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 เดียวกันถึงแม้ว่าการทำ Package ก็ช่วยในเรื่องดังกล่าวแล้วแต่การทำ Nested Classes ทำให้การ group แข็งแรงมากขึ้นอีกขั้น

3.2. จงยกตัวอย่างการสร้าง Inner Class

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 แบบที่ในระบบปฏิบัติการกจะให้โปรแกรมสามารถทำงานพร้อมกันได้ เช่น ฟังเพลงไปด้วยพิมพ์งานไปด้วยก็ได้ นอกจากนี้เธรดยังสามารถทำงานพร้อมกันได้ด้วยเรียกว่า multi-threaประโยชน์จาก Thread นั้นโปรแกรมจะต้องเป็นแบบ Multithreading ซึ่งจะมีข้อได้เปรียบ เช่น มีการตอบสนองของโปรแกรมที่ดีกว่าการประมวลผลเร็วกว่า ใช้ทรัพยากรน้อยกว่า การใช้ประโยชน์จากระบบมากกว่า และการทำงานแบบขนาน

3.6. การเริ่มต้นใช้งาน Thread มีขั้นตอนอย่างไรบ้าง?

```
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 คลาสเธรดขยายคลาสของวัตถุและใช้อินเทอร์เฟซ Runnable คลาส Thread มีตัวสร้างและวิธีการในการสร้างและดำเนินการกบเธรด˜ Runnable เป็นอินเตอร์เฟสในแพ็คเกจ java.lang การใช้อินเตอร์เฟสที่เรียกใช้งาน ได้นั้นเราสามารถกำหนดเธรดได้ส่วนต่อประสานที่รัน ได้มีวิธีการเดียวรัน () ซึ่งนำมาใช้โดยคลาสที่ใช้ส่วนต่อประสาน Runnable มันเป็นที่ต้องการที่จะใช้อินเตอร์เฟซที่เรียกใช้แทนการขยายชั้นเรียนด้ายเนื่องจากการใช้ Runnable ทำให้โค้ดของคุณเชื่อมโยงกนอย่างหลวม ๆ เนื่องจากโค้ดของเธรดต่างจากคลาสที่กำหนดงานให้กบเธรด˜ มันต้องใช้หน่วยความจำน้อยลงและยังช่วยให้ชั้นเรียนที่จะรับช่วงชั้นอื่น ๆ

3.8. สถานะ Deadlock มีลักษณะเป็นอย่างไร? อธิบายพร้อมยกตัวอย่างประกอบ

ซึ่งเป็นสถานการณที่ 2 thread หรือมากกวาถูกล็อก˜ (LOCKED) ตลอดกาล ซึ่งรอกนและกันให้ทำงานให้เสร็จก่อน ซึ่งในบทความนี้จะมาคยกนเรื่องนี˜ โดยใช้ปัญหาอาหารเย็นของนักปราชญ์ (Dining Philosophers) ที่เป็นปัญหาคลาสิกที่กล่าวถึงปัญหาการ sychonization ในสภาวะแวดล้อม multi-thread และให้เห็นภาพทางเทคนิคของการแกไขปัญหาของปัญหาน

4. ลำดับขั้นการปฏิบัติการ

4.1. จงสร้างหน้า GUI เพื่อทำการทดสอบสร้าง Thread ที่มีส่วนประกอบดังต่อไปนี้

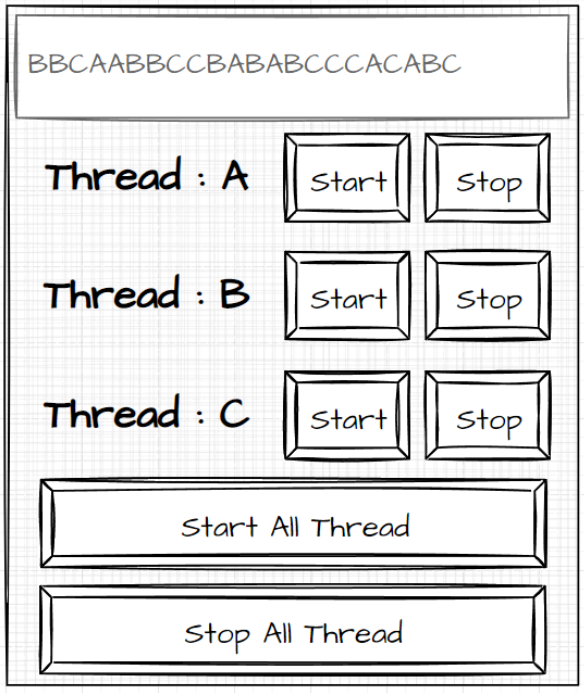
4.1.1. สร้าง Thread A ที่สร้างจาก Inner Class

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

โค้ดโปรแกรมของปุ่ม Start และ Stop ของ Thread B

```java
package lab13;

import java.util.concurrent.TimeUnit;

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();
            }
        }
    }
}
```

โค้ดโปรแกรมของปุ่ม Start และ Stop ของ Thread C

```java
package lab13;

import java.util.concurrent.TimeUnit;

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();
            }
        }
    }
}
```

โค้ดโปรแกรมของปุ่ม Start All Thread

