iscte
INSTITUTO
UNIVERSITÁRIO
DE LISBOA

emprego digital

Módulo 4: Laboratório de Programação

Aula 1
Introdução à Programação Concorrente

UPskill

Sobre o módulo

O módulo de Laboratório de Programação pretende aprofundar e agregar os conhecimentos dos módulos anteriores.

Os conteúdos programáticos vão passar pela programação concorrente, arquiteturas de software, programação web e um projeto.

Conteúdos programáticos

- **Programação Concorrente em Java**
  - Threads
  - Cadeados e Barreiras
  - Sincronização e Coordenação
- **Programação Web em Java**
  - Arquiteturas de Software
  - Java Spring e JSP
  - HTML
  - CSS

Avaliação do módulo

Teste - Programação Concorrente Classificado de A-E	Aula 7 08 / 02 / 2021	25%
Avaliação Contínua Classificado de A-E	Todas as semanas quando o projecto começar	10%
Projeto Classificado de A-E	Aula 36 19 / 03 / 2021	55%
Apresentação Projeto Classificado de A-E	Aula 37 22 / 03 / 2021	10%

iscte INSTITUTO UNIVERSITÁRIO DE LISBOA

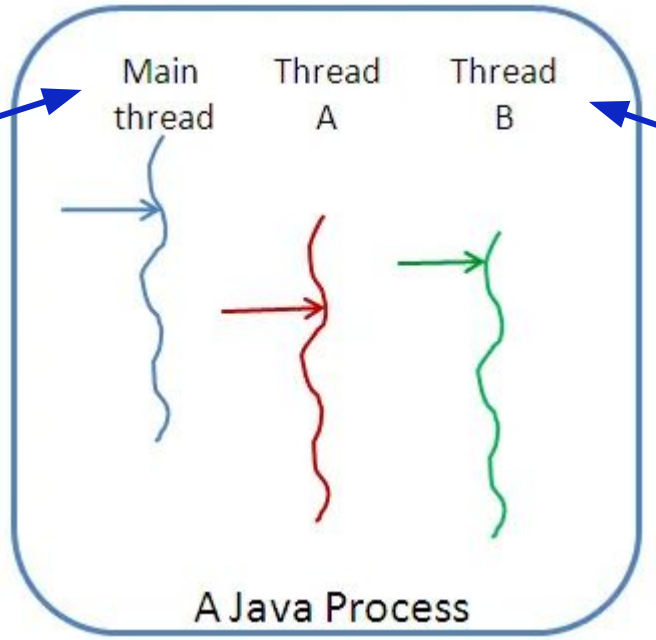
emprego digital

O que é a Programação Concorrente?

- **Programação Concorrente** é um paradigma de programação usado no desenvolvimento de programas que executam várias tarefas em simultâneo.
- No fundo, várias partes de código que se são executadas em paralelo.
- Cada linhas de execução é conhecidas como uma **Thread**.

O que é a Programação Concorrente?

Na programação clássica, temos apenas a Thread do main()



Em programação concorrente, podemos ter mais processos a serem executados em simultâneo

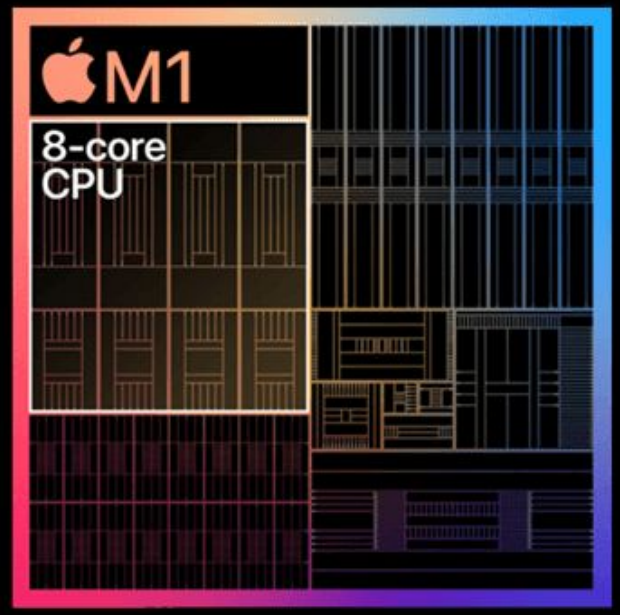
Vantagens da Programação Concorrente

- A principal vantagem do uso de programação concorrente é o **aumento do desempenho**.
- Ao executar vários processos em simultâneo, tiramos também partido dos vários **cores** de um processador.

