**Понимание phaser в java с примером**

Я пытаюсь понять Phaser в java. Я написал пример, который застрял в ожидании прибытия других сторон.

Насколько я понимаю, фазер используется как синхронизация многократного использования (в отличие от CountdownLatch, которая не является многоразовым) барьером с барьерным действием (в отличие от Cyclicbarrier, который используется для совместного использования состояния, Phaser не должен делить состояние в барьерном действии). Поправьте меня, если я ошибаюсь.

Итак, в моем примере, я пытаюсь выполнить некоторый случайный код сложения и вычитания в каждом потоке после того, как определенное количество сторон/потоков достигнет барьера. Что я делаю не так?

import static java.lang.String.\*;

import java.util.Random;

import java.util.concurrent.Callable;

import java.util.concurrent.CompletionService;

import java.util.concurrent.ExecutionException;

import java.util.concurrent.ExecutorCompletionService;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.Phaser;

import java.util.stream.IntStream;

public class PhaserUsage implements Callable<String> {

private static final int THREAD\_POOL\_SIZE = 10;

private final Phaser phaser;

private PhaserUsage(Phaser phaser) {

this.phaser = phaser;

}

public static void main(String a[]) {

ExecutorService execService = Executors.newFixedThreadPool(THREAD\_POOL\_SIZE);

CompletionService<String> completionService = new ExecutorCompletionService<>(execService);

Phaser phaser = new Phaser(1);

IntStream.range(0, THREAD\_POOL\_SIZE)

.forEach(nbr -> completionService.submit(new PhaserUsage(phaser)));

execService.shutdown();

try {

while (!execService.isTerminated()) {

String result = completionService.take().get();

System.out.println(format("Result is: %s", result));

}

} catch (ExecutionException | InterruptedException e) {

e.printStackTrace();

}

}

@Override

public String call() {

String threadName = Thread.currentThread().getName();

System.out.println(format("Registering...%s",threadName));

phaser.register();

System.out.println(format("Arrive and await advance...%s",threadName));

phaser.arriveAndAwaitAdvance(); // await all creation

int a = 0, b = 1;

Random random = new Random();

for (int i = 0; i < random.nextInt(10000000); i++) {

a = a + b;

b = a - b;

}

System.out.println(format("De-registering...%s",threadName));

phaser.arriveAndDeregister();

return format("Thread %s results: a = %s, b = %s", threadName, a, b);

}

}

I.

Вы инициализируете свой Phaser со значением 1:

Phaser phaser = new Phaser(1);

Это означает, что ваш основной поток является одним из тех потоков, которые вы ожидаете, но он никогда не вызывает функцию "прибыть"().

Поскольку число ваших потоков исправлено, вы должны инициализировать Phaser номером потока и удалить вызовы register().

II.

Проблема в том, что вы не **можете** вызвать phaser.register() из зарегистрированной задачи. При использовании фазеров всегда следуйте этим двум правилам:

1. **Только зарегистрированные задачи могут регистрировать другие задачи.** Это означает, что задача не может регистрироваться.
2. **Все зарегистрированные задачи должны отменяться до завершения.** Хорошей практикой является обернуть код, используя фазер вокруг блока finally который отменяет регистрацию в конце (см. Пример).

Вот ваша фиксированная программа (обратите внимание на строку, которая создает фазер):

import static java.lang.String.\*;

import java.util.Random;

import java.util.concurrent.\*;

import java.util.stream.IntStream;

public class PhaserUsage implements Callable<String> {

private static final int THREAD\_POOL\_SIZE = 10;

private final Phaser phaser;

private PhaserUsage(Phaser phaser) {

this.phaser = phaser;

}

public static void main(String a[]) {

ExecutorService execService = Executors.newFixedThreadPool(THREAD\_POOL\_SIZE);

CompletionService<String> completionService = new ExecutorCompletionService<>(execService);

// since we know beforehand how many tasks we have, initialize the

// number of participants in the constructor; other wise register

// \*before\* launching the task

Phaser phaser = new Phaser(THREAD\_POOL\_SIZE);

IntStream.range(0, THREAD\_POOL\_SIZE)

.forEach(nbr -> completionService.submit(new PhaserUsage(phaser)));

execService.shutdown();

try {

while (!execService.isTerminated()) {

String result = completionService.take().get();

System.out.println(format("Result is: %s", result));

}

} catch (ExecutionException | InterruptedException e) {

e.printStackTrace();

}

}

@Override

public String call() {

String threadName = Thread.currentThread().getName();

System.out.println(format("Arrive and await advance...%s",threadName));

phaser.arriveAndAwaitAdvance(); // await all creation

int a = 0, b = 1;

Random random = new Random();

for (int i = 0; i < random.nextInt(10000000); i++) {

a = a + b;

b = a - b;

}

System.out.println(format("De-registering...%s",threadName));

phaser.arriveAndDeregister();

return format("Thread %s results: a = %s, b = %s", threadName, a, b);

}

}