ใบงานการทดลองที่ 13

เรื่อง การใช้งาน Inner Class และการใช้งาน Thread

1. จุดประสงค์ทั่วไป

1.1. รู้และเข้าใจการโปรแกรมเชิงวัตถุ การกำหนดวัตถุ การใช้วัตถุ

1.2. รู้และเข้าใจการทำหลายงานพร้อมกัน

2. เครื่องมือและอุปกรณ์

เครื่องคอมพิวเตอร์ 1 เครื่อง ที่ติดตั้งโปรแกรม Eclipse

3. ทฤษฎีการทดลอง

3.1. Nest Class คืออะไร? มีวัตถุประสงค์เพื่ออะไร? อธิบายพร้อมยกตัวอย่างประกอบ

เป็น Class ที่ประกาศภายใน body ของ Class หรือ Interface อื่นๆ จุดประสงค์หลักของการสร้าง Nested Classes คือการ group Class และ Interface ที่เกี่ยวข้องกันให้อยู่ภายใน File เดียวกัน ถึงแม้ว่าการทำ Package ก็ช่วยในเรื่องดังกล่าวแล้วแต่การทำ Nested Classes ทำให้การ group แข็งแรงมากขึ้นอีกขั้น

```
public class OuterClass {

    static int number = 10;

    static class InnerClass {
        void printData(){
            System.out.println(number);
        }
    }
    public static void main(String[] args) {
        OuterClass.InnerClass outerClass = new OuterClass.InnerClass();
        outerClass.printData();
    }
}
```

3.2. จงยกตัวอย่างการสร้าง Inner Class

```
Package james;

abstract class Myclass<T>{
    abstract T add(T num, T num 2);
}
Public class JavaExample {
Public static void main(String[ ] args) {
        MyClass<Integer> obj = new MyClass<>() {
            Integer add(Integer x, Integer y) {
                Return x+y;
            }
        };
        Integer sum = obj.add(100,101)
        System.out.println(sum);
}
}
```

3.3. จงยกตัวอย่างการเรียกใช้งาน Instance ที่มีการเรียกใช้งาน Properties ภายใน Inner Class

```
Public static void main( String[] args ){
    OuterClass outerClass = new OuterClass.InnerClass();
    outerClass.test += 10 ;
   }
```

3.4. จงยกตัวอย่างการเรียกใช้งาน Instance ที่มีการเรียกใช้งาน Method ภายใน Inner Class

```
Public static void main( String[] args ){
    OuterClass outerClass = new OuterClass.InnerClass();
    outerClass.printData();
   }
```

3.5. Thread คืออะไร? มีประโยชน์อย่างไร? อธิบายพร้อมยกตัวอย่างประกอบ

Thread คือระบบของจาวาสำหรับการสนับสนุนการทำงานแบบ multi-tasking แบบที่ในระบบปฏิบัติการก็จะให้โปรแกรมสามารถทำงานพร้อมกันได้ เช่น ฟังเพลงไปด้วยพิมพ์งานไปด้วยก็ได้ นอกจากนี้เธรดยังสามารถทำงานพร้อมกันได้ด้วยเรียกว่า multi-threa

ประโยชน์จาก Thread นั้นโปรแกรมจะต้องเป็นแบบ Multithreading ซึ่งจะมีข้อได้เปรียบ เช่น มีการตอบสนองของโปรแกรมที่ดีกว่า การประมวลผลเร็วกว่า ใช้ทรัพยากรน้อยกว่า การใช้ประโยชน์จากระบบมากกว่า และการทำงานแบบขนาน

3.6. การเริ่มต้นใช้งาน Thread มีขั้นตอนอย่างไรบ้าง?

```java
public class ThreadExample {
    public static void main(String[] args){
        Thread t1 = new Thread(new MyThread());
        t1.start();
    }
}
class MyThread implements Runnable {
    @Override
    public void run() {
        System.out.println("Thread is running...");
    }
}
```

3.7. ระหว่าง Thread และ Runnable มีรูปแบบการใช้งานที่เหมือนหรือแตกต่างกันอย่างไร?

