Wykład 1

Współbieżność

https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

Struktura wykładu

1. Wątki – wprowadzenie
2. Tworzenie wątków – klasa Thread oraz interfejs Runnable
3. Synchronizacja wątków (blokowanie dostępu wątków do ważnych fragmentów programu) – słowo kluczowe synchronized
4. Koordynacja działań wątków – metody wait, notify, notifyAll

1. Wątki – wprowadzenie

- **Wątkiem** nazywamy sekwencyjny przepływ sterowania w procesie, który wykonuje dany program np. odczytywanie i zapisywanie plików
- Program Javy jest wykonywany w **obrębie jednego wątku**
- Pamięć programu jest wspólna dla wątków potomnych, każdy z nich ma własny stos
- Zablokowanie jednego z wątków nie zawiesza działania innych wątków
- Wznowienie działania wątku następuje od miejsca zawieszenia
- System Windows **przydziela cyklicznie czas procesora wątkom** w kolejności wynikającej z ich priorytetów (taka sytuacja występuje np. w komputerach jednoprocesorowych)
- Wiele działających wątków powoduje, **że program działa asynchronicznie** (nie ma pojedynczej pętli zdarzeń)

- Synchronizacja wykonywanych metod tego danego obiektu przez wątki jest możliwa dzięki monitorowi, który posiada każdy z obiektów. Dodanie słowa **synchronized** do oznaczenia nazwy metody tego obiektu powoduje, że jeśli jeden z wątków wykonuje taką metodę, to żaden z pozostałych wątków nie może wykonać tej i innej synchronizowanej metody tego obiektu.

- Wątki mogą się porozumiewać za pomocą metod ***wait()***, ***notify()*** **i** ***notifyAll()*** wspólnie używanego obiektu – wątek może wykonywać synchronizowaną metodę tego obiektu i po wykonaniu w niej metody *wait()* zostaje uśpiony do chwili, kiedy inny wątek obudzi go wykonując metodę *notify()* wywołanej w metodzie synchronizowanej tego samego obiektu. Jeśli więcej wątków zostało uśpionych po wykonaniu metody *wait()* tego samego obiektu, inny obiekt wywołując metodę *notifyAll()* może je obudzić w kolejności wynikającej z priorytetów tych wątków.

2. Tworzenie wątków – klasa Thread oraz interfejs Runnable

1. Wątki są reprezentowane przez obiekty typu ***Thread***. Klasa ***Thread*** zawiera metody niezbędne do zarządzania wątkami. Obiekt typu ***Thread*** jest obiektem sterującym wątkiem. Klasa ***Thread*** ma metody, które uruchamiają, „usypiają”, zawieszają oraz zatrzymują wątek
2. Tworzenie wątku polega na tworzeniu nowego egzemplarza klasy ***Thread***. Obiekt typu ***Thread*** uruchamia obiekt wątkowy implementujący **interfejs *Runnable***, który definiuje metodę ***run()***:
   2.1. Obiektem wątkowym może być obiekt przekazywany do konstruktora obiektu ***Thread***

```
public Thread(Runnable target) {
        init(null, target, "Thread-" + nextThreadNum(), 0);
}
```

   2.2. Obiektem wątkowym może być instancja klasy ***Thread***

```
public Thread(){
        init(null, null, "Thread-" + nextThreadNum(), 0);
}
```

   2.3. Obiektem wątkowym może być instancja następcy klasy ***Thread***

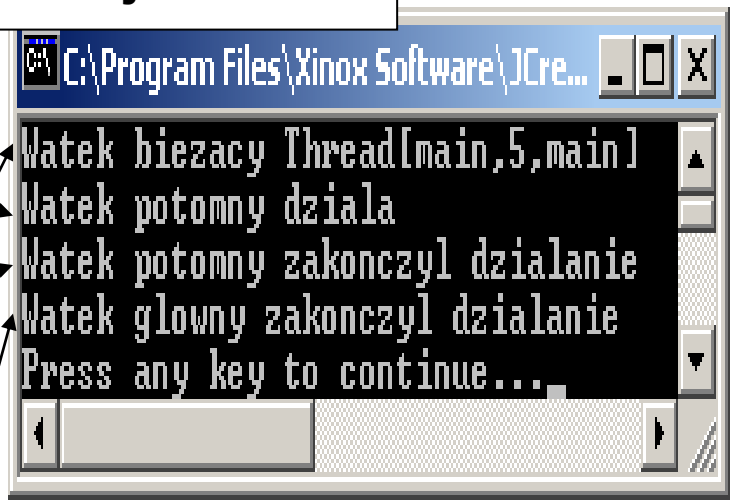
```
public NowyThread extends Thread

public NowyThread() {
        init(null, null, "Thread-" + nextThreadNum(), 0) ;
}
```

3. Po uruchomieniu metody start obiektu klasy ***Thread*** lub jej następcy uruchamiana jest metoda ***run()*** obiektu wątkowego

Przykład 1 (p. 2.1)

Obiekt wątkowy typu Wątek1 implementuje interfejs Runnable

```
class Watek1 implements Runnable
{  public void run()
   { try
      { System.out.println("Watek potomny dziala");
        Thread.sleep(4000);
      } catch(InterruptedException e)
        { System.out.println("Przerwanie watku potomnego");}
     System.out.println("Watek potomny zakonczyl dzialanie");
   } }
public class p6_1
{   void demo()
     { Thread watekglowny = Thread.currentThread();
       System.out.println("Watek biezacy " + watekglowny);
       Watek1 obiekt = new Watek1();
       Thread watekpotomny = new Thread(obiekt, "Watek potomny");
       watekpotomny.start();
       try
        {  Thread.sleep(6000);
        }  catch(InterruptedException e)
           { System.out.println("Przerwanie watku glownego");}
       System.out.println("Watek glowny zakonczyl dzialanie");
     }
    public static void main (String argd[])
   { p6_1 p = new p6_1 ();
     p.demo();   } }
```



