Threads

Vittayasak Rujivorakul

Lecture 9

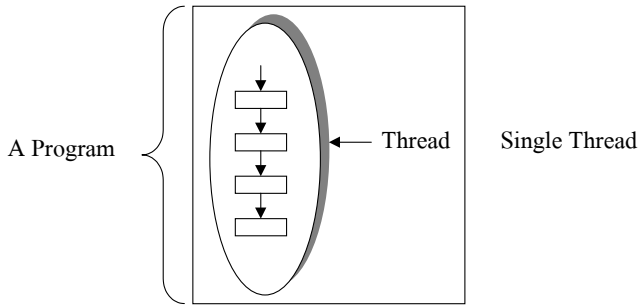
Threads

Vittayasak Rujivorakul

Lecture 9

Threads คืออะไร

- **Definition :** A Thread = a single sequential flow of control within a program
  - ลำดับของ flow ของการควบคุมใน program ใช้ในงานเฉพาะอย่างหนึ่ง

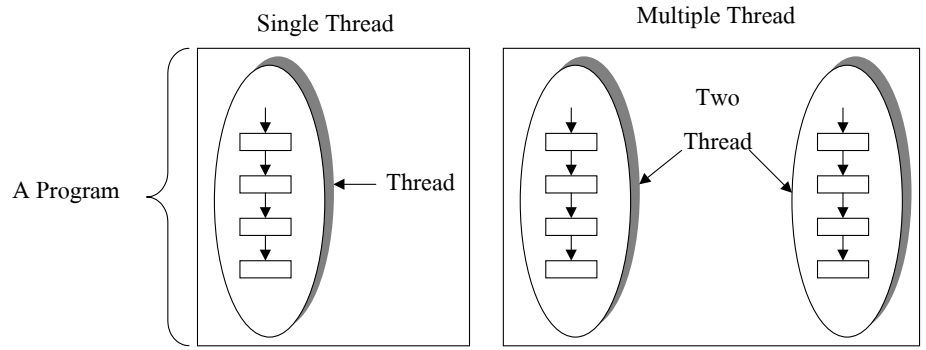


Thread

- Thread ถูกเรียกว่า Execution Context หรือ lightweight process
  - มีจุดเริ่มต้น, ลำดับการทำงาน และจุดสิ้นสุด มีจุดเริ่มการทำงานจุดเดียว
  - แต่ไม่ใช่ program
  - เป็น Lightweight : ทำงานภายใต้ context ของ program โดยใช้ resources ที่จัดสรรและสิ่งแวดล้มของ program นั้น
  - Execution Context : ต้องมี resources บางส่วน (e.g. execution stack และ program counter) เป็นของตนเองภายใต้ program

Multiple Threads

- Multiple Threads = ลำดับของ flow ของการควบคุมหลายอันใน 1 program โดยที่แต่ละ thread นั้นทำงานที่แตกต่างกันและเป็นอิสระต่อกัน แต่ทำงานในเวลาเดียวกัน



run Method

- **"run" Method :** ส่งการทำงานให้กับ Thread
  - code ที่ implement ภายใน run เป็นตัวกำหนดหน้าที่หรือสิ่งที่ต้องทำ
- run Method ใน thread class Java โดย default ไม่ได้ทำอะไร

  สามารถกำหนดการทำงานของ Thread ได้ใน 2 ลักษณะ
  - Subleasing Thread and Overriding run
  - Implementing the Runnable Interface

Subclassing Thread and Overriding run

- ทำการ subclass Thread object
  - Thread โดยตัวมันเองเป็น Runnable object ด้วย
- เขียน run method ขึ้นใหม่

ตัวอย่าง Program

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


ตัวอย่าง Program (ต่อ)

```
Public class TwoThreadTest   {
    public static void main (String [ ] args)   {
        new SimpleThread  ("Phuket").start ( ) ;
        new SimpleThread ("Chiangmai").start ( ) ;
    }
}
```

สังเกต ผลลัพธ์ที่ได้จาก thread แต่ละอันนั้นจะผสมกันกับผลลัพธ์จาก thread อื่น เนื่องจากว่า simple Thread ทั้ง 2 อันนั้นทำงานพร้อมกัน ดังนั้น run methods ของทั้ง 2 จึงทำงานในขณะเดียวกัน โดยที่แต่ละ thread ก็จะทำการออกผลลัพธ์ของตนเอง

