Міністерство освіти і науки України  
Івано-Франківський національний технічний університет нафти і газу

Кафедра програмного   
забезпечення автоматизованих систем

Лабораторна робота №5

Додаткові способи синхронізації та створення

пулу потоків у мові Java

Виконав: ст. гр. ПІ-13-1

Колесник А.

Перевірив:   
Піх В.Я.

м. Івано-Франківськ

2016

**Мета:** Ознайомитися із додатковими методами не блокуючого режиму синхронізованого доступу до спільних даних та навчитися створювати пул потоків з метою ефективного керування множиною потоків у мові Java.

**Порядок виконання роботи:**

1. Ознайомитися із теоретичними відомостями до лабораторної роботи.

2. Написати код який демонструє переваги використання ключових слів

volatile та final у порівнянні із застосуванням блокуючого режиму синхронізованого доступу.

3. Створити два пули потоків: перший із фіксованою кількістю потоків, а другий – на основі кешу потоків.

4. Створити невеликий масив цілих чисел (до 10-ти чисел) та написати код який буде робити певну операцію над кожним елементом масиву (наприклад, множити його на деяке число). Для того, щоб імітувати складність операції над кожним елементом масиву, перед її виконанням викликати метод Thread.sleep() із довільним значенням.

5. Виконати написаний раніше код спочатку в одному потоці, а потім асинхронно, використовуючи фіксований пул потоків та пул на основі кешу потоків. Порівняти час необхідний на виконання одного і того ж коду з використанням різних підходів.

6. Створити список задач у яких передбачається знаходження мінімального та максимального елемента масиву, а також суми та добутку усіх елементів масиву і відправити їх на виконання пулу потоків. Повернути результат виконання кожної із задач та вивести усі результати на екран.

7. Модифікувати попередній код таким чином, щоб тепер кожна задача виконувалася із затримкою у п’ять секунд після виконання попередньої задачі.

8. У звіті до лабораторної роботи представити знімки робочої програми.

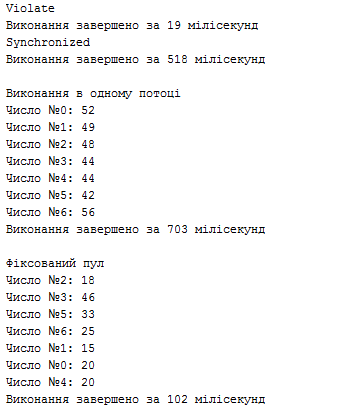
9. Оформити звіт та зробити висновки по роботі.

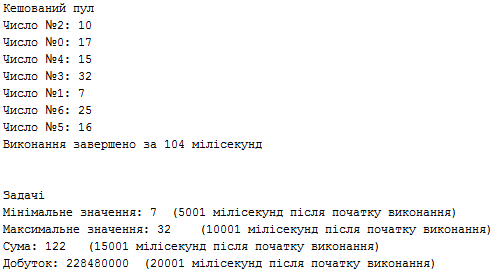
**Виконання Роботи:**

**package** com.company;  
  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.List;  
**import** java.util.concurrent.\*;  
  
**public class** Main {  
  
 **static volatile int**[] *numsVolatile*;  
 **static int**[] *numsSync*;  
  
 **public static int** multiply(**int** value) {  
 **try** {  
 Thread.*sleep*(10);  
 value \*= 8;  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 **return** value;  
 }  
  
 **public static synchronized int** multiplySync (**int** value) {  
 **try** {  
 Thread.*sleep*(10);  
 value \*= 8;  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 **return** value;  
 }  
  
 **public static void** main(String[] args) {  
 ExecutorService fixedExecutor = Executors.*newFixedThreadPool*(7);  
 ExecutorService cashedExecutor = Executors.*newCachedThreadPool*();  
  
 *numsVolatile* = **new int**[50];  
 *numsSync* = **new int**[50];  
 Arrays.*fill*(*numsSync*, 100);  
 Arrays.*fill*(*numsVolatile*, 100);  
  
 ExecutorService violateExecutor = Executors.*newCachedThreadPool*();  
 System.***out***.println(**"Volatile"**);  
 **long** start = System.*currentTimeMillis*();  
 **try** {  
 **for** (**int** j = 0; j < *numsVolatile*.**length**; j++) {  
 **final int** $i = j;  
 violateExecutor.submit(**new** Runnable() {  
 @Override  
 **public void** run() {  
 *multiply*(*numsVolatile*[$i]);  
 }  
 });  
 }  
 violateExecutor.shutdown();  
 violateExecutor.awaitTermination(Long.***MAX\_VALUE***, TimeUnit.***NANOSECONDS***);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 } **finally** {  
 violateExecutor.shutdownNow();  
 }  
 System.***out***.println(**"Виконання завершено за "** + (System.*currentTimeMillis*() - start) + **" мілісекунд"**);  
  
