Лабораторна робота №7

Тема: Багатопоточне програмування в Java

Мета роботи: 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:

Main.java:

```
package com.education.ztu.Task2;


public class Main {
    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        System.out.println("Потік після створення: " + myThread.getState());

        myThread.start();
        System.out.println("Потік запущений...");
```

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

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

        try {
            myThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Стан потоку після завершення: " +
myThread.getState());

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

MyThread.java:

```
package com.education.ztu.Task2;

public class MyThread extends Thread {
        @Override
        public void run() {
            for (int i = 1; i <= 100; i++) {
                System.out.println("Я люблю програмувати!!! " + i);
            }
        }
}
```

Результат виконання програми:

```
Потік після створення: NEW
Потік запущений...
Ім'я потоку: Thread-0
Пріоритет потоку: 5
Я люблю програмувати!!! 1
Я люблю програмувати!!! 2
Я люблю програмувати!!! 3
Я люблю програмувати!!! 4
Я люблю програмувати!!! 5
Я люблю програмувати!!! 6
Я люблю програмувати!!! 7
Я люблю програмувати!!! 8
Чи живий потік? true
Я люблю програмувати!!! 9
Я люблю програмувати!!! 10
Я люблю програмувати!!! 11
Я люблю програмувати!!! 12
Чи є потік демоном? false
```

```
Я люблю програмувати!!! 99
Я люблю програмувати!!! 100
Стан потоку після завершення: TERMINATED
Ім'я головного потоку: main
Пріоритет головного потоку: 5
```

Рис.1 Завдання 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:

Main.java:

```
package com.education.ztu.Task3;


public class Main {
    public static void main(String[] args) {
        MyRunnable task = new MyRunnable();

        Thread thread1 = new Thread(task, "Thread-1");
        Thread thread2 = new Thread(task, "Thread-2");
        Thread thread3 = new Thread(task, "Thread-3");

        thread1.start();
        thread2.start();
        thread3.start();

        try {
            Thread.sleep(2000);
            System.out.println("Перериваємо потоки...");
            thread1.interrupt();
            thread2.interrupt();
            thread3.interrupt();
            thread1.join();
            thread2.join();
            thread3.join();
            System.out.println("Розрахунок завершено!!!");
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

MyRunnable.java:

```
package com.education.ztu.Task3;
```

```
public class MyRunnable implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i <= 10000; i += 10) {
            if (Thread.currentThread().isInterrupted()) {
                System.out.println(Thread.currentThread().getName() + " -
Розрахунок завершено!!!");
                return;
            }
            System.out.println(Thread.currentThread().getName() + ": " + i);
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                System.out.println(Thread.currentThread().getName() + " -
Розрахунок завершено!!!");
                return;
            }
        }
    }
}
```

Результат виконання програми:

```
Thread-3: 1250
Thread-2: 1250
Thread-3: 1260
Thread-2: 1260
Thread-1: 1260
Перериваємо потоки...
Thread-2 - Розрахунок завершено!!!
Thread-3 - Розрахунок завершено!!!
Thread-1 - Розрахунок завершено!!!
Розрахунок завершено!!!
```

Рис.2 Завдання 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 значення до змінної result та чекаємо 0,2 секунду.

- Забезпечити коректну роботу використовуючи синхронізований метод.
- Створити три потоки, які виконують завдання друку значень

Лістинг програми:

Main.java:

```
package com.education.ztu.Task4;

public class Main {
    public static void main(String[] args) {
        MyRunnable task = new MyRunnable();

        Thread thread1 = new Thread(task, "Thread-1");
        Thread thread2 = new Thread(task, "Thread-2");
        Thread thread3 = new Thread(task, "Thread-3");
        thread1.start();
        thread2.start();
        thread3.start();

        try {
            thread1.join();
            thread2.join();
            thread3.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("\nВсі потоки завершили роботу!");
    }
}
```

MyRunnable.java:

```
package com.education.ztu.Task4;

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

    @Override
    public void run() {
        for (int i = 1; i <= 100; i++) {
            printProgression();
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    private synchronized void printProgression() {
        System.out.print(result + " ");
        result++;
    }
}
```

Результат виконання програми:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 1
Всі потоки завершили роботу!
```

Рис.3 Завдання 4

Завдання 5. Переробити 4 завдання використовуючи блок синхронізації.

