โปรแกรมสำหรับวิเคราะห์ความซับซ้อนของ Source code

A program to assist analysis of code complexity.

นายกมล สุวรรณกิจ รหัส 47360037

นายนฤเบศ หาญรักษ์ รหัส 47361969

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมไฟฟ้าและคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร

ปีการศึกษา 2550

ใบรับรองโครงงานวิศวกรรม

หัวข้อโครงงาน	โปรแกรมสำหรับวิเคราะห์ความซับซ้อนของ Source code
ผู้ดำเนินโครงงาน	นายกมล สุวรรณกิจ รหัส 47360037
	นายนฤเบศ หาญรักษ์ รหัส 47361969
อาจารย์ที่ปรึกษา	ดร. สุรเดช จิตประไพกุลศาล
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
ภาควิชา	วิศวกรรมไฟฟ้าและคอมพิวเตอร์
ปีการศึกษา	2550

.....

คณะวิศวกรรมศาสตร์ มหาวิทยาลัยนเรศวร อนุมัติให้โครงงานฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะกรรมการสอบโครงงานวิศวกรรม

.....ประธานกรรมการ
(ดร.สุรเดช จิตประไพกุลศาล)

.....กรรมการ
(ผศ.ดร.ธนิต มาลากร)

.....กรรมการ
(ดร.พนมขวัญ ริยะมงคล)

.....กรรมการ
(อาจารย์ภาณุพงศ์ สอนคม)

หัวข้อโครงงาน	โปรแกรมสำหรับวิเคราะห์ความซับซ้อนของ Source code
ผู้ดำเนินโครงงาน	นายกมล สุวรรณกิจ รหัส 47360037
	นายนฤเบศ หาญรักษ์ รหัส 47361969
อาจารย์ที่ปรึกษา	ดร. สุรเดช จิตประไพกุลศาล
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
ภาควิชา	วิศวกรรมไฟฟ้าและคอมพิวเตอร์
ปีการศึกษา	2550

.....

บทคัดย่อ

โครงงานนี้ได้ทำการพัฒนาโปรแกรมเพื่อใช้ในการคำนวณ Software Metric จาก Source code ภาษา Java โดยวัดและแสดงผลของ Line Counting Metrics, Complexity Metrics และ Halstead Metrics จากการทดลองพบว่าผลที่ได้จากโปรแกรมนี้เป็นค่าเดียวกับผลจากการคำนวณด้วยมือ (Manual Calculation) ประโยชน์ประการหนึ่งของโปรแกรมนี้คือสามารถวัดคุณภาพเชิงปริมาณของ Source code ภาษา Java ได้ ซึ่งอาจใช้เป็นแนวทางในการปรับปรุงการเขียนโปรแกรมให้มีคุณภาพมากขึ้น อนึ่งโปรแกรมได้รับการพัฒนาโดยใช้ JavaCC เป็นตัวช่วยในการ parse source code

Project Title	A program to assist analysis of code complexity.		
Name	Mr. Kamon	Suwannakit	ID. 47360037
	Mr. Naruebet	Hanrak	ID. 47361969
Project Advisor	Dr. Suradet	Jitprapaikulsarn	
Major	Computer Engineering		
Department	Electrical and Computer Engineering		
Academic Year	2007		

.....

ABSTRACT

This project develops a program for determing software metrics from Java source codes. The program measures and shows the following metrics: i) Line Counting Metrics, ii) Complexity Metrics and iii) Halstead Metrics. Based on our experiment, the results from this program are identical to the results from manual calculation of these metrics. One benefit of any programs of this kind is that it can be used to measure some qualities of Java source codes so that we can adjust our program to better quality. We develop this program using Java SDK 1.6 with the assistance from JavaCC, a Java-based parser generator.

กิตติกรรมประกาศ

ขอขอบพระคุณ ดร.สุรเดช จิตประไพกุลศาล อาจารย์ที่ปรึกษาโครงงานนี้ ที่คอยให้คำปรึกษา ความช่วยเหลือตลอดจนคำแนะนำและแนวทางต่างๆ ในการทำโครงงานนี้ และสุดท้ายขอขอบพระคุณอาจารย์ทุกท่านและเพื่อนๆ ทุกคนที่ยังไม่ได้เอ่ยนามที่คอยให้การสนับสนุนผู้ดำเนินโครงงาน ให้สามารถทำโครงงานนี้จนสำเร็จลุล่วงไปได้ด้วยดี

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของโครงงาน

"Software Metrics" คือ การวัดคุณสมบัติของซอฟต์แวร์ โดยเฉพาะคุณสมบัติที่เกี่ยวข้องกับคุณภาพของซอฟต์แวร์ ปัจจุบันมีทฤษฎีหรือวิธีในการวัดคุณสมบัติมีหลายวิธี เช่น Line counting metrics, Complexity metrics และ Halstead ซึ่งในแต่ละคุณสมบัติมีหลายวิธีในการตรวจสอบ

ในปัจจุบันผู้พัฒนาซอฟต์แวร์ได้ผลิตซอฟต์แวร์ขึ้นมาเป็นจำนวนมากด้วยความรวดเร็ว ซึ่งซอฟต์แวร์ที่พัฒนาขึ้นนั้นต้องคำนึงถึงความต้องการและผลลัพธ์ที่ได้ให้แก่ลูกค้าเป็นสำคัญ ซึ่งทำให้มองข้ามถึงสิ่งสำคัญประการหนึ่งในการผลิตซอฟต์แวร์ นั่นก็คือคุณภาพของซอฟต์แวร์ การตรวจวัดคุณภาพของซอฟต์แวร์นั้นมีหลากหลายวิธี แต่ถ้าต้องการตรวจวัดคุณภาพของซอฟต์แวร์หลังจากที่ผลิตซอฟต์แวร์แล้ว สามารถตรวจวัดได้จาก "Software Metrics" ที่บอกถึงคุณสมบัติหลายๆด้านของซอฟต์แวร์นั้นๆ ไม่ว่าเป็น Line counting metrics, Complexity metrics และ Halstead metrics และสามารถนำผลลัพธ์ที่ได้จากการตรวจวัดไปประยุกต์ใช้ในการพัฒนาซอฟต์แวร์ เช่น นำไปประยุกต์ใช้กับการตรวจวัดคุณภาพ Source Code ของ Tester เพื่อประมาณโอกาสการเกิดข้อผิดพลาดของ Source code นั้นๆ หรือตรวจดูว่าควรให้ความสำคัญกับการตรวจวัดใน Method ใดมากกว่ากัน โดยดูจากค่าที่ได้ "Software Metrics" ทำให้ไม่ต้องเสียเวลาดู Source code ทั้งหมด หรือนำไปประยุกต์ใช้ในระหว่างการพัฒนาโปรแกรมโดยให้มีการตรวจวัดความซับซ้อนของ Source code หรือ Method เพื่อดูว่า Source code หรือ Method ใดมีความซับซ้อนขนาดไหน ถ้ามีความซับซ้อนมากก็หาวิธีลดความซับซ้อนลง ทำให้โอกาสที่ข้อผิดพลาดเกิดขึ้นจากความซับซ้อนของซอฟต์แวร์ที่พัฒนาขึ้นนั้นลดลง

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อตรวจวัดคุณสมบัติบางอย่าง ซึ่งมีส่วนเกี่ยวข้องกับคุณภาพของ Source code
2. เพื่อศึกษาการทำงาน และวิธีการเขียนคอมไพเลอร์

1.3 ขอบข่ายของโครงงาน

1. สามารถตรวจวัด Line counting metrics ได้
2. สามารถตรวจวัด Complexity metrics ได้
3. สามารถตรวจวัด Halstead metrics ได้
4. สามารถใช้ตรวจวัดกับ Source code ภาษา Java ได้

1.4 ขั้นตอนการดำเนินการ

1. ศึกษาค้นคว้าหาข้อมูลและกำหนดขอบเขตของงาน
2. เขียนโปรแกรมทำเป็นซอฟต์แวร์
3. ทำการ Test ซอฟต์แวร์โดยผู้จัดทำ
4. ทดสอบซอฟต์แวร์โดยผู้ใช้
5. ทำการสรุปรวบรวมข้อมูลทั้งหมดของโครงงาน
6. จัดทำ Report

1.5 แผนการดำเนินงาน

กิจกรรม	เดือน- ปี										
	2549		2550								
	พฤศจิกายน	ธันวาคม	มกราคม	กุมภาพันธ์	มีนาคม	เมษายน	พฤษภาคม	มิถุนายน	กรกฎาคม	สิงหาคม	กันยายน
1. ศึกษาค้นคว้าหาข้อมูลและกำหนดขอบเขตของงาน	←			→							
2. เขียนโปรแกรมทำเป็นซอฟต์แวร์					←		→				
3. ทดสอบซอฟต์แวร์โดยผู้จัดทำ							←	→			
4. ทดสอบซอฟต์แวร์โดยผู้ใช้								←	→		
5. ทำการสรุปรวบรวมข้อมูลทั้งหมดของโครงงาน									←	→	
6. จัดทำ Report										←	→

1.6 ผลที่คาดว่าจะได้รับ

1. ซอฟต์แวร์ที่สามารถตรวจวัดคุณสมบัติ ซึ่งมีความเกี่ยวข้องกับคุณภาพของ Source code ภาษา Java ได้

2. ได้รับความรู้ความเข้าใจเกี่ยวกับการทำงานของ คอมไพเลอร์

1.7 งบประมาณที่ใช้

1.	ค่าจัดซื้อหนังสือ	1, 000 บาท
2.	ค่าจัดพิมพ์เอกสารและถ่ายเอกสาร	700 บาท
3.	ค่าวัสดุคอมพิวเตอร์	300 บาท
	รวม	2, 000 บาท

**หมายเหตุ ถัวเฉลี่ยทุกรายการ

บทที่ 2
หลักการและทฤษฎีที่เกี่ยวข้อง

ในบทนี้กล่าวถึงในเรื่องของหลักการและทฤษฎีต่างๆที่เกี่ยวข้อง เพื่อนำมาประยุกต์ใช้กับโครงงานนี้ ได้แก่ Software Metrics, คอมไพเลอร์ และ Java Compiler Compiler

2.1 Software Metrics

Software metrics คือ การตรวจวัดคุณสมบัติบางประการของส่วนประกอบของซอร์ฟแวร์ โดยปกติ Software metrics ทำการตรวจวัด metrics ต่างๆดังนี้

- Order of growth
- Source lines of code
- Cyclomatic complexity
- Function point analysis
- Bugs per line of code
- Code coverage
- Number of lines of customer requirements.
- Number of classes and interfaces
- Robert Cecil Martin's software package metrics
- Cohesion
- Coupling

โดยในโครงงานนี้ได้เลือก Metrics มาเพียงบาง Metrics เท่านั้นได้แก่ Source lines of code ซึ่งอยู่ใน Line Counting Metrics, Cyclomatic complexity ซึ่งอยู่ใน Complexity Metrics และ Halstead Metrics

2.1.1 Line Counting Metrics

SLOC (Source line of code) คือ การนับจำนวนบรรทัดของ Source Code แบบ Logical โดยไม่นับ Blank line และบรรทัดที่เป็น Comment ซึ่งช่วยให้รู้ถึงปริมาณของชุดคำสั่งและขนาดของซอฟต์แวร์อย่างคร่าวๆ แต่การทำการวัด Line counting metrics ยังไม่มีมาตราฐานที่แน่นอนในการนับในแต่ละบรรทัด ส่งผลทำให้วิธีการนับที่ใช้ในที่นี้อาจไม่ได้มาตราฐาน ถ้ามีการกำหนดมาตราฐานในการนับบรรทัดขึ้นใหม่ในภายหลัง

จากรูปที่ 2.1 จะเห็นว่าในบรรทัดที่ 1 และ 6 เป็น Line comment และบรรทัดที่ 3 เป็น Blank line จึงไม่นับ ในบรรทัดที่ 9 และ 10 นั้นนับรวมกันเป็น 1 บรรทัดเพราะเป็น Statement ที่ยังไม่สิ้นสุดในหนึ่งบรรทัด ส่วนบรรทัดอื่นที่เป็น Statement ตามรูปแบบปกติก็นับเป็น 1 บรรทัด

```
1  // Java Package
2  import javax.swing.JOptionPane;                          // นับเป็น 1 บรรทัด
3
4  public class Wellcome                                    // นับเป็น 1 บรรทัด
5  {
6      // Main program begins execution of java application
7      public static void main(String[] args)               // นับเป็น 1 บรรทัด
8      {
9          JOptionPane.showMessageDialog(
10             null, "Wellcome to java Programming");       // นับเป็น 1 บรรทัด
11         system.exit(0); // Terminate application with window  // นับเป็น 1 บรรทัด
12     } // exit method main
13 } // end class wellcome
14
```

รูปที่ 2.1 ตัวอย่างการนับบรรทัดใน Source code ภาษา Java [6]

2.1.2 Complexity Metrics

v(G) (Cyclomatic Complexity) คือ การวัดความซับซ้อนของ Source code โดยคำนวณจากการนำ Source code มาสร้างเป็น Connected Graph เพื่อแสดงลำดับการทำงานของ Source code จากนั้นนับจำนวนของ Node และ Edge ในกราฟ แล้วคำนวณผลลัพธ์ตามสมการที่ 2.1 ซึ่งการวัดความซับซ้อนช่วยให้ทราบถึงความซับซ้อนของ Source code ที่นำมาตรวจสอบว่าซับซ้อนเพียงใด โดยนำผลลัพธ์ที่ได้จากการคำนวณในสมการที่ 2.1 นำมาเปรียบเทียบกับค่าของระดับความซับซ้อนในตารางที่ 2.1 ถ้าโปรแกรมมีความซับซ้อนสูงจะได้หาแนวทางแก้ไขเพื่อลดความซับซ้อน ผลที่ได้ทำให้ซอฟแวร์ทำงานได้รวดเร็วมากขึ้น เพราะโปรแกรมมีความซับซ้อนลดลง ซึ่งมีผลดีอย่างมากกับกรณีที่ซอฟแวร์ที่มีความซับซ้อนสูงที่ต้องการเครื่องคอมพิวเตอร์ที่มีประสิทธิภาพสูงที่ราคาค่อนข้างสูงมาทำการประมวลผลซอฟแวร์ ถ้าลดความซับซ้อนของซอฟแวร์ลงด้วยการเปลี่ยนอัลกอริทึมหรือใช้แนวทางอื่นๆ แทนที่การหาเครื่องคอมพิวเตอร์ที่มีประสิทธิภาพสูงเพื่อประมวลผลซอฟแวร์ช่วยให้ลดค่าใช้จ่ายลงได้

$$v(G) = e - n + 2 \qquad (2.1)$$

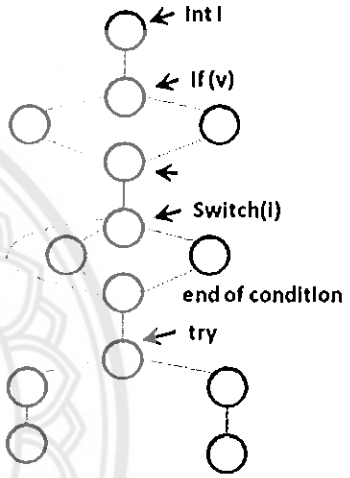
เมื่อ n – จำนวน Node ของกราฟ

e – จำนวน Edge ของกราฟ

ตารางที่ 2.1 ตารางเปรียบเทียบค่า Cyclomatic Complexity

ตาราง Cyclomatic Complexity	
Cyclomatic Complexity	**Risk Evaluation**
1-10	a simple program, without much risk
11-20	more complex, moderate risk
21-50	complex, high risk program
greater than 50	untestable program (very high risk)

```
1.  public class code {
2.    public void x(boolean v) {
3.      int i;
4.      if (v) { i = 1; }
5.      else { i = 2; }
6.      switch(i){
7.        case 1: ++i;
8.        case 2: --i;
9.        default:;
10.     }
11.     try { Boolean b = new Boolean(true);
12.     } catch (Exception e) {
13.             System.err.print(e);
14.     }
15.   }
16. }
```



รูปที่ 2.2 แสดงการนำ Source code มาสร้างเป็น Connected Graph [9]

จากรูปที่ 2.2 แสดงให้เห็นถึงตัวอย่างการนำ Source code มาสร้างเป็น Connected Graph เพื่อแสดงลำดับการทำงานของ Source code จากนั้น จึงนับจำนวนของ Node และ Edge ในกราฟ ตาม Statement ต่างๆ ในภาษา Java

2.1.3 Halstead Metrics

Halstead metrics คือ เครื่องมือในการวัดความซับซ้อนที่ถูกพัฒนาขึ้นมาเพื่อวัดความซับซ้อนของโปรแกรมโดยตรงจาก Source code โดยมีการกำหนดการกระทำในเชิงปริมาณของความซับซ้อนโดยตรงจาก Operators และ Operands ซึ่งการวัดในลักษณะนี้ทำให้รู้ถึงปริมาณความแตกต่างของตัวดำเนินการที่ใช้ใน Source code ว่ามีการใช้มากเพียงใด ถ้ามีการใช้ในปริมาณที่มากทำให้การทำความเข้าใจใน Source code นั้นยากตามไปด้วยเพราะการมีตัวดำเนินการมากให้เกิดความสับสน ซึ่งต่างกับ Source code ที่มีตัวดำเนินการน้อยกว่าทำให้เข้าใจการทำงานได้ง่ายกว่า

