บทที่ 4

การพัฒนาระบบ

ในบทนี้เป็นการอธิบายรายละเอียดของการพัฒนาเครื่องมือวัดซอฟต์แวร์ Chula OOFP Counting ซึ่งใช้เทคนิคการวัดแบบฟังก์ชันพอยต์เชิงวัตถุ การอธิบายจะกระทำในระดับการทำงานของคลาสต่างๆ และความสัมพันธ์ระหว่างคลาสที่มีในระบบ โดยนำเสนอผ่านทางแผนภาพและโค้ดต้นฉบับ ภาษาจาวา รายละเอียดเกี่ยวกับการพัฒนานั้นจะกล่าวถึง การสร้างแพ็กเกจคอมไพเลอร์ภาษาจาวา การสร้างแพ็กเกจคอมไพเลอร์ภาษาเอโอแอล การสร้างข้อมูลภาษาเอโอแอลจากไฟล์ต้นฉบับภาษาจาวา และการนับจำนวนของฟังก์ชันพอยต์เชิงวัตถุจากข้อมูลภาษาเอโอแอล

4.1 การทำงานแบบไคลเอนต์เซอร์ฟเวอร์ของเครื่องมือวัด

การทำงานของเครื่องมือวัด Chula OOFP Counting เป็นแบบไคลเอนต์เซอร์ฟเวอร์ซึ่งประกอบไปด้วย 2 ส่วนหลัก ได้แก่ เครื่องไคลเอนต์ติดตั้งโปรแกรม Chula OOFP Counting เพื่อรองรับการใช้งานจากผู้ใช้ และเครื่องให้บริการเว็บซึ่งบรรจุชุดคำสั่งภาษาเพิร์ลที่ทำหน้าที่แปลงไฟล์ข้อมูลภาษาจาวาให้เป็นภาษาเอโอแอล ดังรูปที่ 3.1

4.1.1 เครื่องไคลเอนต์ติดตั้งโปรแกรม Chula OOFP Counting

โปรแกรม Chula OOFP Counting ถูกพัฒนาขึ้นด้วยภาษาจาวาบนระบบปฏิบัติการวินโดว์ 2000 โดยใช้คลาสไลบรารีของจาวาดีเวลลอปเมนต์คิต (Java Development Kit: JDK) เวอร์ชัน 1.3.1 ซึ่งได้รวมแพ็คเกจจาวาฟาวเดชั่นคลาสไลบรารี่ (Java Foundation Class Library: JFC) เป็นแพ็คเกจส่วนขยายมาตรฐาน ซึ่งมีแพ็คเกจสวิง สำหรับใช้ในการสร้างส่วนติดต่อกับผู้ใช้ กล่าวโดยรวมเครื่องมือที่ใช้พัฒนาและทดสอบโปรแกรม Chula OOFP Counting ในส่วนไคลเอนต์ประกอบไปด้วย

- ระบบปฏิบัติการวินโดว์ 2000
- คลาสไลบรารีของจาวาดีเวลลอปเมนต์คิต เวอร์ชัน 1.3.1
- โปรแกรมอีดิตพลัส เป็นโปรแกรมประมวลผลคำ (Text Editor) สำหรับเขียนไฟล์ต้นฉบับภาษาจาวา
- โปรแกรมจาวาซีซี (JavaCC) เวอร์ชัน 2.1 เป็นโปรแกรมประมวลผลไวยากรณ์ของภาษาที่ผู้ใช้กำหนดขึ้นในรูปแบบบีเอนเอฟ (BNF: Backus Naur Form) แล้วสร้างไฟล์ต้นฉบับภาษาจาวาเพื่อใช้ในการประมวลผลโทเคน (Token) และไวยากรณ์ของภาษานั้นๆ
- โปรแกรมโออีดับบริว (OEW) เวอร์ชัน 3.0 เป็นโปรแกรมสำหรับการสร้างแผนภาพคลาส และสามารถสร้างไฟล์ต้นฉบับภาษาจาวาจากแผนภาพคลาสที่สร้างขึ้นได้

โปรแกรมในส่วนไคลเอนต์นี้ทำหน้าที่เป็นโมดูลหลักในการรับคำสั่งทั้งหมดจากผู้ใช้ โดยหน้าที่หลักได้แก่

1) การจัดการไฟล์ต้นฉบับภาษาจาวาในรูปแบบของโปรแกรมประมวลผลคำ (Text Editor) โดยมีความสามารถพื้นฐานในการสร้าง บันทึก เปิดใช้ การพิมพ์ การทำงานกับคลิ๊ปบอร์ด (Clipboard) เป็นต้น
2) การจัดการโปรเจ็คไฟล์เพื่อรวบรวมไฟล์ต้นฉบับภาษาจาวาให้เป็นหมวดหมู่ และพร้อมที่จะดำเนินการวิเคราะห์และนับจำนวนของฟังก์ชันพอยต์ซึ่งเป็นเป้าหมายหลักของโปรแกรมเครื่องมือวัดนี้
3) การคอมไพล์ไฟล์ต้นฉบับภาษาจาวาและประมวลผลโปรแกรมจาวาได้โดยตรงจากภายในโปรแกรม Chula OOFP Counting
4) ส่งข้อมูลของโปรเจ็คไฟล์ที่กำลังเปิดใช้พร้อมกับไฟล์ต้นฉบับภาษาจาวาจากเครื่องไคลเอนต์ไปยังเครื่องให้บริการเว็บเพื่อวิเคราะห์และสร้างข้อมูลภาษาเอโอแอล
5) การนับจำนวนฟังก์ชันพอยต์เชิงวัตถุจากภาษาเอโอแอล

เนื่องจากโปรแกรม Chula OOFP Counting ในส่วนไคลเอนต์ถูกพัฒนาขึ้นด้วยภาษาจาวา ดังนั้นคุณสมบัติของการไม่ยึดติดกับระบบปฏิบัติการของภาษาจาวาจึงถูกถ่ายทอดมาสู่โปรแกรม Chula OOFP Counting ด้วย แต่ลักษณะของส่วนติดต่อกับผู้ใช้จะแตกต่างกันไปตามระบบปฏิบัติการที่โปรแกรมทำงานอยู่ และด้วยเหตุที่โปรแกรม Chula OOFP Counting ถูกพัฒนาและดำเนินการทดสอบบนระบบปฏิบัติการวินโดว์เท่านั้น ดังนั้นโปรแกรมดังกล่าวจึงถูกแนะนำให้ใช้บนระบบปฏิบัติการวินโดว์เช่นเดียวกัน

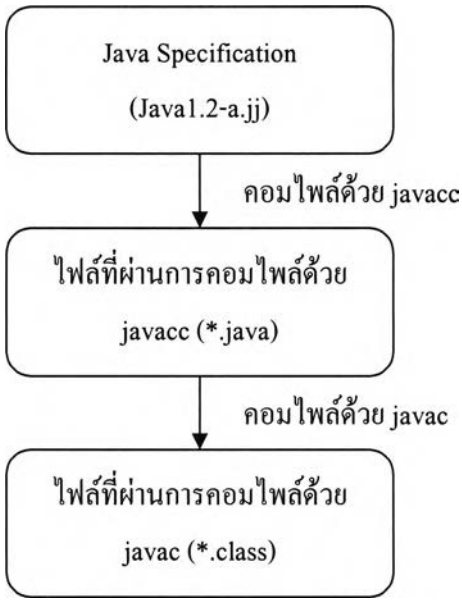
4.1.2 เครื่องให้บริการเว็บพร้อมด้วยชุดคำสั่งภาษาเพิร์ลของโปรแกรม Chula OOFP Counting

ภายในเครื่องให้บริการเว็บ (Webserver) จะเก็บกลุ่มโปรแกรมชุดคำสั่งภาษาเพิร์ลที่ทำหน้าที่วิเคราะห์ภาษาเอโอแอล จากไฟล์ต้นฉบับภาษาจาวาที่ถูกส่งมาจากเครื่องไคลเอนต์ และส่งข้อมูลภาษาเอโอแอล กลับไปยังเครื่องไคลเอนต์เพื่อใช้ในกระบวนการนับจำนวนฟังก์ชันพอยต์เชิงวัตถุต่อไป ในระหว่างการพัฒนาและทดสอบ เครื่องให้บริการเว็บในงานวิจัยนี้ใช้โปรแกรม OmniHTTPd Professional 2.0 เป็นโปรแกรมเว็บเซอร์ฟเวอร์ เพื่อรองรับคำสั่งขอใช้บริการจากโปรแกรมไคลเอนต์ Chula OOFP Counting เพื่อใช้งานโปรแกรมชุดคำสั่งภาษาเพิร์ล กล่าวโดยรวมเครื่องมือที่ใช้พัฒนาและทดสอบโปรแกรม Chula OOFP Counting ในส่วนเซอร์ฟเวอร์ประกอบไปด้วย

- ระบบปฏิบัติการวินโดว์ 2000
- โปรแกรมแปลภาษาเพิร์ลบนวินโดว์เวอร์ชัน 5.6 ซึ่งพัฒนาโดยบริษัท ActiveState
- โปรแกรมให้บริการเว็บออมนิ โปรเฟสชันนอล เวอร์ชัน 2.0 (OmniHTTPd Professional) ซึ่งพัฒนาโดยบริษัท Omnicron

4.2 การสร้างแพ็กเกจคอมไพเลอร์ภาษาจาวา

การสร้างแพ็กเกจคอมไพเลอร์ภาษาจาวา ได้ใช้ชุดเครื่องมือจาวาซีซี ซึ่งพัฒนาโดยบริษัท Metamata ในการสร้างแพ็กเกจคอมไพเลอร์ ผู้วิจัยได้ใช้ไฟล์แสดงไวยากรณ์ซึ่งอธิบายโทเคนและไวยากรณ์ของภาษาจาวา (Java Specification) ได้แก่ไฟล์ Java1.2-a.jj โดยไฟล์นี้ได้จากการติดตั้งชุดเครื่องมือจาวาซีซี จากนั้นใช้ชุดเครื่องมือจาวาซีซีแปลไฟล์แสดงไวยากรณ์ของภาษาจาวาดังกล่าว เพื่อสร้างชุดแพ็กเกจคอมไพเลอร์ภาษาจาวาดังขั้นตอนในรูปที่ 4.1

