Міністерство освіти і науки України
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до виконання індивідуальних домашніх завдань
за темою «Багатопоточне програмування в Java
за допомогою CONCURRENCY API»
з дисципліни

«Паралельні та розподілені обчислення»

для студентів спеціальності
123 "Комп'ютерна інженерія"
всіх форм навчання

2021

Методичні вказівки до виконання індивідуальних домашніх завдань за темою "Багатопоточне програмування в Java за допомогою CONCURRENCY API" з дисципліни "Паралельні та розподілені обчислення" для студентів спеціальності 123 "Комп'ютерна інженерія" всіх форм навчання / Укл.: Р.К. Кудерметов, Т.С. Дьячук, С. Ю. Скрупський – Запоріжжя: НУЗП, 2021. – 19 с.

Автори: Р.К. Кудерметов, к.т.н., доцент
Т.С. Дьячук, асистент
С. Ю. Скрупський, к.т.н., доцент

Рецензент: Киричек Г.Г., к.т.н, доцент

Відповідальний
за випуск: Т.С. Дьячук, асистент

Затверджено
на засіданні кафедри КСМ
Протокол № 6 від 05.02.2021

Рекомендовано до видання
на засіданні НМК факультету КНТ
Протокол № 7 від 26.02.2021

ЗМІСТ

ВСТУП

Метою даних методичних вказівок є поглиблення знань з практичного курсу паралельного програмування для комп'ютерних систем зі спільною пам'яттю.

Засвоєння матеріалів даних методичних вказівок потребує від студентів значного обсягу самостійної роботи, оскільки лише практичне програмування, вирішення конкретних проблем у кожній програмі може закласти міцний фундамент знань та навичок у паралельному програмуванні.

Перед виконанням індивідуального домашнього завдання (ІДЗ) студенти повинні ознайомитися з теоретичними відомостями за темою та рекомендованою літературою, виконати лабораторну роботу "Багатопоточне програмування у Java" з курсу "Паралельні та розподілені обчислення".

Певні деталі опису технології CONCURRENCY API Java не наведено з метою спрощення викладення та сприйняття матеріалу. Звичайно, в дані методичні вказівки неможливо було внести весь матеріал за даною темою. Тому тут містяться основні, базові теоретичні відомості, необхідні для виконання ІДЗ.

Автори вважають, що матеріали, практичні приклади і завдання, викладені у даних методичних вказівках, під силу студентам, які бажають пов'язати свою діяльність із комп'ютерною інженерією та комп'ютерними науками, і будуть сприяти набуттю професійних знань і навичок у паралельному програмуванні.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Мова Java завжди пропонувала широкі можливості для багатопотокового програмування, потоки - це основа мови. Однак дуже часто використання цих можливостей викликало труднощі: створити і налагодити для коректного функціонування багатопоточну програму досить складно. У версії Java SE 5 був доданий пакет java.util.concurrent, можливості класів якого забезпечують більш високу продуктивність, масштабованість, побудову потокобезпечних блоків concurrent класів. Крім цього був вдосконалений виклик утиліт синхронізації, додані класи семафорів та блокувань. Схематично на рисунку зображено структуру пакету.

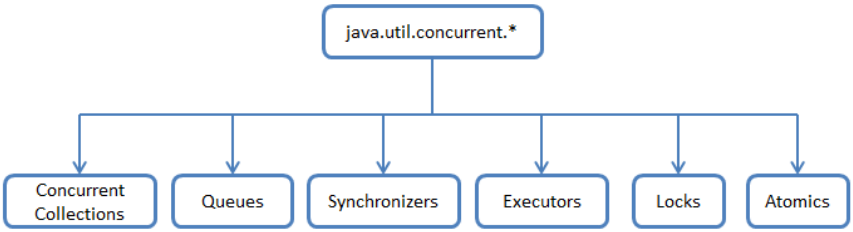

Рисунок – Структура пакету java.util.concurent.*

Concurrency API вводить поняття сервісу-виконавця (ExecutorService) – високорівневу заміну роботи з тредами безпосередньо. Виконавці виконують завдання *асинхронно* і зазвичай використовують пул потоків, які не треба створювати вручну.