โดยการวัดคุณภาพด้วย Halstead Metrics มีพื้นฐานบนตัวดำเนินการ 4 ตัว ที่ได้มาโดยตรงจาก Source code ของโปรแกรม คือ

n1 = จำนวนของ Operators ซึ่งมีความแตกต่างกัน

n2 = จำนวนของ Operands ซึ่งมีความแตกต่างกัน

N1 = จำนวนทั้งหมดของ Operators

N2 = จำนวนทั้งหมดของ Operands

```
1.  public class code {
2.    public void x(boolean v) {
3.      int i;
4.      if (v) { i = 1; }
5.      else { i = 2; }
6.      switch(i){
7.        case 1: ++i;
8.        case 2: --i;
9.        default:;
10.     }
11.     try { Boolean b = new Boolean(true);
12.     } catch (Exception e) {
13.             System.err.print(e);
14.     }
15.   }
16. }
```

Line	N1	n1	N2	n2
2			v	v
4	if, =	If, =	v, i, 1	i, 1
5	=		i,2	2
6	switch	switch	i	
7	++	++	i	
8	--	--	i	
11	=,new	new	b,true	b,true
12			e	e
13	., ., print	., print	System, err, e	System, err
...	11	8	13	9

รูปที่ 2.3 ตัวอย่างการนับจำนวนของ Operators และ Operands [9]

จากรูปที่ 2.3 แสดงให้เห็นถึงตัวอย่างการนับจำนวนของ Operators และ Ooperands จาก Source code เพื่อให้ได้ค่าของตัวดำเนินการพื้นฐาน 4 ตัว คือ จำนวนของ Operators ซึ่งมีความแตกต่างกัน, จำนวนของ Operands ซึ่งมีความแตกต่างกัน, จำนวนทั้งหมดของ Operators และจำนวนทั้งหมดของ Operands ซึ่งตัวดำเนินการทั้ง 4 ตัวนี้สามารถแบ่งการวัดออกได้ 5 ลักษณะดังนี้

1) Halstead program Difficulty (D) คือ การวัดระดับความยากหรือแนวโน้มที่เกิดข้อผิดพลาดของโปรแกรมที่เกิดจากการกระทำระหว่างจำนวนของ Operator ที่ไม่ซ้ำกันและจำนวนทั้งหมดของ Operator ดังสมการที่ 2.2 (ถ้ามีการเรียก Operand ซ้ำกันหลายครั้งในโปรแกรม แล้วอาจมีแนวโน้มที่ทำให้เกิดข้อผิลพลาด)

$$D = (n1/2)*(N2/n2) \qquad (2.2)$$

2) Halstead program Length (N) คือ การวัดความยาวของ Source code โดยการหาผลรวมทั้งหมดของ Operators และ Operands ใน Source code ซึ่งสามารถคำนวณได้ดังสมการที่ 2.3

$$N = N1 + N2 \tag{2.3}$$

3) Halstead program Vocabulary (n) คือ การประเมินขนาดของโปรแกรมจากจำนวนของสิ่งที่ทำให้รู้และเข้าใจโปรแกรม ซึ่งคำนวณได้จาก [จำนวนของ Operators ซึ่งมีความแตกต่างกัน] + [จำนวนของ Operands ซึ่งมีความแตกต่างกัน] ดังสมการที่ 2.4

$$n = n1 + n2 \tag{2.4}$$

4) Halstead program Volume (V) เป็นการปริมาตรขนาดของโปรแกรม คำนวณได้ดังสมการที่ 2.5

$$V = N \log_2 n \tag{2.5}$$

เมื่อ N ได้มาจากสมการที่ 2.3

n ได้มาจากสมการที่ 2.4

5) Halstead program Effort (E) คือ ค่าที่บอกถึงความพยายามที่ต้องใช้ในการอ่านและทำความเข้าใจในโปรแกรม ซึ่งสามารถคำนวณได้จาก [Difficulty] * [Program Volume] คำนวณได้ดังสมการที่ 2.6

$$E = D * V \tag{2.6}$$

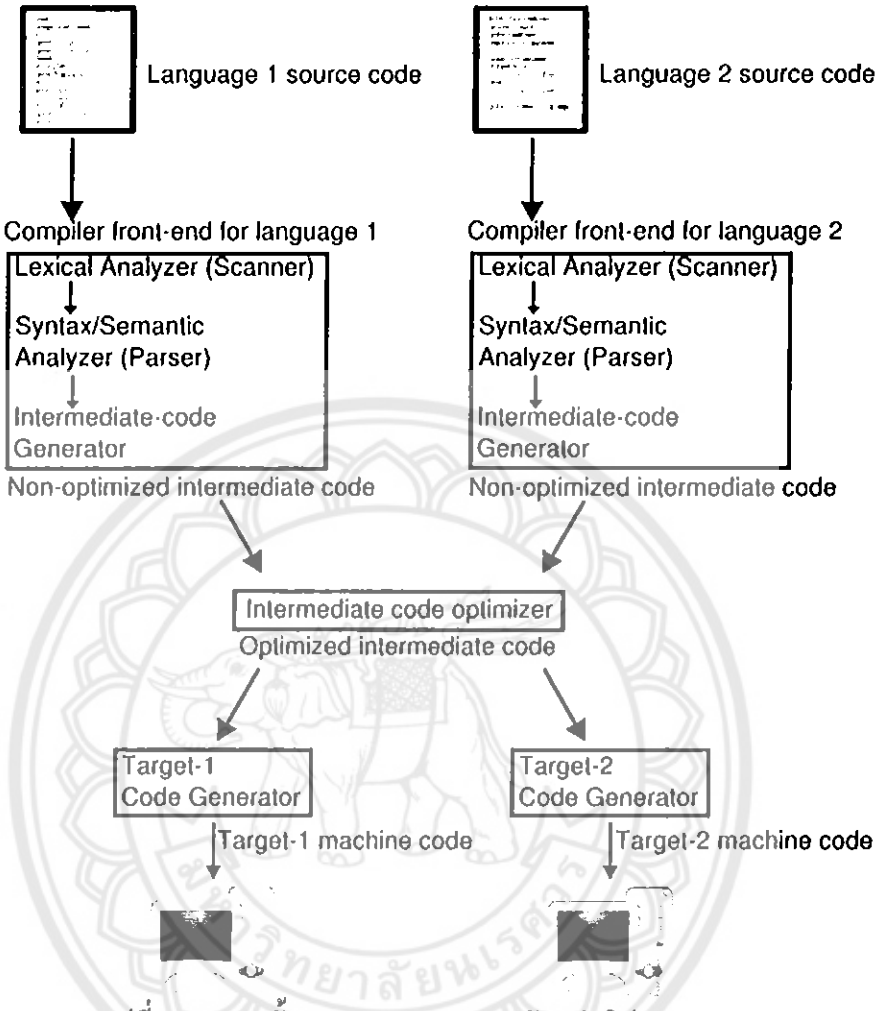
เมื่อ D ได้มาจากสมการที่ 2.2

V ได้มาจากสมการที่ 2.5

2.2 คอมไพเลอร์

คอมไพเลอร์เป็นโปรแกรมคอมพิวเตอร์ทำหน้าที่แปลงชุดคำสั่งภาษาคอมพิวเตอร์หนึ่ง ไปเป็นชุดคำสั่งที่มีความหมายเดียวกัน ในภาษาคอมพิวเตอร์อื่น คอมไพเลอร์ส่วนใหญ่ทำการแปล Source code ที่เขียนในภาษาระดับสูง เป็น ภาษาระดับต่ำ หรือภาษาเครื่อง ซึ่งคอมพิวเตอร์สามารถ

เข้าใจได้ จากรูปที่ 2.4 แสดงขั้นตอนการทำงานของคอมไพเลอร์โดยรับอินพุตไฟล์ Source code แล้วทำงานตามลำดับในแต่ละส่วนตามโครงสร้างของคอมไพเลอร์



รูปที่ 2.4 แสดงขั้นตอนการทำงานของตัวแปลโปรแกรม [10]

อย่างไรก็ตาม การแปลจากภาษาระดับต่ำเป็นภาษาระดับสูง ก็เป็นไปได้ โดยใช้ตัวแปลโปรแกรมย้อนกลับ (Decompiler)

2.2.1 โครงสร้างของคอมไพเลอร์

1. **Lexical Analyzer** ทำหน้าที่ในการอ่านอักขระจาก Source code แล้วแยกอักขระเหล่านั้นออกเป็นกลุ่มเรียกว่า Token กลุ่มอักขระใดที่แยกออกเป็น Token นั้นขึ้นอยู่กับภาษาของ Source code ที่กำหนดไว้เช่น Token ของภาษาระดับสูงทั่วไป ได้แก่ Keyword, Operator, Variable และ ค่า Constant ต่างๆ

2. **Syntax Analyzer** ทำหน้าที่ในการตรวจสอบว่า Token ที่ได้จากการแยกอักขระนั้น มีการเรียงลำดับถูกต้องตาม Syntax ของภาษาหรือไม่ เช่น เรียงกันเป็นนิพจน์ เป็นคำสั่ง IF เป็นคำสั่ง WHILE คำสั่ง FOR อย่างถูกต้องหรือไม่

3. **Semantic Analyzer** ทำหน้าที่ในการตรวจสอบความหมายและตีความเพื่อสร้าง Intermediate Code ที่ทำงานเหมือนกับ Source code

4. **Intermediate Code Generator Phase** เป็นขั้นตอนในการสร้าง Intermediate form หรือ รูปแบบที่เป็นกลาง เพื่อให้ง่ายต่อการแปลงไปเป็น Target language

5. **Code Optimizer** ทำหน้าที่ในการปรับปรุง Intermediate Code ที่ได้จากการตรวจสอบความหมายให้มีประสิทธิภาพมากขึ้น เช่น ขจัดการคำนวณที่ซับซ้อนและไม่มีความจำเป็นออกไป

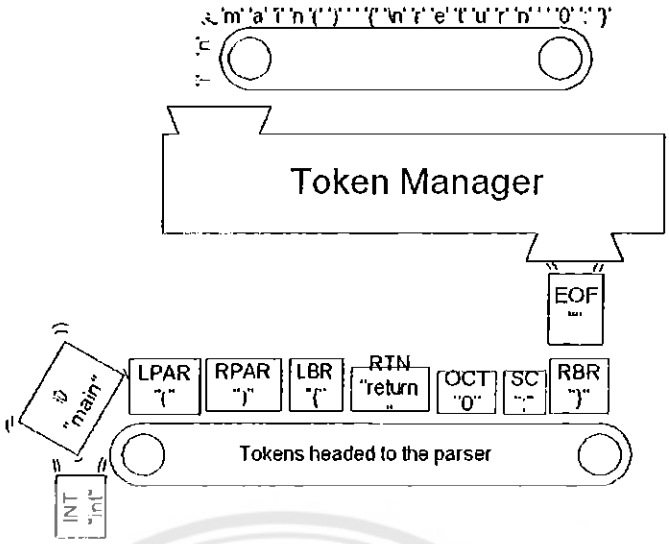
6. **Code generator** ทำหน้าที่ในการเปลี่ยน Intermediate Code ให้เป็นภาษาที่ต้องการ

2.3 Java Compiler Compiler (JavaCC)

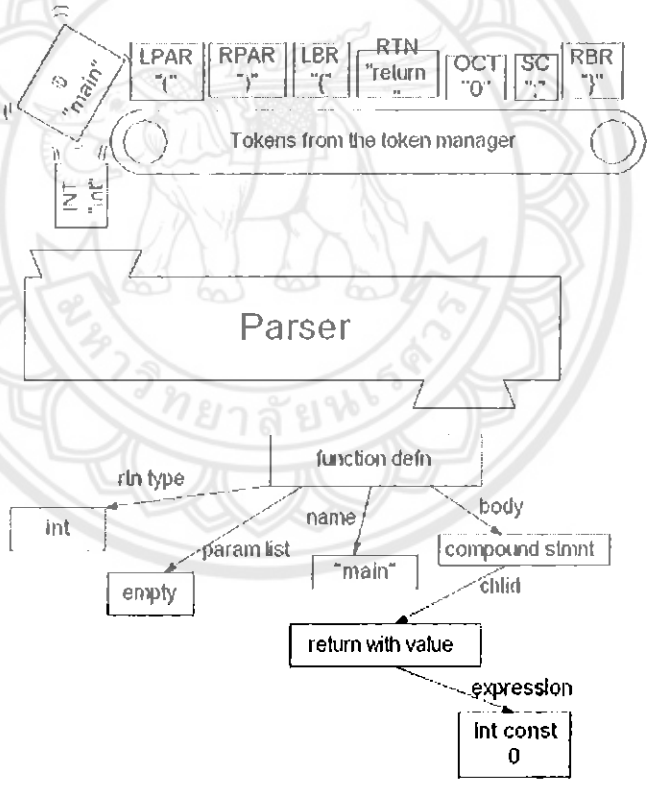
Java Compiler Compiler เป็นตัวสร้าง Parser generator สำหรับโปรแกรมภาษาจาวา Parser generator เป็นเครื่องมือที่ใช้ในการอ่านแกรมมาของภาษาและแปลง Code ของ JavaCC ไปเป็นโปรแกรมภาษา Java นอกจากนี้ Java compiler compiler ยังสามารถสร้าง Parser generator ในมาตราฐานอื่นๆ เช่น Tree actions debuggings เป็นต้น

1) ขั้นตอนการทำงานของคอมไพเลอร์ที่สร้างด้วย Java Compiler Compiler

คอมไพเลอร์ที่ถูกสร้างด้วย Java Compiler Compiler มี Token Manager สำหรับอ่านตัวอักขระ และการจัดกลุ่มของตัวอักขระที่เป็นชนิดเดียวกันและอยู่ติดกันไว้ด้วยกัน ซึ่งหลักการแบ่งชนิดของกลุ่มคำขึ้นอยู่กับ Regular Expression ที่กำหนดไว้



รูปที่ 2.5 ขั้นตอนการทำงานของ Token Manager [4]



รูปที่ 2.6 แสดงขั้นตอนการทำงานของ Parser [4]

จากรูปหลังจากตัวอักษรทั้งหมดได้ผ่านขั้นตอน Token Manager แล้วได้เป็นกลุ่มคำที่เรียกว่า Token หลังจากนั้นนำ Token ไปขั้นตอน Parser เพื่อกำหนดชนิดของ Token แต่ละ Token ดังรูป

บทที่ 3
วิธีการดำเนินงาน

ในบทนี้ได้กล่าวถึงขั้นตอนในการดำเนินงานของโครงงาน เพื่อให้ได้โปรแกรมในการตรวจวัด "Software Metrics" ได้แก่ Line Counting Metrics, Halstead Metrics และ Complexity Metrics จาก Source code ในภาษา Java ซึ่งขั้นตอนสำคัญในการดำเนินงาน คือ ขั้นตอนการศึกษาการเก็บรวบรวมข้อมูลขั้นตอนการทำความเข้าใจการใช้ JavaCC และขั้นตอนการนำทฤษฎีของ "Software Metrics" ที่ได้ศึกษามาเขียนโปรแกรมเพื่อตรวจวัด "Software Metrics"

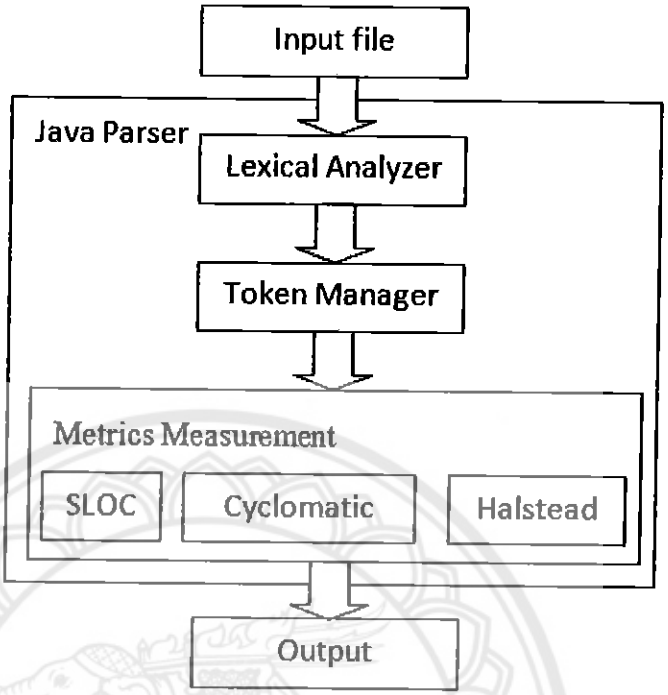
3.1 การออกแบบโปรแกรม

3.1.1 วิเคราะห์โครงสร้างเพื่อเตรียมเขียนโปรแกรม

เมื่อทำการวิเคราะห์หลักการของคอมไพเลอร์และขอบเขตของโครงงานนี้ทำให้ได้โครงสร้างของโปรแกรม ซึ่งสามารถแบ่งออกเป็นส่วนต่างๆดังนี้

