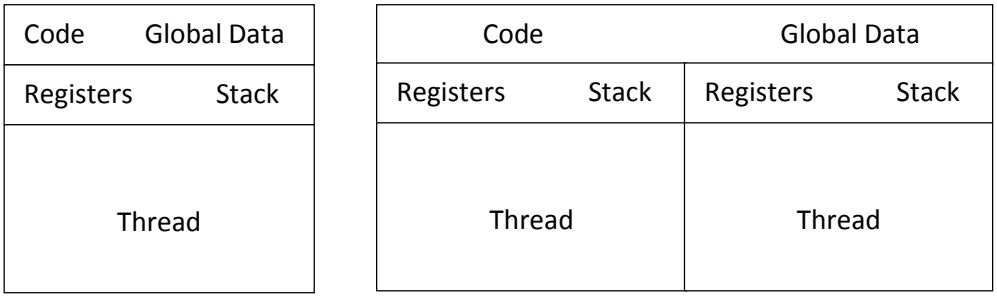
เทรด (Threads)

เนื้อหาประกอบด้วย

- แนวคิดของเทรด และข้อดี
- การใช้งานกับระบบ Multicore Programming
- แบบจำลอง เทรด
- การสร้างเทรด
- การใช้งานอินเตอร์เฟส Runnable
- การจัดการเทรด (Synchronization)

แนวคิดของ เทรด

โปรแกรมหนึ่ง มีงานหนึ่งๆ ที่จะต้องทำ โดยใช้ ทรัพยากร ของ CPU (Code, Global Data, Register, Stack) เพื่องานนั้นให้ลุล่วง ไป โดยโปรแกรมหนึ่งๆ เมื่อทำงาน จะใช้ ส่วนของโปรแกรม (Code) อันเป็นคำสั่งให้ทำงานตามเป้าหมาย ตัวแปรร่วมที่ใช้ร่วมกันได้ (Global Data) หน่วยความจำช่วยคราว (Register) และ ตัวแปรแปรผัน (Stack) ที่เปลี่ยนแปลงขณะทำงาน ซึ่ง ความการทำงานจริงส่วนที่มีการเปลี่ยนแปลงของโปรเซสหนึ่ง จะมีเพียง หน่วยความจำชั่วคราว และตัวแปรแปรผันได้ เท่านั้น ดังนั้น เพื่อให้สามารถใช้ CPU ให้มีประสิทธิภาพมากขึ้น โปรแกรมหนึ่งๆ นั้น สามารถแบ่งเป็นโปรเซสย่อย หรือการทำงานย่อยๆ ได้ ในการทำงานแบบเดิมๆ โดยใช้ทรัยพากร ของ CPU เดิม ซึ่งคือ โปรแกรม และตัวแปรร่วม แต่แบ่ง หน่วยความจำชั่วคราว และตัวแปรแปรผันได้ ที่ทำงานแยกกัน



รูป 1 เทรดเดียว และ มัลติเทรด

การทำให้ทำงานเดิม (Code, Global Data) ได้พร้อมๆ กัน เรียกการทำงานแบบนี้ว่า มัลติเทรด (Multi-thread) จะช่วยโปรแกรมหนึ่งทำงานที่คล้ายกัน ทำงานแยกกันเฉพาะส่วนที่ต่างกันได้ ยกตัวอย่างเช่น Web server รับคำขอใช้บริการ จาก Browser ที่มาจาก Client หากโปรแกรมของ Web Server ไม่สามารถทำทงานแบบมัลติเทรด ได้หมายความ Web Server รองรับการทำงานที่

