Programowanie w Javie
Zadanie 3

Programowanie wielowątkowe

Przykład 1

W przykładzie pierwszym skupimy się na przypadku w sumie bezproduktywnego, wielokrotnego (100000 * 2000) wykonywania operacji mnożenia i dzielenia. Każdy z kilku wątków naszego przykładu będzie wykonywał taką procedurę 5 krotnie, za każdym razem informując o swoich postępach. Wątków będzie łącznie 11, przy czym 1 będzie wykonywany z wysokim priorytetem, a pozostałe z małymi.

```
1   package com.company;
2
3   public class WatkiPodstawy extends Thread{
4       private int licznik=0;
5       private double pom=0;
6       public WatkiPodstawy(int priorytet)
7       {
8           setPriority(priorytet);
9           start();
10      }
11      public void run()
12      {
13          while (true)
14          {
15              for (int i=1; i< 100000; i++)
16                  for (int j=1; j<2000; j++)
17                      pom = pom*pom/pom;
18              licznik++;
19              System.out.print(this);
20              System.out.println(" - wykonanych przejsc - "+licznik);
21              if (licznik==5)
22                  return;
23          }
24      }
25
26      public static void main(String[] args) {
27          System.out.println("Starting Watki_1...");
28          new WatkiPodstawy(Thread.MAX_PRIORITY);
29          for (int i = 0; i<10; i++)
30              new WatkiPodstawy(Thread.MIN_PRIORITY);
31      }
32  }
```

3 – początek klasy z przykładu – jak widać klasa dziedziczy po klasie Thread (to pierwsze z dwóch podstawowych podejść). Dzięki temu obiekty tej klasy będą mogły wykonywać się niezależnie, „w tym samym czasie".
4, 5 – zmienne pomocnicze wykorzystywane przy realizacji zadania każdego z wątków.
6 – początek konstruktora klasy WatkiPodstawy.
8 – każdy wątek powinien mieć ustawiony priorytet. Priorytet może przyjąć wartości od **Thread.MIN_PRIORITY**... **Thread.MAX_PRIORITY** (obecnie 1-10). Priorytet jest swego rodzaju wskazówką od programisty, że oczekiwałby od systemu jak bardzo „ma dbać" o jego wykonanie. MAX_PRIORITY oznacza, że oczekiwalibyśmy, że będzie on realizowany zawsze, gdy tylko będzie to

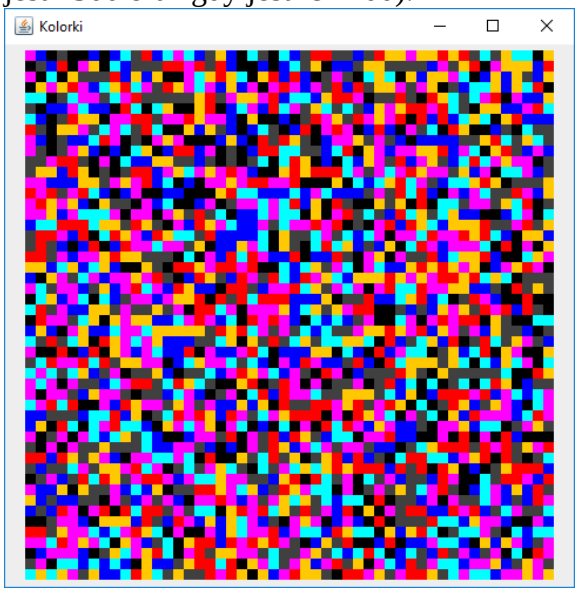
możliwe. MIN_PRIORYTY – gdy akurat nic innego nie będzie do zrobienia. Wartość priorytetu wątku została w tym przypadku przekazana jako parametr konstruktora.
9 – **start()** to metoda uruchamiająca metodę **run()**. Każda klasa dziedzicząca po Thread musi posiadać przeciążoną metodą **run()**. To właśnie w tej metodzie zawarte muszą być operacje, jakie powinny być realizowane w danym wątku.
11 – początek metody run() – ona będzie wywoływana prze metodę start().
13 – 23 – pętla nieskończona, której wykonanie zostanie przerwane po 5 wykonaniach (wiersz 21). W każdym kroku pętli wykonujemy 100000 * 2000 razy operację pom=pom*pom/pom (nie ma to specjalnego sensu ale obciążą procesor). Po wykonaniu obliczeń zwiększamy licznik pętli (**18**) i wyświetlamy na konsoli informację o wątku (19) oraz o ilości wykonanych powtórzeń pętli (**20**). Kluczowe w tym wypadku jest, że oba elementy są wykonywane w dwóch kolejnych krokach programu.
28 – tworzymy nowy obiekt klasy WatkiPodstawy – jak parametr konstruktora przekazujemy wartość priorytetu. Konstruktor oczywiście wykona start() (wiersz **9**) i wątek natychmiast rozpocznie swoją realizację.
29 – pętla w, której utworzone i uruchomione zostają kolejne 10 wątków (w tym wypadku z minimalnymi priorytetami).

