การประเมินค่าการใช้ Heap และ Stack ของ JAVA Object

JAVA OBJECT'S HEAP AND STACK RESOURCES EVALUATION

ชริยา นนทกาญจน์

CHARIYA NONTHAKARN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2547

ISBN 974-15-1224-4

การประเมินค่าการใช้ Heap และ Stack ของ JAVA Object

JAVA OBJECT'S HEAP AND STACK RESOURCES EVALUATION

ชริยา นนทกาญจน์

CHARIYA NONTHAKARN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2547

ISBN 974-15-1224-4

JAVA OBJECT'S HEAP AND STACK RESOURCES EVALUATION

CHARIYA NONTHAKARN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE IN INFORMATION TECNOLOGY

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2004

ISBN 974-15-1224-4

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การประเมินค่าการใช้ HEAP และ STACK ของ JAVA OBJECT

JAVA OBJECT'S HEAP AND STACK RESOURCES EVALUATION

ชื่อนักศึกษา นางชริยา นนทกาญจน์

รหัสประจำตัว 43067010

หลักสูตร วิทยาศาสตรมหาบัณฑิต

สาขาวิชา เทคโนโลยีสารสนเทศ

อาจารย์ผู้ควบคุมวิทยานิพนธ์ ผศ.อัครินทร์ คุณกิตติ

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ผศ.อัครินทร์	คุณกิตติ	
รศ.ดร.รัตติกร	วรากูลศิริพันธุ์	
ผศ.ดร.วรพจน์	กรีสุระเดช	
ดร.ธนารัตน์	ชลิดาพงศ์	
ดร.พรฤดี	เนติโสภากุล	

วัน/เดือน/ปี ที่สอบ 7 ตุลาคม 2547 เวลา 17.00 น. เป็นต้นไป

สถานที่สอบ ณ ห้อง M 03 (ชั้นลอย) อาคารเรียนรวมและปฏิบัติการคณะเทคโนโลยีสารสนเทศ

บัณฑิตวิทยาลัยรับรองแล้ว

(ผศ.ดร.จารุวัตร เจริญสุข)

คณบดีบัณฑิตวิทยาลัย

วันที่.....28.....**เดือน**.....ตุลาคม.....**พ.ศ**.....2547.....

Thesis Title	JAVA Object's Heap and Stack Resources Evaluation
Student	Mrs. Chariya Nonthakarn
Student ID.	43067010
Degree	Master of Science
Programme	Information Technology
Year	2004
Thesis Advisor	Asst.Prof.Akharin Khunkitti

ABSTRACT

Resource Management is a principle role of computer system for the best quality service and corresponding to user's requirement. Concept of evaluating amount of resources can be used for resource management because the system can use information from this in decision support and can help system's resource management for the most efficiency. There were related works that focused on evaluation at runtime that cause many problems. Therefore, this research proposes an approach for solving theses problems by evaluating resources before runtime and the evaluation focus on heap and stack (local variable and operand stack) of Java object. The approach proposes Java's Object Resource Evaluation Framework (JREF) which can be devided into 2 phases; compilation and evaluation. The compilation phase uses bytecode instructions from class file for instructions analysis which use heap and stack resources. Then create heap and stack expressions for all methods .All information from compilation phase will be kept in RDF file. The evaluation phase evaluates all expressions from RDF and get actual amount of resources. The result will show the maximum values in methods .The experimental results of both heap and stack are the same as results from runtime evaluation which excluding system memory . Therefore, the system can use this result for basic information of resource management and use this concept for further work.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้เป็นอย่างดี ด้วยคำแนะนำและให้คำปรึกษาจาก ผศ.อัครินทร์ คุณกิตติ ซึ่งเป็นอาจารย์ผู้ควบคุมวิทยานิพนธ์ ที่ได้ให้ความช่วยเหลือและแก้ปัญหาตลอดจนให้ความรู้และประสบการณ์ที่ดีแก่ข้าพเจ้า ขอกราบขอบพระคุณเป็นอย่างสูง

ขอบคุณเจ้าหน้าที่คณะเทคโนโลยีสารสนเทศทุกท่าน ที่ได้ช่วยดำเนินงานในการจัดการสอบและให้ข้อมูลที่เป็นประโยชน์ในการเตรียมตัวในการเสนอวิทยานิพนธ์

สุดท้าย ขอขอบคุณเพื่อน ๆ ทุกคนที่ให้กำลังใจและให้คำแนะนำที่เป็นประโยชน์ต่อการทำวิทยานิพนธ์

สำหรับคุณงามความดีอันใดที่เกิดจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบให้กับบิดามารดาตลอดจนครูอาจารย์ที่เคารพทุกท่านที่ได้ประสิทธิประสาทวิชาความรู้และถ่ายทอดประสบการณ์ที่ดีแก่ข้าพเจ้า

ชริยา นนทกาญจน์

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป(ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจุบันเทคโนโลยีต่าง ๆ ของระบบคอมพิวเตอร์และระบบเครือข่ายมีการเปลี่ยนแปลงและพัฒนาไปสู่รูปแบบใหม่ ๆ อย่างรวดเร็ว ทำให้การบริการแก่ผู้ใช้ระบบจะต้องมีการจัดการที่เหมาะสมกับเทคโนโลยีนั้น ๆ โดยเฉพาะอย่างยิ่งการจัดการทรัพยากรซึ่งถือได้ว่าเป็นหน้าที่ที่สำคัญที่ระบบจะต้องมีรูปแบบหรือวิธีการในการจัดการเพื่อสนองตอบต่อความต้องการของผู้ใช้ระบบให้มีประสิทธิภาพมากที่สุด จากความสำคัญของการจัดการทรัพยากรทำให้ในปัจจุบันนี้ได้มีการวิจัยและนำเสนอเทคนิคและวิธีการในการจัดการทรัพยากรของระบบในหลาย ๆ รูปแบบที่เหมาะสมกับระบบที่ใช้สถาปัตยกรรมหรือเทคโนโลยีที่ต่างกัน

วิธีการต่าง ๆ ที่ได้มีการนำเสนอได้พยายามที่จะแก้ปัญหาที่เกิดขึ้นกับการจัดการทรัพยากรและเนื่องจากทรัพยากรมีอยู่อย่างจำกัดแต่ความต้องการในการใช้งานระบบมีเพิ่มมากขึ้นและมีการเปลี่ยนแปลงรูปแบบและวิธีการในการใช้งานออกไป แต่แนวความคิดพื้นฐานที่สำคัญที่จะช่วยให้การจัดการระบบมีประสิทธิภาพมากขึ้น คือ ไม่ว่าระบบจะอยู่ในรูปแบบใดหรือใช้สถาปัตยกรรมแบบไหนก็ตาม ถ้าหากระบบทราบปริมาณของทรัพยากรที่จะใช้ในการทำงานแล้ว ก็จะช่วยให้การจัดการทรัพยากรมีความเหมาะสมและสอดคล้องกับความต้องการของผู้ใช้ซึ่งหมายถึงประสิทธิภาพของการทำงานโดยรวมของระบบ

จากความสำคัญดังกล่าว จึงได้มีงานวิจัยที่ได้ทำการศึกษาการวัดการใช้ทรัพยากร โดยเป็นการวัดการใช้ทรัพยากรในขณะ Runtime ซึ่งจะทำการวัดและ Run ไปพร้อม ๆ กัน[4][9][12] ซึ่งการวัดในลักษณะนี้ยังทำให้เกิดปัญหา คือ

1. ระบบจะต้องทำการวัดทรัพยากรทุกครั้งที่มีการทำงาน

2. ในขณะที่มีการวัดการใช้ทรัพยากรจะมีการใช้ทรัพยากรใน 2 ลักษณะ คือ ใช้ทรัพยากรเพื่อการทำงานจริงและใช้ทรัพยากรเพื่อการวัด ซึ่งอาจทำให้เกิดการใช้ทรัพยากรเกินไปกว่าที่จำกัดไว้ได้

3. ในขณะที่ Run ถ้าพบว่าระบบกำลังจะมีการใช้ทรัพยากรเกินไปกว่าที่กำหนดไว้ ระบบอาจจะหยุดการทำงานซึ่งทำให้เกิดการสูญเปล่าของการใช้ทรัพยากรจากงานที่ได้ทำมาก่อนหน้า

4. นอกจากนี้ในการเปรียบเทียบการทำงานของ Object 2 ตัว หากต้องการเปรียบเทียบว่า Object ตัวไหนที่มีความเหมาะสมในการทำงานมากกว่า(ในแง่ของการใช้ทรัพยากร) ก็จะต้องทำการ run Object ทั้ง 2 ก่อนจึงจะสามารถทราบได้ว่าควรจะเลือกใช้ Object ตัวไหนมาทำงาน

จากปัญหาดังกล่าวข้างต้น งานวิจัยนี้จึงนำเสนอแนวคิดในการวัดการใช้ทรัพยากรก่อนที่จะทำงาน (หลังจาก Compile) เพื่อที่ระบบจะได้นำข้อมูลจากการวัดไปใช้ในการตัดสินใจในการจัดการทรัพยากร โดยเป็นการวัดการใช้ทรัพยากรของ Object ที่อยู่ในรูปของโปรแกรมที่เขียนด้วยภาษาเชิงวัตถุ (Object Oriented Programming) คือ ภาษา Java ซึ่งสามารถรองรับการทำงานในลักษณะนี้ได้อย่างดี และเป็นภาษาที่มีความนิยมอย่างแพร่หลายในปัจจุบัน ซึ่งการวิจัยนี้จะเป็นการวัดการใช้ทรัพยากรในการทำงานของ Object โดยใช้ข้อมูลจากการ Compile และเป็นการวัดขณะทำงานบน Java Virtual Machine (JVM)

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

1. เพื่อศึกษาลักษณะของการใช้ทรัพยากรประเภทหน่วยความจำของ Java Object
2. เพื่อศึกษาลักษณะของ Java Class File เพื่อนำข้อมูลที่ได้ไปใช้สำหรับการวิเคราะห์การใช้ทรัพยากร
3. เพื่อศึกษาวิธีการหาหรือวัดปริมาณการใช้หน่วยความจำประเภท Heap และ Stack ของ Java Object
4. เพื่อนำเสนอวิธีการแก้ปัญหาที่เกิดจากการวัดการใช้ทรัพยากรในขณะ Runtime

1.3 ขอบเขตของการศึกษา

1. งานวิจัยนี้ศึกษาถึงวิธีการในการประเมินค่าทรัพยากรของ Object คือ Heap และ Stack(Local Variable และ Operand Stack) Resource ซึ่งจะวัดจากพื้นที่ที่ใช้ในการทำงานของ Java Virtual Machine(JVM)
2. Class/Object ที่นำมาใช้ในการวัดเขียนจากภาษา Java โดย Format ของ Class จะอยู่ในรูปแบบของ Java Virtual Machine Specification[3] [8]
3. การวัดการใช้ทรัพยากรเป็นการวัดการทำงานของ Object บน Java Virtual Machine โดยจะใช้ข้อมูลจาก Class File
4. การวัดในกรณีพื้นที่ในการทำงานจะวัดในขณะที่มีการประมวลผล คือ ใช้เฉพาะ Primary Storage เท่านั้น
5. การทำงานของ Object จะเป็นการทำงานในกรณีที่มีเพียง Thread เดียวเท่านั้น
7. ใช้ Compiler คือ JDK 1.3.0_02
8. Object จะต้องเป็น Object ที่มีความถูกต้องตามไวยากรณ์ของภาษา Java

1.4 เนื้อหาของงานวิจัย

รายละเอียดของการวิจัยและเนื้อหาแบ่งออกได้ดังนี้

บทที่ 1 บทนำ กล่าวถึงความเป็นมาและความสำคัญของปัญหา วัตถุประสงค์ของการวิจัย รวมถึงขอบเขตของการวิจัย

บทที่ 2 โปรแกรมภาษา Java โดยจะเป็นการกล่าวถึงภาพรวมของภาษา Java กลไกการทำงานของ JVM ที่เกี่ยวกับการใช้ทรัพยากร และโครงสร้างของ Java Class File

บทที่ 3 การประเมินค่า Heap และ Stack ของ Java Object ซึ่งจะกล่าวถึง แนวความคิดของการประเมินค่าการใช้ทรัพยากร ขอบเขตและรายละเอียดของการวัดการใช้ทรัพยากร และAlgorithm ที่ใช้ในการวัด

บทที่ 4 การทดลองและผลการทดลอง แสดงผลการทดลองการวัดการใช้ทรัพยากรของทรัพยากรทั้ง 2 ประเภท คือ Heap และ Stack

บทที่ 5 บทสรุปและข้อเสนอแนะ กล่าวถึงผลสรุปของการวิจัย ข้อเสนอแนะและแนวทางการวิจัยต่อไป

1.5 แผนการดำเนินงานวิจัย

ในการวิจัยนี้ได้นำเสนอวิธีการวัดการใช้ทรัพยากรของ Java Object โดยมีขั้นตอนดังต่อไปนี้

1. ศึกษาหลักการและทฤษฎีของการทำงานในเชิงวัตถุ และภาษา Java
2. กำหนดประเภทของทรัพยากรที่จะทำการวัด
3. ศึกษาเครื่องมือวัดการใช้ทรัพยากรขณะ Runtime และกำหนดวิธีการในการตรวจสอบผลจากการวัด
4. กำหนดวิธีการวัดที่เหมาะสมโดยใช้หลักการและทฤษฎีที่เกี่ยวข้อง
5. ออกแบบ Algorithm ที่จะใช้ในการวัด
6. สร้างเครื่องมือที่ใช้ในการวัด
7. ทำการทดลอง
8. ทดสอบและแก้ไข
9. ประเมินผล , หาข้อสรุปและข้อเสนอแนะ
10. จัดทำเอกสาร

1.6 ประโยชน์ที่จะได้รับ

1. ทราบถึงรูปแบบการใช้ทรัพยากรของ Java Object
2. ทราบถึงวิธีการในการวัดการใช้ทรัพยากรแต่ละประเภท
3. ผลที่ได้จากการวัดไปใช้ในการจัดการทรัพยากรได้อย่างมีประสิทธิภาพ
4. เพื่อนำข้อมูลที่ได้มาใช้ในการตัดสินใจการใช้ Object ว่าระบบสามารถที่จะรองรับการทำงานของ Object นั้นได้หรือไม่
5. สามารถลดภาระงานจากการวัดการใช้ทรัพยากรโดยไม่ต้องมีการวัดทุกครั้งที่มีการ Run
6. สามารถลดการใช้ทรัพยากรเนื่องจากการวัดในขณะ Runtime
7. ผลลัพธ์ที่ได้เป็นแนวทางเบื้องต้นที่สามารถนำไปประยุกต์ใช้สำหรับการวัดการใช้ทรัพยากรในระดับต่อไป

บทที่ 2
โปรแกรมภาษา JAVA

2.1 แนวคิดในเชิง Object-Oriented

Object อาจจะหมายถึง ส่วนของการพัฒนา Software หรือฐานข้อมูลแบบ Object หรือในส่วนงานอื่น ๆ โดยมีการนำเอาแนวคิดทางด้าน Object ไปใช้ในงานวิจัยทางด้านต่าง ๆ ตัวอย่างงานที่เกี่ยวข้องกับเทคโนโลยีทางด้าน Object-Oriented เช่น Mobile Agent , Active Network เป็นต้น ดังนั้นจึงนับได้ว่าเป็นเทคโนโลยีที่เข้ามามีบทบาทในการทำงานของระบบคอมพิวเตอร์เพิ่มมากขึ้นในปัจจุบัน เนื่องจากข้อดีหลายประการ คือ

1. Object ใน Computer เปรียบได้กับสิ่งของในโลกของความเป็นจริง ทำให้ออกแบบเพื่อนำมาใช้กับงานจริงได้ง่ายและทำให้ผู้ใช้สามารถใช้งานได้ง่ายกว่า

2. ลักษณะการทำงานของ Object จะช่วยลดความซับซ้อนในการพัฒนาระบบ และช่วยลดค่าใช้จ่ายในการสร้างและบำรุงรักษาระบบ เนื่องจากเมื่อ Object ถูกสร้างหรือ Implement ขึ้นมาครั้งหนึ่งจะสามารถนำไปใช้ในระบบอื่นได้

3. ระบบที่ใช้ Object Technology จะแก้ไขหรือเปลี่ยนแปลงตัวเองได้ง่าย โดยทำการแก้ไขส่วนประกอบภายใน Object ซึ่งจะไม่มีผลกระทบต่อส่วนอื่นของระบบ

โดยทั่วไปแล้ว Object จะหมายถึงสิ่งต่าง ๆ เช่น สิ่งของ เหตุการณ์ สถานที่ หรืออาจจะเป็นตัวโปรแกรม แต่ไม่ว่า Object จะอยู่ในลักษณะใดก็ตาม ก็จะต้องประกอบด้วย 2 ส่วนที่สำคัญ คือ

1. State : เป็นข้อมูลเฉพาะของ Object
2. Behavior : เป็นการกระทำหรือพฤติกรรมต่าง ๆ ของ Object

จากเหตุผลและลักษณะดังกล่าวในข้างต้น งานวิจัยนี้จึงนำเสนอแนวคิดในการวัดการใช้ทรัพยากรของ Object โดยมอง Object ในลักษณะของโปรแกรม เพื่อนำข้อมูลที่ได้ไปใช้ในการตัดสินใจในการจัดการทรัพยากรต่อไป

2.2 ภาษา Java

ภาษา Java เป็นภาษาที่ถูกพัฒนาขึ้นมาสำหรับการทำงานในเชิงของวัตถุ และได้รับความนิยมเป็นอย่างมากในปัจจุบันเนื่องจากเป็นภาษาที่มีคุณสมบัติต่าง ๆ หลายประการ คือ

1. เป็นภาษาที่ง่าย(Simple) ในการเรียนรู้และใช้งาน

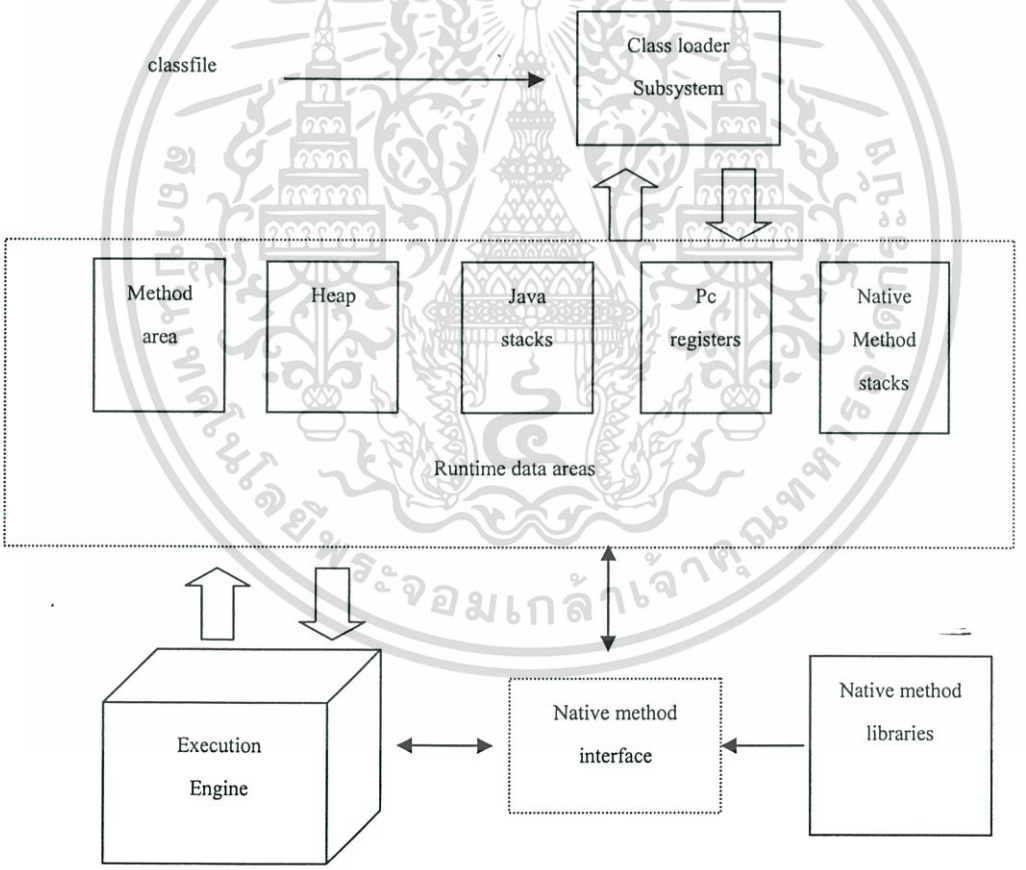
2. โปรแกรมที่สร้างขึ้นด้วยภาษา Java จะไม่มีความผิดพลาดจากข้อบกพร่องของภาษา คือ โปรแกรมจะต้องไม่ล้มเหลวลง ด้วยความผิดที่ไม่เกี่ยวข้องกับตรรกะของโปรแกรม ซึ่งจะเรียกว่าคุณสมบัติคงทน(Robust)