- ส่วนรับอินพุตไฟล์
- ส่วนของ Lexical Analyzer
- ส่วนของ Token Manger
- ส่วนของ Metrics Measurement
- ส่วนการแสดงผลลัพธ์

สามารถเขียนรูปแสดงโครงสร้างของโปรแกรมได้ดังนี้

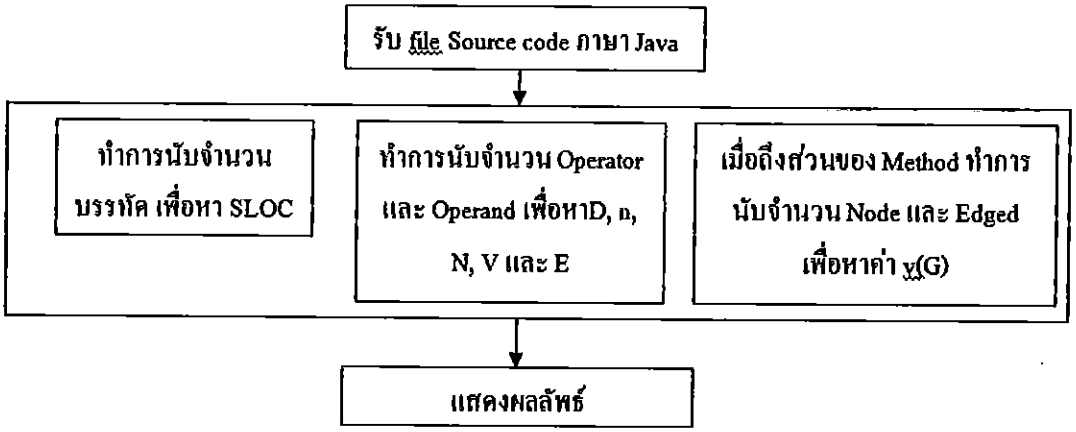


รูปที่ 3.1 โครงสร้างของโปรแกรม

3.1.2 วิเคราะห์ลำดับการทำงานของโปรแกรม

เมื่อได้โครงสร้างของโปรแกรมแล้ว สามารถวิเคราะห์ลำดับการทำงานของโปรแกรมได้ดังนี้

- รับไฟล์อินพุตซึ่งเป็น Source code ในภาษา Java
- ทำการตรวจวัด Line Counting Metrics, Halstead Metrics และ Complexity Metrics
- แสดงผลลัพธ์



รูปที่ 3.2 การทำงานของโปรแกรม

3.2 การเขียนโปรแกรม

การเขียนโปรแกรม Java Parser นั้นแบ่งเป็นส่วนต่างๆตามลักษณะโครงสร้างของโปรแกรมที่ได้ทำการวิเคราะห์แล้วแบ่งออกเป็น 2 ส่วน คือ

3.2.1 ส่วนของ Lexical Analyzer และ Token Manger

ในการเขียนโปรแกรมส่วนของ Lexical Analyzer และ Token Manger ได้ใช้โปรแกรม JavaCC เป็นตัวช่วยในการเพื่อสร้าง Lexical Analyzer และ Token Manger ของภาษา Java โดยใช้ไฟล์ JavaParser.jj ซึ่งเป็นไฟล์ที่มีไวยกรณ์ของภาษา Java version 1.5 ซึ่งเขียนขึ้นโดย Sreenivasa Viswanadha

3.2.2 ส่วนของ Metrics Measurement

ในส่วนนี้คือการเขียนโปรแกรมเพื่อตรวจวัด Line counting metrics, Halstead metrics และ Complexity metrics ซึ่งนำหลักการของ Software Metrics มาทำการประยุกต์ใช้เข้ากับการเขียนโปรแกรม และทำการสรุปหลักการต่างๆได้ดังนี้

3.2.2.1 Line counting metrics

Line counting metrics คือ การนับจำนวนบรรทัดของ Source code แบบ Logical โดยไม่นับ Blank line และบรรทัดที่เป็น Comment ซึ่งมีหลักในการนับดังนี้

1) Package Declaration

```
<modifiers> "package" <name of package> ";"  // นับ 1 บรรทัด
```

2) Import Declaration

```
"import" [ "static" ] <import name> [“.” | “*”] ";" // นับ 1 บรรทัด
```

3) Class or Interface Declaration

```
( "class" | "interface") <IDENTIFIER> [ TypeParameters() ]
[ ExtendsList() ] [ ImplementsList() ] ( ClassBody() |
InterfaceBody() )
// นับ 1 บรรทัด
```

4) Enum Body

```
"enum" <IDENTIFIER> [ ImplementsList() ]
{
<Something>
} // นับ 1 บรรทัด
```

5) Field Declaration

```
Type() VariableDeclarator() ( "," VariableDeclarator() )* ";"
// นับ 1 บรรทัด
```

6) Method Declaration

```
[ TypeParameters() ] ResultType()
  <IDENTIFIER> FormalParameters() ( "[" "]" )*
[ "throws" NameList() ]
(
{
        <Something>
}
|
";"
) // นับ 1 บรรทัด
```

7) Constructor Declaration

```
[ TypeParameters() ] ResultType()
  <IDENTIFIER> FormalParameters() ( "[" "]" )*
[ "throws" NameList() ]
{
        <Something>
} // นับ 1 บรรทัด
```

8) Explicit Constructor Invocation

```
( <IDENTIFIER> "." )* ["this" "." ] [ TypeArguments() ]
("this"|"super") Arguments() ";" // นับ 1 บรรทัด
```

9) Initializer

```
[ "static" ]
{
        <Something>
} // นับ 1 บรรทัด
```

10) Statement Expression

```
<Something> ";" // นับ 1 บรรทัด
```

11) Assertion Statement

```
"assert" <Something> ";" // นับ 1 บรรทัด
```

12) Local Variable Declaration

```
<Data  Type> ชื่อตัวแปร ซึ่งอาจมีการกำหนดค่าหรือไม่ก็ได้ ";"
// นับ 1 บรรทัด
```

13) Switch Statement

```
"switch" (<Something>) // นับ 1 บรรทัด
{
        "case" ":" <Something> // นับ 1 บรรทัด
        "default" ":" <Something> // นับ 1 บรรทัด
        "final" ":" <Something> // นับ 1 บรรทัด
}
```

14) If Statement

```
"if" (<something>) // นับ 1 บรรทัด
{
}
"else" "if" (<something>) // นับ 1 บรรทัด
{
}
"else" // นับ 1 บรรทัด
{
}
```

15) While Statement

```
"while" (<something>) // นับ 1 บรรทัด
{
}
```

16) Do While Statement

```
"do" // นับ 1 บรรทัด
{
}
"while" (<something>) ";"
```

17) For Statement

```
"for" (<something>) // นับ 1 บรรทัด
{
}
```

18) Break Statement

```
"break" <Something> ";" // นับ 1 บรรทัด
```

19) Continues Statement

```
"continue" <Something> ";" // นับ 1 บรรทัด
```

20) Return Statement

```
"return" <Something> ";" // นับ 1 บรรทัด
```

21) Throw Statement

```
"throw" <Something> ";"  // นับ 1 บรรทัด
```

22) Synchronized Statement

```
"synchronized"  <Something> ";"  // นับ 1 บรรทัด
```

23) Try Statement

```
"try"  // นับ 1 บรรทัด
{
}
"catch"( <Something> )
{
}
"finally"
{
}
```

3.2.2.2 Halstead metrics

1) **n1** - จำนวนของ Operators ซึ่งแตกต่างกัน

2) **n2** - จำนวนของ Operands ซึ่งแตกต่างกัน

3) **N1** - จำนวนทั้งหมดของ Operators

4) **N2** - จำนวนทั้งหมดของ Operands

5) **n** - Halstead program Vocabulary $n = n1 + n2$

6) **N** - Halstead program Length $N = N1 + N2$

7) **D** - Halstead program Difficulty $D = (n1/2)*(N2/n2)$

8) **V** - Halstead program Volume $V = N\log_2 n$

9) **E** - Halstead program Effort $E = D * V$

Operator ได้แก่

ตารางที่ 3.1 ตารางแสดง Operator

Loop คือ do while, while และ for	if	switch
New	assert	throw
Break	continue	=
<	!	~
? :	==	<=
>=	!=	||
&&	++	--
+	-	*
/	&	|
^	%	<<
+=	-=	*=
/=	&=	|=
^=	%=	<<=
>>=	>>>=	>>>
>>	>	.
การเรียกใช้ Method		

Operand ได้แก่

ตารางที่ 3.2 ตารางแสดง Operand

value ต่างๆของตัวแปร	ชื่อตัวแปรต่างๆ	true
false	null	this
super		

ในส่วนการเขียนโปรแกรม มีการจำแนก Operand และ Operator โดยใช้แนวคิดในลักษณะที่ว่า Operand นั้นเป็นตัวที่ถูกกระทำโดย Operator และในบางกรณีไม่สามารถระบุได้อย่างชัดเจน

ว่าส่วนใดเป็น Operand หรือ Operator จึงยึดหลักว่าหนึ่ง Keyword เป็นได้เพียง Operand หรือ Operator อย่างใดอย่างหนึ่งเท่านั้น

ตารางที่ 3.3 แสดงการจำแนก Operand และ Operator

Source code	คำอธิบาย
int a;	- "int" และ "a" ไม่เป็นทั้ง Operand และ Operator เพราะเป็นส่วนการประกาศตัวแปร
int a = 0;	- "int" และ "a" ไม่เป็นทั้ง Operand และ Operator เพราะคือการประกาศตัวแปร - "=" เป็น Operator - "0" เป็น Operand ที่ถูกกระทำโดย Operator "=" ซึ่งในส่วนของ
a = 1;	- "=" เป็น Operator - "a" และ "1" เป็น Operand ซึ่งกระทำกันโดย operator "="
BufferedReader bufferIP = new BufferedReader(InputStreamReader(System.in));	- "BufferedReader", "bufferIP" ไม่นับเพราะเป็นส่วนการประกาศตัวแปร - "=", "new" เป็น Operator - "BufferedReader" เป็น Operand ที่ถูกกระทำโดย Operator "new" - "InputStreamReader()" เป็น Operatorเพราะเป็นการเรียกใช้ method - "." เป็น Operator - "System", "in"เป็น Operand ที่ถูกกระทำโดย Operator "."
System.out.println("Hello World");	- "System", "out" และ ""Hello World"" เป็น Operand - ".", ".", "println()" เป็น Operator

หลังจากที่ทราบว่าอะไรจำแนกเป็น Operand และอะไรจำแนกเป็น Operator แล้วจึงทำการนับได้ แล้วจึงเก็บไว้ในค่าของตัวแปร n1, n2, N1 และ N2 เพื่อนำไปคำนวณหาค่าต่างๆได้

3.2.2.3 Complexity metrics

v(G) (Cyclomatic Complexity) คือ การวัดความซับซ้อนของ Source code ซึ่งคำนวณจากการนำ Source code มาสร้างเป็นกราฟแสดงการทำงานของ Source code แล้วคำนวณจากจำนวน Node และ Edge ของกราฟ แต่ในการเขียนโปรแกรมไม่ได้มีการสร้างกราฟเพราะการสร้างกราฟทำให้โปรแกรมทำงานช้ากว่าการที่มีรูปแบบการนับจำนวน Node และ Edge ตามลักษณะของแต่ละ Statement ซึ่งสามารถคำนวณหาค่าของ v(G) (Cyclomatic Complexity) ได้ดังสมการนี้

$$v(G) = e - n + 2 \qquad (3.1)$$

เมื่อ n – จำนวน Node ของกราฟ

e – จำนวน Edged ของกราฟ

ในการเขียนโปรแกรมให้นับ n และ e ตามลักษณะของแต่ละ Statement เพื่อใช้ในการคำนวณหา v(G) ได้ดังนี้

1) Statement Expression

<Something> ";" นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1

2) Assertion Statement

"assert" <Something> ";" นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1

3) Local Variable Declaration

<Data Type> ชื่อตัวแปร ซึ่งอาจมีการกำหนดค่าหรือไม่ก็ได้ ";" นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1

4) Synchronized Statement

"synchronized" <Something> ";" นับ e เพิ่มขึ้น 1 และนับ n

5) Break Statement

"break" <Something> ";" นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1

6) Continues Statement

"continue" <Something> ";" นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1

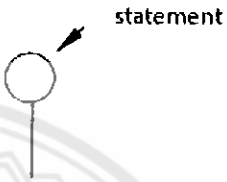
7) Return Statement

"return" <Something> ";" นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1

8) Throw Statement

"throw" <Something> ";" นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1

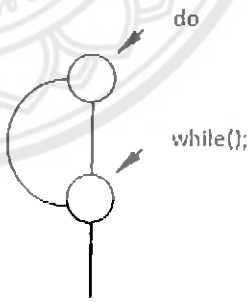
จาก Statement ที่นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1 นั้นสามารถนำมาสร้างเป็นกราฟของ Node และ Edge ได้ดังรูปที่ 3.3



รูปที่ 3.3 แสดงการสร้างกราฟของ Statement ที่นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1

9) Do While Statement

"do" นับ e เพิ่มขึ้น 2 และนับ n เพิ่มขึ้น 1
{
}
"while" () ";"



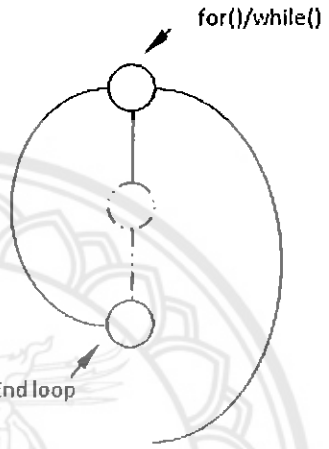
รูปที่ 3.4 แสดงการสร้างกราฟของ Do While Statement

10) For Statement

"for" () นับ e เพิ่มขึ้น 3 และนับ n เพิ่มขึ้น 2
{
}

11) While Statement

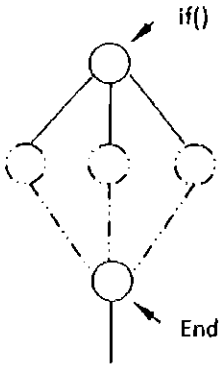
"while" () นับ e เพิ่มขึ้น 3 และนับ n เพิ่มขึ้น 2
{
}



รูปที่ 3.5 แสดงการสร้างกราฟของ For Statement และ While Statement

12) If Statement

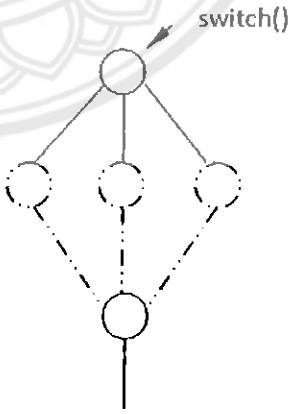
"if" () นับ e เพิ่มขึ้น 3 และนับ n เพิ่มขึ้น 2
{
}
"else" "if" () นับ e เพิ่มขึ้น 1
{
}
"else" ไม่นับเนื่องจากมีการนับรวมอยู่ใน บรรทัด if แล้ว
{
}



รูปที่ 3.6 แสดงการสร้างกราฟของ If Statement

13) Switch Statement

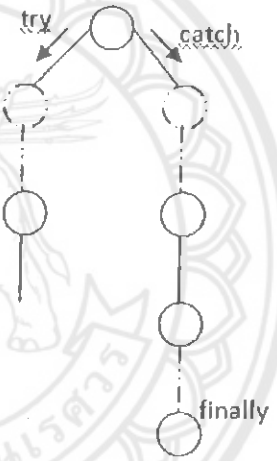
"switch" () นับ e เพิ่มขึ้น 2 และนับ n เพิ่มขึ้น 2
{
"case" : นับ e เพิ่มขึ้น 1
"case" : นับ e เพิ่มขึ้น 1
"default" : ไม่นับเนื่องจากมีการนับรวมอยู่ใน บรรทัด switch แล้ว
}



รูปที่ 3.7 แสดงการสร้างกราฟของ Switch Statement

14) Try Statement

```
try นับ e เพิ่มขึ้น 3 และนับ n เพิ่มขึ้น 3
{
}
catch() ไม่นับเนื่องจากมีการนับรวมอยู่ในบรรทัดเดียวกับ try แล้ว
{
}
finally นับ e เพิ่มขึ้น 1 และนับ n เพิ่มขึ้น 1
{
}
```



รูปที่ 3.8 แสดงการสร้างกราฟของ Try Statement

หลังจากที่โปรแกรมทำการนับด้วยวิธีดังกล่าวเสร็จแล้วโปรแกรมได้ทำการลบค่า Edge ไป 1 เพราะวิธีในการนับของโปรแกรมนั้นเมื่อพบ 1 คำสั่งปกติได้ทำการนับจำนวน Edge เพื่อโยงไปอีกคำสั่งถัดไปเสมอ เพราะฉะนั้นต้องทำการลบ 1 เพราะในคำสั่งสุดท้ายจึงทำการนับ Edge เกินไป 1 ครั้ง หลังจากนั้นนำค่า Edge และ Node ที่ได้ไปคำนวณหาค่า v(G) ต่อไป

