Звіт

Автор: Лунгол О., КІТ101.8б

ЛАБОРАТОРНА РОБОТА №13

# [Паралельне виконання. Багатопоточність](https://oop-khpi.gitlab.io/#tasks)

МЕТА:

* Ознайомлення з моделлю потоків *Java*.
* Організація паралельного виконання декількох частин програми.

ВИМОГИ

1. Використовуючи програми рішень попередніх задач, продемонструвати можливість паралельної обробки елементів контейнера: створити не менше трьох додаткових потоків, на яких викликати відповідні методи обробки контейнера.
2. Забезпечити можливість встановлення користувачем максимального часу виконання (таймаута) при закінченні якого обробка повинна припинятися незалежно від того знайдений кінцевий результат чи ні.
3. Для паралельної обробки використовувати алгоритми, що не змінюють початкову колекцію.
4. Кількість елементів контейнера повинна бути досить велика, складність алгоритмів обробки колекції повинна бути зіставна, а час виконання приблизно однаковий, наприклад:
   * пошук мінімуму або максимуму;
   * обчислення середнього значення або суми;
   * підрахунок елементів, що задовольняють деякій умові;
   * відбір за заданим критерієм;
   * власний варіант, що відповідає обраній прикладної області.

ОПИС ПРОГРАМИ

**Опис змінних**

HDR1 rec1 = *new* HDR1();  
HDR1 rec2 = *new* HDR1();  
HDR1 rec3 = *new* HDR1();

*package* ua.khpi.oop.Lunhol13.Container;  
  
*import* ua.khpi.oop.Lunhol13.Head.HDR1;  
  
*import* java.util.concurrent.*Callable*;  
*import* java.util.concurrent.TimeUnit;  
  
*public class* Threads {  
  
 *public static class* MyTread1 *implements Callable*<Boolean> {  
 Container<HDR1> container;  
  
 @Override  
 *public* Boolean call() *throws* InterruptedException {  
 maxMinPayment();  
 *return true*;  
 }  
  
 *private void* maxMinPayment() *throws* InterruptedException {  
 TimeUnit.*SECONDS*.sleep(3);  
 *int* max = container.getElementByIndex(0).getPayment();  
 *int* min = container.getElementByIndex(0).getPayment();  
 *for* (*int* i = 0; i < container.size(); i++) {  
 *if* (container.getElementByIndex(i).getPayment() < min) {  
 min = container.getElementByIndex(i).getPayment();  
 }  
 *if* (container.getElementByIndex(i).getPayment() > max) {  
 max = container.getElementByIndex(i).getPayment();  
 }  
 }  
 System.*out*.println("Max payment = " + max);  
 System.*out*.println("Min payment = " + min);  
 }  
  
 *public void* set(Container<HDR1> container) {  
 *this*.container = container;  
 }  
  
  
 }  
  
 *public static class* MyTread2 *implements Callable*<Boolean> {  
 Container<HDR1> container;  
  
  
 *private void* sumAvgPayment() {  
 *int* sum = 0;  
 *int* avg = 0;  
 *for* (*int* i = 0; i < container.size(); i++) {  
 sum += container.getElementByIndex(i).getPayment();  
 }  
 avg = sum / container.size();  
 System.*out*.println("Sum payment = " + sum);  
 System.*out*.println("Avg payment = " + avg);  
 }  
  
 *public void* set(Container<HDR1> container) {  
 *this*.container = container;  
 }  
  
 @Override  
 *public* Boolean call() *throws* Exception {  
 sumAvgPayment();  
 *return true*;  
 }  
 }  
  
 *public static class* MyTread3 *implements Callable*<Boolean> {  
 *public static final int HIGHER\_PAYMENT* = 1000;  
 Container<HDR1> container;  
  
 @Override  
 *public* Boolean call() *throws* Exception {  
 countHigherPayment();  
 *return true*;  
 }  
  
 *private void* countHigherPayment() {  
 *int* count = 0;  
 *for* (*int* i = 0; i < container.size(); i++) {  
 *if* (container.getElementByIndex(i).getPayment() >= *HIGHER\_PAYMENT*) {  
 count++;  
 }  
 }  
 System.*out*.println("Number of vacancies with higher payment: " + count);  
 }  
  
 *public void* set(Container<HDR1> container) {  
 *this*.container = container;  
 }  
 }  
}

*package* ua.khpi.oop.Lunhol13.Tests;  
  
*import* ua.khpi.oop.Lunhol13.Container.Container;  
*import* ua.khpi.oop.Lunhol13.Container.Threads;  
*import* ua.khpi.oop.Lunhol13.Head.HDR1;  
  
*import* java.io.IOException;  
*import* java.util.concurrent.\*;  
  
*public class* Demo {  
  
 *public static void* main(String[] args) *throws* IOException {  
 *//Lab13.main(new String[]{"-auto"});  
 // Lab13.main(args);* Container<HDR1> container = *new* Container<>();  
 HDR1 rec1 = *new* HDR1();  
 HDR1 rec2 = *new* HDR1();  
 HDR1 rec3 = *new* HDR1();  
  
 rec1.setPayment(2600);  
 rec2.setPayment(3700);  
 *for* (*int* i = 0; i < 10000; i++) {  
 container.add(rec1);  
 container.add(rec2);  
 }  
 rec3.setPayment(600);  
 *for* (*int* i = 0; i < 1200; i++) {  
 container.add(rec3);  
 }  
 rec2.setPayment(2500);  
 container.add(rec2);  
 rec2.setPayment(500);  
 container.add(rec2);  
 Threads.MyTread1 myTread1 = *new* Threads.MyTread1();  
 myTread1.set(container);  
 Threads.MyTread2 myTread2 = *new* Threads.MyTread2();  
 myTread2.set(container);  
 Threads.MyTread3 myTread3 = *new* Threads.MyTread3();  
 myTread3.set(container);  
 *ExecutorService* executorService = Executors.*newFixedThreadPool*(3);  
 *Future*<Boolean> future = executorService.submit(myTread1);  
 *Future*<Boolean> future1 = executorService.submit(myTread2);  
 *Future*<Boolean> future2 = executorService.submit(myTread3);  
 *try* {  
 future.get(2, TimeUnit.*SECONDS*);  
 future1.get(2, TimeUnit.*SECONDS*);  
 future2.get(2, TimeUnit.*SECONDS*);  
 } *catch* (TimeoutException | InterruptedException | ExecutionException ex) {  
 future.cancel(*true*);  
 future1.cancel(*true*);  
 future2.cancel(*true*);  
 }  
 executorService.shutdown();  
  
 }  
}

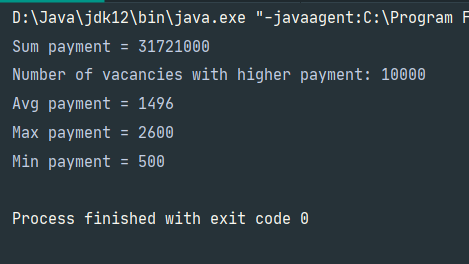


Рисунок 1 – Робота програми

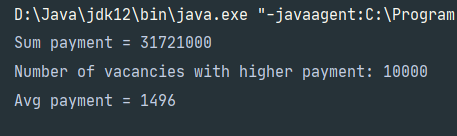


Рисунок 2 – Режим сну 3с , тайм аут чекає 2 с , якщо потік не встигає програма завершується незалежно від того знайдений кінцевий результат чи ні.