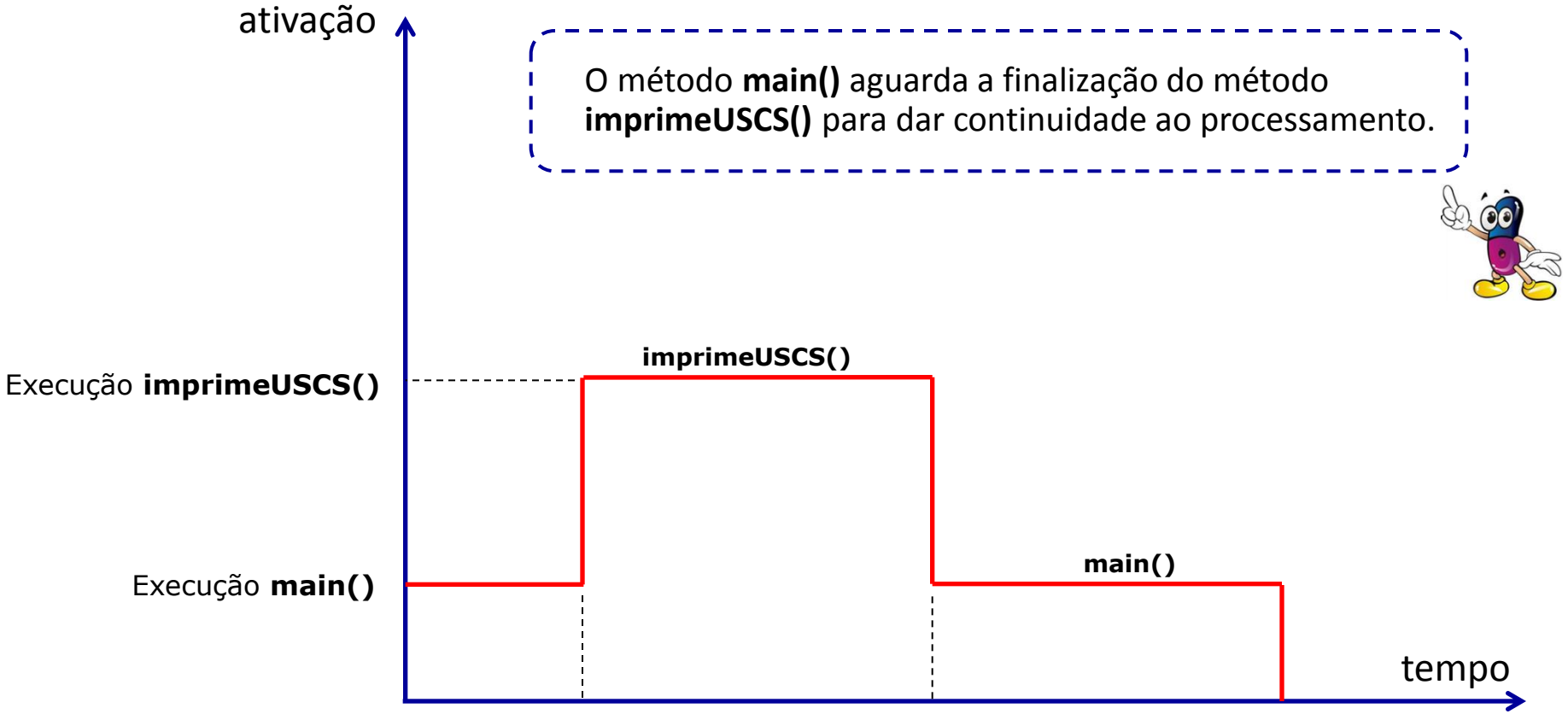
<terminated> TestePrintUSCS [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 25, 2020, 6:28:34 AM)

```
17: USCS....
18: USCS....
19: USCS....
0. Msg - main()...
1. Msg - main()...
2. Msg - main()...
3. Msg - main()...
4. Msg - main()...
5. Msg - main()...
6. Msg - main()...
7. Msg - main()...
8. Msg - main()...
9. Msg - main()...
```

