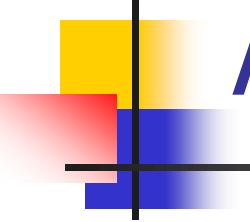


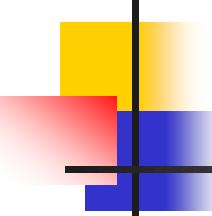
The logo for Threads features a stylized graphic element on the left. It consists of a black vertical bar intersected by a horizontal bar. To the left of this intersection, there are three colored rectangular blocks: a blue one at the top, a red one below it, and a yellow one at the bottom. The text "Threads" is positioned to the right of the graphic.

Threads



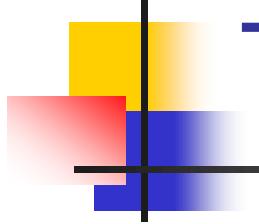
Agenda

- Threads
- Thread status and lifetime
- Thread in Java
- Thread Class
- Thread priorities
- Sleep
- Suspend and Resume
- Join
- Preemptive and Non-preemptive
- Yield
- Stop
- Daemon Thread



Thread

- ในระบบ Multitasking ซึ่งมีหลาย process ทำงานอยู่ในเวลาเดียวกัน ทำให้เกิดการเสียเวลาในการเปลี่ยนจาก process หนึ่งไปสู่อีก process หนึ่ง (Context Switching)
- ในระบบปฏิบัติการแบบ multi-threading สามารถแบ่ง process ออกเป็นหน่วยเล็ก ๆ ซึ่งเรียกว่า **thread**
- 1 thread คือ 1 execution flow ซึ่งการนำหลาย ๆ thread มารวมกันในหนึ่ง process จะทำให้ process นั้นสามารถทำงานได้หลายอย่าง โดยที่ไม่ต้องมีการทำ context switching
- ในระบบ Multi-threading 1 process มีหนึ่ง PCB (Process Control Block) แต่มีหลาย thread อยู่ข้างใน
- Thread ใช้ code segment ร่วมกันและใช้ data segment บางส่วนร่วมกัน
- การจัดการให้ Thread ได้ลุกทำงานเมื่อใด โดย CPU ตัวไหน ขึ้นอยู่กับ OS
- ผู้เขียนโปรแกรมทำได้เพียงสร้าง thread ให้ทำหน้าที่ตามต้องการ โดยควบคุมให้ thread เริ่ม, หยุด หรือรอการทำงาน

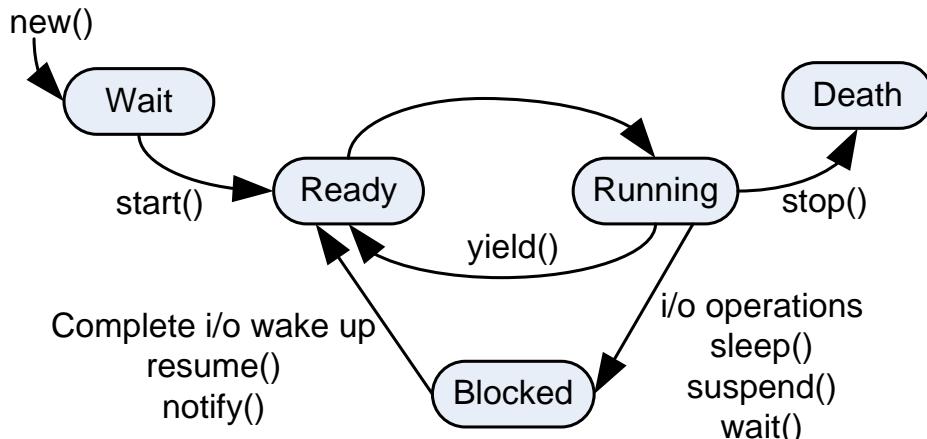


Thread

ข้อแตกต่างระหว่าง Thread กับ Process

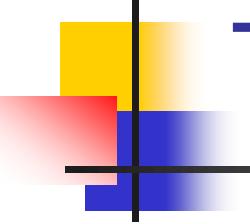
- การทำงานของแต่ละ Thread จะทำงานอยู่ภายใต้ Environment เดียวกัน ซึ่ง Process จะทำงานแบบแยก Environment
- Thread สามารถติดต่อกันได้โดย shared variable และ process ต้องติดต่อกันโดยผ่านทางไฟล์ (pipe)
- Context switching ของ process ต้องเปลี่ยน PCB แต่ Thread ไม่ต้องทำ context switching