บทที่ 4

ผลการทดลอง

ในบทนี้กล่าวถึงการทดสอบ และการแสดงผลของโปรแกรมสำหรับวิเคราะห์ความซับซ้อนของ Source code โดยขั้นตอนการทดสอบโปรแกรมแบ่งออกเป็น 2 ส่วน ส่วนแรกเป็นการเปรียบเทียบกันระหว่าง 2 โปรแกรมโดยใช้โปรแกรมที่ให้ผลลัพธ์เดียวกันมาทำการทดสอบ ส่วนที่สองเป็นการทดสอบโปรแกรมที่ออกแบบโดยใช้ Design Pattern ชนิดต่างๆมาทำการทดสอบว่า Design Pattern ชนิดเดียวกันมีความซับซ้อนแตกต่างกันหรือไม่

4.1 การเปรียบเทียบกันระหว่าง 2 โปรแกรมโดยใช้โปรแกรมที่ให้ผลลัพธ์เดียวกัน

ทดสอบโปรแกรมที่ให้ผลลัพธ์เดียวกันของ 2 โปรแกรมแรกเป็นโปรแกรมที่เขียนโดยมีรูปแบบการเขียนที่แตกต่างกันเห็นได้จากลักษณะของ Source Code ที่แตกต่างกัน ซึ่งอาจมีการใช้คำสั่งที่เหมือนและแตกต่างกันออกไปดังรูป การทำงานของโปรแกรมทั้งสองนี้สร้าง Socket เพื่อทำเชื่อมต่อระหว่าง Client กับ Server เหมือนกัน และให้ผลลัพธ์ที่เหมือนกัน

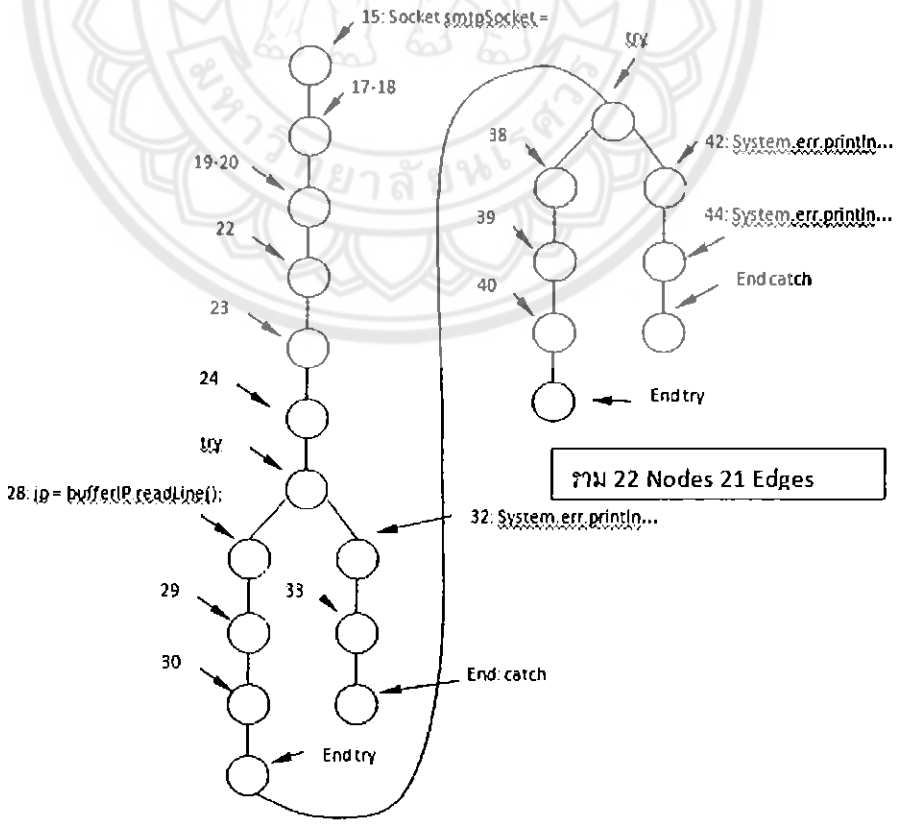
```
1  /**
2   * Socket Client in java
3   * @author Naruebet Hanrak
4   * @date 2007 September 7
5   * @note JDk 1.6.0 update 2
6   */
7  import java.io.*;                                          // นับเป็น 1 บรรทัด
8  import java.net.*;                                         // นับเป็น 1 บรรทัด
9  public class Client {                                      // นับเป็น 1 บรรทัด
10     public static void main(String[] args) {               // นับเป็น 1 บรรทัด
11         // declaration section:
12         // smtpClient: our client socket
13         // bufferIP: input stream
14         // bufferPort: input stream
15         Socket smtpSocket = null;                          // นับเป็น 1 บรรทัด
16         BufferedReader bufferIP = new BufferedReader(
17             new InputStreamReader(System.in));             // นับเป็น 1 บรรทัด
18         BufferedReader bufferPort = new BufferedReader(
19             new InputStreamReader(System.in));             // นับเป็น 1 บรรทัด
20
21         // Initialization section:
22         System.out.print("Enter IP Address: ");           // นับเป็น 1 บรรทัด
23         String ip = null;                                  // นับเป็น 1 บรรทัด
24         int port = 0;                                      // นับเป็น 1 บรรทัด
25
```

รูปที่ 4.1 แสดง Source code ที่อยู่ใน Main Program

```
26          // Try to Initialization Internet Addressing and Port number
27          try {                                                          // นับเป็น 1 บรรทัด
28             ip = bufferIP.readLine();                                   // นับเป็น 1 บรรทัด
29             System.out.print("Enter Port Number: ");                    // นับเป็น 1 บรรทัด
30             port = Integer.parseInt(bufferPort.readLine());             // นับเป็น 1 บรรทัด
31          } catch (IOException ice) {
32             System.err.println("IO error trying to read your IP Address!");  // นับเป็น 1 บรรทัด
33             System.exit(1);                                             // นับเป็น 1 บรรทัด
34          }
35
36          // Try to open a socket on Internet Addressing and port number
37          try {                                                          // นับเป็น 1 บรรทัด
38              smtpSocket = new Socket(ip, port);                         // นับเป็น 1 บรรทัด
39              System.out.println("Connected to the server.");            // นับเป็น 1 บรรทัด
40              smtpSocket.close();                                        // นับเป็น 1 บรรทัด
41          } catch (UnknownHostException e) {
42              System.err.println("Don't know about host: hostname");     // นับเป็น 1 บรรทัด
43          } catch (IOException e) {
44              System.err.println(
45                  "Couldn't get I/O for the connection to: hostname");   // นับเป็น 1 บรรทัด
46          }
47      }                                                                  // นับรวมเป็น 22 บรรทัด
48  }
```

รูปที่ 4.1 (ต่อ) แสดง Source code ที่อยู่ใน Main Program

ในรูปที่ 4.1 เป็น Source code ที่เขียนคำสั่งต่างๆอยู่ใน Main Program เพียงอย่างเดียว โดยไม่มีการแยกคำสั่งออกเป็น Method และแสดงการนับจำนวนบรรทัดด้วยมือซึ่งได้จำนวนบรรทัดทั้งหมด 22 บรรทัด และในรูปที่ 4.2 แสดงจำนวนของ Node และ Edge ที่แปลงจาก Source code



รูปที่ 4.2 แสดงการแปลง Source code ที่เขียนคำสั่งต่างๆอยู่ใน Main Program เป็นกราฟ

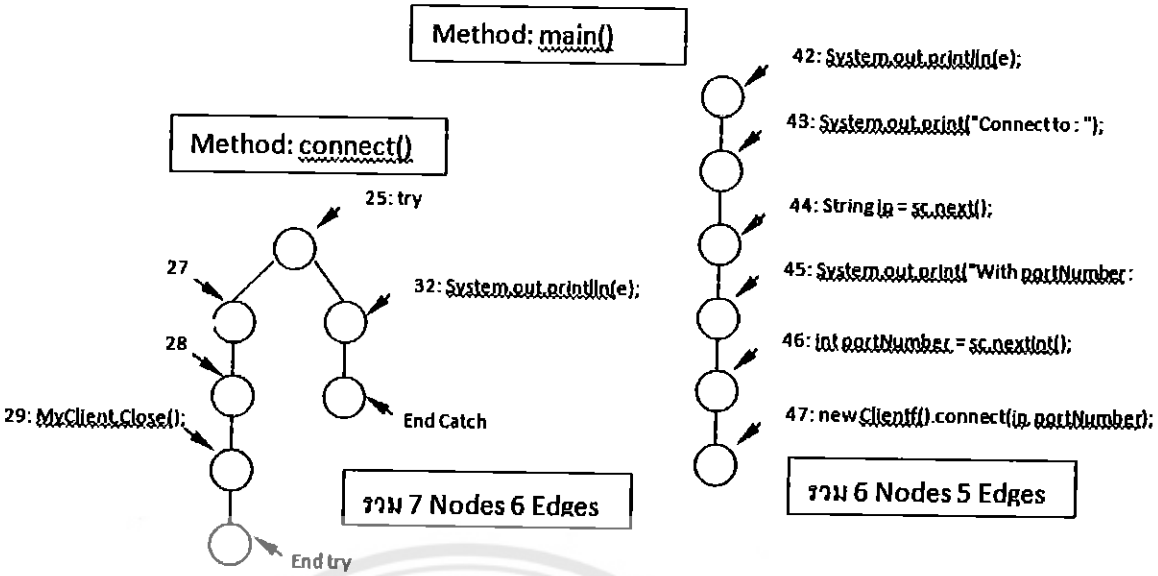
ตารางที่ 4.1 แสดงการจำแนก Operator และ Operand ของ Source code ที่เขียนคำสั่งต่างๆอยู่ใน Main Program

LN	N1	n1	N2	n2
15	=	=	null	Null
16	=, new	new	BufferedReader	BufferedReader
17	new, .	.	InputStreamReader, System, in	InputStreamReader , System, in
18	=, new		BufferedReader	F
19	new, .		InputStreamReader, System, in	
22	., ., print	print	System , out, "Enter IP Address: "	out, "Enter IP Address: "
23	=		null	
24	=		0	0
28	=, . readLine	readLine	ip, bufferIP	ip, bufferIP
29	., ., print		System , out, "Enter Port Number: "	"Enter Port Number: "
30	=, ., parseInt, ., readLine	parseInt	port, Integer, bufferPort	port, Integer, bufferPort
32	., ., println	println	System , err, "IO error trying to read your IP Address!"	err, "IO error trying to read your IP Address!"
33	., exit	exit	System , 1	1
38	=, new		smtpSocket, Socket, ip, port	smtpSocket, Socket
39	., ., println		System , out, "Connected to the server."	"Connected to the server."
40	., close	close	smtpSocket	
42	., ., println		System , err, "Don't know about host: hostname"	"Don't know about host: hostname"
44	., ., println		System , err,	
45			"Couldn't get I/O for the connection to: hostname"	"Couldn't get I/O for the connection to: hostname"
รวม	43	9	41	22

ในรูปที่ 4.3 ที่มีการแยกชุดคำสั่งออกเป็น Method และแสดงการนับบรรทัดจากการคำนวณด้วยมือได้จำนวนบรรทัดทั้งหมด 17 บรรทัด

```
 1  import java.io.IOException;                                    // นับเป็น 1 บรรทัด
 2  import java.net.Socket;                                        // นับเป็น 1 บรรทัด
 3  import java.util.Scanner;                                      // นับเป็น 1 บรรทัด
 4
 5  /*
 6  * @Description : Client connect to the server
 7  * @Author : 47361878 Chatchai Luangmanee
 8  * @Date : 2007 September 7
 9  * @Note : Java Eclipse JDK_1.5.0 (Windows vista Home Premium)
10  */
11  public class Client                                            // นับเป็น 1 บรรทัด
12  {
13      /*
14       * @Description Connect to server
15       * @param ip that server's ip
16       * @param portNumber that server's portNumber
17       * @return void
18       * @contract connect : String int -> void
19       * @example Client().connect("127.0.0.1" , 3)
20       * @example Client().connect("192.168.0.5" , 8888)
21       * @example Client().connect("192.168.50.142" , 20000)
22       */
23      public void connect(String ip,int portNumber)              // นับเป็น 1 บรรทัด
24      {
25          try                                                    // นับเป็น 1 บรรทัด
26          {
27              Socket MyClient = new Socket(ip, portNumber);      // นับเป็น 1 บรรทัด
28              System.out.println("Connected to the server.");  // นับเป็น 1 บรรทัด
29              MyClient.close();                                  // นับเป็น 1 บรรทัด
30          }
31          catch (IOException e) {
32              System.out.println(e);                             // นับเป็น 1 บรรทัด
33          }
34      }
35      /**
36       * MAIN
37       * @param args
38       */
39      public static void main(String[] args)                     // นับเป็น 1 บรรทัด
40      {
41          //TODO Auto-generated method stub
42          Scanner sc = new Scanner(System.in);                   // นับเป็น 1 บรรทัด
43          System.out.print("Connect to : ");                     // นับเป็น 1 บรรทัด
44          String ip = sc.next();                                 // นับเป็น 1 บรรทัด
45          System.out.print("With portNumber : ");               // นับเป็น 1 บรรทัด
46          int portNumber = sc.nextInt();                         // นับเป็น 1 บรรทัด
47          new Clientf().connect(ip, portNumber);                 // นับเป็น 1 บรรทัด
48      }
49                                                                 // นับรวมเป็น 17 บรรทัด
50  }
51
```

รูปที่ 4.3 แสดง Source code ที่มีการแยกคำสั่งต่างๆออกเป็น Method [1]

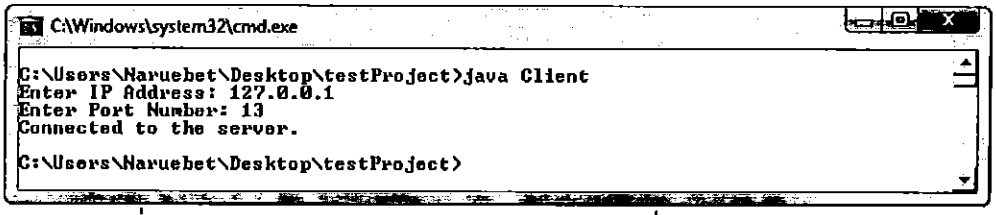


รูปที่ 4.4 แสดงการแปลง Source code ที่มีการแยกคำสั่งต่างๆออกเป็น Method เป็นกราฟ

ตารางที่ 4.2 แสดงการจำแนก Operator และ Operand ของ Source code ที่เขียนคำสั่งต่างๆ แยกออกเป็น Method

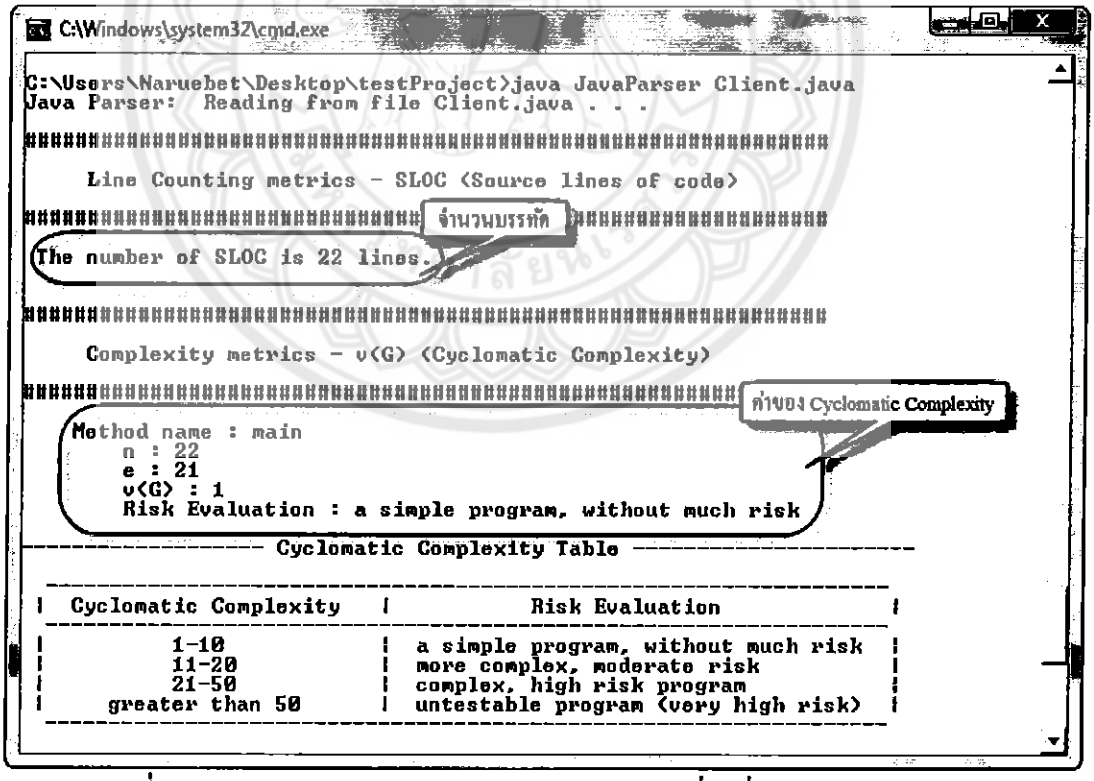
LN	N1	n1	N2	n2
27	=, new	=, new	Socket, ip, portNumber	Socket, ip, portNumber
28	., ., println	., println	System, out, "Connected to the server."	System, out, "Connected to the server."
29	., close	close	MyClient	MyClient
32	., ., println		System, out, e	E
42	=, new, .		Scanner, System, in	Scanner, in
	., ., println		System, out, "Connect to : "	"Connect to : "
44	=, ., next	next	sc	sc
45	., ., println		System, out, "With portNumber : "	"With portNumber : "
46	=, ., nextInt	nextInt	sc	
47	new, ., connect		Clientf, ip, portNumber	Clientf
รวม	28	7	24	14