# Como foi feita a execução ?

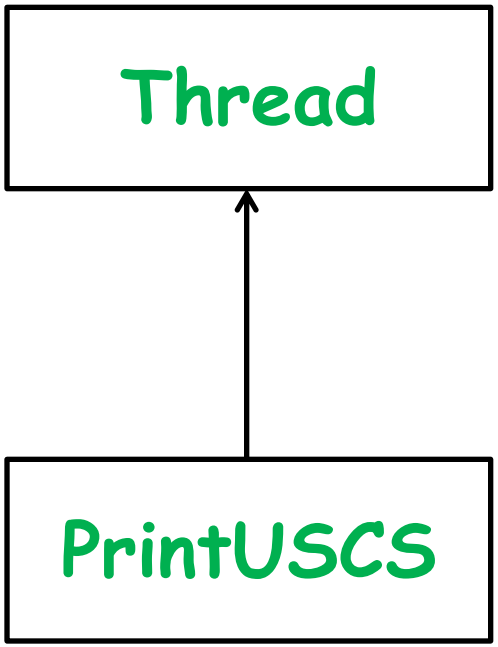


# Execução do Método




# Execução Concorrente

- Para se implementar a concorrência, modificaremos a classe **PrintUSCS**, de modo que ela seja executada em um novo thread;
- Para isso, faremos com que a classe **PrintUSCS** seja filha da classe **Thread**.




# Reescrevendo a classe PrintHello

- A classe **PrintHello** agora é **filha** da classe **Thread**.

```
package br.uscs;

public class PrintUSCS extends Thread {

public void imprimeUSCS() {

        for(int i = 0; i < 20; i++)
                System.out.println(i + ": USCS....");
        }
}
```

# Reexecutando o código

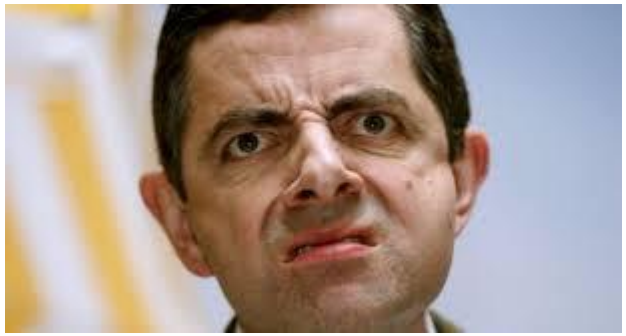
Problems @ Javadoc Declaration Console

<terminated> TestePrintHello [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 25, 2020, 5:56:03 AM)

```
997: Hello World....
998: Hello World....
999: Hello World....
0. Mensagem de main()...
1. Mensagem de main()...
2. Mensagem de main()...
3. Mensagem de main()...
4. Mensagem de main()...
5. Mensagem de main()...
6. Mensagem de main()...
7. Mensagem de main()...
8. Mensagem de main()...
9. Mensagem de main()...
```

# Vé ... Não entendi !!!

## Nada mudou ?
## A execução continuou sequencial ? ? ?



# Porque nada mudou?

- A classe **PrintUSCS** é filha da classe **Thread**, mas o Thread correspondente a essa classe não foi **iniciado**;
- Para iniciarmos o **Thread** será necessário chamarmos a função **start()** da classe **Thread**, na função **main()**;
- Veja a API abaixo:

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED

**PREV CLASS** **NEXT CLASS** FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: F

compact1, compact2, compact3

java.lang

**Class Thread**

`void` **`start()`**

Causes this thread to begin execution;

the Java Virtual Machine calls the run method of this thread.

# Reescrevendo a classe TestPrintUSCS

- Para iniciar o Thread, será necessário chamar a função **start()** na função **main()**.

```
package br.uscs;
public class TestePrintUSCS {

public static void main(String[] args) {

        PrintUSCS printUSCS = new PrintUSCS();

        printUSCS.start();

        for (int i = 0; i < 10; i++)
                System.out.println(i + ". Msg - main()...");
        }
}
```

# Reexecutando a classe TestPrintUSCS

Problems @ Javadoc Declaration Console

<terminated> TestePrintUSCS [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 25, 2020, 6:48:11 AM)

```
0. Msg - main()...
1. Msg - main()...
2. Msg - main()...
3. Msg - main()...
4. Msg - main()...
5. Msg - main()...
6. Msg - main()...
7. Msg - main()...
8. Msg - main()...
9. Msg - main()...
```

# Vé ... Não entendi de novo !!!

## Piorou ? ? ?
## Porque o Thread não foi iniciado ?

# Criação de Threads em Java - Método 1

```
public class Thread
extends Object
implements Runnable
```

- Isto significa que a classe que cria **threads** deve estender a classe **Thread** e, portanto, deve implementar a interface **Runnable**.

