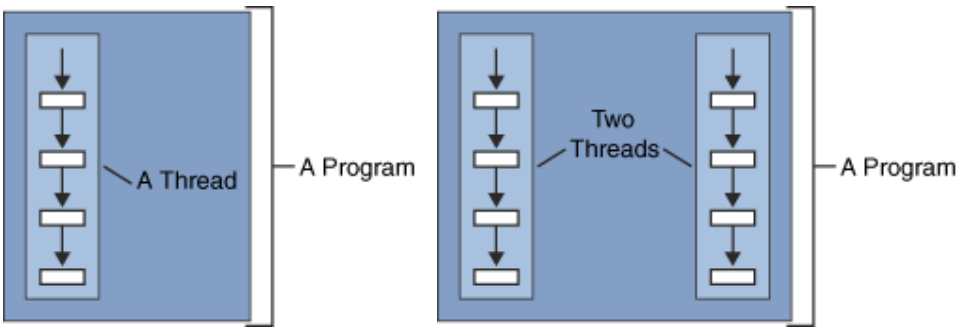
ทำไมถึงต้อง Thread?

Threading เป็นเครื่องมือที่ช่วยให้กิจกรรมหลายอย่างในระบบปฏิบัติการทำงานร่วมกันได้ในเครื่องคอมพิวเตอร์ที่มี CPU เพียงตัวเดียว เราเรียกระบบการทำงานนี้ว่า Multitasking ภาษาที่ใช้ในการเขียนโปรแกรมที่รองรับการทำงานแบบนี้ เช่น Java นั้นก็ใช้ thread ในโปรแกรมอย่างน้อยหนึ่งตัว ซึ่งเราเรียกว่า main thread เมื่อโปรแกรม Java เริ่มการทำงาน (start) JVM จะสร้าง main thread ขึ้นมาพร้อมกับเรียก method main() ที่อยู่ใน thread นั้น ในเวลาเดียวกัน JVM ก็อาจสร้าง thread ตัวอื่นที่เรามองไม่เห็น เช่น thread ที่ใช้ในงานของ garbage collection, thread ที่ใช้ในงานของ object finalization และ thread ที่ใช้ในงานอื่น ๆ ที่เกี่ยวข้อง

ในบางครั้ง thread จะถูกเรียกว่า lightweight process สาเหตุก็เนื่องมาจากว่า thread มี stack, program counter และ local variable ที่เป็นของตัวเอง แต่ใช้ทรัพยากรอื่นร่วมกันกับ thread ตัวอื่นที่มีอยู่ใน process เดียวกัน เช่น memory, file handlers และ pre-process state อื่น ๆ



หนึ่งในสาเหตุหลัก ๆ ที่ต้องมี thread ในโปรแกรมก็เพื่อให้การแสดง UI เป็นไปได้อย่างรวดเร็ว เช่นถ้ามีโปรแกรมที่ใช้เวลาในการประมวลผลใน CPU มาก ๆ ก็จะทำให้การสนองตอบต่อ I/O ของผู้ใช้ช้าลง การนำเอา thread เข้ามาช่วยจะทำให้ดูเหมือนกับว่าเราทำงานสองอย่างในเวลาเดียวกัน แต่ที่จริงแล้ว ระบบทำการสลับการใช้ CPU ระหว่างโปรแกรมต่าง ๆ ด้วยการโยนการควบคุมไปให้ controller ที่เกี่ยวข้องทำงานแทน เมื่อสิ้นสุดก็กลับมาทำงานเดิมต่อ (หรือทำงานอย่างอื่นต่อไป)

Thread เบื้องต้น

การสร้าง thread ที่ง่ายที่สุดมีอยู่สองวิธี คือ 1). สร้าง thread ด้วยการรับคุณสมบัติจาก java.lang.Thread ซึ่งเป็น class ที่มีเครื่องมือสำหรับการสร้างและประมวลผล thread และ method ที่สำคัญที่เราต้องเขียน (override) ก็คือ run() และ method run() ตัวนี้แหละที่จะได้รับการประมวลผล ”simultaneously” กับ thread ตัวอื่นที่มีอยู่ในโปรแกรม และ 2). สร้าง thread จาก interface Runnable โดยเราต้องทำการเขียน method run() ให้ทำงานต่าง ๆ เอง (interface Runnable มีเพียงแค่ abstract method run() เท่านั้น)