Wyjaśnienia:

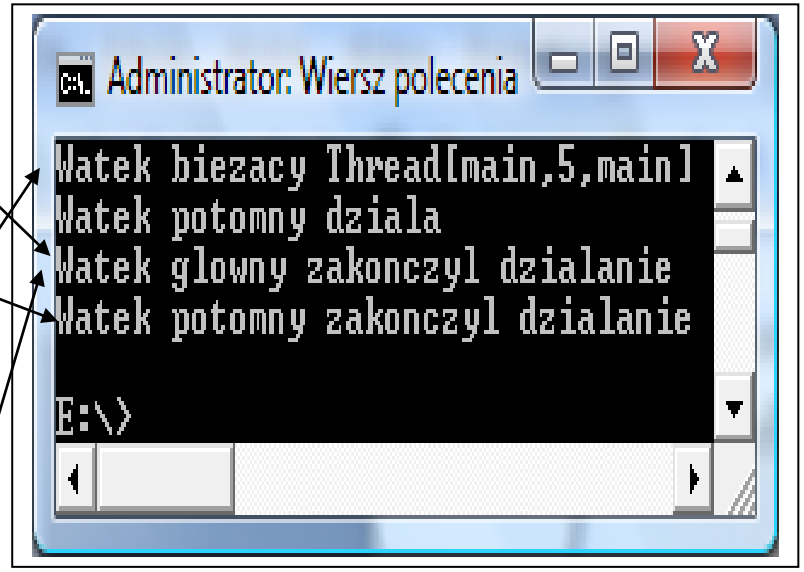
Metoda Thread.sleep powoduje, że bieżący wątek:

- zawiesza swoje działanie na czas określony za pomocą wartości przekazanego parametru. Wartość 4000 oznacza w przybliżeniu 4 sek.
- udostępnia czas procesora innym wątkom aplikacji lub innym aplikacjom, które mogą działać w tym czasie.
- W podanych przykładach musi być uruchomiony w bloku **try...catch** do obsługi wyjątków **InterruptedException** typu **checked exception**, czyli **obowiązkowo sprawdzane błędy**

Przykład 2 – (p.2.3)

Obiekt wątkowy typu Wątek1 dziedziczy po klasie ***Thread*** – przesłania metodę **run**, którą dziedziczy po klasie ***Thread***. Metoda ***run*** w klasie ***Thread*** jest wynikiem implementowania interfejsu ***Runnable*** przez klasę typu ***Thread***

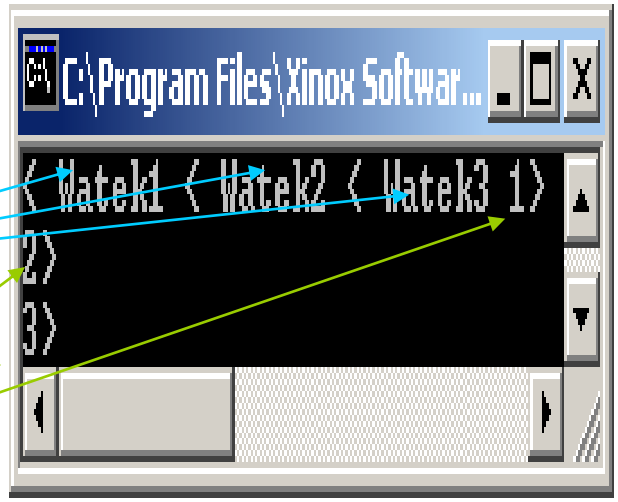
```
class Watek1 extends Thread
{  public void run()
   { try
      { System.out.println("Watek potomny dziala");
        Thread.sleep(400);
      } catch(InterruptedException e)
        { System.out.println("Przerwanie watku potomnego");}
     System.out.println("Watek potomny zakonczyl dzialanie");
   }
 }
public class p6_1
 {  void demo()
     { Thread watekglowny = Thread.currentThread();
       System.out.println("Watek biezacy " + watekglowny);
       Watek1 watekpotomny = new Watek1();
       watekpotomny.start();
       try
        {  Thread.sleep(600);
        }  catch(InterruptedException e)
          { System.out.println("Przerwanie watku glownego");}
       System.out.println("Watek glowny zakonczyl dzialanie");
     }
    public static void main (String argd[])
   { p6_1 p = new p6_1 ();
     p.demo();   } }
```



3. Synchronizacja wątków (blokowanie dostępu wątków do ważnych fragmentów programu) – słowo kluczowe synchronized

1) **Przykład 3** – brak synchronizacji wywołania metody *wyswietl*

```
import java.util.*;
import java.io.*;
import java.lang.*;

class Zasob
{ void wyswietl(String m, int n)
  { System.out.print("< "+m);
    try
     { Thread.sleep(4000);
     } catch(Exception e)
       {  }
    System.out.println(n+">");
  }
}
class Watek2 implements Runnable
{ String s;
  Zasob z;
  int num;
  public Watek2(Zasob zasob, String lan, int n)
  { s=lan;    z= zasob;    num=n; }

  public void run()
  { z.wyswietl(s, num); }
}
```