# Interface Runnable


OVERVIEW PACKAGE CLASS USE TREE DEPRECATED

**PREV CLASS** **NEXT CLASS** FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL:

compact1, compact2, compact3

java.lang

## Interface Runnable

```
public interface Runnable
```

The `Runnable` interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called `run`.

# Reescrevendo a classe PrintUSCS


- Na classe **PrintUSCS** devemos então implementar o método **run()** que será chamado pelo método **start()** em **main()**;

- Vamos então, renomear o método **printUSCS()** para **run()** .

```
package br.uscs;

public class PrintUSCS extends Thread {

public void run() {

        for(int i = 0; i < 20; i++)
                System.out.println(i + ": USCS....");
        }
}
```

# Alterando o código para se visualizar a concorrência

- A classe **PrintUSCS** correspondente ao novo Thread irá imprimir 100 vezes a mensagem "USCS ...";
- A função **main()** na classe TestPrintUSCS irá imprimir 100 vezes a mensagem: "`Msg - main()...`".

# Alterando o código para se visualizar a concorrência

```
package br.uscs;

public class PrintUSCS extends Thread {

        public void run() {

                for(int i = 0; i < 100; i++)
                        System.out.println(i + ": USCS....");
        }
}
```

# Alterando o código para se visualizar a concorrência

```
package br.uscs;
public class TestePrintUSCS {

	public static void main(String[] args) {

		PrintUSCS printUSCS = new PrintUSCS();

		printUSCS.start();

		for (int i = 0; i < 100; i++)
			System.out.println(i + ". Msg - main()...");
	}
}
```

# Reexecutando-se o código

Problems Javadoc Declaration Console

<terminated> TestePrintUSCS [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 25, 2020, 7:10:15 AM)

```
0. Msg - main()...
0: USCS....
1: USCS....
2: USCS....
1. Msg - main()...
3: USCS....
4: USCS....
5: USCS....
6: USCS....
7: USCS....
2. Msg - main()...
8: USCS....
```

Código concorrente ! ! !

# Reexecutando-se o código

Problems | Javadoc | Declaration | Console

<terminated> TestePrintUSCS [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 25, 2020, 7:10:15 AM)

```
0. Msg - main()...
0: USCS....
1: USCS....
2: USCS....
1. Msg - main()...
3: USCS....
4: USCS....
5: USCS....
6: USCS....
7: USCS....
2. Msg - main()...
8: USCS....
```

