Processos

- Programas em execução
- Têm seu próprio **espaço de endereçamento**
- Competem por tempo de CPU
- É só olhar o gerenciador de tarefas

Threads

- “Processos leves”
- Espaço de endereçamento compartilhado
- Menor **unidade de escalonamento**
- Pelo menos uma por processo

Threads são mais leves

- Comunicação muito rápida
- Trocas de contexto auxiliadas por hardware
- Ocupam menos memória
- Criação e terminação mais rápidas

Em Java

- Extensão da classe `Thread` ou implementaçao da interface `Runnable`
- Nos dois casos, implementa-se o método

```
public void run()
```

  - Define o trabalho que será feito em uma thread separada
- A nova thread é iniciada através do método `start()`

```
public class HelloThread extends Thread {

  public void run() {
    System.out.println("Hi from a thread.");
  }

  public static void main(String args[]) {
    (new HelloThread()).start();
  }
}
```

```
public class HelloRunnable
    implements Runnable {

  public void run() {
    System.out.println("Hi from a thread.");
  }

  public static void main(String args[]) {
    (new Thread(new HelloRunnable())).start();
  }
}
```

Exercício

- Implemente um programa que imprime todos os números entre 1 e 2.000.000.000 usando várias threads para particionar o trabalho. Ao final da execução, seu programa deve dizer qual thread terminou sua parte primeiro. Seu programa deve usar no máximo 50 threads.

Fazendo uma thread esperar a outra terminar

- Método `join()`
- Chamado pela thread que deve esperar
  - A partir do objeto representando a thread pela qual esperar-se-á

```
public class HelloThreadThatWaits
    extends Thread {
  public void run() {
    System.out.println("Hi from a thread.");
  }
  public static void main(String args[]) {
    Thread t = new HelloThreadThatWaits();
    t.start();
    try { t.join(); //main thread waits for t to finish
    } catch(InterruptedException ie) {...}
    System.out.println("Hi from the MAIN thread.");
  }
}
```

Exercício

- Implemente um programa que calcula todos os números primos entre 1 e um valor *N* fornecido como argumento. Seu programa deve dividir o trabalho a ser realizado entre *X* threads (onde *X* também é uma entrada do programa) para tentar realizar o trabalho de maneira mais rápida que uma versão puramente sequencial.
- A thread principal do programa deve imprimir os números primos identificados apenas quando as outras threads terminarem.

Exercícios

- Construa um programa onde *N* threads incrementam contadores locais (não-compartilhados) em um laço. Cada thread imprime o valor de seu contador a cada incremento. As threads param quando seus contadores chegam a um valor limite *X*, recebido como entrada pelo programa.
  - Por que as saídas das threads se misturam?
- Agora mude seu programa para que as threads modifiquem um contador global compartilhado entre elas.
  - O que acontece com os resultados?

Threads podem dormir e ser interrompidas

- Como uma thread dorme? Método

```
public void sleep (long millis)
```

Pode lançar `InterruptedException`

- Interrupções: `interrupt()` e `interrupted()`

Exercícios

- Modifique o último programa que você construiu para que, a cada iteração, a thread espere 1ms. A thread que terminar a contagem primeiro (realizar todas as iterações) deve interromper todas as outras que estão executando e dizer quantas foram até o final.