AfroDev

Java Básico - Aula 7 - Parte 1 e 2

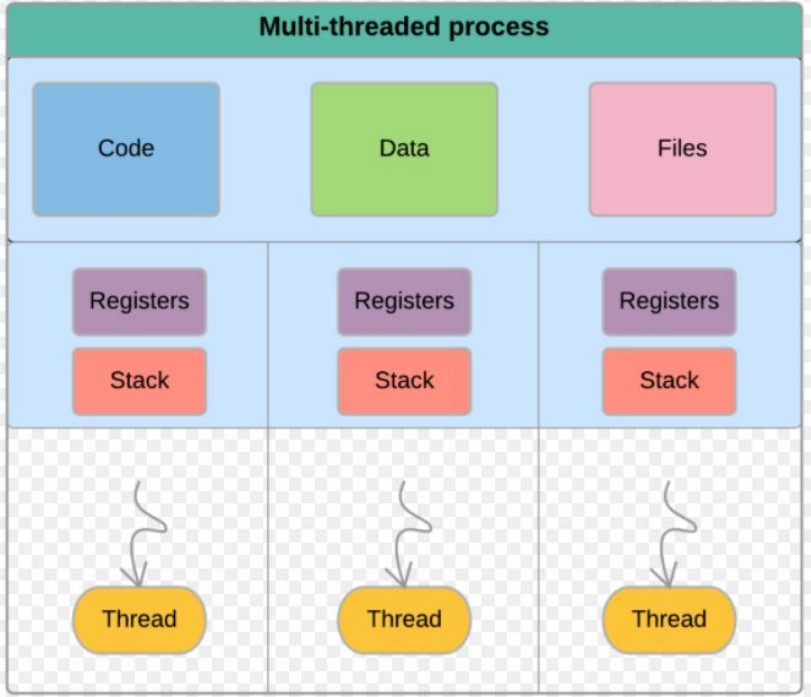
Threads - Concorrência

- Concorrência é necessário
- O computador é capaz de fazer mais de uma coisa ao mesmo tempo: ouvir música, navegar na internet, editar um documento
- Um programa é capaz de fazer mais de uma coisa ao mesmo tempo também: pense no Spotify. Você consegue pesquisar uma nova música enquanto ouve a uma outra
- Em um editor de texto, enquanto você digita ele valida a grafia da palavra
- No Whatsapp você consegue fazer o download de várias fotos ao mesmo tempo

Threads - Conceito

- Conjunto de instruções, de um programa ou processo, em execução
- Mais leve que processo
- Processo tem seu próprio espaço de memória
- Compartilha os dados do contexto
- Um processo, quando “ganha” a CPU, terá várias threads rodando
- Threads podem ser executadas em paralelo