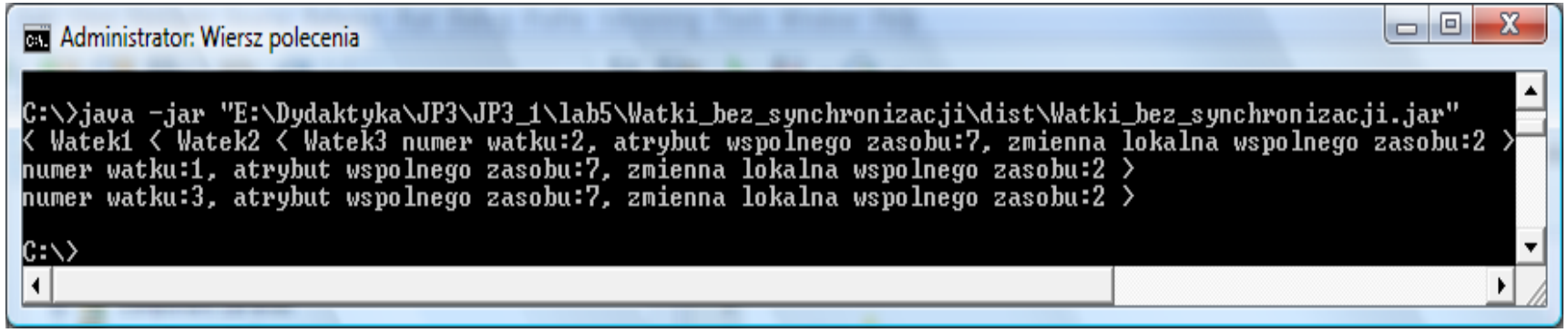
```
public class p6_2
 {
  void demo()
   {
    Zasob zasob = new Zasob();
    Watek2 obiekt1 = new Watek2(zasob, "Watek1 ",1);
    Thread watek1 = new Thread(obiekt1);
    watek1.start();
    Watek2 obiekt2 = new Watek2(zasob, "Watek2 ",2);
    Thread watek2 = new Thread(obiekt2);
    watek2.start();
    Watek2 obiekt3 = new Watek2(zasob, "Watek3 ",3);
    Thread watek3 = new Thread(obiekt3);
    watek3.start();
   }
   public static void main (String argd[])
    {
      p6_2 p = new p6_2 ();
      p.demo();
    }
  }
```

2) Przykład 4– synchronizacja wywołania metody *wyswietl*

```
import java.util.*;
import java.io.*;
import java.lang.*;

class Zasob1
{ synchronized void wyswietl(String m, int n)
  { System.out.print("< "+m);
    try
     { Thread.sleep(4000);
     } catch(Exception e)
       {}
    System.out.println(n+">");     }
}
class Watek2 implements Runnable
{ String s;
  Zasob1 z;
  int num;

 public Watek2(Zasob1 zasob, String lan, int n)
  {s = lan;
   z = zasob;
   num = n; }

 public void run()
  {  z.wyswietl(s,num); }
}
```

```
public class p6_3
{
 void demo()
  {
    Zasob1 zasob = new Zasob1();
    Watek2 obiekt1 = new Watek2(zasob, "Watek1 ",1);
    Thread watek1 = new Thread(obiekt1);
    watek1.start();
    Watek2 obiekt2 = new Watek2(zasob, "Watek2 ",2);
    Thread watek2 = new Thread(obiekt2);
    watek2.start();
    Watek2 obiekt3 = new Watek2(zasob, "Watek3 ",3);
    Thread watek3 = new Thread(obiekt3);
    watek3.start();
  }

  public static void main (String argd[])
   { p6_3 p = new p6_3 ();
     p.demo();
   }
}
```