3. โปรแกรมภาษา Java มักจะถูกส่งผ่านระบบเครือข่าย ดังนั้นภาษา Java จึงต้องมีหลักประกันว่าโปรแกรมจะไม่ทำอันตรายต่อระบบของผู้ที่รับโปรแกรมนั้นไปใช้งาน ซึ่งจะเรียกว่าคุณสมบัติปลอดภัย (Secure)

4. ภาษา Java สามารถทำงานบนเครื่องต่างระบบกันได้ เรียกคุณสมบัตินี้ว่าไม่ขึ้นกับระบบ ดังนั้น Java จึงต้องมีการแปลภาษาแบบทั้ง Compilation และ Interpretation ซึ่งจากคุณสมบัติในข้อนี้ โปรแกรมภาษา Java ที่ถูกเขียนขึ้นมาเพียงครั้งเดียวจะสามารถนำไปใช้ได้กับหลาย ๆ ระบบทำให้ภาษา Java จะต้องมีการแปล 2 ขั้นตอน โดยในขั้นตอนของ Compilation จะได้ผลลัพธ์เป็น Java Class File ซึ่งสามารถไป Run บน Virtual Machine ของระบบใด ๆ

2.3 การทำงานของ Java Virtual Machine(JVM)

จากคุณสมบัติในข้อที่ 4 Java Class File ที่ได้จากการ Compile จะถูกนำไปประมวลผลบน Java Virtual Machine ของระบบปฏิบัติการใด ๆ ซึ่งจะมีการใช้พื้นที่หน่วยความจำ ดังรูป 2.1



รูปที่ 2.1 การใช้พื้นที่หน่วยความจำของ Java Virtual Machine

สำหรับการทำงานของ Java Object จะใช้หน่วยความจำซึ่งเป็นส่วนของ Runtime Data Area ซึ่งประกอบด้วยพื้นที่ต่าง ๆ ที่ทำงานร่วมกัน คือ

1. Methodarea เป็นพื้นที่ในการเก็บข้อมูลต่าง ๆ ของ Type(Class) เพื่อนำไปใช้ในการทำงาน โดยพื้นที่ในส่วนนี้จะเป็นส่วนที่เก็บรายละเอียดเพียงชุดเดียวของ Class ไม่ว่า Class จะมีการสร้าง Object(Instance) กี่ครั้งก็ตาม ซึ่งจะประกอบด้วย

1.1 Fully Qualified Name ของ Class

1.2 Fully Qualified Name ของ Class ที่เป็น Superclass โดยตรงของ Class

1.3 Modifier ของ Class

1.4 Ordered List ของ Fully Qualified Name ของ Superinterface โดยตรง

1.5 Constant Pool ของ Class

1.6 รายละเอียดของ Field ซึ่งแต่ละ Field จะประกอบด้วย

- ชื่อของ Field
- Type ของ Field
- Modifier ของ Field

1.7 รายละเอียดของ Method ซึ่งแต่ละ Method ประกอบด้วย

- ชื่อของ Method
- Return Type ของ Method
- จำนวนและ Type ของ Method Parameter
- Modifier ของ Method
- Bytecode ของ Method
- ขนาดของ Operand Stack และ Local Variable
- Exception Table

1.8 Class Variable

1.9 Reference ไปยัง Class ClassLoader

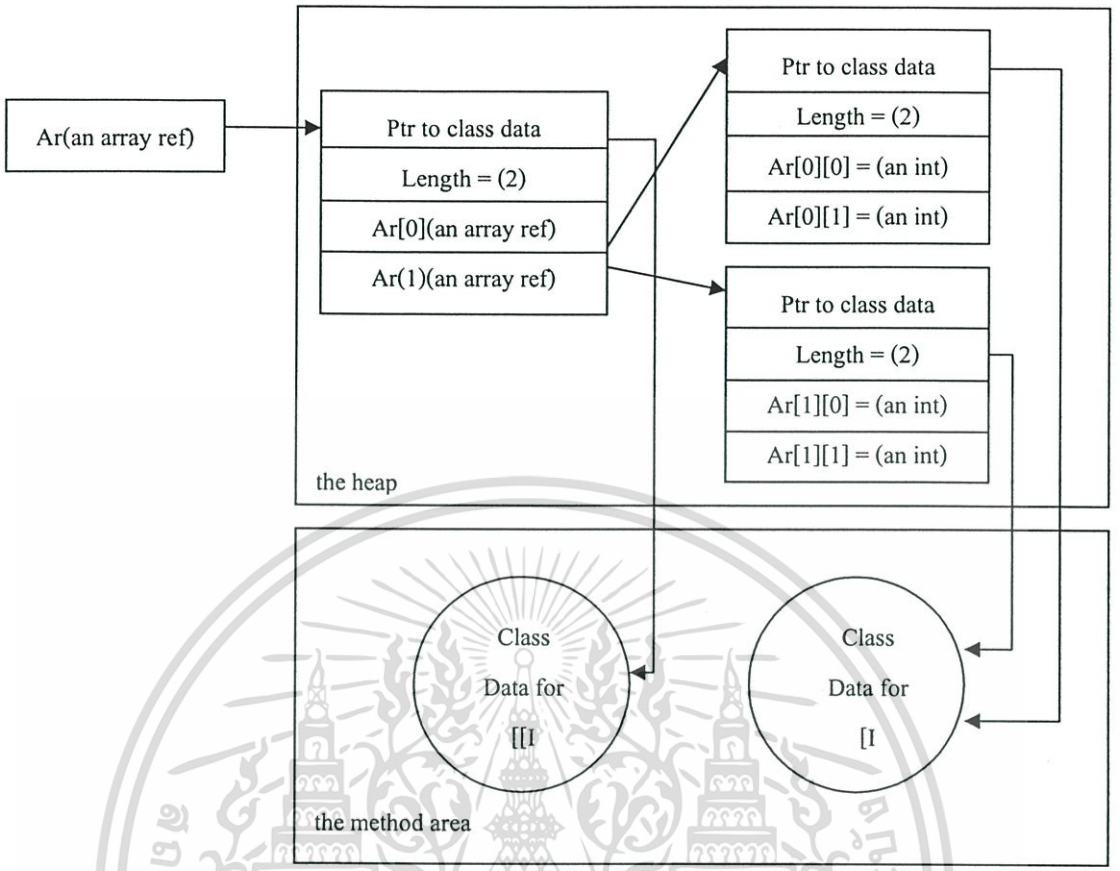
1.10 Reference ไปยัง Class Class

2. Heap จะเป็นพื้นที่สำหรับเก็บข้อมูลต่าง ๆ ของ

2.1 Object

2.2 Array

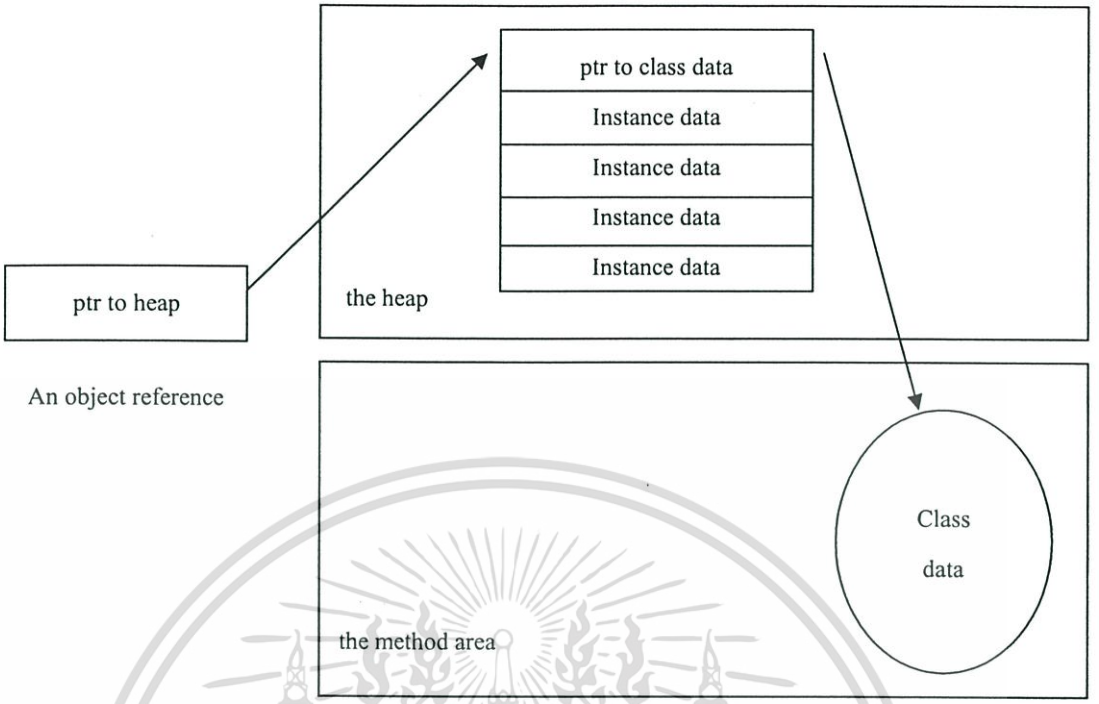
โดยพื้นที่ในส่วนนี้จะมีการใช้เมื่อมีการสร้าง Object หรือ Array ทุกครั้ง ซึ่งจะมีลักษณะของการใช้พื้นที่ ดังรูปที่ 2.2 และ 2.3



รูปที่ 2.2 การใช้พื้นที่หน่วยความจำของ Object

จากรูปจะเห็นว่าในการสร้าง Object จะมีการใช้พื้นที่ใน 2 ส่วน คือ ส่วนของ Methodarea และส่วนของ Heaparea โดยในส่วนของ Methodarea นั้นจะเป็นการใช้ร่วมกันของ Object ทุก Object ที่สร้างขึ้นมาจาก Class เดียวกัน ดังนั้นเมื่อมีการสร้าง Object ในแต่ละครั้งจึงมีการใช้พื้นที่ของ Heap สำหรับเก็บ Instance Data หรือเป็นพื้นที่ของ Instance Variable และพื้นที่ของ Reference หรือ Pointer นั่นเอง

สำหรับ Array ก็เช่นเดียวกันเมื่อมีการสร้าง Array ก็จะมีการใช้พื้นที่ของ Heap โดยจากรูป 2.3 เป็นการใช้พื้นที่ของ Array ของ Integer ที่เป็น 2 มิติ (int[][] ar = new int[2][2]) โดยการใช้พื้นที่ของ Array จะเป็นพื้นที่ในการเก็บค่าของ Array นั้น(ซึ่งในที่นี้ คือ Integer) ไว้ในส่วนของมิติที่อยู่ลึกสุด และเก็บ Reference ไปยังมิติที่อยู่ในระดับบนขึ้นมา รวมทั้ง Reference ไปยัง Class File ใน Methodarea เช่นเดียวกัน



รูปที่ 2.3 การใช้พื้นที่หน่วยความจำของ Array

3. Program Counter เป็นส่วนที่เก็บสถานะในการทำงานในขณะใด ๆ ของ Method คือจะเป็นตัวบอกว่าคำสั่งต่อไปที่จะประมวลผลคือคำสั่งไหน ซึ่ง Program Counter นี้จะมี 1 ตัวต่อ 1 Thread

4. Java Stack เป็นพื้นที่ในการทำงานของ Method ที่ประกอบด้วย Stack Frame หลายตัวซึ่งจะถูกสร้างและลบออกทุกครั้งที่มีการ Invoke Method และ Method นั้นทำงานเสร็จสิ้น โดย Stack Frame แต่ละตัวจะประกอบด้วย

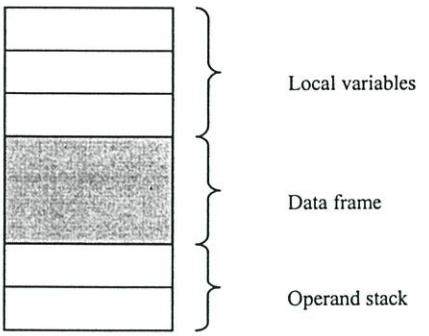
4.1 Local Variable เป็นพื้นที่ที่มีลักษณะเหมือนกับ Array ซึ่งจะเป็นส่วนที่เก็บ Local Variable และ Argument ของ Method

4.2 Operand Stack เป็นส่วนที่ใช้ในการประมวลผลของ Method ซึ่งอาจจะมีการเพิ่มและลดขนาดตลอดเวลาที่มีการประมวลผล

4.3 Data Frame ซึ่งจะประกอบด้วยข้อมูลต่าง ๆ คือ

- ข้อมูลเพื่อ Link ไปยัง Constant Pool
- ข้อมูลเกี่ยวกับการ Invoke Method
- ข้อมูลเกี่ยวกับ Exception

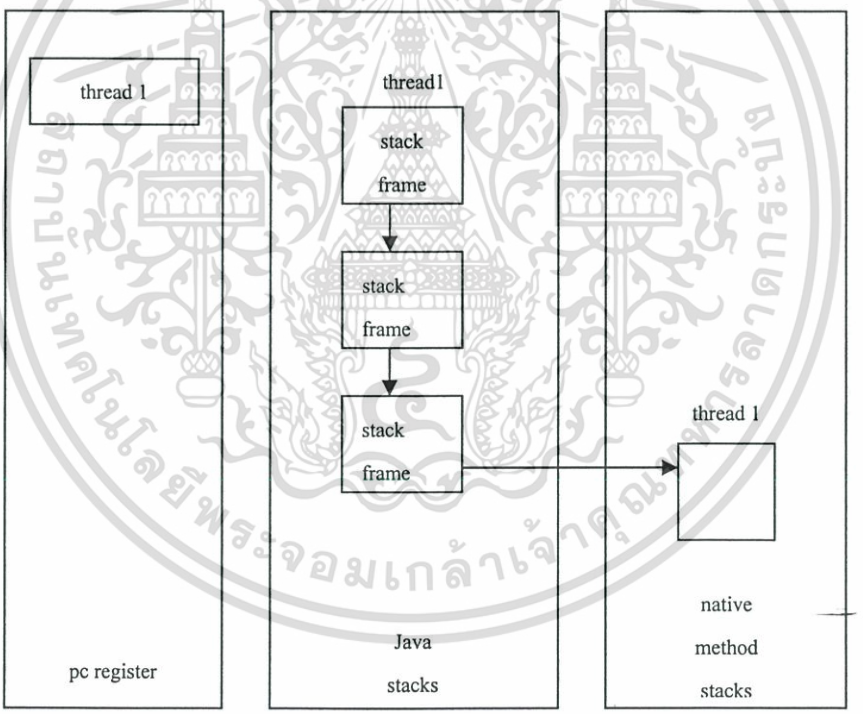
Java Stack Frame มีการใช้พื้นที่ ดังรูป 2.4



รูปที่ 2.4 โครงสร้างของ Java Stack Frame

ซึ่งเมื่อมีการ Invoke Method ในแต่ละครั้งก็จะมีการ Allocate พื้นที่ของ Java Stack Frame ทุกครั้ง

5. Native Method Stack เป็นพื้นที่ในการประมวลผลของ Native Method ซึ่งจะมีลักษณะเช่นเดียวกับ Java Stack ซึ่งจะมีการใช้พื้นที่ของ Stack ในภาพรวมดังรูปที่ 2.5



รูปที่ 2.5 การใช้พื้นที่ของ Stack

2.4 Java Class File

Java Class File เป็น File ที่ได้จากการ Compile Java Source Code ซึ่งจะประกอบด้วยส่วนต่าง ๆ ดังนี้

1. Header ประกอบด้วย Magic Number , Major และ Minor Vertion ของ Compiler ที่ Compile Class File นั้น

2. Constant Pool เป็นส่วนที่เก็บรายละเอียดต่าง ๆ ของ Class File ไว้ ประกอบด้วย String Constant ต่าง ๆ , Class Name , Fields name และ Constant อื่น ๆ ที่ถูกอ้างถึงภายใน Class

3. Access Right จะเก็บรายละเอียดของ Modifier ของ Class เช่น public , private เป็นต้น

4. Implemented Interface จะเก็บรายละเอียดของ Interface ทั้งหมดที่เกี่ยวข้องกับ Class นั้น

5. Field จะเก็บรายละเอียดของ Field ทั้งหมดที่ถูกอ้างถึงภายใน Class

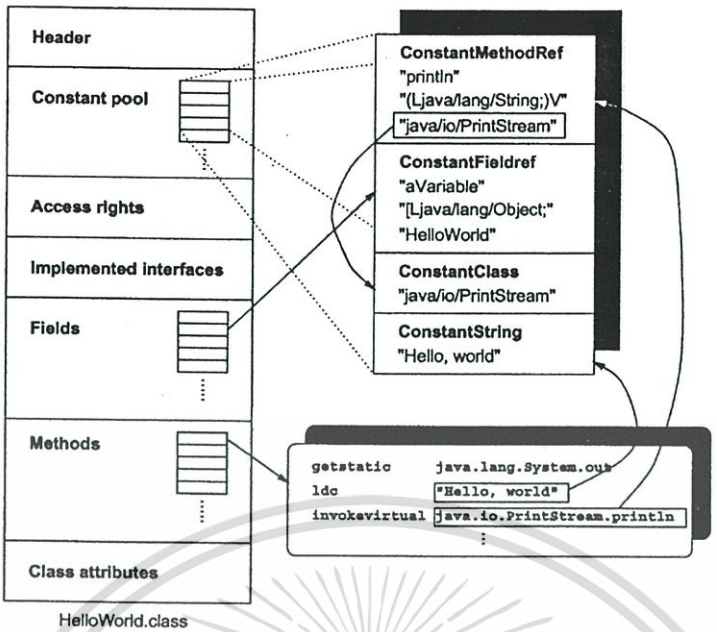
6. Method จะเก็บรายละเอียดของ Method ทั้งหมดที่ถูกอ้างถึงภายใน Class

7. Class Attribute จะเก็บรายละเอียดของ Attribute ทั้งหมดที่ถูกอ้างถึงภายใน Class

การเก็บรายละเอียดทั้งหมดใน Class File ในแต่ละส่วนจะมีรูปแบบที่ถูกกำหนดไว้แน่นอน เช่น รูปแบบการเก็บ Constant_Class_info จะมีรูปแบบเป็น

```
CONSTANT_CLASS_info {
    U1      tag ;
    U2      name_index
    }
```

ซึ่งการเก็บข้อมูลของ Class Constant ทุกตัวจะประกอบด้วย 2 ส่วน คือ Tag และ name_index ซึ่งจะมีขนาดเป็น 1 และ 2 Unsigned Bytes ตามลำดับ และ Class File จะมีโครงสร้างดังรูปที่ 2.6



รูปที่ 2.6 ส่วนประกอบของ Java class file[6]

จากรูปแบบและรายละเอียดต่าง ๆ ที่ปรากฏใน Class File จะเห็นได้ว่าเราสามารถนำข้อมูลเหล่านั้นมาใช้สำหรับการวัดค่าการใช้ทรัพยากรของ Object ได้ โดยข้อมูลส่วนที่สำคัญที่สุดที่จะนำมาใช้ในการวิเคราะห์ คือ Bytecode Instruction ซึ่งอยู่ในส่วนของรายละเอียดของทุก Method ของ Class File เนื่องจาก Bytecode Instruction จะเป็นส่วนที่สามารถบอกถึงพฤติกรรมในการใช้หน่วยความจำทั้ง Heap และ Stack ของ Method ของ Object ที่ถูกสร้างขึ้นมาจาก Class File การนำเอาข้อมูลจากส่วนต่าง ๆ มาใช้นั้นจะได้กล่าวในส่วนต่อไป