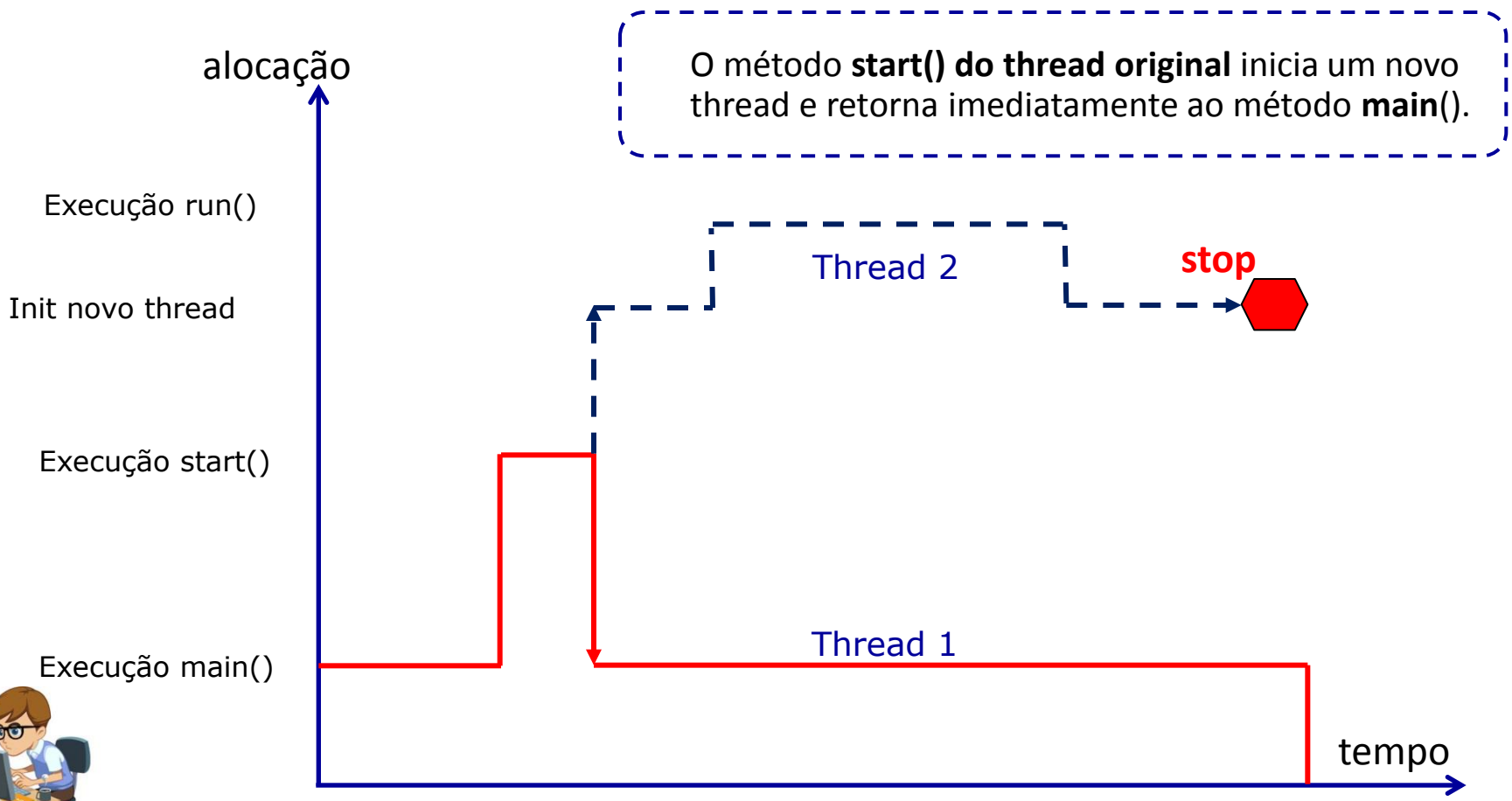
A saída é **entrelaçada** e **não-determinística**...

# Execução Concorrente

- As mudanças foram as seguintes: A classe **PrintUSCS** herdou funções da classe Thread;
- Na classe **PrintUSCS**, sobrescreveu-se o método **run()** que contém o código do novo thread que será processado de forma concorrente ao código principal (função **main()** );
- Na classe **TestPrintUSCS** iniciou-se o novo thread por meio da chamada da função **start()**;
- O método **start()** na função **main()** inicia a execução do novo thread por meio da chamada da função **run()** codificada em **PrintUSCS**.

# Execução Concorrente

- O comportamento agora é um pouco diferente;
- O método start() chama o método run() e inicia um novo thread.




# Alguns métodos da classe Thread

- **Thread()** -> Constrói um objeto thread usando valores default para todas opções.
- **void run()** -> Método que inicia a execução do novo thread criado. Desenvolvedores implementam override deste método.
- **void start()** -> Cria um novo thread e executa o método **run()**.

# run() x main()

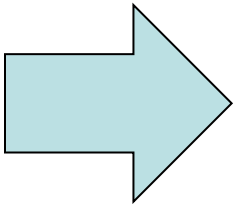
- Em essência, o método **run()** pode ser imaginado como sendo o método **main()** do novo thread criado;
- O novo thread inicia a execução com o método **run()**, da mesma forma que um programa usual inicia sua execução com o método **main()**;
- O novo thread pode receber parâmetros do thread principal.

# Área de Dados do Thread

- Cada thread possue sua própria área de dados;
- Essa área é isolada da área de outros threads.