Vantagens da Programação Concorrente

- Por exemplo: um processador com 8 cores pode executar um programa com 8 threads concorrentes, **que são executadas de forma 100% paralela**.

- Um processador single core também pode correr threads, mas nesse caso vai dividir a atenção do CPU.



Vantagens da Programação Concorrente

- **Exemplo prático**: numa grande empresa, podemos calcular a faturação mensal da empresa executando o cálculo da faturação de forma concorrente, processando várias filiais ao mesmo tempo.
- Neste caso, a programação concorrente permite ganho de tempo pois todas as filiais são processadas ao mesmo tempo, ao invés de processar a faturação de uma filial de cada vez. Esta abordagem é muito usada.

O que é uma Thread?

- Uma *thread* é um pequeno programa que trabalha como um subprocesso.
- Pense numa *thread* como uma **sequência de comandos a ser executados num programa**.
- Se tiver duas *threads*, terá duas sequências de comandos executando ao mesmo tempo no mesmo programa ou processo.
- Mas atenção! Executar o mesmo programa duas vezes não é criar mais *threads*, mas sim criar dois processos do mesmo programa. *Threads* executam concorrentemente num mesmo processo (programa). Tal como os processos executam concorrentemente num sistema operativo.

Porquê várias threads?

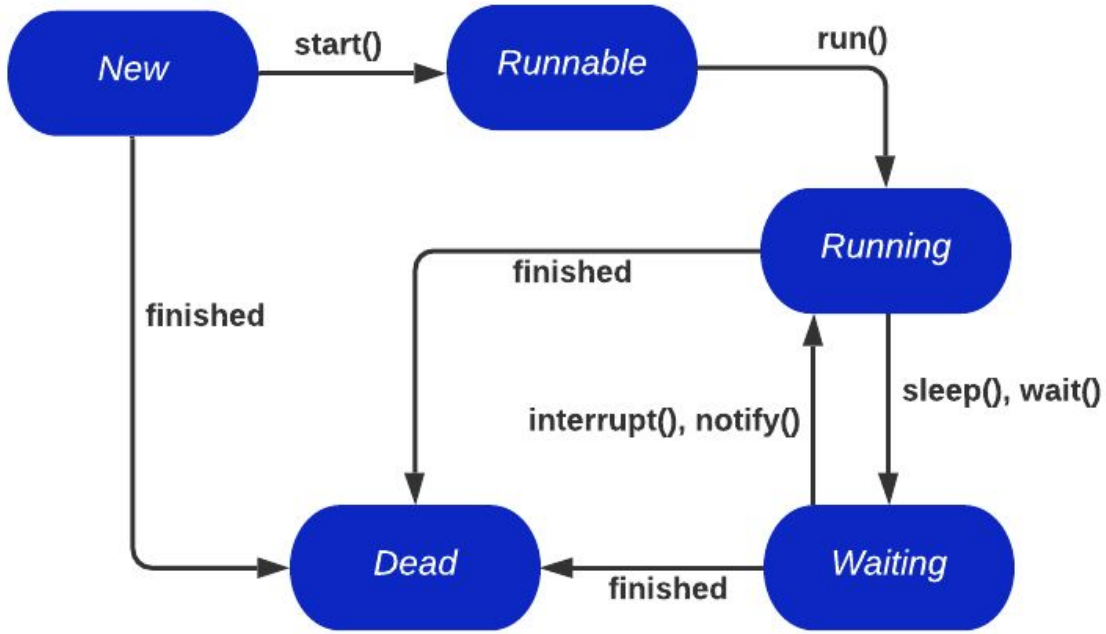
Num programa, podemos utilizar várias threads para:

- Cálculos longos;
- Tratar pedidos (no caso de um servidor);
- Verificação ortográfica (em simultâneo com a introdução de texto);

Assim, o programa pode executar as 3 tarefas ao mesmo tempo, para que seja possível tratar pedidos de um servidor, ao mesmo tempo que é feito um cálculo e é feita uma correcção ortográfica.

