5. Pule wątków

W aplikacjach wielowątkowych często pojawiają się problemy ze zbyt dużą ilością wątków wykonujących proste zadania. Można tworzyć grupy wątków (klasa `ThreadGroup`) lub też samodzielnie organizować algorytmy usprawniające pracę wielu wątków. Skuteczniejszym rozwiązaniem jest wykorzystanie puli wątków.

Egzekutory - wykonawcy

Pule wątków umożliwiają utworzenie kilku wątków, które będą obsługiwały znacznie większą liczbę wątków. Pozwala to na ograniczenie wykorzystywanej jednocześnie liczby wątków, a co za tym idzie zwiększenie wydajności maszyny wirtualnej. Do utworzenia puli wątków wykorzystuje się statyczne metody klasy `Executor`. Najczęściej wykorzystywane pule to:

- `newSingleThreadExecutor()` – pula z jednym wątkiem (przekazane do puli wątki wykonują się sekwencyjnie),
- `newFixedThreadPool()` – pula o określonej liczbie wątków, wątki cały czas działają,
- `newCachedThreadPool()` – pula o rozmiarze tworzonym według zapotrzebowania, wątki nieaktywne przestają istnieć po pewnym czasie,
- `newSheduledThreadPool()` – pula o określonej liczbie wątków, ich wykonanie odbywa się według zaplanowanego czasu.

Metody te zwracają obiekty implementujące interfejs `ExecutorService`. Po utworzeniu puli należy przekazać do niej za pomocą metod `execute()` lub `submit()` obiekty klasy implementującej interfejs `Runnable`. W zależności od rodzaju egzekutora przekazane wątki rozpoczną swoją pracę. Zamknięcie puli na nowe wątki odbywa się za pomocą metody `shutdown()`. Aby sprawdzić czy wszystkie wątki w puli zostały zakończone można wykorzystać metodę `isTerminated()`. Istnieje jeszcze możliwość przerwania wykonywania zadań w puli (metoda `shutdownNow()`). Poniżej przykładowa aplikacja tworząca stałą pule wątków. Najpierw kod pojedynczego wątku

```
public class Watek implements Runnable {

    private String nazwa;
    private int wynik;
    private int wartosc;

    public Watek(String nazwa, int wartosc) {
        this.nazwa = nazwa;
        this.wartosc = wartosc;
    }

    public void run() {

        System.out.println("Wątek: " + nazwa + "(" + Thread.currentThread().getName()
+ ")" + " zaczyna obliczenia...");
        for(int i=0; i<wartosc; i++){
            wynik += wartosc;
        }
        try {
            Thread.sleep((int)(Math.random()*1000));
        } catch (InterruptedException e) {
            System.out.println("Wątek został przerwany");
        }
        System.out.println("Wątek: " + nazwa + "(" + Thread.currentThread().getName()
+ ")" + " skończył obliczenia. Wynik: " + wynik);
```

```
    }
}
```

i przykładowe utworzenie puli:

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class PulaWatkow {

    public static void main(String[] args) {
        ExecutorService executor = Executors.newFixedThreadPool(5);
        for (int i = 1; i < 11; i++) {
            Runnable watek = new Watek("W" + i, i);
            executor.execute(watek);
        }
        executor.shutdown();
        while(!executor.isTerminated()) {}
        System.out.println("Wszystkie wątki zakończyły pracę.");
    }
}
```

Wynik działania aplikacji:

```
Wątek: W1(pool-1-thread-1) zaczyna obliczenia...
Wątek: W3(pool-1-thread-3) zaczyna obliczenia...
Wątek: W4(pool-1-thread-4) zaczyna obliczenia...
Wątek: W2(pool-1-thread-2) zaczyna obliczenia...
Wątek: W5(pool-1-thread-5) zaczyna obliczenia...
Wątek: W3(pool-1-thread-3) skończył obliczenia. Wynik: 9
Wątek: W6(pool-1-thread-3) zaczyna obliczenia...
Wątek: W6(pool-1-thread-3) skończył obliczenia. Wynik: 36
Wątek: W7(pool-1-thread-3) zaczyna obliczenia...
Wątek: W5(pool-1-thread-5) skończył obliczenia. Wynik: 25
Wątek: W8(pool-1-thread-5) zaczyna obliczenia...
Wątek: W4(pool-1-thread-4) skończył obliczenia. Wynik: 16
Wątek: W9(pool-1-thread-4) zaczyna obliczenia...
Wątek: W2(pool-1-thread-2) skończył obliczenia. Wynik: 4
Wątek: W10(pool-1-thread-2) zaczyna obliczenia...
Wątek: W1(pool-1-thread-1) skończył obliczenia. Wynik: 1
Wątek: W10(pool-1-thread-2) skończył obliczenia. Wynik: 100
Wątek: W8(pool-1-thread-5) skończył obliczenia. Wynik: 64
Wątek: W9(pool-1-thread-4) skończył obliczenia. Wynik: 81
Wątek: W7(pool-1-thread-3) skończył obliczenia. Wynik: 49
Wszystkie wątki zakończyły pracę.
```

Łatwo można zauważyć, że aplikacja wykorzystuje tylko pięć wątków jednocześnie.

Interfejs Callable

Standardowe wątki tworzone za pomocą klasy `Thread` czy też interfejsu `Runnable`, mają spore ograniczenia. Jednym z nich jest brak możliwości sprawdzenia wyniku pracy wątku. Metoda `run()` nie zwraca żadnej wartości (typ `void`). Wykorzystując pule wątków oraz interfejs `Callable` można w prosty sposób uzyskać dane zwracane przez wątki. Tworzenie wątków na bazie interfejsu `Callable` sprowadza się do przesłonięcia metody `call()`. Metoda ta, wykorzystując typy generyczne, umożliwia zwrócenie dowolnego typu danych. Aby wykorzystać jej potencjał musimy utworzone wątki dodać do puli wątków za pomocą metody `submit()`. Metoda `submit()` tym różni się od metody `execute()` iż zwraca wyniki (obiekty implementujące interfejs `Future`). Na rzecz tych obiektów możemy wywołać metodę `get()` i uzyskać dostęp do zwróconych danych. Poniżej prosty przykład aplikacji tworzącej pule wątków oraz wykorzystującej interfejs `Callable`. Najpierw wątek, jego zadaniem jest obliczenie sumy liczb z podanego zakresu pewnej tablicy.

```
import java.util.concurrent.Callable;

public class Zadanie implements Callable<Integer>{

    private int poczatek;
    private int koniec;
    private int[] tablica;

    public Zadanie(int[] tablica, int poczatek, int koniec){
        this.poczatek = poczatek;
        this.koniec = koniec;
        this.tablica = tablica;
    }

    public Integer call() throws Exception {
        int wynik = 0;

        for(int i = poczatek ; i < koniec; i++){
            wynik += tablica[i];
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                System.out.println("Błąd");
            }
 }

        System.out.println("Liczyłem sumę od zakresu: " + poczatek + " do zakresu: " +
koniec + " Wynik: " + wynik);

         return wynik;
    }
}
```

Główna aplikacja:

- tworzy tysiąc elementową tablicę pseudolosowych wartości całkowitych z zakresu od 0 do 10,
- tworzy jednowątkową pule wątków,
- zbiera wyniki z wątków,
- wyświetla dane na ekranie.

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class TestZadania {

    public static void main(String[] args) {

        int suma = 0;
        int[] tablicaLiczb = new int[1000];

        for (int i = 0; i < tablicaLiczb.length; i++) {
            tablicaLiczb[i] = (int)(Math.random()*11);
        }

        ExecutorService executor =  Executors.newSingleThreadExecutor();

        Zadanie[] zadania = new Zadanie[10];

        int podzial = tablicaLiczb.length / zadania.length;
        for (int i = 0; i < zadania.length; i++) {
            zadania[i] = new Zadanie(tablicaLiczb, (i * podzial), ((i+1) * podzial));
            Future<Integer> future = executor.submit(zadania[i]);
            try {
                suma += future.get();
            } catch ( InterruptedException | ExecutionException ex) {
```

```
                System.out.println("Błąd");
            }
        }

        executor.shutdown();

        while(!executor.isTerminated()) {}
        System.out.println("Wszystkie wątki zakończyły pracę.");

        System.out.println(suma);
    }
}
```