เมื่อนำ Source code ทั้งสองนี้มาทำการคอมไพล์และประมวลผลโปรแกรมจะเห็นได้ว่าทั้งสองโปรแกรมได้ผลลัพธ์ที่เหมือนกันคือทำการเชื่อมต่อกับ Server โดยที่ทั้งสองโปรแกรมรับค่าของ IP Address และหมายเลข Port ที่ใช้เชื่อมต่อ ดังรูปที่ 4.5 และ 4.6

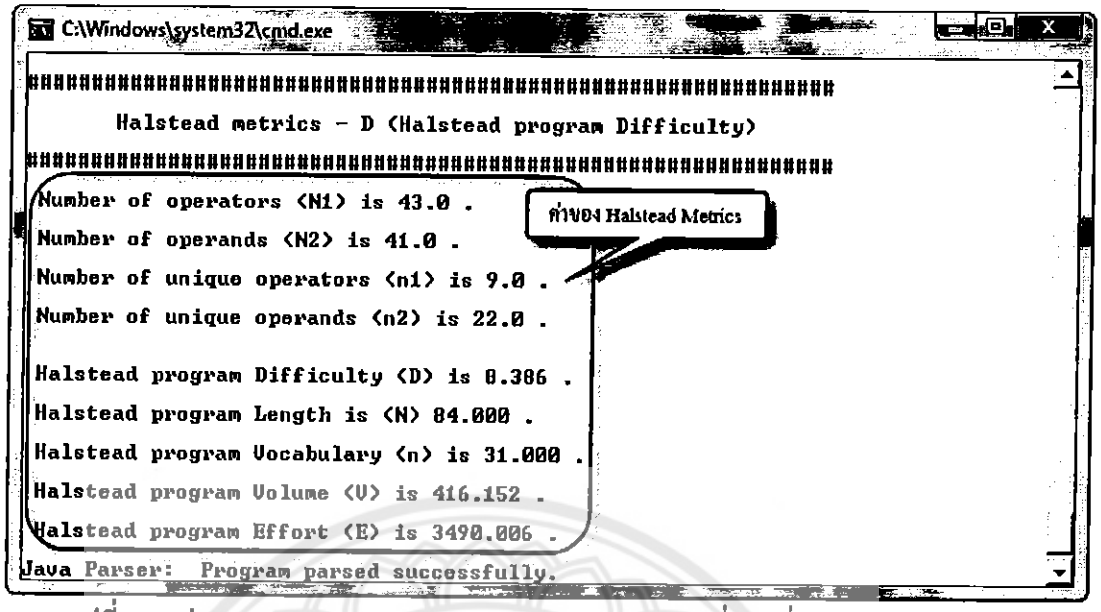


รูปที่ 4.5 ผลการ Run Source Code ของโปรแกรมที่ อยู่ใน Main Program

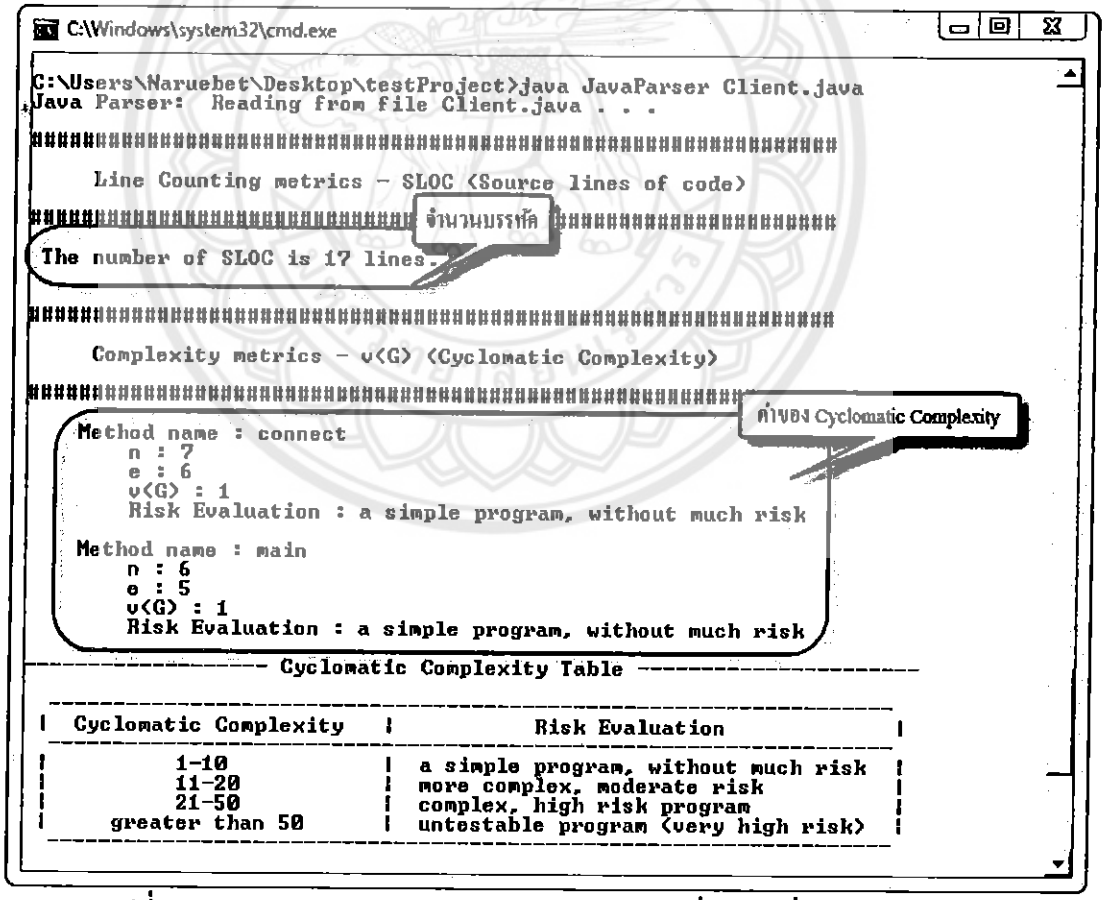
รูปที่ 4.6 ผลการ Run Source Code ของโปรแกรมที่มีการแยกคำสั่งออกเป็น Method



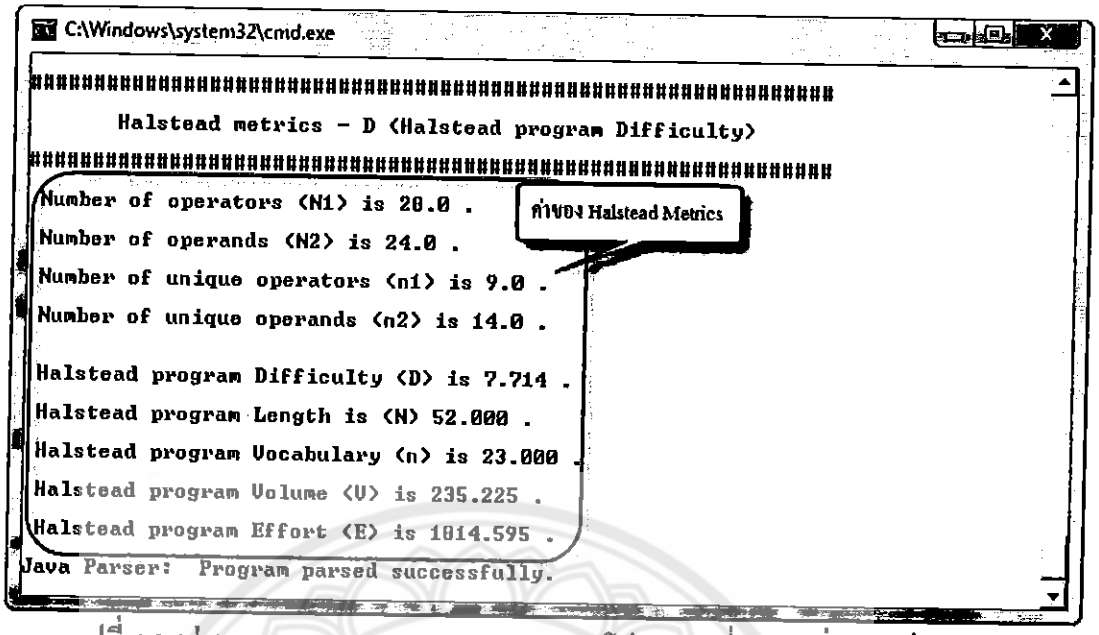
รูปที่ 4.7 ผลการทดสอบ Source Code ของโปรแกรมที่คำสั่งอยู่ใน Main Program



รูปที่ 4.7 (ต่อ) ผลการทดสอบ Source Code ของโปรแกรมที่มีคำสั่งอยู่ใน Main Program



รูปที่ 4.8 ผลการทดสอบ Source Code ของโปรแกรมที่แยกคำสั่งออกเป็น Method



รูปที่ 4.8 (ต่อ) ผลการทดสอบ Source Code ของโปรแกรมที่แยกคำสั่งออกเป็น Method

ในการเปรียบเทียบกันระหว่าง 2 โปรแกรมโดยใช้โปรแกรมที่ให้ผลลัพธ์เดียวกันระหว่าง Source Code ของโปรแกรมที่มีคำสั่งอยู่ใน Main Program (โปรแกรมที่ 1) และ Source Code ของโปรแกรมที่มีการแยกคำสั่งออกเป็น Method (โปรแกรมที่ 2) ได้ผลการทดสอบในส่วนของ Line Counting Metrics และ Halstead Metrics ดังตารางที่ 4.3

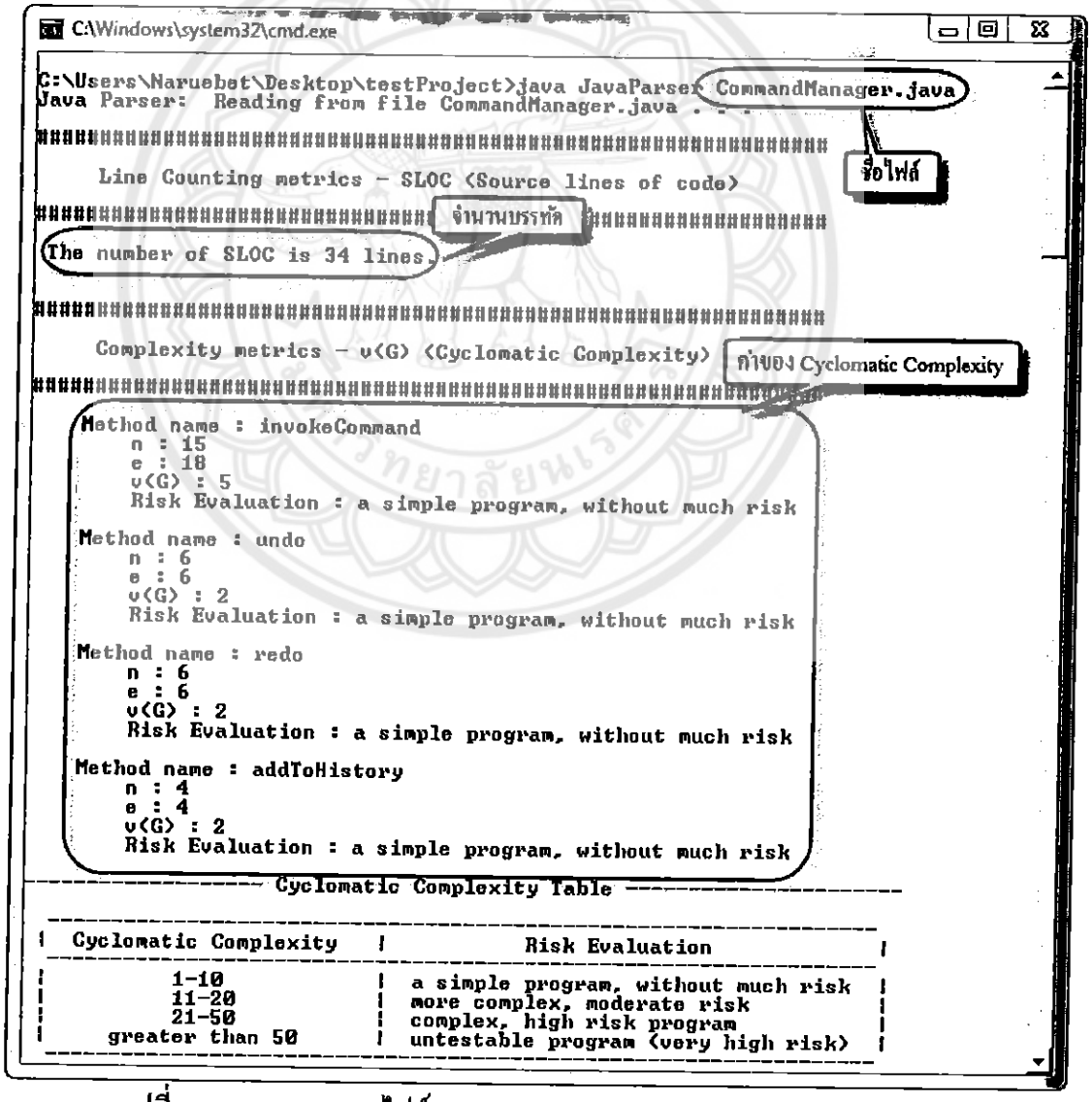
ตารางที่ 4.3 ตารางเปรียบเทียบผลการทดสอบของ โปรแกรมที่ 1และ โปรแกรมที่ 2

	โปรแกรมที่ 1	โปรแกรมที่ 2
SLOC	22	17
Number of operators (N1)	43.0	28.0
Number of operands (N2)	41.0	24.0
Number of unique operators (n1)	9.0	9.0
Number of unique operands (n2)	22.0	14.0
Halstead program Difficulty (D)	8.386	7.714
Halstead program Length (N)	84.0	52.0
Halstead program Vocabulary (n)	31.0	23.0
Halstead program Volume (V)	416.152	235.225
Halstead program Effort (E)	3490.006	1814.595

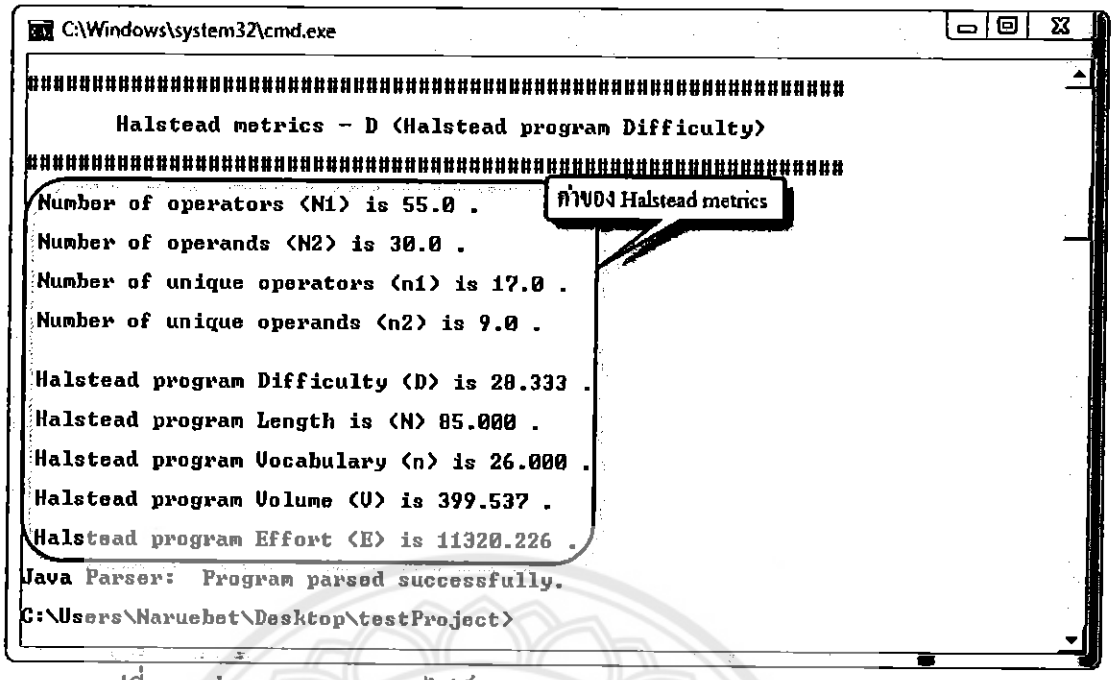
จากตารางแสดงให้เห็นว่าโปรแกรมที่ 1 มีขนาดใหญ่กว่า โปรแกรมที่ 2 และโปรแกรมที่ 1 มีค่าต่างๆ ของ Halstead Metrics มากกว่า โปรแกรมที่ 2 ไม่ว่าเป็นค่าของ Halstead program Difficulty ที่แสดงให้เห็นว่าโปรแกรมที่ 1 มีความซับซ้อนมากกว่าโปรแกรมที่ 2 ทำให้โปรแกรมที่ 1 มีความเสี่ยงที่เกิดข้อผิดพลาดมากกว่าโปรแกรมที่ 2