Implementing the Runnable Interface

- ทำการสร้าง class ซึ่ง implements Runnable interface ซึ่งจะมี run method อยู่ในตัวมัน
- ทำการสร้าง Thread โดยให้ class ข้างต้นเป็น Thread' s target
  - ซึ่งเมื่อสร้างในแบบนี้, Thread จะรับ run method มาจากตัว target ที่เราสร้างขึ้นนั้นเอง

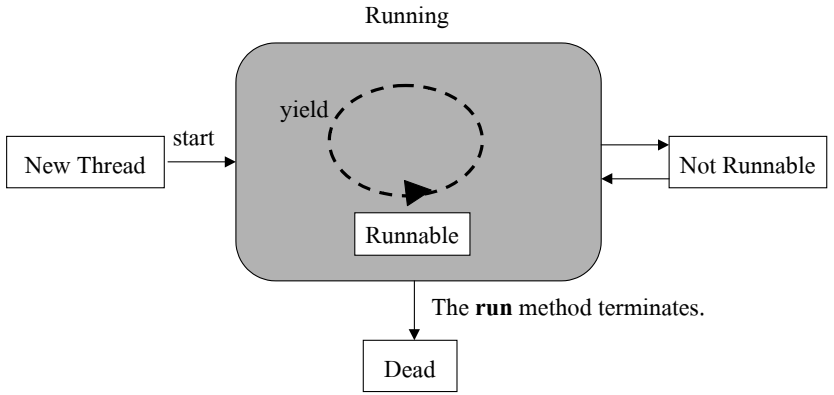
ตัวอย่าง

```
Public class ThreadTest   {
    public static void main (String args [ ] )  {
          MyRun  r  =  new  MyRun ( ) ;
          Thread  t  =  new  Thread ® ;
          t.start ( ) ;
    }
}
class MyRun implements Runnable    {
    int  i ;
    public void run ( )    {
          while  (true)    {
                System.out.println ("Hello" + i + + ) ;
                if  ( i = = 30) break ;
          }
    }
}
```


การเลือกใช้

- เหตุผลในการเลือกใช้วิธีใดนั้น อาจเป็นไปได้ด้วยเหตุหลายกรณีหากแต่
- **Rule of Thumb :**
  - ถ้าหาก class นั้นจำเป็นต้อง subclass จาก class อื่น (e.g., Applet), ต้องเลือกใช้การทำ Runnable ในวิธีที่ 2

วงชีวิตของ Thread



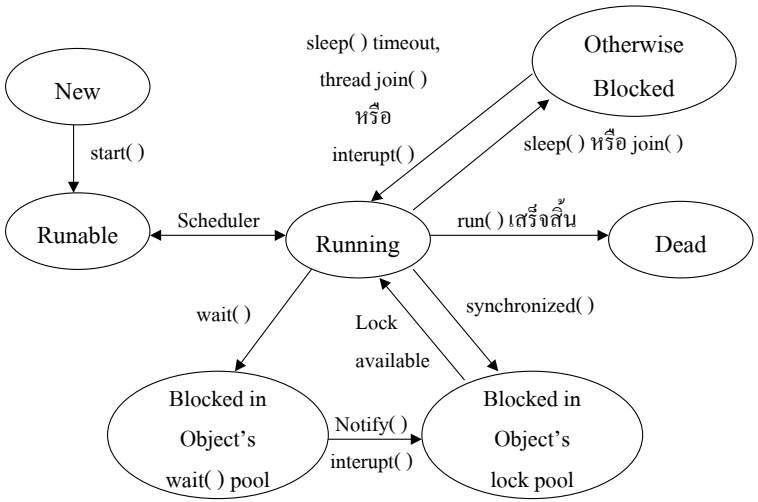
วงชีวิตของ Thread

- สร้าง Thread : clockThread = new Thread(this, “Clock”) ;
- สั่งให้ Thread เริ่มต้นการทำงาน : clockThread.start ( ) ;
  - ทำการ allocate resource พร้อมกันกับเรียก run method ให้ทำงาน
  - ในช่วงนี้ Thread จะอยู่ใน Runnable State (scheduling, share resources)
- Thread จะหยุดการทำงานชั่วคราวหาก (Not Runnable)
  - มีการเรียก sleep method หรือ
  - มีการเรียก wait method เพื่อรอข้อแม้บางอย่างเกิดขึ้นและเป็นจริง หรือ
  - Thread ต้องรอผลการทำงานจาก I/O