 ExecutorService syncExecutor = Executors.*newCachedThreadPool*();  
 System.***out***.println(**"Synchronized"**);  
 start = System.*currentTimeMillis*();  
 **try** {  
 **for** (**int** j = 0; j < *numsSync*.**length**; j++) {  
 **final int** $i = j;  
 syncExecutor.submit(**new** Runnable() {  
 @Override  
 **public void** run() {  
 *multiplySync*(*numsSync*[$i]);  
 }  
 });  
 }  
 syncExecutor.shutdown();  
 syncExecutor.awaitTermination(Long.***MAX\_VALUE***, TimeUnit.***NANOSECONDS***);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 } **finally** {  
 syncExecutor.shutdownNow();  
 }  
 System.***out***.println(**"Виконання завершено за "** + (System.*currentTimeMillis*() - start) + **" мілісекунд\n"**);  
  
 *//////* **int**[] nums = **new int**[7];  
 Arrays.*fill*(nums, 30);  
 start = System.*currentTimeMillis*();  
 System.***out***.println(**"Виконання в одному потоці"**);  
 **try** {  
 **for** (**int** i = 0; i < nums.**length**; i++) {  
 Thread.*sleep*(100);  
 nums[i] \*= Math.*random*() + 1;  
 System.***out***.println(**"Число №"** + i + **": "** + nums[i]);  
 }  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.***out***.println(**"Виконання завершено за "** + (System.*currentTimeMillis*() - start) + **" мілісекунд\n"**);  
  
 start = System.*currentTimeMillis*();  
 System.***out***.println(**"Фіксований пул"**);  
 **try** {  
 **for** (**int** i = 0; i < nums.**length**; i++) {  
 **final int** $i = i;  
 fixedExecutor.submit(**new** Runnable() {  
 **public void** run() {  
 **try** {  
 Thread.*sleep*(100);  
 nums[$i] \*= Math.*random*() + .1;  
 System.***out***.println(**"Число №"** + $i + **": "** + nums[$i]);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 });  
 }  
 fixedExecutor.shutdown();  
 fixedExecutor.awaitTermination(Long.***MAX\_VALUE***, TimeUnit.***NANOSECONDS***);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 fixedExecutor.shutdownNow();  
 }  
 System.***out***.println(**"Виконання завершено за "** + (System.*currentTimeMillis*() - start) + **" мілісекунд\n"**);  
 start = System.*currentTimeMillis*();  
 System.***out***.println(**"\nКешований пул"**);  
 **try** {  
 **for** (**int** i = 0; i < nums.**length**; i++) {  
 **final int** $i = i;  
 cashedExecutor.submit(**new** Runnable() {  
 **public void** run() {  
 **try** {  
 Thread.*sleep*(100);  
 nums[$i] \*= Math.*random*() + .1;  
 System.***out***.println(**"Число №"** + $i + **": "** + nums[$i]);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 });  
 }  
 cashedExecutor.shutdown();  
 cashedExecutor.awaitTermination(Long.***MAX\_VALUE***, TimeUnit.***NANOSECONDS***);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 } **finally** {  
 cashedExecutor.shutdownNow();  
 }  
 System.***out***.println(**"Виконання завершено за "** + (System.*currentTimeMillis*() - start) + **" мілісекунд\n"**);  
  
 System.***out***.println(**"\nЗадачі"**);  
 List<Callable<String>> tasks = **new** ArrayList<Callable<String>>(4);  
 tasks.add(() -> {  
 **int** minIndex = 0;  
 **for** (**int** i = 0; i < nums.**length**; i++) {  
 **if** (nums[minIndex] > nums[i])  
 minIndex = i;  
 }  
 **return "Мінімальне значення: "** + nums[minIndex];  
 });  
 tasks.add(() -> {  
 **int** maxIndex = 0;  
 **for** (**int** i = 0; i < nums.**length**; i++) {  
 **if** (nums[maxIndex] < nums[i])  
 maxIndex = i;  
 }  
 **return "Максимальне значення: "** + nums[maxIndex];  
 });  
 tasks.add(() -> {  
 **int** sum = 0;  
 **for** (**int** i = 0; i < nums.**length**; i++) {  
 sum += nums[i];  
 }  
 **return "Сума: "** + sum;  
 });  
 tasks.add(() -> {  
 **int** prod = 1;  
 **for** (**int** i = 0; i < nums.**length**; i++) {  
 prod \*= nums[i];  
 }  
 **return "Добуток: "** + prod;  
 });  
  
 ScheduledExecutorService scheduledExecutor = Executors.*newScheduledThreadPool*(1);  
 **try** {  
 List<Future<String>> futures = **new** ArrayList<>();  
 start = System.*currentTimeMillis*();  
 **int** delay = 0;  
 **for** (Callable task : tasks) {  
 futures.add((Future<String>) scheduledExecutor.schedule(task, delay += 5, TimeUnit.***SECONDS***));  
 }  
 **for** (Future<String> future : futures) {  
 System.***out***.println(future.get() + **"\t("** + (System.*currentTimeMillis*() - start) + **" мілісекунд після початку виконання)"**);  
 }  
 scheduledExecutor.shutdown();  
 scheduledExecutor.awaitTermination(Long.***MAX\_VALUE***, TimeUnit.***NANOSECONDS***);  
 } **catch** (InterruptedException|ExecutionException e) {  
 e.printStackTrace();  
 } **finally** {  
 cashedExecutor.shutdownNow();  
 }  
 }  
}

Результат виконання програми:

volatile





**Висновок:** На цій лабораторній роботі було розглянуто способи синхронізованого доступу до даних без блокування та створення пулів потоків.