ช่วงชีวิตของ thread

เมื่อพูดถึง thread ของ Java โปรแกรมสิ่งสองสิ่งที่เกี่ยวข้องกันที่เราต้องคำนึงถึงก็คือ thread ที่ทำงานให้เรา (working thread) และ thread object ที่เป็นตัวแทนของ thread ซึ่งในทางปฏิบัตินั้น working thread จะถูกสร้างโดยระบบปฏิบัติการ ส่วน thread object จะถูกสร้างโดย JVM

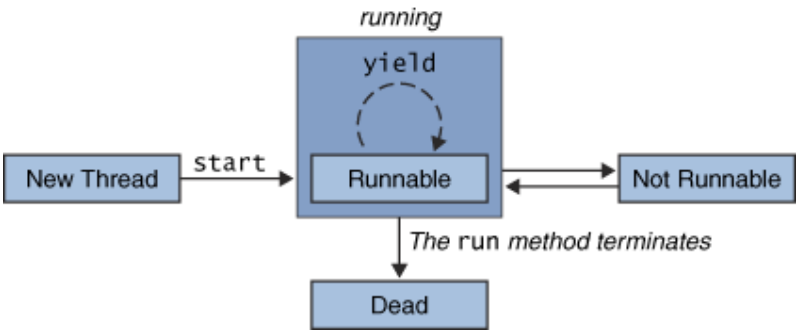
การสร้าง thread และการเริ่มการทำงานของ thread (start) นั้นต่างกัน thread จะยังไม่เริ่มการทำงานจนกว่า thread ตัวอื่นเรียก method start() จาก thread object ที่ถูกสร้างขึ้น เช่น สมมติว่าเรามีการประกาศการใช้ thread ดังนี้

```
CalculatePrimes calPrimes = new CalculatePrimes();
```

ประโยคที่เห็นด้านบนนี้สร้าง thread ชื่อ calPrimes จาก class CalculatePrimes ที่มีการ extends Thread และ thread ตัวนี้มีชีวิตเกิดขึ้น แต่ยังไม่มีการทำงาน จนกว่าจะมีการเรียกใช้ start() เช่น ถ้าเราต้องการให้ thread ทำงานเราก็เรียก

```
calPrimes.start();
```

method start() ก็จะไปเรียก method run() เพื่อทำงานต่าง ๆ ที่เราต้องการ (ชุดคำสั่งที่มีอยู่ใน run()) ช่วงชีวิตของ thread จะสิ้นสุดลงเมื่อ method run() สิ้นสุดการทำงาน



ภาพ (ยังไม่ละเอียดเท่าใด) ด้านบนแสดงช่วงชีวิตของ thread ว่าสามารถอยู่ในสถานะใดได้บ้าง thread จะอยู่ใน dead state เมื่อ method run() จบการทำงาน ส่วนที่เรียกว่า Not Runnable จะเกิดขึ้นเมื่อ method sleep() ถูกเรียก หรือ thread เรียก method wait() เพื่อรอเหตุการณ์อื่น หรือ thread ทำการ blocking I/O

โปรแกรมตัวอย่างต่อไปนี้แสดงการสร้าง thread อย่างง่าย โดยกำหนดให้มี class อยู่สอง class คือ SimpleThread และ TwoThreadsTest

```
public class SimpleThread extends Thread {

   public SimpleThread(String str) {
      super(str);
   }

   public void run() {
      for(int i = 0; i < 10; i++) {
         System.out.printf("%d %s%n", i, getName());
         try {
            sleep((long)(Math.random() * 1000));
         }
         catch(InterruptedException e) {}
      }
      System.out.printf("DONE! %s%n", getName());
   }
}
```