หนึ่งตามคำขอ ทำให้ผู้ร้องขอบริการอื่นๆ ต้องรอจนกว่า คำขอแรกจะทำงานเสร็จ หรือแม้ว่า Web Sever สามารถรับคำขอได้ใหม่ ก็ต้องสร้างโปรเซสใหม่ ซึ่งจะใช้เวลาและทรัพยากรของระบบมากขึ้นไปด้วย เปรียบเทียบกับ Web Server ที่รองรับการทำงานแบบ มัลติเทรด สามารถรองรับคำขอได้พร้อมๆ กันหลายคำขอ ด้วยการใช้การทำงานโปรเซสเดิม แต่แบ่งเป็นเทรดย่อยๆ จาก รูปแบบการขอเดิม ซึ่งใช้ใช้ทรัพยากรแบบเดิม แต่ต่างกันในส่วนรายละเอียดของตัวแปรและพื้นที่หน่วยความจำที่ต้องใช้ นี้คือประโยชน์ของ การทำงานแบบมัลติเทรด

ข้อดีของของมัลติเทรด

ข้อดีของการเขียนโปรแกรมมัลติเทรด สามารถแยกย่อยได้ สี่ประเด็นหลักคือ

1. ตอบสนองได้รวดเร็ว การใช้การทำงานแบบมัลติเทรด จะช่วยให้โปรแกรมทำงานได้รวดเร็วซึ่งรวมทำให้ตอบสนองได้รวดเร็วขึ้นด้วย ถึงแม้โปรแกรมจะบางส่วนของโปรแกรมถูกขัดขวางการทำงานให้มีการดำเนินการที่ยาวนาน เช่น มัลติเทรดในเว็บเซิร์บเวอร์ อาจอนุญาตให้ผู้ใช้งานดำเนินการกับเทรดหนึ่ง ในการฟังเพลง ในขณะที่อีกเทรดหนึ่งทำการอ่านภาพ
2. ส่งเสริมการใช้ทรัพยากรร่วมกัน โปรแกรมที่ทำงานหนึ่งอาจมีการร่วมใช้ทรัพยากรร่วมกัน ในด้านของหน่วยความจำโปรแกรมสั่งการ โดยที่อาจควบคุมโดยการเขียนโปรแกรม ถึงแม้จะมีค่าปริยายที่ระบบปฏิบัติการทำงานให้อยู่แล้วก็ตาม
3. ประหยัดทรัพยากร การใช้ทรัพยากรร่วมกันถือเป็นการประหยัดทรัพยากร ที่ทำให้ระบบมีทรัพยากรเหลือมากขึ้นที่จะนำไปใช้กับการทำงานอื่นๆ ได้อีก รวมถึงทำให้ระบบทำงานได้เร็วขึ้น เช่น การทำงานหนึ่ง ต้องใช้ทรัพยากรจำนวนหนึ่ง ถ้าต้องให้ ซีพียู สร้างโปรเซสใหม่เพื่อรองรับการทำงานใหม่ ระบบจะสลับการทำงานไปยังอีกโปรเซสหนึ่ง จะทำให้ระบบเสียเวลาในการสลับไปทำงาน (Context-switch) เพื่อใช้ทรัพยากรใหม่
4. ขยายการทำงานได้ดี เนื่องด้วยการใช้มัลติเทรด ในระบบที่มีซีพียู หลายตัว หรือ คอร์ (Core) ทำให้แต่ละเทรด แบ่งไปทำงานที่อีกซีพียูอื่นๆ โดยทำงานในลักษณะคู่ขนานได้ แต่ถ้าระบบที่ใช้แบบเทรดเดียว จะไม่สามารถทำงานได้หลายซีพียู

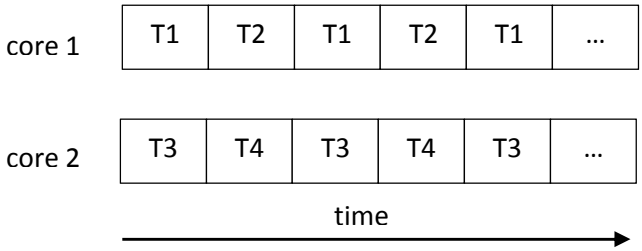
สนับสนุนการการโปรแกรมแบบ ระบบมัลติคอร์ (Multicore Programming)