Tudo acontece em simultâneo!

Ciclo de vida de uma thread



Cenários de execução de threads

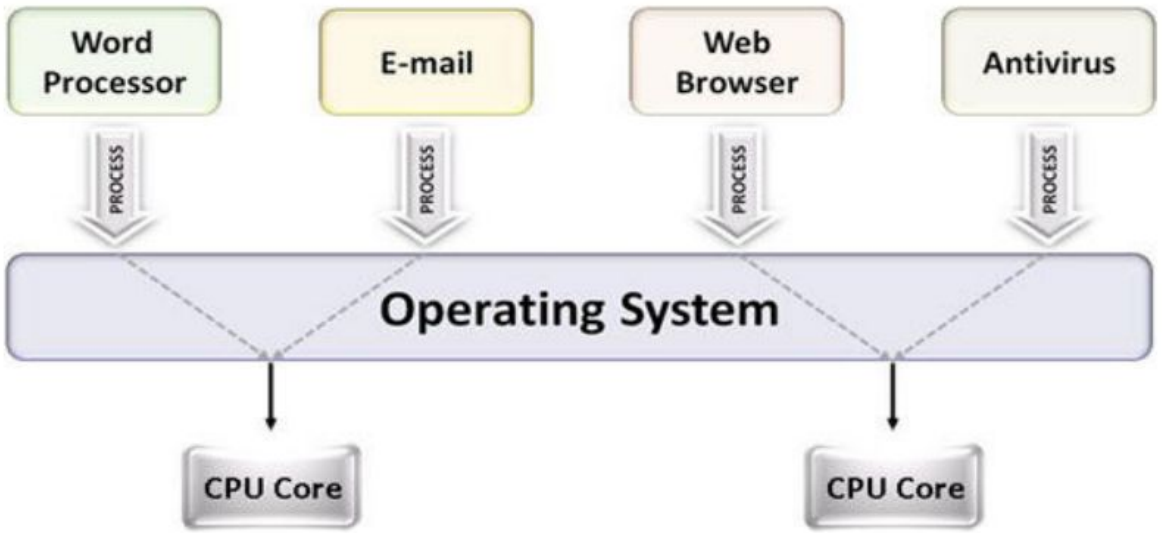
A execução de diferentes threads pode ocorrer em 3 cenários diferentes:

- As threads são executadas em paralelo se houver mais que um processador ou o processador for multi-core;
- As threads são executadas de maneira intercalada num processador, se apenas existir um processador single-core;
- Também é possível ter um cenário híbrido. **Esta gestão fica a cargo do processador e não do programador**.

