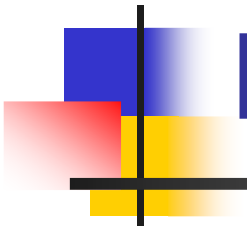
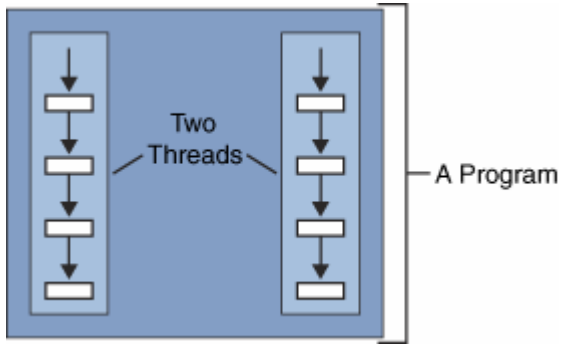


Programowanie w języku Java

Wykład 2: Programowanie współbieżne: podstawy

Wątki w Javie

- Wątek a proces



- Problemy:
  - •tworzenie, usuwanie, start, stop
  - •szeregowanie wątków
  - •synchronizacja wątków
  - •dostęp do wspólnych zasobów

Tworzenie wątków

```
public class SimpleThread extends Thread {
   public SimpleThread(String str) {
      super(str);
   }
   public void run() {
      for (int i = 0; i < 10; i++) {
         System.out.println(i + " " + getName());
         try {
            sleep((long)(Math.random() * 1000));
         } catch (InterruptedException e) {}
      }
      System.out.println("DONE! " + getName());
   }
}
```

```
0 Jamaica
0 Fiji
1 Fiji
1 Jamaica
2 Jamaica
2 Fiji
3 Fiji
3 Jamaica
4 Jamaica
4 Fiji
5 Jamaica
5 Fiji
6 Fiji
6 Jamaica
7 Jamaica
7 Fiji
8 Fiji
9 Fiji
8 Jamaica
DONE! Fiji——Look out, Fiji, here I come!
9 Jamaica
DONE! Jamaica
```

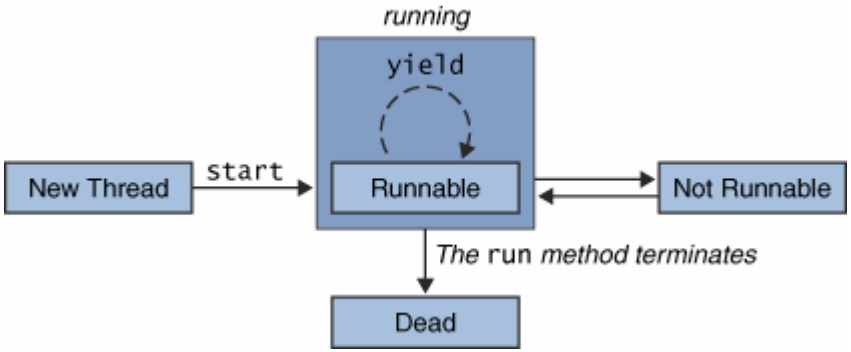
```
public class TwoThreadsTest {
    public static void main (String[] args) {
        new SimpleThread("Jamaica").start();
        new SimpleThread("Fiji").start();
    }
}
```

Interfejs Runnable

```
import java.awt.Graphics;
import java.awt.Color;
import java.util.Date;
public class Clock extends java.applet.Applet
                implements Runnable {
   private Thread clockThread = null;
   public void init() {
      setBackground(Color.white);
   }
   public void start() {
      if (clockThread == null) {
         clockThread = new Thread(this, "Clock");
         clockThread.start();
      }
   }
```

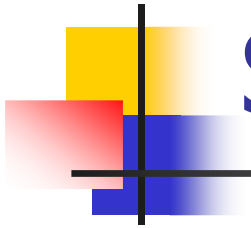
```
public void run() {
      Thread myThread =
Thread.currentThread();
      while (clockThread == myThread) {
         repaint();
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e){ }
      }
   }

public void paint(Graphics g) {
      Date now = new Date();
      g.drawString(now.getHours() + ":" +
       now.getMinutes() + ":" +
       now.getSeconds(), 5, 10);
   }
   public void stop() {
      clockThread = null;
   }
}
```

Cykl życia wątku



- Utworzenie: **clockThread = new Thread(this, "Clock");**
- Uruchomienie: **clockThread.start();**
- Zawieszenie: **Thread.sleep(1000); ....**
- Zatrzymanie: zakończenie metody run()