ในยุคที่มีใช้ตัวประมวลผล ที่มีหลายคอร์ บนชิปเพียงตัวเดียว โดยแต่ละคอร์ทำหน้าที่เหมือนเป็นตัวประมวลผลที่แยกอิสระ การเขียนโปรแกรมแบบมัลติเทรด ใช้ตัวประมวลผที่มีหลายคอร์ ได้มีประสิทธิภาพได้ยิ่งขึ้น และส่งเสริมการทำงานพร้อมกันได้ ยกตัวอย่าง ที่มีโปรแกรมหนึ่งทำแบบมีสี่เทรด บนตัวประมวลผลที่มีสองคอร์ (ดูรูป 2 ประกอบ) โปรเซส จะแบ่งให้เทรดทำงานระหว่างคอร์ร่วมกันได้ ในลักษณะคู่ขนาด

อย่างไรก็ตามในยุคการใช้งานมัลติคอร์ ระบบปฏิบัติการมีการออกแบบการทำงานให้มีตารางการทำงานเพื่อสนับสนุนการเขียนโปรแกรมแบบมัลติคอร์ และนักเขียนโปรแกรมแบบมัลติคอร์ออกแบบโปรแกรมจะต้องคำนึงถึง 5 เรื่องต่อไปนี้

1. การจัดแบ่งกิจกรรมงาน เป็นเกี่ยวกับการทดสอบโปรแกรมเพื่อหาว่า มีส่วนงานได้บ้างที่สามารถแบ่งแยกกันได้ ส่วนงานใดทำงานคู่ขนานกันได้ในแต่ละคอร์

2. การสมดุลงาน ในการแบ่งกิจกรรมงาน แล้วปล่อยให้แต่ละเทรดทำงานในแต่ละคอร์ ถ้าให้แต่ละเทรดทำงานใช้เวลาที่เสร็จใกล้เคียงกันก็จะดี แต่ถ้าปล่อยให้เทรดหนึ่งใช้เวลา นานเกินไป การทำงานในคอร์นั้นก็เสียทรัพยากรสำคัญนั้นไป
3. การแบ่งข้อมูล เมื่อให้เทรดทำงาน การใช้งานข้อมูลที่ทำงานในแต่ละเทรดควรจะแบ่งให้ชัดเจนเพื่อให้ข้อมูลนั้นทำงานได้ในแต่ละคอร์ที่เทรดนั้นทำงานอยู่
4. การจัดการข้อมูลที่ขึ้นแก่กัน การใช้งานข้อมูลที่ใช้ร่วมกัน จะต้องให้โปรแกรมที่ทำงานในลักษณะคู่ขนาน ทำงานได้โดยไม่เกิดการแย่งชิงข้อมูล หรือมีการรอข้อมูลอีกเทรดหนึ่ง
5. การทดสอบ และการแก้ไขข้อบกพร่อง เมื่อโปรแกรมทำงานคู่ขนาน ปัญหายากอย่างหนึ่งที่เกิดขึ้นคือ การทดสอบ และการแก้ไขข้อบกพร่อง ซึ่งมีมากกว่า การทำงานแบบเทรดเดี่ยว นักเขียนโปรแกรมจะต้องใช้ความพยายามสูงขึ้นในการแก้ปัญหา



รูป 2 การทำงานคู่ขนาดของ ระบบมัลติคอร์

แนวทางการสร้างเทรด

การสร้างเทรดด้วยจาวา ทำได้สองวิธีคือ

1. กำหนดคลาสลูก ให้สืบทอดมากจากจากคลาส Thread ซึ่งมี เมธอด run( ) ที่สืบทอดมา ให้เรียกใช้ได้
2. สร้างคลาสลูกให้สืบทอดจากอินเตอร์เฟส Runnable ทำให้ต้องเขียนแต่งเมธอด run( ) และต่อมาสร้างออปเจค เทรด