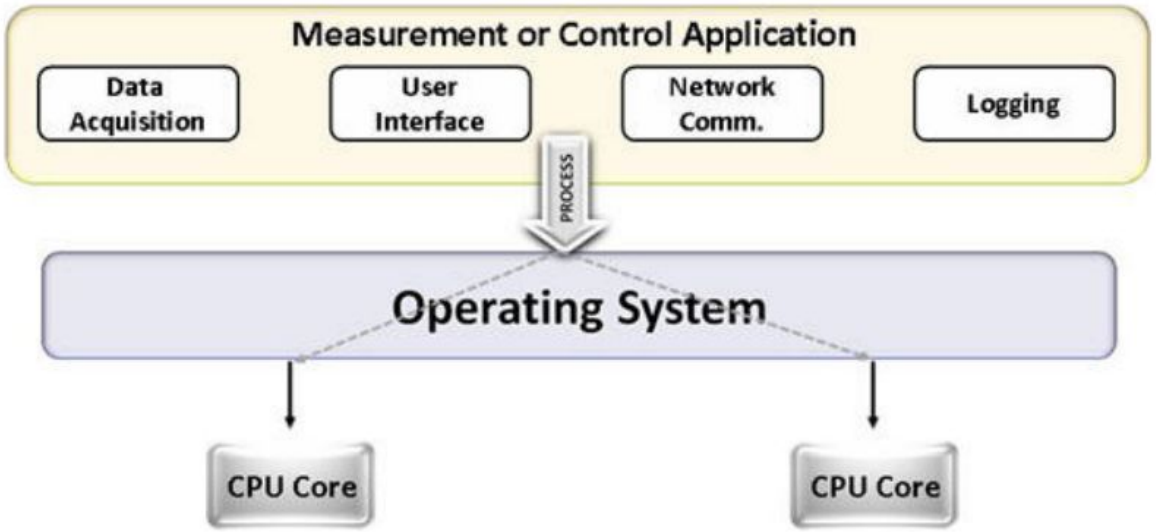
Multitasking vs Multithreading

- **Multitasking**: Um processo (por exemplo, uma aplicação) é executado num espaço de memória separado. Os vários processos podem correr ao mesmo tempo, mas um processo não pode manipular a memória alocada a outro processo.
- **Multithreading**: Um processo pode criar várias threads e elas podem ser executadas no mesmo espaço de memória. O que significa que todas as threads têm acesso aos mesmo objectos, ficheiros, etc.

Multitasking



Multithreading



iscte INSTITUTO UNIVERSITÁRIO DE LISBOA
emprego digital

Diferenças entre processo e thread

- Criar uma thread requer menos recursos do que criar um processo.
- As threads existem dentro de um processo - todos os processos têm pelo menos uma thread.
- Todas a threads compartilham os recursos de um processo.
- **Um processo é, por exemplo, um programa em Java.**
    - **Dentro do nosso programa, podemos ter uma ou mais *threads*.**

Desafios da programação concorrente

- A criação de programas concorrentes (com várias *threads*), acarreta um grande conjunto de desafios:
  - O que acontece se duas threads acederem aos mesmos objetos, ao mesmo tempo? (Ex. uma thread insere um elemento em uma lista enquanto outra thread remove outro elemento)
  - Como podemos coordenar as várias threads? (Ex. uma thread esperar por outra)
- Todos estes desafios serão estudados hoje e nas próximas aulas.

Multithreading em Java

- Cada thread começa a sua execução num local específico.

- Cada thread é executada independentemente de outras threads no programa, no entanto existem mecanismos que permitem as threads de colaborar.

Como criar uma Thread em Java?

- A classe Thread fornece construtores e métodos para criar e executar operações numa thread.
- A nossa thread deve estender a classe “Thread” do Java.
- O método run() é implementado para realizar as ações de uma thread.
- O método start() inicia a execução de uma thread e é chamado o método run.

```
public class Exemplo1 extends Thread {

public void run(){
   System.out.println(“Hello World!!!”);
}

public static void main(String [] args){
   Thread t = new Exemplo1();
   t.start();
}

}
```

iscte INSTITUTO UNIVERSITÁRIO DE LISBOA emprego digital

Exercício 1

Criar uma thread que recebe um número na consola e diz se ele é par ou ímpar.

Resolução Exercício 1

```
public class Exercicio1 extends Thread{

    public void run(){
        Scanner s = new Scanner(System.in);
        System.out.println(“Digite o número escolhido”);
        int valor=s.nextInt();
        if(valor%2==0){
            System.out.println(“É par”);
        }else{  System.out.println(“É impar”);}
    }
    public static void main(String [] args){
        Thread t=new Exercicio1();
        t.start();
    }
}
```

Classe Thread em Java

- Cada thread está associada a uma instância da classe Thread.
- Um programa que crie uma ou mais threads, necessita de implementar o código a executar nessas threads (implementação do método run()).
- Existem duas maneiras de especificar esse código:
  - Como no slide anterior, estendendo a classe Thread.
  - Ou implementar a interface Runnable.

Interface Runnable

- A interface Runnable deve ser implementada por qualquer classe cujas instâncias sejam executadas por uma thread.
- Cria uma instância da classe Thread e passa o seu objeto Runnable para o seu construtor como parâmetro.
- **É uma outra possibilidade para criar uma Thread.**

```
public class Exemplo2 implements Runnable{

@Override
public void run(){
   System.out.println("Hello World!!!");
}
public static void main(String [] args){
   Runnable omt=new Exemplo2();
   Thread t=new Thread(omt);
   t.start();
}
}
```

Exercício 2

Fazer o Exercício 1 implementando a classe Runnable.