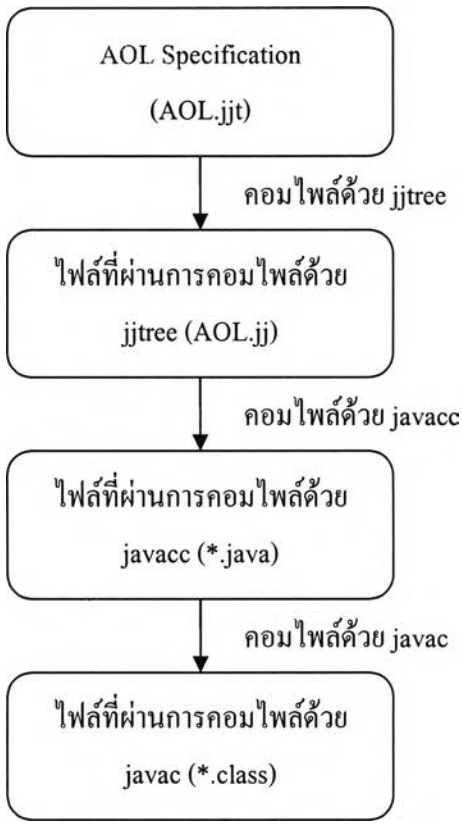


รูปที่ 4.1 แผนภาพแสดงขั้นตอนการสร้างแพ็กเกจคอมไพเลอร์ภาษาจาวา

จากรูปที่ 4.1 เป็นแผนภาพแสดงขั้นตอน (Activity Diagram) การสร้างแพ็กเกจคอมไพเลอร์ภาษาจาวาโดยเริ่มจากไฟล์ข้อกำหนดของภาษาจาวา ซึ่งบรรจุรายละเอียดเกี่ยวกับข้อกำหนดโทเคน (Lexical Specifications) และข้อกำหนดไวยากรณ์ (Grammar Specification) โดยเขียนอยู่ในรูปไวยากรณ์ของจาวาซีซี ไฟล์ข้อกำหนดภาษาจาวานี้มีอยู่ในชุดเครื่องมือจาวาซีซี ซึ่งเป็นจาวาเวอร์ชัน 1.2 (Java1.2-a.jj) จากนั้นนำไฟล์ข้อกำหนดภาษาจาวา มาประมวลผลด้วยโปรแกรมจาวาซีซี โดยจาวาซีซีจะประมวลผลโทเคน และไวยากรณ์แล้วสร้างเป็นไฟล์ต้นฉบับภาษาจาวาจำนวนหนึ่ง จากนั้นนำไฟล์ต้นฉบับภาษาจาวาเหล่านี้มาประมวลผลด้วยโปรแกรมจาวาซี (javac) ซึ่งเป็นคอมไพเลอร์ของจาวาดีเวลลอปเมนต์คิต และสิ่งที่ได้คือคอมไพเลอร์ภาษาจาวา ในเครื่องมือวัดที่พัฒนาขึ้นนี้ใช้แพ็กเกจคอมไพเลอร์ภาษาจาวาที่สร้างขึ้นข้างต้นในการเก็บข้อมูลของคลาส เมธอดและแอตริบิว เพื่อนำไปใช้ในการคำนวณฟังก์ชันพอยต์ โดยจะต้องทำการแก้ไขปรับปรุง เพิ่มเติมในไฟล์ข้อกำหนดภาษาจาวาซึ่งอยู่ในรูปไวยากรณ์ของจาวาซีซี ก่อนการประมวลผลด้วยโปรแกรมจาวาซีซี เพื่อให้คอมไพเลอร์ภาษาจาวาที่สร้างขึ้นใหม่นี้สามารถเก็บข้อมูลของคลาส เมธอดและแอตริบิวที่ต้องการได้

4.3 การสร้างแพ็กเกจคอมไพเลอร์ภาษาเอโอแอล

การสร้างแพ็กเกจคอมไพเลอร์ภาษา เอโอแอลมีรูปแบบคล้ายคลึงกับการสร้างแพ็กเกจคอมไพเลอร์ภาษาจาวา แต่จะเพิ่มขั้นตอนของการคอมไพล์ด้วยโปรแกรมทรีบิวเดอร์ ซึ่งเป็นโปรแกรมหนึ่งในชุดเครื่องมือจาวาซีซี ดังรูปที่ 4.2



รูปที่ 4.2 แผนภาพแสดงขั้นตอนการสร้างแพ็กเกจคอมไพเลอร์ภาษาเอโอแอล

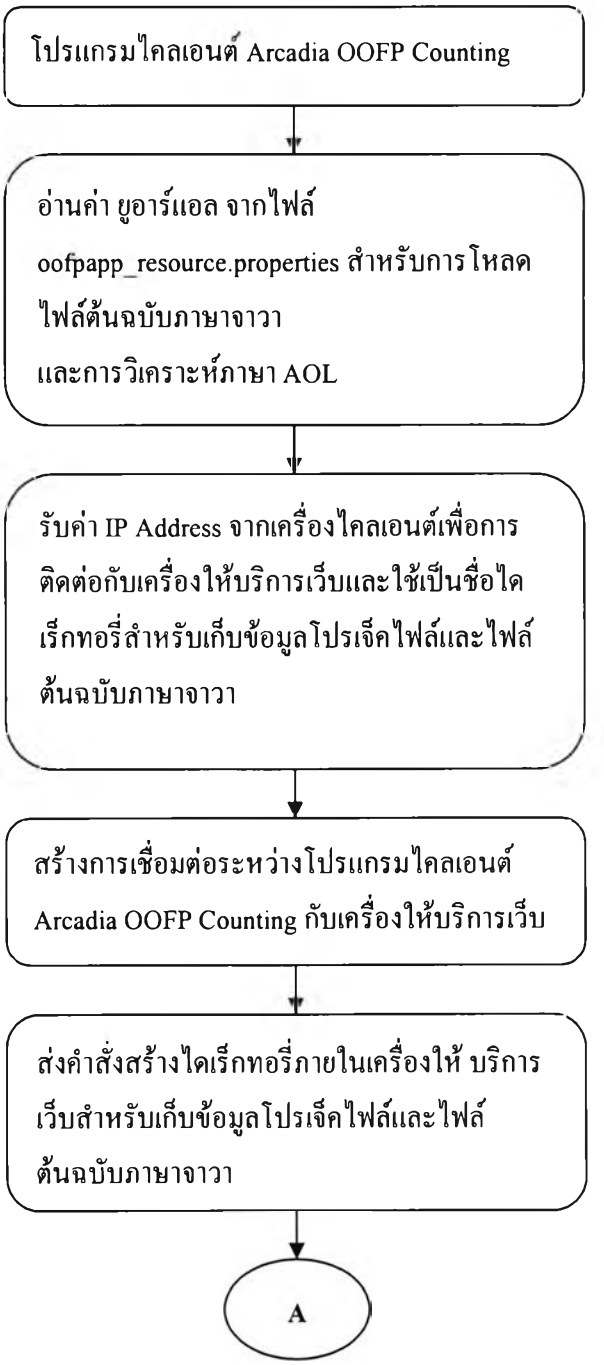
จากรูปที่ 4.2 เป็นแผนภาพแสดงขั้นตอน การสร้างแพ็กเกจคอมไพเลอร์ภาษาเอโอแอล โดยเริ่มจากไฟล์ข้อกำหนดของภาษา เอโอแอล จะบรรจุรายละเอียดเกี่ยวกับข้อกำหนดโทเคนและข้อกำหนดไวยากรณ์ของภาษา ซึ่งเขียนอยู่ในรูปไวยากรณ์ของจาวาซีซี โดยผู้วิจัยได้พัฒนาไฟล์ข้อกำหนดภาษาเอโอแอล (AOL.jjt) นี้ขึ้น นอกจากความเข้าใจในโทเคนและไวยากรณ์ของภาษาเอโอแอลแล้ว โค้ดของอัลกอริทึมในการคำนวณจำนวนของฟังก์ชันพอยต์จะอยู่ภายในไฟล์ข้อกำหนดนี้ด้วย และจากนั้นนำไฟล์ข้อกำหนดดังกล่าว มาประมวลผลด้วยโปรแกรมทรีบิวเดอร์ (Tree Builder) ซึ่งอยู่ในชุดของเครื่องมือจาวาซีซี โดยโปรแกรมทรีบิวเดอร์จะประมวลผลและสร้างไฟล์ AOL.jj จากนั้นนำไฟล์ AOL.jj มาประมวลผลด้วยโปรแกรมจาวาซีซี เพื่อประมวลผลโทเคนและไวยากรณ์และสร้างไฟล์ต้นฉบับภาษาจาวาซึ่งเป็นโค้ดของคอมไพเลอร์ภาษาเอโอแอล จากนั้นนำไฟล์เหล่านี้มา

ประมวลผลด้วยโปรแกรมจาวาซี ซึ่งสิ่งที่ได้รับคือชุดแพ็กเกจภาษาจาวาที่เข้าใจโทเคนและไวยากรณ์ของภาษาเอโอแอล และมีความสามารถในการคำนวณหาจำนวนของฟังก์ชันพอยต์จากภาษาเอโอแอลได้

