Programowanie wielowątkowe

Programowanie wielowątkowe

- Wątek (ang. thread) – część programu wykonywana współbieżnie w obrębie jednego procesu
- Różnica między zwykłym procesem a wątkiem polega na współdzieleniu przez wszystkie wątki działające w danym procesie przestrzeni adresowej oraz wszystkich innych struktur systemowych (np. listy otwartych plików, gniazd itp.) – z kolei procesy posiadają niezależne zasoby.

Programowanie wielowątkowe

- Wątki wymagają mniej zasobów do działania od procesów i też mniejszy jest czas ich tworzenia.
- Dzięki współdzieleniu przestrzeni adresowej (pamięci) wątki jednego zadania mogą się między sobą komunikować w bardzo łatwy sposób, niewymagający pomocy ze strony systemu operacyjnego. Przekazanie dowolnie dużej ilości danych wymaga przesłania jedynie wskaźnika, zaś odczyt (a niekiedy zapis) danych o rozmiarze nie większym od słowa maszynowego nie wymaga synchronizacji (procesor gwarantuje atomowość takiej operacji).

Wątki - synchronizacja

- Jeśli dwa lub więcej wątków zapisują i odczytują tą samą zmienną, to możemy mieć kłopot:
  - Wyobraźmy sobie sytuację, że przetwarzamy w oddzielnych wątkach wszystkie pliki tekstowe z danego katalogu. Jeśli jeden z wątków zacznie swoje obliczenia na pliku1 i NIE ustawi gdzieś informacji, że plik ten jest już przetwarzany, to drugi z wątków może wykonać tą samą pracę jeszcze raz.

Tworzymy wątki

- W zasadzie, to wystarczą dwie rzeczy:

1. Implementacja interfejsu Runnable
2. Implementacja metody run()
3. Rozpoczęcie wątku.

Tworzymy wątki – interfejs Runnable

```java
/**
 *
 * @author tjach
 */
public class Watek implements Runnable {
    String name;

    Watek(String _name) {
        this.name = _name;
        System.out.println("Jestem sobie nowy wesoly watek " + name);
    }
    @Override
    public void run() {
        System.out.println(name + " rozpoczyna prace");

    }

}
```

Tworzymy wątki – korzystamy z wątków

```
11  * @author tjach
12  */
13 public class ThreadTest {
14     private static final int ILE = 10;
15
16     /**
17      * @param args the command line arguments
18      */
19     public static void main(String[] args) {
20         Watek[] watekPool = new Watek[ILE];
21
22         for(int i=0;i<ILE;i++) {
23             watekPool[i] = new Watek("Numer " + i);
24             watekPool[i].run();
25         }
26     }
27
28 }
```

ThreadTest > main > watekPool

Tworzymy wątki – subklasa Thread

```
11  * @author tjach
12  */
13 public class Watek extends Thread {
14     String name;
15
16     Watek(String _name) {
17         this.name = _name;
18         System.out.println("Jestem sobie nowy wesoly watek " + name);
19     }
20     @Override
   public void run() {
22         try{
23             System.out.println(name + " rozpoczyna prace");
24             Thread.sleep(100);
25         }
26         catch(InterruptedException e) {
27             System.out.println("Przerwano prace watku " + name);
28         }
29
```

Przerywanie pracy wątków

- Czasami chcemy przerwać pracę któregoś z wątków i potem do niej wrócić.
- Przykładowo w trakcie długich obliczeń czasami dajemy szansę użytkownikowi na porzucenie długotrwałego czekania (bo się rozmyślił)
- Aby to osiągnąć możemy przerwać wątek, ale tylko w trakcie jego „spania”