Resolução Exercício 2

```
public class Exercicio2 implements Runnable{

    @Override
    public void run(){
        Scanner s=new Scanner(System.in);
        System.out.println(“DIgite o número escolhido”);
        int valor=s.nextInt();
        if(valor%2==0){
        System.out.println(“É par”);
        }else{  System.out.println(“É impar”);}
    }
    public static void main(String [] args){
        Runnable omt=new Exercicio2();
        Thread t=new Thread(omt);
        t.start();
    }
}
```

O método Sleep em threads

- O *sleep()* é um método estático que coloca em pausa a execução de uma thread por um determinado período de tempo.
- Isso deixa mais tempo para as outras threads e outros processos serem executados.
- A duração do sleep é especificada em milissegundos.
- Pode lançar uma *InterruptedException* quando outra thread interrompe uma thread que está a dormir.

Exemplo Sleep

```
public class Exemplo3 extends Thread{

public void run(){
   for(int i=0;i<10;i++){
       System.out.println(“Hello World!!!”+i);
       try{ sleep(4000);}
       catch(InterruptedException e){ }
   }
}
public static void main(String [] args){
   Thread t=new Exemplo3();
   t.start();
}
}
```

Exercício 3

Altere o código do exercício 1, para ser pedido um número ao utilizador 10 vezes. Depois de ser dito se o número é par ou ímpar, a thread entra num sleep de 3 segundos e volta a pedir outro número.

Resolução Exercício 3

```
public class Exercicio3 extends Thread{

    public void run(){
        Scanner s=new Scanner(System.in);
        for(int i=0;i<=9;i++){
            System.out.println("Digite o número escolhido");
            int valor=s.nextInt();
            if(valor%2==0){
                System.out.println("É par");
            }else{
                System.out.println("É impar");
            }
            try{ sleep(3000);}
            catch(InterruptedException e){ e.printStackTrace();}
        }
    }
}
```

O método Interrupted em threads

Uma thread pode ser interrompida invocando o método *interrupt()* em uma outra *thread*, para que a mesma seja interrompida:

- **void interrupt()** - este método interrompe um thread;
- **boolean isInterrupted()** - este método verifica se uma thread foi interrompida ou não;

Uma interrupção não pára a thread nem a coloca em pausa. A interrupção apenas indica que algo aconteceu e ela pode ter que fazer algo de diferente;

Exemplo Interrupted

```
public class Exemplo4 extends Thread {
     public void run() {
          String alfabeto[] = { "A", "B", "C", "D" };
          for (int i = 0; i < alfabeto.length; i++) {
               try {
                    // Pause for 4 seconds
                    System.out.println(currentThread() + ": sleep for 4 seconds");
                    sleep(4000);
                    // Print a message
                    System.out.println("\t" + alfabeto[i]);
               } catch (InterruptedException e) {
                    System.out.println(currentThread() + ": Ops! I was interrupted!");
               }
          }
     }

     public static void main(String args[]) throws InterruptedException {
          Thread t = new Exemplo4();
          t.start();
          int pausa = (new Random()).nextInt(16000);
          System.out.println(currentThread() + ": sleep for " + pausa / 1000 + " seconds");
          sleep(pausa);
          t.interrupt();
     }
}
```

O que acontece quando a thread é interrompida

Interrupção da Thread

O método Join em threads

- A classe Thread fornece o método join() que permite que uma thread espere que outra thread termine a sua execução.
- Se *t* for um objeto *Thread* e está em execução, então *t.join()* vai garantir que *t* acaba antes que a própria thread continue. **Ou seja, a thread que invoca o *t.join()* vai ficar em espera até que *t* termine.**
- O método *t.join(mills)* coloca a thread em espera até que **o tempo especificado (millis) passe** ou a thread *t* termine, e apenas depois continua.