Thread เป็นคลาสในแพ็คเกจ java.lang คลาสเธรดขยายคลาสของ วัตถุ และใช้อินเทอร์เฟซ Runnable คลาส Thread มีตัวสร้างและวิธีการในการสร้างและดำเนินการกับเธรด

Runnable เป็นอินเตอร์เฟสในแพ็คเกจ java.lang การใช้อินเตอร์เฟสที่เรียกใช้งานได้นั้นเราสามารถกำหนดเธรดได้ ส่วนต่อประสานที่ รัน ได้มีวิธีการเดียว รัน () ซึ่งนำมาใช้โดยคลาสที่ใช้ส่วนต่อประสาน Runnable มันเป็นที่ต้องการที่จะใช้อินเตอร์เฟซที่เรียกใช้แทนการขยายชั้นเรียนด้าย เนื่องจากการใช้ Runnable ทำให้โค้ดของคุณเชื่อมโยงกันอย่างหลวม ๆ เนื่องจากโค้ดของเธรดต่างจากคลาสที่กำหนดงาน ให้กับเธรด มันต้องใช้หน่วยความจำน้อยลงและยังช่วยให้ชั้นเรียนที่จะรับช่วงชั้นอื่น ๆ

3.8. สถานะ Deadlock มีลักษณะเป็นอย่างไร? อธิบายพร้อมยกตัวอย่างประกอบ

ซึ่งเป็นสถานการณ์ที่ 2 thread หรือมากกว่าถูกล็อก (LOCKED) ตลอดกาล ซึ่งรอกันและกันให้ทำงานให้เสร็จก่อน ซึ่งในบทความนี้จะมาคุยกันเรื่องนี้โดยใช้ปัญหาอาหารเย็นของนักปราชญ์ (Dining Philosophers) ที่เป็นปัญหาคลาสิกที่กล่าวถึงปัญหาการ sychonization ในสภาวะแวดล้อม multi-thread และให้เห็นภาพทางเทคนิคของการแก้ไขปัญหาของปัญหานี้

```java
class Philosopher implements Runnable {
    private Object leftFork;
    private Object rightFork;
    public Philosopher(Object leftFork, Object rightFork) {
        this.leftFork = leftFork;
        this.rightFork = rightFork;
    }
    private void doAction(String action) throws InterruptedException {
        System.out.println(Thread.currentThread().getName() + " " + action);
        Thread.sleep((int)(Math.random() * 100));
    }
    @Override
    public void run() {
        try {
            while (true) {
                doAction(System.nanoTime() + ": Thinking");
                synchronized (leftFork) {
                    doAction(System.nanoTime() + ": Pick up left fork");
                    synchronized (rightFork) {
                        doAction(System.nanoTime() + ": Pick up right fork");
                        doAction(System.nanoTime() + ": Eating");
                        doAction(System.nanoTime() + ": Put down right fork");
                    }
                    doAction(System.nanoTime() + ": Put down left fork");
                }
            }
        }
        catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return;
        }
    }
}

public class Demo {
    public static void main(String[] args) {
        Philosopher[] philosophers = new Philosopher[5];
        Object[] forks = new Object[philosophers.length];
        for(int i = 0; i < forks.length; i++) {
            forks[i] = new Object();
        }
        for(int i = 0; i < philosophers.length; i++) {
            Object leftFork = forks[i];
            Object rightFork = forks[(i + 1)%forks.length];
            philosophers[i] = new Philosopher(leftFork, rightFork);
            Thread t = new Thread(philosophers[i], "Philosopher " + (i + 1));
            t.start();
        }
    }
}
```

```
Philosopher 1 68627059269375: Thinking
Philosopher 3 68627059388407: Thinking
Philosopher 2 68627059288283: Thinking
Philosopher 4 68627059795612: Thinking
Philosopher 5 68627060109813: Thinking
Philosopher 2 68627089085695: Pick up left fork
Philosopher 2 68627107648184: Pick up right fork
Philosopher 4 68627112998381: Pick up left fork
Philosopher 4 68627117087953: Pick up right fork
Philosopher 2 68627129442996: Eating
Philosopher 4 68627154087269: Eating
Philosopher 1 68627155391687: Pick up left fork
Philosopher 2 68627217280798: Put down right fork
Philosopher 3 68627251259623: Pick up left fork
Philosopher 2 68627251253126: Put down left fork
Philosopher 4 68627252752908: Put down right fork
....
```

4. ลำดับขั้นการปฏิบัติการ

4.1. จงสร้างหน้า GUI เพื่อทำการทดสอบสร้าง Thread ที่มีส่วนประกอบดังต่อไปนี้

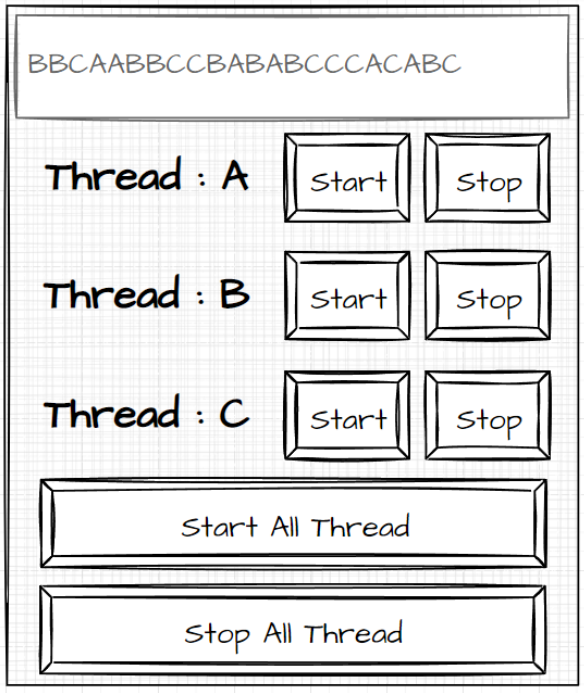
4.1.1. สร้าง Thread A ที่สร้างจาก Inner Class