Przykładowy wynik działania aplikacji:

```
Liczyłem sumę od zakresu: 0 do zakresu: 100 Wynik: 481
Liczyłem sumę od zakresu: 100 do zakresu: 200 Wynik: 560
Liczyłem sumę od zakresu: 200 do zakresu: 300 Wynik: 458
Liczyłem sumę od zakresu: 300 do zakresu: 400 Wynik: 462
Liczyłem sumę od zakresu: 400 do zakresu: 500 Wynik: 496
Liczyłem sumę od zakresu: 500 do zakresu: 600 Wynik: 505
Liczyłem sumę od zakresu: 600 do zakresu: 700 Wynik: 453
Liczyłem sumę od zakresu: 700 do zakresu: 800 Wynik: 529
Liczyłem sumę od zakresu: 800 do zakresu: 900 Wynik: 540
Liczyłem sumę od zakresu: 900 do zakresu: 1000 Wynik: 508
Wszystkie wątki zakończyły pracę.
4992
```

Zadania

1. Wykorzystując pule wątków napisz aplikację, która będzie sprawdzać czy są dostępne połączenia z różnymi hostami. Dla hostów utwórz prostą tablicę typu `String`, np.

   ```
   String[] hosty = {"onet.pl", "gazeta.pl" ... "pudelek.pl", "facebook.com"};
   ```

   Do sprawdzenia czy host jest osiągalny czy też nie posłuż się poniższym kodem:

   ```
   Socket socket = null;
   boolean reachable = false;
   try {
       socket = new Socket(hostName, 80);
        reachable = true;
   } catch (UnknownHostException e) {
   } catch (IOException e) {
   } finally {
      if (socket != null)
          try {
             socket.close();
           } catch(IOException e) {}
   }
   ```

   Zastanów się z jakiej puli wątków skorzystać oraz jak dobrać optymalną liczbę wątków zaangażowanych w proces sprawdzania. Sprawdź poprawność działania aplikacji dodając hosty, co do których masz pewność, że nie działają (np. dfre.pl).

2. Napisz aplikację Restauracja.

   - restauracja ma określoną liczbę miejsc (wykorzystaj pulę wątków),
   - utwórz klasę odpowiadającą za klienta restauracji (wątek), każdy klient ma mieć swój numer np. K1, K2 itd.
   - utwórz cztery razy więcej wątków klienta niż miejsc w restauracji,

- zasymuluj w wątku klienta standardowe operacje (wchodzi i siada, wybiera z menu, zamawia, konsumuje, płaci i wychodzi), każda z tych operacji ma zajmować odpowiednio dopasowany przedział czasowy,
- sprawdź działanie aplikacji.

3. Sprawdź jeszcze raz działanie aplikacji pobierającej od wątków wynik ich działania. Czy aplikacja w pełni wykorzystuje wielowątkowość? Przyspiesz działanie aplikacji tworząc pule wątków o stałym rozmiarze. W jaki sposób zapisywać wyniki zwracane przez poszczególne wątki? Wykorzystując klasę `Stoper`, dobierz optymalny rozmiar puli wątków.
4. Napisz aplikację wykorzystującą pule wątków do sprawdzenia czy dana liczba jest liczbą pierwszą. Pobierz od użytkownika liczbę, utwórz określoną liczbę wątków, przekaż im dane do obliczeń, odbierz wyniki i wyświetl informację czy dana liczba jest czy nie jest liczbą pierwszą. Do sprawdzenia liczby wykorzystaj algorytm naiwny.