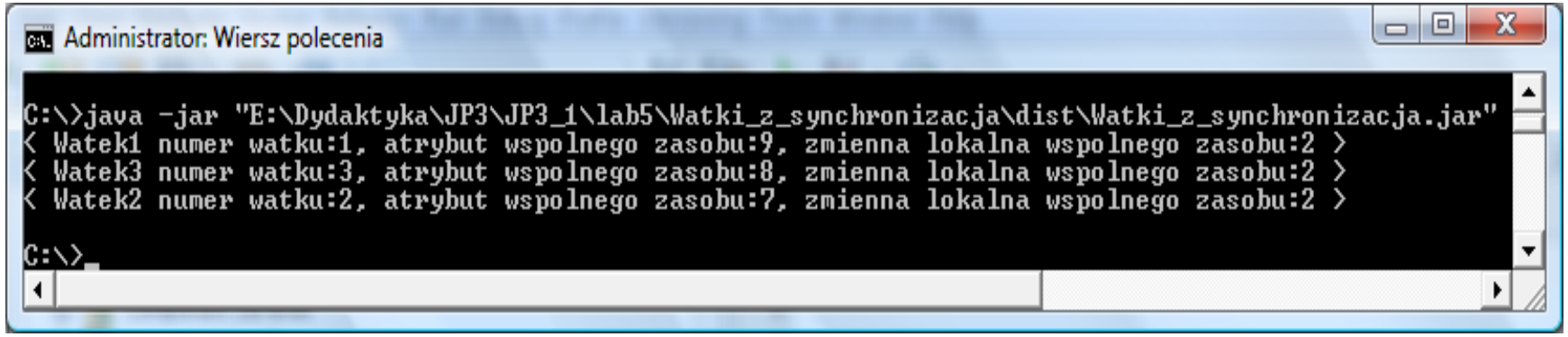
Przykład 5 - brak synchronizacji wywołania metody *wyswietl.* Każdy wątek, który jako pierwszy kończy metodę *wyswietl* (w przykładzie wątek 2) oraz pozostałe wątki, używają atrybutu wspólnego zasobu zmienionego **przez ostatni wątek, który rozpoczął** wykonywanie tej metody (w przykładzie wątek 3). Każdy wątek posiada własną zmienną lokalną – wywołuje własny egzemplarz metody i posiada własny stos.




```
class Zasob
{ int a=10;                                  //atrybut wspólnego zasobu
  void  wyswietl(String m, int n)
  { int b=3;                                 //zmienna lokalna wspólnego zasobu
    b--;     a--;
    System.out.print("< "+m);
    try
     { Thread.sleep(4000);
     } catch(Exception e)
        {    }
    System.out.println(("numer watku:"+n+", atrybut wspolnego zasobu:" +a+
                        ", zmienna lokalna wspolnego zasobu:"+b)+" >");   } }
```

```
class Watek2 implements Runnable
{ String s;  Zasob z;  int num;
  public Watek2(Zasob zasob, String lan, int n)
   {   s = lan;    z= zasob;    num=n; }

   public void run()
     {  z.wyswietl(s,num);   }
}

public class Watki_bez_synchronizacji
 { void demo()
   { Zasob zasob = new Zasob();
     Watek2 obiekt1 = new Watek2(zasob, "Watek1 ",1);
     Thread watek1 = new Thread(obiekt1);
     watek1.start();
     Watek2 obiekt2 = new Watek2(zasob, "Watek2 ",2);
     Thread watek2 = new Thread(obiekt2);
     watek2.start();
     Watek2 obiekt3 = new Watek2(zasob, "Watek3 ",3);
     Thread watek3 = new Thread(obiekt3);
     watek3.start();
   }

   public static void main (String argd[])
    { Watki_bez_synchronizacji p = new Watki_bez_synchronizacji ();
      p.demo();     } }
```

Przykład 6 - synchronizacja wywołania metody *wyświetl.* Każdy wątek, który wykonuje metodę *wyswietl* (w przykładzie wątek 2), używa atrybutu wspólnego zasobu zmienionego **przez ostatni wątek, który przed nim zakończył** wykonywanie tej metody. Każdy wątek posiada własną zmienną lokalną – wywołuje własny egzemplarz metody i posiada własny stos.



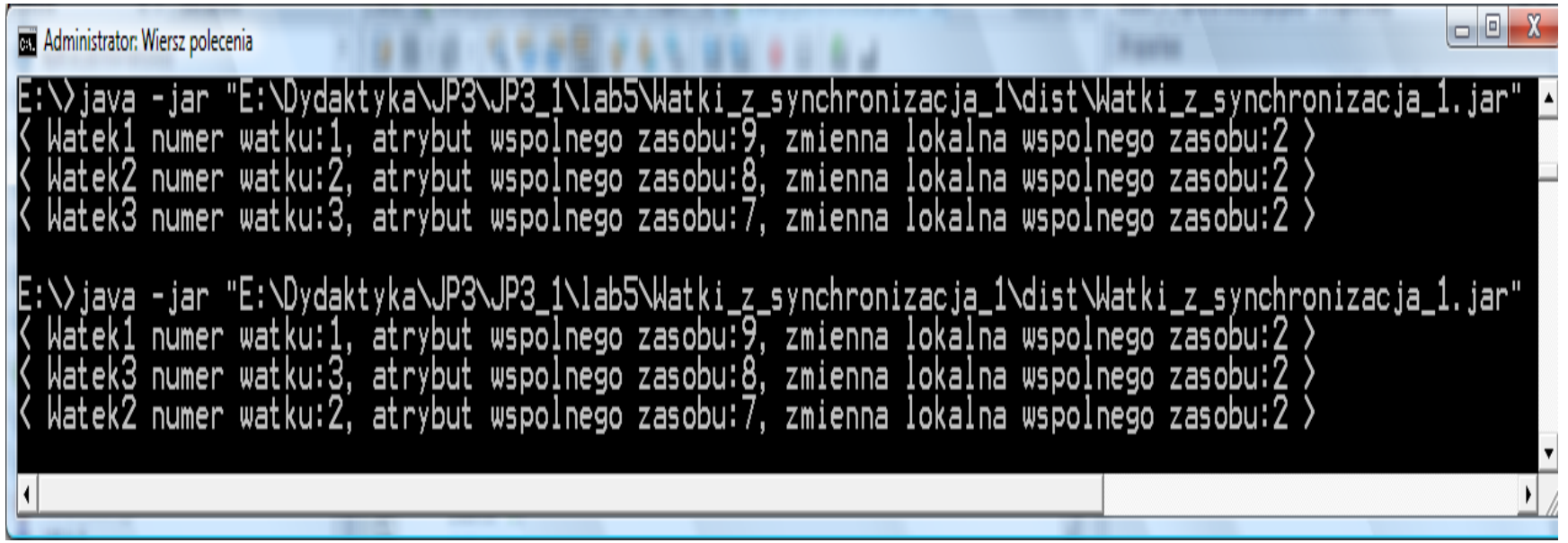

```
class Zasob
{ int a=10;                                  //atrybut wspólnego zasobu
  synchronized void  wyswietl(String m, int n)
   { int b=3;                                //zmienna lokalna wspólnego zasobu
     b--;     a--;
     System.out.print("< "+m);
     try
      { Thread.sleep(4000);
      } catch(Exception e)
        {    }
     System.out.println(("numer watku:"+n+", atrybut wspolnego zasobu:" +a+
                         ", zmienna lokalna wspolnego zasobu:"+b)+" >");   } }
```

```
class Watek2 implements Runnable
{ String s;  Zasob z;  int num;
  public Watek2(Zasob zasob, String lan, int n)
   {   s = lan;    z= zasob;    num=n; }

   public void run()
     {  z.wyswietl(s,num);   }
}

public class Watki_z_synchronizacja
 { void demo()
   { Zasob zasob = new Zasob();
     Watek2 obiekt1 = new Watek2(zasob, "Watek1 ",1);
     Thread watek1 = new Thread(obiekt1);
     watek1.start();
     Watek2 obiekt2 = new Watek2(zasob, "Watek2 ",2);
     Thread watek2 = new Thread(obiekt2);
     watek2.start();
     Watek2 obiekt3 = new Watek2(zasob, "Watek3 ",3);
     Thread watek3 = new Thread(obiekt3);
     watek3.start();
   }

   public static void main (String argd[])
    { Watki_z_synchronizacja p = new Watki_z_synchronizacja ();
      p.demo();    } }
```

