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.

Лістинг програми:

```
package com.education.ztu;

class MyThread extends Thread {
    @Override
    public void run() {
        System.out.println("Потік " + getName() + " запущений (RUNNING).");
        for (int i = 1; i <= 100; i++) {
            System.out.println(i + ": Я люблю програмувати!!!");
        }
        System.out.println("Потік " + getName() + " завершив роботу
(TERMINATED).");
    }
}

public class Task_2 {
    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        System.out.println("Створено потік " + myThread.getName() + " (NEW)");

        System.out.println("Початковий пріоритет: " + myThread.getPriority());
        System.out.println("Чи живий? " + myThread.isAlive());
        System.out.println("Чи демон? " + myThread.isDaemon());
```

```
        myThread.setName("MyCustomThread");
        myThread.setPriority(Thread.MAX_PRIORITY);
        System.out.println("Оновлене ім'я: " + myThread.getName());
        System.out.println("Оновлений пріоритет: " + myThread.getPriority());

        myThread.start();

        try {
            myThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        Thread mainThread = Thread.currentThread();
        System.out.println("Головний потік: " + mainThread.getName());
        System.out.println("Пріоритет головного потоку: " +
mainThread.getPriority());
    }
}
```

Результат виконання:

```
92: Я люблю програмувати!!!
93: Я люблю програмувати!!!
94: Я люблю програмувати!!!
95: Я люблю програмувати!!!
96: Я люблю програмувати!!!
97: Я люблю програмувати!!!
98: Я люблю програмувати!!!
99: Я люблю програмувати!!!
100: Я люблю програмувати!!!
Потік MyCustomThread завершив роботу (TERMINATED).
Головний потік: main
Пріоритет головного потоку: 5
```

Рис. 1 Результат виконання звдання №2

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().

Лістинг програми:

```
package com.education.ztu;

class MyRunnable implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i <= 10000; i++) {
            if (Thread.currentThread().isInterrupted()) {
```

```
                System.out.println("Розрахунок завершено!!!");
                return;
            }
            if (i % 10 == 0) {
                System.out.println(Thread.currentThread().getName() + ": " + i);
            }
        }
    }
}

public class Task_3 {
    public static void main(String[] args) {
        Thread t1 = new Thread(new MyRunnable(), "Thread-1");
        Thread t2 = new Thread(new MyRunnable(), "Thread-2");
        Thread t3 = new Thread(new MyRunnable(), "Thread-3");

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

        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

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

Результат виконання:

```
Thread-1: 9870
Thread-1: 9880
Thread-1: 9890
Thread-1: 9900
Thread-1: 9910
Thread-1: 9920
Thread-1: 9930
Thread-1: 9940
Thread-1: 9950
Thread-1: 9960
Thread-1: 9970
Thread-1: 9980
Thread-1: 9990
Thread-1: 10000
```

Рис. 2 Результат виконання звдання №3

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 зна-

чення до змінної result та чекаємо 0,2 секунду. • Забезпечити коректну роботу використовуючи синхронізований метод. • Створити три потоки, які виконують завдання друку значень.

Лістинг програми:

```
package com.education.ztu;

class ArithmeticRunnable implements Runnable {
    private static int result = 1;

    private static synchronized void printNext() {
        System.out.print(result + " ");
        result++;
    }

    @Override
    public void run() {
        while (true) {
            synchronized (ArithmeticRunnable.class) {
                if (result > 100) break;
                printNext();
            }
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                return;
            }
        }
    }
}

public class Task_4 {
    public static void main(String[] args) {
        Thread t1 = new Thread(new ArithmeticRunnable(), "Thread-1");
        Thread t2 = new Thread(new ArithmeticRunnable(), "Thread-2");
        Thread t3 = new Thread(new ArithmeticRunnable(), "Thread-3");

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

Результат виконання:

```
5.2.2\lib\idea_rt.jar=62506" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath C:\Users\User\Desktop\Java-IPZ-23-1\Lab-7\out\production\Lab-7 com.education.
7 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

Рис. 3 Результат виконання звадання №4

Завдання 5. Переробити 4 завдання використовуючи блок синхронізації.

Лістинг програми:

```
package com.education.ztu;

class ArithmeticRunnableSync implements Runnable {
    private static int result = 1;
```

```
    @Override
    public void run() {
        while (true) {
            synchronized(ArithmeticRunnableSync.class) {
                if (result > 100) break;
                System.out.print(result + " ");
                result++;
            }
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                return;
            }
        }
    }
}

public class Task_5 {
    public static void main(String[] args) {
        Thread t1 = new Thread(new ArithmeticRunnableSync(), "Arithmetic-1");
        Thread t2 = new Thread(new ArithmeticRunnableSync(), "Arithmetic-2");
        Thread t3 = new Thread(new ArithmeticRunnableSync(), "Arithmetic-3");

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

Результат виконання:

```
5.2.2\lib\idea_rt.jar=62518" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath C:\Users\User\Desktop\Java-IPZ-23-1\Lab-7\out\production\Lab-7 com.education.
7 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

Рис. 4 Результат виконання звдання №5

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().

Лістинг програми:

```
package com.education.ztu;

import java.util.Scanner;

class SharedData {
    String data;
    boolean available = false;
}

class Reader extends Thread {
    private final SharedData shared;
    private final Scanner scanner;

    public Reader(SharedData shared) {
        this.shared = shared;
```

```
        this.scanner = new Scanner(System.in);
    }

    @Override
    public void run() {
        while (true) {
            synchronized (shared) {
                while (shared.available) {
                    try {
                        shared.wait();
                    } catch (InterruptedException e) {
                        return;
                    }
                }
                System.out.print("Введіть текст (exit для виходу): ");
                String input = scanner.nextLine();
                if (input.equalsIgnoreCase("exit")) {
                    System.exit(0);
                }
                shared.data = input;
                shared.available = true;
                shared.notify();
            }
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                return;
            }
        }
    }
}

class Printer extends Thread {
    private final SharedData shared;

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

    @Override
    public void run() {
        while (true) {
            synchronized (shared) {
                while (!shared.available) {
                    try {
                        shared.wait();
                    } catch (InterruptedException e) {
                        return;
                    }
                }
                System.out.println("Вивід: " + shared.data);
                shared.available = false;
                shared.notify();
            }
        }
    }
}

public class Task_6 {
    public static void main(String[] args) {
        SharedData shared = new SharedData();
        Reader reader = new Reader(shared);
        Printer printer = new Printer(shared);
```

```
        reader.start();
        printer.start();
    }
}
```

Результат виконання:

```
Введіть текст (exit для виходу): Java !!!
Вивід: Java !!!
Введіть текст (exit для виходу): exit
```

Рис. 5 Результат виконання звдання №6

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:

```
package com.education.ztu;

import java.util.Random;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.ArrayList;
import java.util.List;

public class Task_7 {

    private static int[] generateArray(int size) {
        Random random = new Random();
        int[] array = new int[size];
        for (int i = 0; i < size; i++) {
            array[i] = random.nextInt(1000); // числа від 0 до 999
        }
        return array;
    }

    private static long sumDigitsSingleThread(int[] array) {
        long sum = 0;
        for (int num : array) {
            while (num > 0) {
                sum += num % 10;
                num /= 10;
            }
        }
```

```
        return sum;
    }

    private static long sumDigitsMultiThread(int[] array, int numThreads) throws
InterruptedException, ExecutionException {
        ExecutorService executor = Executors.newFixedThreadPool(numThreads);
        List<Future<Long>> futures = new ArrayList<>();
        int chunkSize = array.length / numThreads;

        for (int i = 0; i < numThreads; i++) {
            final int start = i * chunkSize;
            final int end = (i == numThreads - 1) ? array.length : start +
chunkSize;

            Callable<Long> task = () -> {
                long sum = 0;
                for (int j = start; j < end; j++) {
                    int num = array[j];
                    while (num > 0) {
                        sum += num % 10;
                        num /= 10;
                    }
                }
                return sum;
            };
            futures.add(executor.submit(task));
        }

        long totalSum = 0;
        for (Future<Long> future : futures) {
            totalSum += future.get();
        }

        executor.shutdown();
        return totalSum;
    }

    public static void main(String[] args) throws InterruptedException,
ExecutionException {
        int size = 1_000_000;
        int[] array = generateArray(size);

        long startSingle = System.currentTimeMillis();
        long sumSingle = sumDigitsSingleThread(array);
        long endSingle = System.currentTimeMillis();
        System.out.println("Однопотокова сума цифр: " + sumSingle);
        System.out.println("Час виконання однопотокового: " + (endSingle -
startSingle) + " мс");

        long startMulti = System.currentTimeMillis();
        long sumMulti = sumDigitsMultiThread(array, 5);
        long endMulti = System.currentTimeMillis();
        System.out.println("Багатопотокова сума цифр: " + sumMulti);
        System.out.println("Час виконання багатопотокового: " + (endMulti -
startMulti) + " мс");
    }
}
```

Результат виконання:

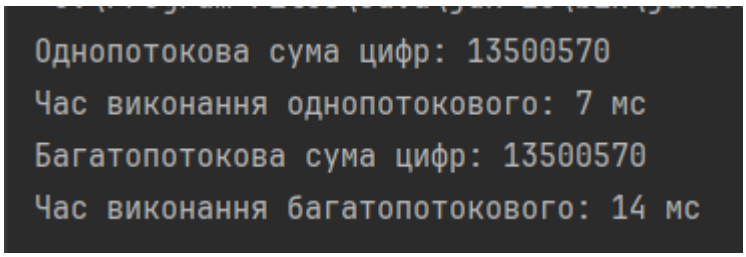

Рис. 6 Результат виконання звдання №7

Посилання на репозиторій: https://github.com/Oleg-Ischuk/Java-IPZ-23-1

Висновок: попрактикувався з потоками в Java.