Всі потоки з пулу будуть використані повторно після виконання завдання, а значить, ми можемо створити в додатку стільки завдань, скільки потрібно, використовуючи одного виконавця.

Виклик виконавця на один тред виглядає наступним чином:

```
ExecutorService executor = Executors.newSingleThreadExecutor();
```

На N потоків:

```
ExecutorService executor = Executors.newFixedThreadPool(N);
```

Постановка завдання в чергу пулу тредів на асинхронне виконання:

```
executor.submit(new Runnable()
        {
              @Override
              public void run()
              {
                    String threadName = Thread.currentThread().getName();
                   System.out.println("Hello " + threadName);
              }
        });
```

Крім Runnable, виконавці можуть приймати інший вид завдань, який називається Callable. Callable - це також інтерфейс, але, на відміну від Runnable, він може повертати значення.

```
Callable<Integer> task = new Callable<Integer>()
      {
              @Override
              public Integer call() throws Exception
              {
                    TimeUnit.SECONDS.sleep(1);
                return 123;
              }
        };
```

Callable - завдання також можуть бути передані виконавцям. Оскільки метод submit() виконавця не чекає завершення завдання, виконавець не може повернути результат завдання безпосередньо. Замість цього виконавець повертає спеціальний об'єкт Future, у якого ми зможемо запросити результат завдання.

```
     Future<Integer> future = executor.submit(task);
            System.out.println("future    done?    "    +
future.isDone());
            Integer result=0;
            try
            {
//Blocks    if    necessary    for    the    computation
//to complete, and then retrieves its result.
                  result = future.get();
            } catch (InterruptedException e1)
            {
                  e1.printStackTrace();
            }
            catch (ExecutionException e2)
            {
                  e2.printStackTrace();}
            System.out.println("future    done?    "    +
future.isDone());
            System.out.println("result: " + result);
```

Роботу виконавців треба завершувати явно. Для цього в інтерфейсі ExecutorService є два методи: shutdown(), який чекає завершення запущених завдань, та shutdownNow(), який зупиняє виконавця негайно.

```
     try {
                  System.out.println("attempt to shutdown
executor");
                executor.shutdown();

   executor.awaitTermination(5, TimeUnit.SECONDS);
            }
            catch (InterruptedException e) {
                System.err.println("tasks     interrup"-
ed");
            }
            finally
            {
                if (!executor.isTerminated())
                {
```

```
                    System.err.println("cancel      non-
finished tasks");
                }

                executor.shutdownNow();
                System.out.println("shutdown
finished");
        }
```

В Java 8: з'явилася можливість замість використання фіксованої кількості потоків створити такий пул, який за замовчуванням дорівнює кількості ядер машини.

```
    ExecutorService              service              =
Executors.newWorkStealingPool();
            Calc c = new Calc(5, 12);
            Future<Integer> res = service.submit(c);
            while(!res.isDone())
            {
                  System.out.println("waiting...");
                  Thread.sleep(1000);
            }
            System.out.println(res.get());
            service.shutdown();
            service.awaitTermination(5,
TimeUnit.SECONDS);

    public class Calc implements Callable<Integer>
    {
            int a;
            int b;
            public Calc(int a, int b)
            {
                  this.a = a;
                  this.b = b;
            }
            @Override
            public Integer call()
            {
                  try
                  {
                        Thread.sleep(2000);
                  } catch (InterruptedException e)
```

```
                {
                        e.printStackTrace();
                }
                return a+b;
        }
}
```

Виконавці можуть приймати список завдань на виконання за допомогою методу invokeAll(), який приймає колекцію callable-завдання та повертає список Future.

```
    ExecutorService executor = Executors.newWorkStealingPool();
     ArrayList<Callable<Integer>> tasks = new ArrayList<>();
     tasks.add(new Calc(1,2));
    tasks.add(new Calc(2,7)); tasks.add(new Calc(-1,8));
     List<Future<Integer>> results = executor.invokeAll(tasks);
           for(Future<Integer> res : results)
           {
                /* Блокує поки завдання не буде виконано. Недолік у тому, що ми опитуємо завдання по одному, а на виконані вони можуть бути в іншому порядку */
                System.out.println(res.get());
           }

           /* Інший спосіб - віддати на виконання кілька завдань - методом invokeAny(). В місці повернення Future він блокує потік до того, як завершиться хоч одна задача і повертає її результат.*/

           int res = executor.invokeAny(tasks);
           System.out.println(res);
           executor.shutdown();
```

