ใบงาน 10

วัตถุประสงค์ เพื่อศึกษาหลักการการใช้ java Semaphore และ กลไกอื่นๆที่เทียบเท่า

กิจกรรมที่ 1 แนะนำ java thread แบบ inner class

```
public class Q0ChildThreadDriver {
  public static void main(String[] args) {
    int nThread = 100000; // hundred thousands
    SharedNum1 sn = new SharedNum1();
    Thread[] thr = new Thread[nThread];
    for (int i = 0; i < nThread; i++) {
      thr[i] = new ChildSimple(sn);
      thr[i].start();
    }
    for (int i = 0; i < nThread; i++) {
      try {  thr[i].join();
      } catch (InterruptedException ie) { }
    }
    if (sn.getVal() < nThread) {
       System.out.printf("v0 val = %d Not ");
    System.out.printf("100,000\n", sn.getVal());
    } else {
       System.out.printf("v0 good job! %d "
       System.out.printf("\n", sn.getVal());
    }
}

class ChildSimple extends Thread {
    SharedNum1 resource;

    ChildSimple(SharedNum1 ref) {
        resource = ref;
    }

    public void run() {
        resource.increment();
    }
}
```

```
public class Q1InnerAndNotSyncDriver {
  public static void main(String[] args) {
    int nThread = 100000;
    SharedNum1 sn = new SharedNum1();
    Thread[] thr = new Thread[nThread];
    for (int i = 0; i < nThread; i++) {
      thr[i] = new Thread(new Runnable() {
                @Override
                public void run() {
                   sn.increment();
                }
              });
            thr[i].start(); /* A */
    }
    for (int i = 0; i < nThread; i++) {
      try {   thr[i].join();
      } catch (InterruptedException ie) { }
    }
    if (sn.getVal() < nThread) {
        System.out.printf("v1 val = %d Not ");
   System.out.printf("100,000\n", sn.getVal());
    } else {
       System.out.printf("v1 good job! %d \n");
        System.out.printf("\n", sn.getVal());
    }
}
```

```
class SharedNum1 {
    private int val = 0;

    void increment() {    val++;  }

    int getVal() {    return val;  }
}
```

1.1 การเขียนแบบ inner class ทำให้เราไม่ต้องตั้งชื่อ class (สร้าง class) เราเพียง implement ส่วนที่ต้อง implement

1.2 หาก จะให้ thread เป็นคนเรียก .increment ย่อมต้องสร้าง constructor เพื่อส่ง reference ให้ thread นั้นเห็น shared variable (แต่แบบ anonymous เห็น sn เพราะต่างอยู่ใน main())

1.3 จะเห็นว่าทั้ง 2 code ยังไม่มีกลไกสำหรับการจัดจังหวะ (synchronization)

หมายเหตุ statement cnt++; ก็เกิด race condition ได้ (ก่อนหน้านี้ เราใช้ tmp = cnt; tmp = tmp + 1; cnt = tmp;

เพื่อให้เกิด data inconsistency ง่ายขึ้น (ค่าเลยต่ำกว่า 2,000,000 มาก เพราะหลายรอบจะได้ค่า tmp เก่า)

กิจกรรมที่ 2 semaphore, synchronized method, synchronized block

เบื้องต้นเราสามารถใช้กลไกทั้ง 3 ที่ java ให้มา

```
public class Q2SemaphoreDriver {
  public static void main(String[] args) {
    int nThread = 100000; // hundred thousands
    SharedNum2 sn = new SharedNum2(); int v = 2;
  //SharedNum2 sn = new SharedNum3(); int v = 3;
  //SharedNum2 sn = new SharedNum4(); int v = 4;
  Thread[] thr = new Thread[nThread];
    for (int i = 0; i < nThread; i++) {
      thr[i] = new Thread(new Runnable() {
                  @Override
                  public void run() {
                      sn.increment();
                  }
                });
              thr[i].start(); /* A */
    }
    for (int i = 0; i < nThread; i++) {
      try { thr[i].join();
      } catch (InterruptedException ie) {}
    }
    if (sn.getVal() < nThread){
        System.out.printf("v%d val = %d Not ");
   System.out.printf("100,000\n",v, sn.getVal());
    } else {
       System.out.printf("v%d good job! %d \n");
       System.out.printf("\n",v, sn.getVal());
    }
}
```

```
import java.util.concurrent.Semaphore;

class SharedNum2 { // semaphore
  private int val = 0;
  private Semaphore mutex;

  SharedNum2() {
    mutex = new Semaphore(1);
    val = 0;
  }

  void increment() {
    try {
        mutex.acquire();
        val++;
        mutex.release();
    } catch (InterruptedException ie) { }
  }

  int getVal() {  return val;  }
}
```

```
class SharedNum3 { // synchronized method
  private int val = 0;

  // SharedNum3() { val = 0; }
  synchronized void increment() {
        val++;
  }

  int getVal() { return val; }
}
```

```
class SharedNum4 { // synchronized block

    private int val = 0;

    void increment() {
        synchronized (this) {  val++;  }
    }

    int getVal() { return val; }
}
```

(เปลี่ยน class ของ sn ในแต่ละ รอบ)

2.1 ใช้ Semaphore ของ java เพื่อควบคุมการเข้าถึง val โดยที่ .acquire() และ release() ต้องอยู่ใน try – catch block (การ acquire() / release() ผิดจังหวะ อาจทำให้เกิดสภาวะ deadlock)

