Kurs programowania

Wykład 8

Wojciech Macyna

21 kwiecień 2016

Współbieżność

Wątki w JAVA-ie

Współbieżność może być realizowana na poziomie systemu operacyjnego (procesy) lub na poziomie aplikacji (wątki). W JAVA-ie powszechnie stosuje się wątki. Na wątkach zbudowana jest większość aplikacji w tym GUI.

Definiowanie zadania

Wątek wykonuje pewne zadanie - aby je zdefiniować trzeba zaimplementować interfejs `Runnable` z metodą `run()` oraz skojarzyć to zadanie z określonym wątkiem.

Przykład

RunnableDemo.java

```
1  class DisplayMessage implements Runnable {
2    private String message;
3    public DisplayMessage(String message) {
4      this.message = message; }
5    public void run() {
6      while(true) System.out.println(message); }
7  }
8  public class RunnableDemo {
9    public static void main(String [] args) {
10     System.out.println("Creating␣the␣hello␣thread...");
11     DisplayMessage hello = new DisplayMessage("Hello");
12     Thread thread1 = new Thread(hello);
13     System.out.println("Starting␣the␣hello␣thread...");
14     thread1.start();
15     System.out.println("Creating␣the␣goodbye␣thread...");
16     DisplayMessage bye = new DisplayMessage("Goodbye");
17     Thread thread2 = new Thread(bye);
18     System.out.println("Starting␣the␣goodbye␣thread...");
19     thread2.start();
20   }
21 }
```

Klasa `Thread`

Klasa `Thread` jest klasą bazową wątków. Jej konstruktor jako argument przyjmuje klasę z zaimplementowanym interfejsem `Runnable` a uruchamia wątek metodą `start()` która wywołuje metodę `run()`. Wątki mogą też być definiowane jako pochodne klasy `Thread`.

Usypianie wątku – `Thread.sleep(int ms)`

Zawieszenie wykonywania wątku. Metoda wywołuje wyjątek `InterruptedException` i musi być umieszczona w środowisku `try`.

Próba przełączenia wątku – `Thread.yield()`

Sugerowanie zarządcy wątków, że w tym momencie można przełączyć sterowanie na inny wątek (o tym samym priorytecie).

Klasa `Thread`

Priorytet wątku – `getPriority()` i `setPriority(int)`

Wątki mogą mieć ustawione priorytety pomiędzy `Thread.MIN_PRIORITY` a `Thread.MAX_PRIORITY`. Wątki o wyższym priorytecie są wykonywane w pierwszej kolejności co może spowodować, ze wątki o niższym priorytecie nie będą wykonywane. Wątki są startowane z domyślnym priorytetem: `Thread.NORM_PRIORITY`.

Wątki-demony – `setDaemon()`

Demon to wątek który działa w tle programu. Program nie czeka np. z zakończeniem działania na wątki które są demonami. Ustawienie wątku jako demona musi się odbyć przed jego uruchomieniem.

Klasa `Thread`

Łączenie wątków – `join()`

Czeka aż zakończy się wykonywanie wątku. Dopiero potem rozpoczynają swoje działanie następne wątki.

Nadawanie nazwy – `setName(string name)`

Nadanie nazwy która jest między innymi wyświetlana przez metodę `toString()` obiektu.

Przykład - ThreadDemo.java

```
1  class GuessANumber extends Thread {
2    private int number;
3    public GuessANumber(int number) { this.number = number; }
4    public void run() {
5      int counter = 0;
6      int guess = 0;
7      do {
8        Thread.yield();
9        guess = (int) (Math.random() * 1000000 + 1);
10       counter++; }
11     while(guess != number);
12     System.out.println("** Correct! "+this.getName()+
13                        " in "+counter+" guesses.**");
14   }
15 }
16 public class ThreadDemo {
17   public static void main(String [] args) {
18     GuessANumber player1 = new GuessANumber(20);
19     GuessANumber player2 = new GuessANumber(20);
20     GuessANumber player3 = new GuessANumber(20);
21     player1.start(); player2.start(); player3.start();
22   }
23 }
```

Współdzielenie zasobów

Wątki z punktu widzenia programu są wykonywane niezależnie i nie wiadomo w jakiej kolejności zostaną wywołane. Stąd przy korzystaniu ze wspólnych zasobów musimy zapewnić im pewną synchronizację i zapobiec kolizjom. JAVA ma wbudowany mechanizm zapobiegający takim sytuacjom.

Słowo kluczowe `synchronized`

Za pomocą tego słowa oznaczamy metodę która nie powinna być przerwana ze względu na możliwość kolizji. Metoda ta zawsze będzie wykonywana w całości bez względu na możliwość przejścia na inny wątek.

Środowisko `synchronized(object){...}`

Powiązanie nieprzerywalnego ciągu komend z obiektem który jest traktowany jako semafor.

Współdziałanie wątków

Metody `wait()` i `notifyAll()`

Metoda `sleep()` nie zwalnia blokad. Metoda `wait(int ms)` zawiesza wykonywanie wątku na `ms` milisekund i zwalnia blokady. Jej działanie kończy się w wyniku upłynięcia czasu, albo wywołania metod `notify()` lub `notifyAll()`. Dla `wait()` bez argumentów nie ma warunku czasowego.

Zakleszczenie

Niewłaściwe stosowanie metody `wait()` przez różne wątki może doprowadzić do zakleszczenia, czyli blokady.

Przykład

`SynchronizedDemo1.java` – brak synchronizacji

`SynchronizedDemo2.java` – zastosowanie metody `synchronized`

`SynchronizedDemo3.java` – zastosowanie metody `synchronized` oraz `wait()` i `notifyAll()`

`SynchronizedDemo4.java` – zastosowanie metody `synchronized` oraz `wait()` i `notify()`

`SynchronizedDemo5.java` – zastosowanie środowiska `synchronized`

`SynchronizedDemo6.java` – przykład zakleszczenia