# Área de Dados do Thread

Área de Dados da Thread



```
package br.uscs.threads;

public class ThreadArea extends Thread {

    // area de dados da Thread
    private String area;
    private int contador;

    //Construtor da Thread
    public ThreadArea(String area) {
            this.area = area;
            this.contador = 0;
    }
```

# Área de Código do Thread

```
// area de código da Thread
public void run() {
        while (this.contador < 50) {
                System.out.println("Contador: " +
                        contador +
                        " de " + this.area);
                this.contador++;
        }
    }
}
```

# Execução da Thread

```
package br.uscs.threads;

public class TesteThreadArea {

	public static void main(String[] args) {
		ThreadArea thArea1 = new ThreadArea( "USCS...");
		thArea1.start();

		ThreadArea thArea2 = new ThreadArea( "Computação...");
		thArea2.start();
	}

}
```

# Log de Execução da Thread

Problems | Javadoc | Declaration | Console

<terminated> TesteThreadArea [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 25, 2020, 3:53:04 PM)

```
Contador: 0 de USCS...
Contador: 0 de Computação...
Contador: 1 de USCS...
Contador: 2 de USCS...
Contador: 3 de USCS...
Contador: 4 de USCS...
Contador: 5 de USCS...
Contador: 6 de USCS...
Contador: 7 de USCS...
Contador: 1 de Computação...
Contador: 8 de USCS...
Contador: 2 de Computação...
Contador: 9 de USCS...
Contador: 10 de USCS...
```

# Atribuindo nomes às Threads

- Ao se criar um thread pode-se dar nome a ele;
- Este nome irá diferenciá-lo de outros threads em execução;
- Pode-se associar nomes às threads por meio da função Thread.setName();
- O nome pode ser recuperado por meio da função Thread.getName().

# Atribuindo nomes às Threads

```
Thread thread = new Thread("New Thread") {
   public void run(){
     System.out.println("run by: " + getName());
   }
};


thread.start();
System.out.println(thread.getName());
```

- Nesse exemplo, o string "New Thread" passado como parâmetro para o construtor é o nome do thread;
- O nome também pode ser obtido por meio do método getName().

# Atribuindo nomes às Threads

```
package br.uscs.threads;

public class ThreadName extends Thread {

        // area de dados da Thread
        private int contador;

        //Construtor da Thread
        public ThreadName() {
                this.contador = 0;
}
```

# Atribuindo nomes às Threads

```
// area de código da Thread
 public void run() {

        this.setName("MinhaThread...");

        while (contador <= 50) {
               System.out.println(this.getName() + "\t" + contador);
               contador++;
        }
    }
}
```

# Classe para execução

```
package br.uscs.threads;

public class TesteThreadName {

        public static void main(String[] args) {

                ThreadName thName1 = new ThreadName();
                thName1.start();
        }

}
```

# Log de Execução

Problems @ Javadoc Declaration Console

<terminated> TesteThreadName [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 25, 2020, 4:02:29 PM)

```
MinhaThread...  0
MinhaThread...  1
MinhaThread...  2
MinhaThread...  3
MinhaThread...  4
MinhaThread...  5
MinhaThread...  6
MinhaThread...  7
MinhaThread...  8
MinhaThread...  9
MinhaThread...  10
MinhaThread...  11
MinhaThread...  12
```

# Fazendo thread dormir....

- Pode-se forçar que uma determinada thread entre em espera (dormindo) por um certo período de tempo definido em milissegundos;
- Essa espera poder ser feita pela chamada da função Thread.sleep(tempo);

# Fazendo thread dormir....

```
package br.uscs.threads;

public class MinhaThread extends Thread {

        // area de dados da Thread
        private int contador;
        private int limite;
        private int tempoSleep;

        //Construtor da Thread
                public MinhaThread(int limite, int tempoSleep) {
                        super();
                        this.contador = 0;
                        this.limite = limite;
                        this.tempoSleep = tempoSleep;
        }
```

# Fazendo thread dormir....

```
// area de código da Thread
 public void run() {

        while (contador <= limite) {
                System.out.println(super.getName() + "\t" + contador);
                contador++;
                // codigo para colocar a Thread para "dormir" 1 segundo
                try {
                        Thread.sleep(tempoSleep);
                }
                catch( InterruptedException e ) {
                        e.printStackTrace(System.err);
                }
        }
    }
}
```