Exemplo Join

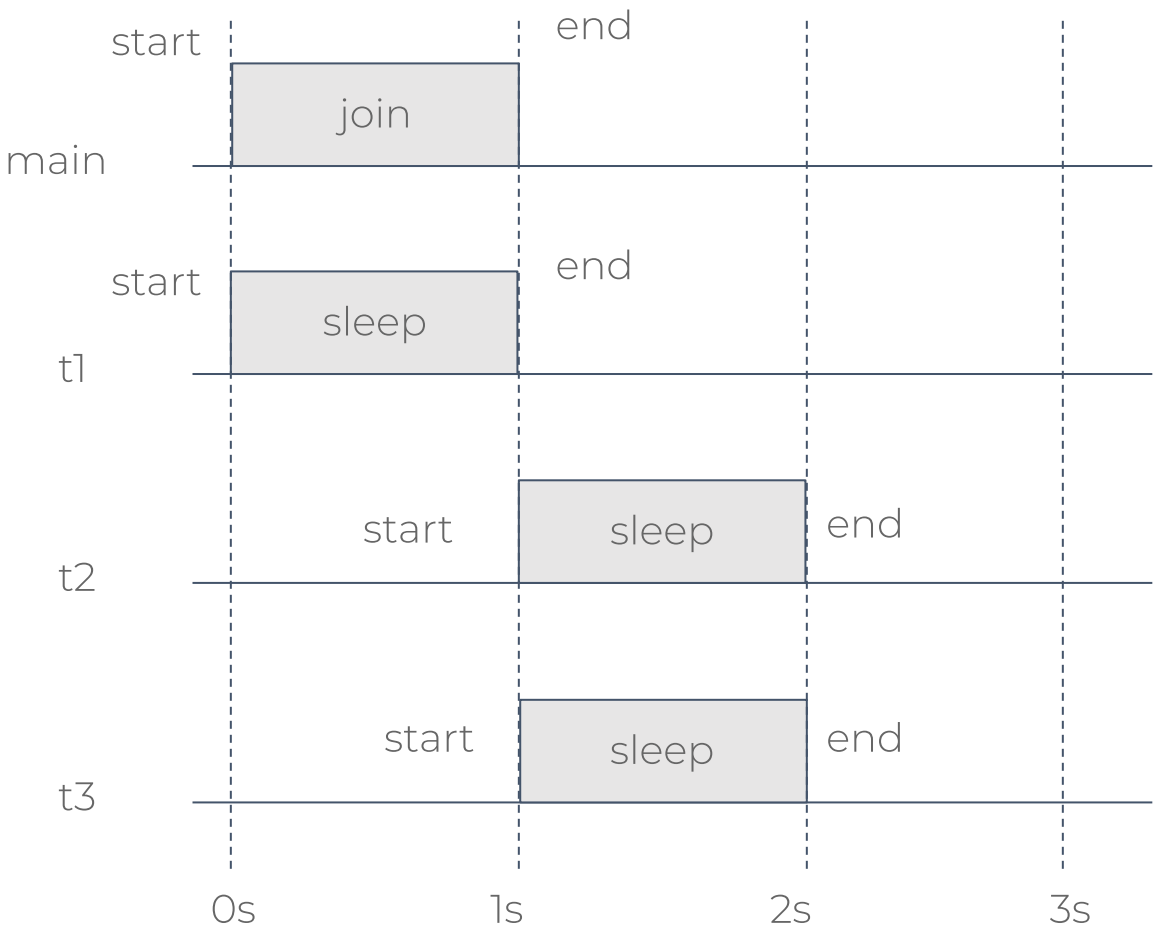
```
public class Exemplo5 extends Thread {
     String name;
     public Exemplo5(String name) { this.name = name; }
     public void run() {
          System.out.println(name + " started!");
          try {
               Thread.sleep(1000);
          } catch (InterruptedException e) { }
          System.out.println(name + " ended!");
     }
}
public class Main {
     public static void main(String[] args) throws InterruptedException {
          Thread t1 = new Exemplo5("t1"); Thread t2 = new Exemplo5("t2"); Thread t3 = new Exemplo5("t3");
          t1.start();
          t1.join(2000);
          t2.start();
          t3.start();
          System.out.println("Exiting main thread");
     }
}
```

Resultado na consola:

```
t1 started!
t1 ended!
Exiting main thread
t2 started!
t3 started!
t2 ended!
t3 ended!
```

Podem ocorrer por outra ordem

O que está a acontecer?

