**Phaser Основной пример**

1. **Phaser через CompletionService**

import java.util.ArrayList;  
import java.util.concurrent.\*;  
  
import static java.lang.String.*format*;  
  
public class Main3 {  
  
 private static final int *THREAD\_POOL\_SIZE* = 3; //число должно быть не меньше чем максимальное количество потоков на любой из фаз  
  
 public static void main(String[] args) {  
 Phaser phaser = new Phaser(*THREAD\_POOL\_SIZE*);  
   
 ExecutorService execService = Executors.*newFixedThreadPool*(*THREAD\_POOL\_SIZE*);  
 CompletionService<String> completionService = new ExecutorCompletionService<>(execService);  
 ArrayList<Future<String>> futures = new ArrayList<>();  
  
 futures.add(completionService.submit(new Washer1(phaser)));  
 futures.add(completionService.submit(new Washer2(phaser)));  
 futures.add(completionService.submit(new Washer3(phaser)));  
  
 execService.shutdown();  
  
// try {  
// while (!execService.isTerminated()) { //пока выполняются поставленные в очередь задачи  
// String result = completionService.take().get();  
// System.out.println(format("Result is: %s", result));  
// }  
// } catch (ExecutionException | InterruptedException e) {  
// e.printStackTrace();  
// }  
 }  
  
}

import java.util.concurrent.Callable;  
import java.util.concurrent.Phaser;  
  
public class Washer1 implements Callable<String> {  
 Phaser phaser;  
  
 public Washer1(Phaser phaser) {  
 this.phaser = phaser;  
 }  
  
 @Override  
 public String call() {  
 System.*out*.println(Thread.*currentThread*().getName() + " washing the interior1");  
 phaser.arriveAndAwaitAdvance();  
 System.*out*.println(Thread.*currentThread*().getName() + " washing the interior2");  
 phaser.arriveAndAwaitAdvance();  
 System.*out*.println(Thread.*currentThread*().getName() + " washing the interior3");  
 phaser.arriveAndAwaitAdvance();  
 phaser.arriveAndDeregister();  
 return "Washer2 finished his work ";  
 }  
}

import java.util.concurrent.Callable;  
import java.util.concurrent.Phaser;  
  
public class Washer2 implements Callable<String> {  
 Phaser phaser;  
  
 public Washer2(Phaser phaser) {  
 this.phaser = phaser;  
 }  
  
 @Override  
 public String call() {  
 System.*out*.println(Thread.*currentThread*().getName() + " washing the exterior1");  
 phaser.arriveAndAwaitAdvance();  
 System.*out*.println(Thread.*currentThread*().getName() + " washing the exterior2");  
 phaser.arriveAndDeregister();  
 return "Washer2 finished his work ";  
 }  
}

import java.util.concurrent.Callable;  
import java.util.concurrent.Phaser;  
  
public class Washer3 implements Callable<String> {  
 Phaser phaser;  
  
 public Washer3(Phaser phaser) {  
 this.phaser = phaser;  
 }  
  
 @Override  
 public String call(){  
 for (int i = 0; i < 4; i++) {  
 System.*out*.println(Thread.*currentThread*().getName() + " washing the wills");  
 phaser.arriveAndAwaitAdvance();  
 }  
 phaser.arriveAndDeregister();  
 return "Washer3 finished his work ";  
 }  
}

1. **Phaser через автоматически раздаваемые потоки из ForkJoinPool**

import java.util.concurrent.Phaser;  
  
public class Main1 {  
  
 private static final int *THREAD\_POOL\_SIZE* = 3;  
  
 public static void main(String[] args) {  
 Phaser phaser = new Phaser(*THREAD\_POOL\_SIZE*);  
  
 new Washer1(phaser);  
 new Washer2(phaser);  
 new Washer3(phaser);  
 }  
}

import java.util.concurrent.Phaser;  
  
public class Washer1 extends Thread{  
 Phaser phaser;  
  
 public Washer1(Phaser phaser) {  
 this.phaser = phaser;  
 start();  
 }  
  
 @Override  
 public void run() {  
 System.*out*.println(*currentThread*().getName() + " washing the interior1");  
 phaser.arriveAndAwaitAdvance();  
 System.*out*.println(*currentThread*().getName() + " washing the interior2");  
 phaser.arriveAndAwaitAdvance();  
 System.*out*.println(*currentThread*().getName() + " washing the interior3");  
 phaser.arriveAndDeregister();  
 }  
}

import java.util.concurrent.Phaser;  
  
public class Washer2 extends Thread{  
 Phaser phaser;  
  