Раціональний спосіб - отримувати результати за допомогою спеціального сервісу CompletionService. Він допоможе діставати за-

вдання в порядку завершення їх виконання (метод take), а не в порядку запуску.

```
    ExecutorService              executor              =
Executors.newWorkStealingPool();
    CompletionService<Integer> taskCompletionService =
new ExecutorCompletionService<>(executor);
     ArrayList<Callable<Integer>>    tasks    =    new
ArrayList<>();
     tasks.add(new Calc(1,2));
    tasks.add(new Calc(2,7));
    tasks.add(new Calc(-1,8));
           for(Callable<Integer> task : tasks)
           {
                 taskCompletionService.submit(task);
           }

           for(int i=0;i<tasks.size();i++)
           {
                 Future<Integer>   result   =   taskC-
?mpletionService.take();
                 System.out.println(result.get());
           }
           executor.shutdown();
```

Для того щоб періодично запускати завдання, ми можемо використовувати пул потоків з планувальником ScheduledExecutorService, який здатний запускати завдання один або кілька разів через встановлені проміжки часу.

```
    ScheduledExecutorService         executor         =
Executors.newScheduledThreadPool(1);
           Calc task = new Calc(1,2);
//запустити задачу через 3 сек від поточного часу
     ScheduledFuture<Integer>          res          =
executor.schedule(task, 3, TimeUnit.SECONDS);
           while(!res.isDone())
           {
System.out.println(res.getDelay(TimeUnit.SECONDS))
                 Thread.sleep(1000);
           }
```

```
            System.out.println(res.get());


    Runnable r = new Runnable()
            {
                  @Override
                  public void run()
                  {
                        System.out.println("OK");
                  }
            };
            /* Ставимо завдання із затримкою в одну секунду між закінченням виконання завдання і початком наступного, перший старт через секунду */

  executor.scheduleWithFixedDelay(r,        1,        2, TimeUnit.SECONDS);

            Thread.sleep(7000);
            /* Метод scheduleAtFixedRate не бере до уваги час виконання завдання. Так, якщо ви поставите завдання, яке виконується дві секунди, з інтервалом в одну, пул потоків рано чи пізно переповниться */
     executor.scheduleAtFixedRate(r,        1,        2, TimeUnit.SECONDS);
            executor.shutdown();
```

Синхронізація ресурсу ключовим словом synchronized накладає досить жорсткі правила на звільнення цього ресурсу. Інтерфейс Lock є деяким узагальненням синхронізації. З'являється можливість провести опитування про блокування, встановити час очікування блокування та умови її переривання. Інтерфейс також оптимізує роботу JVM процесами конкурування за ресурси, що звільняються.

Клас ReentrantLock представляє два основні методи:

void lock () - отримує блокування екземпляру. Якщо екземпляр блокований іншим потоком, то потік відключається та не діє до звільнення екземпляра;

void unlock () - звільняє блокування екземпляру. Якщо поточний потік не є власником блокування, генерується виключення IllegalMonitorStateException.

Шаблонне застосування цих методів після оголошення екземпляра locking класу ReentrantLock:

```
 locking.lock();
try {
 // some code here
finally {
 locking.unlock();
```

Дана конструкція копіює функціональність блоку synchronized.

ЗАВДАННЯ ДО РОБОТИ

Ознайомитися з теоретичними відомостями та рекомендованою літературою, необхідною для виконання роботи.

Розробити багатопоточний додаток. Використовувати можливості, що надаються пакетом java.util.concurrent. Не застосовувати слово synchronized. Всі об’єкти, що бажають отримати доступ до ресурсу, повинні бути потоками. Використовувати можливості ООП. Не застосовувати графічний інтерфейс. Додаток має бути консольним.

Оформити звіт з роботи.

Варіанти завдань

Варіант 1:

Порт. Кораблі заходять в порт для розвантаження/завантаження контейнерів. Число контейнерів, що знаходяться в поточний момент в порту та на кораблі, має бути невід'ємним та перевищує задану вантажопідйомність судна і місткість порту. У порту працює кілька причалів. У одного причалу може стояти один корабель. Корабель може завантажуватися біля причалу, розвантажуватися або виконувати обидві дії.

