IF2032 – Pemrograman Java Konkuren

Achmad Imam Kistijantoro

Semester II 2008/2009

Overview

- Pengantar konkuren
- Dukungan pemrograman konkuren pada Java
- Contoh

Definisi Konkuren

- Konkuren
  - Melakukan 2 atau lebih aktivitas pada saat bersamaan
- Dalam konteks komputer, aktivitas konkuren:
  - Sistem operasi mampu menjalankan beberapa program bersama-sama
  - Sebuah aplikasi dapat melakukan aktivitas bersamaan: menunggu event dari user (keyboard, mouse), melakukan penyimpanan file ke hard disk, melakukan pencarian teks
- Proses & thread
  - Proses: satuan kontrol eksekusi sebuah program yang memiliki alokasi memori dan user context terpisah
  - Thread: (lightweight process), satuan kontrol eksekusi dalam sebuah program, namun menggunakan memori dan user context yang digunakan bersama dengan thread lain yang masih berada dalam 1 proses yang sama

mengapa konkuren?

- memanfaatkan multicore processor
- simplicity of modeling
- simplified handling of asynchronous event
- more responsive user interface

Problem concurrency

- Sinkronisasi
  - Bagaimana 2 atau lebih dapat berkoordinasi satu sama lain, sehingga dapat menjamin kebenaran program.
  - Dua jenis tipe sinkronisasi: Akses eksklusif dan sinkronisasi kondisi
- Akses eksklusif (mutual exclusion)
  - Resource (memori, external device) hanya boleh digunakan oleh 1 thread saja pada suatu saat tertentu
  - Jika ada 2 atau lebih thread mengakses resource yang sama, mengakibatkan terjadinya race condition: update lost
- Sinkronisasi kondisi (condition synchronization)
  - Sebuah thread baru boleh melanjutkan aktivitasnya setelah kondisi tertentu terpenuhi
  - Buffer management: pembaca buffer baru bisa membaca buffer jika telah ada thread lain yang menuliskan data ke buffer tersebut

Problem concurrency

- Implementasi solusi konkuren yang tidak benar dapat mengakibatkan hasil yang salah, deadlock, starvation dan livelock
- Deadlock: jika 2 atau lebih thread/proses saling menunggu, dan tidak ada yang dapat melanjutkan aktivitasnya
- starvation: jika ada 1 atau lebih thread/proses yang tidak dapat masuk ke resource bersama yang digunakan, karena selalu 'kalah cepat' dibandingkan thread lainnya
- Livelock: sama dengan deadlock, namun setiap thread tidak dalam kondisi menunggu, namun aktif melakukan usaha untuk menghindari terjadinya masalah concurrency

Contoh

- non thread-safe sequence generator

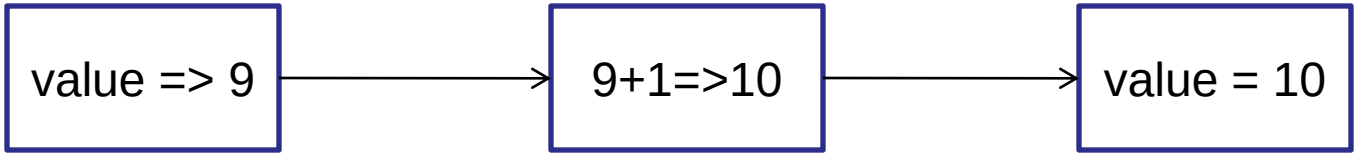
```
public class UnsafeSequence {
  private int value;

  public int getNext() {
    return value;
  }
}
```

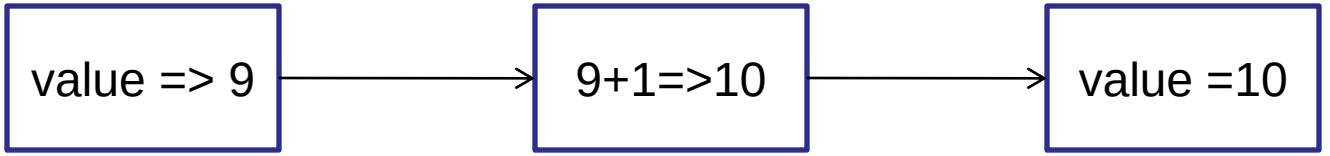
contoh

- operasi value++ sesungguhnya terdiri atas 3 operasi: retrieve value, increment value, store value
- eksekusi berselingan dari ketiga operasi ini mengakibatkan race condition

A



B



Contoh

Akses eksklusif

- P1
  - masuk CS
  - CS
  - Keluar CS
  - Non CS
- P2
  - masuk CS
  - CS
  - Keluar CS
  - Non CS