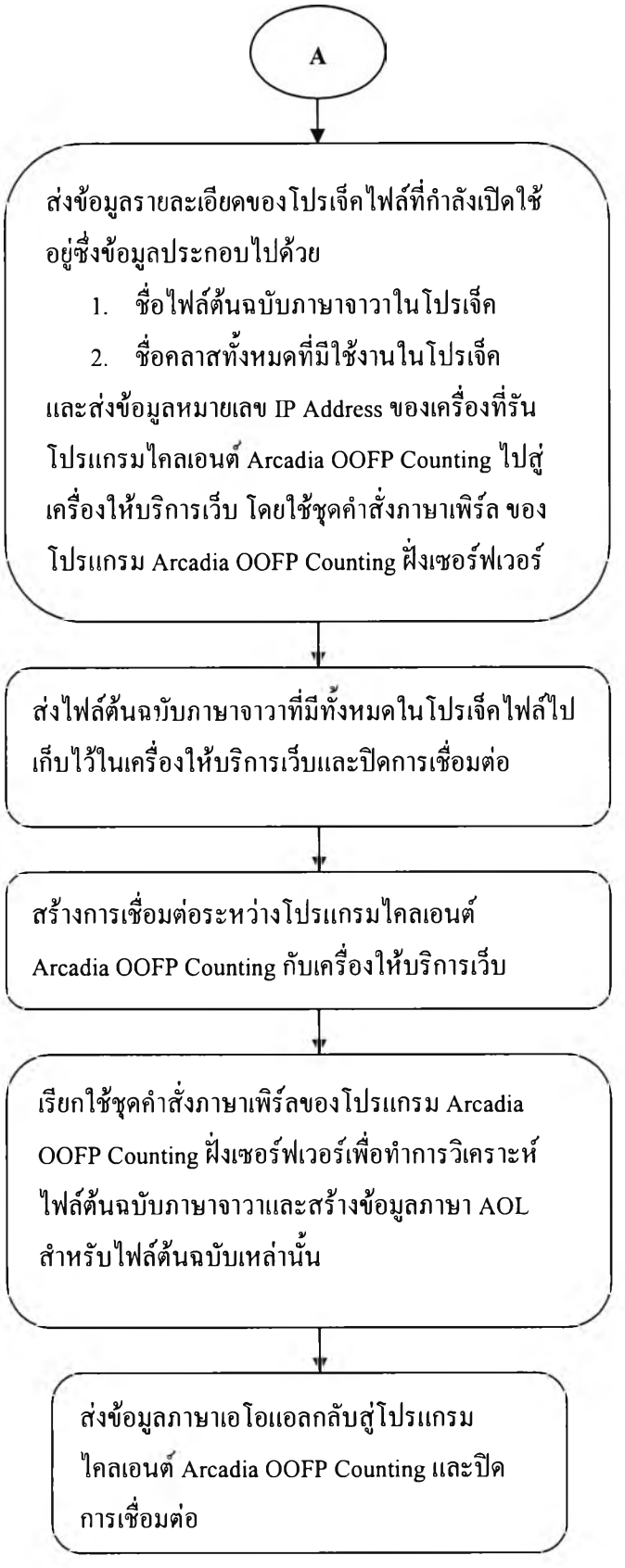
4.4 การสร้างข้อมูลภาษาเอโอแอลจากไฟล์ต้นฉบับภาษาจาวา

การวิเคราะห์ข้อมูลภาษาเอโอแอลจากไฟล์ต้นฉบับภาษาจาวาเป็นหน้าที่ของกลุ่มไฟล์ชุดคำสั่งภาษาเพิร์ลในเครื่องให้บริการเว็บ โดยที่โปรแกรมไคลเอนต์ Chula OOFP Counting ทำหน้าที่ส่งข้อมูลของโปรเจ็คไฟล์ที่กำลังเปิดใช้อยู่ ชื่อคลาสทุกๆคลาสที่มีใช้งานในโปรเจ็ค หมายเลขไอพีแอดเดรส ของเครื่องไคลเอนต์ที่รันโปรแกรม และไฟล์ต้นฉบับภาษาจาวาทั้งหมดที่ใช้ในการวิเคราะห์ ไปยังเครื่องให้บริการเว็บ โดยที่กลุ่มไฟล์ชุดคำสั่งภาษาเพิร์ลจะรับหน้าที่ในการวิเคราะห์และสร้างข้อมูลภาษา เอโอแอล อีกต่อหนึ่ง รูปที่ 4.3 แสดงภาพขั้นตอนรายละเอียดเกี่ยวกับการพัฒนาในส่วนของการสร้างข้อมูลภาษาเอโอแอล จากไฟล์ต้นฉบับภาษาจาวา (ชุดคำสั่งภาษาเพิร์ลสำหรับการรับไฟล์ต้นฉบับภาษาจาวาจากโปรแกรมไคลเอนต์ Chula OOFP Counting และจัดการเก็บบันทึกลงในเครื่องให้บริการเว็บ ดูรายละเอียดได้ที่ ในภาคผนวก ข.)

จากรูปที่ 4.3 การทำงานเริ่มจากโปรแกรมไคลเอนต์ Chula OOFP Counting ทำการอ่านค่า ยูอาร์แอล (Uniform Resource Locator: URL) จากไฟล์ oofpapp_resource.properties ซึ่งยูอาร์แอลนี้แสดงตำแหน่งที่อยู่ของเครื่องให้บริการเว็บและไฟล์ชุดคำสั่งภาษาเพิร์ล ซึ่งทำหน้าที่อ่านไฟล์ต้นฉบับภาษาจาวาและแปลงเป็นภาษาเอโอแอล จากนั้นทำการอ่านค่า IP Address ของเครื่องเพื่อเตรียมสร้างการติดต่อกับเครื่องให้บริการเว็บและค่า IP Address นี้ยังถูกใช้เป็นชื่อไดเร็กทอรี่สำหรับเก็บข้อมูลโปรเจ็คไฟล์และไฟล์ต้นฉบับภาษาจาวาในเครื่องให้บริการเว็บด้วย และเมื่อพร้อมแล้วโปรแกรมไคลเอนต์ Chula OOFP Counting จะสร้างการเชื่อมต่อไปยังเครื่องให้บริการเว็บเพื่อส่งคำสั่งสร้างไดเร็กทอรี่สำหรับการเก็บข้อมูลโปรเจ็คไฟล์และไฟล์ต้นฉบับภาษาจาวาที่กำลังถูกส่งมา จากนั้นจึงทำการส่งข้อมูลรายละเอียดของโปรเจ็คไฟล์ที่กำลังเปิดใช้อยู่ซึ่งข้อมูลประกอบไปด้วยชื่อไฟล์ต้นฉบับภาษาจาวาในโปรเจ็ค ชื่อคลาสทั้งหมดที่มีใช้งานและส่งข้อมูลหมายเลข IP Address ของเครื่องที่รันโปรแกรมไคลเอนต์ Chula OOFP Counting ไปสู่เครื่องให้บริการเว็บ โดยใช้ชุดคำสั่งภาษาเพิร์ลของโปรแกรม Chula OOFP Counting ฝั่งเซอร์ฟเวอร์ ต่อมาจึงเริ่มส่งไฟล์ต้นฉบับภาษาจาวาที่มีทั้งหมดในโปรเจ็คไฟล์ไปเก็บไว้ในเครื่องให้บริการเว็บและปิดการเชื่อมต่อ ต่อมาจึงสร้างการเชื่อมต่อใหม่ระหว่างโปรแกรมไคลเอนต์ Chula OOFP Counting กับเครื่องให้บริการเว็บเพื่อเรียกใช้ชุดคำสั่งภาษาเพิร์ลของโปรแกรม Chula OOFP Counting ฝั่งเซอร์ฟเวอร์เพื่อทำการวิเคราะห์ไฟล์ต้นฉบับภาษาจาวาและสร้างข้อมูลภาษาเอโอแอล ท้ายสุดทำการส่งข้อมูลภาษาเอโอแอลกลับสู่โปรแกรมไคลเอนต์และปิดการเชื่อมต่อ



รูปที่ 4.3 แผนภาพแสดงขั้นตอนการสร้างข้อมูลภาษาเอโอแอลจากไฟล์ต้นฉบับภาษาจาวา



รูปที่ 4.3 แผนภาพแสดงขั้นตอนการสร้างข้อมูลภาษาเอโอแอลจากไฟล์ต้นฉบับภาษาจาวา (ต่อ)

ชุดคำสั่งภาษาจาวาสำหรับขั้นตอนการสร้างข้อมูลภาษาเอโอแอลจากไฟล์ต้นฉบับภาษาจาวา มีรายละเอียดดังต่อไปนี้

4.4.1 ส่วนการอ่านค่ายูอาร์แอล จากไฟล์คอนฟิกกูเรชันและการรับค่าไอพีแอดเดรสจากเครื่องไคลเอนต์

รูปที่ 4.4 แสดงโค้ดการอ่านค่ายูอาร์แอล จากไฟล์คอนฟิกกูเรชันที่ชื่อ oofpapp_resource.properties โดยภายในไฟล์ดังกล่าวจะปรากฏบรรทัดที่ระบุยูอาร์แอลสำหรับการส่งไฟล์ต้นฉบับภาษาจาวาไปยังเครื่องให้บริการเว็บ

cgi_upstream=http://localhost/cgi-bin/OOFP/upload.pl

cgi_aol_extract=http://localhost/cgi-bin/OOFP/ExtractClassInfo.pl

ในตัวอย่างข้างต้นโดเมนเนม (Domain Name) สำหรับเครื่องให้บริการเว็บก็คือเครื่องที่รันโปรแกรม Chula OOFP Counting นั่นเองดังนั้นจึงปรากฏชื่อโดเมนเนมเป็น localhost และ upload.pl เป็นชื่อของไฟล์ชุดคำสั่งภาษาเพิร์ลที่ทำหน้าที่บันทึกไฟล์ต้นฉบับภาษาจาวาลงในเครื่องให้บริการเว็บหลังจากที่โปรแกรมไคลเอนต์ได้ส่งข้อมูลดังกล่าวให้ จากนั้นจึงถึงขั้นตอนการรับค่าหมายเลขไอพีแอดเดรสจากเครื่องไคลเอนต์เพื่อใช้ในการติดต่อกับเครื่องให้บริการเว็บและใช้เป็นชื่อไดเร็กทอรี่สำหรับเก็บข้อมูลโปรเจ็คไฟล์และไฟล์ต้นฉบับภาษาจาวาด้วย

```
// get "cgi_upstream=" string from oofpapp_resource.properties
strCGI = OOFPApp.AP_getResourceBundleString(OOFPResource.CGI_UPSTREAM);
if (strCGI == null)
{
    throw new Exception();
}
try
{
        // get IP address for using as owned directory in web server
        localHost = InetAddress.getLocalHost();
        strIPAddress = localHost.getHostAddress();
}
catch (java.net.UnknownHostException unknown)
{
        strIPAddress = "127.0.0.1";
}
m_modelOutputMsg.addElement(JMsg.JM_BUILD_2 + strIPAddress);
// Call cgi script for upload
strParam = new String[3];
strValue = new String[3];
strParam[0] = new String(OOFPResource.CGI_UPSTREAM_PARAM1); // cgi name
"filename".
strParam[1] = new String(OOFPResource.CGI_UPSTREAM_PARAM2); // cgi name
"content".
strParam[2] = new String(OOFPResource.CGI_UPSTREAM_PARAM3); // cgi name
"directory".
```

รูปที่ 4.4 โค้ดการอ่านค่ายูอาร์แอลจากไฟล์คอนฟิกกูเรชันและการรับค่าไอพีแอดเดรสจากเครื่องไคลเอนต์