Лістинг програми:

Main.java:

```
package com.education.ztu.Task5;

public class Main {
    public static void main(String[] args) {
        MyRunnable task = new MyRunnable();

        Thread thread1 = new Thread(task, "Thread-1");
        Thread thread2 = new Thread(task, "Thread-2");
        Thread thread3 = new Thread(task, "Thread-3");
        thread1.start();
        thread2.start();
        thread3.start();

        try {
            thread1.join();
            thread2.join();
            thread3.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("\nВсі потоки завершили роботу!");
    }
}
```

MyRunnable.java:

```
package com.education.ztu.Task5;

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

    @Override
    public void run() {
        for (int i = 1; i <= 100; i++) {
            synchronized (MyRunnable.class) {
                System.out.print(result + " ");
                result++;
            }
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Результат виконання програми:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 1
Всі потоки завершили роботу!
```

Рис.4 Завдання 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:

Main.java:

```
package com.education.ztu.Task6;

public class Main {
    public static void main(String[] args) {
        SharedResource sharedResource = new SharedResource();

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

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

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

        System.out.println("Програма завершена.");
    }
}
```

Printer.java:

```
package com.education.ztu.Task6;

public class Printer implements Runnable {
    private SharedResource sharedResource;

    public Printer(SharedResource sharedResource) {
        this.sharedResource = sharedResource;
    }

    @Override
    public void run() {
        while (true) {
            String data = sharedResource.getData();
```

```
            if (data.equalsIgnoreCase("exit")) {
                System.out.println("Програма завершується...");
                break;
            }

            System.out.println("Виведено: " + data);

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Reader.java:

```
package com.education.ztu.Task6;

import java.util.Scanner;

public class Reader implements Runnable {
    private SharedResource sharedResource;
    private Scanner scanner;

    public Reader(SharedResource sharedResource) {
        this.sharedResource = sharedResource;
        this.scanner = new Scanner(System.in);
    }

    @Override
    public void run() {
        while (true) {
            System.out.print("Введіть рядок: ");
            String input = scanner.nextLine();

            if (input.equalsIgnoreCase("exit")) {
                sharedResource.setData(input);
                break;
            }
            sharedResource.setData(input);

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

SharedResource.java:

```
package com.education.ztu.Task6;

public class SharedResource {
    private String data = null;

    public synchronized void setData(String data) {
        this.data = data;
        notify();
    }
```

```
    public synchronized String getData() {
        try {
            while (data == null) {
                wait();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        String result = data;
        data = null;
        return result;
    }
}
```

Результат виконання програми:

```
Введіть рядок: hello
Виведено: hello
Введіть рядок: hi
Виведено: hi
Введіть рядок: exit
Програма завершується...
Програма завершена.
```

Рис.5 Завдання 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:

Main.java:

```
package com.education.ztu.Task7;

import java.util.Random;
import java.util.concurrent.ExecutionException;

public class Main {
    public static void main(String[] args) throws InterruptedException,
ExecutionException {
        int arraySize = 1000000;
        int[] array = new int[arraySize];
        Random random = new Random();
```

```
        for (int i = 0; i < arraySize; i++) {
            array[i] = random.nextInt(10000);
        }

        long startTime = System.currentTimeMillis();
        long sumSingleThread = SingleThreadSum.sumDigitsSingleThread(array);
        long endTime = System.currentTimeMillis();
        System.out.println("Однопотокова сума цифр: " + sumSingleThread);
        System.out.println("Час виконання в однопоточному режимі: " + (endTime -
startTime) + " мс");

        startTime = System.currentTimeMillis();
        long sumMultiThread = MultiThreadSum.sumDigitsMultiThread(array, 5);
        endTime = System.currentTimeMillis();
        System.out.println("Багатопотокова сума цифр: " + sumMultiThread);
        System.out.println("Час виконання в багатопоточному режимі: " + (endTime
- startTime) + " мс");
    }
}
```

MultiThreadSum.java:

```
package com.education.ztu.Task7;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.*;

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

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

            futures.add(executor.submit(() -> {
                long sum = 0;
                for (int j = startIndex; j < endIndex; j++) {
                    sum += sumDigits(array[j]);
                }
                return sum;
            }));
        }

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

        executor.shutdown();
        return totalSum;
    }

    private static int sumDigits(int number) {
        int sum = 0;
        while (number > 0) {
            sum += number % 10;
```

```
            number /= 10;
        }
        return sum;
    }
}
```

SingleThreadSum.java:

```
package com.education.ztu.Task7;

public class SingleThreadSum {
    public static long sumDigitsSingleThread(int[] array) {
        long sum = 0;
        for (int num : array) {
            sum += sumDigits(num);
        }
        return sum;
    }

    private static int sumDigits(int number) {
        int sum = 0;
        while (number > 0) {
            sum += number % 10;
            number /= 10;
        }
        return sum;
    }
}
```

Результат виконання програми:

```
Однопотокова сума цифр: 17992374
Час виконання в однопоточному режимі: 19 мс
Багатопотокова сума цифр: 17992374
Час виконання в багатопоточному режимі: 33 мс
```

Рис.6 Завдання 7

Висновок: під час виконання лабораторної роботи я попрактикувала роботу з потоками в Java.