Condition synchronization

- P1
  - Aktivitas
  - Deposit buffer
  - Aktivitas
- P2
  - Aktivitas
  - Ambil buffer
  - aktivitas

Primitif yang umum disediakan untuk konkurensi

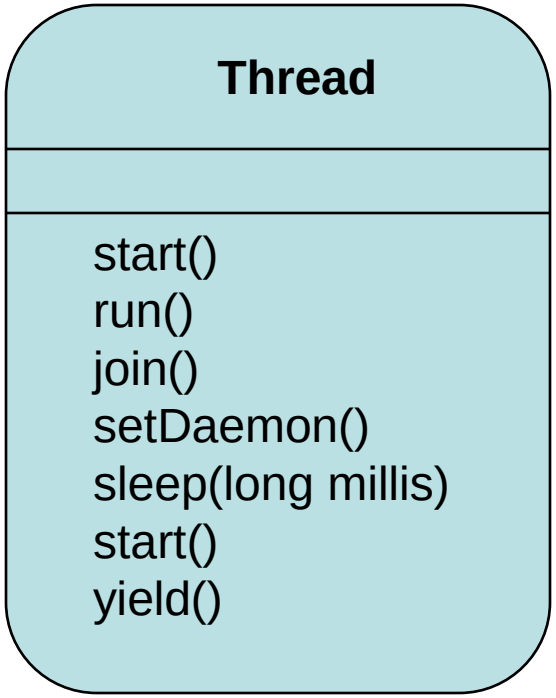
- Atomic variable: tipe yang memiliki primitif set, get, getandset, testandcompare
- Lock: hanya sebuah thread setiap saatnya yang berhasil mendapatkan lock
- Semaphore: lock dengan counter
- Monitor: fasilitas bahasa pemrograman untuk menyatakan sebuah blok kode/data hanya dapat diakses secara eksklusif
- Barrier: beberapa thread dapat menunggu pada barrier hingga semua thread sampai ke lokasi tersebut, baru masing-masing dapat melanjutkan aktivitasnya
- Count down latch: serupa dengan barrier, kriteria lengkapnya berdasarkan jumlah, bukan thread/proses

Dukungan java untuk konkurensi

- Level bawah:
  - kelas Thread, ThreadLocal, ThreadGroup, Timer, TimerTask
  - synchronized, wait, notify, notifyAll
- Level atas (JDK5): java.util.concurrent:
  - Lock
  - Semaphore
  - Executor
  - Atomic variable
  - Concurrent collection