4.4.2 สร้างการเชื่อมต่อระหว่างโปรแกรมไคลเอนต์ Chula OOFP Counting กับเครื่องให้บริการเว็บ คลาส JServConnect ทำหน้าที่ในการสร้างการเชื่อมต่อระหว่างโปรแกรมไคลเอนต์ Chula OOFP Counting กับเครื่องให้บริการเว็บ โดยส่งคำสั่งร้องขอใช้บริการของโปรโตคอลเอชทีทีพี (HTTP Request) แบบโพสต์ รูปที่ 4.5 แสดงรายละเอียดคลาส JServConnect โดยที่ฟังก์ชันการใช้งานของคลาสนี้มีเพียงฟังก์ชันเดียวคือ

public static String JS_cgiPost(String sCGI, String[] name, String[] value) throws IOException

พารามิเตอร์สำหรับฟังก์ชันนี้ประกอบไปด้วย

String sCGI : รับสตริงที่เก็บค่ายูอาร์แอลสำหรับการรับส่งข้อมูลระหว่างไคลเอนต์และเครื่องให้บริการเว็บ

String[] name: เก็บลิสต์ของชื่อพารามิเตอร์ที่ใช้กับยูอาร์แอลที่ระบุใน sCGI

String[] value: เก็บลิสต์ของค่าสำหรับพารามิเตอร์ที่ระบุใน name

```
class JServConnect
{
   public JServConnect() {}

  public static String JS_cgiPost(String sCGI, String[] name, String[] value) throws IOException
  {
        URL url = new URL(sCGI);
        URLConnection connection = url.openConnection();
        connection.setDoOutput(true);
        PrintWriter out = new PrintWriter(connection.getOutputStream());

        if (name != null && value != null) {
              int size = name.length;
              for (int i = 0; i < size; i++)
                    out.print(name[i] + "=" + URLEncoder.encode(value[i]) + "&");
              out.print("endpost=\n");
        }
        out.close();
        BufferedReader in;
        try {
            in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        }
       catch(FileNotFoundException eFile) {
              InputStream err = ((HttpURLConnection) connection).getErrorStream();
              if (err == null)
                    throw eFile;
              in = new BufferedReader(new InputStreamReader(err));
        }
        StringBuffer sRet = new StringBuffer();
        String line;
        while ((line = in.readLine()) != null)
        {
              sRet.append(line + "\n");
        }
        in.close();

        return sRet.toString();
   }
}
```

รูปที่ 4.5 โค้ดการเชื่อมต่อระหว่างโปรแกรมไคลเอนต์ Chula OOFP Counting กับเครื่องให้บริการเว็บ

สำหรับตัวอย่างการใช้งานคลาส JServConnect นั้นแสดงดังลิสต์ของชุดคำสั่งภาษาจาวาดังรูปที่ 4.6

```
strRet = JServConnect.JS_cgiPost(strCGI, strParam, strValue);

if (strRet.compareTo("OK\n") == 0) {
        // Upload completely.
}
else {
        // Upload error
}
```

รูปที่ 4.6 โค้ดตัวอย่างการวิธีการใช้งานคลาส JServConnect

4.4.3 การส่งข้อมูลรายละเอียดของโปรเจ็คไฟล์จากโปรแกรมไคลเอนต์ไปยังเครื่องให้บริการเว็บ

งานสำคัญในขั้นตอนนี้คือการส่งข้อมูลโปรเจ็คไฟล์ที่กำลังเปิดใช้อยู่ไปยังเครื่องให้บริการเว็บ ข้อมูลดังกล่าวประกอบด้วย ชื่อไฟล์ต้นฉบับภาษาจาวา ชื่อคลาสทั้งหมดที่มีในโปรเจ็ค หมายเลขไอพีแอดเดรสของเครื่องที่รันโปรแกรมไคลเอนต์ Chula OOFP Counting ซึ่งใช้เป็นชื่อไดเร็กทอรี่ในเครื่องให้บริการเว็บเพื่อเก็บข้อมูลโปรเจ็คไฟล์และไฟล์ต้นฉบับภาษาจาวา และเช่นเดียวกันไฟล์ต้นฉบับภาษาจาวาที่ระบุในโปรเจ็คไฟล์จะถูกส่งไปเก็บไว้ในเครื่องให้บริการเว็บเพื่อทำการแปลงเป็นภาษาเอโอแอล รายละเอียดของชุดคำสั่งที่ดำเนินงานในขั้นตอนนี้ดังรูป 4.7

```
strParam = new String[3];
strValue = new String[3];
strParam[0] = new String(OOFPResource.CGI_UPSTREAM_PARAM1); // cgi name "filename".
strParam[1] = new String(OOFPResource.CGI_UPSTREAM_PARAM2); // cgi name "content".
strParam[2] = new String(OOFPResource.CGI_UPSTREAM_PARAM3); // cgi name "directory".
////////////////////////////////////////////////////////////////////////////////
// send active project information to web server for creating the project file there.
// prepare content for project file
int count = m_clsActiveProject.m_vecFileLst.size();
int idx, n;
String sContent = "";
File obj;

for (idx = 0; idx < count; idx++) {
    obj = new File((String) m_clsActiveProject.m_vecFileLst.elementAt(idx)); // absolute path name
    n = obj.getAbsolutePath().indexOf('\\');
    if (n != -1) // the character occur
        // ./192.168.100.4/<directory>/<filename>
        sContent += ".\\" + strIPAddress + obj.getAbsolutePath().substring(n);
    else
        sContent += obj.getName();
    sContent += "\n";
}
// Perl is able to get '\' or'/' for directory separation
strValue[0] = m_clsActiveProject.m_sPrjName; // project name as for CGI param "filename"
strValue[1] = sContent;
strValue[2] = ".\\" + strIPAddress;
```

รูปที่ 4.7 โค้ดการส่งข้อมูลรายละเอียดของโปรเจ็คไฟล์จากโปรแกรมไคลเอนต์ไปยังเครื่องให้บริการเว็บ

```
// send post-HTTP request to web server for uploading
m_modelOutputMsg.addElement(JMsg.JM_BUILD_3);
m_modelOutputMsg.addElement(JMsg.JM_BUILD_4);
strRet = JServConnect.JS_cgiPost(strCGI, strParam, strValue);
if (strRet.compareTo("OK\n") == 0) {
    // Upload completely.
}
else {
   // Upload error
   m_modelOutputMsg.addElement(JMsg.JM_BUILD_6);
   throw new Exception();
}
//////////////////////////////////////////
// prepare content for all classes in active project
// send all class names in active project information to web server for creating the class file there.
count = m_vecClassListInActiveProject.size();
sContent = "";

for (idx = 0; idx < count; idx++)
{
    sContent += (String) m_vecClassListInActiveProject.elementAt(idx); // class name
    sContent += "\n";
}
strValue[0] = CLASS_FILE; // class file name as for CGI param "filename"
strValue[1] = sContent;
strValue[2] = ".\\" + strIPAddress;
// send post-HTTP request to web server for uploading
m_modelOutputMsg.addElement(JMsg.JM_BUILD_10);
strRet = JServConnect.JS_cgiPost(strCGI, strParam, strValue);

if (strRet.compareTo("OK\n") == 0)
{
   // Upload completely.
}
else
{
  // Upload error
   m_modelOutputMsg.addElement(JMsg.JM_BUILD_6);
   throw new Exception();
}
//////////////////////////////////////////
// upload all files in project to webserver
String sCurDir; // current directory
int nStart, nStop;

count = m_clsActiveProject.m_vecFileLst.size();
for (idx = 0; idx < count; idx++)
{
   obj = new File((String) m_clsActiveProject.m_vecFileLst.elementAt(idx)); // absolute path name
   strValue[0] = obj.getName();
   m_modelOutputMsg.addElement(JMsg.JM_BUILD_8 + obj.getAbsolutePath());
   // send post-HTTP request to web server for uploading
   strRet = JServConnect.JS_cgiPost(strCGI, strParam, strValue);
```

รูปที่ 4.7 โค้ดการส่งข้อมูลรายละเอียดของโปรเจ็คไฟล์จากโปรแกรมไคลเอนต์ไปยังเครื่องให้บริการเว็บ (ต่อ)

4.4.4 การเรียกใช้ชุดคำสั่งภาษาเพิร์ลเพื่อทำการแปลงไฟล์ต้นฉบับภาษาจาวาเป็นข้อมูลภาษาเอโอแอล

ในขั้นตอนนี้โปรแกรมไคลเอนต์ Chula OOFP Counter จะสร้างการเชื่อมต่อไปยังเครื่องให้บริการเว็บอีกครั้งเพื่อเรียกใช้ชุดคำสั่งภาษาเพิร์ลให้ทำการแปลงไฟล์ต้นฉบับภาษาจาวาเป็นข้อมูลภาษาเอโอแอล จากนั้นจะส่งข้อมูลภาษาเอโอแอลกลับสู่โปรแกรมไคลเอนต์และปิดการเชื่อมต่อ รายละเอียดของชุดคำสั่งดังรูปที่ 4.8

```
////////////////////////////////////////////////
// extract AOL specification for all file/class on active project
strParam = null;
strValue = null;

// call cgi script for AOL extraction
// get "cgi_aol_extract=" string from oofpapp_resource.properties
strCGI = OOFPApp.AP_getResourceBundleString(OOFPResource.CGI_AOL_EXTRACT);
if (strCGI == null)
{
    m_modelOutputMsg.addElement(JMsg.JM_ERROR_6);
    throw new Exception();
}

strParam = new String[3];
strValue = new String[3]; // cgi value parameter
try
{
    // Need the project file as an input of aol extraction
 strParam[0] = new String(OOFPResource.CGI_AOL_EXTRACT_PARAM1); // cgi name "project_file".
 strParam[1] = new String(OOFPResource.CGI_AOL_EXTRACT_PARAM2); // cgi name "class_file".
 strParam[2] = new String(OOFPResource.CGI_AOL_EXTRACT_PARAM3); // cgi name "directory".

 obj = new File(m_clsActiveProject.m_sPrjName);  strValue[0] = obj.getName(); // project file
 obj = new File(CLASS_FILE);                     strValue[1] = obj.getName(); // class file
                                                 strValue[2] = strIPAddress;     // directory
  m_modelOutputMsg.addElement(JMsg.JM_BUILD_12);
  // send post-HTTP request to web server for aol extraction
  strRet = JServConnect.JS_cgiPost(strCGI, strParam, strValue);
  // get AOL specification
  JUtils.writeToFile(AOL_FILE, strRet);
  m_modelOutputMsg.addElement(JMsg.JM_BUILD_11);
}
catch (Exception exception)
{
   m_modelOutputMsg.addElement(JMsg.JM_BUILD_7);
   throw new Exception();
}
```