Przykład 7 - synchronizacja wywołania metody *wyświetl.* Każdy wątek, który wykonuje metodę *wyswietl* (w przykładzie wątek 2) w metodzie run w bloku synchronizowanym

synchronized (obiekt) instrukcja

używa atrybutu wspólnego zasobu zmienionego **przez ostatni wątek, który przed nim zakończył** wykonywanie tej metody. Każdy wątek posiada własną zmienną lokalną – wywołuje własny egzemplarz metody i posiada własny stos.



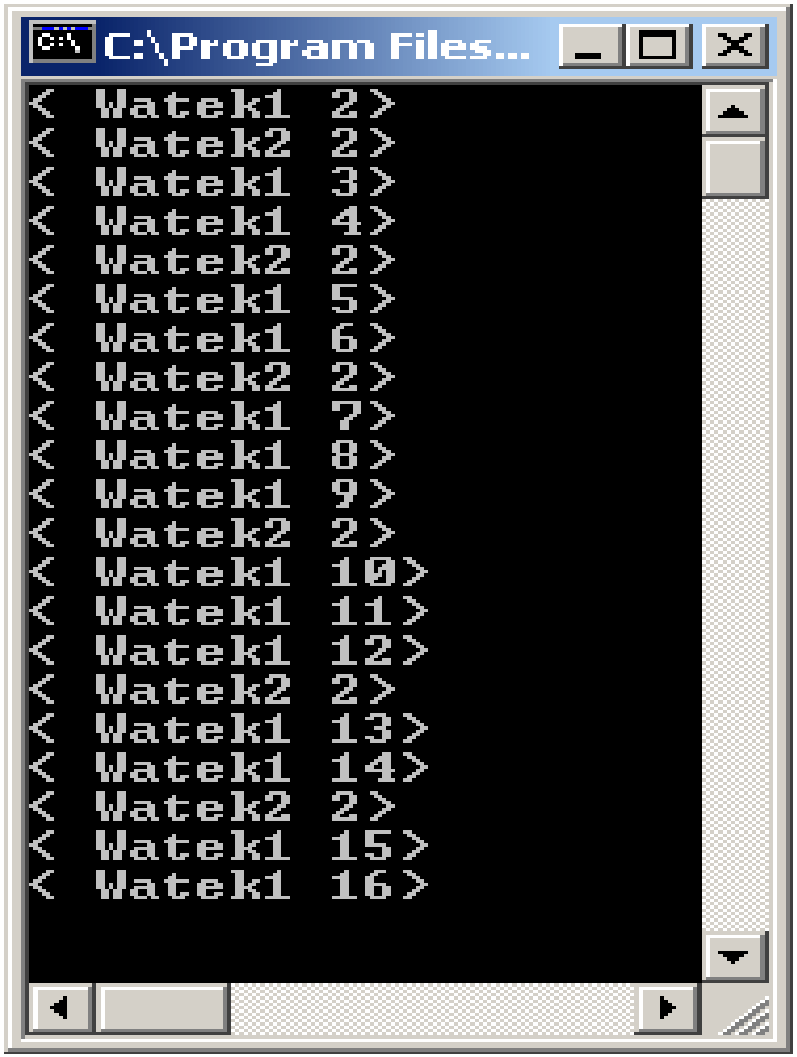
```
class Zasob
{ int a=10;                                  //atrybut wspólnego zasobu
  void  wyswietl(String m, int n)
   { int b=3;                                //zmienna lokalna wspólnego zasobu
     b--;
     a--;
     System.out.print("< "+m);
     try
      { Thread.sleep(4000);
      } catch(Exception e)       {    }
     System.out.println(("numer watku:"+n+", atrybut wspolnego zasobu:" +a+
                         ", zmienna lokalna wspolnego zasobu:"+b)+" >");
   }
}
class Watek2 implements Runnable
{ String s;  Zasob z;  int num;
  public Watek2(Zasob zasob, String lan, int n)
   {   s = lan;    z= zasob;    num=n; }

   public void run()
   {  synchronized(z) {
          z.wyswietl(s,num);      }
   }
}
```

```
public class Watki_z_synchronizacja
 { void demo()
   { Zasob zasob = new Zasob();
     Watek2 obiekt1 = new Watek2(zasob, "Watek1 ",1);
     Thread watek1 = new Thread(obiekt1);
     watek1.start();
     Watek2 obiekt2 = new Watek2(zasob, "Watek2 ",2);
     Thread watek2 = new Thread(obiekt2);
     watek2.start();
     Watek2 obiekt3 = new Watek2(zasob, "Watek3 ",3);
     Thread watek3 = new Thread(obiekt3);
     watek3.start();
   }

   public static void main (String argd[])
    { Watki_z_synchronizacja  p = new Watki_z_synchronizacja ();
      p.demo();     }  }
```