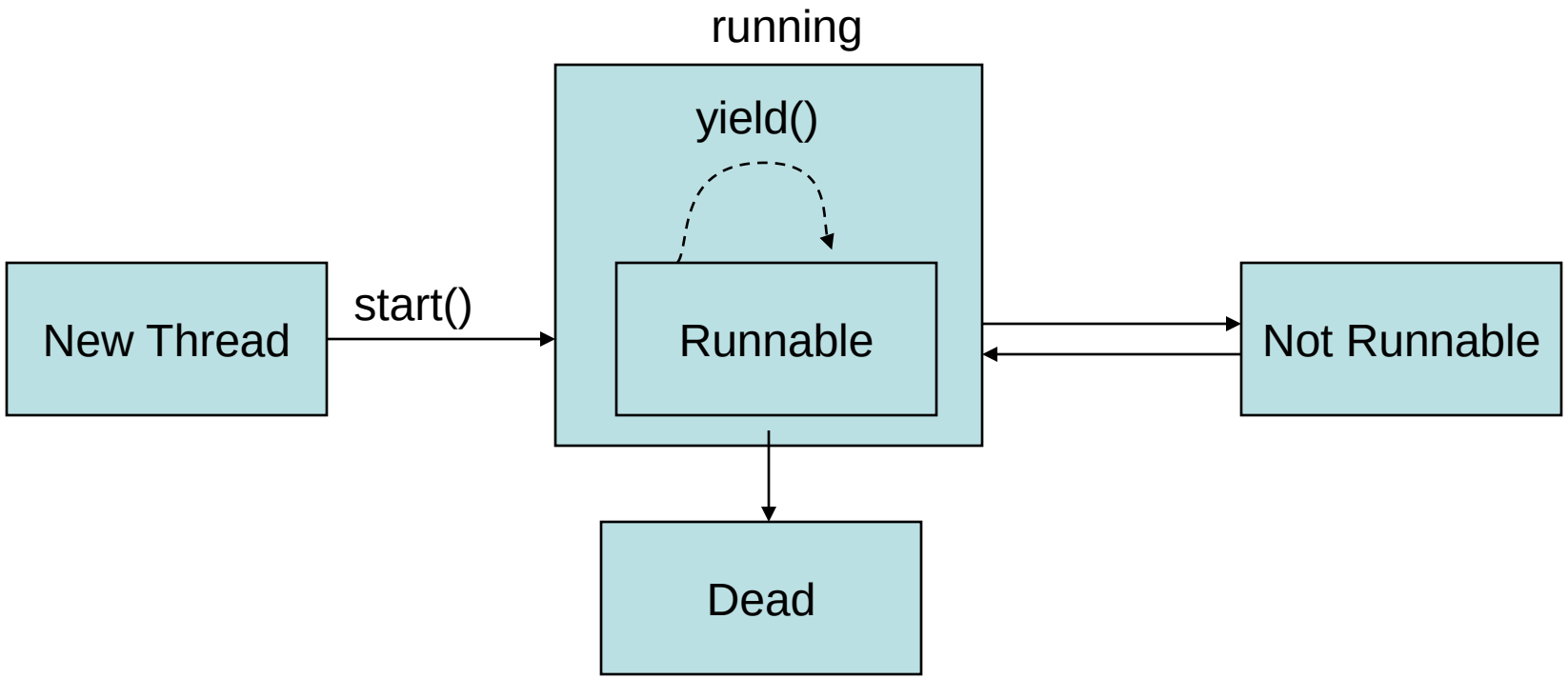
Thread

- Setiap program java memiliki sebuah thread default, yaitu main thread
- Kita dapat membuat thread baru dengan mengaktifkan instance dari kelas Thread atau turunannya



Mengaktifkan thread

- Thread diaktifkan dengan memanggil method Thread.start()
- Thread yang baru aktif akan menjalankan method run()



Mengaktifkan thread

- method run() standar dari kelas Thread tidak melakukan apa2
- kita dapat membuat kelas turunan dari thread, dan mendefinisikan ulang method run()

```
public class T1 extends Thread {
  String s;
  public T1(String str) {
    s = str;
  }
  public void run() {
    for(int i=0;i<100;i++) {
      System.out.println(s);
      try { sleep(1000); } catch(InterruptedException e){}
    }
  }
}
```

Mengaktifkan thread

```
public class Main {

  public static void main(String[] args) {
    T1 t1 = new T1("Thread 1");
    T1 t2 = new T1("Thread 2");
    t1.start();
    t2.start();
    try { t1.join();
          t2.join();
    } catch(InterruptedException e) {}
    System.out.println("Program selesai");
  }
}
```

Mengaktifkan thread: Runnable interface

- kita dapat membuat implementasi dari interface Runnable, dan memberikan objek implementasi Runnable tadi ke objek kelas Thread

```
public class T2 implements Runnable {
  String s;
  public T2(String str) {
    s = str;
  }
  public void run() {
    for(int i=0;i<100;i++) {
      System.out.println(s);
      try { sleep(1000); } catch(InterruptedException e){}
    }
  }
}
```