2.5 Java Bytecode Instruction

จากโครงสร้างของ Java Class File ข้างต้นส่วนประกอบที่มีความสำคัญที่สามารถอธิบายถึงพฤติกรรมของ Java Object ได้ คือ ส่วนของ Method ซึ่งทุก Method จะประกอบด้วย Java Bytecode Instruction ที่เป็นคำสั่งที่ใช้พื้นที่ในการเก็บ 1 Byte โดย Bytecode Instruction (Opcode) จะมีทั้งหมด 256 คำสั่ง ซึ่งคำสั่งแต่ละตัวจะมีผลต่อการใช้หน่วยความจำที่แตกต่างกัน โดยทุกคำสั่งที่จะถูกประมวลผลมีผลต่อขนาดของ Operand Stack ของ Java Stack ซึ่งจะมีการเปลี่ยนขนาดตลอดเวลาที่มีการประมวลผลในคำสั่งใหม่ (ขนาดของ Operand Stack ที่เกิดการเปลี่ยนแปลงในแต่ละคำสั่งดูรายละเอียดในภาคผนวก ก) และ Bytecode แบ่งออกเป็นกลุ่มได้ดังต่อไปนี้

1. Load and Store Instruction : เป็นคำสั่งที่ใช้สำหรับการใส่และดึงค่าจาก Stack และ Local Variable

2. Arithmetic Instruction : เป็นคำสั่งที่เกี่ยวกับการประมวลผลทางคณิตศาสตร์

3. Type Conversion Instruction : เป็นคำสั่งที่ใช้สำหรับเปลี่ยนชนิดของตัวแปร

4. Object Creation and Manipulation : เป็นคำสั่งที่ใช้สำหรับการสร้าง Object และ Array ซึ่งหากมีการใช้คำสั่งในกลุ่มนี้จะมีการใช้พื้นที่ของ Heap

5. Operand Stack Management Instruction : เป็นคำสั่งที่ใช้สำหรับจัดการ Stack โดยตรง

6. Control Transfer Instruction : เป็นคำสั่งที่เกี่ยวกับการลำดับการไหลในการทำงานของคำสั่ง

7. Method Invocation and Return Instruction : เป็นคำสั่งที่ใช้การ Invoke Method และ Return ค่าจาก Method ซึ่งผลจากการใช้คำสั่งนี้จะมีการสร้าง Java Stack Frame ขึ้นมาใหม่สำหรับใช้ในการทำงานของ Method ที่ถูก Invoke นั้น

8. Throwing and Handling Exception : เป็นคำสั่งที่ใช้จัดการ Exception หรือการดัก Error ของคำสั่ง

9. Implementing Finally : เป็นคำสั่งที่ใช้จัดการเกี่ยวกับการทำ Finally

10. Synchronization : เป็นคำสั่งในการจัดการ Thread

(รายละเอียดของแต่ละคำสั่งดูในภาคผนวก ก)

ตัวอย่างการแปล Java Source Code เป็น Java Class File ยกตัวอย่างเช่น มี Code

```
void whileInt(){
        int i = 0;
        while(i < 100){
                i++;
        }
}
```

ซึ่งเมื่อ Compile จะอยู่ในรูปแบบของ Bytecode Instruction ที่อยู่ใน Format

<index><opcode>[<operand1>][<operand2>]…

คือ

Method void whileInt()

```
0  iconst_0
1  istore_1
2  goto 8
5  iinc 1 1
8  iload_1
9  bipush 100
11 if_icmplt 5
14 return
```

จากรายละเอียดที่กล่าวมาข้างต้นผู้วิจัยได้นำหลักการมาใช้ในการวัดการใช้ทรัพยากรทั้ง 2 ประเภท ซึ่งรายละเอียดในการวัดจะกล่าวถึงในบทที่ 3

บทที่ 3

การประเมินค่าการใช้ Heap และ Stack ของ Java Objects

3.1 แนวความคิด (Concept) ของการประเมินค่าการใช้ทรัพยากร

จากความสำคัญของการจัดการทรัพยากรและปัญหาที่เกิดขึ้นงานวิจัยนี้จึงนำเสนอแนวคิดในการวัดการใช้ทรัพยากรก่อนการ Run ของ Java Object โดยใช้ข้อมูลจาก Class File ที่สามารถบ่งบอกถึงพฤติกรรมในการใช้ทรัพยากรของ Object ที่จะถูกสร้างจาก Class File นั้นได้ เนื่องจาก Java มีคุณสมบัติที่สำคัญ คือ Platform Independent นั่นคือ Java Program จะถูก Compile เป็น Class File แล้วสามารถนำ Class File นั้นไป Run บน Platform ใดๆ โดยไม่มีการเปลี่ยนแปลง ดังนั้นจึงสามารถใช้ข้อมูลจาก Class File ในการวิเคราะห์การใช้ทรัพยากรของ Java Object ซึ่งรูปแบบการแปลของ Java จะประกอบด้วยการทำงานในการแปล 2 ช่วง ดังรูปที่ 3.1

รูปที่ 3.1 การแปลของ Java

โดยการวิจัยนี้จะเป็นการวัดการใช้ทรัพยากรของ Java Object ใน 2 ส่วน คือ

1. Heap ซึ่งจะเป็นพื้นที่ในการเก็บข้อมูลของ Object(Instance ของ Class) และ Array
2. Stack เป็นพื้นที่ในการทำงานของ Method ของ Object ซึ่งสิ่งที่จะวัดจะประกอบด้วย
   2.1 Local Variable เป็นพื้นที่ในการเก็บ Local Variable และ Argument ของ Method
   2.2 Operand Stack เป็นพื้นที่ในการทำงานของ Method

โดย Class File มีลักษณะดังนี้

1. ใช้ Java Class File ของ Java Application สำหรับการประเมินค่าการใช้ทรัพยากรและมี Format ตาม [8]
2. Class File ทำงานบน Thread เพียง Thread เดียว
3. Class File ที่ทำการวัดนั้นไม่มีการนำค่าจากการ Invoke Method , ค่าจาก Array มาใช้ในการกำหนดความยาวของ Array และกำหนดจำนวนของการวนลูป
4. ลูปที่ปรากฏใน Class File จะต้องมีลักษณะคงที่(นับจำนวนได้แน่นอน) และ Class File นั้นไม่มี Error

5. Class File ไม่มีการเรียกใช้ API ที่มีการสร้าง Instance ในลักษณะ Dynamic เช่น การ Read File Stream เป็นต้น

6. การวัดจะไม่กล่าวถึง Garbage Collected เนื่องจากเป็นกลไกภายใน JVM ที่จะต้องทำการตัดสินใจซึ่งไม่สามารถคาดการณ์ได้ล่วงหน้า

3.2 Java Object's Resource Evaluation FrameWork(JREF)

งานวิจัยนี้เป็นการนำเสนอ Framework เพื่อใช้ในการประเมินค่าการใช้ทรัพยากรของ Java Object ซึ่งมีโครงสร้างดังรูปที่ 3.2 โดย Framework(ในส่วนที่ 1) จะประกอบด้วยการทำงานในขั้นตอนหลัก 2 ขั้นตอน คือ

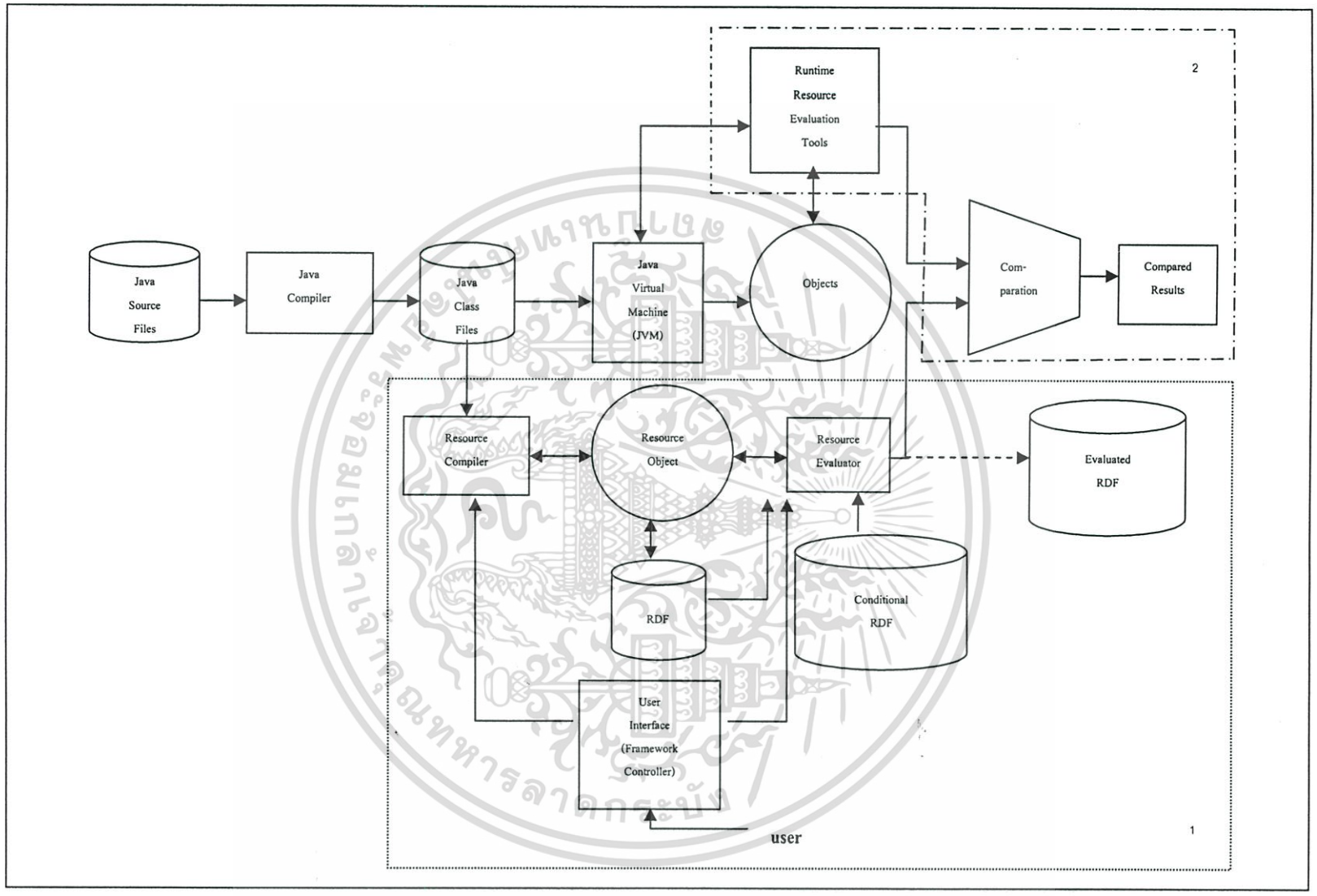
1. Resource Compiler

เป็นขั้นตอนของการนำเอาข้อมูลจาก Java Class File มาเพื่อวิเคราะห์และสร้าง Resource Object (RO) และ Resource Description File(RDF) ที่สอดคล้องกับโครงสร้างของ Resource Object ซึ่งจะมีโครงสร้างเหมือนกับการแบ่งประเภทของทรัพยากรที่กำลังจะวัด โดยในขั้นตอนของการ Compile นี้ Resource Compiler จะสร้าง Resource Object และสร้าง Resource Description File(RDF) ที่สอดคล้องกับ Resource Object โดยขั้นตอนของการ Compile นี้ทำให้ระบบไม่จำเป็นต้องมีการ Compile ทุกครั้งที่ Class File หรือ Object นั้นเข้าสู่ระบบ (ในกรณีที่ Class File นั้นไม่มีการเปลี่ยนแปลง) เพราะระบบสามารถนำข้อมูลจาก RDF นั้นมาเข้าสู่กระบวนการของการ Evaluation ได้เลย

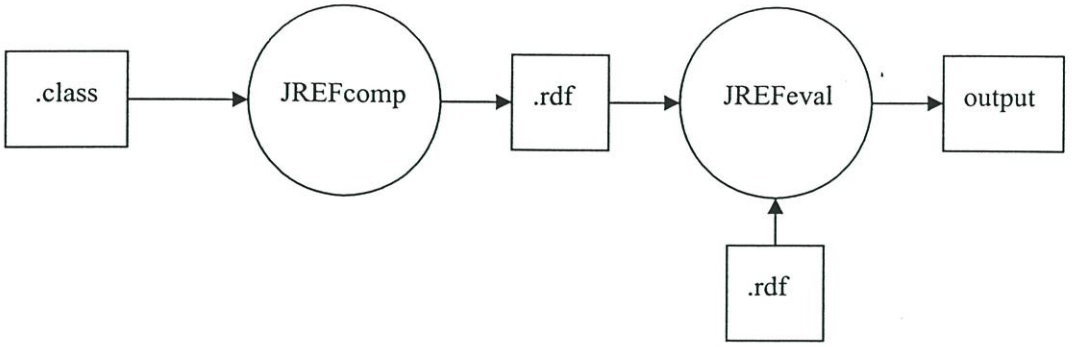
2. Resource Evaluator

Resource Evaluator จะนำ RDF มาเข้าสู่กระบวนการเพื่อหาค่าปริมาณของทรัพยากรในแต่ละประเภทที่จะใช้ และจะทำงานควบคู่ไปกับ Resource Object ซึ่งทำให้ในขณะที่มีการ Evaluate ตัว Class File นั้นไม่ต้องทำการ Run และในขั้นตอนนี้หาก Object นั้นมีการเรียกใช้ Object อื่น ก็จะต้องมีการเรียกใช้ RO และ RDF ของ Object ที่ถูกเรียกใช้นั้นมาเพื่อการประเมินเพื่อให้ได้ผลลัพธ์ด้วย และในกรณีที่ Class File นั้นจะต้องมีการป้อนค่าของตัวแปรเพื่อประมวลผลก็จะต้องมีการส่งค่าจาก User เข้ามาเพื่อการ Evaluate ด้วย

การทำงานทั้งใน 2 ขั้นตอนจะถูกควบคุมโดย User Interface ซึ่งจะทำหน้าที่เสมือนเป็น Framework Controller เพื่อควบคุมให้ Framework ทำงานตามที่กำหนด ซึ่งการทำงานใน 2 ขั้นตอนสรุปได้ดังรูปที่ 3.3



รูปที่ 3.2 JAVA Object's Resource Evaluation Framework (JREF)

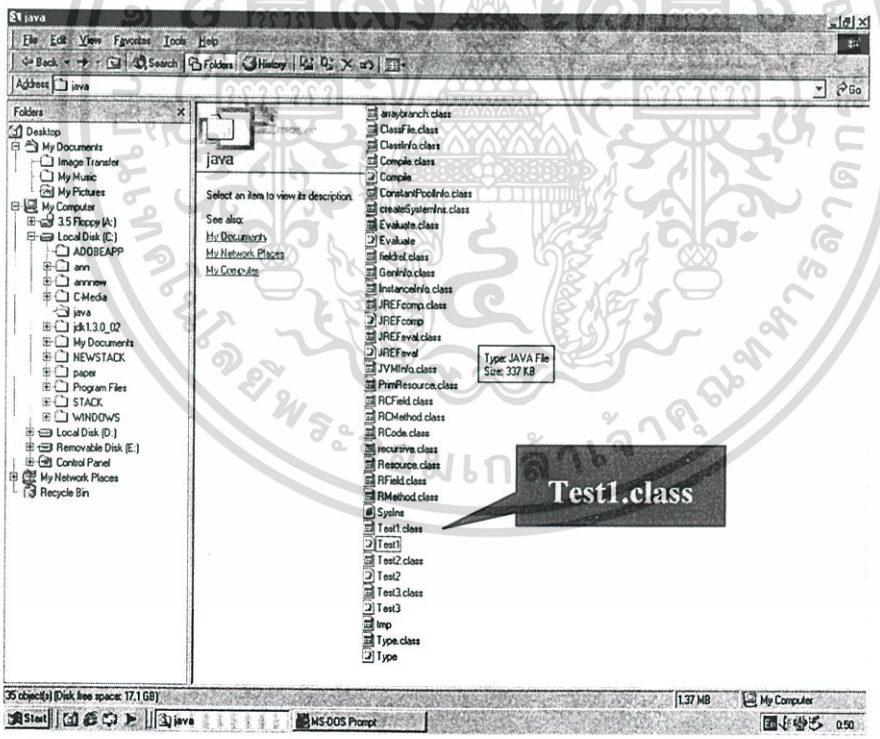


รูปที่ 3.3 การทำงานของ Framework

ซึ่งลักษณะของการ Compile แสดงได้ดังรูปที่ 3.4 , 3.5 และ 3.6 และการ Evaluate แสดงได้ดังรูปที่ 3.7 และ 3.8

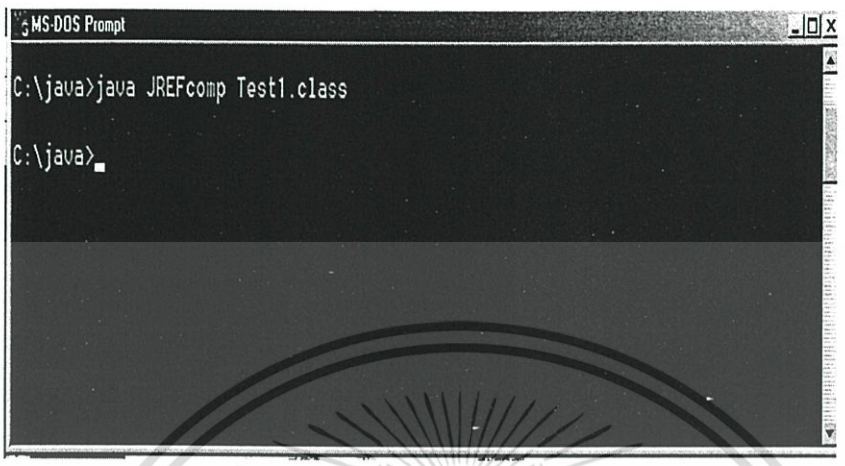
การ Compile

ในที่นี้จะเป็นการ Compile class Test1.class ซึ่งก่อนการ Compile จะมีรายละเอียดของ File ดังรูป



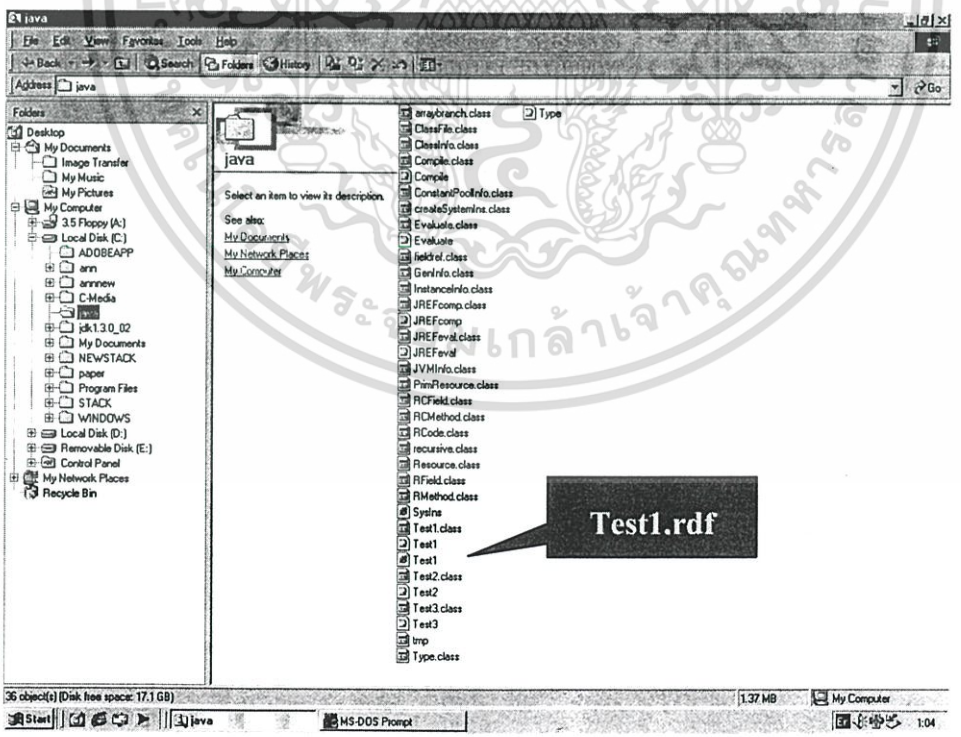
รูปที่ 3.4 ตัวอย่างรายละเอียดของ File ก่อนการ Compile