4. Koordynacja działań wątków – metody wait, notify, notifyAll

1) Przykład 7 - brak zachowania kolejności wykonania metody *wyswietl* przez 2 wątki

```
class Zasob2
{
  int num;
  synchronized void wyswietl(String m, int n)
   {
     System.out.print("< "+m);
     num = n;
     System.out.println(num+">");
   }
}
```



```
class Watek1 implements Runnable
{   String s;
    Zasob2 z;
    int num;
public Watek1(Zasob2 zasob, String lan,
                    int n)
  {  s = lan;    z = zasob;    num = n;   }

public void run()
  { while (true)
    { z.wyswietl(s, num++);
      try
       {
         Thread.sleep(400);
       }
       catch(InterruptedException e)
         {       }
     }
  } /*koniec run*/
} /*koniec Watek1*/
```

```
class Watek2 extends Watek1
{
public Watek2(Zasob2 zasob, String lan,
                    int n)
    {   super(zasob, lan, n);  }

public void run()
   { while (true)
     {  z.wyswietl(s,num);
        try
        {
            Thread.sleep(1000);
        }
       catch(InterruptedException e)
         {     }
       }
   } /*koniec run*/
} /*koniec Watek2*/
```

```
public class p6_4
 {
  void demo()
   {
    Zasob2 zasob = new Zasob2();
    Watek1 obiekt1 = new Watek1(zasob, "Watek1 ", 1);
            Thread watek1 = new Thread(obiekt1);
              watek1.start();
    Watek2 obiekt2 = new Watek2(zasob, "Watek2 ", 2);
            Thread watek2 = new Thread(obiekt2);
             watek2.start();
   }


 public static void main (String argd[])
   {
     p6_4 p = new p6_4 ();
     p.demo();
    }
}
```

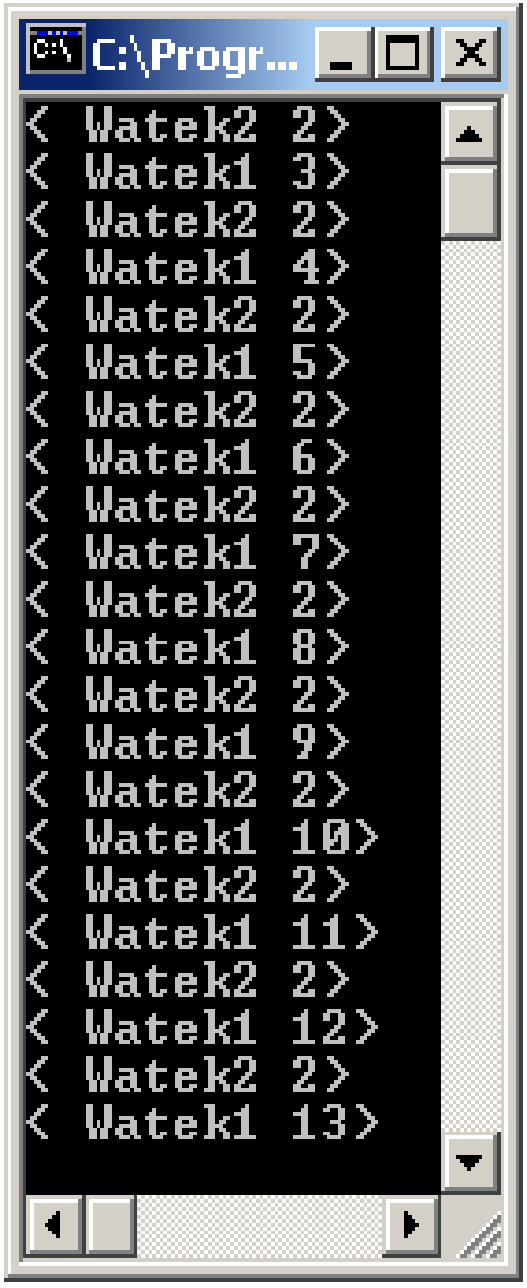
2) Przykład 8 - zachowanie kolejności wykonania metody *wyswietl* przez 2 wątki

- Ten sam obiekt, używany przez dwa wątki, posiada tzw. monitor. **Wywołanie metody wait() tego obiektu w metodzie synchronizowanej tego obiektu przez wątek spowoduje usunięcie tego wątku z monitora obiektu (uśpienie wątku)** i zawieszenie wykonania metody synchronizowanej. **Jeśli inny wątek, znajdujący się monitorze tego obiektu wywoła metodę *notify()* tego obiektu (w metodzie synchronizowanej tego obiektu), spowoduje powrót do tego monitora wątku znajdującego się poza monitorem (obudzenie wątku), który najwcześniej został uśpiony metodą wait** i kontynuowanie wykonania metody synchronizowanej obiektu. W przypadku wywołania metody ***notifyAll()*** do monitora wracają uśpione wątki – pierwszy, który został wprowadzony z monitora, wraca do niego najwcześniej.