วงจรชีวิตของ Thread

- Thread จะตื่นขึ้นมาทำงานอีกครั้ง เมื่อ
  - เมื่อถึงครบช่วงเวลาที่กำหนดไว้ให้นอน หากถูกสั่งให้ sleep
  - หากถูกสั่งให้รอข้อแม้บางอย่างจาก wait เมื่อมี object อื่นมาปลุกตามข้อและทำการ notify หรือ notifyAll
  - เมื่อ I/O ทำงานเสร็จ หาก block รอการทำงานของ I/O
- การหยุดทำงานของ Thread
  - หยุดเมื่อทำงานเสร็จสิ้นตามที่กำหนดไว้ใน run method
  - ต่างจากการ start และ stop Applet

Thread States



การตรวจสอบ Thread

- “**isAlive**” Method ใช้ในการตรวจสอบสถานะของ Thread ว่ายังมีชีวิตอยู่หรือสิ้นสุด
  - true : ถ้า Thread มีชีวิตอยู่ (ได้ start แล้วและยังไม่ stop) นั่นคือ
    - Runnable
    - Not Runnable
  - false : Thread นั้นไม่ได้มีการทำงานแล้ว อาจเป็น
    - New Thread
    - Dead Thread

Thread Priority

- ในเครื่องที่มี 1 CPU, เมื่อหลาย threads มีการทำงาน
  - ระบบทำงานคล้ายกับว่าหลาย threads นั้นทำงานพร้อมกัน
  - อาศัยการจัดคิวลำดับการทำงานที่เรียกว่า การทำ scheduling
- Java ใช้การทำ scheduling แบบง่าย ๆ : Fixed Priority Scheduling
  - การจัดลำดับขึ้นกับ priority ของ thread นั้นเปรียบเทียบกับ thread อื่น ๆ
  - เป็น preemptive scheduling (higher priority task preempts lower one)
- Thread จะ inherit priority จาก Thread ที่สร้างมันขึ้นมา
  - เปลี่ยน priority ได้โดยใช้ setPriority method หลังจาก Thread เกิดขึ้น

การเลือก thread ขึ้นมาทำงาน

- การเลือก thread ขึ้นมาทำงาน
  - เลือก thread ที่มี priority สูงกว่าทำงานก่อน
  - priority เท่ากันจะเลือกให้เข้าทำงานในลักษณะ round-robin
- thread ที่ถูกเลือกให้ทำงานจะหยุดการทำงาน ถ้า
  - มี thread ที่มี priority สูงกว่าที่มีหรือเปลี่ยนสถานะเป็น Runnable
  - เสร็จงาน หรือ run method สั่ง exit
  - หมดช่วงเวลาที่ให้ทำงานได้ ในระบบที่สนับสนุน time-slicing

Synchronizing Threads

- ในบางกรณี เราต้องการให้ thread นั้นทำงานประสานกัน
  - การทำงานของ thread ตัวหนึ่งอาจต้องรอผลจากอีก thread หนึ่ง (e.g. Producer-Consumer) โดยการ Synchronized การทำงานนั้น
  - เมื่อทำการ synchronized แล้ว threads จะทำการ locks objects เมื่ออีก threads ต้องการเรียกใช้ก็ต้องรอให้ thread แรกทำเสร็จและ unlock object นั้น
  - Java สนับสนุนวิธีการ wait, notify และ notifyAll เพื่อให้ threads เลือกคอย (wait) condition หรือถูกเรียกให้กลับขึ้น (notify) มาทำงาน

Grouping Threads

- การกำหนดกลุ่มให้ threads เพื่อให้เกิดความง่ายในการจัดการ
- โดย default thread จะมีกลุ่มเดียวกับตัวสร้างมัน
  - ทั่วไปหากไม่กำหนดใด ๆ thread จะอยู่ภายใต้ ThreadGroup ซึ่ง main
- กำหนดให้ threads เป็นสมาชิกของกลุ่มของ thread ได้

public Thread (ThreadGroup group, Runnable target)

public Thread (ThreadGroup group, String name)

public Thread (ThreadGroup group, Runnable target, String name)

- สามารถสร้างกลุ่ม thread ขึ้นเองได้จาก ThreadGroup class