และใช้คำสั่งในการ Compile class Test1.class ดังรูป



รูปที่ 3.5 ตัวอย่างการใช้คำสั่ง Compile

และหลังจากการ Compile ก็จะมีการ generate file ที่มีชื่อเดียวกันกับ class Test1.class คือ Test1.rdf ซึ่งเป็น Resource Description File ของ Class นั้นนั่นเอง แสดงได้ดังรูป



รูปที่ 3.6 ตัวอย่างรายละเอียดของ file หลังการ Compile

การ Evaluate

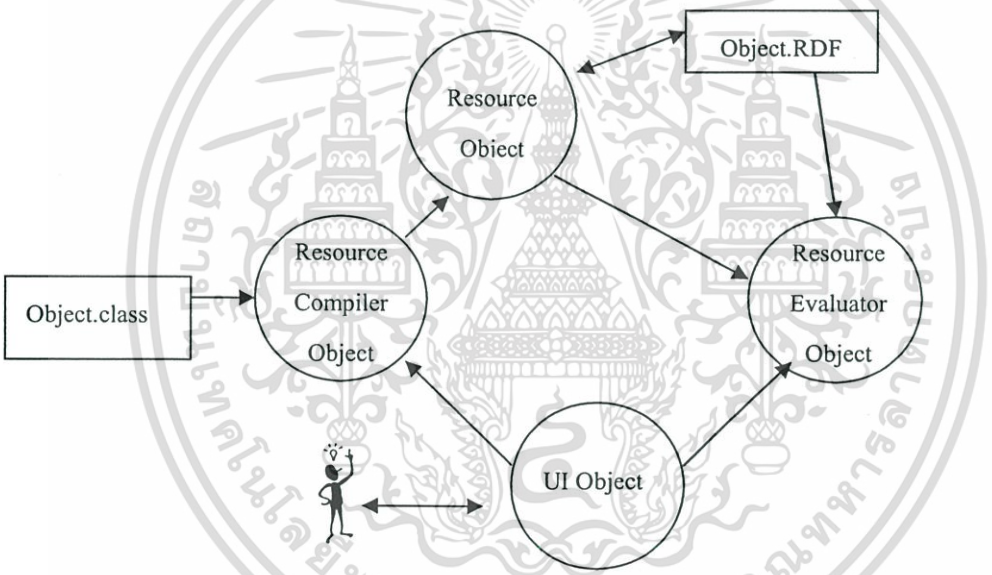
การ Evaluate จะเป็นการใช้คำสั่งในการ Evaluate ตามด้วยชื่อ RDF ที่ต้องการ Evaluate ซึ่งแสดงผลลัพธ์ได้ดังรูป

รูปที่ 3.7 ผลลัพธ์จากการ Evaluate (ส่วนที่ 1)

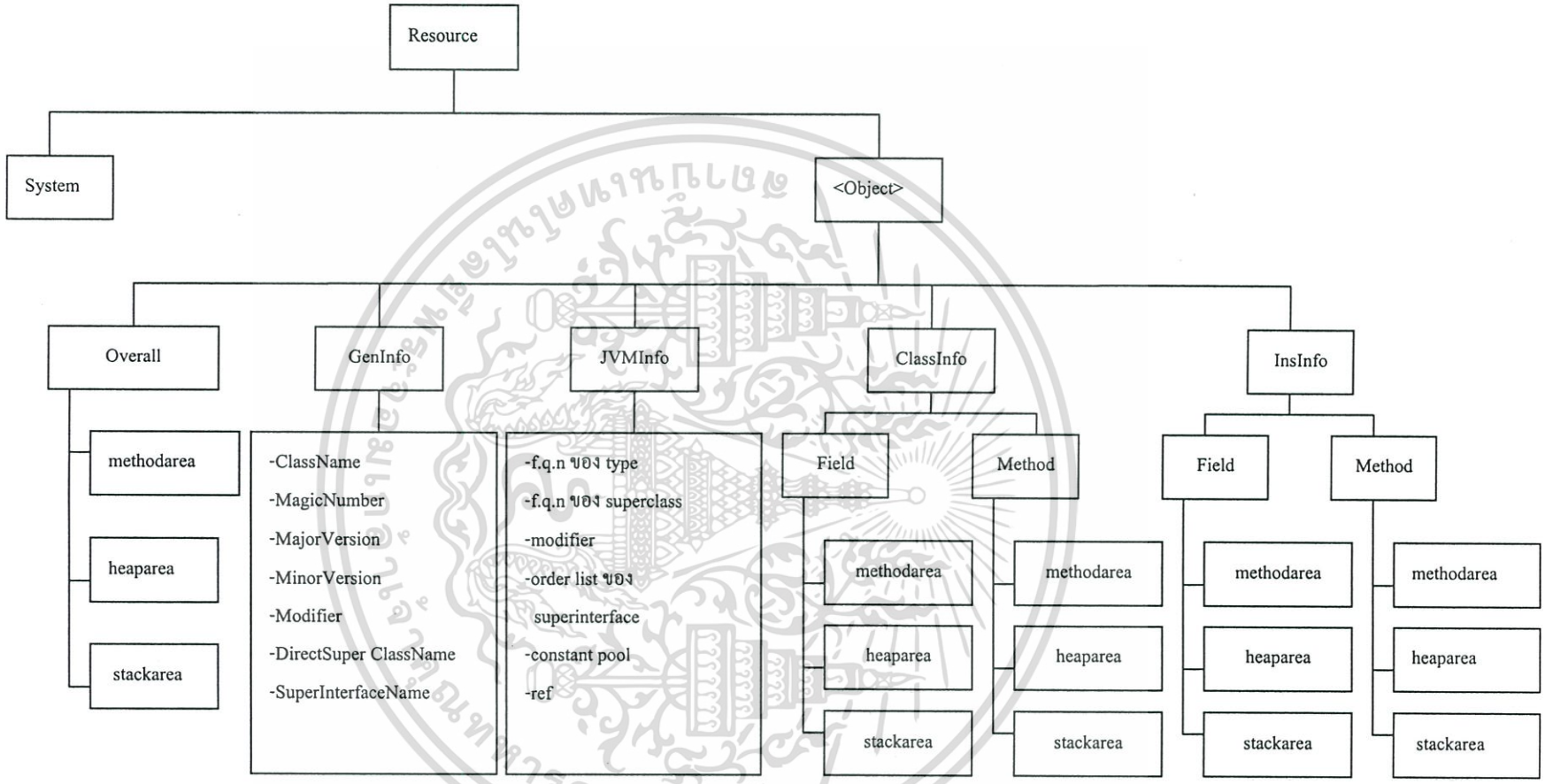
รูปที่ 3.8 ผลลัพธ์จากการ Evaluate (ส่วนที่ 2)

3.3 Resource Object (RO)

Resource Object เป็น Object หลักในกระบวนการทำงานของการวัดการใช้ทรัพยากรซึ่งจะสร้างขึ้นมาจากการ Compile จากนั้นก็จะนำข้อมูลที่ได้จาก Resource Object ไปสร้าง Resource Description File (RDF) ที่จะเก็บข้อมูลการใช้ทรัพยากรของ Object ที่กำลังทำการ Compile และในขั้นตอนของการ Evaluate เมื่อมีการ Load RDF ขึ้นมาเพื่อทำการ Evaluate ก็จะมีการสร้าง RO ที่สอดคล้องกับ RDF นั้นเพื่อทำการ Run และหาค่าปริมาณการใช้ทรัพยากร โดย Resource Object จะมีโครงสร้างตามการแบ่งประเภทของทรัพยากรที่จะทำการวัด ซึ่งในงานวิจัยนี้จะเป็นการวัดการใช้ทรัพยากรหน่วยความจำ 2 ประเภท คือ Heap และ Stack ดังนั้นจะได้โครงสร้างของ RO ดังรูปที่ 3.9



รูปที่ 3.9 โครงสร้างการทำงานของ Resource Object



รูปที่ 3.10 โครงสร้างของ Resource Object Attributes

3.3.1 Resource Object Attribute

เนื่องจากจากโครงสร้างของการใช้พื้นที่ของ Java Object ที่กล่าวมาข้างต้นประกอบด้วยพื้นที่พื้นฐาน 3 ส่วน คือ

1. Methodarea
2. Heaparea
3. Stackarea

ดังนั้นจึงมีการกำหนดให้ Attribute 3 ตัว คือ Methodarea , Heaparea และ Stackarea เป็น Attribute พื้นฐานสำหรับเก็บค่าการใช้ทรัพยากรของ Filed และ Method ทุก ๆ ตัวของ Object โดย Methodarea เป็นส่วนที่ดึงขึ้นมาได้โดยตรงจาก Class File ดังนั้นในงานวิจัยนี้จึง Focus ไปที่ 2 ส่วน คือ Heaparea และ Stackarea ซึ่งจะต้องทำการคำนวณหาโดย Attribute ของ Resource Object ประกอบด้วย 5 ส่วน คือ

1. GenInfo
2. JVMInfo
3. ClassInfo
4. InsInfo
5. PrimResource

โดยแต่ละส่วนจะประกอบด้วย attribute ต่าง ๆ ดังต่อไปนี้

1. GenInfo เป็นส่วนที่เก็บรายละเอียดทั่ว ๆ ไปเกี่ยวกับ Class และมี Attribute คือ
   - classname : ชื่อของ Class ของ Object
   - magicnumber : ค่า Magicnumber ของ Class
   - majorversion : ค่า Majorversion ของ Class
   - minorversion : ค่า Minorversion ของ Class
   - modifier : ค่า Modifier จะเป็นตัวบอกถึงสิทธิในการเข้าใช้ Class นี้
   - superclassname : ชื่อของ Class ที่เป็น Direct Super Class ของ Class นี้
   - interface* : ชื่อของ Interface ที่ถูก Implement โดย Class นี้ ซึ่งอาจมีได้หลายค่า
2. JVMInfo เป็นส่วนที่เก็บรายละเอียดของการใช้พื้นที่หน่วยความจำของรายละเอียดต่าง ๆ ของ Class โดยตามโครงสร้างดังรูปที่ 2 จะอยู่ในส่วนของ Methodarea ซึ่งประกอบด้วย Attribute ต่าง ๆ คือ
   - typenamesize : เป็นพื้นที่ที่ใช้เก็บชื่อของ Class
   - supertypenamesize : เป็นพื้นที่ที่ใช้เก็บชื่อของ Superclass ของ Class
   - modifiersize : เป็นพื้นที่ที่ใช้เก็บ Modifier
   - sunum : เป็นจำนวนของ Interface ที่ถูก Implement โดย Class นี้

- superinsize : เป็นพื้นที่ที่ใช้เก็บชื่อของ Interface ที่ถูก Implement โดย class นี้
- constantpoolsize : เป็นพื้นที่ของ Constant Pool ของ Class
- pcsize : เป็นพื้นที่ของ Program Counter
- reftoclassClasssize : เป็นพื้นที่ของ Reference ไปยัง Class Class
- reftoclassClassLoadersize : เป็นพื้นที่ของ Reference ไปยัง Class ClassLoader

โดย Attribute ทั้งหมดที่ปรากฏในส่วนของGenInfo และ JVMInfo เป็นค่าที่ดึงโดยตรงจาก Class File

3. ClassInfo เป็นส่วนรายละเอียดที่เกี่ยวข้องกับการใช้พื้นที่ของ Field และ Method ของ Class ที่มีเพียงค่าเดียวไม่ว่า Class จะมีการสร้าง Instance หรือ Object กี่ครั้งก็ตาม (เป็นพื้นที่ในส่วนของ Methodarea) ซึ่งประกอบด้วย

3.1 Field เก็บรายละเอียดของ Field และประกอบด้วย Attribute ย่อย คือ

- fname : ชื่อของ Field
- ftype : ชนิดของ Field
- fmodifier : Modifier ของ Field
- fnamesize : พื้นที่ที่ใช้เก็บชื่อของ Field
- ftypesize : พื้นที่ที่ใช้เก็บ Type ของ Field
- fmodsize : พื้นที่ที่ใช้เก็บ Modifier ของ Field

ซึ่งค่าของ Attribute ทั้งหมดข้างบนจะได้จากการดึงจาก Class File

- PrimResource ที่มี Attribute ย่อย คือ
  - methodarea : พื้นที่ของ Methodarea
  - heaparea : พื้นที่ของ Heaparea
  - stackarea : พื้นที่ของ Stackarea

ซึ่งเป็น Attribute ที่กำหนดขึ้นในงานวิจัยนี้

3.2 Method เก็บรายละเอียดของ Method และประกอบด้วย Attribute ย่อย คือ

- mname : ชื่อของ Method
- mnamsize : พื้นที่ที่ใช้เก็บชื่อของ Method
- mreturnparasize : พื้นที่ที่ใช้เก็บ Parameter
- mmodsize : พื้นที่ที่ใช้เก็บ Modifier
- mcodesize : พื้นที่ที่ใช้เก็บ Byte Code Instruction
- exceptsize : พื้นที่ที่ใช้เก็บรายละเอียดของ Exception

เป็นค่าที่ได้จากการดึงจาก Class File

- PrimResource : ประกอบด้วย
- methodarea : พื้นที่ของ Methodarea
- heaparea : พื้นที่ของ Heaparea
- stackarea : พื้นที่ของ Stackarea

4. InsInfo เป็นส่วนที่เก็บรายละเอียดของ Field และ Method ในการใช้พื้นที่หน่วยความจำในกรณีที่มีการสร้าง Instance และ Instance นั้นมีการทำงาน ซึ่งจะมีการใช้พื้นที่ของ Heaparea และ Stackarea โดยประกอบด้วย

4.1 Field เก็บรายละเอียดของ Field และประกอบด้วย Attribute ย่อย คือ

- fname : ชื่อของ Field
- ftype : ชนิดของ Field
- fmodifier : Modifier ของ Field

เป็นค่าที่ได้จากการดึงจาก Class File

- PrimResource : มี Attribute ย่อย คือ
- methodarea : พื้นที่ของ Methodarea
- heaparea : พื้นที่ของ Heaparea
- stackarea : พื้นที่ของ Stackarea

4.2 Method เก็บรายละเอียดของ Method และประกอบด้วย Attribute ย่อย คือ

- mname : ชื่อของ Method

เป็นค่าที่ได้จากการดึงจาก Class File

- mmaxstack : ค่าของ Operand Stack ที่ได้จากการคำนวณ
- mmaxlocal : ค่าของ Local Variable ที่ได้จากการคำนวณ
- PrimResource : มี Attribute ย่อย คือ
- methodarea : พื้นที่ของ Methodarea
- heaparea : พื้นที่ของ Heaparea
- stackarea : พื้นที่ของ Stackarea

5. Overall เป็นส่วนที่เก็บผลรวมของการใช้ทรัพยากรของ Object ซึ่งประกอบด้วย

- methodarea : พื้นที่ของ Methodarea
- heaparea : พื้นที่ของ Heaparea
- stackarea : พื้นที่ของ Stackarea

3.3.2 รายละเอียดของ Class ของ Resource Object

สำหรับ Class ต่าง ๆ ที่ทำงานร่วมกันที่ประกอบกันขึ้นเป็น RO จะเป็นไปตามโครงสร้างของ Resource Object Attributte ซึ่งมีโครงสร้างตาม UML ดังรูปที่ 3.11

จากรูปจะแสดง UML ของ Class ที่ทำงานร่วมกันเป็น Resource Object จะเห็นได้ว่าจะมีโครงสร้างเหมือนกับ RO Attributes ดังที่ได้กล่าวมาแล้วข้างต้น โดยมีรายละเอียด คือ

1. Class Resource : เป็น Class เริ่มต้นและมีการสร้าง Instance ทั้งหมด 5 ตัว คือ

   - gen เป็น Instance ของ Class GenInfo
   - jvm เป็น Instance ของ Class JVMInfo
   - cclass เป็น Instance ของ Class ClassInfo
   - instance เป็น Instance ของ Class InstanceInfo
   - overall เป็น Instance ของ Class PrimResource

   และมี Method คือ

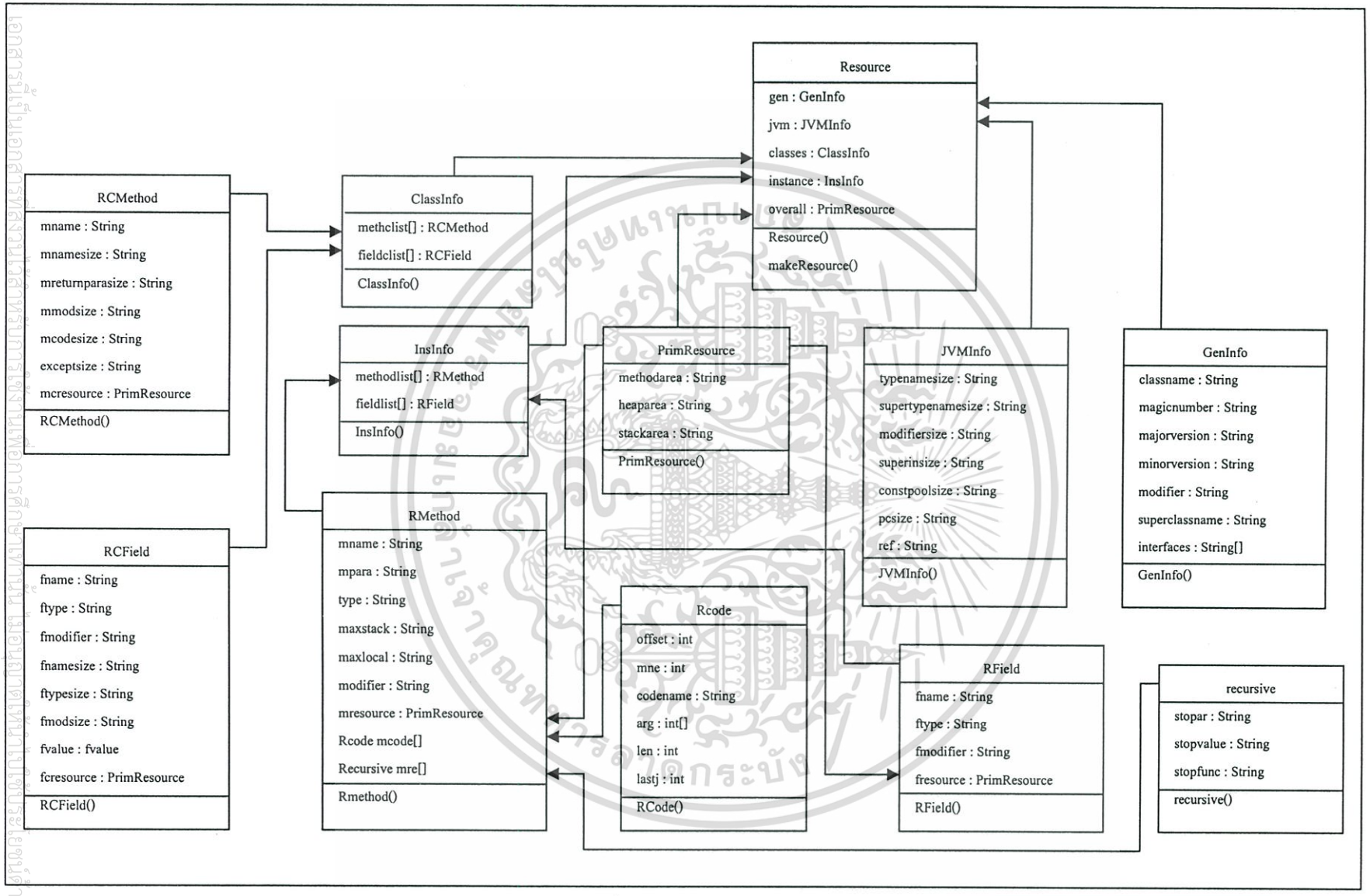
   - Resource() ซึ่งเป็น Method Constructor ของ Class นี้

2. Class GenInfo เป็น Class ที่ประกอบด้วย Filed ที่สอดคล้องกับ Attribute ที่ปรากฏในส่วนของ GenInfo ใน RO และมี Method คือ

   - GenInfo() ซึ่งเป็น Method Constructor ของ Class นี้

3. Class JVMInfo เป็น Class ที่ประกอบด้วย Filed ที่สอดคล้องกับ Attribute ที่ปรากฏในส่วนของ JVMInfo ใน RO และมี Method คือ

