****

**UNIVERSIDADE FEDERAL DE GOIÁS**

**ENGENHARIA DE SOFTWARE**

**SOFTWARE CONCORRENTE E DISTRIBUIDO – INF0298**

Discente:

Adriel Lenner Vinhal Mori

Docente:

Vagner Jose Do Sacramento Rodrigues

**Lab Thread 1 – 16 jun 2023**

Goiânia

2023

**1.**

public class ThreadSimples implements Runnable {

public void run() {

System.out.println("Olá de uma thread!");

}

}

public class ExecutaThread {

// Nova classe denominada ExecutaThread com apenas o método main()

public static void main(String[] args) {

ThreadSimples simples = new ThreadSimples();

simples.run();

}

}

**2.**

public class ExecutaThread {

public static void main(String[] args) {

// Nova classe denominada ExecutaThread com apenas o método main()

ThreadSimples simples = new ThreadSimples();

simples.run();

}

}

class ThreadSimples implements Runnable {

public void run() {

System.out.println("Hello from a thread!");

}

}

**3.**

public class ThreadSimples implements Runnable {

public static void main(String args[]) throws InterruptedException {

String info[] = {

"Java",

"é uma boa linguagem.",

"Com threads",

"é melhor ainda.",

};

// Cria uma instância da classe ThreadSimples

ThreadSimples simples = new ThreadSimples();

// Cria uma nova Thread e passa a instância de ThreadSimples como parâmetro

Thread thread = new Thread(simples);

// Inicia a execução da nova thread

thread.start();

for (int i = 0; i < info.length; i++) {

Thread.sleep(10000);

System.out.println(info[i]);

}

}

// Método run() da interface Runnable

public void run() {

// Obtém o nome da thread principal

String nomeThreadPrincipal = Thread.currentThread().getName();

System.out.println(nomeThreadPrincipal);

}

}

**4.**

public class ThreadSimples {

// Método para exibir mensagens na saída padrão

static void mensagem(String mensagem) {

// Obtém o nome da thread atual

String nomeThread = Thread.currentThread().getName();

// Imprime o nome da thread e a mensagem

System.out.println(nomeThread + " " + mensagem);

}

// Classe interna que implementa a interface Runnable

private static class Loop implements Runnable {

public void run() {

// Array de mensagens

String info[] = {

"Java",

"é uma boa linguagem.",

"Com threads,",

"é melhor ainda."

};

try {

for (int i = 0; i < info.length; i++) {

// Pausa a execução da thread por 4 segundos

Thread.sleep(4000);

// Chama o método mensagem para exibir a mensagem atual

mensagem(info[i]);

}

} catch (InterruptedException e) {

// Caso a thread seja interrompida enquanto está dormindo

mensagem("Nada feito!");

}

}

}

public static void main(String args[]) throws InterruptedException {

// Define o tempo de paciência inicial para 1 hora

long paciencia = 1000 \* 60 \* 60;

if (args.length > 0) {

try {

// Se um argumento for fornecido, converte-o para um valor inteiro e define como tempo de paciência

paciencia = Long.parseLong(args[0]) \* 1000;

} catch (NumberFormatException e) {

// Caso o argumento fornecido não seja um número inteiro válido

System.err.println("Argumento deve ser um inteiro.");

System.exit(1);

}

}

// Inicia uma nova thread com a classe Loop como Runnable

mensagem("Iniciando a thread Loop");

long inicio = System.currentTimeMillis();

Thread t = new Thread(new Loop());

t.start();

mensagem("Esperando que a thread Loop termine");

while (t.isAlive()) {

mensagem("Ainda esperando...");

t.join(1000);

// Verifica se o tempo de paciência foi excedido e a thread ainda está em execução

if (((System.currentTimeMillis() - inicio) > paciencia) && t.isAlive()) {

mensagem("Cansado de esperar!");

// Interrompe a thread

t.interrupt();

t.join();

}

}

mensagem("Finalmente!");

}

}

DESCRIÇÃO:

O código acima mostrado no exercício 4 cria uma nova thread para executar uma função run() da classe Loop que implementa a interface Runnable, estando em paralelo com a thread principal instanciada na classe main. A thread principal aguarda a conclusão da thread *t* por um tempo determinado. Se o tempo de paciência for excedido, a thread *t* é interrompida e o programa continua sua execução.

Sabendo disto, podemos demostrar que:

* Dentro da classe *ThreadSimples*, há uma classe interna chamada *Loop* que implementa a interface Runnable. Essa classe possui o método run(), que será executado em uma thread separada quando for iniciada dentro da classe main.
* A função run() contém um array de mensagens e um loop que itera sobre essas mensagens. A cada iteração, a execução da thread é pausada por 4 segundos usando e a mensagem atual é exibida chamando o método mensagem.
* A função main recebe argumentos por linha de comando, caso sejam fornecidos. Se um argumento for fornecido, ele é convertido em um valor inteiro e utilizado como tempo de paciência, medido em milissegundos.
* É exibida a mensagem "Iniciando a thread Loop" chamando o método mensagem. Em seguida, é obtido o tempo atual em milissegundos para calcular o tempo decorrido.
* Uma nova thread é iniciada com a classe *Loop* como objeto Runnable. Essa nova thread é armazenada na variável *t* e iniciada chamando o método start.
* É exibida a mensagem "Esperando que a thread Loop termine" chamando o método mensagem. Em seguida, entra em um loop enquanto a thread *t* estiver em execução.
* Dentro do loop, é exibida a mensagem "Ainda esperando..." chamando o método mensagem e a thread principal aguarda 1 segundo usando *t.join(1000)*. Isso significa que a thread principal espera por um segundo até que a thread *t* termine sua execução ou até que esse tempo expire.
* Verifica-se por meio da condicional se o tempo de paciência foi excedido e se a thread *t* ainda está em execução. Caso isso ocorra, é exibida a mensagem "Cansado de esperar!" chamando o método mensagem. A thread *t* é interrompida chamando o método *interrupt()*, e o método *join()* é chamado novamente para garantir que a thread termine sua execução.
* Quando a thread *t* não está mais em execução, ou seja, o loop é interrompido, é exibida a mensagem "Finalmente!" chamando o método mensagem.