4.1.2. สร้าง Thread B และ C จาก Class ปกติ

4.1.3. แต่ละ Thread จะมีปุ่ม Start เพื่อเริ่มต้นพิมพ์ตัวอักษรของ Thread ลงในช่อง Textbox และ Stop เพื่อหยุดการพิมพ์ตัวอักษรของ Thread ในช่อง Textbox

4.1.4. สร้างปุ่ม Start All Thread เพื่อทำให้ Thread แต่ละตัวทำงานพร้อมกัน

4.1.5. สร้างปุ่ม Stop All Thread เพื่อให้ Thread แต่ละตัวหยุดทำงานพร้อมกัน



โค้ดโปรแกรมของปุ่ม Start และ Stop ของ Thread A

```
Start
Threadouter outer = new Threadouter();
Threadouter.ThreadA threadA = outer.new ThreadA() ;
threadA.start();
public class Threadouter {
 public class ThreadA extends Thread {
 Threadlab window = new Threadlab();
 int count = 0;
 boolean state = true;
 public void stateA() { state = false; }
 public void stateAstart() { state = true; }
 public void run() {
 while( state ) {
 this.window.text = text + "A" ;
 System.out.print( this.window.text );
 try {
 TimeUnit.SECONDS.sleep(1);
 } catch (InterruptedException e) {
 e.printStackTrace();
 }
 }
 }
 }
 }
Stop
threadA.stateA();
```

โค้ดโปรแกรมของปุ่ม Start และ Stop ของ Thread B

```
Start
ThreadB threadB = new ThreadB();
threadA.start();
public class ThreadB extends Thread {
 Threadlab window = new Threadlab();
boolean state = true;
public void stateB() { state = false; }
public void run() {
 while( state ) {
 this.window.text = window.text + "B" ;
 System.out.print( this.window.text );
 try {
 TimeUnit.SECONDS.sleep(1);
 } catch (InterruptedException e) {
 e.printStackTrace();
 }
 }
 }
}
Stop
threadB.stateB();
```

โค้ดโปรแกรมของปุ่ม Start และ Stop ของ Thread C

```
Start
ThreadC threadC = new ThreadC();
threadC.start();
public class ThreadC extends Thread {
 Threadlab window = new Threadlab();
boolean state = true;
public void stateC() { state = false; }
public void run() {
 while( state ) {
 this.window.text = window.text + "C" ;
 System.out.print( this.window.text );
 try {
 TimeUnit.SECONDS.sleep(1);
 } catch (InterruptedException e) {
 e.printStackTrace();
 }
 }
 }
}
Stop
threadC.stateC();
```

โค้ดโปรแกรมของปุ่ม Start All Thread

```
1   threadA.start();
2   threadB.start();
3   threadC.start();
```

โค้ดโปรแกรมของปุ่ม Stop All Thread

```
1   threadA.stateA();
2   threadB.stateB();
3   threadC.stateC();
```

5. สรุปผลการปฏิบัติการ

การใช้งาน thread นั้นเป็นการทำงานแบบขนานที่ทำงานหลายๆ คำสั่งพร้อมๆกัน โดยที่ไม่ต้องทำงานเป็นลำดับ งานใดทำเสร็จก่อนก็ทำการ return ก่อน

6. คำถามท้ายการทดลอง

6.1. Inner Class แตกต่างจาก Class แบบปกติอย่างไร?

การใช้งาน thread นั้นเป็นการทำงานแบบขนานที่ทำงานหลายๆคำสั่งพร้อมๆกัน โดยที่ไม่ต่นเองทำงานเป็นลำดับ งานใดทำเสร็จก่อนก็ทำการ return ก่อน

6.2. เมื่อใดจึงเป็นช่วงเวลาที่ดีที่สุดในการใช้งาน Inner Class

หาก code เริ่มที่จะซับซ้อนและจำเป็นที่จะต้องสร้างอีก class แต่ไม่อยากทำไฟล์แยก

6.3. ข้อควรระวังในการใช้งาน Thread คืออะไร?

คำสั่งที่จะป้อนให้ thread นั้นจำเป็นที่จะต้องมีจุดสิ้นสุดไม่ deadlock