Threads - Representação Gráfica



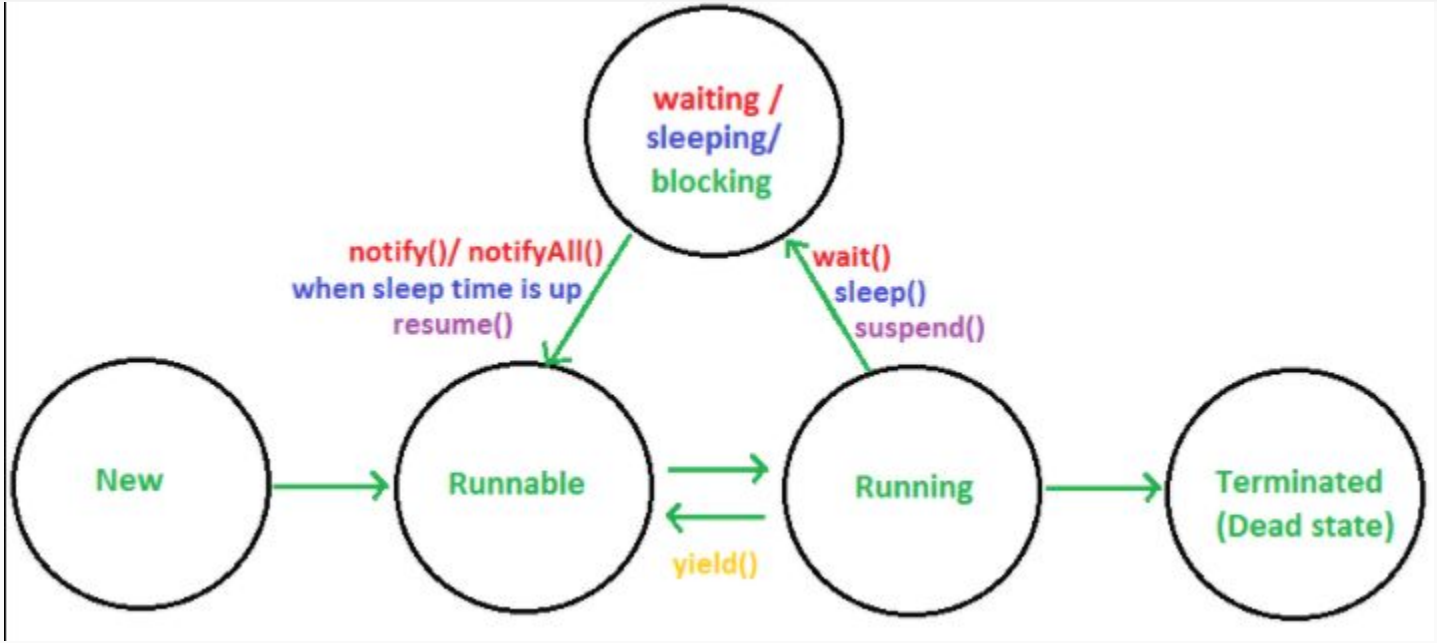
Threads - Java

- Thread é um objeto Java (equivale a uma thread no Sistema Operacional)
- Você define o código que uma thread vai executar duas maneiras:
  - **Implementar a interface Runnable (método run)**
  - Estender a classe Thread (sobrescrever método run)
- Para executar de fato uma thread deve-se fazer: Thread.**start**

Threads - Exemplo

- Exemplo: https://docs.oracle.com/javase/tutorial/essential/concurrency/examples/SleepMessages.java
- Você consegue pensar em exemplos de threads?
- Onde você usaria uma thread?
- As coisas acontecem em paralelo e você já sabe que a memória é algo compartilhado entre Threads. Isso é um problema?
- Uma thread esperar outra “eternamente” é um problema?

Threads - Ciclo de vida



Threads - Interferência

- Exemplo: https://docs.oracle.com/javase/tutorial/essential/concurrency/examples/Counter.java
- Sessão crítica/semáforo
- Uma thread tem que “ganhar” o lock (exclusividade no processamento)
- Após terminar o que tem que fazer tem que liberar o lock
- Counter “corrigido”: https://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html

Threads - Ciclo de Vida - “Starvation”

- Threads muito rápidas disputam recurso com recursos lentos. Exemplo: gargalo de rede de dados ou disco com leitura lenta
- Você cria um programa que escreve um texto muito grande em um arquivo. Cada linha do texto é escrita por uma thread, mas o arquivo está em um disco muito lento.
- O fluxo precisa fluir de fim-a-fim.
- O destino final não pode engargalar o processo

Threads - Ciclo de Vida - “Livelock”

- O estado não é blocked igual ao deadlock, mas o trabalho simplesmente não acaba
- Ocorre quando uma thread “A” espera o trabalho da “B” acabar, mas o trabalho não acaba
- Imagina duas pessoas passando num corredor apertado
- Imagine o deadlock, mas aqui a diferença está no tempo do processamento. Lá a coisa é hard e não escapatória

Threads - Baixo nível

- Threads são consideradas de baixo nível
- Existe uma grande de você cometer erros
- Código “difícil” de entender
- Se um erro acontecer, como pegar o bug?
- Erro muito relacionado a volume e ambiente, ou seja, difícil de reproduzir em ambiente de desenvolvimento. E agora?
- Existe algo pra facilitar a vida? Tem!

Threads - Alto nível

- Java oferece o pacote java.util.concurrent
- Permite fazer o mesmo que a classe Threads (e periféricos), mas de maneira mais amigável
- APIs de baixo nível são difíceis de serem empregadas em escala e processamentos complexos

Threads - Alto nível - Sessão crítica

- Java oferece o pacote java.util.concurrent.locks para isso
- Substitui a instrução synchronized
- Deixa o código mais elegante e claro em relação a sessão crítica/semáforo
- Mais suporte na comunidade
- Exemplo simples: https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html
- Exemplo...

Threads - Alto nível - Sessão crítica

```
3  public class SafelockDemo {
4
5      private SafelockDemo(){}
6
7      private static int count = 0;
8
9      public static int increment() {
10         return ++count;
11     }
12
13     public static int getCount() {
14         return count;
15     }
16
17 }
```

```
6  public class Safelock implements Runnable {
7
8      private final Lock lock = new ReentrantLock();
9
10     @Override
11     public void run() {
12         do {
13             lock.lock();
14             try {
15                 System.out.println(Thread.currentThread().getName() + " count: " + SafelockDemo.increment());
16             } finally {
17                 lock.unlock();
18             }
19         } while (SafelockDemo.getCount() < 11);
20     }
21
22     public static void main(String[] args) {
23         new Thread(new Safelock()).start();
24         new Thread(new Safelock()).start();
25     }
26
27 }
```