# Classe para execução

```
package br.uscs.threads;

public class TesteMinhaThread {

        public static void main(String[] args) {
                MinhaThread mt1 = new MinhaThread( 20, 1500 );
                mt1.setName("MinhaThread1....");
                mt1.start();

                MinhaThread mt2 = new MinhaThread( 20, 1000 );
                mt2.setName("MinhaThread2....");
                mt2.start();

                MinhaThread mt3 = new MinhaThread( 15, 500 );
                mt3.setName("MinhaThread3....");
                mt3.start();
```

# Classe para execução

```
        MinhaThread mt4 = new MinhaThread( 25, 2000 );
        mt4.setName("MinhaThread4....");
        mt4.start();

        MinhaThread mt5 = new MinhaThread( 25, 400 );
        mt5.setName("MinhaThread5....");
        mt5.start();

    }

}
```

# Log de execução

Problems | Javadoc | Declaration | Console

<terminated> TesteMinhaThread [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 25, 2020, 4:11:47 PM)

```
MinhaThread3....          0
MinhaThread5....          0
MinhaThread4....          0
MinhaThread5....          1
MinhaThread3....          1
MinhaThread5....          2
MinhaThread3....          2
MinhaThread2....          1
MinhaThread5....          3
MinhaThread3....          3
MinhaThread1....          1
```

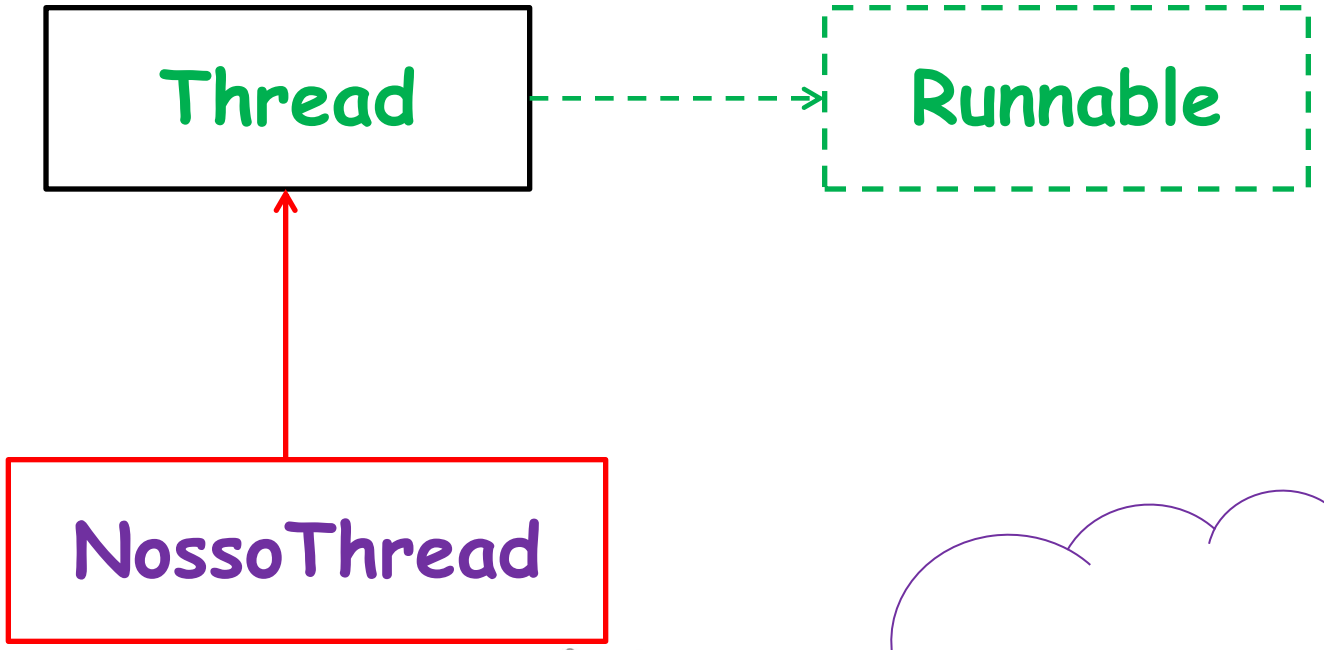
# Já vimos que ...

