# Teoria Współbieżności

## Laboratorium 8 - Asynchroniczne wykonanie zadań w puli wątków przy użyciu wzorców Executor i Future.

Magdalena Pastuła

Data laboratorium: 24.11.2020

## 1. Zadanie do wykonania.

Celem laboratorium jest zapoznanie się z wzorcami Executor i Future. W tym celu należało zaimplementować obliczanie zbioru Mandelbrota, następnie przetestować pod względem czasu wykonania.

## 2. Koncepcja rozwiązania.

Wykonana implementacja wykorzystuje interfejs `Callable` oraz klasę `Future`. W każdym wątku wartość piksela jest obliczana i ustawiana, a następnie zwracana jest liczba 0. Zwracanie stałej wartości może wydawać się niepotrzebne, jednakże dzięki oczekiwaniu na tą wartość główny wątek wie, kiedy wszystkie podrzędne wątki sie zakończyły i sam nie kończy pracy przedwcześnie.

Program wykonano w dwóch wersjach: w pierwszej każdy wątek oblicza wartość tylko jednego danego piksela. W drugiej wersji, jeden wątek oblicza i ustawia wartość pikselów dla podanego wiersza.

## 3. Implementacja i wyniki.

Cały kod źródłowy do tego laboratorium znajduje się w załączonym archiwum.

Implementacja konstruktora głównej klasy:

```
public Mandelbrot(int maxIter, double zoom)
{
    super("Mandelbrot Set");
    setBounds(100, 100, 800, 600);
    setResizable(false);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    I = new BufferedImage(getWidth(), getHeight(), BufferedImage.TYPE_INT_RGB);

    ExecutorService pool = Executors.newWorkStealingPool();
    LinkedList<Future<Integer>> futures = new LinkedList<>();
    long start = System.nanoTime();

    for (int y = 0; y < getHeight(); y++) {
        for (int x = 0; x < getWidth(); x++) {
            ForkMandelbrot mandelbrot = new ForkMandelbrot(I, maxIter, zoom, x,
y);
            Future<Integer> future = pool.submit(mandelbrot);
            futures.add(future);
        }
```

```
    }

    for(Future<Integer> f : futures) {
        try {
            f.get();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    long end = System.nanoTime();

    double elapsedInMilis = (end - start) / 1000000.0;

    System.out.println("Elapsed time: " + elapsedInMilis + " ms");
}
```

Implementacja funkcji `call` z klasy `ForkMaldenbrot`, która oblicza wartość dla jednego piksela:

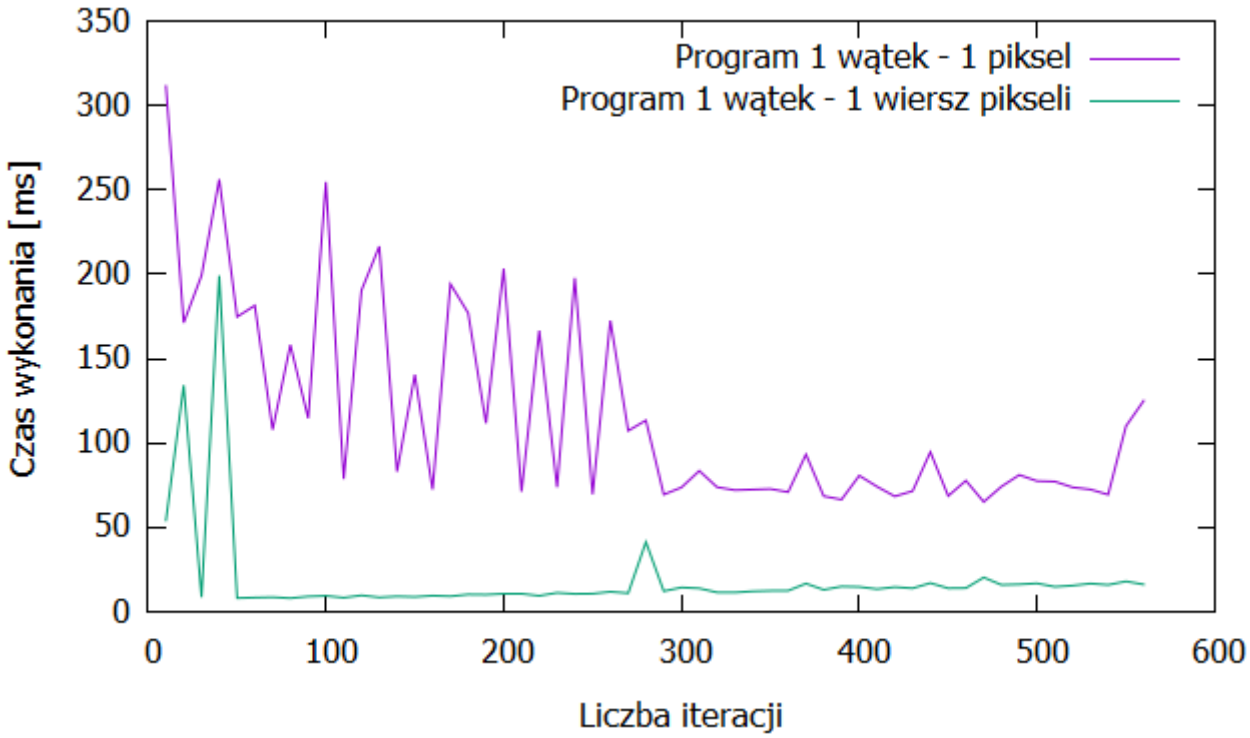
```
public Object call() throws Exception
{
    zx = zy = 0;
    cX = (x - 400) / zoom;
    cY = (y - 300) / zoom;
    int iter = maxIterationsNo;
    while (zx * zx + zy * zy < 4 && iter > 0) {
        tmp = zx * zx - zy * zy + cX;
        zy = 2.0 * zx * zy + cY;
        zx = tmp;
        iter--;
    }
    image.setRGB(x, y, iter | (iter << 8));

    return 0;
}
```

Implementacja funkcji `call` z klasy `ForkMandelbortRow`, która oblicza wartości pikseli w podanym wierszu:

```
public Object call() throws Exception
{
    for (int x=0;x<xWidth;x++)
    {
        zx = zy = 0;
        cX = (x - 400) / zoom;
        cY = (y - 300) / zoom;
        int iter = maxIterationsNo;
        while (zx * zx + zy * zy < 4 && iter > 0) {
            tmp = zx * zx - zy * zy + cX;
            zy = 2.0 * zx * zy + cY;
```

```
                zx = tmp;
                iter--;
            }
            image.setRGB(x, y, iter | (iter << 8));
        }
        return 0;
    }
```

Poniższy wykres przedstawia zależność czasu wykonania programu od liczby iteracji dla dwóch wersji programu, przy czym etykieta "1 wątek - 1 piksel" oznacza program obliczający w danym wątku wartość tylko dla jednego piksela, natomiast etykieta "1 wątek - 1 wiersz pikseli" oznacza program obliczający w jednym wątku wartości pikseli dla całego wiersza.




## 4. Wnioski z ćwiczenia.

Na powyższym wykresie można zauważyć dwie zależności. Po pierwsze, program obliczający w jednym wątku wartości pikseli w całym wierszu jest definitywnie szybszy niż program obliczający w każdym wątku osobno wartość pojedynczego piksela. Można z tego wyciągnąć wniosek, że wykonanie pojedynczego wątku nie powinno być jak najkrótsze, ponieważ wtedy większość czasu wykonania programu to tworzenie wątków i oczekiwanie na ich zakończenie.

Drugą zależnością, którą można zauważy, to fakt, że dla małej liczby iteracji otrzymujemy dłyższy czas wykonania programu. Ponownie może mieć to związek z liczbą operacji do wykonania w wątku, a czasem tworzenia wątku. Jednakże, po wersji liczącej w wątku wartości dla całego wiersza można zauważyć delikatny wzrost czasu wykonania od liczby iteracji, co jest już bardziej zgodne z intuicją.

## 5. Bibliografia.

1. Dokumentacja klasy Executors w Javie.

2. Przykład implementacji wyliczania zbioru Mandelbrota w Javie.
3. Dokumetacja klasy ExecutorService w Javie.
4. Dokumentacja interfejsu Callable.