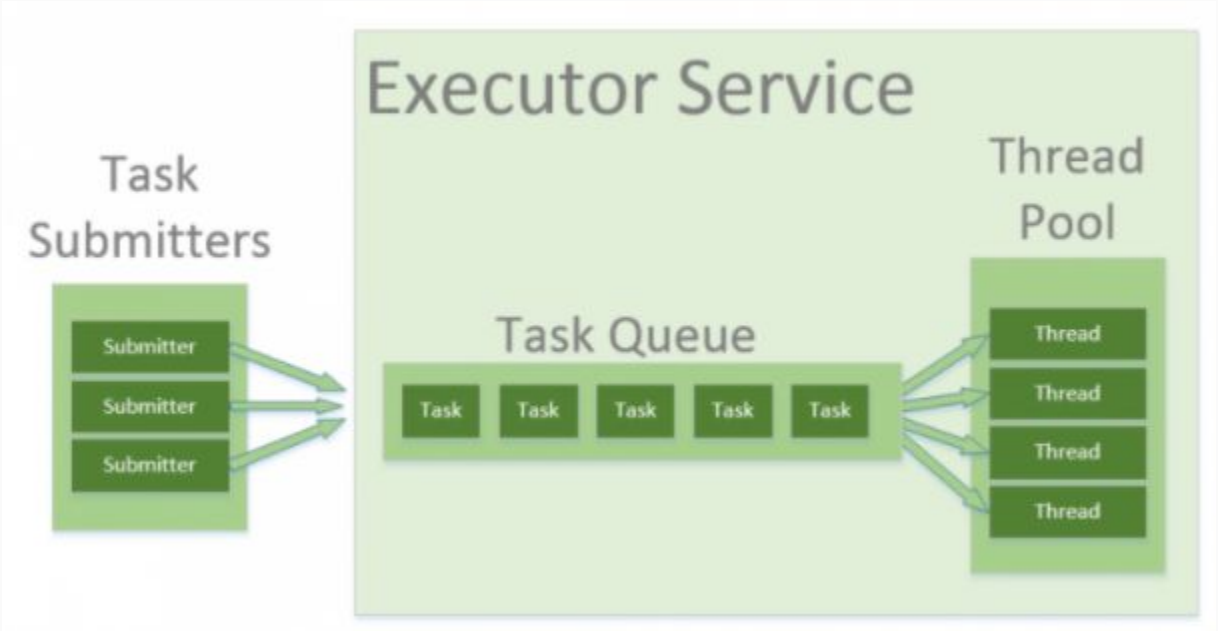
Threads - Alto nível - Sessão crítica

- Execute o código do slide anterior
- Que saída você viu no console?
- E se você adicionar mais uma thread para execução?

Threads - Alto nível - Executor

- É uma interface (de fato) simples para rodar uma thread (task)
- ExecutorService enriquece a interface Executor com boas práticas para trabalhar com threads (uma fila virtual e um pool de threads)
- ScheduledExecutorService enriquece a interface ExecutorService para executar tarefas de forma agendada

Threads - Alto nível - Executor



Threads - Alto nível - Thread Pool

- É uma “piscina” de threads
- Criar thread pode ter um overhead de processamento
- Essa técnica deixa threads prontas para uso de prontidão para serem utilizadas a qualquer tempo (*newFixedThreadPool*)
- Tamanho do pool precisa ser bem dimensionado
- Normal o tamanho do pool é igual ao número de processadores que a máquina tem. Isso é uma boa prática?
- Tamanho igual 1 (*newSingleThreadExecutor*)
- Processamento é assíncrono. Alguém sabe explicar? E pra que é útil?

Threads - Alto nível - Thread Pool

```
3  import java.util.concurrent.ExecutorService;
4  import java.util.concurrent.Executors;
5
6  public class Safelock implements Runnable {
7
8      @Override
9      public void run() {
0          do {
1              System.out.println(Thread.currentThread().getName() + " count: " + SafelockDemo.increment());
2          } while (SafelockDemo.getCount() < 11);
3      }
4
5      public static void main(String[] args) {
6          ExecutorService pool = Executors.newFixedThreadPool(2);
7          pool.execute(new Safelock());
8          pool.execute(new Safelock());
9      }
0
1  }
```

Revisão

- Threads são pedaços de código que podem ser executados em paralelo
- Thread é o objeto Java equivalente a uma thread do S.O. e é considerada uma abordagem de baixo nível
- O pacote java.util.concurrent oferece um framework completo para utilização de processamento em paralelo de forma mais alto nível
- Dimensionar o tamanho do pool corretamente e pensar nos casos de uso é muito importante