```
package lab13;

import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.wb.swt.SWTResourceManager;

import org.eclipse.swt.widgets.Label;

import java.util.concurrent.TimeUnit;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;

public class Threadlab extends Thread {

    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();
                    }
                }
            }
        }
    }
```

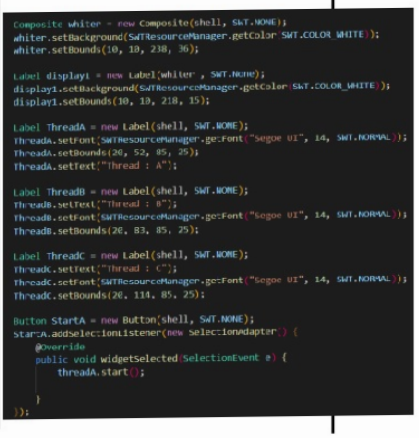
```
}
int n = 0;
String text = "" ;
protected Shell shell;
public static void main(String[] args) {
    try {
        Threadlab window = new Threadlab();
        window.open();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void open() {
    Display display = Display.getDefault();
    createContents();
    shell.open();
    shell.layout();
    while (!shell.isDisposed()) {
        if (!display.readAndDispatch()) {
            display.sleep();
        }
    }
}

public void createContents() {

    shell = new Shell();
    shell.setSize(274, 261);
    shell.setText("SWT Application");

    Threadouter outer = new Threadouter();
    threadouter.threadA threadA = outer.new threadA() ;
    ThreadB threadB = new ThreadB();
    ThreadC threadC = new ThreadC();
```



```
Composite whiter = new Composite(shell, SWT.NONE);
whiter.setBackground(SWTResourceManager.getColor SWT.COLOR_WHITE));
whiter.setBounds(10, 10, 238, 36);

Label display1 = new Label(whiter , SWT.NONE);
display1.setBackground(SWTResourceManager.getColor(SWT.COLOR_WHITE));
display1.setBounds(10, 10, 218, 15);

Label ThreadA = new Label(shell, SWT.NONE);
ThreadA.setFont(SWTResourceManager.ge:Font("Segoe UI", 14, SWT.NORMAL));
ThreadA.setBounds(20, 52, 85, 25);
ThreadA.setText("Thread : A");

Label ThreadB = new Label(shell, SWT.NONE);
ThreadB.setText("Thread : B");
ThreadB.setFont(SWTResourceManager.ge:Font("Segoe UI", 14, SWT.NORMAL));
ThreadB.setBounds(20, 83, 85, 25);

Label ThreadC = new Label(shell, SWT.NONE);
ThreadC.setText("Thread : C");
ThreadC.setFont(SWTResourceManager.ge:Font("Segoe UI", 14, SWT.NORMAL));
ThreadC.setBounds(20, 114, 85, 25);

Button StartA = new Button(shell, SWT.NONE);
StartA.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        threadA.start();
    }
});
```

โค้ดโปรแกรมของปุ่ม Stop All Thread

```
StartA.setBounds(131, 55, 52, 25);
StartA.setText("Start");

Button StartB = new Button(shell, SWT.NONE);
StartB.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        threadB.start();
    }
});
StartB.setText("Start");
StartB.setBounds(131, 86, 52, 25);

Button StartC = new Button(shell, SWT.NONE);
StartC.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        threadC.start();

    }
});
StartC.setText("Start");
StartC.setBounds(131, 117, 52, 25);

Button StopA = new Button(shell, SWT.NONE);
StopA.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        threadA.stateA();
    }
});
StopA.setText("Stop");
StopA.setBounds(189, 55, 52, 25);
```

```
Button StopB = new Button(shell, SWT.NONE);
StopB.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        threadB.stateB();
    }
});
StopB.setText("Stop");
StopB.setBounds(189, 86, 52, 25);

Button StopC = new Button(shell, SWT.NONE);
StopC.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        threadC.stateC();
    }
});
StopC.setText("Stop");
StopC.setBounds(189, 117, 52, 25);

Button StartAll = new Button(shell, SWT.CENTER);
StartAll.addSelectionListener(new SelectionAdapter() {
    @Override
    public void widgetSelected(SelectionEvent e) {
        threadA.start();
        threadB.start();
        threadC.start();
    }
});
StartAll.setBounds(30, 153, 192, 25);
StartAll.setText("Start All Thread");
```

```
        StartAll.setBounds(30, 153, 192, 25);
        StartAll.setText("Start All Thread");

        Button StopAll = new Button(shell, SWT.CENTER);
        StopAll.addSelectionListener(new SelectionAdapter() {
            @Override
            public void widgetSelected(SelectionEvent e) {
                threadA.stateA();
                threadB.stateB();
                threadC.stateC();
            }
        });
        StopAll.setText("Stop All Thread");
        StopAll.setBounds(30, 184, 192, 25);

    }
}
```

5. สรุปผลการปฏิบัติการ

การใช้งาน thread นั้นเป็นการทำงานแบบขนานที่ทำงานหลายๆคำสั่งพร้อมๆกัน โดยที่ไม่ต้องตนเองทำงานเป็นลำดับ งานใดทำเสร็จก่อนกทำการ return ก่อน

6. คำถามท้ายการทดลอง

6.1. Inner Class แตกต่างจาก Class แบบปกติอย่างไร?

thread นั้นเป็นการทำงานแบบขนานที่ทำงานหลายๆคำสั่งพร้อมๆกัน โดยที่ไม่ต้องทำงานเป็นลำดับ งานใดทำเสร็จก่อนกทำการ return ก่อน
การใช้งาน thread นั้นเป็นการทำงานแบบขนานที่ทำงานหลายๆคำสั่งพร้อมๆกัน โดยที่ไม่ต้องตนเองทำงานเป็นลำดับ งานใดทำเสร็จก่อนกทำการ return ก่อน

6.2. เมื่อใดจึงเป็นช่วงเวลาที่ดีที่สุดในการใช้งาน Inner Class

หาก code เริ่มที่จะซับซ้อนและจำเป็นที่จะต้องสร้างอีก class แต่ไม่อยากทำไฟล์แยก

6.3. ข้อควรระวังในการใช้งาน Thread คืออะไร?

คำสั่งที่จะป้อนให้ thread นั้นจำเป็นที่จะต้องมีจุดสิ้นสุดไม่ deadlock