4.2 การทดสอบโปรแกรมใน Design Pattern ชนิดต่างๆ

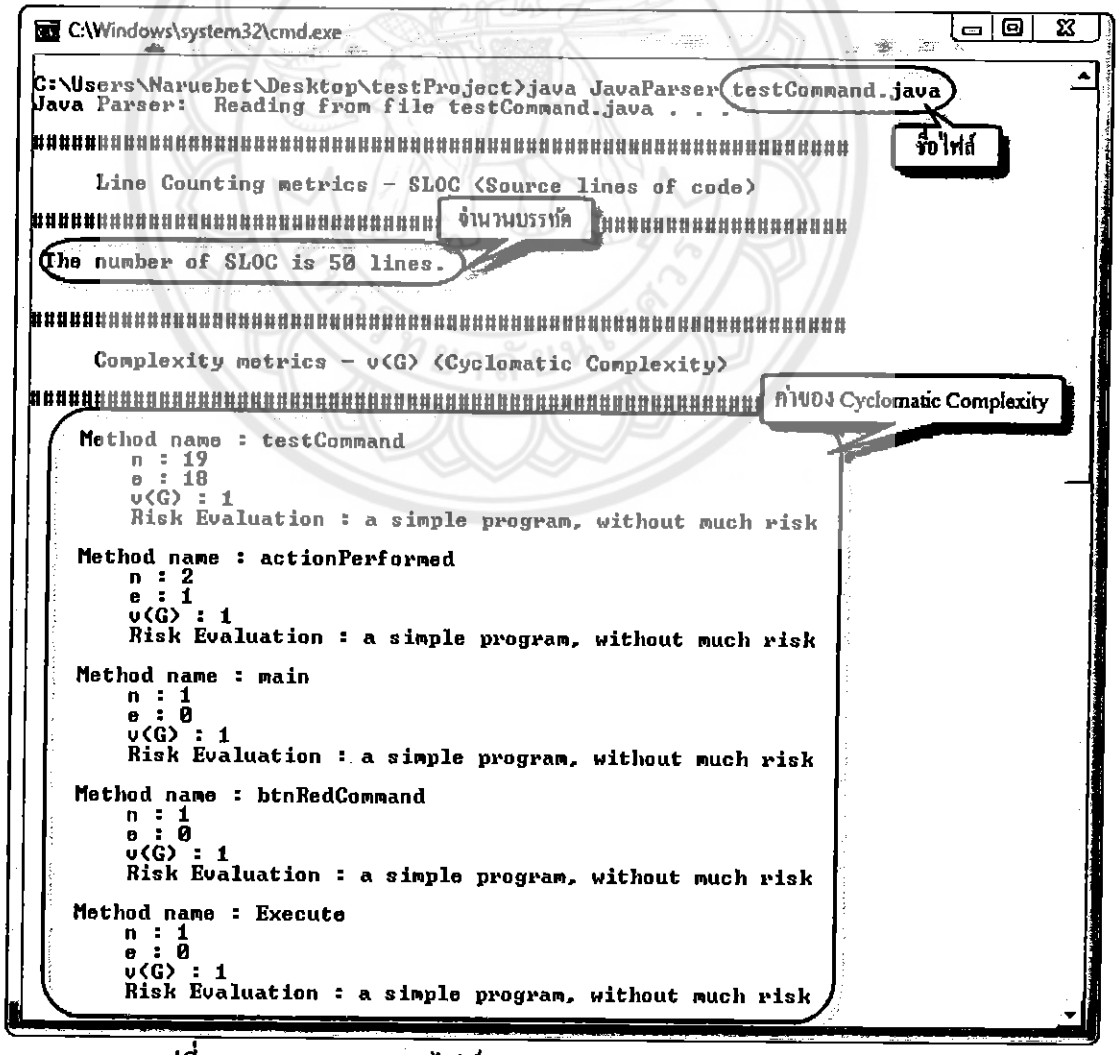
ส่วนนี้เป็นการนำ Source Code ใน Pattern ต่างๆมาทำการทดสอบ โดยในที่นี้นำมาทดสอบ 3 Pattern คือ Command Pattern, Composite Pattern และ Visitor Pattern ซึ่งผลการทดสอบโปรแกรมแสดงผลลัพธ์จำนวน 1 รูป ต่อการทดสอบ 1 โปรแกรม โดยรูปแสดงชื่อไฟล์ที่ใช้ทดสอบและผลลัพธ์ของการทดสอบ



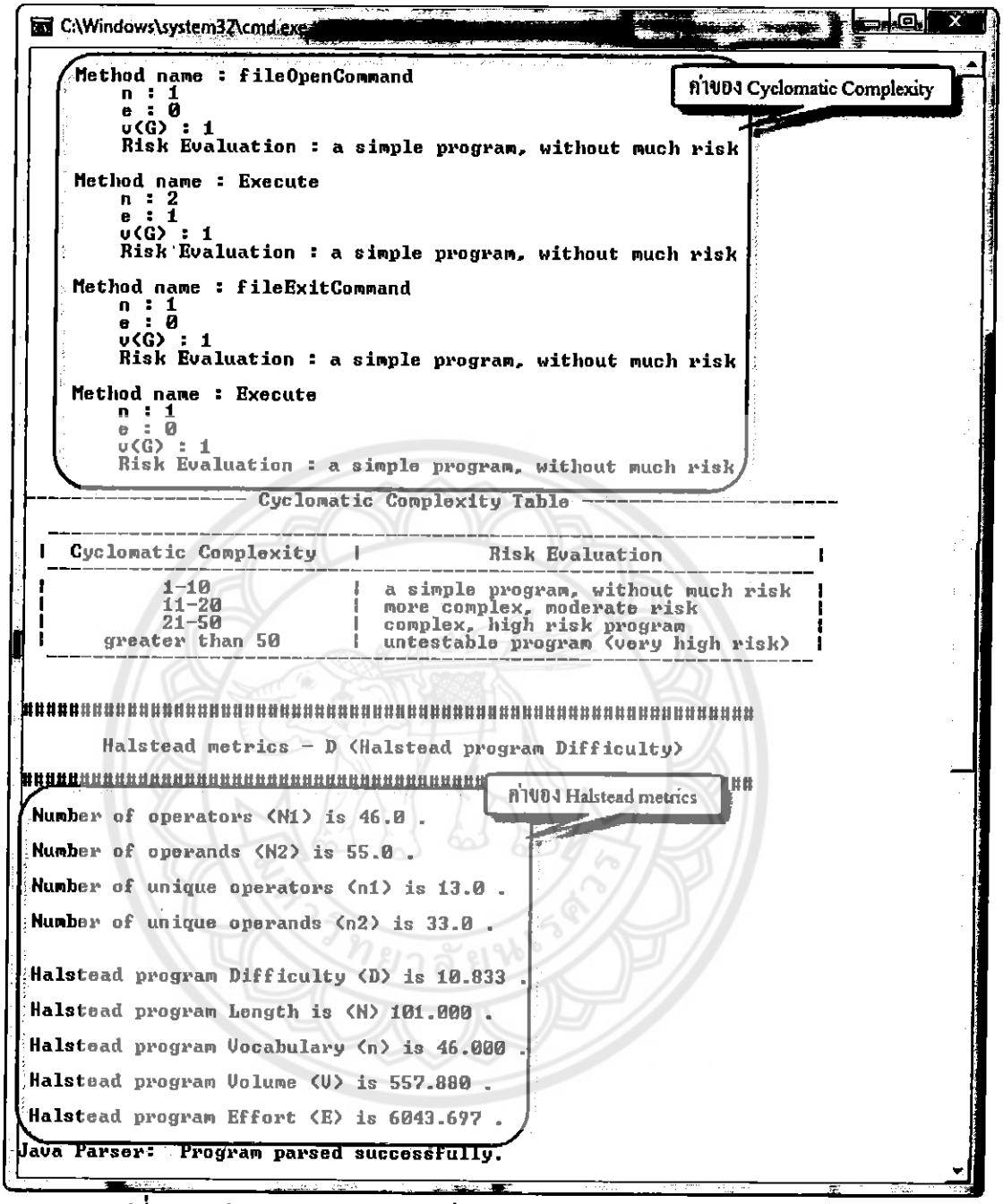
รูปที่ 4.9 ผลการทดสอบไฟล์ CommandManager.java ของ Command Pattern



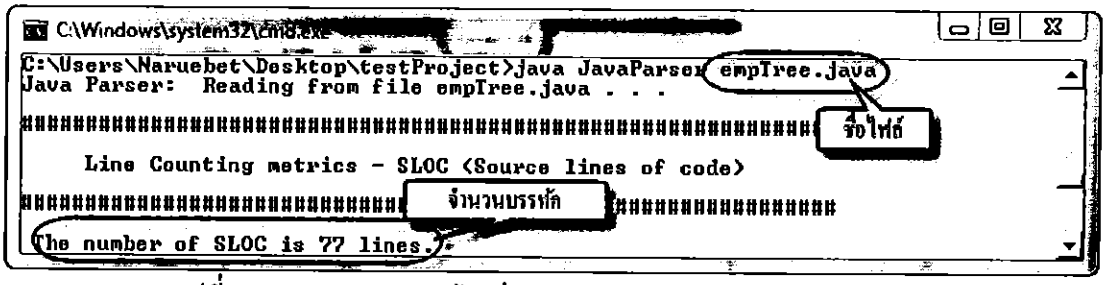
รูปที่ 4.9 (ต่อ) ผลการทดสอบไฟล์ CommandManager.java ของ Command Pattern



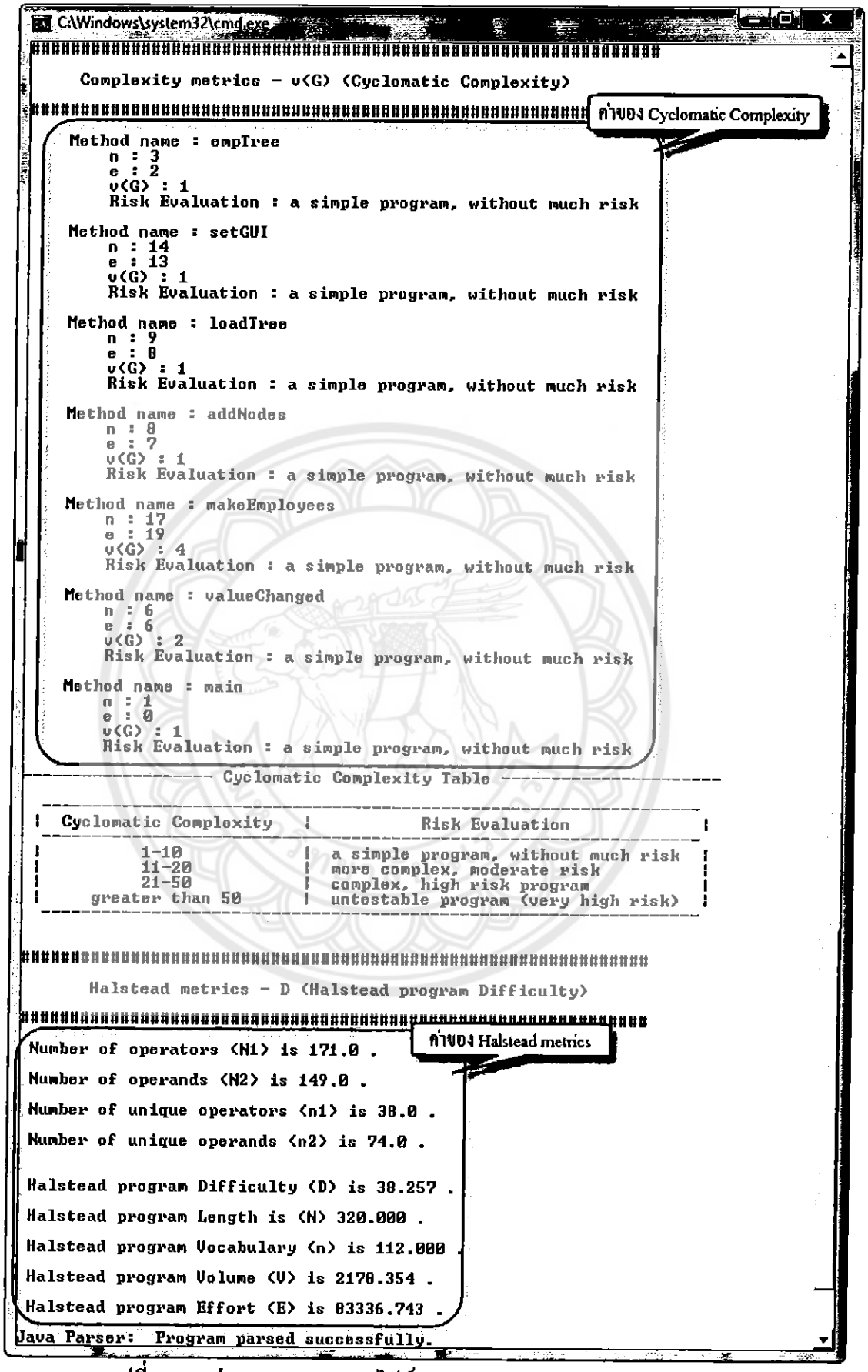
รูปที่ 4.10 ผลการทดสอบไฟล์ testCommand.java ของ Command Pattern



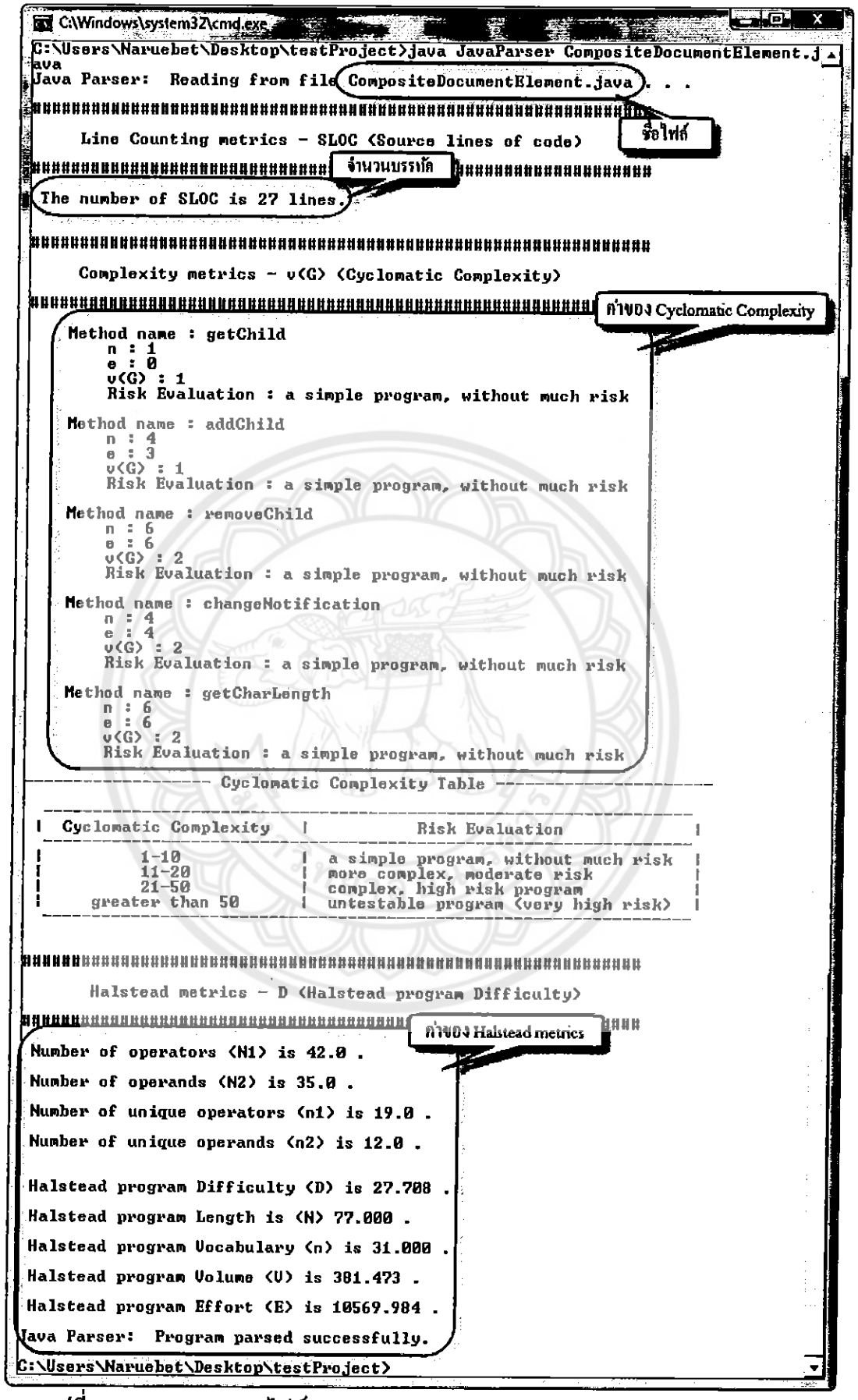
รูปที่ 4.10 (ต่อ) ผลการทดสอบไฟล์ testCommand.java ของ Command Pattern



