ЛАБОРАТОРНА РОБОТА № 7

Багатопоточне програмування в Java

Мета: практика роботи з потоками в Java

Хід роботи:

Завдання 1. Створити консольний Java проект java_lab_7 з пакетом com.education.ztu.

Завдання 2. Створити клас, що розширює Thread: • Створити клас MyThread, що розширює Thread. • Перевизначити метод run(). У циклі for вивести на консоль повідомлення «Я люблю програмувати!!!» 100 разів. • Створити екземпляр класу та запустити новий потік. • Вивести ім'я створеного потоку, його пріорітет, превірити чи він живий, чи є потоком демоном. • Змінити ім'я, пріорітет створеного потоку та вивести в консоль оновлені значення. • Після завершення роботи створеного потоку (використати метод join()) вивести ім’я головного потоку, та його пріорітет. • Відобразити в консолі, коли ваш потік буде в стані NEW, RUNNUNG, TERMINATED.

Завдання 3. Створити клас, що реалізує інтерфейс Runnable для виводу в консоль чисел від 0 до 10000, що діляться на 10 без залишку: • Створити клас MyRunnable, який реалізує інтерфейс Runnable. • Імплементувати метод run(). • Визначити умову, якщо потік хочуть перервати, то завершити роботу потоку та вивести повідомлення «Розрахунок завершено!!!» • Створити три потоки, які виконують завдання друку значень. • Використовуємо статичний метод Thread.sleep(), щоб зробити паузу на 2 секунди для головного потоку, а після цього викликати для створених потоків метод interrupt().

Завдання 4. Створити клас, що реалізує інтерфейс Runnable для вививедення арифметичної прогресії від 1 до 100 з кроком 1: • Створити клас, який реалізує інтерфейс Runnable. • Створити об'єкт зі статичною змінною result для збереження значення арифметичної прогресії. • Перевизначити метод run().

Створити цикл for. У циклі виводимо через пробіл значення змінної result. Та додаємо наступне значення до змінної result та чекаємо 0,2 секунду. • Забезпечити корректну роботу використовуючи синхронізований метод. • Створити три потоки, які виконують завдання друку значень.

Завдання 5. Переробити 4 завдання використовуючи блок синхронізації.

Завдання 6. Створити два потоки Reader та Printer. Reader зчитує введені дані з консолі та записує в змінну. Після цього інформує потік Printer та засипає на 1 секунду, а потік Reader виводить дотриманий рядок. І так повторюється знову, поки користувач не завершить роботу програми. • Змінну треба використати як об'єкт для синхронізації. • Тут необхідно використати wait() і notify().

Завдання 7. Створити програму для знаходження суми цифр в масиві на 1 000 000 елементів: • Заповнити масив числами використовуючи клас Random. • Реалізувати задачу в однопоточному та багатопоточному середовищі. • Для багатопоточного середовища використати ExecutorService на 5 потоків та об'єкти потоків, що імплементують інтерфейси Runnable або Callable. • Заміряти час виконання обох варіантів завдання використовуючи System.currentTimeMillis() та вивести результати в консоль.

Лістинг програми:

Main.java:

```
package com.education.ztu;

import java.util.concurrent.*;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws InterruptedException,
ExecutionException {
        task2();
        task3();
        task4_5();
        task6();
        task7();
    }

    private static void task2() throws InterruptedException {
        System.out.println("*** Завдання 2: Thread ***");

        MyThread myThread = new MyThread("Custom-Thread-1");

        System.out.println("Ім'я потоку: " + myThread.getName());
        System.out.println("Пріоритет: " + myThread.getPriority());
        System.out.println("Чи живий? " + myThread.isAlive());
```

```
        System.out.println("Чи демон? " + myThread.isDaemon());

        myThread.setName("New-Custom-Thread-Processor");
        myThread.setPriority(Thread.MAX_PRIORITY);

        System.out.println("Оновлена інформація:");
        System.out.println("Нове ім'я потоку: " + myThread.getName());
        System.out.println("Новий пріоритет: " + myThread.getPriority());

        myThread.start();

        myThread.join();

        System.out.println("Потік " + myThread.getName() + " завершив роботу.
Стан: " + myThread.getState());

        Thread mainThread = Thread.currentThread();
        System.out.println("Інформація про головний потік:");
        System.out.println("Ім'я головного потоку: " + mainThread.getName());
        System.out.println("Пріоритет головного потоку: " +
mainThread.getPriority());
        System.out.println("Стан головного потоку: " + mainThread.getState());

        System.out.println("-------------------------");
    }

    private static void task3() throws InterruptedException {
        System.out.println("\n*** Завдання 3: Runnable (interrupt) ***");

        Thread t1 = new Thread(new MyRunnable("Runner-1"), "T3-1");
        Thread t2 = new Thread(new MyRunnable("Runner-2"), "T3-2");
        Thread t3 = new Thread(new MyRunnable("Runner-3"), "T3-3");

        t1.start();
        t2.start();
        t3.start();

        System.out.println("Головний потік чекає 2 секунди...");
        Thread.sleep(2000);

        t1.interrupt();
        t2.interrupt();
        t3.interrupt();

        t1.join();
        t2.join();
        t3.join();

        System.out.println("-------------------------");
    }

    private static void task4_5() throws InterruptedException {
        System.out.println("\n*** Завдання 4 & 5: Арифметична прогресія (Синхроні-
зація) ***");

        Thread t4_1 = new Thread(new MyProgressor(), "P-T1");
        Thread t4_2 = new Thread(new MyProgressor(), "P-T2");
        Thread t4_3 = new Thread(new MyProgressor(), "P-T3");

        t4_1.start();
        t4_2.start();
        t4_3.start();
```

```
        t4_1.join();
        t4_2.join();
        t4_3.join();

        System.out.println("-------------------------");
    }

    private static void task6() throws InterruptedException {
        System.out.println("\n*** Завдання 6: Reader/Printer (wait/notify) ***");

        SharedData sharedData = new SharedData();
        try (Scanner scanner = new Scanner(System.in)) {
            Reader reader = new Reader(sharedData, scanner);
            Printer printer = new Printer(sharedData);

            Thread readerThread = new Thread(reader, "Reader");
            Thread printerThread = new Thread(printer, "Printer");

            readerThread.start();
            printerThread.start();

            readerThread.join();
            printerThread.join();
        }

        System.out.println("-------------------------");
    }

    private static void task7() throws InterruptedException, ExecutionException {
        System.out.println("\n*** Завдання 7: Сума цифр в масиві (ExecutorService)
***");

        final int ARRAY_SIZE = 1_000_000;
        final int NUM_THREADS = 5;

        int[] array = new int[ARRAY_SIZE];
        java.util.Random random = new java.util.Random();
        for (int i = 0; i < ARRAY_SIZE; i++) {
            array[i] = random.nextInt(10000);
        }

        long startTimeSingle = System.currentTimeMillis();
        long singleThreadSum = calculateSumOfDigits(array, 0, ARRAY_SIZE);
        long endTimeSingle = System.currentTimeMillis();

        System.out.println("\n--- Однопотокове обчислення ---");
        System.out.println("Загальна сума: " + singleThreadSum);
        System.out.println("Час виконання: " + (endTimeSingle - startTimeSingle) +
" мс");

        long startTimeMulti = System.currentTimeMillis();
        ExecutorService executor = Executors.newFixedThreadPool(NUM_THREADS);
        int chunkSize = ARRAY_SIZE / NUM_THREADS;
        long multiThreadSum = 0;

        java.util.List<Future<Long>> futures = new java.util.ArrayList<>();

        for (int i = 0; i < NUM_THREADS; i++) {
            int start = i * chunkSize;
            int end = (i == NUM_THREADS - 1) ? ARRAY_SIZE : (i + 1) * chunkSize;

            SumFinder sumFinder = new SumFinder(array, start, end);
            futures.add(executor.submit(sumFinder));
```

```
        }

        for (Future<Long> future : futures) {
            multiThreadSum += future.get();
        }

        executor.shutdown();
        long endTimeMulti = System.currentTimeMillis();

        System.out.println("\n--- Багатопотокове обчислення (Callable) ---");
        System.out.println("Загальна сума: " + multiThreadSum);
        System.out.println("Час виконання: " + (endTimeMulti - startTimeMulti) + "
мс");

        System.out.println("Результати збігаються: " + (singleThreadSum ==
multiThreadSum));

        System.out.println("-------------------------");
    }

    private static long calculateSumOfDigits(int[] array, int start, int end) {
        long sum = 0;
        for (int i = start; i < end; i++) {
            int number = array[i];
            while (number != 0) {
                sum += number % 10;
                number /= 10;
            }
        }
        return sum;
    }
}
```