สร้างเทรด จากคลาส Thread

ตัวอย่างคลาส FromThread เป็นคลาสที่สร้างขึ้นจาก คลาส Thread คลาส FromThread แต่มีการเขียนการทำงานของ run( ) ในที่นี้ ให้เมธอด run( ) พิมพ์ค่าชื่อ และสกุล เสมอ จนกว่าเทรด จะหยุดทำงาน

```
public class FromThread extends Thread{
    private String fname, lname;
    private long delay;
    public String getFname(){return this.fname;}
    public String getLname(){return this.lname;}
    public FromThread(String fname, String lanme, long delay){
        this.fname = fname;
        this.lname = lanme;
        this.delay = delay;
        setDaemon(true);
    }
```

```
    @Override
    public void run(){
        try{
            while(true){
                System.out.print(fname);
                System.out.println("\t"+lname);
               sleep(delay);
            }
        }
        catch(InterruptedException e){
            System.out.println( "Exeption: " + e);
        }
        return;
    }
}                                                                                    //Code 1.
```

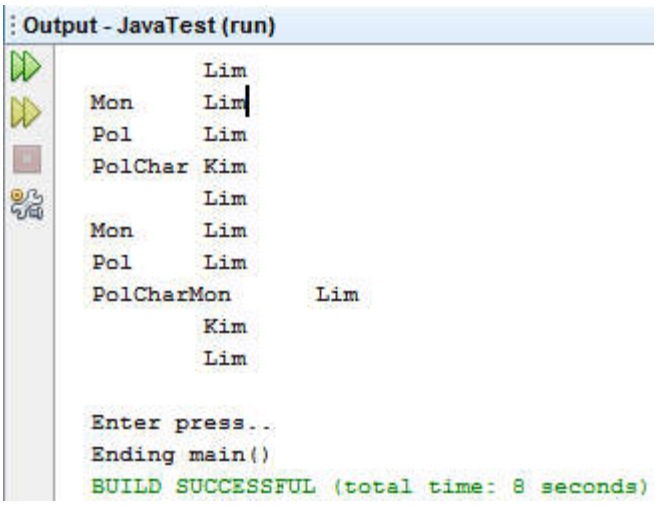
ในเมธอดหลัก (main( )) จะเป็นการสร้างวัตถุ จากคลาส FromThread โดยวัตถุทั้งสามตัว (thread1, thread2, thread3) จะทำงานไปเรื่อยๆ แต่โปรแกรมนี้มีการรอรับข้อมูลจากแป้นพิมพ์ ในคีย์ใดๆ (System.in.read( )) ถ้ายังไม่กดอะไรลงไป เทรดก็จะทำงานไปเรื่อยๆ แต่ถ้ากดคีย์ด้วยคีย์ Enter แล้วจะทำในบรรทัดต่อไป (กด Enter บนคีย์บอร์ด ในหน้า output) จนจบคำสั่งของเมธอด Main ซึ่งเป็นการจบการทำงานของเมธอด main( ) นี้ ทำให้ วัตถุในเมธอด จบการทำงานไปด้วย เนื่องด้วยคลาส FromThread มีการกำหนด Daemon(true) ซึ่งหมายความว่า เมื่อเมธอดหลักที่ทำงานอยู่ จบการทำงานไป ออปเจคของเทรด ที่อาศัยอยู่ ก็จะจบการทำงานไปด้วย

```
public static void main(String[] agrs){
        FromThread thread1 = new FromThread("Pol", "Lim", 200L);
        FromThread thread2 = new FromThread("Mon", "Lim", 300L);
        FromThread thread3 = new FromThread("Char", "Kim", 400L);

        System.out.println("Press enter to stop");

        thread1.start();
        thread2.start();
        thread3.start();

        try{
            System.in.read();
            System.out.println("Enter press..");
        }
        catch(IOException e){
            System.out.println(e);
        }
        System.out.println("Ending main()");
}                                                                                    //Code 2.
```