รูปที่ 4.8 โค้ดการเรียกใช้ชุดคำสั่งภาษาเพิร์ลเพื่อวิเคราะห์ไฟล์ต้นฉบับภาษาจาวาและสร้างข้อมูลภาษาเอโอแอล

4.5 การนับจำนวนฟังก์ชันพอยต์เชิงวัตถุจากข้อมูลภาษาเอโอแอล

สำหรับขั้นตอนนี้ ต้องใช้ข้อมูลภาษาเอโอแอลที่ได้จากหัวข้อ 4.4 เป็นพื้นฐานในการนับ

4.5.1 การสร้างแพ็กเกจคอมไพเลอร์ภาษาเอโอแอล

จากที่กล่าวข้างต้นแล้วว่าการนับจำนวนของฟังก์ชันพอยต์เชิงวัตถุเป็นจุดประสงค์หลักของงานวิจัยนี้ การวิเคราะห์และการคำนวณจะทำบนข้อมูลภาษาเอโอแอล ซึ่งเป็นภาษากลางที่ถูกออกแบบมาเพื่อใช้อธิบายโมเดลเชิงวัตถุในรูปแบบของข้อความตัวอักษร ซึ่งทำให้ง่ายเมื่อถูกนำไปใช้ในการวิเคราะห์และออกแบบการประมวลผลในรูปแบบต่างๆเพื่อขยายแนวความคิดในการประยุกต์ใช้โมเดลเชิงวัตถุต่อไป ซึ่งงานวิจัยนี้เป็นหนึ่งในงานประยุกต์ใช้ภาษาเอโอแอล เพื่อคำนวณหาจำนวนของฟังก์ชันพอยต์เชิงวัตถุ (สามารถศึกษารายละเอียดของไวยากรณ์และตัวอย่างของ ภาษาเอโอแอล ได้ในภาคผนวก ก)

เครื่องมือที่นำมาใช้ในการประมวลผลโทเคนและไวยากรณ์ภาษาเอโอแอล ได้แก่โปรแกรมจาวาซีซี เวอร์ชัน 2.1 โดยการประมวลผลจะเริ่มจากการสร้างไฟล์ที่ทำหน้าที่อธิบายทุกๆโทเคนและทุกๆไวยากรณ์ที่ใช้ในภาษาเอโอแอล โดยไฟล์ที่ได้นี้เรียกว่า ข้อกำหนดของภาษาเอโอแอล (AOL Specification) โดยอยู่ในรูปแบบของไฟล์ไวยากรณ์จาวาซีซี จากนั้นนำไฟล์ข้อกำหนดนี้มาประมวลผลด้วยชุดเครื่องมือจาวาซีซี โดยเครื่องมือดังกล่าวจะประมวลผลโทเคนและไวยากรณ์ภาษาเอโอแอล ที่กำหนดและสร้างไฟล์ต้นฉบับภาษาจาวาจำนวนหนึ่งเพื่อใช้ในการคอมไพล์ไฟล์ข้อมูลภาษาเอโอแอล ตามหลักไวยากรณ์ที่กำหนด ในขณะเดียวกันขั้นตอนการนับจำนวนฟังก์ชันพอยต์เชิงวัตถุจะถูกทำในระหว่างการประมวลผลโทเคนและไวยากรณ์ เพื่อให้ขั้นตอนการนับจำนวนฟังก์ชันพอยต์เชิงวัตถุเกิดขึ้นได้ ภายในไฟล์ข้อกำหนดของภาษา เอโอแอล ต้องเพิ่มส่วนการวิเคราะห์โทเคนเพื่อนำมาใช้ประกอบการนับ โดยรายละเอียดการเพิ่มเติมเหล่านี้จะปรากฏในหัวข้อ 4.5.2 ไฟล์ต้นฉบับภาษาจาวาที่ถูกสร้างโดยโปรแกรมจาวาซีซีมีอยู่ด้วยกันหลายไฟล์ อาทิ ไฟล์เอโอแอลพาร์เซอร์ (AOLParser) ไฟล์เอโอแอลพาร์เซอร์คอนสแตนท์ (AOLParserConstants.java) ไฟล์โทเคน (Token.java) และไฟล์เอโอแอลพาร์เซอร์โทเคนเมเนเจอร์ (AOLParserTokenManager.java) เป็นต้น เมื่อได้ไฟล์สำหรับการคอมไพล์ภาษา เอโอแอล แล้วไฟล์เหล่านี้จะถูกคอมไพล์โดยโปรแกรมจาวาคอมไพเลอร์ (javac.exe) ซึ่งเป็นคอมไพเลอร์ภาษาจาวาของจาวาดีเวลลอปเมนต์คิด การสร้างแพ็กเกจคอมไพเลอร์ภาษา เอโอแอล มีขั้นตอนดังรูปที่ 4.2

4.5.2 การสร้างไฟล์ต้นฉบับสำหรับอธิบายไวยากรณ์ภาษาเอโอแอล พร้อมกับฟังก์ชันและคลาสต่างๆที่ใช้ประกอบการนับจำนวนของฟังก์ชันพอยต์เชิงวัตถุ

การสร้างเริ่มจากพิจารณาโทเคนและไวยากรณ์ทั้งหมดของภาษาเอโอแอล แล้วทำการสร้างไฟล์แสดงไวยากรณ์ภาษา เอโอแอล ตามหลักวิธีการของโปรแกรมจาวาซีซี ทุกๆโทเคนในไฟล์แสดงไวยากรณ์จะเป็นไปตามกฎเกณฑ์ของภาษาจาวา (เนื่องจากโปรแกรมจาวาซีซี อิงภาษาจาวาเป็นหลักในการสร้างไฟล์ต้นฉบับที่เป็นคอมไพเลอร์) เหตุนี้ทำให้การประกาศตัวแปร สตริงตัวอักษร อักขระ การสร้างฟังก์ชันพิเศษต่างๆ จะเป็นไปตาม

ข้อกำหนดกฎเกณฑ์ของภาษาจาวาทั้งสิ้น (วิธีการเขียนไฟล์ไวยากรณ์สำหรับโปรแกรมจาวาซีซีสามารถศึกษาได้จากเอกสารคู่มือจากบริษัทผู้ผลิต)

นอกจากนี้ภายในไฟล์แสดงไวยากรณ์ภาษาเอโอแอลยังประกอบด้วยคลาสและฟังก์ชันพิเศษจำนวนมากที่ช่วยในการนับจำนวนของฟังก์ชันพอยต์เชิงวัตถุ ในที่นี้จะแสดงรายละเอียดเฉพาะคลาสและฟังก์ชันหลักในไฟล์แสดงไวยากรณ์ภาษาเอโอแอล (AOL.jjt) โดยหัวข้อที่อธิบายได้แก่

- ค่าตัวเลือกซึ่งถูกใช้โดยโปรแกรมจาวาซีซี
- คลาสสำหรับการประมวลผลโทเคนและไวยากรณ์
- คลาสสำหรับเก็บรายละเอียดของคลาสและความสัมพันธ์ระหว่างคลาส
- โทเคนที่ประกาศใช้สำหรับไวยากรณ์ของภาษาเอโอแอล
- คลาสและฟังก์ชันประกอบการคอมไพล์ภาษาเอโอแอล และการคำนวณหาจำนวนของฟังก์ชันพอยต์เชิงวัตถุ

4.5.2.1 ค่าตัวเลือกซึ่งถูกใช้โดยโปรแกรมจาวาซีซี

ภายในไฟล์ไวยากรณ์ (Grammar File) ภาษาเอโอแอล จะเริ่มต้นด้วยลิสต์ของค่าตัวเลือกต่างๆ ซึ่งจะระบุหรือไม่ก็ได้ แต่ถ้าไม่มีการระบุ ค่าดีฟอลต์ของตัวเลือกจะถูกนำมาใช้ ค่าตัวเลือกที่ถูกใช้ในไฟล์ไวยากรณ์ภาษาเอโอแอล ดังรูปที่ 4.9

```
options
{
        JAVA_UNICODE_ESCAPE = true;
        STATIC              = false;
        MULTI               = true;
        IGNORE_CASE         = true;
}
```

รูปที่ 4.9 โค้ดแสดงค่าตัวเลือกซึ่งถูกใช้โดยโปรแกรมจาวาซีซี

ความหมายของค่าตัวเลือกข้างต้นมีรายละเอียดดังต่อไปนี้

JAVA_UNICODE_ESCAPE: เป็นค่าตัวเลือกตรรกะ (Boolean) โดยมีค่าปริยายเป็นเท็จ (False) เมื่อกำหนดให้ค่านี้เป็นจริง โปรแกรมพาร์เซอร์ที่ถูกสร้างขึ้นเมื่ออ่านข้อมูลเข้าเข้ามาจะต้องผ่านกระบวนการแปลงอักขระพิเศษที่อยู่ในรูปจาวายูนิโค้ดเอสเคป (Java Unicode Escape: "\u") ให้เป็นอักขระยูนิโค้ดปกติ ก่อนการส่งชุดอักขระเข้าเหล่านี้ไปยังโทเคนเมเนเจอร์ (Token Manager) ต่อไป

STATIC: เป็นค่าตัวเลือกตรรกะ (Boolean) โดยมีค่าปริยายเป็นจริง (True) เมื่อกำหนดให้ค่านี้เป็นจริงทุกๆเมธอดและแอตริบิวของคลาสจะถูกกำหนดให้เป็น สแตติก (Static) เสมอ ในกลุ่มไฟล์พาร์เซอร์และโทเคนเมเนเจอร์ที่ถูกสร้างขึ้น สิ่งนี้ทำให้สามารถมีวัตถุพาร์เซอร์ได้เพียง 1 ตัวเท่านั้น และถ้าต้องการให้ 1 วัตถุพาร์เซอร์ในโปรแกรมสามารถประมวลผลโทเคนและไวยากรณ์ของภาษาได้หลายๆครั้ง ต้องเรียกใช้เมธอด ReInit() เสมอ