ArraySummer.java:

```
package com.education.ztu;

public class ArraySummer implements Runnable {
    private final int[] array;
    private final int start;
    private final int end;
    private final int[] partialSum;
    private final int index;

    public ArraySummer(int[] array, int start, int end, int[] partialSum, int
index) {
        this.array = array;
        this.start = start;
        this.end = end;
        this.partialSum = partialSum;
        this.index = index;
    }

    @Override
    public void run() {
        int sum = 0;
        for (int i = start; i < end; i++) {
            int number = array[i];
            while (number != 0) {
                sum += number % 10;
                number /= 10;
            }
        }
        partialSum[index] = sum;
```

```
    }
}
```

MyProgressor.java:

```
package com.education.ztu;

public class MyProgressor implements Runnable {
    private static int result = 1;

    @Override
    public void run() {
        printAndAddSynchronized();
    }
    public static synchronized void printAndAddSynchronized() {
        while (result <= 100) {
            System.out.print(Thread.currentThread().getName() + ": " + result + "
");
            result += 1;
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                return;
            }
        }
        System.out.println("\n" + Thread.currentThread().getName() + " завершив
роботу.");
    }
}
```

MyRunnable.java:

```
package com.education.ztu;

public class MyRunnable implements Runnable {
    private final String name;

    public MyRunnable(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        System.out.println(name + ": Початок розрахунку.");
        try {
            for (int i = 0; i <= 10000; i++) {
                if (Thread.currentThread().isInterrupted()) {
                    throw new InterruptedException();
                }

                if (i % 10 == 0) {
                    System.out.println(name + ": " + i);
                }
            }
        } catch (InterruptedException e) {
            System.out.println("\n*** " + name + ": Розрахунок завершено!!! ***");
            return;
        }
        System.out.println(name + ": Розрахунок завершено (без interrupt).");
    }
}
```

MyThread.java:

```
package com.education.ztu;

public class MyThread extends Thread {
    public MyThread(String name) {
        super(name);
        System.out.println("--- Завдання 2 ---");
        System.out.println("Потік " + this.getName() + " створено. Стан: " +
this.getState()); // NEW
    }

    @Override
    public void run() {
        System.out.println("Потік " + this.getName() + " почав роботу. Стан: " +
this.getState()); // RUNNABLE/RUNNING

        for (int i = 0; i < 100; i++) {
            System.out.println("Я люблю програмувати!!! (потік " + this.getName()
+ ")");
            if (i == 50 && Thread.currentThread().isInterrupted()) {
                System.out.println("Потік " + this.getName() + " перервано.");
                break;
            }
        }

        System.out.println("Потік " + this.getName() + " завершив роботу.");
    }
}
```

Printer.java:

```
package com.education.ztu;

public class Printer implements Runnable {
    private final SharedData sharedData;

    public Printer(SharedData sharedData) {
        this.sharedData = sharedData;
    }

    @Override
    public void run() {
        System.out.println("Printer почав роботу.");

        synchronized (sharedData.lock) {
            while (true) {
                while (!sharedData.isDataAvailable()) {
                    try {
                        System.out.println("[Printer] - Очікування нових да-
них...");
                        sharedData.lock.wait();
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        return;
                    }
                }

                String data = sharedData.getData();

                if ("exit".equalsIgnoreCase(data)) {
                    System.out.println("[Printer] - Отримано команду 'exit'. Заве-
ршення роботи.");
                    sharedData.lock.notifyAll(); // Будимо Reader, щоб він теж за-
вершив роботу
                    break;
```

```
                }

                System.out.println("\n*** [Printer] Отриманий рядок: " + data + "
***");

                sharedData.consumeData();
                sharedData.lock.notifyAll();
            }
        }
    }
}
```