Przerywamy wątek

```
10  *
11  * @author tjach
12  */
13 public class ThreadTest {
14     private static final int ILE = 10;
15
16     /**
17      * @param args the command line arguments
18      */
19     public static void main(String[] args) {
20         Watek[] watekPool = new Watek[ILE];
21
22         for(int i=0;i<ILE;i++) {
23             watekPool[i] = new Watek("Numer " + i);
24             watekPool[i].start();
25             if(i%5==0) watekPool[i].interrupt();
26         }
27     }
28
```

ThreadTest > main > for (int i = 0; i < ILE; i++) > if (i % 5 == 0) >

Usages | Output - ThreadTest (run)

```
run:
Jestem sobie nowy wesoly watek Numer 0
Jestem sobie nowy wesoly watek Numer 1
Numer 0 rozpoczyna prace
Jestem sobie nowy wesoly watek Numer 2
Przerwano prace watku Numer 0
Numer 1 rozpoczyna prace
Jestem sobie nowy wesoly watek Numer 3
Numer 2 rozpoczyna prace
Jestem sobie nowy wesoly watek Numer 4
Numer 3 rozpoczyna prace
Jestem sobie nowy wesoly watek Numer 5
Numer 4 rozpoczyna prace
```

Komunikacja między wątkami

- Za pomocą metody Thread.join() możemy zmusić wątek do oczekiwania na zakończenie pracy innego wątku.
- Każda metoda opatrzona słówkiem synchronized ma zagwarantowane, że dokładnie jeden wątek może ją wykonywać w jednym czasie.

Problemy z synchronizacją

```
11      * @author tjach
12      */
13     public class Counter {
14         private int c;
15         Counter() {
16             c=0;
17         }
18         public void add() {
19             c++;
20         }
21         public void rem() {
22             c--;
23         }
24         public int ile() {
25             return c;
26         }
27
28     }
```

Zdefiniujmy prostą klasę, która przechowuje jedną zmienną c.
Trzy metody inkrementują, dekrementują i zwracają wartość przechowywanego licznika (pole c).

Problemy z synchronizacją

```
18      */
        public static void main(String[] args) throws InterruptedException {
20          Watek[] watekPool = new Watek[ILE];
21          Counter licznik = new Counter();
22          for(int i=0;i<ILE;i++) {
23              watekPool[i] = new Watek("Numer " + i, licznik);
24              watekPool[i].start();
25              if(i%2==0)
26                  watekPool[i].l.add();
27              else
28                  watekPool[i].l.rem();
29              System.out.println("Jest teraz: " + watekPool[i].l.ile());
30          }
31      }
32
33  }
34
```

ThreadTest

Usages | Output - ThreadTest (run)

```
run:
Jestem sobie nowy wesoly watek Numer 0
Numer 0 rozpoczyna prace
Jest teraz: 1
Jestem sobie nowy wesoly watek Numer 1
Jest teraz: 0
Jestem sobie nowy wesoly watek Numer 2
Numer 1 rozpoczyna prace
Numer 2 rozpoczyna prace
Jest teraz: 1
Jestem sobie nowy wesoly watek Numer 3
Jest teraz: 0
Jestem sobie nowy wesoly watek Numer 4
```

Raz dodaję, raz usuwam.

Zwróć uwagę na kolejność komunikatów poniżej – są całkowicie wymieszane.

Wniosek: wykonują się w losowej kolejności

Ćwiczenia

1. Napisz prostą „zip-bombę”. Stwórz 10 wątków, z których każdy tworzy plik binarny z wartościami losowymi o długości ~1MB. Nie twórz więcej wątków, bo zabijesz komputer.
   Co by było gdyby każdy z tych wątków rekurencyjnie wywoływał 10 kolejnych? Jeśli chcesz to sprawdzić, to użyj wirtualnej maszyny!
2. Napisz program liczący wartość silni iteracyjnie i rekurencyjnie. Obie z tych metod włącz w oddzielnych wątkach i policz jak długo zajęły obliczenia. Wykorzystaj do tego proste GUI i zaimplementuj przycisk „Nie chcę dłużej czekać”