- Frequentemente, temos a necessidade de implementar **nossos próprios** threads;
- Toda aplicação Java **possui ao menos um thread** que corresponde ao método **main()**;
- Adicionalmente, a partir do método **main()** pode-se criar outros threads para implementar programação concorrente;
- O código do novo thread deve ser implementado no método **run()** o qual especifica então, o **comportamento** do novo thread;
- A classe Thread implementa um thread cujo método **run()** inicialmente está vazio;
- Assim, cabe ao programador fazer override do método **run()** e definir o código (comportamento) do thread.

# Formas de se implementar Threads

- Com a Linguagem Java, há duas formas de se implementar Threads;
- A primeira, já vimos, e corresponde ao se implementar uma nova classe que é filha da classe Thread e, em seguida, fazer-se overriding no método run() que conterá o comportamento (código) do novo Thread;
- Por meio do construtor daremos nome ao novo Thread;
- Pode-se também, em tempo de execução, dar um novo nome ao Thread, por meio da função Thread.setname();
- A recuperação do nome do thread pode ser feita pela chamada da função Thread.getname();
- Para iniciar a execução do novo thread, usamos a função start();
- A função start(), na verdade, chamará a execução da função run().

# Formas de se implementar Threads




Nosso thread, nesse caso, corresponde a uma classe que é **filha** da classe Thread !

# Formas de se implementar Threads

- Temos também com a Linguagem Java, uma outra forma de se implementar Threads;
- Por meio da interface **Runnable**.

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.lang

## Interface Runnable

**All Known Subinterfaces:**

RunnableFuture<V>, RunnableScheduledFuture<V>

**All Known Implementing Classes:**

AsyncBoxView.ChildState, ForkJoinWorkerThread, FutureTask, Re

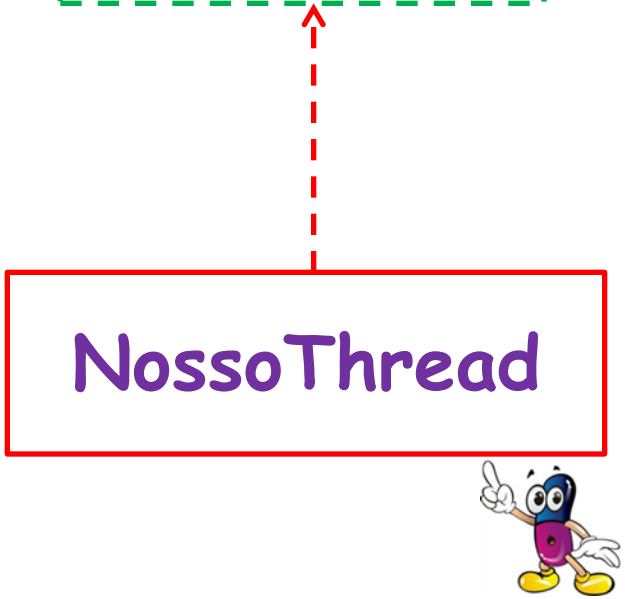
**Functional Interface:**

This is a functional interface and can therefore be used as t

# Interface Runnable

- Para isso, a classe que desejamos que seja executada como thread deve definir e implementar um método (sem argumentos) chamado run().


Nosso thread, nesse caso, corresponde a uma classe que implementa a interface **Runnable** !




# Interface Runnable - Exemplo

```
package br.uscs.threads;

public class MinhaThread2 implements Runnable {

// area de dados da Thread
private int contador;
private int limite;
private int tempoSleep;

//Construtor da Thread
public MinhaThread2(int limite, int tempoSleep) {
        super();
        this.contador = 0;
        this.limite = limite;
        this.tempoSleep = tempoSleep;
}
```

# Interface Runnable - Exemplo

```
// area de código da Thread
 public void run() {

   while (contador <= limite) {
        System.out.println(Thread.currentThread().getName() +
                           "\t" + contador);
        contador++;
        // codigo para colocar a Thread para "dormir" 1 segundo
        try {
                Thread.sleep(tempoSleep);
        }
        catch( InterruptedException e ) {
        e.printStackTrace(System.err);
        }
   }
 }
}
```