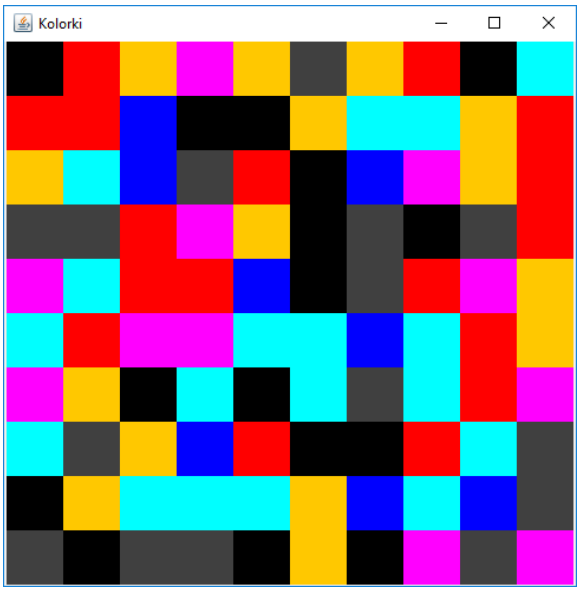
Efekt wykonania programu zaprezentowano poniżej:

```
Thread[Thread-0,10,main] - wykonanych przejsc - 1
Thread[Thread-1,1,main]Thread[Thread-10,1,main]Thread[Thread-2,1,main] - wykonanych przejsc - 1
Thread[Thread-8,1,main]Thread[Thread-9,1,main] - wykonanych przejsc - 1
Thread[Thread-7,1,main]Thread[Thread-6,1,main] - wykonanych przejsc - 1
Thread[Thread-5,1,main] - wykonanych przejsc - 1
 - wykonanych przejsc - 1
 - wykonanych przejsc - 1
 - wykonanych przejsc - 1
Thread[Thread-4,1,main] - wykonanych przejsc - 1
Thread[Thread-3,1,main] - wykonanych przejsc - 1
 - wykonanych przejsc - 1
Thread[Thread-0,10,main] - wykonanych przejsc - 2
Thread[Thread-2,1,main] - wykonanych przejsc - 2
Thread[Thread-9,1,main] - wykonanych przejsc - 2
Thread[Thread-5,1,main] - wykonanych przejsc - 2
Thread[Thread-10,1,main] - wykonanych przejsc - 2
Thread[Thread-1,1,main] - wykonanych przejsc - 2
Thread[Thread-8,1,main] - wykonanych przejsc - 2
Thread[Thread-3,1,main] - wykonanych przejsc - 2
Thread[Thread-7,1,main] - wykonanych przejsc - 2
Thread[Thread-6,1,main] - wykonanych przejsc - 2
Thread[Thread-4,1,main] - wykonanych przejsc - 2
Thread[Thread-0,10,main] - wykonanych przejsc - 3
Thread[Thread-2,1,main] - wykonanych przejsc - 3
Thread[Thread-9,1,main] - wykonanych przejsc - 3
Thread[Thread-5,1,main] - wykonanych przejsc - 3
Thread[Thread-10,1,main] - wykonanych przejsc - 3
Thread[Thread-1,1,main] - wykonanych przejsc - 3
Thread[Thread-0,10,main] - wykonanych przejsc - 4
Thread[Thread-8,1,main] - wykonanych przejsc - 3
Thread[Thread-7,1,main] - wykonanych przejsc - 3
Thread[Thread-3,1,main] - wykonanych przejsc - 3
Thread[Thread-4,1,main] - wykonanych przejsc - 3
Thread[Thread-6,1,main] - wykonanych przejsc - 3
Thread[Thread-0,10,main] - wykonanych przejsc - 5
Thread[Thread-8,1,main] - wykonanych przejsc - 4
Thread[Thread-7,1,main] - wykonanych przejsc - 4
Thread[Thread-2,1,main] - wykonanych przejsc - 4
Thread[Thread-9,1,main] - wykonanych przejsc - 4
Thread[Thread-5,1,main] - wykonanych przejsc - 4
Thread[Thread-3,1,main] - wykonanych przejsc - 4
Thread[Thread-10,1,main] - wykonanych przejsc - 4
Thread[Thread-4,1,main] - wykonanych przejsc - 4
Thread[Thread-1,1,main] - wykonanych przejsc - 4
Thread[Thread-6,1,main] - wykonanych przejsc - 4
Thread[Thread-8,1,main] - wykonanych przejsc - 5
Thread[Thread-7,1,main] - wykonanych przejsc - 5
Thread[Thread-3,1,main] - wykonanych przejsc - 5
Thread[Thread-4,1,main] - wykonanych przejsc - 5
Thread[Thread-9,1,main] - wykonanych przejsc - 5
Thread[Thread-2,1,main] - wykonanych przejsc - 5
Thread[Thread-5,1,main] - wykonanych przejsc - 5
Thread[Thread-1,1,main] - wykonanych przejsc - 5
Thread[Thread-10,1,main] - wykonanych przejsc - 5
Thread[Thread-6,1,main] - wykonanych przejsc - 5
```