   - JVMInfo() ซึ่งเป็น Method Constructor ของ Class นี้



รูปที่ 3.11 UML ของ RO

4. Class ClassInfo

โดย Class จะมีการสร้าง Array ของ Instance ของ 2 Class คือ

- fieldclist ซึ่งเป็น Array ของ Class RCField ที่มีขนาดเท่ากับจำนวน Field ของ Class ของ RO
- methodclist ซึ่งเป็น Array ของ Class RCMethod ที่มีขนาดเท่ากับจำนวน Method ของ Class ของ RO

และมี Method คือ

- ClassInfo() ซึ่งเป็น Method Constructor ของ Class นี้

5. Class InsInfo

โดย Class จะมีการสร้าง Array ของ Instance ของ 2 Class คือ

- fieldlist ซึ่งเป็น Array ของ Class RField ที่มีขนาดเท่ากับจำนวน Field ของ Class ของ RO
- methodlist ซึ่งเป็น Array ของ Class RMethod ที่มีขนาดเท่ากับจำนวน Method ของ Class ของ RO

และมี Method คือ

- InsInfo() ซึ่งเป็น Method Constructor ของ Class นี้

6. Class PrimResource ซึ่งถูกสร้างเป็น Object พื้นฐาน โดยประกอบด้วย Field

- methodarea
- heaparea
- stackarea

7. Class RCField เป็น Class ที่ประกอบด้วย Field ที่เป็นรายละเอียดของ Field ของ Class File โดย Filed จะสอดคล้องตาม RO Attribute ในส่วน ของ Field ของ ClassInfo และมีการสร้าง Instance คือ

- fcresource ซึ่งสร้างจาก Class PrimResource

และมี Method คือ

- RCField() ซึ่งเป็น Method Constructor ของ Class นี้

8. Class RCMethod เป็น Class ที่ประกอบด้วย Field ที่เป็นรายละเอียดของ Method ของ Class File โดย Filed จะสอดคล้องตาม RO Attribute ในส่วน ของ Method ของ ClassInfo และมีการสร้าง Instance คือ

- mcresource ซึ่งสร้างจาก Class PrimResource

และมี Method คือ

- RCMethod() ซึ่งเป็น Method Constructor ของ Class นี้

9. Class RField เป็น Class ที่ประกอบด้วย Field ที่เป็นรายละเอียดของ Field ของ Class เมื่อมีการสร้าง Instance และมีการเรียกใช้ Field นี้ โดย Filed จะสอดคล้องตาม RO Attribute ในส่วน ของ Field ของ InstanceInfo และมีการสร้าง Instance คือ

- fresource ซึ่งสร้างจาก Class PrimResource

และมี Method คือ

- RField() ซึ่งเป็น Method Constructor ของ Class นี้

10. Class RMethod เป็น Class ที่ประกอบด้วย Field ที่เป็นรายละเอียดของ Method ของ Class เมื่อมีการสร้าง Instance และมีการเรียกใช้ Method นี้ โดย Filed จะสอดคล้องตาม RO Attribute ในส่วน ของ Method ของ InstanceInfo และมีการสร้าง Instance คือ

- mresource ซึ่งสร้างจาก Class PrimResource
- mcode ซึ่งเป็น Array ของ Class RCode ที่เก็บรายละเอียดของ Bytecode Instruction ของ Method
- mre เป็น Array ของ Class recursive ที่เก็บรายละเอียดของการ Recursive ของ Method

และมี Method คือ

- RMethod() ซึ่งเป็น Method Constructor ของ Class นี้

11. Class RCode เป็น Class ที่เก็บรายละเอียดของ Bytecode Instruction ของ Method โดยประกอบด้วย Field ดังต่อไปนี้

- offset เป็นค่าตำแหน่งของ Bytecode ใด ๆ เมื่อนับจาก Bytecode แรก ของ Method
- mne เป็นค่า Mnemonic ของ Bytecode
- codename เป็นชื่อของ Bytecode
- arg เป็น Array ของ Integer ซึ่งจะมีความยาวเท่ากับ Argument ของ Bytecode นั้น
- len เป็นความยาวของ Bytecode แต่ละตัว (จำนวน Byte ที่แทน Bytecode นั้น)
- lastj เป็นตำแหน่งสุดท้าย ของ Bytecode ก่อนที่จะมาทำที่ Bytecode นี้

และมี Method คือ

- RCode() ซึ่งเป็น Method Constructor ของ Class นี้

12. Class recursive เป็น Class ที่เก็บรายละเอียดของการ Recursive ของ Method ประกอบด้วย Field

- stopar คือ Argument ที่จะใช้ในการหยุดการ Recursive
- stopvalue คือ ค่าที่จะใช้ในการหยุดการ Recursive
- stopfunce คือ function ที่ใช้เมื่อการ Recursive หยุด

และมี Method คือ

- recursive() ซึ่งเป็น Method Constructor ของ Class นี้

Class ทั้งหมดนี้จะถูกสร้างเป็น Instance เพื่อเป็นส่วนประกอบภายใน Resource Object

3.4 Resource Description File (RDF)

Resorce Description File เป็น File ผลลัพธ์จากการ Compile ซึ่งจะเก็บข้อมูลการใช้ทรัพยากรของ Object เพื่อที่ Object จะไม่ต้องเข้าสู่กระบวนการของการ Compile ทุกครั้ง (ในกรณีที่ Class File ไม่มีการเปลี่ยนแปลง) และในขั้นตอนของการ Evaluate จะต้อง Load RDF ขึ้นมาเพื่อใช้ข้อมูลจาก RDF ในการประเมินค่าการใช้ทรัพยากรและ RDF จะมีโครงสร้างตามโครงสร้างของ Resource Object Attributes (รูปที่ 3.10) โดยข้อมูลที่เก็บใน RDF แบ่งเป็น 2 ประเภท คือ

1. ค่าคงที่ซึ่งเป็นค่าที่สามารถดึงมาโดยตรงจาก Class File
2. Expression ซึ่งติดค่าตัวแปรไว้และจะต้องทำการ Evaluate เพื่อหาค่าการใช้ทรัพยากรจริงของ Object

และจากโครงสร้างของ Resource Object Attributes จะสามารถสร้าง Resource Description File(RDF) ที่สอดคล้องกับโครงสร้างข้างต้น โดย RDF จะเขียนด้วย XML

3.4.1 Document Type Definiton ของ RDF

จะมีรายละเอียดที่สามารถอธิบายด้วย DTD(Document Type Definition) ซึ่งเป็นภาษาที่ทำหน้าที่กำหนดกฎ กติกาให้กับ Element , Attribute และ Entity ที่อยู่ในแหล่งข้อมูลของ XML ซึ่งในที่นี้ DTD ของ RDF จะมีลักษณะดังนี้ดังรูปที่ 3.12

```
<?xml version="1.0" encoding="windows-874"?>
<!DOCTYPE Object[
<!ELEMENT Object(GenInfo,JVMInfo,ClassInfo,InstanceInfo,Overall)>
<!ELEMENT GenInfo
 (classname,magicnumber,majorversion,minorversion,modifier,superclassname,interface*)>
<!ELEMENT classname (#PCDATA)>
<!ELEMENT magicnumber (#PCDATA)>
<!ELEMENT majorversion (#PCDATA)>
<!ELEMENT minorversion (#PCDATA)>
<!ELEMENT modifier (#PCDATA)>
<!ELEMENT superclassname (#PCDATA)>
<!ELEMENT interface (#PCDATA)>
<!ELEMENT JVMInfo
 (typenamesize,supertypenamesize,modifiersize,superinsize,constpoolsize,pcsize,ref )>
<!ELEMENT typenamesize (#PCDATA)>
<!ELEMENT supertypenamesize (#PCDATA)>
<!ELEMENT modifiersize (#PCDATA)>
<!ELEMENT superinsize (#PCDATA)>
<!ELEMENT constpoolsize (#PCDATA)>
<!ELEMENT pcsize (#PCDATA)>
<!ELEMENT ref (#PCDATA)>
<!ELEMENT ClassInfo (RCField,RCMethod)>
<!ELEMENT RCField
 (fnumber,fname,ftype,fmodifier,fnamesize,ftypesize,fmodsize,PrimResource)>
<!ELEMENT fnumber (#PCDATA)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT ftype (#PCDATA)>
<!ELEMENT fmodifier (#PCDATA)>
<!ELEMENT fnamesize (#PCDATA)>
```

```
<!ELEMENT ftypesize (#PCDATA)>
<!ELEMENT fmodsize (#PCDATA)>
<!ELEMENT PrimResource (methodarea,heaparea,stackarea)>
<!ELEMENT RCMethod
 (mname,mreturnparasize,mmodsize,mcodesize,exceptsize,PrimResource)>
<!ELEMENT mname (#PCDATA)>
<!ELEMENT mreturnparasize (#PCDATA)>
<!ELEMENT mmodsize (#PCDATA)>
<!ELEMENT mcodesize (#PCDATA)>
<!ELEMENT exceptsize (#PCDATA)>
<!ELEMENT PrimResource (methodarea,heaparea,stackarea)>
<!ELEMENT InsInfo (fieldsum,RField,RMethod)>
<!ELEMENT fieldsum(#PCDATA)
<!ELEMENT RField (fname,ftype,fmodifier,PrimResource)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT ftype (#PCDATA)>
<!ELEMENT fmodifier (#PCDATA)>
<!ELEMENT PrimResource (methodarea,heaparea,stackarea)>
<!ELEMENT RMethod
 (mname,mparameter,mtype,maxstack,maxlocal,modifier,PrimResource)>
<!ELEMENT mname (#PCDATA)>
<ELEMENT mparameter(#PCDATA)>
<ELEMENT mtype(#PCDATA)>
<!ELEMENT maxstack (#PCDATA)>
<!ELEMENT maxlocal (#PCDATA)>
<!ELEMENT modifier (#PCDATA)>
<!ELEMENT PrimResource (methodarea,heaparea,stackarea)>
]>
```

รูปที่ 3.12 โครงสร้างของ Resource Description File(RDF)

3.4.2 ค่าที่ปรากฏใน RDF

ค่าต่าง ๆ ที่จะปรากฏใน RDF แบ่งได้เป็น 2 ประเภท คือ

1. ค่าคงที่ต่าง ๆ ที่สามารถดึงขึ้นมาโดยตรงจาก Class File เช่น ชื่อของ Class File เป็นต้น ซึ่งจะปรากฏใน RDF แสดงได้เป็น

```
<classname>test</classname>
```

2. Expression ซึ่งจะอยู่ในรูปแบบที่มีการติดค่าของตัวแปรไว้ โดยตัวแปรนั้นต้องสอดคล้องกับโครงสร้างของ RO และ RDF ที่กล่าวมาข้างต้นซึ่งจะอยู่ในลักษณะ เช่น Attribute Heaparea ของ Object ที่ปรากฏใน RDF แสดงได้เป็น

```
<heaparea> (classname).instance.method1().mresource.heaparea + 10</ heaparea>
```

โดยใน Expression จะประกอบด้วยสิ่งต่าง ๆ ต่อไปนี้ (ตาม Format ใน[19])

1. operator: +, -, *, /, ^, %, cos, sin, tan, acos, asin, atan, sqrt, sqr, log, min, max, ceil, floor, abs, neg, rnd
2. ตัวเลข 0-9
3. ตัวแปรซึ่งจะอยู่ในรูปที่สอดคล้องกับโครงสร้างของ RO และ RDF เช่น

(classname)..instance.method1().mresource.heaparea

3.4.3 รูปแบบของ Expression ของ Stackarea และ Heaparea

3.4.3.1 Stackarea Expression

สิ่งที่จะปรากฏใน Stackarea Function ของ Method จะประกอบด้วย

- ค่า Stackarea ของ Method ที่ Method นั้น Invoke ซึ่งอยู่ในรูปแบบ

```
(classname).instance.methodname(type of parameter : value of parameter).mresource.stackarea
```

ตัวอย่าง (java/io/PrintStream).instance.println(I:5];).mresource.stackarea

ตัวอย่างของ Stackarea Expression

max((java/io/PrintStream).instance.println(I:l_int[invoke19.xI];).mresource.stackarea) + 2 + 2

3.4.3.2 Heaparea Expression

สิ่งที่จะปรากฏใน Heaparea Expression ของ Method ประกอบด้วย

1. ค่า Heaparea ของ Method ที่ Method นั้น Invoke ซึ่งอยู่ในรูปแบบ

(classname).instance.methodname(type of parameter : value of parameter).mresource.heaparea

ตัวอย่าง (java/io/PrintStream).instance.println(I:l_int[invoke19.xI];).mresource.heaparea

2. Array 1 มิติ ซึ่งอยู่ในรูปแบบ

[arraytype[arraylength]

ตัวอย่าง [I[5]

3. Array หลายมิติ ซึ่งอยู่ในรูปแบบ

[[…arraytype[arraylength of dimension 1][arraylength of dimension 2]…

ตัวอย่าง [[I[5][6]

โดย Arraytype มีความหมายดังตาราง

ตาราง 3.1 Type ของ Array ที่ปรากฏใน Expression

Arraytype	Mean
Z	Boolean type
B	Byte type
C	Character type
S	Short type
I	Integer type
F	Float type
L	Long type
D	Double type
L…	Reference type

4. Variable fieldsum ของ Class ที่ Method นี้มีการสร้าง Instance ซึ่งอยู่ในรูปแบบ

(classname).instance.fieldsum

ตัวอย่าง (Invoke19).instance.fieldsum

ตัวอย่างของ Heaparea Expression

(invoke6).instance.fieldsum+(invoke6).instance.init(:).mresource.heaparea

โดยค่าของ value of parameter , arraylength จะประกอบด้วย

1. ค่าคงที่
   - เลขจำนวนเต็ม
   - เลขทศนิยม
   - String
2. Argument ที่ส่งเข้ามายัง Method นั้น ซึ่งอยู่ในรูปแบบ

argument[index]

ตัวอย่าง argument[0]

3. ค่าจาก Local Variable
4. ค่าจาก Method ซึ่งจะอยู่ในรูปแบบ

(classname).instance.methodname(type of parameter : value of parameter).value

ตัวอย่าง (invoke19).instance.getA(I:5).value

5. ค่าความยาวจาก Array อื่น ซึ่งจะเป็นค่าคงที่
6. Class Variable ซึ่งอยู่ในรูปแบบ

classname.fieldnamefiledtype

ตัวอย่าง invoke19.xI

7. Instance Variable ซึ่งอยู่ในรูปแบบ

Lclassname(parametertype:parametervalue).fieldnamefiledtype

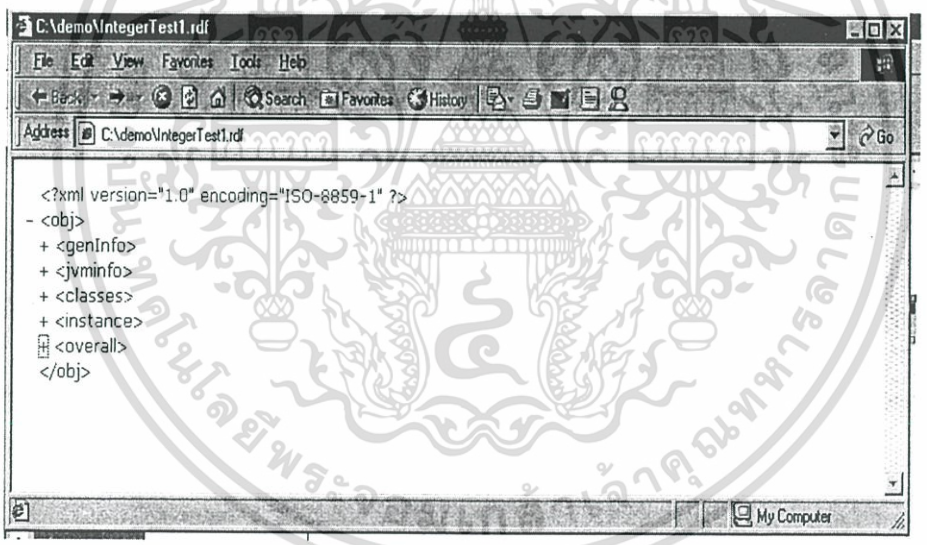
ตัวอย่าง Ltann2(:).xI;)

8. ค่าจาก Array ซึ่งอยู่ในรูปแบบ

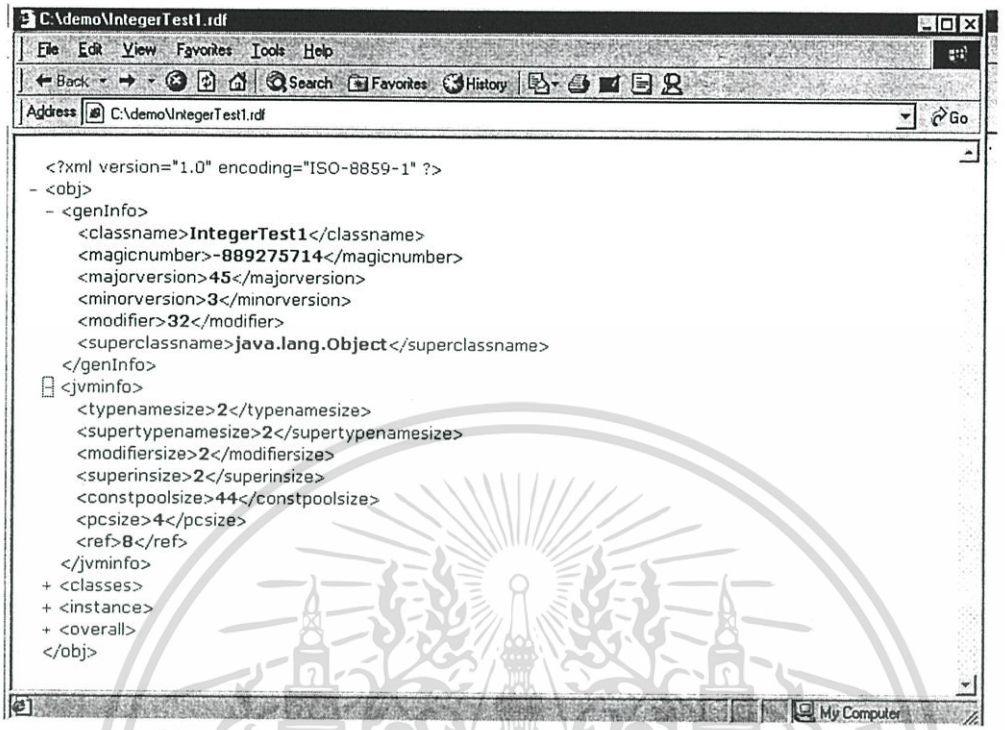
aload_index of array_index of data

ตัวอย่าง aload_0_0

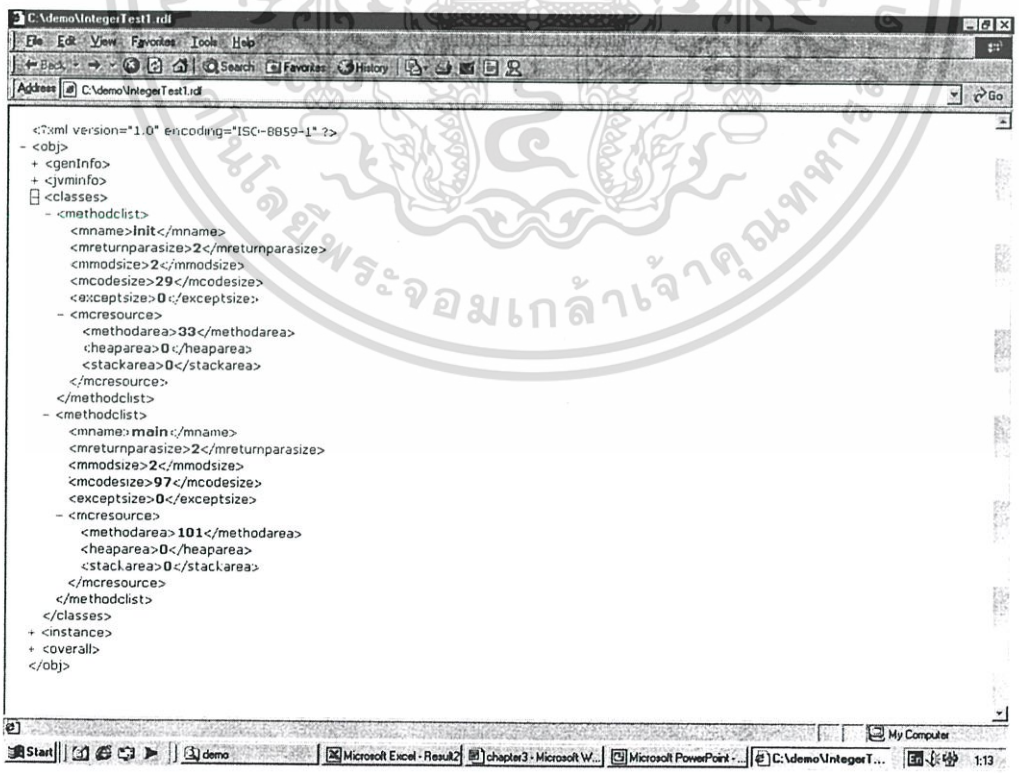
ตัวอย่างของ Resource Description File(RDF) แสดงได้ดังรูป