Thread status and lifetime



1. thread ถูกสร้างขึ้นจากการ new instance ของ class thread และจะอยู่ในสถานะ wait เมื่อได้รับคำสั่ง start จะย้ายสถานะไปอยู่ใน ready (เป็นการเข้าคิว) เพื่อรอ OS เรียกไปทำงานสถานะ running
2. เมื่อได้รับคำสั่ง start จะย้ายสถานะไปอยู่ใน ready (เป็นการเข้าคิว) เพื่อรอ OS เรียกไปทำงานสถานะ running

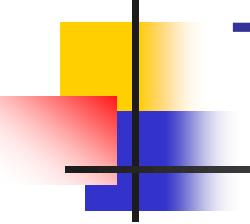
3. สถานะ running เป็นสถานะที่ทำการ run thread เมื่อไม่มี thread ที่สถานะนี้ OS จะทำการเรียก thread ที่อยู่ในสถานะ ready เข้ามาทำงาน สถานะ running สามารถเปลี่ยนสถานะดังไปได้ 3 สถานะคือ
 1. Death เมื่อได้รับคำสั่ง stop ซึ่งทรัพยากรของ thread นั้นจะถูกคืนสู่หน่วยความจำ
 2. Ready เมื่อ thread ถูก interrupt หรือได้รับคำสั่ง yield
 3. Blocked เมื่อได้รับคำสั่ง wait, suspend หรือ sleep
4. thread ที่อยู่ในสถานะ blocked จะเปลี่ยนเป็น ready เมื่อได้รับคำสั่ง notify, resume หรือหมดเวลาของการ sleep หรือได้รับทรัพยากรที่กำลังรออยู่



Thread in Java

Class Thread มี data members ดังนี้

- **private char name[];**
 - เป็นชื่อของ thread โดยกำหนดผ่าน constructor
- **private int priority;**
 - ระดับความสำคัญของ thread
- **private boolean daemon = false;**
 - เป็นการกำหนดว่า thread นี้เป็น daemon thread หรือไม่

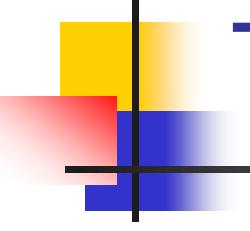


Thread in Java

แสดงถึง **thread** ที่มีอยู่แล้วใน **main()**

```
// MainThread.java
class MainThread {
    public static void main(String args[]) {
        Thread t = Thread.currentThread();
        System.out.println(t.getName());
        System.out.println(t.getPriority());
        System.out.println(t.isAlive());
        System.out.println(t.isDaemon());
    }
}
```

main
5
true
false



Thread in Java

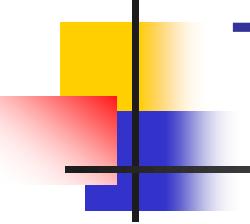
คลาส **thread** มี method สำหรับกำหนดค่าดังนี้

public final void setName(String name)

public final String getName()

public final void setPriority(int newPriority)

public final int getPriority()



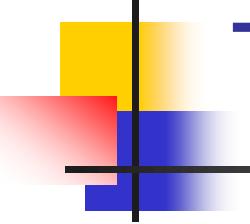
Thread class

เมื่อ thread ได้รับคำสั่ง start() มันจะไปเข้าคิวเพื่อรอการเรียก เมื่อมันถูกเรียกไปทำงาน โปรแกรมใน run() ก็จะถูกทำงาน

คลาส thread มี method สำหรับควบคุมให้ thread เข้าสู่สถานะต่าง ๆ เช่น start(), yield(), sleep(), suspend(), resume() และ stop() ซึ่ง method เหล่านี้เราจะไม่ override

```
public void run ()
```

เป็น method ที่ว่างเปล่า มีไว้สำหรับ override ด้วย code ที่ต้องการ ซึ่งถ้าไม่ทำการ override method นี้ จะทำให้เราได้ thread ที่ไม่มีคำสั่งใด ๆ ให้ run



Thread class

แสดงการขยายคลาส **thread**

```
// ThreadTest.java
class MyThread extends Thread {
    MyThread(String n){ super(n); }
    public void run(){
        for (int i =0; i < 100; i++)
            System.out.print(getName());
    }
}
class ThreadTest {
    public static void main(String args[]){
        new MyThread("A").start();
        new MyThread("B").start();
    }
}
```