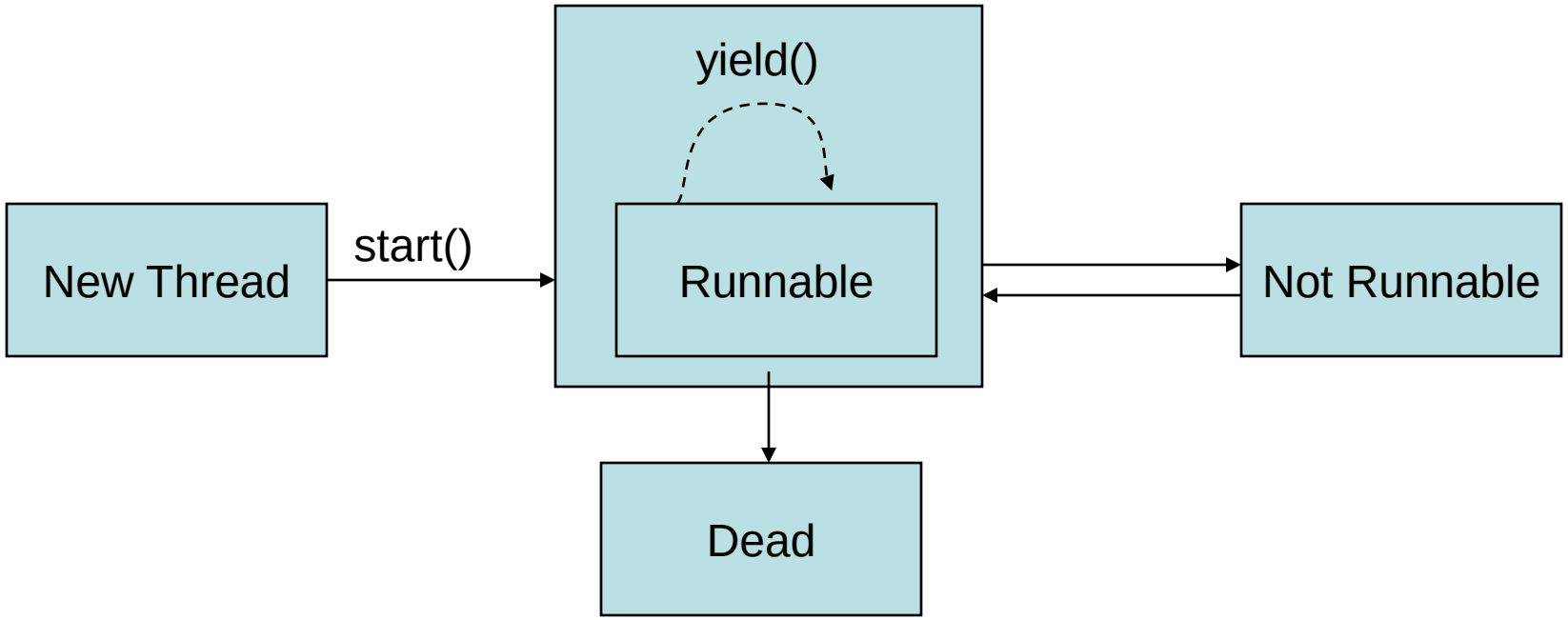
Mengaktifkan thread: Runnable interface

```
public class Main {

  public static void main(String[] args) {
    T2 t1 = new T2("Thread 1");
    T2 t2 = new T2("Thread 2");
    Thread th1 = new Thread(t1);
    Thread th2 = new thread(t2);
    th1.start();
    th2.start();
    try { th1.join();
          th2.join();
    } (InterruptedException e) {}
    System.out.println("Program selesai");
  }
}
```

Siklus hidup thread

- sebuah thread yang sedang aktif dapat dihentikan sementara:
  - menunggu I/O
  - memanggil sleep
  - menunggu event (karena pemanggilan primitif sinkronisasi)



Timer

- Java menyediakan kelas Timer yang memiliki thread terpisah dan dapat menjalankan task tertentu secara periodik
- Setiap objek Timer memiliki background thread untuk menjalankan task secara periodik
- task yang dijalankan objek Timer harus diturunkan dari TimerTask
- Timer:
  - schedule(TimerTask t, long delay);
  - schedule(TimerTask t, long delay, long period);
  - scheduleAtFixedRate(TimerTask t, long delay);
  - scheduleAtFixedRate(TimerTask t, long delay, long period);

Timer

```
public class Reminder {
  public static void main(String args[] ) {
    Timer timer = new Timer();
    timer.schedule( new ReminderTask(), 2000);
    System.out.println("task terjadwal");
  }
}

public class ReminderTask extends TimerTask {
  public void run() {
    System.out.println("Waktu habis!");
  }
}
```

Contoh masalah sinkronisasi: akses eksklusif

- Ada 2 buah thread yang menggunakan printer yang sama untuk mencetak sebuah halaman yang terdiri atas header, isi dan footer. Ketiga bagian ini harus dicetak berurutan untuk setiap threadnya, tidak boleh selang seling.

```
public class Printer {
  public void cetak(String nmThread) {
    System.out.println("cetak header untuk"+nmThread);
    System.out.println("cetak isi untuk"+nmThread);
    System.out.println("cetak footer untuk"+nmThread);
    System.out.println(); // baris kosong
  }
}
```

Contoh masalah sinkronisasi: akses eksklusif

```
public class T3 extends Thread {
  String nama;
  Printer prn;
  public T3(String s, Printer p) {
    nama = s;
    prn = p;
  }
  public void run() {
    for(int i=0;i<100;i++) {
      p.cetak( nama );
    }
  }
}
```

Contoh masalah sinkronisasi: akses eksklusif

```
public class Main {

  public static void main(String[] args) {
    Printer p = new Printer();
    T3 t1 = new T3("Thread 1", p);
    T3 t2 = new T3("Thread 2", p);
    t1.start();
    t2.start();
    try { t1.join();
          t2.join();
    } catch(InterruptedException e) {}
    System.out.println("Program selesai");
  }
}
```