- W przykładzie dwie metody *wyswietlx* wspólnego obiektu typu ***Zasob3*** (z monitorem) wzajemnie się informują, czy wątek ostatnio ją wywoływał – metoda *wyswietl1* należy do ***watek1***, a metoda *wyswietl2* należy do ***watek2*****.** Wywołanie metody *wait()* w tych metodach powoduje automatyczne wstrzymanie jej wykonania przez bieżący wątek (testowanie składowej *num*):
  - **prawdziwy warunek num!=2** oznacza, że **wątek1 oparty na obiekcie typu *Watek1*** ostatnio wykonał metodę *wyswietl1* lub wywołał metodę *wyswietl1* po raz pierwszy i obecnie przejdzie w stan *uśpienia*
  - **prawdziwy warunek *num==2*** oznacza, że **wątek2 oparty na obiekcie typu *Watek2*** jako ostatni wykonał metodę *wyswietl2* i obecnie przejdzie w stan *uśpienia*
- ***watek2* wykonuje metodę *wyswietl2,* gdy *num!=2*** (czyli po ***watek1*** lub po raz pierwszy w programie) i ustawia w zmiennej *num* wartość 2 i wywołuje metodę *notify* (*notifyAll()*), **która wznawia wykonanie metody *wyswietl1* dla *watek1.***
- lub ***watek1* wykonuje metodę *wyswietl1*, gdy num==2** (czyli po ***watek2***) i ustawia w zmiennej *num* wartość bieżąca >2 i wywołuje metodę *notify* (*notifyAll()*), **która wznawia wykonanie metody *wyswietl2* dla watek2.**

Przykład 8

```
class Zasob3
{ int num=3;

 synchronized void wyswietl1(String m, int n)
   { if (num!=2)
     try
      {/*System.out.println("wait1:" +num+" "+n); */ wait(); }
     catch(InterruptedException e) { }
    System.out.print("< "+m);
    num = n;
    System.out.println(num+">");
    /*System.out.println("notify1:" +num+" "+n); */ notify();
   }

synchronized void wyswietl2( String m,int n )
  { if (num == 2)
     try
      {/*System.out.println("wait2:" +num+" "+n); */ wait(); }
     catch(InterruptedException e) { }
    System.out.print("< "+m);
    num = n;
    System.out.println(num+">");
    /*System.out.println("notify2:" +num+" "+n); */  notify();
  }
}
```

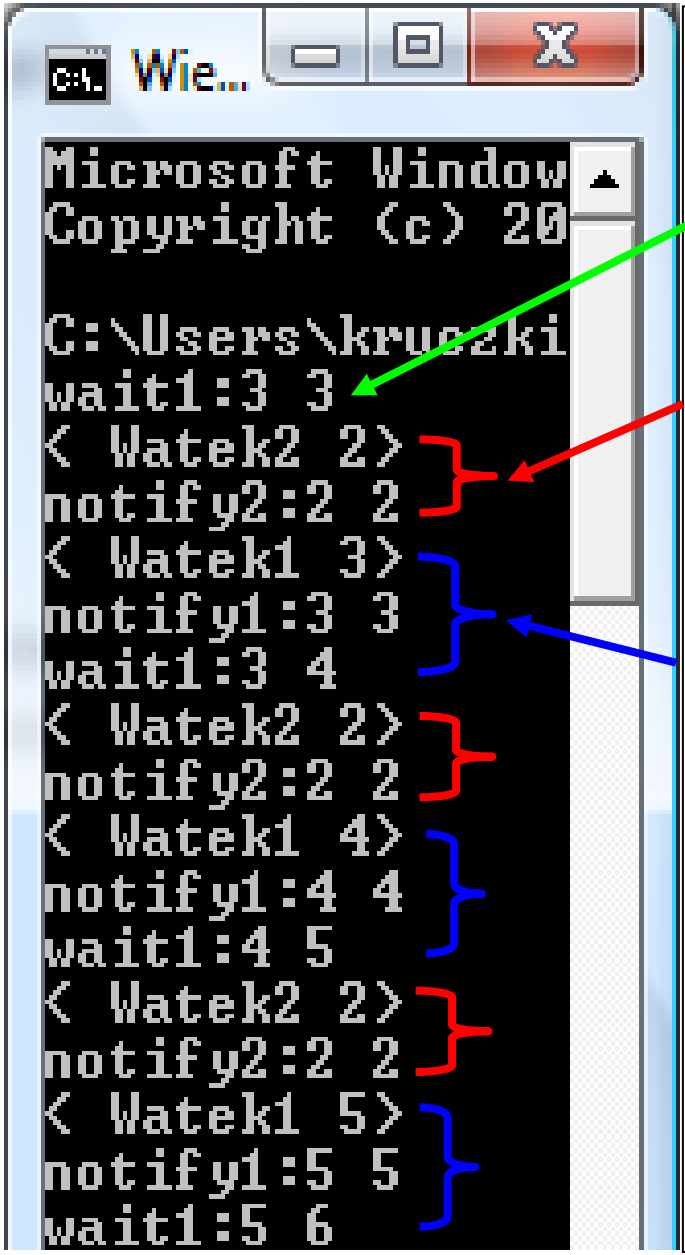


```
class Watek1 implements Runnable
{ String s;
  Zasob3 z;
  int num;
  public Watek1(Zasob3 zasob, String lan,
                        int n)
   { s = lan;    z = zasob;    num = n; }


 public void run()
  {
    while (true)
    {  z.wyswietl1(s, num++);
      try
       { Thread.sleep(400);  /*wylaczenie
opoznienia w drugim eksperymencie*/
       } catch(Exception e) {}
    } }
}
```

```
class Watek2 extends Watek1
{

public Watek2(Zasob3 zasob, String lan,
                        int n)
  { super(zasob, lan, n);}


public void run()
  {
    while (true)
     {  z.wyswietl2(s, num);
       try
         { Thread.sleep(1000); /*wylaczenie
opoznienia w drugim eksperymencie*/
         } catch(Exception e) {}
    } }
}
```

```
public class p6_5
 {
  void demo()
   { Zasob3 zasob = new Zasob3();
     Watek1 obiekt1 = new Watek1(zasob, "Watek1 ", 3);
     Thread watek1 = new Thread(obiekt1);
      watek1.start();
     Watek2 obiekt2 = new Watek2(zasob, "Watek2 ", 2);
     Thread watek2 = new Thread(obiekt2);
      watek2.start();
    }
 public static void main (String argd[])
    { p6_5 p = new p6_5 ();
      p.demo(); }
 }
```