รูป 3 ผลลัพธ์ของการรัน Code 1

Daemon

การใช้งานเทรด บางครั้งสามารถสร้างเทรดย่อย ที่เรียกว่า เดมอน (Daemon) ซึ่งจะทำงานเป็นพื้นหลังของเทรดหลัก (เขียนเทรดให้มีเมธอดว่า setDaemon(true)) ดังนั้นแล้ว เมื่อเทรดหลักจบการทำงาน เดมอนเทรด นี้ก็จบการทำงานตามไปด้วย การที่เทรดใดไม่มีเดมอนเทรด แสดงว่าเมื่อเทรดหลักจบการทำงานไปแล้ว เทรดที่ไม่มีเดมอน จะยังคงทำงานต่อไปอีก จนกว่า ที่ต้องกำหนดหรือเขียนให้ชัดว่าหยุดโดยเฉพาะ หรือทำลายเทรดโดยเฉพาะ

สมมุติว่า โปรเซสหนึ่ง ชื่อ Main จะเทรดย่อยๆ ว่า A, B, C กำหนดให้ A, B และ C มี เดมอนเทรด เมื่อจบการทำงานของ Main เทรด A, B และ C ก็จะจบการทำงานไปด้วย ดังตัวอย่างที่ผ่านมา ในกรณีที่ เทรด C ไม่ได้ถูกกำหนดเดมอน เมื่อจบการทำงานของ เมธอดหลัก จะทำให้ เทรด A, B, C ทุกตัวยังคงทำงานอยู่

ค่าปริยายของเทรด ไม่กำหนดเดมอนไว้ การทำให้เป็นเดมอน ก็กำหนดให้เป็น true และต้องกำหนดเป็นเดมอน ก่อนเรียกเมธอด start( )

การทดลอง ที่ 1

1. สร้างคลาส ที่ไม่กำหนดเดมอน จากตัวอย่างเดิม คลาส FromThread ลบการกำหนด เดมอน ในคอนสรัทเตอร์
2. สร้างวัตถุ ที่สร้างขึ้นจากคลาส FromThread ขึ้นมาสามตัว ตามตัวอย่างก่อนหน้านี้ แต่ กำหนด เดมอน เฉพาะวัตถุที่ 1 และ 2 ส่วนวัตถุที่ 3 ไม่ต้องกำหนดเดมอน เช่น

```
thread1.setDaemon(true);
thread1.start( )
```

3. รันโปรแกรม ดูผล

ดังเห็นได้จากการทดลอง การกำหนดเดมอน บางตัว ไม่เป็นผล ยังมีอีกวิธีหนึ่ง ที่กำหนด ให้เทรดหยุดทำงานเฉพาะเทรด ที่กำหนดเท่านั้นคือ การใช้เมธอด interrupt( ) เช่น thread1.interrupt( ) การกำหนดเช่นนี้จะทำให้ thread1 หยุดทำงาน ในขณะที่เทรดอื่นๆ ยังคงทำงานต่อไป

```
    try{
                System.in.read();
                t1.interrupt(); //end thread 1
                System.out.println("Enter press..");
                System.in.read();
                t2.interrupt(); //end thread 2
                System.in.read();
          }
          catch(IOException e){
                System.out.println(e);
          }
          System.out.println("Ending main()");
          return; //end all thread
    }                                                                    //Code 3.
```

สร้างเทรด จากอินเตอร์เฟส Runnable

สำหรับการสร้างคลาสที่สืบทอดจาก Runnable ที่จำเป็นต้องเขียนการทำงานของ run( ) เพราะใช้การสืบทอดจากอินเทอร์เฟส Runnable

```
public class FromRunnable implements Runnable {
    private String fname, lname;
    private long delay;
    public String getFname(){return this.fname;}
    public String getLname(){return this.lname;}
    public long getDelay(){return this.delay;}
    public FromRunnable(String fname, String lname, long delay){
        this.fname = fname;
        this.lname = lname;
        this.delay = delay;
    }
    @Override
    public void run() {
        try{
            while(true){
                System.out.print(fname);
                System.out.println("\t"+lname);
                Thread.sleep(delay);//give a change the other thread to run
            }
        }
        catch(InterruptedException e){
            System.out.println("Exception:" + e.getMessage());
        }
    }
}                                                                        //Code 4.
```

