Imię i Nazwisko Patryk Twardosz	Kierunek Informatyka Techniczna	Rok studiów i grupa I rok, Gr. 9
Data zajęć 13.11.2024r.	Numer i temat sprawozdania Lab 6 i 7 – Java	

Lab6:

Cel:

- Opanowanie podstaw tworzenia wątków w Javie.
- Opanowanie podstawowych metod synchronizacji w Javie

fork() - Tworzy nowy proces jako kopię procesu rodzica.

clone() - Bardziej elastyczna funkcja, umożliwiająca większą kontrolę nad współdzielonymi zasobami.

Zadanie:

1. Przygotowanie projektu
2. Obliczanie histogramu odbywało się w sposób równoległy, przy użyciu wątków Javy

   - kalasa wątku i funkcja obliczająca histogram dla danego symbolu:

```
package dev.twardosz.pr;

public class Watek1 extends Thread {

    private final Obraz obraz;
    private final char symbol;

    public Watek1(Obraz obraz, char symbol) {
        this.obraz = obraz;
        this.symbol = symbol;
    }

    public void run() {
        obraz.calculate_histogram_parallel1(symbol);
    }
}


public void calculate_histogram_parallel1(char symbol) {
    for(int i=0;i<size_n;i++) {
        for(int j=0;j<size_m;j++) {
            if(tab[i][j] == symbol)
                histogram[symbol - 33]++;
        }
    }
}
```

```
static Obraz parallel1(Obraz obraz0) {
    var threads = new Thread[94];
    Obraz obraz = new Obraz(obraz0);

    for (int i = 0; i < 94; i++) {
        (threads[i] = new Watek1(obraz, (char)(i+33))).start();
    }

    await(threads);

    return obraz;
}
```

3. Wariant 2. z podziałem liter na podzbiory

```
static Obraz parallel2(Obraz obraz0, int num_threads) {
    var threads = new Thread[num_threads];
    Obraz obraz = new Obraz(obraz0);

    int symbolCount = 94;
    int perThread = symbolCount / num_threads;
    int remainder = symbolCount % num_threads;


    for (int i = 0; i < num_threads; i++) {
        char symbolStart = (char)(i * perThread);
        char symbolEnd = (char)(symbolStart + perThread + (i ==
num_threads - 1 ? remainder : 0));

        (threads[i] = new Thread(() ->
obraz.calculate_histogram_parallel2(symbolStart, symbolEnd))).start();
    }

    await(threads);

    return obraz;
}



public void calculate_histogram_parallel2(char symbolStart, char
symbolEnd) {
    for(int x = 0; x < size_n; x++)
        for (int y = 0; y < size_m; y++)
            for(int k = symbolStart; k < symbolEnd; k++)
                if (tab[x][y] == tab_symb[k])
                    histogram[k]++;
}
```

4. Wariant 3. z podziałem danych na bloki

```
static Obraz parallel3(Obraz obraz0, int num_threads) {
    var threads = new Thread[num_threads];
    Obraz obraz = new Obraz(obraz0);

    int m = obraz.size(1);
    int perThread = obraz.size(0) / num_threads;
    int remainder = obraz.size(0) % num_threads;

    for (int i = 0; i < num_threads; i++) {
        int start = i * perThread;
        int end = start + perThread + (i == num_threads - 1 ? remainder :
0);

        (threads[i] = new Thread(() ->
obraz.calculate_histogram_parallel3(start, end, 1, 0, m, 1))).start();
    }

    await(threads);

    return obraz;
}



public void calculate_histogram_parallel3(int x1, int x2, int dx, int y1,
int y2, int dy) {
    int[] localHistogram = new int[histogram.length];
    for (int x = x1; x < x2; x += dx)
        for (int y = y1; y < y2; y += dy)
            for(int k = 0; k < 94; k++)
                if (tab[x][y] == tab_symb[k])
                    localHistogram[k]++;

    synchronized (histogram) {
        for (int i = 0; i < histogram.length; i++)
            histogram[i] += localHistogram[i];
    }
}
```

5. Wariant 4. z cyklicznym podziałem danych

```
static Obraz parallel4(Obraz obraz0, int num_threads) {
    var threads = new Thread[num_threads];
    Obraz obraz = new Obraz(obraz0);

    int n = obraz.size(0);
    int m = obraz.size(1);

    for (int i = 0; i < num_threads; i++) {
        int start = i;
        int stride = num_threads;
        int end = n;

        (threads[i] = new Thread(() ->
obraz.calculate_histogram_parallel3(start, end, stride, 0, m, 1))).start();
    }

    await(threads);

    return obraz;
}
```

Wnioski:

- Zabezpieczenie ścieżki krytycznie w języku Java zapewniane jest przez użycie *synchronized*
- Zmniejszanie obszaru bloku synchronized pozwala na wydajniejsze wykonywanie obliczeń
- Podział zadań na nie zależne bloki danych pozwala na pozbycie się konieczności istnienia ścieżki krytycznej.

