Wprowadzenie do programowania współbieżnego

Motywacja: Współbieżność pozwala istotnie poprawić działanie programu w przypadku gdy komputer z jednej strony musi wykonywać dość żmudne obliczenia a z drugiej nadzorować stan interfejsu użytkownika. Sytuacja tak często ma miejsce w programach symulacyjnych (oraz grach komputerowych).

Każdy uruchamiany program tworzy **proces** w systemie operacyjnym. Dla każdego procesu alokowane są wymagane przez niego zasoby: wirtualna konsola, zasoby pamięciowe i plikowe.

Wątek - jednostka wykonawcza procesu. Jest to sekwencja działań, zapisanych w programie, która może wykonywać się (praktycznie) równolegle z innymi sekwencjami (innych) działań w kontekście danego procesu (programu).

Każdy proces ma co najmniej jeden wykonujący się wątek, ale może mieć ich wiele.

Równoległość działań w ramach procesu osiągamy przez uruchomienie wielu (różnych) wątków.

Uruchamianie nowego wątku

Uruchamianiem i zarządzaniem wątkiem zajmuje się klasa ***Thread.*** Aby utworzyć wątek, należy utworzyć obiekt klasy *Thread*.
Uruchomienie wątku osiągamy przez zastosowanie metody *start()* wobec obiektu klasy *Thread.*

Wątek jest obiektem klasy implementującej interfejs ***Runnable*** **.** Implementacja tego interfejsu wymaga zdefniowania metody *run().* W metodzie *run* podajemy działania, które ma wykonywać dany wątek.

Klasa *Thread* implementuje interfejs *Runnable* (podając pustą metodę *run*).

Dwa najczęściej używane sposoby tworzenia i uruchamiania wątku:

Pierwszy sposób
1. zdefiniować własną klasę dziedziczącą Thread.
2. przedefiniować odziedziczoną metodę run().
3. utworzyć obiekt utworzonej tak klasy.
4. wysłać mu komunikat start().

Drugi sposób
1. zdefiniować klasę implementującą interfejs Runnable (class X implements Runnable).
2. zdefiniować metodę run.
3. utworzyć obiekt utworzonej tak klasy (X x = new X()).
4. utworzyć obiekt typu Thread, przekazując w konstruktorze referencję do obiektu klasy implementującej Runnable (Thread watek = new Thread(x)).

5. Wywołać na rzecz tego obiektu metodę start (watek.start()).

Zmodyfikuj program tak aby wątek tworzony był poprzez implementację interfejsu Runnable:

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
class Timer extends Thread{
    Label tl;
    Timer (Label l){tl=l;}
    public void run(){
       int time=0;
       while (time<10){
           try{Thread.sleep(1000);} catch (InterruptedException exc){return;}
           time++;
           tl.setText(String.valueOf(time));
       }
    }
}
public class Watek_applet extends Applet implements ActionListener{
  Button start = new Button("Start");
  Label l = new Label("0000");
  Timer tm;
  public void init(){
    add(l);
    add(start);
    start.addActionListener(this);
  }
  public void actionPerformed(ActionEvent e){
      tm = new Timer(l);
      tm.start();
  }
}
```

Zamień przycisk Start/Stop na Ressume/Susspend. Na starcie uruchamia się wątek odliczający. Wciśnięcie przycisku wstrzymuje wątek. Ponowne wciśnięcie przywraca odliczanie. Zastosuj konstrukcję wait-notify.

Zmodyfikuj przycisk na Start/Stop. Ponowne wciśnięcie kończy wątek (wyjście z metody run). Można to zrealizować np. za pomocą flagi ustawianej w actionPerformed i sprawdzanej w run.

Dodaj przycisk Suspend/Resume. Wykorzystaj konstrukcję wait/notify do wstrzymywania wątku.

Producent-Konsument

```
//Błędna implementacja problemu producenta i konsumenta.
class Q {
int n;

synchronized int get() {
 System.out.println("Pobrano: " + n);
 return n;
}

synchronized void put(int n) {
```

```
 this.n = n;
 System.out.println("Włożono: " + n);
}
}

class Producer implements Runnable {
Q q;

Producer(Q q) {
 this.q = q;
 new Thread(this, "Producent").start();
}

public void run() {
 int i = 0;

 while(true) {
   q.put(i++);
 }
}
}

class Consumer implements Runnable {
Q q;

Consumer(Q q) {
 this.q = q;
 new Thread(this, "Konsument").start();
}

public void run() {
 while(true) {
   q.get();
 }
}
}

class PC_wrong {
public static void main(String args[]) {
 Q q = new Q();
 new Producer(q);
 new Consumer(q);
}
}
```

Utwórz poprawną wersję programu Producent-Konsument z użyciem konstrukcji wait/notify.

Uzupełnij definicję klasy NewThread (konstruktor + toString):