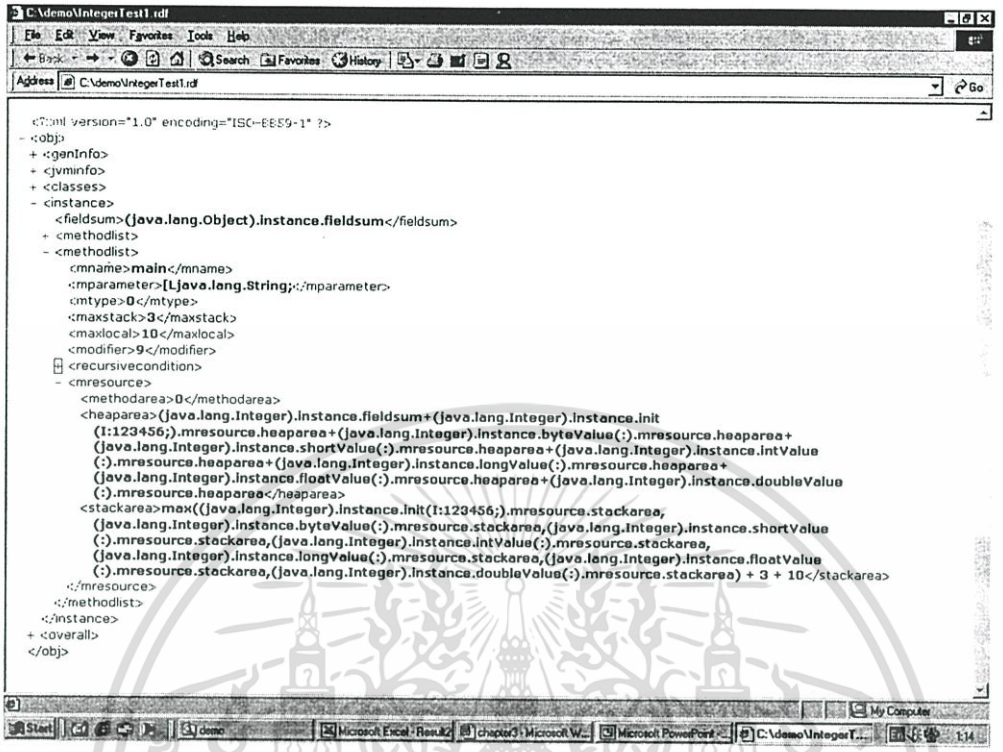
รูปที่ 3.13 ภาพรวมของ Resource Description File



รูปที่ 3.14 Resource Description File ในส่วนของ genInfo และ jvmInfo



รูปที่ 3.15 Resource Description File ในส่วนของรายละเอียดของ Class



รูปที่ 3.16 Resource Description File ในส่วนของรายละเอียดของ Instance

3.5 การวัดค่าการใช้ทรัพยากร

ในส่วนนี้จะกล่าวถึงการภาพรวมของการหาค่าการใช้ทรัพยากรทั้ง Heaparea และ Stackarea

3.5.1 Heaparea

สำหรับ Heaparea จะหมายถึง Attribute heaparea ที่ปรากฏใน RO และ RDF ประกอบด้วย

1. Heaparea ของ Instance จะมีการใช้พื้นที่ 3 ส่วน คือ
   - พื้นที่ในการเก็บ Reference ของ Instance
   - พื้นที่ในการเก็บ Reference ไปยัง Class ของ Object
   - พื้นที่ของ Instance ซึ่งก็คือ Non-Static Variable ของ Class และพื้นที่ของ Non-Static Variable ของ Super Class (ถ้ามี)

โดยการใช้พื้นที่ Heaparea ของ Object จะมีลักษณะดังรูปที่ 2.2

จากรูปจะเห็นได้ว่า Object จะใช้พื้นที่ใน 2 ส่วน คือ ส่วนของ Heaparea และ Methodarea โดยในส่วนของ Heaparea จะเก็บ Reference ไปยัง Instance Data , Reference ไปยัง Class Data ของ Object นั้น และ Instance Data ของ Object ดังนั้นจากรูปเราสามารถหาพื้นที่ของ Heap ที่ Object แต่ละตัวได้ดังนี้ คือ

พื้นที่ heaparea ของ object = ro + rc + fieldsum

เมื่อ ro เป็น พื้นที่เก็บ Reference ของ Instance(Object)

rc เป็น พื้นที่ในการเก็บ Reference ไปยัง Class ของ Object

filedsum เป็น ผลรวมของพื้นที่ของ Field ที่ไม่มี Modifier เป็น Static (Instance Variable)

2. Heaparea ของ Array ซึ่งแบ่งได้เป็น

- Array ของ Primitive Type เช่น Array ของ Int , Long จะประกอบด้วยพื้นที่ของ

1. Reference ของ Array
2. พื้นที่ของ Element ซึ่งจะได้จากการตรวจสอบชนิดของ Array นั้น * length

- Array ของ Reference Type เช่น Array ของ Class จะประกอบด้วยพื้นที่ของ

1. Reference ของ Array
2. พื้นที่เก็บ Element ของ Array ที่เป็น Reference ไปยัง Class Instance = 4 * length ของมิติ
3. พื้นที่ของ Instance แต่ละตัวที่นำมาเป็นสมาชิกของ Array ซึ่งจะเป็นพื้นที่ Heaparea ของ Class ที่นำมาสร้างเป็น Array Object นั้น

ตัวอย่างการใช้พื้นที่ Heaparea ของ Array int[][] ar = new int[2][2] จะมีลักษณะดังรูปที่ 2.3

จากรูปจะเห็นว่า Array จะมีการใช้หน่วยความจำในส่วนของ Heaparea สำหรับเก็บข้อมูลของ Array และ Reference ไปยังมิติอื่น ๆ , Reference ไปยัง Class Data , พื้นที่เก็บความยาวในแต่ละมิติของ Array ดังนั้นพื้นที่ของ Heaprea ที่ใช้สำหรับเก็บข้อมูลของ Array สามารถคำนวณได้จาก

1. กรณี Array 1 มิติ

สามารถคำนวณหาพื้นที่ของ Array ได้โดย

พื้นที่ของ array = พื้นที่ของ reference + พื้นที่สำหรับเก็บข้อมูลตาม type และความยาวของ array

2. กรณี Array ตั้งแต่ 2 มิติขึ้นไป

สามารถคำนวณหาพื้นที่ของ array ได้โดย

กำหนดให้

n เป็นมิติของ array

$l_0 \ldots l_{n-1}$ เป็นความยาวในแต่ละมิติของ array

ดังนั้นพื้นที่ของ Array จะมีค่าเท่ากับ

พื้นที่ของมิติที่ 1 + พื้นที่ของมิติที่ 2 + … + พื้นที่ของมิติที่ n – 1 + พื้นที่ของมิติที่ n

หรือ

พื้นที่ของ array type reference ตามความยาวของ l_0 +
(พื้นที่ของ array type reference ตามความยาวของ $l_1 * l_0$) +… +
(พื้นที่ของ array type reference ตามความยาวของ $l_{n-2} * l_{n-3} *…*l_0$) +
(พื้นที่ของ array type ของ array นั้นตามความยาวของ $l_{n-1} * l_{n-2} *…*l_0$)

โดยพื้นที่ของแต่ละมิติหาได้เช่นเดียวกับการหาพื้นที่ของ Array 1 มิติ

3.5.2 Stackarea

Stackarea ซึ่งเป็นพื้นที่ในการทำงานของ Method ที่ประกอบด้วย

1. Operand Stack ซึ่งเป็นพื้นที่ในการทำงานของ Bytecode Instruction โดยจะมีขนาดที่เพิ่มและลดตลอดเวลา (ตาม Bytecode Instruction นั้น) ดังนั้นในการวัดการใช้พื้นที่จะเป็นการวัดพื้นที่สูงสุดที่คาดว่า Method นั้นจะใช้ โดยการเพิ่ม-ลดค่าของ Operand Stack (ดูการเปลี่ยนแปลงของแต่ละคำสั่งตามภาคผนวก ก) ซึ่งในการทำงานของ JVM จะมีการ Allocate พื้นที่ของ Operand Stack นี้เมื่อมีการ Invoke Method[3][8]
2. Local Variable ซึ่งเป็นพื้นที่ของ
   1. Argument ของ Method
   2. Local Variable ของ Method

และในการทำงานของ JVM จะมีการ Allocate พื้นที่ของส่วนนี้เมื่อมีการ Invoke Method[3][8] เช่นเดียวกับ Operand Stack

เนื่องจากการทำงานของ Java Stack จะทำงานในหน่วยของ Word ดังนั้นการวัดในกรณีนี้จึงวัดเป็นหน่วย Word

3.6 Algorithm ในการประเมินค่าการใช้ทรัพยากร

ในที่นี้จะกล่าวถึงโดยแยกเป็น 2 ส่วน คือ Compile และ Evaluate

3.6.1 การ Compile

Concept ของการ Compile

1. เป็นการหารายละเอียดการใช้ heap และ stack ของทุก method ของ class file
2. ค่าที่ได้จากการ Compile อาจเป็นค่าคงที่หรือ Expression
3. ใน Method แต่ละ Method อาจจะมีการอ้างถึง Object/Class อื่นทำให้ต้องมีการติดค่าในรูปแบบของ Expression ย่อย
4. โดย Expression เกิดจากการหา Bytecode ที่เป็นคำสั่งที่ทำให้เกิดการใช้ Heap และ Stack ซึ่งจะมีรูปแบบตาม RDF

Algorithm ของการ Compile

```
static Resource obj;
public static void main(String args[]){
        obj = new Resource();
        File file = new File(args[0]);BCClass classFile = new BCClass(file);
        ClassFile cl = getCP(args[0]);
        getGenInfo(classFile);getJVMInfo(classFile,obj.geninfo.interfaces.length);
        getClass(classFile);getInstance(classFile,cl);
        getOverall();createRDF(obj , args[0]);
}
getInstance(classFile,cl){
        getMethod();
        for i = 1 to n { //n is number of method
                getBytecode();
                StringBuffer heapexpression = new StringBuffer();
                StringBuffer stackexpression = new StringBuffer();
                int begin = 0;
                getExpression(bytecode,heapexpression.stackexpression,begin)
        }
}
```

รูปที่ 3.17 Algorithm ของการ Compile

Concept ของการหา Expression ของ Heaparea

1. สร้าง Stringbuffer ขึ้นมาเพื่อ Append Expression ย่อย
2. หา Bytecode ที่จะมีการใช้ Heaparea ซึ่งประกอบด้วย
   - new
   - newarray ; anewarray,multianewarray
   - invokevirtual,invokestatic,invokespecial,invokeinterface
3. แล้ว Append รายละเอียดลง Stringbuffer
4. ในกรณีพบทางเลือกหา Target ของทางเลือกนั้น แล้ว Append "max" ลง Stringbuffer เพื่อใช้ในการเปรียบเทียบในแต่ละทางเลือก แล้วหา Bytecode ไปตามทางเลือก

Algorithm ของการหา Expression ของ Heaparea

```
getExpression(bytecode,heapexpression,stackexpression,begin){
        for i = begin to n{
            if(bytecode[i] = new) then
                getInstanceDescription();
                heapexpression.append(instance expression);
            else if(bytecode[i] = newarray or anewarray or multianewarray) then
                getArrayDescription();
                heapexpression.append(array expression);
            else if(bytecode[i] = invokevirtual or invokestatic or invokespecial or
                invokeinterface) then
                getMethodDescription();
                heapexpression.append(method expression);
            else if(bytecode[i] is branch instruction) then
                int x = GetInstructiontarget();
                heapexpression.append("max")
                getExpression(bytecode,heapexpression.stackexpression,x);
                int y = i – 1;
                getExpression(bytecode,heapexpression.stackexpression,y);
        }
}
```

รูปที่ 3.18 Algorithm ของการหา Expression ของ Heaparea

ตัวอย่างของการหา Expression ของ Heaparea

สมมติมี class

```
Class Test{
    public static void main(Sting args[]){
        int a = Integer.parseInt(args[0]);
        if(a<10){
            int[] x = new int[5];
        }else{
            tmethod();
        }
    }
    static void tmethod(){
        Type a = new Type();
    }
}
```

รูปที่ 3.19 ตัวอย่าง Class ในการหา Expression

ซึ่งจะมี Bytecode เป็น

```
MS-DOS Prompt
Method void main(java.lang.String[])
   0 aload_0
   1 iconst_0
   2 aaload
   3 invokestatic #2 <Method int parseInt(java.lang.String)>
   6 istore_1
   7 iload_1
   8 bipush 10
  10 if_icmpge 20
  13 iconst_5
  14 newarray int
  16 astore_2
  17 goto 23
  20 invokestatic #3 <Method void tmethod()>
  23 return

Method void tmethod()
   0 new #4 <Class Type>
   3 dup
   4 invokespecial #5 <Method Type()>
   7 astore_0
   8 return
```

รูปที่ 3.20 Bytecode Instruction ของตัวอย่าง Class ในการหา Expression

และจะได้ Expression ของ Heaparea ของ Method main และ tmethod คือ

```
main :
    <heaparea>(java.lang.Integer).instance.parseInt(Ljava.lang.String;:aload_0_0;).mresource.heaparea+
        max(((Test).instance.tmethod(:).mresource.heaparea),([I[5]))</heaparea>
```

```
tmethod :
    <heaparea>(Type).instance.fieldsum+(Type).instance.init(:).mresource.heaparea</heaparea>
```

Concept ของการหา Expression ของ Stackarea

สำหรับการสร้าง Function ของ Stackarea เป็นการหาว่าใน Method นี้มีการ Invoke ไปยัง Method อื่นบ้างหรือไม่เนื่องจากการเมื่อมีการ Invoke Method จะมีการ Allocate พื้นที่ที่เรียกว่า Stack Frame เพื่อเป็นพื้นที่ในการทำงานของ Method นั้น ๆ ซึ่งประกอบด้วย Operand Stack , Local Variable และจะมีการคืนพื้นที่เมื่อการทำงานของ Method นั้นสิ้นสุดลง ดังนั้นจะเป็นการวัดว่า Method ที่ถูก Invoke ทั้งหมดใน Method นี้มี Method ไหนที่ใช้พื้นที่สูงสุดรวมกับพื้นที่ของ Operand Stack และ Local Variable ของ Method ปัจจุบัน

Algorithm ของการหา Expression ของ Stackarea

```
stackexpression.append("max");
getExpression(bytecode,heapexpression.stackexpression,begin){
   // ในขณะที่มีการหา expression จะมีการหา maxstack และ maxlocal of method ปัจจุบันไปด้วย
        for i = 1 to n {
                if(bytecode[i] = invoke method instruction) then
                        getMethodDescription();
                        stackexpression.append(method expression);
                endif
        }
        stackexpression.append("+");stackexpression.append(maxstack);
        stackexpression.append("+");stackexpression.append(maxlocal);
}
```

รูปที่ 3.21 Algorithm ของการหา Expression ของ Stackarea

Concept ของการหา Maxstack ของ Method

เป็นการหา Operand Stack ซึ่งเป็นพื้นที่ในการทำงานของ Bytecode Instruction ซึ่ง JVM จะทำงานในลักษณะของ Stack Oriented โดยการ Push และ Pull ค่าจาก Oprand Stack ซึ่งการวัดจะเป็นการหาค่าพื้นที่ของ Operand Stack สูงสุดที่ Bytecode Instruction ของ Method ใช้เป็นพื้นที่ในการทำงาน ยกตัวอย่างมีชุดของ Bytecode

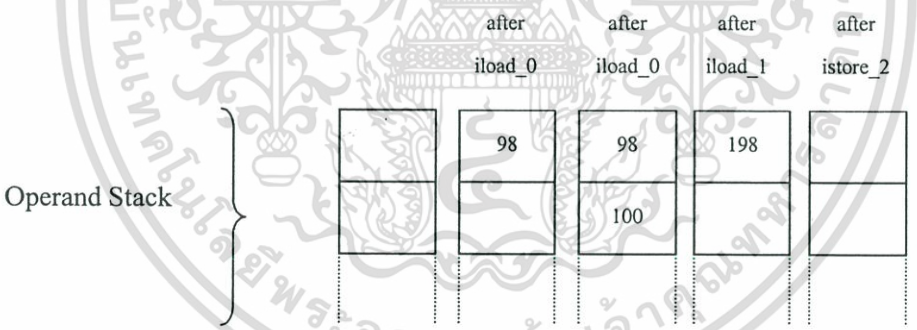
iload_0 : เป็นคำสั่งสำหรับ load ค่าจาก local variable ตำแหน่งที่ 0 สู่ Operand Stack ซึ่งจะทำให้ขนาดของ Operand Stack เพิ่มขึ้น 1 word

iload_1 : เป็นคำสั่งสำหรับ load ค่าจาก local variable ตำแหน่งที่ 0 สู่ Operand Stack ซึ่งจะทำให้ขนาดของ Operand Stack เพิ่มขึ้น 1 word

iadd : เป็นคำสั่งสำหรับบวกค่า 2 ค่าที่อยู่บนสุดของ Operand Stack ซึ่งจะทำให้ขนาดของ Operand Stack ลดลง 1 word

istore_2 : เป็นคำสั่งสำหรับนำค่าที่อยู่บนสุดของ Operand Stack ไปเก็บใน local variable ตำแหน่งที่ 2 ซึ่งจะทำให้ขนาดของ Operand Stack ลดลง 1 word

ซึ่งจะมีการใช้ Operand Stack ดังรูป



รูปที่ 3.22 ตัวอย่างการใช้พื้นที่ของ Operand Stack

และจากตัวอย่างนี้จะมีการใช้พื้นที่ของ Operand Stack (maxstack) สูงสุด คือ 2 word

Algorithm ของการหา Maxstack

```
int maxstack = 0 , m = 0
for i = 1 to n{ //  n is  bytecode instruction's number
        if(bytecode[i] is invoking method bytecode) then
                int a = getParameterNumber();int b = getReturnNumber();
                m = m – a + b;
        else
                int stackchange = getBytecodeStack();
        m = m + stack change;
        endif
        if(m>maxstack) then
                maxstack = m;
        end if
}
```

รูปที่ 3.23 Algorithm ของการหา Maxstack

ตัวอย่างการหา Maxstack จากชุดของ Bytecode

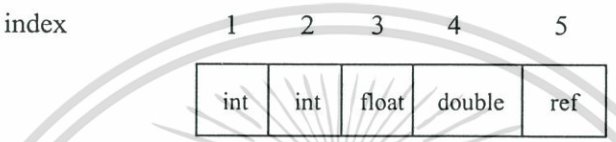
Method void main(java.lang.String[])		
0 iconst_5	m = m+1 = 1	maxstack = 1
1 istore_1	m = m-1 = 0	maxstack = 1
2 iload_1	m = m+1 = 1	maxstack = 1
3 bipush 10	m = m+1 = 2	maxstack = 2
5 if_icmpge 15	m = m-2 = 0	maxstack = 2
8 iconst_5	m = m+1 = 1	maxstack = 2
9 newarray int	m = m+0 = 1	maxstack = 2
11 astore_2	m = m-1 = 0	maxstack = 2
12 goto 18	m = m+0 = 0	maxstack = 2
15 invokestatic #2 <Method void tmethod()>	m = m-0+0 = 0	maxstack = 2
18 return	m = m-0 = 0	maxstack = 2