Method ตัวแรกใน SimpleThread เป็น constructor ที่มี String เป็น argument และเราทำการเรียกใช้ constructor ของ super class ของ Thread ซึ่งทำการกำหนดชื่อของ thread ด้วย String ที่ส่งไปให้ ส่วน method ตัวที่สองที่ถือว่าเป็นหัวใจสำคัญของ thread ก็คือ method run() เป็นที่ที่ thread ทำงานต่าง ๆ โดยภายใน run() เรามี for/loop ที่กำหนดให้มีการประมวลผล 10 ครั้ง ในแต่ละครั้งเราจะแสดงตัวเลข และชื่อของ thread หลังจากนั้น เราก็จะเรียกใช้ sleep() เพื่อหยุดการทำงานเป็นจำนวนที่มาจากการสุ่ม โดยให้ค่าสูงสุดที่เป็นไปได้เท่ากับ หนึ่งวินาที คำสั่งสุดท้ายหลังออกจาก for/loop ก็เป็นการแสดงค่า “DONE!” ตามด้วยชื่อของ thread

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

สำหรับโปรแกรมเรียกใช้ thread: TwoThreadsTest นั้นเรามี method main() ที่สร้าง SimpleThread 2 ตัว Bangkok และ Chiang Mai โดยเราจะเรียก method start() ทันทีหลังจากการสร้าง thread ทั้งสอง ซึ่งทำให้ method run() ถูกเรียกขึ้นมาทำงานโดยอัตโนมัติ และเมื่อ compile และ run โปรแกรมผลลัพธ์ที่เราได้ คือ

```
0: Bangkok
0: Chiang Mai
1: Bangkok
1: Chiang Mai
2: Bangkok
3: Bangkok
2: Chiang Mai
3: Chiang Mai
4: Bangkok
4: Chiang Mai
5: Bangkok
6: Bangkok
5: Chiang Mai
6: Chiang Mai
7: Bangkok
7: Chiang Mai
8: Bangkok
8: Chiang Mai
9: Chiang Mai
9: Bangkok
DONE! Chiang Mai
DONE! Bangkok
```

จากผลลัพธ์ที่ได้เราจะเห็นว่ามีการแสดงค่าของ thread สลับกันไปมา สาเหตุก็เพราะว่า thread ทั้งสองตัว run พร้อม ๆ กัน (concurrent execution) ซึ่งทำให้ method run() ของ thread ทั้งสองถูกประมวลผล เราจึงได้รับผลลัพธ์ดังที่เห็น และเมื่อ for/loop ยุติการทำงาน thread ทั้งสองก็ยุติการทำงาน (terminate)

โปรแกรมตัวอย่างต่อไปเป็นการสร้าง thread จาก interface Runnable

```
public class SimpleThread2 implements Runnable {
   private String name;

   public SimpleThread2(String str) {
      name = str;
   }

   public void run() {
      for(int i = 0; i < 10; i++) {
         System.out.printf("%d: %s%n", i, this.name);
         try {
            Thread.sleep((long)(Math.random() * 1000));
         }
         catch(InterruptedException e) {}
      }
      System.out.printf("DONE! %s%n", this.name);
   }
}
```

เราเปลี่ยนจากการ extends Thread มาเป็น implements Runnable ในโปรแกรม SimpleThread2 พร้อมกับกำหนดให้มี class variable: name สำหรับเก็บชื่อของ thread ภายใน method run() เราก็เปลี่ยนการเรียกจาก sleep() ก่อนหน้านี้มาเป็น Thread.sleep() แทน

```
public class TwoThreadsTest2 {
   public static void main(String[] args) {
      Runnable bkk = new SimpleThread2("Bangkok");
      Thread bangkok = new Thread(bkk);
      bangkok.start();

      Runnable cnx = new SimpleThread2("Chiang Mai");
      Thread chiangmai = new Thread(cnx);
      chiangmai.start();
   }
}
```

การสร้าง thread ในตัวอย่างนี้ต้องสร้างจาก Runnable object ซึ่งหลังจากที่สร้างเสร็จเราก็เรียก method start() ของ thread ตัวนี้ เช่น thread ที่ชื่อ bangkok และ chiangmai ผลลัพธ์ที่ได้ก็คล้าย ๆ กับโปรแกรมตัวอย่างที่แสดงให้ดูก่อนหน้านี้