AAAABBBBBBBBBBBBBB
BBBBBBBBBAAAAAAA
AAAAABBBBBBBBBBBB
BBBBBBBBBBBBBAAAA
AAAAAAAAAAAAAAA
AAAAAAAAAAAAAAA
AAAAABBBBBBBBBBBB
BBBBBBBBBBBBBBBBBB
BAAAAAAA
AAAAAAABBBBBBBB
BBBBBBBAAAAAA

Thread Priority

ภาษา Java กำหนดให้ค่า priority ได้ 7 มีค่าอยู่ระหว่างต่ำสุดคือ 1 และสูงสุดคือ 10 ซึ่งในคลาส thread มีค่าคงที่ของ thread priority ไว้ 3 ค่าคือ

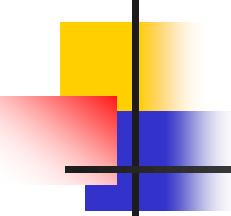
```
public final static int MIN_PRIORITY = 1;
```

```
public final static int NORM_PRIORITY = 5;
```

```
public final static int MAX_PRIORITY = 10;
```

```
// Priority.java
// using class MyThread from ThreadTest.java
class Priority {
    public static void main(String[] args) {
        Thread a = new MyThread("A");
        Thread b = new MyThread("B");
        a.setPriority(6);
        a.start();
        b.start();
    }
}
```

```
AAAAAAA
AAAAAAA
AAAAAAA
AAAAAAA
AAAAAAA
AAAAAAAABBBBBB
BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBB
```



Sleep

ภาษา Java สามารถทำให้ thread ทำงานช้าลงได้โดยการใช้คำสั่ง

```
public static native void sleep (long m) throws InterruptedException
```

โดย long m เป็นตัวกำหนดระยะเวลาที่จะให้ thread นั้นหลับ(อยู่ในสถานะ blocked) หน่วยเป็น millisecond

```
//SleepTest1.java
class MyThread extends Thread {
    long t;
    MyThread(String n, long t){ super(n); this.t = t; }
    public void run(){
        for (int i =0; i < 100; i++){
            System.out.print(getName());
            try { sleep(t); }
            catch (InterruptedException e){ }
        }
    }
}
class SleepTest1 {
    public static void main(String args[]){
        new MyThread("A", 10).start();
        new MyThread("B", 20).start();
    }
}
```

```
ABAABAABAAABAABAABAABAA
BAABAABAABAABAABAABAABA
ABAABAABAABAABAABAABAAB
AABABAABAAABAABAABAABAABAA
BAABAABAABAABAABAABAABAABA
ABAABAABAABAABB BBB BBBB BBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBB
```

suspend and resume

เราสามารถสั่งให้ thread ที่อยู่ในสถานะ running ย้ายไปอยู่ในสถานะ blocked โดยไม่มีการกำหนดเวลาได้โดยการใช้คำสั่ง suspend และสั่งให้ thread นั้นออก จากสถานะ blocked โดยการใช้คำสั่ง resume

```
ABABABBABABABABABA  
A is suspended  
BBBBBBBB  
A is resumed  
ABABABABABABABABABABABAAB  
ABABABABABAAAAAA
```

```
// SusRes.java  
class MyThread extends Thread {  
    MyThread(String s){ super(s); }  
    public void run(){  
        for (int i =0; i < 50; i++){  
            System.out.print(getName());  
            try { sleep(100); }  
            catch (InterruptedException e) { }  
        }  
    }  
}  
class SusRes {  
    public static void main(String args[]){  
        MyThread a = new MyThread("A");  
        MyThread b = new MyThread("B");  
        a.start();  
        b.start();  
        try {  
            Thread.sleep(1000);  
            a.suspend();  
            System.out.println("\n A is suspended");  
            Thread.sleep(1000);  
            a.resume();  
            System.out.println("\n A is resumed");  
        } catch (InterruptedException e) { }  
    }  
}
```

Join

ปกติ thread ลูกทุกตัวควรจะต้องจบก่อน thread แม่ เพื่อที่จะได้คืนทรัพยากรของ thread ลูกให้อย่างสมบูรณ์ ซึ่งถ้า thread แม่จะรอ thread ลูกโดยใช้คำสั่ง sleep หรือ suspend จะไม่หมายความว่าจากไม่สามารถคำนวณเวลาของ thread ลูกได้