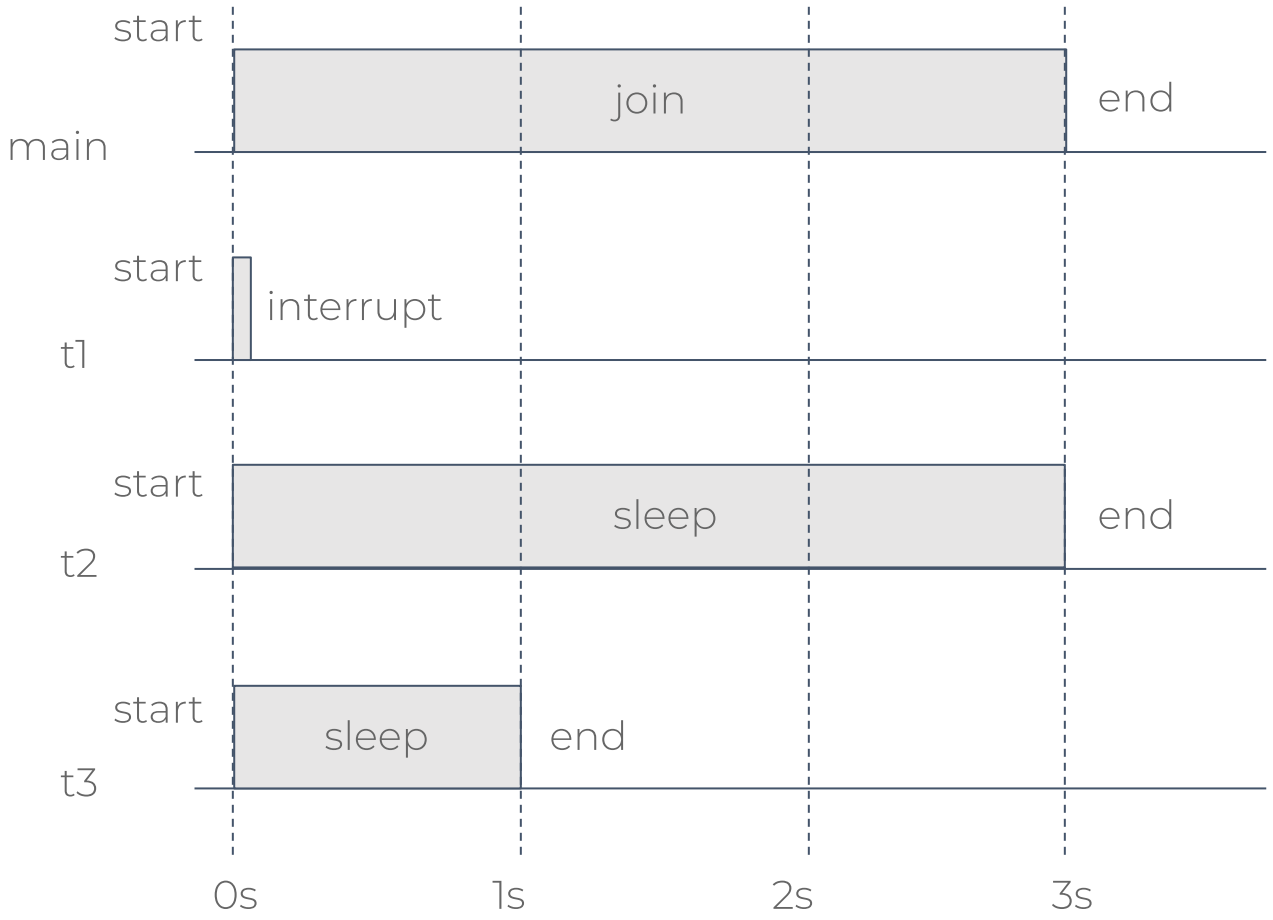


Exercício 4

Indique o que é escrito na consola decorrente da execução do seguinte código.

```
public class Exercicio4 extends Thread {
     double time; String name;
     public Exercicio4(double time, String name) { this.time = time; this.name = name; }
     public void run() {
          System.out.println(name + “ started!”);
          try {
               Thread.sleep(time * 1000);
          } catch (InterruptedException e) { }
          System.out.println(name + “ ended!”);
     }
}
public class Main {
     public static void main(String[] args) throws InterruptedException {
          Thread t1 = new Exercicio4(2, “t1”); Thread t2 = new Exercicio4(3, “t2”); Thread t3 = new Exercicio4(1,“t3”);
          t1.start();
          t1.interrupt();
          t2.start();
          t3.start();
          t2.join(1000);
          System.out.println("Exiting main thread");
     }
}
```

Resolução Exercício 4



Resultado na consola:

```
t1 started!
t3 started!
t2 started!
t1 ended!
t3 ended!
t2 ended!
Exiting main thread
```

Exercício 5

Crie um programa com duas threads que fazem coisas distintas. Uma thread imprime números pares e a outra imprime números ímpares. A thread dos pares imprime os números pares até 1000 e tem um sleep de 1s. A thread dos ímpares imprime os números ímpares até 150 e tem um sleep de 2s.