Podsumowanie: wywołanie metody *wait* przez dany obiekt bieżącego wątku powoduje automatyczne przerwanie wykonywania tego wątku (uśpienie tego wątku) i również automatyczne wznowienie go (obudzenie) po wywołaniu metody *notify* (lub *notifyAll*) przez inną synchronizowaną metodę tego samego obiektu wykonaną przez inny wątek. W przypadku wielu przerwanych wątków uruchamiany jest ten, który pierwszy został przerwanym (*notify*); jeśli wywołano *notifyAll* wątki są wznawiane według ich priorytetów lub kolejności ich usypiania.



Eksperyment 1

Działanie metod **run**, które po zakończeniu metod **wyswietlx wchodzą** w stany opóźnień Thread.sleep(1000) oraz Thread.sleep(400).

Faza początkowa

Wątek typu ***Watek1*** jako pierwszy wywołał metodę ***wyswietl1*** i wszedł w stan wait (prawdziwy warunek **num!=2**) po wywołaniu metody ***wait***.

Faza 1

Wątek typu ***Watek2*** wykonuje metodę ***wyswietl2*** nie wchodząc w stan wait (fałszywy warunek **num==2**), nadając wartość zmiennej ***num*** wartość 2 i wywołując metodę ***notify*** budzi wątek typu ***Watek1*** i sam przechodzi do stanu opóźnienia ***sleep(1000).***

Faza 2

Watek typu ***Watek1*** wznawia działanie metody ***wyswietl1*** nadając zmiennej ***num*** wartość >2 i wywołując metodę ***notify***, która jednak nie budzi wątku typu ***Watek2***, ponieważ jest on jeszcze w stanie sleep. Watek typu ***Watek1*** przechodzi do stanu opóźnienia ***sleep(400).*** Po upływie czasu 400 ms wywołuje ponownie metodę ***wyswietl1*** i przechodzi do stanu wait (prawdziwy warunek **num!=2**) po wywołaniu metody ***wait***.

Każdy wątek powtarza cyklicznie Fazę1 i Fazę 2.

W eksperymencie wykazano, że wolniejszy wątek nie zdąży wejść w stan wait, jednak synchronizacja wątków jest poprawna.

Eksperyment 2

Działanie metod **run**, które po zakończeniu metod wyswietlx **nie wchodzą** w stany opóźnienia Thread.sleep(1000) oraz Thread.sleep(400).

Faza początkowa

Wątek typu ***Watek1*** jako pierwszy wywołał metodę ***wyswietl1*** i wszedł w stan wait (**num!=2**) po wywołaniu metody ***wait***.

Wątek typu ***Watek2*** wykonał metodę ***wyswietl2*** nie wchodząc w stan wait (fałszywy warunek **num==2**), nadając wartość zmiennej ***num*** wartość 2 i wywołując metodę ***notify*** budzi wątek typu ***Watek1.*** Następnie wątek typu ***Watek2*** ponownie wywołuje metodę ***wyswietl2*** i wywołując metodę ***wait*** przechodzi do stanu ***wait*** (prawdziwy warunek **num==2**).

Faza 1:

Watek typu ***Watek1*** wznawia działanie metody ***wyswietl1*** nadając zmiennej ***num*** wartość >2 i wywołując metodę ***notify*** budzi wątek typu ***Watek2***, ponieważ jest on w stanie ***wait***. Następnie wątek typu ***Watek1*** ponownie wywołuje metodę ***wyswietl1*** i wywołując metodę ***wait*** przechodzi do ***wait*** (prawdziwy warunek **num!=2**).

Faza 2:

Watek typu ***Watek2*** wznawia działanie metody ***wyswietl2*** nadając zmiennej ***num*** wartość 2 i wywołując metodę ***notify*** budzi wątek typu ***Watek1***, ponieważ jest on w stanie ***wait***. Następnie watek typu ***Watek2*** ponownie wywołuje metodę ***wyswietl2*** i wywołując metodę ***wait*** przechodzi do ***wait*** (prawdziwy warunek **num==2**).

Każdy wątek powtarza cyklicznie Fazę1 i Fazę2.

W eksperymencie wykazano, że dwa wątki o porównywalnej szybkości działania są synchronizowane za pomocą metod ***wait*** i ***notify*** monitora wspólnego obiektu typu ***Zasob***.