Hasil

```
Thread 1:cetak header
Thread 2:cetak header
Thread 1:cetak isi
Thread 2:cetak isi
Thread 1:cetak footer
Thread 1:cetak header
Thread 1:cetak isi
Thread 1:cetak footer
Thread 1:cetak header
```

synchronized

- Pada java, setiap objek memiliki lock internal
- penggunaan konstruksi synchronized akan menjamin hanya ada 1 thread saja yang aktif menjalankan blok yang dilingkupi oleh konstruksi synchronized

```
public class Printer {
  public synchronized void cetak(String nmThread) {
    System.out.println("cetak header untuk"+nmThread);
    System.out.println("cetak isi untuk"+nmThread);
    System.out.println("cetak footer untuk"+nmThread);
    System.out.println(); // baris kosong
  }
}
```

synchronized

- synchronized dapat meliputi blok, tidak harus berupa method:

```
public class Printer {
  Lock lock;
  public void setLock(Lock l) {
    lock = l;
  }
  public void cetak(String nmThread) {
    synchronized (lock) {
      System.out.println("cetak header untuk"+nmThread);
      System.out.println("cetak isi untuk"+nmThread);
      System.out.println("cetak footer untuk"+nmThread);
    }
    System.out.println(); // baris kosong
  }
}
```

Contoh masalah sinkronisasi: buffer management

- Ada dua jenis thread: produsen dan konsumen. Produsen membangkitkan bilangan, dan menyimpannya pada buffer. Konsumen membaca bilangan, dan menampilkannya ke layar.

```
public class Produsen
extends Thread {
  Buffer buf;
  public Produsen(Buffer b) {
    buf = b;
  }
  public void run() {
    for(int i=0;i<100;i++) {
      buf.set(i);
    }
  }
}
```

```
public class Konsumen
extends Thread {
  Buffer buf;
  public Konsumen(Buffer b) {
    buf = b;
  }
  public void run() {
    int x;
    for(int i=0;i<100;i++) {
      x = buf.get(i);
      System.out.println("ambil x"+x);
    }
  }
}
```

Contoh masalah sinkronisasi: buffer management

```
public class Buffer {
  int value;

  public void set(int v) {
    value = v;
  }

  public int get() {
    return value;
  }
}
```

```
public class Main {
  public static void
   main(String args[]) {
     Buffer b = new Buffer();
     Produsen p = new
       Produsen(b);
     Konsumen k = new
       Konsumen(b);
     p.start();
     k.start();
     try { p.join();
           k.join();
}catch(InterruptedException e)
   {}
}}
```

synchronized tidak cukup!

- produsen dapat menghasilkan bilangan 2 kali tanpa terbaca konsumen
- konsumen dapat membaca bilangan yang sama 2 kali tanpa data baru dari produsen

```
public class Buffer {
  int value;

  public synchronized void set(int v) {
    value = v;
  }

  public synchronized int get() {
    return value;
  }
}
```

solusi: menggunakan variable

```
public class Buffer {
  int value;
  boolean available = false;

  public void set(int v) {
    while( available ) {}
    value = v;
    available = true;
}

  public int get() {
    int result;
    while( !available ) {}
    result = value;
    available = false;
    return result;
  }
}
```

kelemahan

- busy waiting, menggunakan resource CPU
- tidak efisien

wait & notify

- wait(): mekanisme yang disediakan oleh setiap kelas (diimplementasikan pada kelas Object) untuk menunggu event tertentu
- wait() dan notify() hanya dapat dipanggil oleh thread yang memegang lock untuk objek tersebut (dipanggil dari dalam blok synchronized)
- saat thread t memanggil wait(), maka thread tersebut akan melepaskan monitor objek m (sehingga thread lain dapat masuk ke blok synchronized), dan t dimasukkan ke dalam daftar waiting list untuk objek m.
- saat thread t diaktifkan kembali (keluar dari waiting list) dengan pemanggilan notify atau notifyAll oleh thread lainnya, t harus mendapatkan monitor untuk m agar dapat meneruskan operasinya

wait & notify

- saat thread t memanggil notify() dari objek m, maka jika ada thread lain yang berada dalam waiting list untuk objek m, maka thread lain tersebut akan dikeluarkan dari waiting list
- saat thread t memanggil notifyAll() dari objek m, maka semua thread yang berada dalam waiting list objek m akan dikeluarkan

```
public class Buffer {
  int value;
  boolean available = false;
  public synchronized void
   set(int v) {
    while(available) {
      try {
        wait();
      }
   catch(InterruptedException
   e) {}
    }
    value = v;
    available = true;
    notify();
  }
```

```
 public synchronized int get()
   {
    while(!available) {
      try(
        wait();
      }
    catch(InterruptedException e)
    {}
     }
     available = false;
     notify();
     return value;
   }
}
```