เพื่อเริ่มต้นกระบวนการใหม่อีกครั้ง ซึ่งข้อดีก็คือทำให้การใช้ทรัพยากรของระบบน้อยลง แต่เมื่อกำหนดให้ค่านี้เป็นเท็จ จะสามารถสร้างวัตถุพาร์เซอร์ได้มากเท่าที่ต้องการ

IGNORE_CASE: เป็นค่าตัวเลือกตรรกะ (Boolean) โดยมีค่าปริยายเป็นเท็จ เมื่อกำหนดให้ค่านี้เป็นจริงจะทำให้โทเคนเมเนเจอร์ที่ถูกสร้างขึ้นไม่พิจารณาอักขระว่าเป็นตัวพิมพ์ใหญ่หรือตัวพิมพ์เล็กในการวิเคราะห์โทเคน ซึ่งค่าตัวเลือกนี้จะมีประโยชน์กับการเขียนไวยากรณ์ของภาษาอย่าง HTML เป็นต้น และเมื่อกำหนดค่านี้เป็นเท็จ โทเคนเมเนเจอร์จะกระทำในสิ่งที่ตรงกันข้ามกับที่กล่าวข้างต้น

MULTI: เป็นค่าตัวเลือกตรรกะ (Boolean) โดยมีค่าปริยายเป็นเท็จ เมื่อกำหนดให้ค่านี้เป็นจริง โปรแกรมจาวาซีซีจะสร้างทรีในลักษณะมัลติโหมด (multi mode parse tree) แต่ถ้ากำหนดเป็นเท็จจะเป็นการสร้างทรีธรรมดาในโปรแกรมพาร์เซอร์

ดังนั้นจากค่าตัวเลือกที่ได้กำหนด วัตถุพาร์เซอร์และโทเคนเมเนเจอร์ที่ใช้ในคอมไพเลอร์ภาษา เอโอแอล จะมีคุณลักษณะดังนี้

- ต้องมีกระบวนการแปลงอักขระพิเศษที่อยู่ในรูปจาวายูนิโค้ดเอสเคป (Java Unicode Escape: "\u") ให้เป็นอักขระยูนิโค้ดปกติ ก่อนการส่งชุดอักขระเข้าเหล่านี้ไปยังโทเคนเมเนเจอร์ (Token Manager)
- สามารถสร้างวัตถุพาร์เซอร์ได้มากเท่าที่ต้องการเพื่อใช้ในการประมวลผลโทเคนและไวยากรณ์ภาษา เอโอแอล
- โทเคนเมเนเจอร์จะไม่พิจารณาอักขระว่าเป็นตัวพิมพ์ใหญ่หรือตัวพิมพ์เล็กในการวิเคราะห์โทเคนของภาษา เอโอแอล

4.5.2.2 คลาสสำหรับการประมวลผลโทเคนและไวยากรณ์

ภายในไฟล์ไวยากรณ์ ภาษาเอโอแอล ถัดจากค่าตัวเลือกดังกล่าวข้างต้นแล้ว จะเป็นส่วนที่เรียกว่าจาวาคอมพิเลชันยูนิต (Java Compilation Unit) ซึ่งหมายถึงส่วนของโค้ดโปรแกรมภาษาจาวานั่นเอง โดยจะถูกครอบระหว่างบรรทัด "PARSER_BEGIN(name)" และ "PARSER_END(name)" และหลังจากบรรทัดเหล่านี้จะเป็นลิสต์ของกลุ่มฟังก์ชันที่เรียกว่า แกรมมาโปรดักชัน (Grammar Production) ซึ่งเป็นส่วนที่ใช้แสดงไวยากรณ์ของภาษา เอโอแอล นั่นเอง โดย name ที่ตามหลัง "PARSER_BEGIN" และ "PARSER_END" ต้องเหมือนกันและต้องเป็นชื่อของคลาสพาร์เซอร์ที่จะถูกสร้างขึ้น จากไฟล์ไวยากรณ์ภาษา เอโอแอล ที่พัฒนา name ก็คือชื่อคลาส AOLParser และกลุ่มของไฟล์ต่อไปนี้จะถูกสร้างขึ้นมาโดยโปรแกรมจาวาซีซี

AOLParser.java : เป็นไฟล์ต้นฉบับภาษาจาวาสำหรับคลาสพาร์เซอร์ (Parser)

AOLParserTokenManager.java : เป็นไฟล์ต้นฉบับภาษาจาวาสำหรับคลาสโทเคนเมเนเจอร์ (Token Manager)

AOLParserConstants.java : เป็นไฟล์ต้นฉบับภาษาจาวาซึ่งเก็บค่าคงที่ต่างๆที่ใช้

นอกจากนี้ยังมีไฟล์อื่นๆ อาทิ Token.java ParserError.java เป็นต้นที่ถูกสร้างขึ้นมาด้วย อย่างไรก็ตาม ไฟล์ต้นฉบับภาษาจาวาเหล่านี้จะเหมือนๆกันสำหรับไวยากรณ์ของทุกๆภาษา ระหว่างบรรทัด "PARSER_BEGIN" และ "PARSER_END" จะเป็นโค้ดโปรแกรมภาษาจาวาธรรมดา ซึ่งอาจจะเป็นการประกาศคลาสใช้งานเพิ่มเติม ประกาศเมธอดในคลาสใดๆ รวมถึงการประกาศใช้ตัวแปรแอตริบิวในคลาส เป็นต้น ดังนั้นกล่าวได้ว่า โดยทั่วไปส่วนที่กำลังกล่าวถึงนี้ จะปรากฏในไฟล์ไวยากรณ์ในลักษณะ

```
PARSER_BEGIN(parser_name)
.....
class parser_name ... {
.....
}
.....
PARSER_END(parser_name)
```

โดยในไวยากรณ์ภาษา เอโอแอล นี้ parser_name จะถูกแทนที่ด้วย AOLParser รายละเอียดดังโค้ดตัวอย่างในรูปที่ 4.10

```
PARSER_BEGIN(AOLParser)
public class AOLParser
{
  public DefaultListModel          m_vecOutput;
  // Identifying logical file
  // 1. Single Class
  final static int SC_TYPE = 0;
  // 2. Aggregation
  final static int AB_TYPE = 1;
  // 3. Generalization / Specialization
  final static int GB_TYPE = 2;
  // 4. Mixed (GB + AB)
  final static int MB_TYPE = 3;
   // constant complexity
   private int LOW_CPX = 0; // low complexity
   private int AVG_CPX = 1; // average complexity
   private int HGH_CPX = 2; // high complexity
   private int m_nLF_TOT_LOW = 0; // low complexity accumulator (ILF/EIF)
   private int m_nLF_TOT_AVG = 0; // average complexity accumulator (ILF/EIF)
   private int m_nLF_TOT_HGH = 0; // high complexity accumulator (ILF/EIF)
   private int m_nSR_TOT_LOW = 0; // low complexity accumulator (SR)
   private int m_nSR_TOT_AVG = 0; // average complexity accumulator (SR)
   private int m_nSR_TOT_HGH = 0; // high complexity accumulator (SR)
   // Logical File (LF) counter
   private int m_nLF_ACC_DET = 0;
   private int m_nLF_ACC_RET = 0;
   // Service Request (SR) counter
   private int m_nSR_ACC_DET = 0;
   private int m_nSR_ACC_FTR = 0;
// this hashtable stores <Class Name (key), JClassInfo Object (value)> for "CLASS" in an AOL file
  private Hashtable m_hashClsInfo        = new Hashtable();
```

รูปที่ 4.10 รายละเอียดคลาสสำหรับการประมวลผลโทเคนและไวยากรณ์

```
// this hashtable stores <Container Class Name (key), JCLSAggregationInfo Object (value)> for
// AGGREGATION in an AOL file
  private Hashtable m_hashAggreClsInfo = new Hashtable();
// this hashtable stores <Child Class Name (key), JCLSGeneralizationInfo Object (value)> for
// GENERALIZATION/SPECIALIZATION in an AOL file
  private Hashtable m_hashGeneralizationClsInfo = new Hashtable();

 // this vector contains all class name embedded in any container
  private Vector m_vecEmbeddedObject= new Vector();
 // this vector contains all class name based on any subclasses
  private Vector m_vecBasedClass                = new Vector();
 // for showing in summary counting table
  private Object[][]                m_tableData;
   public AOLParser(String sFileName, DefaultListModel model, Object[][] tableData) throws
Exception
  {
        this(new FileInputStream(sFileName));

        m_vecOutput = model;
        m_tableData = tableData;

        try
        {
                CompilationUnit();
        }
        catch (ParseException e)
        {
                e.printStackTrace();
                throw new Exception("AOL Parser:  Encountered errors during parse.");
        }
   }
}
```

รูปที่ 4.10 รายละเอียดคลาสสำหรับการประมวลผลโทเคนและไวยากรณ์ (ต่อ)

4.5.2.3 คลาสสำหรับเก็บรายละเอียดของคลาสและความสัมพันธ์ระหว่างคลาส

คลาส JClassInfo เป็นคลาสเฉพาะที่ถูกใช้งานเกี่ยวกับฟังก์ชันคำนวณฟังก์ชันพอยต์ โดยทำหน้าที่เก็บข้อมูลในส่วน ลอจิคอลไฟล์ ในฟังก์ชันพอยต์เชิงวัตถุซึ่งในที่นี้ก็คือค่า LF DET และ RET นอกจากนี้ยังเก็บข้อมูลในส่วน เซอร์วิสรีเควส (Service Request) ซึ่งได้แก่ค่า SR DET และ FTR โดยแต่ละคลาสที่ปรากฏในไฟล์เอโอแอล ในระหว่างขั้นตอนการนับฟังก์ชันพอยต์จะมีการเก็บข้อมูลเหล่านี้ รายละเอียดของคลาสดังรูป 4.11

```
class JClassInfo
{
        public String     m_sClsName; // class name
        public int        m_nLF_DET; // logical file (det)
        public int        m_nLF_RET; // logical file (ret)
        public int        m_nSR_DET; // service request (det)
        public int        m_nSR_FTR; // service request (ftr)

        JClassInfo()
        {
                m_sClsName    = "";
                m_nLF_DET     = 0;
                // one RET is associated to each ILF/EIF, because it represents a
                // "user recognizable group of logically related data"
                m_nLF_RET     = 1;
                m_nSR_DET     = 0;
                m_nSR_FTR     = 0;
        }
}
```