การสร้างเทรด ที่มาจากการอิมพลีเม้นท์ Runnable จะต้องสร้างคลาส Thread และให้ ออปเจค Runanble เป็นตัวแปรในคอนสตรักเตอร์ การเรียกให้เทรด ทำงานใช้การทำงานแบบ สร้างคลาสจากคลาส Thread โดยการเรียก start( ) โดยไม่จำเป็นจำจะต้องเรียก run( ) ดูตัวอย่างต่อไป ซึ่งได้ผลการทำงาน เหมือนกับ รูปที่ 3

```
public static void main(String[] args){
        FromRunnable run1 = new FromRunnable("Pol", "Lim", 200L);
        FromRunnable run2 = new FromRunnable("Mon", "Lim", 300L);

        Thread thread1 = new Thread(run1);
        thread1.setDaemon(true);
        Thread thread2 = new Thread(run2);
        thread2.setDaemon(true);
        thread1.start();
        thread2.start();

        try{
            System.in.read();
            System.out.println("Enter press..");
        }
        catch (IOException e){
            System.out.println(e);
        }
        System.out.println("Ending main()");
        return;
}                                                                    //Code 5.
```

ซิงโครไนเซชัน (Synchronization)

สมมุติเหตุการณ์หนึ่ง (ซึ่งมักใช้สมมุติกัน) เมื่อ มีลูกค้ารายหนึ่งมาฝากเงินที่ธนาคาร และมีการถอนเงินผ่านตู้ ATM มีลำดับเวลาดังนี้

เวลา: 1 ที่ธนาคาร ลูกค้าต้องการฝากเงิน พนักงาน ดูเงินในบัญชีก่อนฝากมีอยู่ 1000 บาท

เวลา: 2 ที่ตู้เอทีเอ็ม ลูกค้ารายนี้ให้บัตร ATM ให้ญาติไปถอดเงิน ดูเงินในบัญชี ก็มียอดอยู่ที่ 1000 บาท

เวลา: 3 ที่ธนาคาร ลูกค้าฝากเงินไป 500 บาท

เวลา: 4 ที่ตู้เอทีเอ็ม มีการถอนเงินไป 1000 บาท

เวลา: 5 ที่ธนาคาร พนังงาน ปรับปรุงยอดเงิน เป็น 1500 บาท

เวลา: 6 ที่ตู้เอทีเอ็ม ยอดเงินที่ตู้เอทีเอ็ม เหลือเงิน 0 บาท

จากเหตุการณ์สมมุติ บัญชีเดียวกัน แต่ไม่ทำงานร่วมกัน ผลเลย ไม่เป็นตามคาดหวัง เหตุการณ์แบบนี้จะไม่เกิด หากว่า มีการทำงานร่วมกัน ที่เรียกว่า ซิงโครไนเซชัน ของกระบวนการทำงานต่างๆ หรือ เทรด ต่างๆ ให้ทำงานร่วมกันได้

เป้าหมายของการทำซิงโครไนท์ ก็เพื่อต้องการจัดการเทรด ให้เข้าใช้ทรัพย์กรที่มีอยู่ที่เดียว ให้ทำงานได้ทีละหนึ่งเทรด การทำซิงโครไนท์ ทำได้สองทาง คือ