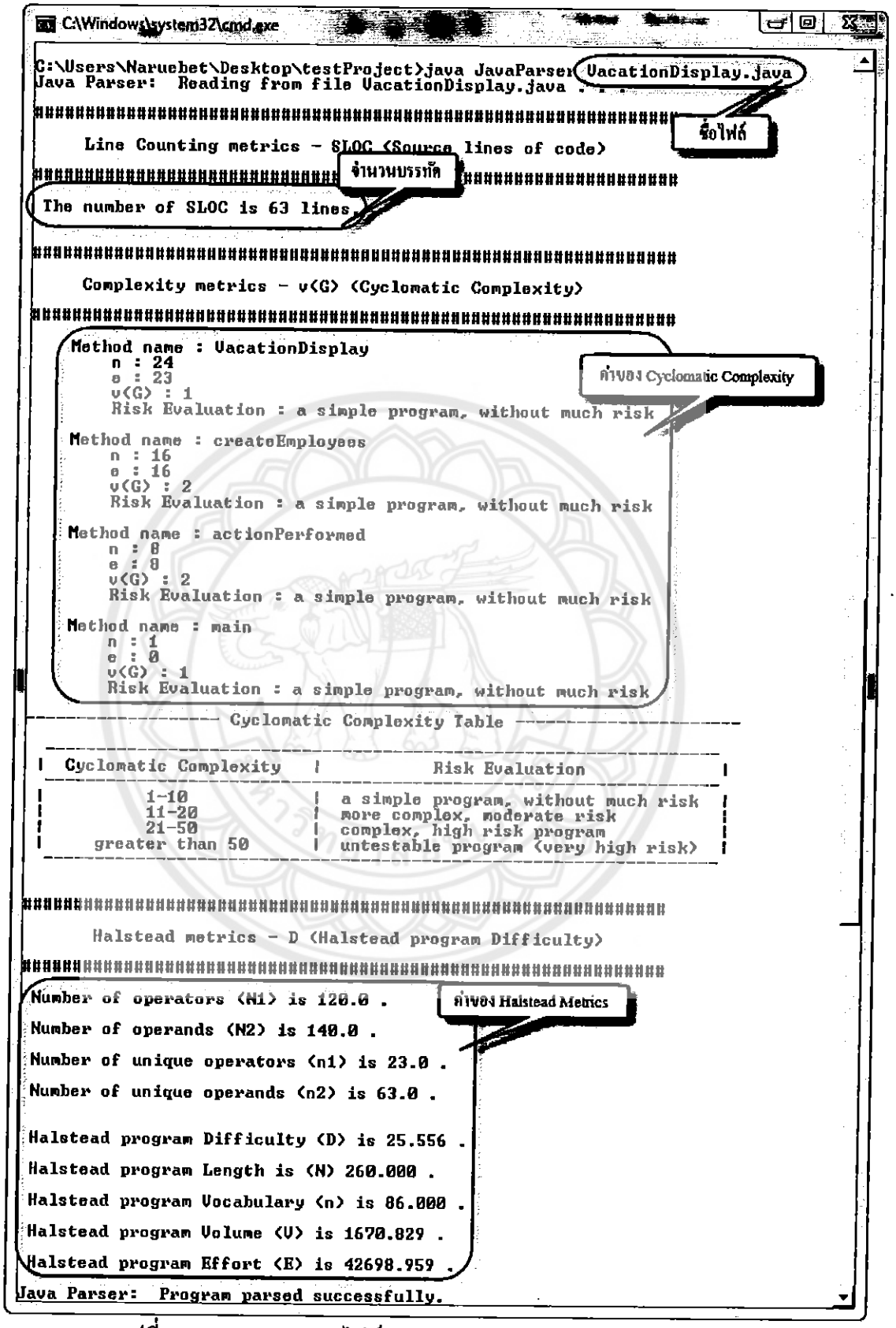
รูปที่ 4.11 ผลการทดสอบไฟล์ empTree.java ของ Composite Pattern



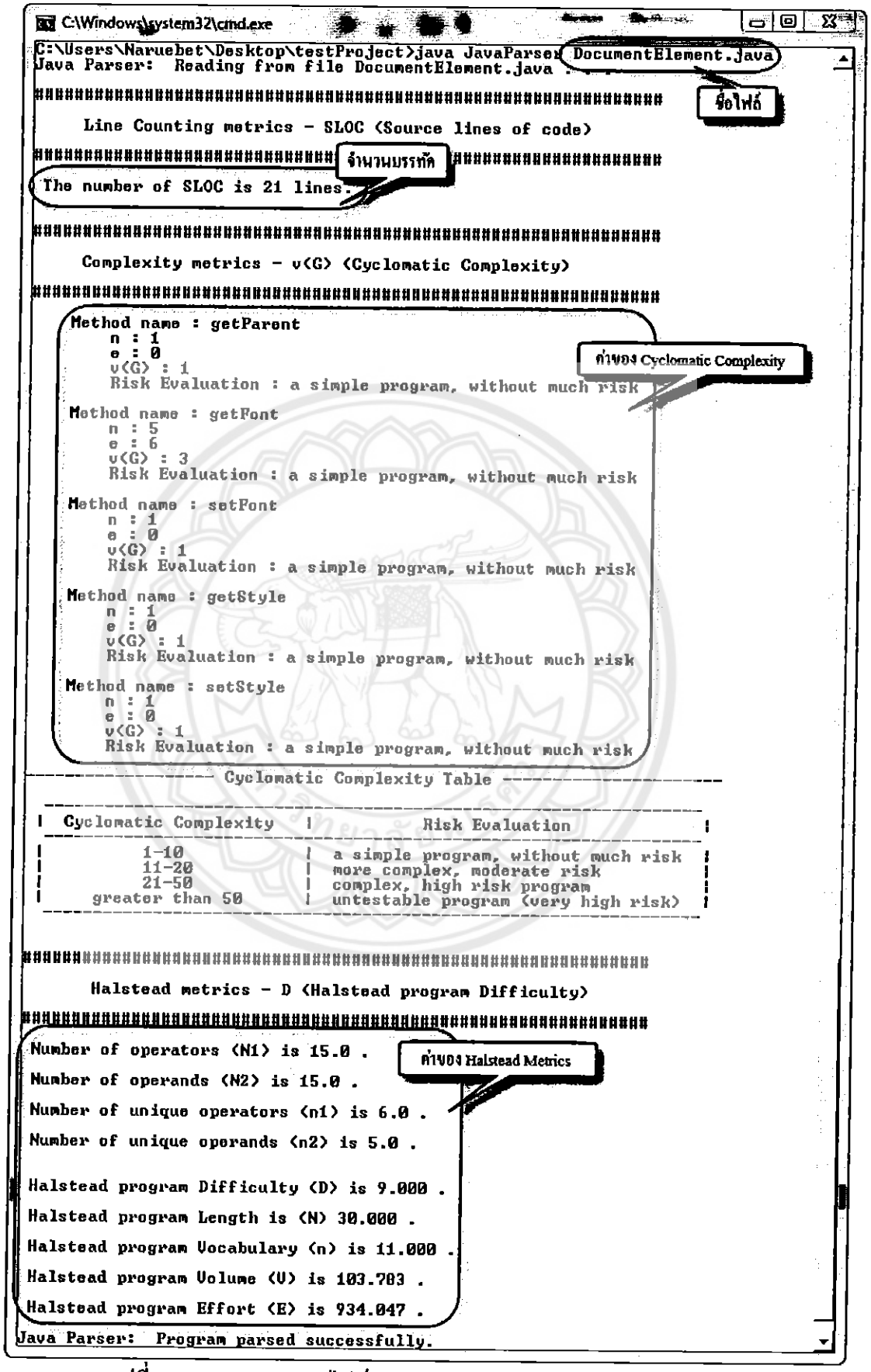
รูปที่ 4.11 (ต่อ) ผลการทดสอบไฟล์ empTree.java ของ Composite Pattern



รูปที่ 4.12 ผลการทดสอบไฟล์ CompositeDocumentElement.java ของ Composite Pattern



รูปที่ 4.13 ผลการทดสอบไฟล์ VacationDisplay.java ของ Visitor Pattern



รูปที่ 4.14 ผลการทดสอบไฟล์ DocumentElement.java ของ Visitor Pattern

ในส่วนของการทดสอบโปรแกรมใน Design Pattern ชนิดต่างๆ ได้ผลการทดสอบในส่วนของ Line Counting Metrics และ Halstead Metrics ดังตารางที่ 4.4

จากตารางที่ 4.4 แสดงให้เห็นว่าโปรแกรมที่ออกแบบโดยใช้ Design Pattern ชนิดเดียวกัน ค่าต่างๆ ของ Halstead Metrics ให้ผลลัพธ์ไม่สัมพันธ์กัน เนื่องจากค่าต่างๆนั้นขึ้นอยู่กับจำนวนของ Operator และ Operand ซึ่งในแต่โปรแกรมมีจำนวนที่แตกต่างกันและโปรแกรมต่างๆสร้างขึ้นมาเพื่อวัตถุประสงค์ที่แตกต่างกัน เมื่อจำนวนของ Operator และ Operand แตกต่างกันค่าต่างๆ ของ Halstead Metrics จึงไม่สัมพันธ์กัน ไม่สามารถนำค่ามาเปรียบเทียบกันได้

ตารางที่ 4.4 ตารางเปรียบเทียบผลการทดสอบโปรแกรมใน Design Pattern ชนิดต่างๆ

Pattern	Command Pattern		Composite Pattern		Visitor Pattern	
ชื่อไฟล์.java	CommandManager.java	testCommand.java	empTree.java	CompositeDocumentElement.java	VacationDisplay.java	DocumentElement.java
SLOC	34	50	77	27	63	21
Number of operators (N1)	55.0	46.0	171.0	42.0	120.0	15.0
Number of operands (N2)	30.0	55.0	149.0	35.0	140.0	15.0
Number of unique operators (n1)	17.0	13.0	38.0	19.0	23.0	6.0
Number of unique operands (n2)	9.0	33.0	74.0	12.0	63.0	5.0
Halstead program Difficulty (D)	28.333	10.833	38.257	27.708	25.556	9.000
Halstead program Length (N)	85.0	101.000	320.000	77.000	260.000	30.000

ตารางที่ 4.4 (ต่อ) ตารางเปรียบเทียบผลการทดสอบโปรแกรมใน Design Pattern ชนิดต่างๆ

Pattern	Command Pattern		Composite Pattern		Visitor Pattern	
ชื่อไฟล์.java	CommandManager.java	testCommand.java	empTree.java	CompositeDocumentElement.java	VacationDisplay.java	DocumentElement.java
SLOC	34	50	77	27	63	21
Halstead program Vocabulary (n)	26.0	46.000	112.000	31.000	86.000	11.000
Halstead program Volume (V)	399.537	557.880	2178.354	381.473	1670.829	103.783
Halstead program Effort (E)	11320.226	6043.697	83336.743	10569.984	42698.959	934.047

บทที่ 5
บทสรุป

โครงงานนี้ได้พัฒนาโปรแกรมเพื่อตรวจสอบความซับซ้อนของ Source code ของภาษา Java โดยใช้การตรวจวัด Line counting Metrics, Complexity Metrics และ Halstead Metrics ซึ่งพัฒนาโดยใช้ JavaCC เป็นตัวช่วยในการ Parse source code

5.1 วิเคราะห์ผลการทดลอง

จากผลการทดลองในบทที่ 4 ข้อมูลที่ได้จากการวัดสามารถนำไปประกอบการพิจารณาในการทำการตรวจสอบ Source code โดยสามารถนำค่าต่างๆ ที่ได้มาจากการวัดมาใช้เช่น ค่าของ Halstead program Difficulty บอกถึงระดับความยากหรือแนวโน้มที่เกิดข้อผิดพลาดของโปรแกรม ค่าของ Halstead program Vocabulary สามารถใช้ทำการประเมินขนาดของโปรแกรมอย่างคร่าวๆ ได้ ค่าของ Halstead program Effort เป็นค่าซึ่งบอกถึงระดับความพยายามที่ต้องใช้ในการอ่านและทำความเข้าใจในโปรแกรมโดยที่ Tester ทำการทดสอบ Source code ก่อนด้วยโปรแกรมแล้วนำค่าที่ได้จากการวัดมาพิจารณาดูว่า Source code เป็นอย่างไร ถ้าค่าที่ได้นั้นแสดงถึงความเสี่ยงที่ Source code เกิดข้อผิดพลาดนั้นมีสูง Tester ก็ไม่จำเป็นต่องทำการตรวจสอบ Source code ด้วยมือซึ่งทำให้ไม่เสียเวลาในการทดสอบเพื่อว่า Source code เกิดข้อผิดพลาดหรือไม่ นอกจากนี้ยังนำไปประยุกต์ใช้ในการพัฒนาโปรแกรมได้ โดยการนำค่าที่ได้จากการวัดจาก Source code ที่กำลังพัฒนาอยู่มาพิจารณาดูถึงระดับความซับซ้อน และพัฒนาหรือปรับปรุง Source code เดิมแล้วลองทดสอบเพื่อเปรียบเทียบความซับซ้อนก่อนและหลังการพัฒนาว่า Source code ไหนมีความซับซ้อนมากกว่ากัน เพื่อเลือกใช้ Source code ที่มีความซับซ้อนน้อยที่สุดมาใช้งาน ทำให้โอกาสที่โปรแกรมที่พัฒนาขึ้นนั้นเกิดข้อผิดพลาดลดลง

5.2 สรุปผลการทดลอง

โปรแกรมสำหรับวิเคราะห์ความซับซ้อนของ Source code ภาษา Java นั้นสามารถนำมาใช้ในการทดสอบ Line counting metrics, Halstead metrics และ Complexity metrics โดย Line counting metrics และHalstead metrics นั้นสามารถนำ Source code ซึ่งให้ผลลัพธ์เดียวกันมาทำการทดสอบเพื่อเปรียบเทียบค่าความซับซ้อนของ Source code ได้ แต่ Complexity metrics สามารถให้ผลลัพธ์แก่ Source code นั้นๆได้โดยบอกถึงความซับซ้อนของ Source code นั้นๆได้เลยว่ามีความซับซ้อนเพียงใด

เมื่อเทียบระหว่างการใช้โปรแกรมกับการคำนวณด้วยมือแล้วมีความถูกต้องและสามารถวัดผลได้จริงดังจะเห็นได้จากการสรุปผลในตารางที่ 5.1

ตารางที่ 5.1 ตารางสรุปผลการทดลอง

SoftwareMetrics	ผลการทดสอบ
Line Counting Metrics	
- SLOC	√
Complexity Metrics	
- Cyclomatic Complexity	√
Halstead Metrics	
- Number of operators (N1)	√
- Number of operands (N2)	√
- Number of unique operators (n1)	√
- Number of unique operands (n2)	√
- Halstead program Difficulty (D)	√
- Halstead program Length (N)	√
- Halstead program Vocabulary (n)	√

5.3 ข้อเสนอแนะ

โปรแกรมสำหรับวิเคราะห์ความซับซ้อนของ Source code ภาษา Java นั้นสามารถนำไปพัฒนาต่อ โดยทำการเพิ่มหลักการของ Software Metrics ชนิดอื่นๆเข้าไปในโปรแกรมเพื่อให้สามารถตรวจวัดคุณภาพของซอฟต์แวร์ ได้ดียิ่งขึ้น หรือสามารถนำโปรแกรมนี้ไปประยุกต์ใช้สำหรับสร้างโปรแกรมให้สามารถวิเคราะห์ความซับซ้อนของภาษาอื่นๆ เช่น ภาษา C/C++ เป็นต้น

เอกสารอ้างอิง

[1] คุณฉัตรชัย เหลืองมณี. การบ้าน Socket, วิชา Network System Programming

[2] AN ATTRIBUTE GRAMMAR APPROACH TO SPECIFYING HALSTEAD'S METRICS Malaysia Journal of Computer Science, Vol. 9 No. 1, June 1996, pp. 56-67

[3] Carnegie Mellon University. "Cyclomatic Complexity". [Online]. Available: http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html.2007

[4] CollabNet, Inc. "Java Compiler Compiler". [Online]. Available: https://javacc.dev.java.net. 2007

[5] Dolores R. Wallace , "Cyclomatic Complexity". [Online]. Available: "Measurement of Halstead Metrics with Testwell CMT++ and CMTJava". [Online]. Available: http://www.verifysoft.com/en_halstead_metrics.html. 2007

[6] Harvey M. Deitel and Paul J. Deitel. "Java How to program Sixth Edition". Sixth Edition. New Jersey: Prentice Hall, Inc. 2004

[7] Power software, Inc. "Essential Metrics". [Online]. Available: http://www.powersoftware.com/. 2007

[8] T. Baar. SWE Course 06/07. Principles of Good Design. 2007

[9] Vladimir Riabov . Department of Computer Science, Rivier College. NW Software Code Studies . "Networking Systems Code Studies with Structured Testing Methodology: Code Analysis", Instrumentation, Test & Code Coverage". 2007

