Spis treści

Temat: Wątki
Czym są wątki. Grafika. Proste animacje. Małe podsumowanie materiału.

Podstawa: Arnold, Gosling; Boone; Eckel.

1 Wątki

Zwykłe operacje w programach komputerowych są wykonywane sekwencyjnie, tzn. kolejno, jedna po drugiej. Są to tzw. programy jednowątkowe. Java^{TM} stwarza możliwość wykonywania programu wielowątkowo. Wątki mogą przy tym korzystać i modyfikować te same zasoby danych. Wymaga to koordynacji przebiegu poszczególnych wątków. Wątki nie mogą przeszkadzać sobie nawzajem i dostęp do zasobów przez poszczególne wątki nie powinien odbywać się chaotycznie. Mechanizmy obsługi wątków są w Java^{TM} wysoko wyspecjalizowane. Są to blakady zasobów, odpytywanie itp.

2 Tworzenie wątków

Wątki w Java^{TM} udostępnia klasa `Thread`. Wątek tworzymy poleceniem

```
Thread przebieg = new Thread();
```

Następnym krokiem jest konfiguracja utworzonego wątku. Można go nazwać, nadać mu priorytet początkowy i następnie wykonać uruchamiając metodę `start()` wątku. Tak uruchomiony wątek wykonuje następnie metodę `run()`. Po wykonaniu tej metody wątek ginie. Wątek można też zatrzymać uruchamiając

metodę `stop()` wątku. Metoda `sleep(milis)` usypia wątek na `milis` milisekund.

Poniższy przykład pochodzi z podręcznika Arnolda i Goslinga, Java^{TM}.

```
//
// K. Arnold, J. Gosling, Java
// PingPong.class;
// Wątki
//
class PingPong extends Thread {
    String word; // co wypisać
    int delay;

    PingPong(String whatToSay, int delayTime) {
        word  = whatToSay;
        delay = delayTime;
    }
    public void run() {
        try {
            for (;;) {
                System.out.print(word + " ");
                sleep(delay);    // czekaj na nastepna kolejke
            }
        } catch (InterruptedException e) {
            return;              //  zakoncz ten watek
        }
    }
    public static void main(String[] args) {
        new PingPong("ping", 33).start();    // 1/30 sec
        new PingPong("PONG",100).start();    // 1/100 sec
    }
}
```

Klasa `PingPong` jest potomkiem klasy `Thread`. Konstruktor obiektów tej klasy ustawia dwie zmienne. Jedna z nich to `word`, zawierająca dowolny napis (tutaj `ping` lub `PONG`), a druga jest czasem który wątek *przesypia*, a więc jest nieaktywny. Metoda `main` deklaruje dwa wątki i uruchamia je. Jeden z nich (`ping`) *budzi się* co 33/1000 ms, a drugi (`PONG`) co 1/10 ms. Pilnuje tego metoda `run()` wątku, która zatrzymuje go na ten czas wywołując metodę `sleep(delay)` z parametrem `delay`, określającym liczbę przesypianych milisekund.

Wynik programu będzie podobny do następujacego.

```
ping PONG ping ping ping PONG ping ping ping PONG ping ping
ping PONG ping ping ping PONG ping ping ping PONG ping ping
ping PONG ping ping ping PONG ping ping ping PONG ping ping
ping PONG ping ping ping PONG ping ping ping PONG ping ping
PONG ping ping ping PONG ping ping ping PONG ping ping ping
PONG ping ping ping PONG ping ping ping PONG ping ping ping
```

```
PONG ping ping ping PONG ping ping ping PONG ping ping ping
PONG ping ping ping PONG ping ping ping PONG ping ping ping
PONG ping ping PONG ping ping ping PONG ping ping PONG ping
ping PONG ping ping PONG ping ping PONG ping ping ping PONG
ping ping ping PONG ping ping ping PONG ping ping ping PONG
ping ping ping PONG ping ping ping PONG ping ping ping PONG
```

Widzimy, że napisy `ping` i `PONG` różnią się częstością występowania.

Metoda `setName` nadaje nazwę wątkowi, a `getName` odczytuje ją. Nazwą można też nadać przekazując konstruktorowi wątku obiekt klasy `String`.