Варіант 2:

Маленька бібліотека. Доступні для читання кілька книг. Однакових книг в бібліотеці немає. Деякі видаються на руки, деякі тільки в читальний зал. Читач може брати на руки і в читальний зал кілька книг.

Варіант 3:

Автомийка. Доступно декілька місць для миття машин. На одному місці може знаходитися тільки один автомобіль. Мийка одного автомобіля займає фіксований час. Якщо всі місця зайняті, то автомобіль не стане чекати більше певного часу та поїде на іншу автомийку.

Варіант 4:

CallCenter. В організації працює декілька операторів. Оператор може обслуговувати тільки одного клієнта, інші повинні чекати своєї

черги. Клієнт може покласти трубку та передзвонити ще раз через деякий час.

Варіант 5:

Автобусні зупинки. На маршруті декілька зупинок. На одній зупинці може зупинятися декілька автобусів одночасно, але не більше заданого числа.

Варіант 6:

Вільна каса. У ресторані швидкого обслуговування є декілька кас. Відвідувачі стоять в черзі в конкретну касу, але можуть перейти в іншу чергу при зменшенні або зникненні там черги.

Варіант 7:

Тунель. В горах існує два залізничних тунелі, за якими поїзди можуть рухатися в обох напрямках. По обох кінцях тунелю зібралося багато поїздів. Необхідно забезпечити безпечне проходження тунелів в обох напрямках. Поїзд можна перенаправити з одного тунелю в інший при перевищенні заданого часу очікування на проїзд.

Варіант 8:

Банк. Є банк з касирами, клієнтами та їх рахунками. Клієнт може знімати / поповнювати / переводити / оплачувати / обмінювати грошові кошти. Касир послідовно обслуговує клієнтів. Потік-спостерігач стежить, щоб в касах завжди була готівка, при скупченні грошей більш певної суми, частина їх перекладається в сховище, при виснаженні запасів готівки відбувається поповнення зі сховища.

Варіант 9:

Аукціон. На торги виставляється декілька лотів. Учасники аукціону роблять заявки. Заявку можна коригувати в бік збільшення кількість разів за торги одного лота. Аукціон визначає переможця та переходить до наступного лоту. Учасник, що не заплатив за лот в заданий проміжок часу, буде відсторонений на кілька лотів від торгів.

Варіант 10:

Прокат лиж. Доступно для прокату певна кількість лиж. Однакових лиж у прокаті немає. Деякі видаються на день, деякі можуть

бути видані на кілька днів але не більше тижня. Клієнт може брати у прокат декілька пар лиж одночасно.

Варіант 11:

Аеропорт. Посадка / висадка пасажирів може здійснюватися через кінцеве число терміналів та наземним способом через кінцеве число трапів. Літаки бувають різної місткості та дальності польоту. Організувати функціонування аеропорту, якщо пунктів призначення 4-6, та зон дальності 2-3.

Варіант 12:

Паркування. Доступно декілька місць для машин. На одному місці може знаходитися тільки один автомобіль. Якщо всі місця зайняті, то автомобіль не стане чекати більше певного часу та поїде на іншу стоянку.

Варіант 13:

Обмін валют. Є пункт обміну валют з декількома касирами. Клієнти можуть обмінювати грошові кошти. Кожен касир послідовно обслуговує клієнтів. На початок дня в касі є певна сума у кожній валюті (гривня, долар, євро). Далі проходить обмін в межах наявних коштів. Якщо сума не достатня, клієнт чекає певний час, та обмінює скільки є в наявності в касі.

Варіант 14:

Медогляд. В поліклініці працює декілька профільних лікарів на медогляді. Кожен відвідувач повинен пройти всіх лікарів. Відвідувачі стоять в черзі до конкретного лікаря, але можуть перейти в іншу чергу при зменшенні або зникненні там черги. Лікар може обслуговувати тільки одного пацієнта, інші повинні чекати своєї черги. На одного пацієнта лікар тратить фіксований час.