รูปที่ 3.24 ตัวอย่างการหา Maxstack

ซึ่งจะได้ค่าของ Maxstack = 2 word

Concept ของการหา Maxlocal ของ Method

สำหรับ Local Variable จะเป็นพื้นที่สำหรับการเก็บ Local Variable ของ Method นั้น ซึ่งเกิดจากผลรวมของ จำนวน Argument ที่ถูกส่งผ่านเข้ามายัง Method และ จำนวน Local Varible ที่กำหนดใน Method โดยเมื่อมีการ Invoke Method จะมีการ Allocate พื้นที่สำหรับเก็บ Argument และ Local Variable สำหรับ Method นั้นซึ่งพื้นที่จะอยู่ในรูปแบบของ array และเมื่อมีการอ้างถึงจะอ้างตาม Index ของ Array ดังรูป



รูปที่ 3.25 ตัวอย่างการใช้พื้นที่ของ Local Variable

โดยในงานวิจัยนี้ค่าของ Maxlocal หาจากการอ้างถึง Array ของ Local Variable ที่มี Index สูงสุดใน Method

Algorithm ของการหาค่า Maxlocal ของ Method

```
int maxlocal = 0
for i = 1 to n do   // n is bytecode's number
        if(bytecode[i] is local variable instruction) then
                int index = getBytecodeIndex();
                if(index > maxlocal) then
                        maxlocal = index;
                endif
        endif
}
maxlocal = maxlocal + 1                 //index of array begin with 0
```

รูปที่ 3.26 Algorithm ของการหา Maxlocal

ตัวอย่างการหาค่าของ Maxlocal จากชุดของ Bytecode

```
Method void main(java.lang.String[])
  0 iconst_5
  1 istore_1                    index = 1      maxlocal = 1
  2 iload_1                     index = 1      maxlocal = 1
  3 bipush 10
  5 if_icmpge 15
  8 iconst_5
  9 newarray int
 11 astore_2                    index = 2      maxlocal = 2
 12 goto 18
 15 invokestatic #2 <Method void tmethod()>
 18 return
```

รูปที่ 3.27 ตัวอย่างการหาค่า Maxlocal

ซึ่งจะได้ค่าของ Maxlocal = 2 + 1 = 3 words

และจากตัวอย่าง Class ในรูปที่ 3.19 จะได้ค่าของ Expression ของ Stackarea ของ Method main และ tmethod เป็น

```
main :
<stackarea>max((java.lang.Integer).instance.parseInt(Ljava.lang.String;:aload_0_0;).
mresource.stackarea,(Test).instance.tmethod(:).mresource.stackarea) + 2 + 3</stackarea>
```

```
tmethod :
        <stackarea>max((Type).instance.init(:).mresource.stackarea) + 2 + 1</stackarea>
```

3.6.2 การ Evaluate

Concept ของการ Evaluate

1. เป็นการหาค่าจาก Expression ของ Heaparea และ Stackarea ของ Method
2. โดยในที่นี้จะเริ่ม Evaluate จาก Method main
3. ถ้าใน Expression นั้นมีการอ้างถึง Class/Object อื่นต้องเข้าไป Evaluate Expression ที่ถูกอ้างถึงของ Class/Object นั้น

Algorithm ของการ Evaluate

```
public static void main(String args[]){
        Resource obj =  readXML(args[0],obj);
        for i = 1 to n{ // n is number of method
            if(methodname[i]="main"){
                evaluateHeap(heaparea expression);
                evaluateStack(stackarea expression);
            }
        }
}
```

รูปที่ 3.28 Algorithm ของการ Evaluate

ผลลัพธ์ที่ได้จะเป็นผลจากการ Evaluate RDF file นี้

Concept สำหรับการ Evaluate Heaparea Expression และ Stackarea Expression

1. หา Expression ย่อยทั้งหมดใน Expression ของ Heaparea หรือ Stackarea นั้น ซึ่งประกอบด้วย
   - Array Expression เช่น [I[10] ,[D[argument[0]] ซึ่งจะเป็น Expression ย่อยใน Heaparea Expression
   - Instance Expression เช่น (class).instance.fieldsum ซึ่งจะเป็น Expression ย่อยในHeaparea Expression
   - Method Expression เช่น (class).instance.a(I:5).mresource.heaparea
2. ถ้า Parameter อยู่ในรูปแบบ Expression จะต้อง Evaluate Parameter ก่อน
3. Evaluate Expression ย่อยแต่ละตัว
4. สำหรับ Heaparea ค่าที่ได้เป็นหน่วย Byte
5. สำหรับ Stackarea ค่าที่ได้เป็นหน่วย Word

Algorithm ของการ Evaluate Expression

```
evaluate(expression){
        changeExpression(expression);
        getArgument();
        MathEvaluator mh = new MathEvaluater(expression);
        while(length of expression > 0){
                if(found array expression) then
                        evaluateArrayLength();
                        double da = getArrayValue();
                        mh.addVariable(arrayexpression,di);
                else if(found instance expression) then
                        getInstanceDescription();
                        Resource obj2 = readXML(obj2class,obj2);
                        getInstanceValue();
                        double di = getInstanceFunction ();
                        mh.addVariable(instanceexpression,di);
                else if(found method expression) then
                        double dm = getMethodValue(methodexpression);
                endif
                substringexpression();
        }
        double result = mh.getValue();
}
double getMethodValue(methodexpression){
        getParameterValue();
        Resource obj3 = readXML(obj3class,obj3);
        String s = getMethodInvoked();
        double dmi = evaluate(s);
        return dmi;
}
```

รูปที่ 3. 29 Algorithm ของการ Evaluate Expression

ยกตัวอย่างเช่น มี Heaparea Expression

max(([I[argument[0]+1]+

(heaprecursive).instance.hr(I:argument[0]-1;).mresource.heaparea),(0))

ซึ่งประกอบด้วย Expression ย่อย 2 ตัว คือ

- [I[argument[0]+1] ซึ่งในการ Evaluate จะมีขั้นตอน คือ

1. Evaluate ความยาวของ Array ซึ่งในที่นี้คือ argument[0]+1 และสมมติให้ argument[0] เป็น 9 ดังนั้น ความยาวของ Array นี้เป็น 10
2. Evaluate ค่าของ Array ซึ่งเป็น ชนิด Integer มีความยาว เท่ากับ 10 ซึ่งจะได้เท่ากับ 56

- (heaprecursive).instance.hr(I:argument[0]-1;).mresource.heaparea) ซึ่งในการ Evaluate จะมีขั้นตอน คือ

1. Evaluate ค่าของ Parameter ซึ่งในที่นี้ คือ argument[0] -1 และจะได้เท่ากับ 8
2. หา ชื่อ Class ของ Expression นี้ ซึ่ง คือ heaprecursive
3. ถ้า class นั้นไม่ใช่ Class ปัจจุบัน Read RDF ของ Class heaprecursive มาเก็บใน RO
4. หาค่า Heaparea ของ Method hr โดยมี Parameter ที่ถูกส่งเข้า Method เท่ากับ 8
5. ถ้า Heaparea อยู่ในรูปแบบของ Expression Evaluate ต่อ
6. สมมติเท่ากับ 5

ดังนั้นจะได้ค่าของ Heaparea Expresstion นี้หลังจาก Evaluate แล้ว คือ 56 + 5 = 61

และจากสมมติมี Stackarea Expression

max((heaprecursive).instance.hr(I:argument[0]-1;).mresource.stackarea) + 2 + 2

ซึ่งมีขั้นตอนของการ Evaluate คือ

1. Evaluate ค่าของ Parameter ซึ่งในที่นี้ คือ argument[0]-1 และจะได้เท่ากับ 8
2. หา ชื่อ Class ของ Expression นี้ ซึ่ง คือ heaprecursive
3. ถ้า Class ไม่ใช่ Class ปัจจุบัน Read RDF ของ Class heaprecursive มาเก็บใน RO
4. หาค่า Stackarea ของ Method hr โดยมี Parameter ที่ถูกส่งเข้า Method เท่ากับ 8
5. ถ้า Stackarea อยู่ในรูปแบบของ Expression Evaluate ต่อ
6. สมมติเท่ากับ 5

ดังนั้นจะได้ค่าของ Stackarea Function นี้หลังจาก Evaluate แล้ว คือ 5+2+2 = 9 word

จากสมมติฐานที่กล่าวมาข้างต้น ผู้วิจัยได้ทำการ Implement Framework และ Algorithm ที่นำเสนอนี้ และจะกล่าวถึงผลการทดลองในบทที่ 4 ต่อไป

บทที่ 4

การทดสอบการใช้ Heap และ Stack ของ Java Objects

งานวิจัยนี้เป็นการนำเสนอวิธีการวัดการใช้ทรัพยากรของ Java Object โดยการใช้ข้อมูลจาก Java Class File ตาม Framework และ Alogorithm ที่นำเสนอและนำผลที่ได้มาทดสอบตามประเภทของทรัพยากรของการวัด

โดยการทดลองนี้ใช้ Compiler คือ Java 2 SDK Standard Edition (jdk1.3.0_02) และ Run บน JAVA 2 Runtime Environment บนระบบปฏิบัติการ Windows โดยการทดลองจะประกอบด้วย

1. การทดลองที่ 1 เป็นการทดลองการใช้พื้นที่ Heaparea ของ Object และ Array
2. การทดลองที่ 2 เป็นการทดลองการใช้พื้นที่ Heaparea ของการการ Invoke Method
3. การทดลองที่ 3 เป็นการทดลองการใช้พื้นที่ของ Operand Stack และ Local Variable
4. การทดลองที่ 4 เป็นการทดลองการใช้ Stack ในการทำงานแบบ Recursive
5. การทดลองที่ 5 เป็นการทดลองการใช้เวลาในการ Compile และ Evaluate

4.1 การทดลองที่ 1

เป็นการทดลองเพื่อศึกษาถึงพฤติกรรมของการใช้ Heaparea ของ Object และ Array เนื่องจากการใช้ Heaparea เกิดจาก 3 ส่วนต่อไปนี้ คือ

1. การสร้าง Object
2. การสร้าง Array
3. การ Invoke ไปยัง Method อื่น ซึ่ง Method นั้นอาจจะมีการใช้พื้นที่ของทั้ง 3 ส่วนนี้

สำหรับการทดลองในส่วนนี้จะเป็นการทดลองการใช้พื้นที่ของ Object และ Array โดยกำหนดการทดลองดังนี้

4.1.1 การทดลองการใช้พื้นที่ Heaparea ของ Object

สร้าง Object จาก Class ที่มี Variable เป็น Type ต่าง ๆ ทั้ง 9 ชนิด (Class ละ Type) และเพิ่มจำนวน Variable ครั้งละ 1 Variable โดยกำหนด Class ดังนี้

Declare Variable ในลักษณะ

<Type> $var_1,var_2,var_3,\ldots,var_n$

เมื่อ

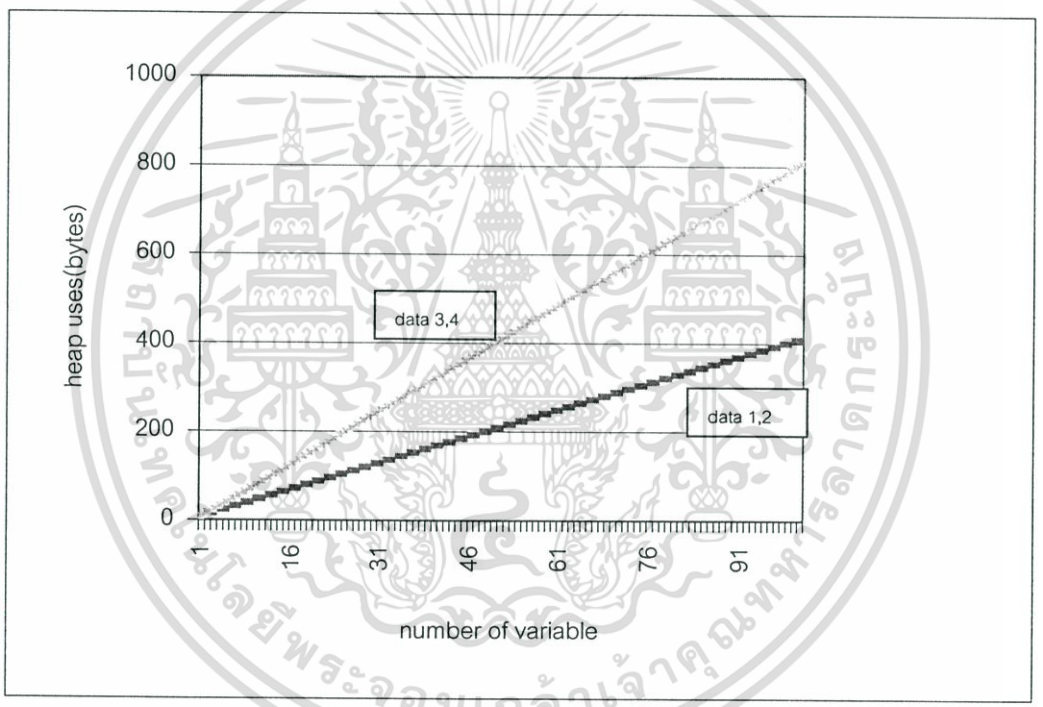
Type เป็น Type ของ Variable ซึ่งประกอบด้วย Boolean , Byte , Short , Char , Int , Float , Long , Double และ Reference

และ n เป็นจำนวน Variable = 100

ตัวอย่าง Class

```
class Test{
        int i1,i2,i3;
}
```

และทดสอบผลกับ Runtime Tool ของ Vladimir Roubtsov ("Java Tip 130 : Do you know your data size"., http :// www.javaworld.com/javaworld/javatips/jw-javatip130.html) ตามภาคผนวก ข ใน Code ที่ 1 ซึ่งจะเป็นการวัดว่า Object แต่ละตัวจะใช้พื้นที่เป็นเท่าไหร่ โดยใช้ Method freememory ของ Class java.lang.Runtime เพื่อหา Freememory ก่อนและหลังการสร้าง Object และนำมาหาค่าความต่างของการใช้ Memory ในการสร้าง Object นั้น ซึ่งผลที่ได้ คือ



รูปที่ 4.1 กราฟแสดงการใช้ Heaparea ของ Object

โดย

data 1 : ผลลัพธ์จาก Runtime Approach ของ Object ของ Class ที่ประกอบVariable Type boolean , byte , short , char ,int , float และ reference

data 2 : ผลลัพธ์จาก Propose Approach ของ Object ของ Class ที่ประกอบด้วย Variable Type boolean , byte , short , char ,int , float และ reference
ซึ่งผลการทดลองจะได้ผลลัพธ์ที่ตรงกัน

data 3 : ผลลัพธ์จาก Runtime Approach ของ Object ของ Class ที่ประกอบด้วย Variable Type long และ double

data 4 : ผลลัพธ์จาก Propose Approach ของ Object ของ Class ที่ประกอบด้วย Variable Type long และ double
ซึ่งผลการทดลองจะได้ผลลัพธ์ที่ตรงกัน

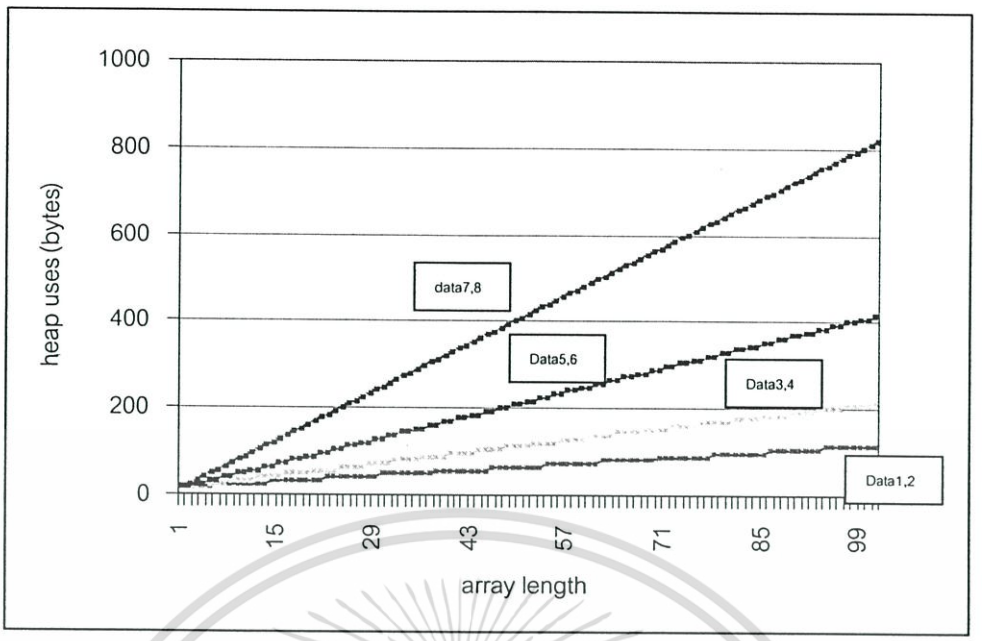
จากกราฟพบว่าเมื่อมีการสร้าง Object จาก Class ซึ่งกำหนด Variable เป็นชนิดและจำนวนต่าง ๆ สรุปได้ว่า

1. มี Overhead เริ่มต้น 8 Bytes
2. การ Allocate จะมีลักษณะ 8-Bytes Block คือ จะ Allocate ครั้งละ 8 Bytes ถึงแม้ว่า Variable นั้นจะไม่ได้มีขนาดเท่ากับ 8 Bytes ก็ตาม
3. การ Allocate สำหรับ Variable ที่มี Primitive Type น้อยกว่า 8 จะใช้พื้นที่ 2 Variable ต่อ 8 Byte ส่วน Type Long และ Double จะ Allocate ตามจำนวนที่มีการ Declare จริง
4. การวัดจะวัดเฉพาะ Variable ที่ไม่มี Modifier เป็น Static เท่านั้น (วัดเฉพาะ Instance Variable ซึ่งจะต้องมีการ Allocate พื้นที่ให้ทุกครั้งที่สร้าง Instance ของ Class)
5. ผลลัพธ์จากวิธีการที่นำเสนอตรงกับการใช้ Runtime Tool

4.1.2 การทดลองการใช้พื้นที่ Heaparea ของ Array

1. สร้าง Array ขนาด 1 มิติของ Type ทั้ง 9 ชนิด และเพิ่มความยาวครั้งละ 1
2. สร้าง Array หลายมิติโดยกำหนด Type และจำนวนมิติต่าง ๆ

และเปรียบเทียบผลที่ได้จากวิธีการที่นำเสนอกับ Runtime Tool เช่นเดียวกับการทดลองการใช้พื้นที่ของ Object ซึ่งผลการทดลองสำหรับ Array 1 มิติผลลัพธ์ที่ได้แสดงในรูปที่ 4.2



รูปที่ 4.2 กราฟแสดงการใช้ Heaparea ของ Array 1 มิติ

โดย

data 1 : ผลลัพธ์จาก Runtime Approach ของ Array Type boolean และ byte

data 2 : ผลลัพธ์จาก Propose Approach ของ Array Type boolean และ byte

ซึ่งผลการทดลองจะได้ผลลัพธ์ตรงกัน

data 3 : ผลลัพธ์จาก Runtime Approach ของ Array Type short และ char

data 4 : ผลลัพธ์จาก Propose Approach ของ Array Type short และ char

ซึ่งผลการทดลองจะได้ผลลัพธ์ตรงกัน

data 5 : ผลลัพธ์จาก Runtime Approach ของ Array Type int , float และ ref

data 6 : ผลลัพธ์จาก Propose Approach ของ Array Type int , float และ ref

ซึ่งผลการทดลองจะได้ผลลัพธ์ตรงกัน

data 7 : ผลลัพธ์จาก Runtime Approach ของ Array Type long และ double

data 8 : ผลลัพธ์จาก Propose Approach ของ Array Type long และ double

ซึ่งผลการทดลองจะได้ผลลัพธ์ตรงกัน

จากกราฟพบว่าเมื่อมีการสร้าง Array ชนิดและความยาวต่าง ๆ สรุปได้ว่า

1. มี Overhead เริ่มต้นที่ 16 Bytes

2. การ Allocate จะมีลักษณะ 8-Bytes Block คือ จะ Allocate ครั้งละ 8 Bytes ถึงแม้ว่า Type ของ Array นั้นจะไม่ได้มีขนาดเท่ากับ 8 Bytes ก็ตาม เช่นเดียวกับ Object