# Interface Runnable - Exemplo

```
package br.uscs.threads;

public class TesteMinhaThread2 {

        public static void main(String[] args) {
                MinhaThread mt1 = new MinhaThread( 20, 1500 );
                mt1.setName("MinhaThread1....");
                mt1.start();

                MinhaThread mt2 = new MinhaThread( 20, 1000 );
                mt2.setName("MinhaThread2....");
                mt2.start();
```

# Interface Runnable - Exemplo

```
        MinhaThread mt3 = new MinhaThread( 15, 500 );
        mt3.setName("MinhaThread3....");
        mt3.start();

        MinhaThread mt4 = new MinhaThread( 25, 2000 );
        mt4.setName("MinhaThread4....");
        mt4.start();

        MinhaThread mt5 = new MinhaThread( 25, 400 );
        mt5.setName("MinhaThread5....");
        mt5.start();

    }

}
```

# Interface Runnable - Execução

Problems @ Javadoc Declaration Console

<terminated> TesteMinhaThread2 [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 26, 2020,

```
MinhaThread1....          0
MinhaThread3....          0
MinhaThread2....          0
MinhaThread5....          0
MinhaThread4....          0
MinhaThread5....          1
MinhaThread3....          1
MinhaThread5....          2
MinhaThread2....          1
MinhaThread3....          2
```

# Thread.currentThread()

- No exemplo anterior foi utilizado o método **Thread.currentThread()**;
- Este método retorna uma **referência** ao objeto Thread que está sendo executado;
- Isto permite que se acesse o objeto **Thread** representando o thread em execução;
- Este método permite que outros **atributos** do Thread sejam recuperados, conforme mostrado adiante.

# Método getId()

- O método getId() é usado para se retornar o identificador único do thread;
- O Thread Id é um número único positivo o qual é gerado durante a instanciação do Thread;
- O Thread Id permanece inalterado durante sua vida útil;
- Quando o thread é terminado, o iD do thread pode ser reusado.

# Método getId()

```
package br.uscs.threads;

public class MinhaThread3 extends Thread {

// area de dados da Thread
private int contador;
private int limite;
private int tempoSleep;

//Construtor da Thread
public MinhaThread3(int limite, int tempoSleep) {
        super();
        this.contador = 0;
        this.limite = limite;
        this.tempoSleep = tempoSleep;
}
```

# Método getId()

```
// area de código da Thread
 public void run() {

    while (contador <= limite) {
       System.out.println(Thread.currentThread().getName() +
              "\t" + contador);

       contador++;
       // codigo para colocar a Thread para "dormir" 1 segundo
       try {
              Thread.sleep(tempoSleep);
       }
       catch( InterruptedException e ) {
              e.printStackTrace(System.err);
       }
    }
  }
}
```

# Método getId()

```
package br.uscs.threads;

public class TesteMinhaThread3 {

public static void main(String[] args) {
        MinhaThread3 mt1 = new MinhaThread3( 3, 10 );
        mt1.setName("MinhaThread1....");
        System.out.println("Nome do Thread mt1: " +
        mt1.getName());
        System.out.println("Id do Thread mt1: " + mt1.getId() ) ;
        mt1.start();

        MinhaThread mt2 = new MinhaThread( 3, 10 );
        mt2.setName("MinhaThread2....");
        System.out.println("Nome do Thread mt2: " +
        mt2.getName());
        System.out.println("Id do Thread mt2: " + mt2.getId() ) ;
        mt2.start();
    }
}
```

# Método getId() - Execução

Problems | Javadoc | Declaration | Console

<terminated> TesteMinhaThread3 [Java Application] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Feb 26, 2020, 1:23:07 PM)

```
Nome do Thread mt1: MinhaThread1....
Id do Thread mt1: 10
MinhaThread1....          0
Nome do Thread mt2: MinhaThread2....
Id do Thread mt2: 11
MinhaThread2....          0
MinhaThread1....          1
MinhaThread2....          1
MinhaThread1....          2
MinhaThread2....          2
MinhaThread1....          3
```