2.2 ใช้ synchronized (คีย์เวิร์ด) เพื่อกำกับทั้ง method ให้สามารถเรียก method นั้นๆ ได้ทีละ thread

2.3 ใช้ synchronized block เพื่อกำกับเฉพาะ critical section เพื่อเพิ่มโอกาสให้ thread อื่นได้เข้าสู่ critical section ทั้งนี้เนื่องจาก ระบบไม่ทราบว่าล็อคนี้เป็น object ใด วิธีที่สะดวกที่สุดคือใช้ instance sn นั้นเป็นล็อค

กิจกรรมที่ 3 Java Object class's wait() and notifyAll()

```
public class Q5WaitNotifyDriver {
  public static void main(String[] args) {
    SharedNum5 sn = new SharedNum5();
    // unnamed thread receiver
    /* Thread recver = */ (new Thread(new Runnable() {
      @Override
      public void run() {
        System.out.println("got " + sn.getVal());
      }
    })).start();
    try {
      Thread.sleep(2); // main
    } catch (InterruptedException ie) {  }
    // unnamed thread sender
    /* Thread sender = */ (new Thread(new Runnable() {
      @Override
      public void run() {
        sn.setVal(2021);
      }
    })).start();
    // since no reference do not care for join
    System.out.println("from main");
  }
} // class
```

```
class SharedNum5 {
  private int val = 0;
  Object lock;

  SharedNum5() {
    val = 0;

  }
  synchronized int getVal() {




    return val;
  } // getVal

  synchronized void setVal(int x) {


  } // setVal
}
```

3.1 ศึกษา Java Object class's wait() และ notifyAll()

3.2 เขียน SharedNum5 ให้ receiver ออกจาก wait() ใน getVal() เมื่อ setVal() ส่ง notifyAll()

กิจกรรมที่ 4 ศึกษา Dining Philosopher Problem

Monitor Concept

```
10  monitor DP {
20    enum {THINKING, HUNGRY,
          EATING} state[5];
30    condition self[5];
40    void pickup (int i) {
50      state[i] = HUNGRY;
60      test(i);
70      if (state[i] != EATING) self[i].wait;
80    }
90    void putdown (int i) {
100     state[i] = THINKING;
110     test((i+4) % 5); //notify right person
120     test((i+1) % 5); //notify left person
130   }
140   void test (int i) {
150     if ( (state[(i+4)%5] != EATING) &&
160          (state[i] == HUNGRY) &&
170          (state[(i+1)%5] != EATING) ) {
180       state[i] = EATING;
190       self[i].signal();
200     }
210   }
220   initialization() {
230     for (int i = 0; i < 5; i++)
240       state[i] = THINKING;
250   }
260 }
```

```
Semaphore  chopstick[5];

do{
   wait(chopstick[i]);  // left chopstick
   wait(chopstick[(i+1)%5 ]); // right chopstick
      // eat
   signal(chopstick[i]);  // left chopstick
   signal(chopstick[(i+1)%5 ]); // right chopstick
      // think
   } while(TRUE);
```

http://lass.cs.umass.edu/~shenoy/courses/fall12/lectures/Lec10.pdf

ใช้ semaphore อาจเกิด deadlock เพราะไม่ทำ atomicity ของตะเกียบสองข้าง หากทุกคนหยิบตะเกียบซ้าย

วิธีอ่าน pseudo นี้ที่ง่ายที่สุดคือเริ่มต้นทุกคนมีสถานะ THINKING (250) (ด้วยไวยากรณ์ enum)

สิ่งที่ pseudo นี้ขาดคือการโยงว่าถ้าฉันจะ HUNGRY ฉันจะเรียก pickup() ซึ่งจะเปลี่ยนสถานะตัวเอง (จาก THINKING เป็น HUNGRY (60) (ไม่ต้องเช็คอะไร ก็ฉันจะหิวอ่ะ) ความจริงคือเรียกจาก main()

จะหยิบตะเกียบใน pickup() ต้อง test()

ใน test() จะมี atomicity อยู่กล่าวคือ จะหยิบตะเกียบได้ คนซ้าย (i+1) ต้องไม่กินอยู่ และ คนขวา (i+4) ต้องไม่กินอยู่ และฉันต้อง HUNGRY (160 – 180) 3 เงื่อนไขพร้อมกัน ถึงจะได้กิน (190)

ความคมของ DP คือ ถ้าฉัน test() ไม่ผ่าน(state[i] != EATING) ฉันจะ wait() ที่ (80)

กินเสร็จฉันจะ signal() เผื่อมี self[i] ที่ wait() อยู่แล้ว putdown() (เรียกจาก main()) เพื่อเปลี่ยนสถานะ (กลับ)เป็น THINKING การเรียก test() ใน putdown ให้คนซ้ายและขวา เผื่อคน ซ้าย/ขวา รออยู่ เสมือนไปบรรทัดที่ (70) เพื่อชิงกันเปลี่ยนสถานะให้ได้เป็น EATING (ใน test() หากฉันได้ signal แต่ไม่หิว ฉันก็ไม่กิน

นึกภาพ คนที่ 1 และ 3 ทานเสร็จ คนที่สองที่หิวย่อมรอทั้งสองคนกินเสร็จ

กล่าวมาทั้งปวงเพื่อพิสูจน์ว่า monitor ทำงานได้ (เก่งกว่า ง่ายกว่า semaphore) โปรดศึกษาวิธีการใช้งาน monitor ให้ชำนาญ