Reader.java:

```
package com.education.ztu;

import java.util.Scanner;

public class Reader implements Runnable {
    private final SharedData sharedData;
    private final Scanner scanner;

    public Reader(SharedData sharedData, Scanner scanner) {
        this.sharedData = sharedData;
        this.scanner = scanner;
    }

    @Override
    public void run() {
        System.out.println("Reader почав роботу. Введіть 'exit' для завершення.");

        synchronized (sharedData.lock) {
            while (true) {
                while (sharedData.isDataAvailable()) {
                    try {
                        System.out.println("[Reader] - Очікування, поки Printer
використає дані...");
                        sharedData.lock.wait();
                    } catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        return;
                    }
                }

                System.out.print("Введіть рядок: ");
                String input = scanner.nextLine();

                if ("exit".equalsIgnoreCase(input)) {
                    sharedData.setData(input); // Повідомляємо про вихід
                    sharedData.lock.notifyAll(); // Будимо Printer
                    break;
                }

                sharedData.setData(input);

                sharedData.lock.notifyAll();

                try {
                    System.out.println("[Reader] - Засинаю на 1 секунду...");
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    break;
                }
```

```
            }
        }
    }
}
```

SharedData.java:

```
package com.education.ztu;

public class SharedData {
    private String data = null; // Змінна для зберігання даних
    private boolean isDataAvailable = false; // Флаг для контролю

    public final Object lock = new Object();

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
        this.isDataAvailable = true;
    }

    public void consumeData() {
        this.data = null;
        this.isDataAvailable = false;
    }

    public boolean isDataAvailable() {
        return isDataAvailable;
    }
}
```

SumFinder.java:

```
package com.education.ztu;

import java.util.concurrent.Callable;

public class SumFinder implements Callable<Long> {
    private final int[] array;
    private final int start;
    private final int end;

    public SumFinder(int[] array, int start, int end) {
        this.array = array;
        this.start = start;
        this.end = end;
    }

    @Override
    public Long call() {
        long sum = 0;
        for (int i = start; i < end; i++) {
            int number = array[i];
            while (number != 0) {
                sum += number % 10;
                number /= 10;
            }
        }
        return sum;
    }
}
```

Результат програми:

```
*** Завдання 2: Thread ***
--- Завдання 2 ---
Потік Custom-Thread-1 створено. Стан: NEW
Ім'я потоку: Custom-Thread-1
Пріоритет: 5
Чи живий? false
Чи демон? false
Оновлена інформація:
Нове ім'я потоку: New-Custom-Thread-Processor
Новий пріоритет: 10
Потік New-Custom-Thread-Processor почав роботу. Стан: RUNNABLE
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
Я люблю програмувати!!! (потік New-Custom-Thread-Processor)
```

Рис. 1 Завдання 2 (100 повідомлень “я люблю програмувати”)

```
*** Завдання 3: Runnable (interrupt) ***
Головний потік чекає 2 секунди...
Runner-1: Початок розрахунку.
Runner-2: Початок розрахунку.
Runner-3: Початок розрахунку.
Runner-2: 0
Runner-2: 10
Runner-2: 20
```

Рис. 2 Завдання 3

```
*** Завдання 4 & 5: Арифметична прогресія (Синхронізація) ***
P-T1: 1 P-T1: 2 P-T1: 3 P-T1: 4 P-T1: 5 P-T1: 6 P-T1: 7 P-T1: 8 P-T1:
P-T1 завершив роботу.
```

Рис. 3 Завдання 4-5

```
*** Завдання 6: Reader/Printer (wait/notify) ***
Reader почав роботу. Введіть 'exit' для завершення.
Введіть рядок: Printer почав роботу.
перший рядок
[Reader] - Засинаю на 1 секунду...
[Reader] - Очікування, поки Printer використає дані...

*** [Printer] Отриманий рядок: перший рядок  ***
[Printer] - Очікування нових даних...
Введіть рядок: другий рядок
[Reader] - Засинаю на 1 секунду...
[Reader] - Очікування, поки Printer використає дані...

*** [Printer] Отриманий рядок: другий рядок  ***
[Printer] - Очікування нових даних...
Введіть рядок: exit
[Printer] - Отримано команду 'exit'. Завершення роботи.
------------------------
```

Рис. 4 Завдання 6

```
*** Завдання 7: Сума цифр в масиві (ExecutorService) ***

--- Однопотокове обчислення ---
Загальна сума: 18003665
Час виконання: 9 мс

--- Багатопотокове обчислення (Callable) ---
Загальна сума: 18003665
Час виконання: 17 мс
Результати збігаються: true
-------------------------
```

Рис. 5 Завдання 7

Посилання на репозиторій: https://github.com/Sasha1845/Java

Висновок: Я попрактикував роботи з потоками в Java