ใบงานการทดลองที่ 13

เรื่อง การใช้งาน Inner Class และการใช้งาน Thread

1. จุดประสงค์ทั่วไป

1.1. รู้และเข้าใจการโปรแกรมเชิงวัตถุ การกำหนดวัตถุ การใช้วัตถุ

1.2. รู้และเข้าใจการทำหลายงานพร้อมกัน

2. เครื่องมือและอุปกรณ์

เครื่องคอมพิวเตอร์1 เครื่อง ที่ติดตั้งโปรแกรม Eclipse

3. ทฤษฎีการทดลอง

3.1. Nest Class คืออะไร? มีวัตถุประสงค์เพื่ออะไร? อธิบายพร้อมยกตัวอย่างประกอบ

Nest Class (หรือ Inner Class) คือการประกาศคลาสภายในคลาสอีกอันหนึ่งโดยคลาสที่อยู่ภายในจะถือว่าเป็นส่วนประกอบของคลาสภายนอกนั้น ๆ โดย Nest Class จะอยู่ภายใต้ขอบเขตของคลาสภายนอกซึ่งทำให้สามารถเข้าถึงสมาชิกและเมทอดของคลาสภายนอกได้โดยตรง

3.2. จงยกตัวอย่างการสร้าง Inner Class

```
private int x;
 public class InnerClass { public void printX() {
 System.out.println("x = " + x); }
 } public void createInnerClass() { InnerClass inner = new InnerClass();
 inner.printX(); }}
```

3.3. จงยกตัวอย่างการเรียกใช้งาน Instance ที่มีการเรียกใช้งาน Properties ภายใน Inner Class

```
public class OuterClass {
 private int x;
 public class InnerClass {
 public void printX() {
 System.out.println("x = " + x); }
```

3.4. จงยกตัวอย่างการเรียกใช้งาน Instance ที่มีการเรียกใช้งาน Method ภายใน Inner Class

```
public class OuterClass {
 private int x; public class InnerClass {
 public void printX() {
```

```
System.out.println("x = " + x); } }
```

3.5. Thread คืออะไร? มีประโยชน์อย่างไร? อธิบายพร้อมยกตัวอย่างประกอบ

Thread คือเทรด (thread) ที่ใช้ในการทำงานแบบพร้อมกันกับเทรดอื่นๆ ในโปรแกรม หรือก็คือเทรดเป็นเหมือนเส้นทางที่มีการทำงานของโปรแกรมซึ่งแตกต่างจากกระบวนการ (process) ที่เป็นเหมือนห้องทดลองหนึ่งห้องที่มีสายงานหลายสายงานแต่แตกต่างกันตรงที่กระบวนการมีการจัดการแยกกันของหน่วยงานเป็นก้อนๆ ทำให้การทำงานของส่วนต่างๆ ของโปรแกรมไม่สามารถเข้าถึงหรือแชร์กันได้ในขณะที่ Thread สามารถใช้แบ่งแยกและประมวลผลงานได้พร้อมกัน

3.6. การเริ่มต้นใช้งาน Thread มีขั้นตอนอย่างไรบ้าง?

สร้าง Instance ของ Thread object โดยสร้าง object ใหม่ของคลาสที่สืบทอดมาจากคลาส Thread หรือผ่านการสร้าง Inner Class กำหนดชื่อเรียกของ Thread โดยใช้ constructor ของ Thread objectสร้างเมธอด run() ที่ประกอบไปด้วยงานที่ต้องการให้ Thread ทำเรียกเมธอด start() เพื่อเริ่มการทำงานของ Thread

3.7. ระหว่าง Thread และ Runnable มีรูปแบบการใช้งานที่เหมือนหรือแตกต่างกันอย่างไร?

Thread และ Runnable เป็นคลาสที่ใช้ในการสร้าง Thread ในภาษา Java โดยมีรูปแบบการใช้งานที่แตกต่างกันอย่างมาก

3.8. สถานะ Deadlock มีลักษณะเป็นอย่างไร? อธิบายพร้อมยกตัวอย่างประกอบ

Deadlock เกิดขึ้นเมื่อสองหรือมากกว่า Thread จะไม่สามารถดำเนินการต่อไปได้เนื่องจากมีการรอคอยกันอยู่ เช่น Thread A รอให้ Thread B ทำงานเสร็จก่อน ในขณะเดียวกัน Thread B ก็รอให้ Thread A ทำงานเสร็จก่อน จนกระทั่งไม่มี Thread ใดสามารถดำเนินการต่อไปได้อีก

4. ลำดับขั้นการปฏิบัติการ

4.1. จงสร้างหน้า GUI เพื่อทำการทดสอบสร้าง Thread ที่มีส่วนประกอบดังต่อไปนี้

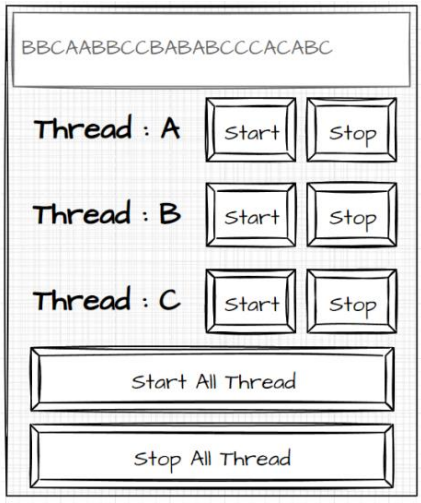
4.1.1. สร้าง Thread A ที่สร้างจาก Inner Class