เป็น method ที่อนุญาตให้ thread หนึ่งทำการ join กับอีก thread หนึ่ง ทำให้ thread ผู้เรียกหยุดการทำงานและรอไปจนกว่า thread ผู้ถูกเรียกจะทำงานเสร็จ จึงจะทำงานต่อจากจุดเดิมที่เรียก join

```
// JoinTest.java  
// using MyThread class from SusRes.java  
class JoinTest {  
    public static void main(String args[]) {  
        MyThread a = new MyThread("A");  
        MyThread b = new MyThread("B");  
        a.start(); b.start();  
        try {  
            a.join();  
            b.join();  
        } catch (InterruptedException e) {}  
        System.out.println("\n Main stops");  
    }  
}
```

```
ABBAABBABAABBABABABBABAB  
ABABABABABABABBAABBABABABAB  
BBABABABABBAABABABABAB  
Main stops
```

Yield

public static native void yield()

เป็นคำสั่งที่ทำให้ thread ที่ทำงานอยู่ย้อนกลับไปสู่สถานะ ready เพื่อให้ thread อื่นเข้ามารажงานแทน ซึ่งต่างกับ sleep และ suspend ตรงที่

sleep จะมีการกำหนดเวลาหลับ

suspend ต้องทำการ **resume**

แต่ yield จะยอมเข้าไปอยู่ในคิว ซึ่งมันจะถูกเรียกขึ้นมาใหม่เมื่อทำการตัดสินใจของ OS

```
// YieldTest.java
class MyThread extends Thread {
    MyThread(String s){ super(s); }
    public void run(){
        for (int i=0; i < 100; i++){
            System.out.print(getName());
            yield();
        }
    }
}
class YieldTest {
    public static void main(String[] args){
        new MyThread("A").start();
        new MyThread("B").start();
    }
}
```

**ABBAABBABAABBABABABBBB
ABABABABABABABBAABBABABABAB
BBBABABABBAAAABABABABABB
ABABABABABABABAaaaaaaaaaaaa
ABABABABABABABABABABABABA
BABABABABABABABABABABABAB
ABABABABABABABABABABABABABB
BBBBBBBBBB**

Stop

- คำสั่ง stop จะทำให้ thread เข้าสู่สถานะ death และทรัพยากรทั้งหมดก็จะถูกส่งคืน OS
- ในบางกรณีที่เราต้องการจะให้เกิดการทำงานอย่างได้อย่างหนึ่งก่อนที่ thread นั้นจะถูก stop (เพราะ thread มักถูก stop โดย thread อื่นโดยไม่รู้ตัว)
สามารถทำได้โดยการใช้ try-catch block เพื่อดักจับคลาส ThreadDeath (Java Interpreter จะทำการโยน Instance ของ class นี้เข้าไปสู่ thread ที่ถูก stop)
- ThreadDeath เป็นคลาสลูกของคลาส Error

```
// StopTest.java
class MyThread extends Thread {
    public void run() {
        try {
            while(true) {
                sleep(1000);
                System.out.println("running");
            }
        } catch(ThreadDeath e) {
            System.out.println("killed");
            throw(e);
        } catch(InterruptedException e) { }
        System.out.println("Stop");
    }
}
class StopTest {
    public static void main(String args[]) {
        MyThread a = new MyThread();
        a.start();
        try { Thread.sleep(3000); } catch(InterruptedException e) { }
        a.stop();
    }
}
```

running
running
killed

Daemon Thread

- ในบางครั้ง thread หนึ่งต้องทำการสร้าง thread ลูกเพื่อทำการสนับสนุนตัวมันเอง และ thread ลูกเหล่านั้นก็จะต้องถูก stop ก่อนที่ thread แม่จะจบชีวิตแล้ว แต่ถ้ามี thread ลูกมากก็จะเป็นปัญหาในการจัดการ
- Java จึงสร้าง thread พิเศษขึ้นชื่อเรียกว่า **daemon thread** ซึ่งเป็น thread ที่ไม่สามารถทำงานได้โดยปราศจาก thread ธรรมดานะ
- JVM จะทำการตรวจสอบว่าถ้า thread ที่ทำงานอยู่มีแต่ daemon thread เท่านั้น JVM จะทำการหยุดตัวเองลงซึ่งจะทำให้ daemon thread ตายลงโดยอัตโนมัติ

```
// DaemonThread.java
class MyThread extends Thread {
    MyThread(String n){ super(n); }
    public void run(){
        while(true){
            try { sleep(1000);
            } catch(Exception e){ }
            System.out.println(getName());
        }
    }
}
class DaemonThread {
    public static void main(String args[]){
        MyThread a = new MyThread("a");
        a.setDaemon(true);
        a.start();
        try { Thread.sleep(3000);
        } catch(Exception e){ }
        System.out.println("Main stops");
    }
}
```

a
a
Main stop