Thread.getState() - zwraca stan wątku



Szeregowanie wątków

- Wg stałego priorytetu

  setPrioryty(), getPrioryty(), MIN_PRIORYTY, MAX_PRIORYTY,

- zasady przydziału czasu zależą od implementacji

  yield() – rezygnacja z czasu procesora

Synchronizacja wątków

```
public class Producer extends Thread {
   private CubbyHole cubbyhole;
   private int number;
  public Producer(CubbyHole c, int number) {
      cubbyhole = c;
      this.number = number;
   }
public void run() {
      for (int i = 0; i < 10; i++) {
         cubbyhole.put(i);
         System.out.println("Producer #" +
               this.number + " put: " + i);
         try {
            sleep((int)(Math.random() * 100));
         } catch (InterruptedException e) { }
      }
   }
}
```

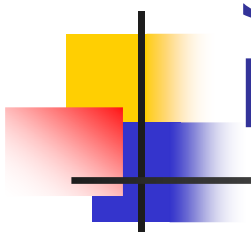
```
public class Consumer extends Thread {
   private CubbyHole cubbyhole;
   private int number;

   public Consumer(CubbyHole c, int number) {
      cubbyhole = c;
      this.number = number;
   }

   public void run() {
      int value = 0;
      for (int i = 0; i < 10; i++) {
         value = cubbyhole.get();
         System.out.println("Consumer #" +
            this.number + " got: " + value);
      }
   }
}
```

```
   ...
Consumer #1 got: 3
Producer #1 put: 4
Producer #1 put: 5
Consumer #1 got: 5
   ...
```

Consumer missed 4

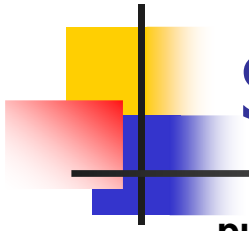
```
   ...
Producer #1 put: 4
Consumer #1 got: 4
Consumer #1 got: 4
Producer #1 put: 5
   ...
```

Consumer got 4 twice



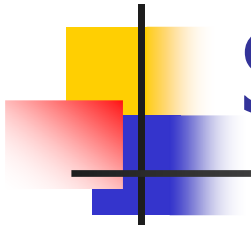
Synchronizacja wątków: sekcje krytyczne

```
public class CubbyHole {
    private int contents;
    private boolean available = false;

    public synchronized int get(int who) {
        ...
    }

    public synchronized void put(int who, int value) {
        ...
    }
}
```



Synchronizacja wątków: wait/notify

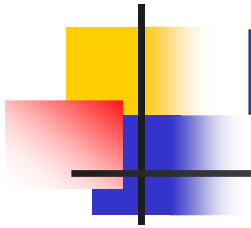
```
public synchronized int get() {
        while (available == false) {
                try {       //wait for Producer to put value
                        wait();
                } catch (InterruptedException e) { }
        }
        available = false;
                //notify Producer that value has been retrieved
        notifyAll();
        return contents;
}
public synchronized void put(int value) {
        while (available == true) {
                try {       //wait for Consumer to get value
                        wait();
                } catch (InterruptedException e) { }
        }
        contents = value;
        available = true;
        //notify Consumer that value has been set
        notifyAll();
}
```



Sekcje krytyczne w metodach

```
synchronized(syncObject) {
// sekcja krytyczna
}
```

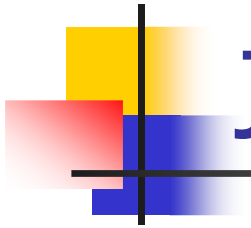
- syncObject – obiekt synchronizujący (zwykle this)
- Jeden obiekt może nadzorować wiele sekcji krytycznych
- Jeden wątek nie może sam siebie zablokować !!



Przerwania

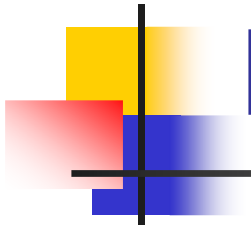
- Metoda interrupt() – wysyła przerwanie do wątku
- Obsługa przerwań:
  - Metody biblioteczne np. sleep() – generacja wyjątku InterruptedException
  - Obsługa w kodzie:

```
if (Thread.interrupted()) {
        throw new InterruptedException();
}
```



join()

- t.join() - Oczekiwanie na zakończenie danego wątku
- t.join(czas) – czekanie max. czas



Demony

setDaemon(), isDaemon()

- Należy ustawić przed start()
- Program kończy działanie po zakończeniu wszystkich wątków nie będących demonami

Koniec