Resolução Exercício 5

```
public class ThreadOdd extends Thread{

    public void run(){
        for(int i=0;i<150;i++){
            try {
                if(i%2!=0) {
                    System.out.println(i);
                }
                sleep(2000);
            } catch (InterruptedException e) {}
        }
    }
}
```

```
public class ThreadEven extends Thread{

    public void run(){
        for(int i=0;i<1000;i++){
            try {
                if(i%2==0) {
                    System.out.println(i);
                }
                sleep(1000);
            } catch (InterruptedException e) {}
        }
    }
}
```

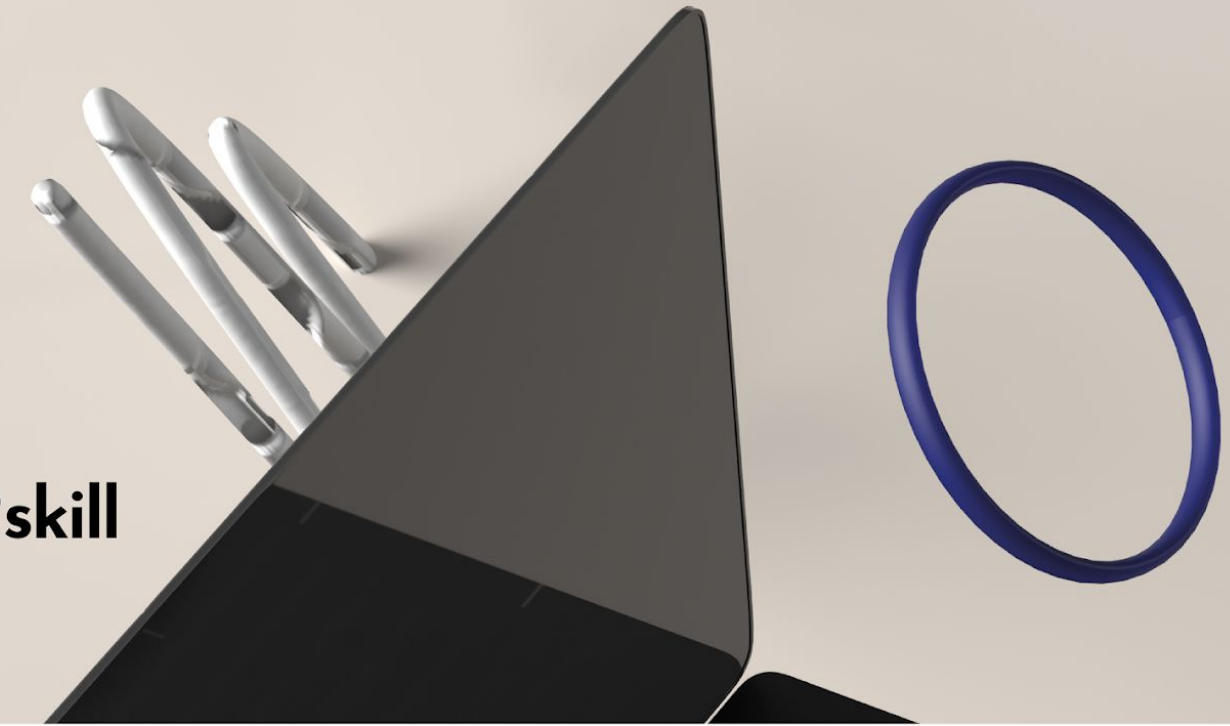
Resolução Exercício 5

```
public class Main {

   public static void main(String[] args) {
       Thread t1=new ThreadEven();
       Thread t2=new ThreadOdd();

       t1.start();
       t2.start();
   }
}
```

O futuro profissional começa aqui



iscte INSTITUTO UNIVERSITÁRIO DE LISBOA

emprego digital

UPskill