Programação_Concorrente-Studies/ Aula10: Locks , Pare de usar Synchronized nas threads e use a classe de locks

- Nessa aula iremos utilizar as classes ReentrantLock e a classe ReentrantReadWriteLock
- Mas para que serve o Lock?, o lock serve para quando você tem um recurso compartilhado qualquer que é alterado acessado ou qualquer outra coisa várias vezes e por multiplas threads, então usamos os locks para evitar as condições de corrida, logo os locks são bem proximos do exemplo do synchonized que vimos nas aulas anteriores.

- Primeiro vamos começar com o ReentrantLock, e para isso vamos utilizar um executor cached para entender melhor o funcionamento dessa classe:

```java
 * @author MarcusCSPereira
 */
public class ReentrantLock_1 {

  private static int i = -1;

  private static Lock lock = new ReentrantLock();

  Run | Debug
  public static void main(String[] args) {
    ExecutorService executor = Executors.
    newCachedThreadPool();

    Runnable r1 = () -> {
      lock.lock();
      String name = Thread.currentThread().getName
      ();
      i++;
      System.out.println(name + ": " + i);
      lock.unlock();
    };

    for (int i = 0; i < 6; i++) {
      executor.execute(r1);
    }

    executor.shutdown();
  }

}
```

- Assim criamos uma interface lock como um objeto ReentrantLock, e por meio dele travamos o momento em que as nossas Threads vão utilizar do recurso compartilhado, travamos utilizando

o método lock() do nosso lock, e após esse momento soltamos e deixamos livre os proximos blocos de código utilizando o método unlock() do nosso lock.

- O funcionamento do lock em si funciona quando uma thread obtém o lock e automaticamente as outras threads que chegarem no ponto de lock da runnable vão ficar paradas esperando o unlock do nosso lock , logo ele funciona como um semáforo de uma única vaga.
- Mas qual a diferença disso para o synchronized?, a vantagem do lock sobre o synchronized é por que o lock é muito mais flexivel, o lock permite o controle exato sobre qual bloco de código o recurso será bloqueado, ele permite esse controle de lock e unlock em linhas diferentes além de permitir que o lock seja feito em uma classe ou método e o unlock seja feito em outra classe ou outro método, já o synchronized não é assim, permitindo apenas a chamada em um bloco, que eu n posso dizer onde termina e onde começa se n for na mesma classe.
- Além disso o lock tem outros métodos, como o tryLock() que retorna um booleano permitindo um controle sobre se a Thread conseguiu o não a vaga naquele lock, temos também o tryLock(int i, time) que passamos por parâmetro um tempo para não voltar.
- * Obs: podemos delcarar diversos locks, contanto que tenhamos os seus devidos unlocks

- Agora vamos ver sobre a classe ReentrantReadWriteLock que é mais interessante:

```java
/**
 * JAVA MULTITHREAD - ReentrantReadWriteLock
 * @author MarcusCSPereira
 */
public class ReentrantReadWriteLock_1 {

private static int i = -1;

  private static ReadWriteLock lock = new
  ReentrantReadWriteLock();

  Run | Debug
  public static void main(String[] args) {
    ExecutorService executor = Executors.
    newCachedThreadPool();

    Runnable r1 = () -> {
      Lock writeLock = lock.writeLock();
      writeLock.lock();
      String name = Thread.currentThread().getName
      ();
      System.out.println(name + " - Escrevendo: "
      + i);
      i++;
      System.out.println(name + " - Escrito: " +
      i);
      writeLock.unlock();
    };
```

```
    Runnable r2 = () -> {
      Lock readLock = lock.readLock();
      readLock.lock();
      System.out.println("Lendo: " + i);
      System.out.println("Lido: " + i);
      readLock.unlock();
    };

    for (int i = 0; i < 6; i++) {
      executor.execute(r1);
      executor.execute(r2);
    }

    executor.shutdown();
  }

}
```

- Por meio do uso do ReentrantReadWriteLock(RRWL), podemos dividir os locks entre a thread que escreve uma informação , “escreve” um recurso e a classe que lê ou que trabalha com esse recurso.
- A primeira diferença do Lock para o RRWL é que primeiro devemos utilizar os métodos lock.writeLock() e lock.readLock() para recebermos a nossa variável de escrita e leitura.
- Essas variáveis de escrita e leitura tem os mesmos métodos lock e unlock.
- Nessa código oq buscamos entender é que entre uma chamada de escrevendo e escrito nunca terá outra chamada entre elas, afinal temos um lock específico para a escrita que é o writeLock que bloqueia a obtenção de outras Threads do writeLock e também do readLock().
- Porém o readLock funciona diferentemente do WriteLock, pois ele não bloqueia outro readLock, ele pode ter diversas Threads obtendo o readLock inclusive ao mesmo tempo, mas já para eu obter um lock de escrita, ele trava tanto outras threads de se obter o lock de escrita quanto o lock de leitura .

- Logo o writeLock: bloqueia tanto a obtenção da thread de um lock de escrita, quanto um lock de leitura.
- Já o readLock: não bloqueia a obtenção da Thread de nenhum dos locks de leitura , afinal na leitura queremos o maior número de Threads geralmente nesse processo, porém ela não permite que ninguem pegue o lock de escrita enquanto ela está lendo.

- Então o uso do Read Write lock permite você separar as tarefas que só estão querendo fazer a leitura do recurso compartilhado e quais são as que querem modificar ou escrever nesse mesmo recurso.

#Concorrent