```
class NewThread implements Runnable {
  String name;
  public void run() {
    try {
      for(int i = 5; i > 0; i--) {
        System.out.println(name + ": " + i);
        Thread.sleep(100);
      }
```

```
    } catch (InterruptedException e) {
      System.out.println("Przerwano " + name);
    }
    System.out.println("Zakończenie " + name);
  }
}

class ThreadTest {
  public static void main(String args[]) {
    new NewThread("Jeden");
    Thread w = Thread.currentThread();
    w.setName("watek_glowny");
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      System.out.println("Przerwano wątek główny");
    }

    System.out.println("Koniec: "+w);
  }
}
```

Wydruk Finalny:

```
Nowy wątek: mój_wątek:Jeden
Jeden: 5
Jeden: 4
Jeden: 3
Jeden: 2
Jeden: 1
Zakończenie Jeden
Koniec: Thread[watek_glowny,5,main]
```

W klasie ThreadTest utwórz trzy wątki: „Jeden”, „Dwa” i „Trzy”.

Uruchom te wątki z użyciem metody join().

W metodzie main utwórz wątek, który sprawdza w metodzie run czy wartość i jest parzysta a jeżeli nie to drukuje jej wartość i się zatrzymuje.

```
public class AlwaysEven {
  static int i;
  static void next() { i++; i++; }
  public static void main(String[] args) {

// TODO

    while(true)
      next();
  }
}
```

Wskazówka:

```
new Thread("Watcher") {
      public void run() {
        }
    }.start();
```

Przykładowe rozwiązanie:

```
public static void main(String[] args) {
    new Thread("Watcher") {
      public void run() {
        while(true) {
          if(i % 2 != 0) {
            System.out.println(i);
            System.exit(0);
          }
        }
      }
    }.start();
    while(true)
      next();
  }
```

Uzupełnij kod. Wątek klasy testsynchro uruchamia się po czym czeka na powiadomienie o ukończeniu działania metody valchange.

```
public class testsynchro extends Thread{

String st="Hello";

public static void main(String args[]) {

       testsynchro czekaj=new testsynchro();

czekaj.start();

new Example1(czekaj);

}

public void run(){

try {

synchronized(this){

wait();

System.out.println("testsynchro :"+st);

}

}catch(Exception e){}

}

public void valchange(String st){
```

```
// TODO

}

}

class Example1 extends Thread{

      testsynchro wait;

// TODO

public void run(){

System.out.println("Example1: "+wait.st);

wait.valchange("Hello World");

}

}
```

Tak aby wydruk był postaci:

```
Example1: Hello
testsynchro :Hello World
```

Zakleszczenie

Dwa lub więcej wątków zakładając rygle (synchronized) na obiektach mogą to zrobić w sposób prowadzący do ich wzajemnego zablokowania:

```
public class TestThread {
   public static Object Lock1 = new Object();
   public static Object Lock2 = new Object();

   public static void main(String args[]) {

      ThreadDemo1 T1 = new ThreadDemo1();
      ThreadDemo2 T2 = new ThreadDemo2();
      T1.start();
      T2.start();
   }

   private static class ThreadDemo1 extends Thread {
      public void run() {
         synchronized (Lock1) {
            System.out.println("Thread 1: Holding lock 1...");
            try { Thread.sleep(10); }
            catch (InterruptedException e) {}
            System.out.println("Thread 1: Waiting for lock 2...");
            synchronized (Lock2) {
               System.out.println("Thread 1: Holding lock 1 & 2...");
            }
         }
      }
```

```
   }
   private static class ThreadDemo2 extends Thread {
      public void run() {
         synchronized (Lock2) {
            System.out.println("Thread 2: Holding lock 2...");
            try { Thread.sleep(10); }
            catch (InterruptedException e) {}
            System.out.println("Thread 2: Waiting for lock 1...");
            synchronized (Lock1) {
               System.out.println("Thread 2: Holding lock 1 & 2...");
            }
         }
      }
   }
}
```

Konsola:

```
Thread 1: Holding lock 1...
Thread 2: Holding lock 2...
Thread 2: Waiting for lock 1...
Thread 1: Waiting for lock 2...
```

Zmień kolejność synchronizacji tak aby wątki nie zakleszczały się:

```
Thread 1: Holding lock 1...
Thread 1: Waiting for lock 2...
Thread 1: Holding lock 1 & 2...
Thread 2: Holding lock 1...
Thread 2: Waiting for lock 2...
Thread 2: Holding lock 1 & 2...
```