4.1.2. สร้าง Thread B และ C จาก Class ปกติ

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

4.1.4. สร้างปุ่ม Start All Thread เพื่อทำให้Thread แต่ละตัวทำงานพร้อมกัน

4.1.5. สร้างปุ่ม Stop All Thread เพื่อให้Thread แต่ละตัวหยุดทำงานพร้อมกัน



โค้ดโปรแกรมของปุ่ม Start และ Stop ของ Thread A

```
private class ThreadA implements Runnable {
        private Thread thread;
        private volatile boolean isRunning = false;
        public void startThread() {
            isRunning = true;
            thread = new Thread(this);
            thread.start();
        }
        public void stopThread() {
            isRunning = false;
    }
```

โค้ดโปรแกรมของปุ่ม Start และ Stop ของ Thread B

```
private class ThreadB extends Thread {

        private volatile boolean isRunning = false;

        public void startThread() {

                isRunning = true;

                start();

        }

        public void stopThread() {

                isRunning = false;

        }

        public void run() {

                while (isRunning) {

                 textArea.append("B");

                        try {

                                Thread.sleep(500);
                                } catch (InterruptedException e) {
```

```
                    e.printStackTrace();
                }
            }
        }
}
```

โค้ดโปรแกรมของปุ่ม Start และ Stop ของ Thread C

```
private class ThreadC extends Thread {
    private volatile boolean isRunning = false;
    public void startThread() {
    isRunning = true;
    start();
    }
    public void stopThread() {
    isRunning = false;
    }
    public void run() {
        while (isRunning) {
            textArea.append("C");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

โค้ดโปรแกรมของปุ่ม Start All Thread

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == startAllBtn) {
        threadA.startThread();
        threadB.startThread();
        threadC.startThread();
    } else if (e.getSource() == stopAllBtn) {
        threadA.stopThread();
        threadB.stopThread();
        threadC.stopThread();
    }
}
```

โค้ดโปรแกรมของปุ่ม Stop All Thread

```
        } else if (e.getSource() == stopAllBtn) {
         threadA.stopThread();
         threadB.stopThread();
         threadC.stopThread();
         }
}
```

5. สรุปผลการปฏิบัติการ

การทำงานในชุดคำสั่งสามารถแทรกแซงเข้าหากันได้

6. คำถามท้ายการทดลอง

6.1. Inner Class แตกต่างจาก Class แบบปกติอย่างไร?

Inner Class เป็น Class ภายใน Class อีกชั้นหนึ่ง ซึ่งมีลักษณะเด่นต่างจาก Class แบบปก

6.2. เมื่อใดจึงเป็นช่วงเวลาที่ดีที่สุดในการใช้งาน Inner Class

การใช้งาน Inner Class เหมาะสมมากกับกรณีที่ต้องการใช้งานคลาสภายในคลาสอื่นๆ และมีความสัมพันธ์กันแบบแตกต่างกัน เช่น การใช้งาน Listener ใน GUI ซึ่งส่วนมากจะถูกนำเข้าเป็น Inner Class เพื่อให้สามารถเข้าถึงตัวแปรและเมทอดของ Outer Class ได้ง่าย ๆ และเพิ่มความสะดวกในการจัดการโค้ด

6.3. ข้อควรระวังในการใช้งาน Thread คืออะไร?

Race Condition: เกิดจากการมี Thread ที่แก้ ไขข้อมูลเดียวกันพร้อมกัน ข้อมูลอาจเสียหายหรือผิดพลาด

Deadlock: เกิดจากการรออยู่ระหว่าง Thread 2 หรือมากกว่าและแต่ละ Thread กำลังรอให้ทราบสถานะจาก Thread อื่น ๆ และทั้งหมดติดกัน

Starvation: เกิดจากการไม่ได้กำหนดลำดับการประมวลผล Thread ทำให้ Thread บางตัวไม่สามารถทำงานได้

Thread-Safety: เป็นการรักษาความปลอดภัยของการใช้งาน Thread ในรูปแบบต่าง ๆ เพื่อป้องกันการแชร์ข้อมูลที่ไม่เหมาะสม ซึ่งข้อมูลอาจเสียหายหรือผิดพลาด

Memory Visibility: เกี่ยวข้องกับการเข้าถึงและอัปเดตข้อมูลในหน่วยความจำที่ใช้ร่วมกันได้ ในกรณีที่ Thread ต่างกันมองเห็นค่าข้อมูลที่แตกต่างกัน ข้อมูลอาจเสียหายหรือผิดพลาด

การระบุและจัดการกับข้อควรระวังดังกล่าวจะช่วยให้การใช้งาน Thread มีประสิทธิภาพและปลอดภัยยิ่งขึ้น