Przykład 2

W przykładzie drugim zaprezentujemy sytuację, w której jednocześnie będziemy wykonywali bardzo dużo operacji graficznych. Operacje te, będą polegały na ciągłym rysowaniu kolorowych kwadracików w losowo ustalonych kolorach. Każda „animacja migającego kwadratu" realizowana jest oczywiście przez niezależny wątek. Wątków może być generalnie dowolnie wiele (poniżej zaprezentowano dwa przypadki – gdy wątków jest 2500 oraz gdy jest ich 100).






```
4   import java.util.*;
5   import java.awt.*;
6   import javax.swing.*;
7
8   class TKwadraciki extends JPanel implements Runnable{
9       private Thread watek;
10      private int pauza;
11      private static Color[] tablica_kolorow = {Color.BLACK, Color.BLUE, Color.CYAN, Color.DARK_GRAY, Color.MAGENTA, Color.ORANGE, Color.RED};
12      private static Random sierotka = new Random();
13      private static Color nowyKolor(){return tablica_kolorow[sierotka.nextInt(tablica_kolorow.length)];}
14
15      private Color BiezacyKolor = nowyKolor();
16      public void paintComponent(Graphics g)
17      {
18          super.paintComponent(g);
19          g.setColor(BiezacyKolor);
20          Dimension s = getSize();
21          g.fillRect( x: 0, y: 0, s.width,s.height);
22      }
23      public TKwadraciki(int pauza)
24      {
25          this.pauza=pauza;
26          watek = new Thread( target: this);
27          watek.setPriority(Thread.NORM_PRIORITY);
28          watek.start();
29      }
30      public void run(){
31          while(true){
32              BiezacyKolor = nowyKolor();
33              repaint();
34              try
35              {
36                  watek.sleep(pauza);
37              }
38              catch (InterruptedException e)
39              {
40                  throw new RuntimeException(e);
41              }
42
43          }
44      }
45  }
```