รูปที่ 4.11 โค้ดแสดงคลาสที่ทำหน้าที่เก็บข้อมูลรายละเอียดของคลาสในไฟล์ เอโอแอล

สำหรับการพิจารณาข้อมูลในส่วน ลอจิคอลไฟล์ (Logical File: LF) ในฟังก์ชันพอยต์เชิงวัตถุซึ่งในที่นี้ก็คือค่า LF Data Element Types (DET) และ Record Element Types (RET) มีรายละเอียดการพิจารณาดังนี้

โมเดลเชิงวัตถุหรือแผนภาพคลาสของระบบจะถูกนำมาวิเคราะห์เพื่อพิจารณาหากลุ่มของคลาสในระบบสำหรับใช้ในการวัด ซึ่งมีอยู่ด้วยกัน 4 วิธี

1) แบบซิงเกิ้ลคลาส

โดยที่วิธีการพิจารณาในลักษณะหนึ่งคลาสเปรียบเสมือนหนึ่งลอจิคอลไฟล์ โดยจะแยกแต่ละคลาสออกจากกัน ไม่มีการพิจารณาความสัมพันธ์ระหว่างคลาสไม่ว่าจะเป็นแบบแอกกรีเกชันหรือเจนเนอร์รัลไลเซชัน ดังรูปที่ 2.4

2) แบบแอกกรีเกชัน

เป็นวิธีการพิจารณาในลักษณะความสัมพันธ์ของคลาสในลักษณะการฝังตัวหรือเป็นส่วนประกอบหนึ่งของคลาสอื่นๆ ชุดของคลาสที่มีลักษณะดังกล่าวจะถูกพิจารณาเสมือนหนึ่งลอจิคอลไฟล์ ดังรูปที่ 2.5

3) แบบเจนเนอร์รัลไลเซชัน / สเปเชี่ยลไลเซชัน

เป็นวิธีการพิจารณาความสัมพันธ์ของคลาสในลักษณะสืบทอด กล่าวคือ ถ้ามีความสัมพันธ์ของคลาสเกิดขึ้นในลักษณะนี้ คลาสต่างๆที่มีการสืบทอดกันมาตั้งแต่รากจนกระทั่งปลายสุดของแต่ละเส้นทางการสืบทอดจะถูกพิจารณาเสมือนเป็นหนึ่งลอจิคอลไฟล์ ดังรูปที่ 2.6

4) แบบมิกซ์

เป็นวิธีการพิจารณาที่ใช้ทั้งแบบแอกกรีเกชันและเจนเนอร์รัลไลเซชัน / สเปเชี่ยลไลเซชัน ร่วมกัน

และในขั้นตอนต่อมา ทำการพิจารณานับฟังก์ชันพอยต์เชิงวัตถุของลอจิคอลไฟล์และเซอร์วิสรีเควสซึ่งมีรายละเอียดดังต่อไปนี้

ลอจิคอลไฟล์ (Logical File)

การพิจารณาแผนภาพคลาสในแบบแอกกรีเกชัน เจนเนอร์รัลไลเซชัน / สเปเชี่ยลไลเซชัน และแบบมิกซ์ คลาสที่ถูกจัดกลุ่มในลักษณะดังกล่าวเรียกว่า คอมโพสิตลอจิคอลไฟล์ (Composite Logical File) และแผนภาพคลาสที่ถูกพิจารณาแบบซิงเกิ้ลคลาส คลาสเหล่านั้นจะถูกเรียกว่า ซิมเพิ้ลลอจิคอลไฟล์ (Simple Logical File) แต่ละลอจิคอลไฟล์จะถูกคำนวณหาจำนวนของ DET (Data Element Types) และ RET (Record Element Types) กฎเกณฑ์การคำนวณหาค่า DET และ RET จะแตกต่างกันไปเล็กน้อยสำหรับลอจิคอลไฟล์แบบซิมเพิ้ลและแบบคอมโพสิต โดยทั้งสองกรณี จะมีอยู่ 1 RET แน่นอนสำหรับแต่ละลอจิคอลไฟล์ เพราะแต่ละลอจิคอลไฟล์แสดงถึงกลุ่มของข้อมูลตามกฎเกณฑ์ของฟังก์ชันพอยต์ ที่ว่า "user recognizable group of logically related data" เมื่อ DET และ RET ของแต่ละลอจิคอลไฟล์ได้ถูกนับ ตารางแสดงความซับซ้อนของลอจิคอลไฟล์จะถูกนำมาพิจารณาหาค่าความซับซ้อนของลอจิคอลไฟล์หนึ่งๆว่าอยู่ในเกณฑ์ ต่ำ เฉลี่ย หรือสูง ดังตารางที่ 4.1

ตารางที่ 4.1 ตารางแสดงความซับซ้อนของลอจิคอลไฟล์ (Logical Complexity Matrix) [5]

	0 to 19 DET	**20 to 50 DET**	**51 or more DET**
1 RET	Low	Low	Average
2 to 5 RET	Low	Average	High
6 or more RET	Average	High	High

ลอจิคอลไฟล์มีอยู่ด้วยกัน 2 ประเภทได้แก่ ซิมเพิ้ลลอจิคอลไฟล์และคอมโพสิตลอจิคอลไฟล์

ซิมเพิ้ลลอจิคอลไฟล์ (Simple Logical File): แอตริบิวของคลาสที่เป็นชนิดข้อมูลพื้นฐาน (Primitive Data Type) เช่น เลขจำนวนเต็ม (integer) และสตริง (string) จะถูกพิจารณาเป็น DET เพราะ DET หมายถึง "unique user recognizable, non-recursive field of the logical file" ความสัมพันธ์ระหว่างคลาสแบบแอกรีเกชัน ต้องถูกนับเช่นเดียวกัน เพราะว่าความสัมพันธ์ลักษณะนี้ก่อให้เกิดความซับซ้อนมากขึ้นในคลาส ซึ่งปกติวัตถุมักจะถูกประกาศเป็นสมาชิกในคลาสไว้ (Data Member) ซึ่งแน่นอนว่าวัตถุดังกล่าวสามารถถูกเรียกใช้ในเมธอดต่างๆ ของคลาสเพื่อเรียกใช้บริการต่างๆของมัน ดังนั้นจึงถูกพิจารณาเป็น RET เพราะ RET หมายถึง "a user recognizable subgroup of data elements within a logical file"

คอมโพสิตลอจิคอลไฟล์ (Composite Logical File): DET และ RET ถูกนับในแต่ละคลาสภายในคอมโพสิตและนำผลที่ได้จากแต่ละคลาสในคอมโพสิตมารวมกันเพื่อให้ได้ผลรวมทั้งหมดของ DET และ RET ในคอมโพสิตหนึ่งๆ วิธีการนับ DET และ RET จะใช้กฎเกณฑ์เดียวกับซิมเพิ้ลลอจิคอลไฟล์ ยกเว้นถ้ามีความสัมพันธ์แบบแอกกรีเกชันในคอมโพสิตเกิดขึ้น วิธีการนับต้องต่างออกไปเพราะว่าในคอมโพสิตลอจิคอลไฟล์หนึ่งๆ จะถูกมองว่าเป็นกลุ่มย่อยที่เก็บข้อมูล (Subgroup) ดังนั้น 1 RET จะถูกนับแน่นอนสำหรับแต่ละแอกรีเกชัน และค่า RET นี้จะถูกบวกเพิ่มเข้าไปในคลาสคอนเทนเนอร์

ในทางปฏิบัติค่าของ DET และ RET สำหรับลอจิคอลไฟล์ใดๆจะถูกคำนวณโดยการนับจำนวน DET และ RET ของแต่ละคลาสที่ลอจิคอลไฟล์นั้นเป็นเจ้าของอยู่ แล้วนำจำนวน DET และ RET ของแต่ละคลาสดังกล่าวมาบวกรวมกันเพื่อให้เป็นค่า DET และ RET สำหรับลอจิคอลไฟล์

เซอร์วิสรีเควส (Service Request)

เซอร์วิลรีเควสสามารถเรียกได้อีกอย่างหนึ่งว่าเมธอด แต่ละเซอร์วิสรีเควสในแต่ละคลาสในระบบที่กำลังพิจารณา เมธอดแบบ Abstract จะไม่ถูกนับ มีแต่เฉพาะเมธอดจริงที่ถูกนับในคลาสที่ประกาศใช้และถูกนับเป็น 1 เมธอดเท่านั้น ถึงแม้ว่าเมธอดเหล่านี้จะถูกสืบทอดไปยังคลาสลูกก็ตาม แต่ก็ยังนับเป็น 1 เมธอดเพราะว่าจริงๆ แล้วโค้ดของแต่ละเมธอดมีอยู่เพียง 1 แห่งเท่านั้นไม่ได้มีตามจำนวนของคลาสที่สืบทอดไป เมื่อพิจารณาพารามิเตอร์ของเมธอด ชนิดข้อมูลของพารามิเตอร์จะถูกพิจารณาดังนี้

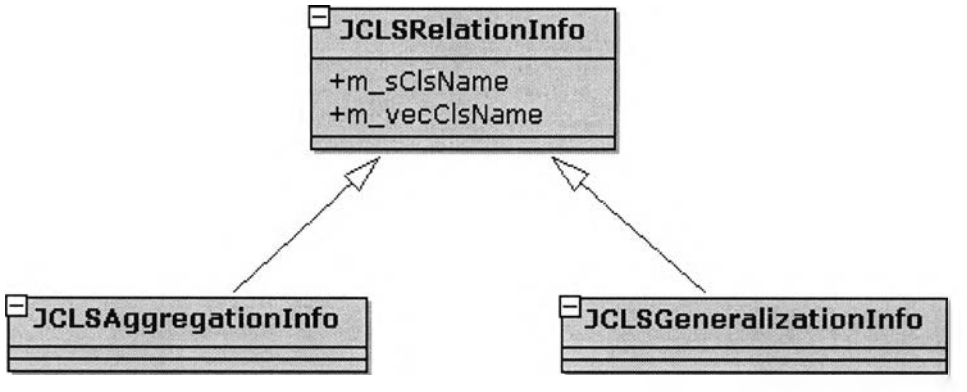
- พารามิเตอร์ที่มีชนิดข้อมูลพื้นฐานจะถูกพิจารณาเป็น DET
- พารามิเตอร์ที่มีชนิดข้อมูลเป็นคลาสหรืออ้างอิงไปยังคลาสจะถูกพิจารณาเป็น FTR