Lab7:

Cel:

- Nabycie umiejętności pisania programów w języku Java z wykorzystaniem puli wątków

Zadanie:

1. Przygotowanie projektu
2. Analiza programu SumaCallable
3. Wykonanie obliczania Całki sekwencyjnie

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.println("Enter dx: ");
    double dx = sc.nextDouble();

    Calka_callable calka = new Calka_callable(0, 3.14, dx);
    calka.compute_integral();
}
```

4. Wykonanie obliczania Całki z wykorzystaniem interfejsu *Callable*, obiektu *Future* i ExecutorService

```
// Implementacja Callable
public class Calka_callable implements Callable<Double> {
    @Override public Double call() {
        return compute_integral();
    }
}

public class Calka_executor_test {
    public static void main(String[] args) {
       ExecutorService executor = Executors.newFixedThreadPool(NTHREADS);
       List<Future<Double>> results = new ArrayList<>();

       double range = (END - START) / NTASKS;

       for (int i = 0; i < NTASKS; i++) {
          double start = START + i * range;
          double end = start + range;
          Callable<Double> task = new Calka_callable(start, end, DX);
          results.add(executor.submit(task));
       }
       executor.shutdown();

       double totalIntegral = 0.0;
       try {
          for (Future<Double> result : results) {
             totalIntegral += result.get();          }
       } catch (Exception e) {
          e.printStackTrace();
       }
       System.out.println("Wynik całki: " + totalIntegral);
    }
}
```

5. Wykonanie implementacji MergeSort przy użyciu ForkJoinPool

```
class DivideTask extends RecursiveTask<int[]> {

    int[] arrayToDivide;

    public DivideTask(int[] arrayToDivide) {
        this.arrayToDivide = arrayToDivide;
    }

    protected int[] compute() {
        if (arrayToDivide.length <= 1)
            return arrayToDivide;

        // Podział tablicy na podtablice
        int mid = arrayToDivide.length / 2;
        int[] leftPart = new int[mid];
        int[] rightPart = new int[arrayToDivide.length - mid];
        // Tworzenie nowych zadań dla podtablic
        DivideTask task1 = new DivideTask(leftPart);
        DivideTask task2 = new DivideTask(rightPart);
        invokeAll(task1, task2);
        //Wait for results from both tasks
        int[] tab1 = task1.join();
        int[] tab2 = task2.join();
        int[] scal_tab = new int[tab1.length + tab2.length];
        scal_tab(tab1, tab2, scal_tab);
        return scal_tab;
    }

    private void scal_tab(int[] tab1, int[] tab2, int[] scal_tab) {
        /* Implementacja była podana */
    }

    public static void main(String[] args) {
        // Testowa tablica
        int[] array = {38, 27, 43, 3, 9, 82, 10};

        // Utworzenie ForkJoinPool
        ForkJoinPool pool = new ForkJoinPool();

        // Rozpoczęcie zadania
        DivideTask mainTask = new DivideTask(array);
        int[] sortedArray = pool.invoke(mainTask);
    }
}
```

Wnioski:

- Efektywność zarządzania wątkami: Zastosowanie puli wątków znacząco zwiększa wydajność aplikacji wielowątkowych, eliminując potrzebę ciągłego tworzenia i niszczenia wątków.
- Optymalne wykorzystanie zasobów: *ThreadPool* umożliwia ograniczenie liczby równoczesnych wątków, co zapobiega nadmiernemu obciążeniu systemu.
- Łatwość implementacji: Mechanizm puli wątków ułatwia implementację równoległości dzięki dostarczonym przez Javę abstrakcjom, takim jak *ExecutorService*.

- Kontrola nad wykonaniem zadań: Dzięki elastycznym metodom, takim jak *submit* czy *invokeAll*, programista ma większą kontrolę nad zarządzaniem zadaniami.
- Zastosowania praktyczne: Mechanizmy *ThreadPool* znajdują zastosowanie w aplikacjach wymagających przetwarzania dużej liczby zadań, takich jak serwery aplikacji czy przetwarzanie danych.
- Praktyczne umiejętności: W laboratorium poznano sposób tworzenia i konfiguracji puli wątków, a także obsługi wyjątków i śledzenia stanu zadań.