Programação_Concorrente-Studies/ Aula11: SynchronousQueue e Exchanger , Trocando dados entre duas Threads de forma síncrona e fácil

- **Como vimos anteriormente temos diversas classes que servem para trocar informações entre Threads, porém essas classes dessa aula são especializadas em fazer a troca de informações entre duas Threads especificamente, elas até podem ser usadas para trocar dados e informações entre mais threads, mas não é o recomendado.**
- **OBS: o método put e take não serve apenas para essa tipo de fila em especifico, ele serve também para outros como LinkedBlockingDeque, e provavelmente serve para todos os tipos de Classes Atômicas, e mantendo o mesmo funcionamento de "compartilhamento" de recursos entre Threads, evitando a ocorrência de condições de corrida e de deadlock's.**

- **Vamos começar com a nossa SynchronousQueue, utlizando um executor cached:**

```java
public class SynchronousQueue_1 {

  private static final SynchronousQueue<String>
  FILA =
      new SynchronousQueue<>();

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

    Runnable r1 = () -> {
      put();
      System.out.println(x:"Escreveu na fila!");
    };
    Runnable r2 = () -> {
      String msg = take();
      System.out.println("Pegou da fila! " + msg);
    };

    executor.execute(r1);
    executor.execute(r2);

    executor.shutdown();
  }
```

```
  private static String take() {
    try {
      return FILA.take();
//      return FILA.poll(timeout, unit);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
      e.printStackTrace();
      return "Exceção!";
    }
  }

  private static void put() {
    try {
      FILA.put(e:"Cesinha0203");
//      FILA.offer(e, timeout, unit);
    } catch (InterruptedException e) {
      Thread.currentThread().interrupt();
      e.printStackTrace();
    }
  }

}
```

- Primeiramente chamamos o método put() que adiciona ou escreve algo na nossa fila, porém é de extrema importância entendermos que quando chamamos esse método a nossa Thread fica parada esperando, mas oq ela espera? , ela espera que uma outra Thread receba essa requisição de put() , afinal estamos trabalhando com uma SynchronousQueue, ou seja, uma fila sincrona, que necessita de uma chamada para solicitar por algo nela, quando uma chamada para receber algo e por nela “ao mesmo tempo”, e nesse caso é o método take(0 que chamamos na nossa Thread 2.

- Outra importante informação é a necessidade de usar no tratamento de exceções o método Thread.currentThread().interrupt(), pois é necessário para evitar o acontecimento de erros em que a execução e a espera para por algo na fila ou receber algo na fila, sejam infinitas.
- O nosso método put() passa como parâmetro algo, que nesse caso é oq desejamos colocar na fila, já nsso método take, retorna o que ele adicionou a fila.
- Além do take() temos o método poll(), que também pega algo da fila porém com um determinado tempo que ele espera por um put ou offer para tentar fazer tal processo.
- Além do put() temos o método offer(), que tambem coloca algo na fila porém com um determinado de tempo que espera receber por um take ou poll para receber tal informação

- Agora vamos analisar o uso da classe Exchanger:

```java
 * JAVA MULTITHREAD - Exchanger
 * @author MarcusCSPereira
 */
public class Exchanger_1 {

  private static final Exchanger<String>
  EXCHANGER = new Exchanger<>();

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

    Runnable r1 = () -> {
      String name = Thread.currentThread().getName
      ();
      System.out.println(name + " toma isso");
      String msg = "Toma isso!";
      String retorno = exchange(msg);
      System.out.println(name + " - " + retorno);
    };
    Runnable r2 = () -> {
      String name = Thread.currentThread().getName
      ();
      System.out.println(name + " obrigado");
      String msg = "Obrigado!";
      String retorno = exchange(msg);
      System.out.println(name + " - " + retorno);
    };
```

- Podemos notar que o exchanger é bem parecido com o nosso Synchronous queue porém ao invés de put e take, usamos apenas o método exchange.
- O método exchange além de enviar uma mensagem que passamos como parâmetro ele tabém recebe uma mensagem como retorno, então em ordem de execução seria como se a Thread 1 tentasse enviar uma mensagem para a thread 2, caso a Thread 2 receba essa mensagem ela envia algo para a thread 1 também, e então a Thread 1 tem seu retorno do método exchange sendo a mensagem enviada pela thread 2 pelo método exchange e a Thread 2 tem seu retorno do método exchange pelo método exchange enviado pela thread1, logo não tendo 1 thread que começe em si a execução do método de troca de informação que será realizado de forma sincrona enre as threads.
- Logo o Exchange também necessita do uso do método Thread.currentThread().interrupt(), pois oambas as Threads necessitam de esperar que uma envie e a outra receba e vice-versa.

- Logo a classe SynchronousQueue serve para uma Thread enviar algo para outra e a outra retornar oq foi enviada, e o Exchanger faz uma troca de dados, uma entrega algo para outra mas também uma recebe algo da outra, sendo assim uma permuta de informação.

#Concorrent