เมื่อ Data Element Types (DET) และ File Types Referenced (FTR) ของเมธอดถูกนับ ตารางที่ 4.2 จะถูกนำมาใช้พิจารณาความซับซ้อนของเมธอด

ตารางที่ 4.2 ตารางแสดงความซับซ้อนของเซอร์วิสรีเควส [5]

	0 to 4 DET	**5 to 15 DET**	**16 or more DET**
0 to 1 FTR	Low	Low	Average
2 FTR	Low	Average	High
3 or more FTR	Average	High	High

กลุ่มของคลาสในรูปที่ 4.12 ทำหน้าที่เก็บข้อมูลความสัมพันธ์ระหว่างคลาสต่างๆที่พิจารณาเพื่อเป็นไปตามวิธีพิจารณาโมเดลเชิงวัตถุหรือแผนภาพคลาสที่กล่าวข้างต้น โดยมีคลาส JCLSRelationInfo เป็นคลาสแม่ และมีคลาส JCLSAggregationInfo และคลาส JCLSGeneralizationInfo เป็นคลาสที่สืบทอดลงมาจากคลาส JCLSRelationInfo รายละเอียดโค้ดของคลาสที่ทำหน้าที่เก็บข้อมูลความสัมพันธ์ระหว่างคลาสในไฟล์ เอโอแอล ดังรูปที่ 4.13



รูปที่ 4.12 แผนภาพคลาสที่ทำหน้าที่เก็บข้อมูลความสัมพันธ์ระหว่างคลาสต่างๆในไฟล์ เอโอแอล

```
class JCLSRelationInfo
{
        public String     m_sClsName; // container class
        public Vector     m_vecClsName; // list of aggregated class in one layer
        JCLSRelationInfo()
        {
                m_sClsName = "";
                m_vecClsName = new Vector();
        }
}
class JCLSAggregationInfo extends JCLSRelationInfo
{
        JCLSAggregationInfo()  { super(); }
}
class JCLSGeneralizationInfo extends JCLSRelationInfo
{
    JCLSGeneralizationInfo() { super(); }
}
```

รูปที่ 4.13 โค้ดแสดงคลาสที่ทำหน้าที่เก็บข้อมูลความสัมพันธ์ระหว่างคลาสในไฟล์ เอโอแอล

4.5.2.4 โทเคนที่ประกาศใช้สำหรับไวยากรณ์ของภาษา เอโอแอล

จากที่กล่าวแล้วว่าภาษาเอโอแอล เป็นภาษาที่สามารถอธิบายโมเดลเชิงวัตถุหรือแผนภาพคลาสได้ในรูปแบบของข้อความและคำอธิบาย และโปรแกรม Chula OOFP Counting ได้นำภาษาเอโอแอล นี้มาใช้เป็นภาษากลางสำหรับการเชื่อมต่อระหว่างโมเดลเชิงวัตถุกับการคำนวณหาจำนวนของฟังก์ชันพอยต์เชิงวัตถุ ดังนั้นภาษาเอโอแอล จึงเป็นเครื่องมือสำคัญในงานวิจัยนี้ รายละเอียดของภาษาเอโอแอล สามารถพิจารณาได้จากภาคผนวก ก ซึ่งมีอยู่ในรูปบีเอนเอฟ (BNF: Backus Naur Form) พร้อมทั้งตัวอย่างของภาษาเอโอแอล ในรูปที่ 4.14 แสดงโทเคนต่างๆที่ใช้ในภาษาเอโอแอล

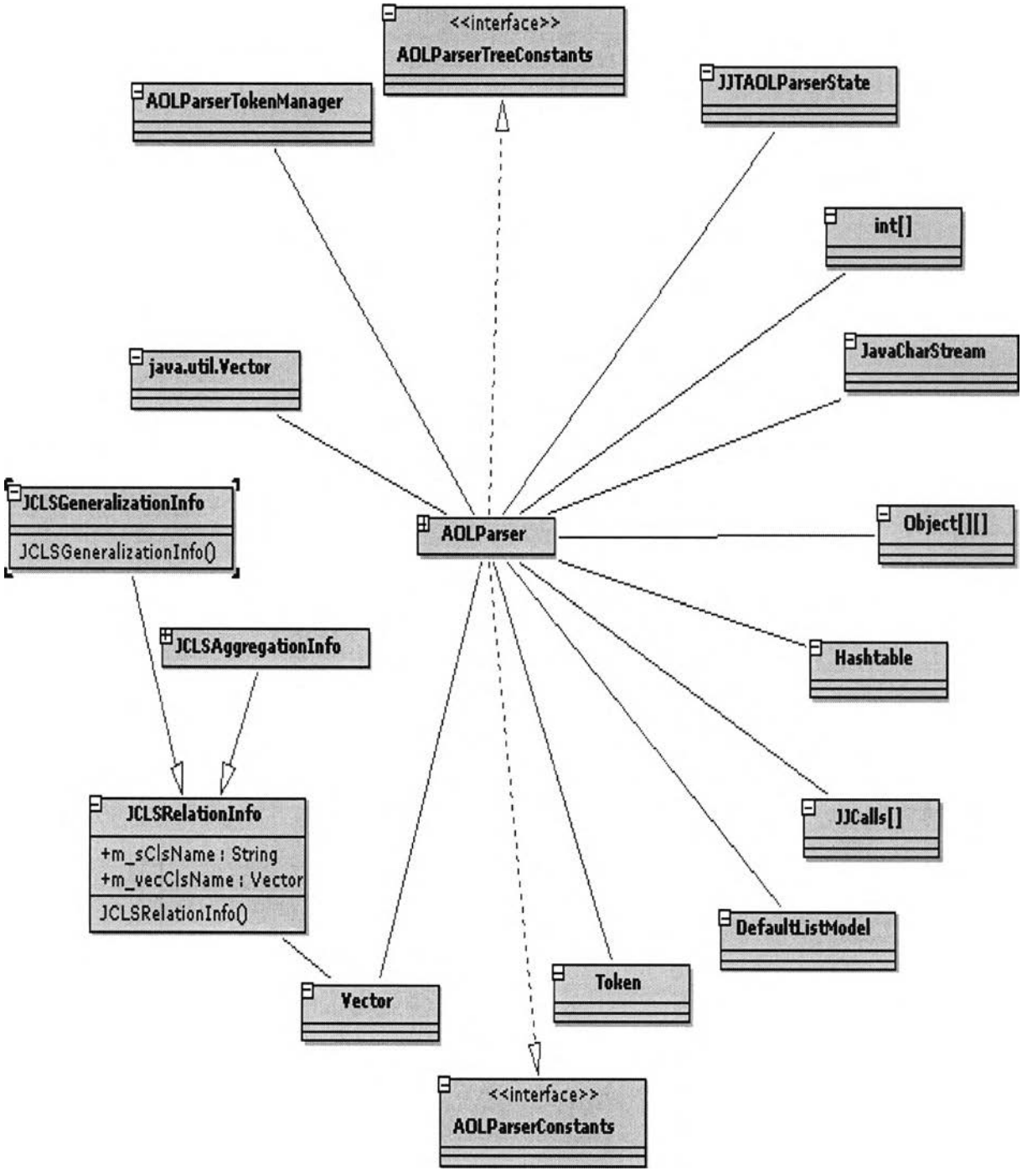
```
/* RESERVED WORDS AND LITERALS */

TOKEN :
{
  < ABSTRACT  : "ABSTRACT" >
| < EXTERNAL  : "EXTERNAL" >
| < BOOLEAN             : "boolean" >
| < BYTE                : "byte" >
| < CHAR                : "char" >
| < CLASS               : "CLASS" >
| < CONST               : "const" >
| < CONST_CHAR  : "CONST_char">
| < _DEFAULT   : "default" >
| < DOUBLE              : "double" >
| < FLOAT               : "float" >
| < INT                 : "int" >
| < LONG                : "long" >
| < UNSIGNED  : "unsigned" >
| < PRIVATE             : "PRIVATE" >
| < PROTECTED           : "PROTECTED" >
| < PUBLIC              : "PUBLIC" >
| < UNDEF_SCOPE        : "UNDEF_SCOPE" >
| < SHORT               : "short" >
| < VOID                : "void" >
| < ATTRIBUTES          : "ATTRIBUTES" >
| < OPERATIONS          : "OPERATIONS" >
| < SHARED              : "SHARED" >
| < STRING              : "String" >
| < RELATION   : "RELATION" >
| < ROLES               : "ROLES" >
| < NAME                : "NAME" >
| < SCOPE               : "::" >
| < MULT                : "MULT" >
| < QUALIFIER  : "QUALIFIER" >
| < AGGREGATION         : "AGGREGATION" >
| < CONTAINER           : "CONTAINER" >
| < PARTS               : "PARTS" >
| < GENERALIZATION  : "GENERALIZATION" >
| < SUBCLASSES          : "SUBCLASSES" >
| < DISCRIMINATOR      : "DISCRIMINATOR" >
}
```

รูปที่ 4.14 โค้ดแสดงโทเคนที่ประกาศใช้สำหรับไวยากรณ์ของภาษาเอโอแอล

4.5.2.5 คลาสและฟังก์ชันประกอบการคอมไพล์ภาษาเอโอแอล และการคำนวณหาจำนวนของฟังก์ชันพอยต์เชิงวัตถุ

ภายในไฟล์ AOL.jjt ประกอบด้วยคลาสและฟังก์ชันสำหรับการประมวลผลโทเคนและไวยากรณ์ภาษาเอโอแอล จำนวนมาก อีกทั้งภายในคลาสและฟังก์ชันเหล่านี้ยังมีการเก็บข้อมูลของคลาสรวมถึงการวิเคราะห์หาจำนวนของฟังก์ชันพอยต์เชิงวัตถุของคลาสต่างๆในไฟล์ เอโอแอลไปพร้อมๆกัน แผนภาพคลาส (เฉพาะคลาสหลัก) ของโมดูลพาร์เซอร์และโมดูลการคำนวณฟังก์ชันพอยต์เชิงวัตถุบนภาษาเอโอแอล แสดงดังรูปที่ 4.15



รูปที่ 4.15 แผนภาพคลาสของโมดูลพาร์เซอร์และโมดูลการคำนวณฟังก์ชันพอยต์เชิงวัตถุบนภาษาเอโอแอล