3 Synchronizacja

Metody Java^{TM} mogą być synchronizowane (`synchronized`). Jeśli jakiś wątek wykonuje taką metodę dla danego obiektu, to wchodzi jednocześnie do tzw. monitora obiektu i wszystkie inne wywołania metod synchronizowanych dla tego obiektu muszą czekać, aż wątek opuści monitor. Dzieje się to wtedy gdy metoda synchronizowana zakończy się. Synchronizacja wymusza wzajemne wykluczanie wykonywania sie wątków `w tym samym czasie`.

P r z y k ł a d. 1.
Poniższa metoda zamienia wszystkie elementy tablicy na dodatnie, zastępując wszystkie ujemne elementy ich wartością bezwzględną. Fragment tej metody musimy synchronizować po to, by żaden inny wątek oprócz wątku aktualnie wywołującego metode nie mógł przypadkowo zmienić już zmienionych danych.

```
// zamień wszystkie elementy tablicy na nieujemne
// Arnold, Gosling
public static void abs(int[] values) {
    synchronixed (values) {
        for(int i = 0; i < values.length; i++) {
            if(values[i] < 0)
                values[i] = -values[i];
        }
    }
}
```

Operacje na elementach tablicy są synchronizowane. Dopiero po wykonaniu całej pętli obiekt zajęty przez wątek zostaje zwolniony. Mamy pewność, że cała tablica została przebadana i żaden inny przypadkowy proces nie zmienił jej zawartości.

Dwie metody `wait` i `notify`, zdefiniowane w klasie `Object`, pozwalaja na dodatkową komunikację między wątkami. Metoda `wait` każe wątkowi czekać, aż jakieś wydarzenie zostanie zrealizowane, a metoda `notify` informuje o tym fakcie.

Schemat, wg. którego stosuje się metodę `wait` pokazany jest niżej.

```
synchronized void wykonajWarunkowo() {
    while( !warunek)
        wait();
```

```
        ... kod do wykonania po spełnieniu warunku
}
```

Nie będziemy tu omawiać problemów tzw. zakleszczania wątków, licząc na to, że nasze programy nie będą zbyt złożone.

4 Dodatki

Wątki można zawieszac (`suspend`). Jest to przydatne gdy chcemy by działały one w pewnych tylko chwilach, a więc wtedy, gdy potrzeba.

Elegancki sposób zatrzymywania wątku polega nie na jego przerwaniu metodą `stop`, lecz inaczej. Zazwyczaj w metodzie `run` sprawdza się jakąś zmienna logiczną, której wartość może się zmieniać w zależności od warunków.

5 Algorytmy sortowania

Klasycznym już przykładem ilustrującym trzy algorytmy sortowania, zrealizowanym w Java^{TM} pokazana w ilustracjach dystrybucji Java SDK 2. (patrz `jdk1.3//demo//applets//SortDemo`).

6 Klasa `Runnable`

Oprócz klas, Java^{TM} dostarcza tzw. interfejsów (sprzęg). Interfejs stanowi typ składający sie tylko z metod abstrakcyjnych oraz stałych. *Interfejs jest przykładem czystej formy projektu, klasy zaś są mieszanką projektu i implementacji* (Arnold, Gosling). Klasy mogą implementować metody interfejsu tak, jak chce programista. W ten sposób interfejs ma dużo więcej możliwych implementacji niż klasa. Klasa może implementowac wiele interfejsów naraz.

Jednym z interfejsów Java^{TM} jest `Runnable`. `Runnable` deklaruje jedną metodę:

```
public void run();
```

Oto jak wyglada program PingPong (teraz RunPingPong), gdzie zastosowano interfejs `Runnable`.

```
//
// K. Arnold, J. Gosling, Java
// PingPong.class;
// Wątki
//
class RunPingPong implements Runnable {
    String word; // co wypisać
    int delay;                  // ile czekać

    RunPingPong(String whatToSay, int delayTime) {
        word  = whatToSay;
        delay = delayTime;
```

```
    }

    public void run() {
        try {
            for (;;) {
                System.out.print(word + " ");
                sleep(delay);    // czekaj na nastepna kolejke
            }
        } catch (InterruptedException e) {
            return;              //  zakoncz ten watek
        }
    }
    public static void main(String[] args) {
        Runnable ping = new RunPingPong("ping", 33);
        Runnable pong = new RunPingPong("PONG",100);
        new Thread(ping).start();
        new Thread(pong).start();
    }
}
```

Tworzone obiekty (`ping` i `pong`) klasy `Thread` są uruchamiane poprzez wywołanie metody `start()`. Wynik działania programu jest podobny do poprzedniego.