Варіант 15:

Автозаправна станція. Доступно декілька місць для заправки автомобіля. На одному місці може знаходитися тільки один автомобіль. заправка одного автомобіля займає фіксований час. Якщо всі місця зайняті, то автомобіль не стане чекати більше певного часу та поїде на іншу автозаправну станцію.

Варіант 16:

Супермаркет. У супермаркеті є декілька кас. Частина кас приймає тільки готівку. Покупці можуть мати або готівку, або платіжну картку, або і те й інше. Відвідувачі стоять в черзі в конкретну касу, але можуть перейти в іншу чергу при зменшенні або зникненні там черги при умові, що зможуть там заплатити.

Варіант 17:

Екскурсія. На екскурсійному маршруті декілька зупинок. На одній зупинці може зупинятися декілька екскурсійних груп одночасно, але не більше заданого числа. Кожна група на одній зупинці тратить певний час на екскурсію та фотографування.

Варіант 18:

Біржа. На торгах брокери пропонують акції кількох фірм. На біржі здійснюються дії з купівлі-продажу акцій. Залежно від кількості проданих-куплених акцій їх ціна змінюється. Брокери пропонують до продажу деяку частину акцій. Від активності та зростання-падіння котирувань акцій змінюється індекс біржі. Біржа може призупинити торги при різкому падінні індексу.

Варіант 19:

Вантажні перевезення. Вантажівки приїжджають на склад для розвантаження/завантаження контейнерів. Число контейнерів, що знаходяться в поточний момент на складі та у вантажівці, має бути невід'ємним та перевищує задану вантажопідйомність вантажівки та місткість складу. На складі є кілька терміналів. У одного термінала може стояти одна вантажівка. Вантажівка може завантажуватися біля терміналу, розвантажуватися або виконувати обидві дії.

Варіант 20:

Булочна. У булочній є декілька кас. Відвідувачі стоять в черзі в конкретну касу, але можуть перейти в іншу чергу при зменшенні або зникненні там черги.

ЗМІСТ ЗВІТУ

1 Титульний лист.

2 Завдання на ІДЗ.

3 Алгоритм роботи програми згідно завданню.

4 Лістинг розробленої програми згідно завданню.

5 Результати роботи програми та їх інтерпретація.

6 Виписати усі функції для організації багатопоточних обчислень на мові Java, використані у даній лабораторній роботі, навести їх короткі описи.

7 Відповіді на контрольні питання. Вміти їх пояснювати в усній формі.

8 Висновки, що відображають результати виконання роботи та їх критичний аналіз.

КОНТРОЛЬНІ ЗАПИТАННЯ

1 Структура пакету java.util.concurency.*
2 Поняття та види Concurrent Collections.
3 Черги та їх види.
4 Структура класів для активного управління потоками.
5 Асинхронне виконання. Future and Callable класи.
6 Інтерфейси виконувача.
7 Засоби синхронізації ресурсу.
8 Використання паралелізму у Java.

ЛІТЕРАТУРА

1. Блинов, И.Н., Романчик, В. С. Java. Методы программирования : уч.-мет. пособие. — Минск : издательство «Четыре четверти», 2013. — 896 с. ISBN 978-985-7058-30-3.

2. Хорстман, К. С. Java. Библиотека профессионала, том 1. Основы. 9-е изд. / К. С. Хорстман, Г. Корнелл. – М : Вильямс, 2014.

3. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 512 с.

4. Гергель, В. П. Высокопроизводительные вычисления для многоядерных многопроцессорных систем: учеб. пособие / В. П. Гергель. – Нижний Новгород : ННГУ им. Н.И. Лобачевского, 2010.

5. Parallel.ru – информационно-аналитический центр по параллельным вычислениям – http://www.parallel.ru.

6. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.

7. Бройнль Т. Паралельне програмування: Початковий курс: Навч. посібник / Вступ. Слово А.Ройтера; Пер. з нім. В.А. Святного. – К.: Вища школа, 1997. – 358 с.

8. Букатов А.А., Дацюк В.Н., Жегуло А.И. Программирование многопроцессорных вычислительных систем. – Ростов-на- Дону: Изд-во ООО «ЦВВР», 2003. – 208 с.

9. Корнеев В.В. Параллельные вычислительные системы. – М.: Нолидж, 1999. – 320 с.