8 – początek klasy realizującej przykład z wątkiem rysującym. Klasa będzie rysować kwadrat na powierzchni obiektu JPanel (taki płaski obszar interfejsu graficznego), dlatego w rzeczywistości po nim

dziedziczy przeciążając jego metody rysujące. Implementuje również interfejs Runnable (to drugie podejście do programowanie wielowątkowego).
9 – obiekt klasy Thread – to on w rzeczywistości będzie realizował nasz zdanie rysunkowe.
10 – zmienna pomocnicza – będzie przechowywać informację o tym, na jak długo watek powinien zatrzymać swoje wykonaniu po narysowaniu prostokąta. Jest to konieczne, szczególnie przy tak dużej ilości wątków, aby nie doszło do zjawiska tzw. zagłodzenia wątków.
11 – tablica obiektów klasy Color – to z niej losowana będą kolory rysowanych kwadratów
12 – generator liczb losowych (tak dla żartu nazwany sierotką)
13 – metoda losująca i zwracająca losowy kolor.
15 – aktualnie wylosowany kolor
16 – metoda przeciążająca metodę paintComponent (odziedziczoną po JPanel).
18 – wywołanie konstruktora klasy JPanel (klasy pierwotnej)
19 – ustawienie bieżącego koloru w kontekście rysowania (g) na kolor taki jaki został wylosowany.
20 – obiekt **s** klasy Dimension – w nim zapamiętujemy bieżące wymiary obszaru rysowania naszego panelu (nie wiemy gdzie będzie rysowany docelowo).
21 – na kontekście rysowania **g** (czyli de facto naszym panelu) rysujemy prostokąt o wymiarach zapamiętanych w obiekcie **s** począwszy od punktu (0,0).
23 – konstruktor klasy TKwadraciki. Jako parametr otrzymuje wartość pauzy (wiersz 10).
25 – ustalamy wartość pauzy zgodnie z wartością parametru.
26 i 27 – wywołujemy konstruktor klasy Thread na potrzeby obiektu **watek**, a następnie poprzez **start()** uruchamiamy **run()**
30 – metoda **run()** w której wątek wykonując pętlę nieskończoną kolejno losuje nowy kolor a następnie wywołuje metodę **reapaint()** co skutkuje wywołaniem **paintComponent()**, a zatem odrysowaniem prostokąta. Ostatecznie wątek zatrzymuje swoją pracę na czas określony przez **pauza**.

```
47  public class Kolorki extends JFrame {
48      public Kolorki() {
49          super( title: "Kolorki");
50
51          setSize( width: 500, height: 500);
52          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
53          Container Kontener = getContentPane();
54          Kontener.setLayout(new GridLayout( rows: 10, cols: 10));
55          for (int i=0; i<100; i++)
56              Kontener.add(new TKwadraciki( pauza: 100));
57          setVisible(true);
58      }
59      public static void main(String[] args) {
60          Kolorki kolory=new Kolorki();
61      }
62  }
```

Zadanie 1 (25 pkt)
Proszę przygotować program, który będzie wykonywał 10 niezależnie działających od siebie animacje (każda identyczna)). Animacje powinny być możliwie złożone obliczeniowo – każda klatka powinna być rysowana przez możliwie złożony algorytm i powinna być estetyczna (w jednym kroku przesuwać/obracać/modyfikować powinny się setki elementów) - można np. wykorzystać do tego jakiś prosty fraktal itp. Należy umożliwić ustalenie priorytetów każdej z nich przed ich uruchomieniem. Animacja ma np. polegać na ciągłym odrysowywaniu pojedynczej klatki w innym miejscu (np. przesuwanie, obracanie).

Np. 8 klatek jednej z 10 animacji – w każdej klatce fraktal jest rysowany nieco w prawo, gdy osiągnie krawędź obszaru rysowanie następuje odwrócenie kierunku i ostatecznie powrót do pozycji początkowej. W tym momencie rysowanie rozpoczynane jest od nowa i tak bez końca.