 public Washer2(Phaser phaser) {  
 this.phaser = phaser;  
 start();  
 }  
  
 @Override  
 public void run() {  
 System.*out*.println(*currentThread*().getName() + " washing the exterior1");  
 phaser.arriveAndAwaitAdvance();  
 System.*out*.println(*currentThread*().getName() + " washing the exterior2");  
 phaser.arriveAndDeregister();  
 }  
}

import java.util.concurrent.Phaser;  
  
public class Washer3 extends Thread{  
 Phaser phaser;  
  
 public Washer3(Phaser phaser) {  
 this.phaser = phaser;  
 start();  
 }  
  
 @Override  
 public void run() {  
 for (int i = 0; i < 4; i++) {  
 System.*out*.println(*currentThread*().getName() + " washing the wills");  
 phaser.arriveAndAwaitAdvance();  
 }  
 phaser.arriveAndDeregister();  
 }  
}

1. **Работа с файлами папки (копирование файлов с расширением “.java” из одной папки в другую) в одном потоке на первом шаге Phaser-а.**

**package** com.company.main3;  
  
**import** java.io.IOException;  
**import** java.nio.file.\*;  
**import** java.nio.file.attribute.\*;  
  
**class MyFileFindVisitor** **extends** SimpleFileVisitor<Path> {  
 **private** PathMatcher **matcher**;  
  
 **public** MyFileFindVisitor(String pattern) {  
 **try** {  
 **matcher** = FileSystems.*getDefault*().getPathMatcher(pattern);  
 } **catch** (IllegalArgumentException iae) {  
 System.***err***.println(**"Invalid pattern; did you forget to prefix \"glob:\" or \"regex:\"?"**);  
 System.*exit*(1);  
 }  
 }  
  
 **private void** find(Path path) **throws** IOException {  
 Path finishPath = Paths.*get*(**"D:\\Java\\example\\src\\classes\\comparator33"**); *//путь куда копировать пишу тут т.к. не знаю как передать его в "вызывающем" классе* Path name = path.getFileName();  
 **if** (**matcher**.matches(name)) {  
 Path finishPathAll = finishPath.resolve(path.getFileName());  
 **if** (!Files.*exists*(finishPath)) { *//создаю папку куда копировать если такой нет*  
 **try** {  
 Files.*createDirectory*(finishPath);  
 } **catch** (IOException e) {  
 System.***err***.println(e);  
 }  
 }  
 **try** { *//копирую каждый найденный файл в другую папку*  
 Files.*copy*(path, finishPathAll, StandardCopyOption.***REPLACE\_EXISTING***);  
 System.***out***.println(Thread.*currentThread*().getName() + **" Source file "** + path + **" To: "** + finishPath + **" copied successfully"**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }

//Дальше идут два метода выполняющие одно и тоже. Почему не хватает реализации только первого из методов, выполняющегося при доступе к найденому файлу, я не знаю!!!  
 **public** FileVisitResult visitFile(Path path, BasicFileAttributes fileAttributes) **throws** IOException {  
 find(path);  
 **return** FileVisitResult.***CONTINUE***;  
 }  
 **public** FileVisitResult preVisitDirectory(Path path, BasicFileAttributes fileAttributes) **throws** IOException {  
 find(path);  
 **return** FileVisitResult.***CONTINUE***;  
 }  
}

**package** com.company.main3;  
  
**import** java.io.IOException;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.util.concurrent.Callable;  
**import** java.util.concurrent.Phaser;  
  
**public class Washer2** **implements** Callable<String> {  
 Phaser **phaser**;  
  
 **public** Washer2(Phaser phaser) {  
 **this**.**phaser** = phaser;  
 }  
  
 @Override  
 **public** String call() {  
  
 Path startPath = Paths.*get*(**"D:\\Java\\example\\src\\classes\\comparator"**); *//Введите сюда путь к каталогу для поиска* String pattern = **"glob:\*.java"**;*//Строка с glob-шаблоном  
 //String pattern = "regex:\\S+\\.java"; //Строка с regex-шаблоном* **try** {  
 Files.*walkFileTree*(startPath, **new** MyFileFindVisitor(pattern));  
 System.***out***.println(Thread.*currentThread*().getName() + **"File search completed!"**);  
 } **catch** (IOException e) { e.printStackTrace(); }  
 **phaser**.arriveAndAwaitAdvance();  
  
 System.***out***.println(Thread.*currentThread*().getName() + **" washing the exterior2"**);  
 **phaser**.arriveAndDeregister();  
 **return "Washer2 finished his work "**;  
 }  
}