[10] Wikimedia Foundation, Inc. "โปรแกรมแปลโปรแกรม". [Online]. Available: http://th.wikipedia.org/wiki/Compiler. 2007

[11] Wikimedia Foundation, Inc. "Software metrics". [Online]. Available: http://en.wikipedia.org/wiki/Software_metrics. 2007

ภาคผนวก ก.
ตัวอย่างโปรแกรม

Command Pattern

File name: CommandManager.java

```
1.  import java.util.LinkedList;
2.  /**
3.  Instances of this class are responsible for managing the execution of
4.  commands.  More specifically, for the purposes of this word processing
5.  program, instances of this class are responsible for maintaining a
6.  command history for undo and redo.
7.  */
8.  class CommandManager {
9.  // The maximum number of command to keep in the history
10. private int maxHistoryLength = 100;
11.
12. private LinkedList history = new LinkedList();
13. private LinkedList redoList = new LinkedList();
14.
15. /**
16. Invoke a command and add it to the history,
17. *<p>
18. If the command object's doIt method was previously called, then
19. it is expected to return false.
20. */
21. public void invokeCommand(AbstractCommand command) {
22. if (command instanceof Undo) {
        a.  undo();
        b.  return;
23. } // if undo
```

```
24. if (command instanceof Redo) {
        a. redo();
        b. return;
25. } // if redo
26. if (command.doIt()) {
        a. // doIt returned true, which means it can be undone
        b. addToHistory(command);
27. } else { // command cannot be undone
        a. history.clear();
28. } // if
29. // After a command that is not an undo or a redo, ensure that
30. // the redo list is empty.
31. if (redoList.size() > 0)
32. redoList.clear();
33. } // invokeCommand(AbstractCommand)
34.
35. /**
36. Undo the most recent command in the commmand history.
37. */
38. private void undo() {
39. if (history.size() > 0) { // If there are commands in the history
        a. AbstractCommand undoCommand;
        b. undoCommand = (AbstractCommand)history.removeFirst();
        c. undoCommand.undoIt();
        d. redoList.addFirst(undoCommand);
40. } // if
41. } // undo()
42.
43. /**
44. Redo the most recently undone command
45. */
46. private void redo() {
```

```
47. if (redoList.size() > 0) { // If the redo list is not empty
        a. AbstractCommand redoCommand;
        b. redoCommand = (AbstractCommand)redoList.removeFirst();
        c. redoCommand.doIt();
        d. history.addFirst(redoCommand);
48. } // if
49. } // redo()
50.
51. /**
52. Add a command to the command history.
53. @param command the command to add to the history.
54. */
55. private void addToHistory(AbstractCommand command) {
56. history.addFirst(command);
57. // If size of history has exceded maxHistoryLength, remove
58. // the oldest command from the history
59. if (history.size() > maxHistoryLength)
60. history.removeLast();
61. } // addToHistory(AbstractCommand)
62. } // class CommandManager
```

File name: testCommand.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3.
4. public class testCommand extends Frame
5. implements ActionListener
6. {
7. Menu mnuFile;
8. fileOpenCommand mnuOpen;
```

```
9.  fileExitCommand mnuExit;
10. btnRedCommand btnRed;
11. Panel p;
12. Frame fr;
13. //---------------------------------------
14. public testCommand()
15. {
16. super("Frame without commands");
17. fr = this;    //save frame object
18. MenuBar mbar = new MenuBar();
19. setMenuBar(mbar);
20.
21. mnuFile = new Menu("File", true);
22. mbar.add(mnuFile);
23.
24. mnuOpen = new fileOpenCommand("Open...");
25. mnuFile.add(mnuOpen);
26. mnuExit = new fileExitCommand("Exit");
27. mnuFile.add(mnuExit);
28.
29. mnuOpen.addActionListener(this);
30. mnuExit.addActionListener(this)
31.
32. btnRed = new btnRedCommand("Red");
33. p = new Panel();
34. add(p);
35. p.add(btnRed);
36.
37. btnRed.addActionListener(this);
38. setBounds(100,100,200,100);
39. setVisible(true);
40. }
```

```
41. //---------------------------------------
42. public void actionPerformed(ActionEvent e)
43. {
44. Command obj = (Command)e.getSource();
45. obj.Execute();
46. }
47. //---------------------------------------
48. static public void main(String argv[])
49. {
50. new testCommand();
51. }
52. //=====----=========-----inner class----=========----
53. class btnRedCommand extends Button implements Command
54. {
55. public btnRedCommand(String caption)
56. {
57. super(caption);
58. }
59. public void Execute()
60. {
61. p.setBackground(Color.red);
62. }
63. }
64. //---------------------------------------
65. class fileOpenCommand extends MenuItem implements Command
66. {
67. public fileOpenCommand(String caption)
68. {
69. super(caption);
70. }
71. public void Execute()
72. {
```

```
73. FileDialog fDlg=new FileDialog(fr,"Open file");
74. fDlg.show();
75. }
76. }
77. //----------------------------------------
78. class fileExitCommand extends MenuItem implements Command
79. {
80. public fileExitCommand(String caption)
81. {
82. super(caption);
83. }
84. public void Execute()
85. {
86. System.exit(0);
87. }
88. }
89. }
90. //========================================
```

Composite Pattern

File name: emptree.java

```
1. import java.awt.*;
2. import java.awt.event.*;
3. import java.util.*;
4.
5. //swing classes
6. import com.sun.java.swing.text.*;
7. import com.sun.java.swing.*;
8. import com.sun.java.swing.event.*;
9. import com.sun.java.swing.border.*;
```

```
10. import com.sun.java.swing.tree.*;
11.
12. public class empTree extends JxFrame
13. implements TreeSelectionListener
14. {
15. Employee boss, marketVP, prodVP;
16. Employee salesMgr, advMgr;
17. Employee prodMgr, shipMgr;
18.
19. JScrollPane sp;
20. JPanel treePanel;
21. JTree tree;
22. DefaultMutableTreeNode troot;
23. JLabel cost;
24.
25. public empTree()
26. {
27. super("Employee tree");
28. makeEmployees();
29. setGUI();
30. }
31. //-------------------------------------
32. private void setGUI()
33. {
34. treePanel = new JPanel();
35. getContentPane().add(treePanel);
36. treePanel.setLayout(new BorderLayout());
37.
38. sp = new JScrollPane();
39. treePanel.add("Center", sp);
40. treePanel.add("South", cost = new JLabel("         "));
```

```
41. treePanel.setBorder(new BevelBorder(BevelBorder.RAISED));
42. troot = new DefaultMutableTreeNode(boss.getName());
43. tree= new JTree(troot);
44. tree.setBackground(Color.lightGray);
45. loadTree(boss);
46. /* Put the Tree in a scroller. */
47.
48. sp.getViewport().add(tree);
49. setSize(new Dimension(200, 300));
50. setVisible(true);
51.
52. }
53. //-----------------------------------
54. public void loadTree(Employee topDog)
55. {
56. DefaultMutableTreeNode troot;
57. troot = new DefaultMutableTreeNode(topDog.getName());
58. treePanel.remove(tree);
59. tree= new JTree(troot);
60. tree.addTreeSelectionListener(this);
61. sp.getViewport().add(tree);
62.
63. addNodes(troot, topDog);
64. tree.expandRow(0);
65. repaint();
66. }
67. //---------------------------------------
68. private void addNodes(DefaultMutableTreeNode pnode, Employee emp)
69. {
70. DefaultMutableTreeNode node;

71. Enumeration e = emp.elements();
```

```
72. while(e.hasMoreElements())
        a. {
        b. Employee newEmp = (Employee)e.nextElement();
        c. node = new DefaultMutableTreeNode(newEmp.getName());
        d. pnode.add(node);
        e. addNodes(node, newEmp);
        f. }
73. }
74. //-----------------------------------
75. private void makeEmployees()
76. {
77. boss = new Employee("CEO", 200000);
78. boss.add(marketVP = new Employee("Marketing VP", 100000));
79. boss.add(prodVP = new Employee("Production VP", 100000));
80.
81. marketVP.add(salesMgr = new Employee("Sales Mgr", 50000));
82. marketVP.add(advMgr = new Employee("Advt Mgr", 50000));
83.
84. for (int i=0; i<5; i++)
85. salesMgr .add(new Employee("Sales "+new Integer(i).toString(), 30000.0F
    +(float)(Math.random()-0.5)*10000));
86. advMgr.add(new Employee("Secy", 20000));
87.
88. prodVP.add(prodMgr = new Employee("Prod Mgr", 40000));
89. prodVP.add(shipMgr = new Employee("Ship Mgr", 35000));
90. for (int i = 0; i < 4; i++)
91. prodMgr.add( new Employee("Manuf "+new Integer(i).toString(), 25000.0F
    +(float)(Math.random()-0.5)*5000));
92. for (int i = 0; i < 3; i++)
93. shipMgr.add( new Employee("ShipClrk "+new Integer(i).toString(), 20000.0F
    +(float)(Math.random()-0.5)*5000));
```

```
94. }
95. //-------------------------------------
96. public void valueChanged(TreeSelectionEvent evt)
97. {
98. TreePath path = evt.getPath();
99. String selectedTerm = path.getLastPathComponent().toString();
100.
101.        Employee emp = boss.getChild(selectedTerm);
102.        if(emp != null)
103.        cost.setText(new Float(emp.getSalaries()).toString());
104.        }
105.        //-------------------------------------
106.        static public void main(String argv[])
107.        {
108.        new empTree();
109.        }
110.        }
```

File name: CompositeDocumentElement.java

```
1.  import java.util.Vector;
2.
3.  /**
4.  Instances of this class are composite objects that contain
5.  DocumentElement objects.
6.  */
7.  abstract class CompositeDocumentElement extends DocumentElement {
8.  // Collection of this object's children
9.  private Vector children = new Vector();

10. // The cached value from the previous call to getCharLength or -1 to
```

```
11. // indicate that charLength does not contain a cached value.
12. private int cachedCharLength = -1;
13.
14. /**
15. Return the child object of this object that is at the given
16. position.
17. @param index The index of the child.
18. */
19. public DocumentElement getChild(int index) {
20. return (DocumentElement)children.elementAt(index);
21. } // getChild(int)
22.
23. /**
24. Make the given DocumentElement a child of this object.
25. */
26. public synchronized void addChild(DocumentElement child) {
27. synchronized (child) {
        a. children.addElement(child);
        b. child.parent = this;
        c. changeNotification();
28. } // synchronized
29. } // addChild(DocumentElement)
30.
31. /**
32. Make the given DocumentElement NOT a child of this object.
33. */
34. public synchronized void removeChild(DocumentElement child) {
35. synchronized (child) {
        a. if (this == child.parent)
        b. child.parent = null;
        c. children.removeElement(child);
        d. changeNotification();
```

```
36. } // synchronized
37. } // removeChild(DocumentElement)
38.
39. //...
40.
41. /**
42. A call to this method means that one of this object's children
43. has changed in a way that invalidates whatever data this object
44. may be cahcing about its children.
45. */
46. public void changeNotification() {
47. cachedCharLength = -1;
48. if (parent != null)
49. parent.changeNotification();
50. } // changeNotification()
51.
52. /**
53. Return the number of characters that this object contains.
54. */
55. public int getCharLength() {
56. int len = 0;
57. for (int i = 0; i < children.size(); i++) {
        a. len += ((DocumentElement)children.elementAt(i)).getCharLength();
58. } // for
59. cachedCharLength = len;
60. return len;
61. } // getCharLength()
62. } // class CompositeDocumentElement
```

Visitor Pattern

File name: VacationDisplay.java

```
1.  import java.awt.*;
2.  import java.awt.event.*;
3.  import java.util.*;
4.
5.  //swing classes
6.  import com.sun.java.swing.text.*;
7.  import com.sun.java.swing.*;
8.  import com.sun.java.swing.event.*;
9.  import com.sun.java.swing.border.*;
10.
11. public class VacationDisplay extends JxFrame
12. implements ActionListener
13. {
14. JawtList empList;
15. JTextField total, btotal;
16. JButton Vac;
17. Employee[] employees;
18. //-------------------------------------
19. public VacationDisplay()
20. {
21. super("Vacation Display");
22. JPanel jp = new JPanel();
23. getContentPane().add(jp);
24. jp.setLayout(new GridLayout(1,2));
25. empList = new JawtList(10);
26. jp.add(empList);
27.
28. createEmployees();

29. Box box = Box.createVerticalBox();
```

```
30. jp.add(box);
31. total = new JTextField(5);
32. total.setHorizontalAlignment(JTextField.CENTER);
33. box.add(total);
34. box.add(Box.createVerticalStrut(10));
35. btotal = new JTextField(5);
36. btotal.setHorizontalAlignment(JTextField.CENTER);
37. box.add(btotal);
38. box.add(Box.createVerticalStrut(10));
39. Vac = new JButton("Vacations");
40. box.add(Vac);
41. Vac.addActionListener(this);
42. setSize(300,200);
43. setVisible(true);
44. total.setText(" ");
45. total.setBackground(Color.white);
46.
47. }
48. //---------------------------------------
49. public void createEmployees()
50. {
51. employees = new Employee[7];
52. int i = 0;
53. employees[i++] =new Employee("Susan Bear",  55000, 12, 1);
54. employees[i++] = new Employee("Adam Gehr",  150000, 9, 0);
55. employees[i++] = new Employee("Fred Harris",  50000, 15, 2);
56. employees[i++] = new Employee("David Oakley",  57000, 12, 2);
57. employees[i++] = new Employee("Larry Thomas",  100000, 20, 6);
58. Boss b = new Boss("Leslie Susi", 175000, 16,4);
59. b.setBonusDays(12);
60. Boss b1 = new Boss("Laurence Byerly", 35000, 17,6);
61. b1.setBonusDays(17);
```

```
62. employees[i++] = b;
63. employees[i++] = b1;
64.
65. for ( i = 0; i < employees.length; i++)
66. {
67. empList.add(employees[i].getName());
68. }
69. }
70. //---------------------------------------
71. public void actionPerformed(ActionEvent e)
72. {
73. VacationVisitor vac = new VacationVisitor();
74. bVacationVisitor bvac = new bVacationVisitor();
75. for (int i = 0; i < employees.length; i++)
76. {
77. employees[i].accept(vac);
78. employees[i].accept(bvac);
79. }
80. total.setText(new Integer(vac.getTotalDays()).toString());
81. btotal.setText(new Integer(bvac.getTotalDays()).toString());
82. }
83. //---------------------------------------
84. static public void main(String argv[])
85. {
86. new VacationDisplay();
87. }
88. }
```

File name: DocumentElement.java

```
1.  import java.awt.Font;
2.
3.  /**
4.  All elements of a document belong to a subclass of this abstract class.
5.  */
6.  abstract class DocumentElement {
7.  // This is the font associated with this object.  If the font
8.  // variable is null, then this object's font will be inherited
9.  // through the container hierarchy from an enclosing object.
10. private Font font;
11.
12. // The name of the style associated with this element
13. private String style;
14.
15. CompositeDocumentElement parent; // this object's container
16.
17. //...
18.
19. /**
20. Return this object's parent or null if it has no parent.
21. */
22. public CompositeDocumentElement getParent() {
23. return parent;
24. } // getParent()
25.
26. /**
27. Return the Font associatiated with this object.  If there is no
28. Font associated with this object, then return the Font associated
29. with this object's parent. If there is no Font associated
30. with this object's parent the return null.
31. */
```

```
32. public Font getFont() {
33. if (font != null)
34. return font;
35. else if (parent != null)
36. return parent.getFont();
37. else
38. return null;
39. } // getFont()
40.
41. /**
42. Associate a Font with this object.
43. @param font The font to associate with this object
44. */
45. public void setFont(Font font) {
46. this.font = font;
47. } // setFont(Font)
48.
49. /**
50. Return the number of characters that this object contains.
51. */
52. public abstract int getCharLength() ;
53.
54. /**
55. Return the name of the style associated with this document
56. element.
57. */
58. public String getStyle() { return style; }
59.
60. /**
61. Set the namd of the style associated with this document element.
62. */
63. public void setStyle(String style) { this.style = style; }
```

```
64. } // class DocumentElement
```

ประวัติผู้เขียนโครงงาน

ชื่อ นายกมล สุวรรณกิจ

ภูมิลำเนา 34 ถนนเจ้าพระยา ตำบลในเมือง อำเภอเมือง
จังหวัดพิษณุโลก 65000

ประวัติการศึกษา

- จบการศึกษาระดับมัธยมศึกษาจาก
  โรงเรียนพิษณุโลกพิทยาคม
- ปัจจุบันกำลังศึกษาอยู่ชั้นปีที่ 4
  สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
  มหาวิทยาลัยนเรศวร

E-mail birdcpe@hotmail.com

ชื่อ นายนฤเบศ หาญรักษ์

ภูมิลำเนา 76 หมู่ 3 ตำบลท่าแดง อำเภอหนอง จังหวัดเพชรบูรณ์ 67140

ประวัติการศึกษา

- จบการศึกษาระดับมัธยมศึกษาจาก
  โรงเรียนหนองไผ่
- ปัจจุบันกำลังศึกษาอยู่ชั้นปีที่ 4
  สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
  มหาวิทยาลัยนเรศวร

E-mail bet_comeng@hotmail.com