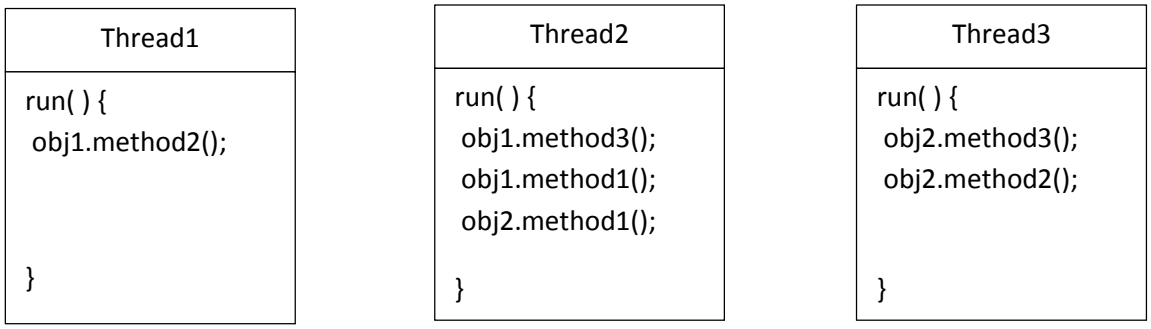
1. การจัดการโปรแกรมในระดับ เมธอด ซึ่งเป็นการจัดการการทำงานร่วมกันของเมธอด
2. การจัดการโปรแกรมในระดับ กลุ่มปีกกา ซึ่งเป็นการจัดการทำงานร่วมกันในกลุ่มปีกกา

การจัดการเทรดในการทำงานร่วมกัน

ในการจัดการการทำงานร่วมกันในระดับเมธอด การทำให้เมธอหนึ่งให้ทำงานให้เสร็จก่อนที่จะเริ่มทำเมธอดต่อไป ภายในคลาสหนึ่งนั้น ใช้คำหลักคือ synchronized ยกตัวอย่างเช่น

```
class MyClass{
        synchronized public void mehtod1(){}
        synchronized public void method2(){}
        public void method3(){}
}                                                                    //Code 6.
```

เมื่อคลาสนี้ถูกสร้างเป็นวัตถุ เมธอดที่คำสำคัญ synchronized จะทำงานได้ทีละเมธอดเท่านั้น หรือไม่สามารถทำงานพร้อมกันได้ (method1 กับ method2) แต่เมธอดที่ไม่มี synchronized จะทำงานพร้อมกันได้ เช่น method3 ดูรูปต่อไปนี้อธิบายการทำงานของวัตถุ สองวัตถุ (obj1, obj2) ที่สร้างจากคลาส MyClass กับเทรด ต่างๆ



รูป 4. เทรด 3 ตัว ของออปเจค obj1 และ obj2

ลำดับการทำงานจากรูปข้างต้น

1. Thread1

   obj1.method2 ทำงานเริ่มทำงาน
2. Thread2

   obj1.method3 ทำงานได้พร้อมกับ Thread1

   obj1.method1 ยังทำงานไม่ได้ต้องรอให้ obj1.method2 หยุดทำงานก่อน

   obj2.method1 ทำงานได้เพราะเป็นออปเจคใหม่

3. Thread3

   obj2.method2 ทำงานได้ทันที

   obj2.method2 ยังทำงานไม่ได้ต้องรอให้ obj2.method1 ทำงานเสร็จก่อน

แบบฝึกหัด

1. ให้มีเมตริกสองตัว คือ A กับ B โดยเมตริก A มีขนาด M แถว และ N คอลัมน์ (M x K) เมตริก B มีขนาด K แถว และ N คอลัมน์ (K x N) การคูณกันของเมตริก (Matrix Product) เป็นนำผลบวกกันของการคูณระหว่าง แถว และหลัก ดูรูปต่อไปนี้เป็นตัวอย่าง

รูปตัวอย่าง เมตริกขนาด 4 x 3 คูณกับ เมตริก ขนาด 3 x 2 จะได้เมตริกขนาด 4 x 2

ที่มาของรูป : https://en.wikipedia.org/wiki/Matrix_multiplication

ให้ C ขนาด (M x N) เป็นผลคูณของเมตริก A ขนาด (M x K) และ B ขนาด (K x N)

$$C(i,j) = \sum_{n=1}^{K} A(i,n) \times B(n,j)$$

อ้างอิง

- Horton, Ivor, “Beginning Java 7 Edition”, John Wiley & Sons, Inc., 2011.
- Silberschatz, Abraham, Galvin, Peter Baer, and Gagne, Greg, “Operating System Concepts” 9 ed., John Wiley & Sons, Inc., 2012.
- https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html (June 9, 2016)