3. หลังจาก Allocate 8 Bytes แล้วเมื่อมีการเพิ่ม Element จะใช้พื้นที่ตามขนาดของ Primitive Type

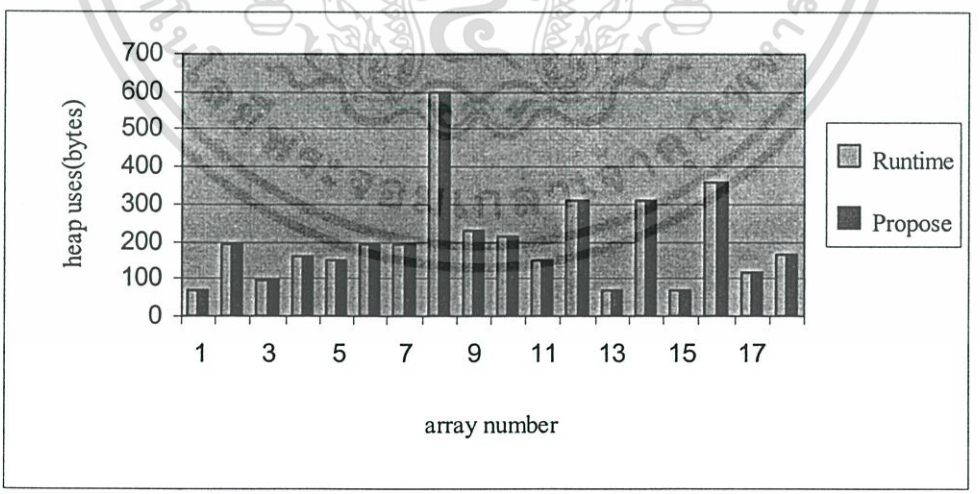
4. ผลลัพธ์จากวิธีการที่นำเสนอตรงกับการใช้ Runtime Tool

ตารางที่ 4.1 การใช้พื้นที่ของ Array Type ต่าง ๆ

Array Type	Type	Size
Z	Boolean	1 byte
B	Byte	1 byte
C	Char	2 bytes
S	Short	2 bytes
I	Int	4 bytes
J	Long	8 bytes
F	Float	4 bytes
D	Double	8 bytes
L..	reference	4 bytes

เช่น Array Type เป็น Char เมื่อมีการ Allocate 8 Bytes แล้วจะต้องเพิ่ม Element อีก 3 Element (2 * 4 = 8) จึงจะ Allocate อีก 8 Bytes เป็นต้น

ผลการทดลองการใช้ Heaparea ของ Array หลายมิติ



รูปที่ 4.3 กราฟแสดงการใช้ Heaparea ของ Array หลายมิติ

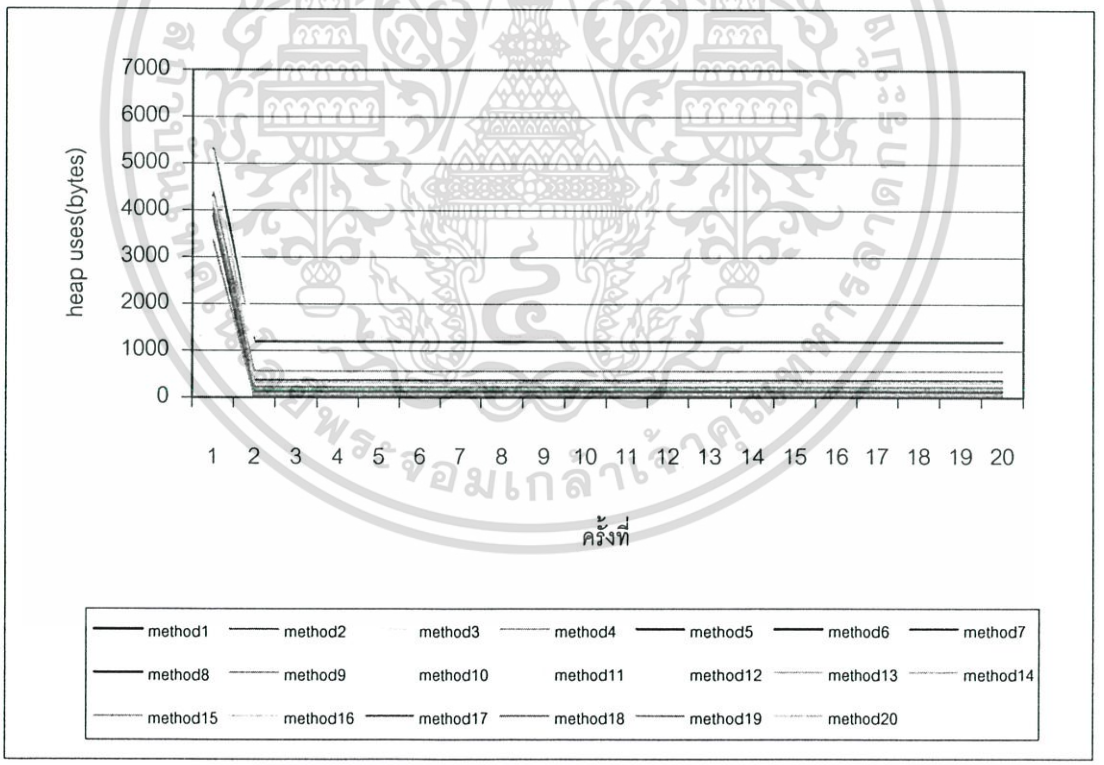
ซึ่งผลลัพธ์จากวิธีการที่นำเสนอจะตรงกับ Runtime Tool

4.2 การทดลองที่ 2

สำหรับการใช้พื้นที่ของ Heaparea สำหรับ Method ซึ่งเป็นการนำเสนอในงานวิจัยนี้จะทำการทดลองโดยการกำหนด Class ขึ้นมาและวัดว่า Method แต่ละ Method นั้นมีการใช้พื้นที่เป็นเท่าไร โดยการทดลองจะมีรายละเอียดดังนี้

1. วัดการใช้พื้นที่ของ Heaparea ของ Method
2. เพื่อศึกษาถึงลักษณะการใช้พื้นที่ Heaparea ในการ Invoke Method (เริ่มจาก main)
3. Method ที่วัดจำนวน 20 Method
4. ทดลองโดยทำการเรียกแต่ละ Method จำนวน 20 ครั้ง
5. เปรียบเทียบผลจากวิธีการที่นำเสนอกับผลจากการใช้พื้นที่ที่เกิดจากการ Run จริง โดยการใช้ Method freememory ของ Class java.lang.Runtime เพื่อทดสอบว่า freememory ก่อนและหลังจากการ Invoke Method นั้นมีความแตกต่างกันอย่างไร (ดูCode ที่ 2 ตามภาคผนวก ข)

ซึ่งจะได้ผลการทดลองการ Invoke Method ดังกราฟ



รูปที่ 4.4 กราฟแสดงการใช้พื้นที่ของ Heaparea ในการ Invoke Method

โดยการทดลองจะเป็นการ Invoke Method แต่ละ Method จำนวน 20 ครั้ง 20 Method จากกราฟจะเห็นได้ว่า ในการเรียกครั้งแรกจะมี Overhead ซึ่งอาจจะเกิดจากกระบวนการก่อนการทำงานจริงของ Method เช่น การ Load Class หรือ การ Initialize Class แต่ผลการทดลองตั้งแต่ครั้งที่ 2 เป็นต้นไปจะได้ค่าที่เท่ากัน

ดังนั้นเมื่อวัดการใช้พื้นที่ในการทำงานของ Method เปรียบเทียบวิธีการที่นำเสนอกับผลจากการ Run จริง(ตั้งแต่ครั้งที่ 2 เป็นต้นไป ซึ่งเป็นการใช้พื้นที่ในการทำงานของ Method ที่ไม่เกี่ยวกับการทำงานของ System) จะได้ผลลัพธ์

การทดลองในขั้นต่อไปเป็นการเปรียบเทียบผลลัพธ์จากวิธีการที่นำเสนอกับผลที่ได้จากการใช้ freeMemory ในครั้งที่ 2 โดย Method ที่ใช้ทดสอบประกอบด้วย

1. Method ที่มีลักษณะทางเลือก จำนวน 31 Method
2. Method ที่ไม่มีลักษณะทางเลือก จำนวน 134 Method ซึ่งจะได้ผลการทดลอง

ผลการทดลองกรณี Method มีทางเลือก

ตารางที่ 4.2 ผลการทดลองการใช้ Heaparea ของ Method ที่มีลักษณะทางเลือก

method	Runtime	Propose	ความแตกต่าง	%ความแตกต่าง
1	72	72	0	0
2	112	112	0	0
3	32	32	0	0
4	112	112	0	0
5	72	72	0	0
6	216	288	72	33.33
7	160	160	0	0
8	32	32	0	0
9	112	112	0	0
10	32	32	0	0
11	32	32	0	0
12	480	480	0	0
13	336	336	0	0
14	32	56	24	75
15	112	112	0	0
16	0	0	0	0
17	56	56	0	0

ตารางที่ 4.2 (ต่อ)

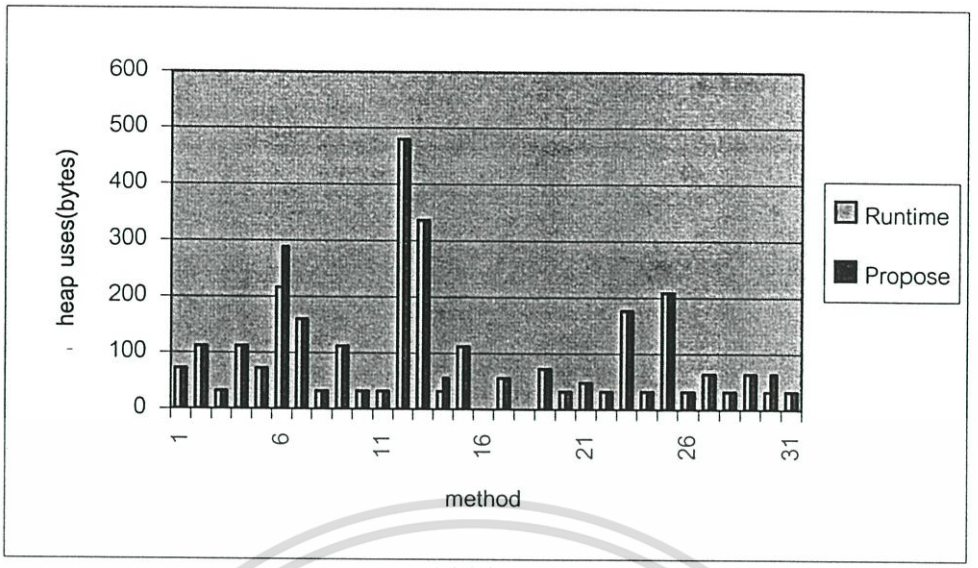
method	Runtime	Propose	ความแตกต่าง	%ความแตกต่าง
18	0	0	0	0
19	72	72	0	0
20	32	32	0	0
21	48	48	0	0
22	32	32	0	0
23	176	176	0	0
24	32	32	0	0
25	208	208	0	0
26	32	32	0	0
27	64	64	0	0
28	32	32	0	0
29	64	64	0	0
30	32	64	32	100
31	32	32	0	0

ซึ่งจะเห็นได้ว่ามีบาง Method ที่ค่าที่ได้จากวิธีการที่นำเสนอต่างจากผลจาก Runtime ซึ่งสามารถสรุปความผิดพลาดได้ดังตาราง

ตารางที่ 4.3 เปอร์เซนต์ความผิดพลาดในการวัดการใช้ Heaparea ของ Method ที่มีลักษณะทางเลือก

จำนวน method	%ความผิดพลาด
31	6.72

และผลการทดลองสามารถแสดงได้ดังกราฟ



รูปที่ 4.5 กราฟแสดงการใช้พื้นที่ของ Heaparea ในการ Invoke Method ที่มีทางเลือก

ซึ่งจากผลการทดลองสามารถสรุปผลได้ดังนี้

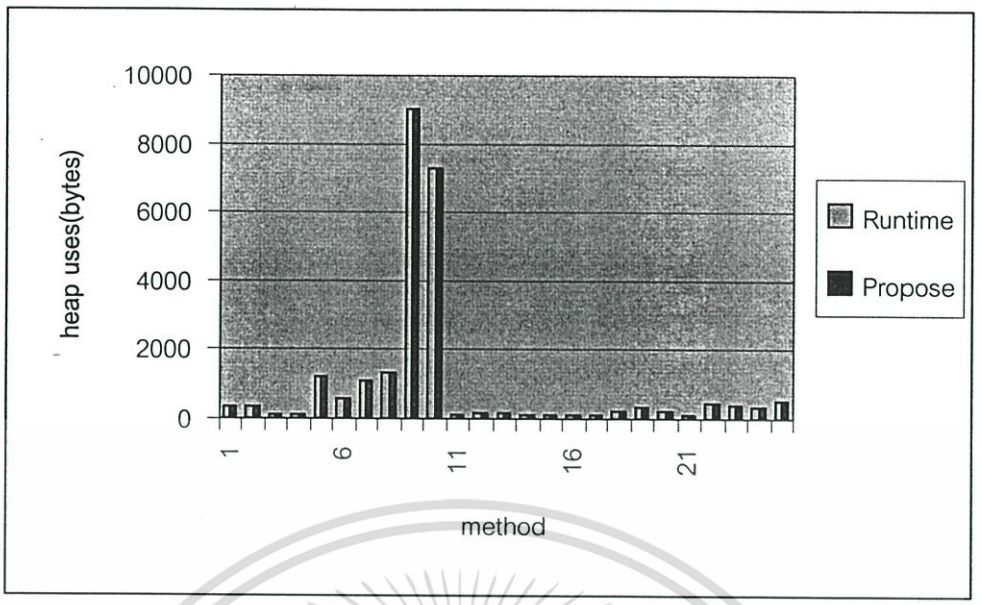
1. ในกรณีที่ได้ผลลัพธ์ไม่เท่ากันเกิดจากทางเลือกที่มีการใช้ Heaparea สูงสุดเมื่อเปรียบเทียบในทุกทางเลือกของ Method
2. เมื่อทดลอง Run ในทุกกรณีจะไม่เกินค่า Maximum ที่หาได้

ผลการทดลองกรณี Method ไม่มีทางเลือก

ตารางที่ 4.4 สรุปผลความถูกต้องในการวัดการใช้พื้นที่ Heaparea ของ Method ที่ไม่มีทางเลือก

	จำนวน METHOD ทดสอบ	ตรง	ไม่ตรง	% ความถูกต้อง
Heaparea	132	132	0	100%

และผลลัพธ์ของ Method จำนวน 25 Method ได้ดังกราฟ



รูปที่ 4.6 กราฟแสดงการใช้พื้นที่ของ Heaparea ในการ Invoke Method ที่ไม่มีทางเลือก

โดยการผลทดลองนี้เป็นการทดลอง Invoke Method main ของ Class ตาม Class ตัวอย่างในภาคผนวก ค

4.3 การทดลองที่ 3

สำหรับ Operand Stack และ Local Variable จะทดลองวิธีการที่นำเสนอกับการใช้ คำสั่ง javap –v ซึ่งเป็นคำสั่งที่ใช้แสดงข้อมูลของการใช้ Operand Stack และ Local Variable ของ Class ยกตัวอย่างเช่น

% javap –v heaprecursive

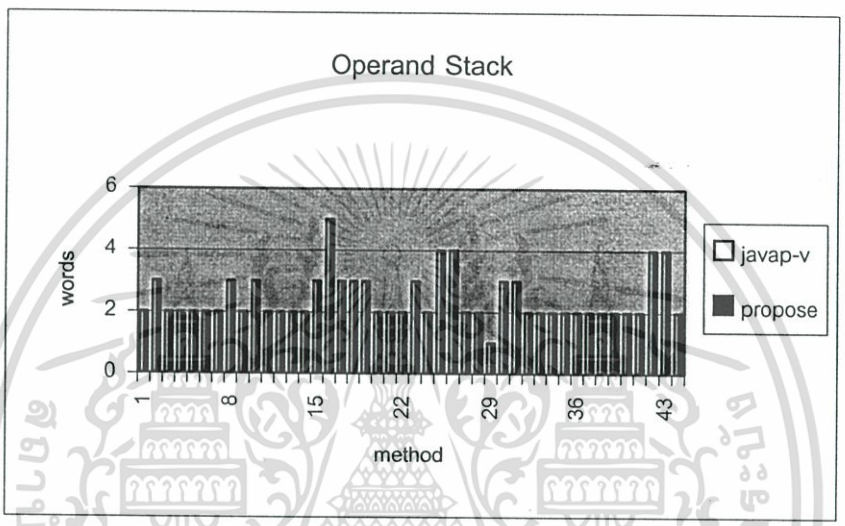
ซึ่งจะได้ผลที่แสดง คือ

```
Compiled from heaprecursive.java
class heaprecursive extends java.lang.Object
    heaprecursive();
        /* Stack=1, Locals=1, Args_size=1 */
    public static void main(java.lang.String[]);
        /* Stack=1, Locals=2, Args_size=1 */
    public static int hr(int);
        /* Stack=2, Locals=2, Args_size=1 */
```

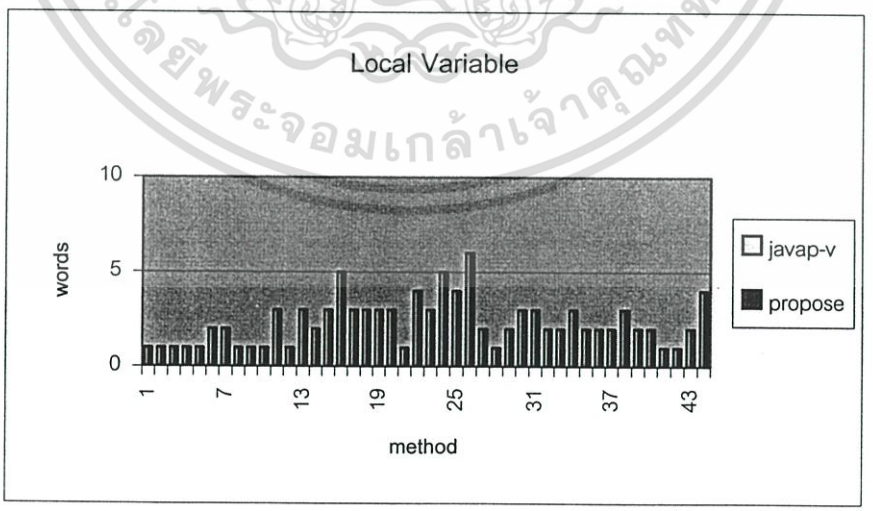
ซึ่งผลการทดลองจะได้ ดังตาราง 4.5 และรูป 4.7 และ 4.8

ตารางที่ 4.5 สรุปผลการวัด Operand Stack และ Local Variable

	จำนวน METHOD ทดสอบ	ตรง	ไม่ตรง	% ความถูกต้อง
Operand stack	364	364	0	100%
Local variable	364	364	0	100%



รูปที่ 4.7 กราฟแสดงผลการวัด operand stack ของวิธีการที่นำเสนอกับการใช้ javap –v



รูปที่ 4.8 กราฟแสดงผลการวัด local variable ของวิธีการที่นำเสนอกับการใช้ javap –v

4.4 การทดลองที่ 4

สำหรับการทดลองนี้จะเป็นการทดลองการใช้พื้นที่ของ Java Method Stack เพื่อวิเคราะห์หาจำนวนรอบของการทำงานของ Method ที่มีลักษณะ Recursive ที่คาดว่าจะทำงานได้จากวิธีการนำเสนอเปรียบเทียบกับค่าขนาดของ default stack ซึ่งเท่ากับ 400 kilobytes โดยการทดลองจะใช้เป็นการกำหนด Class ที่มีการทำงานในลักษณะ Recursive คือ

```
class testforfact{
        public static void main(String args[]){
                int x = Integer.parseInt(args[0]);
                x = fac(x);
        }
        public static int fac(int x){
                if(x==0){
                        return 0;
                }else{
                        return fac(x+1);
                }
        }
}
```

รูปที่ 4.9 Class ที่ใช้ในการทดลองที่ 4

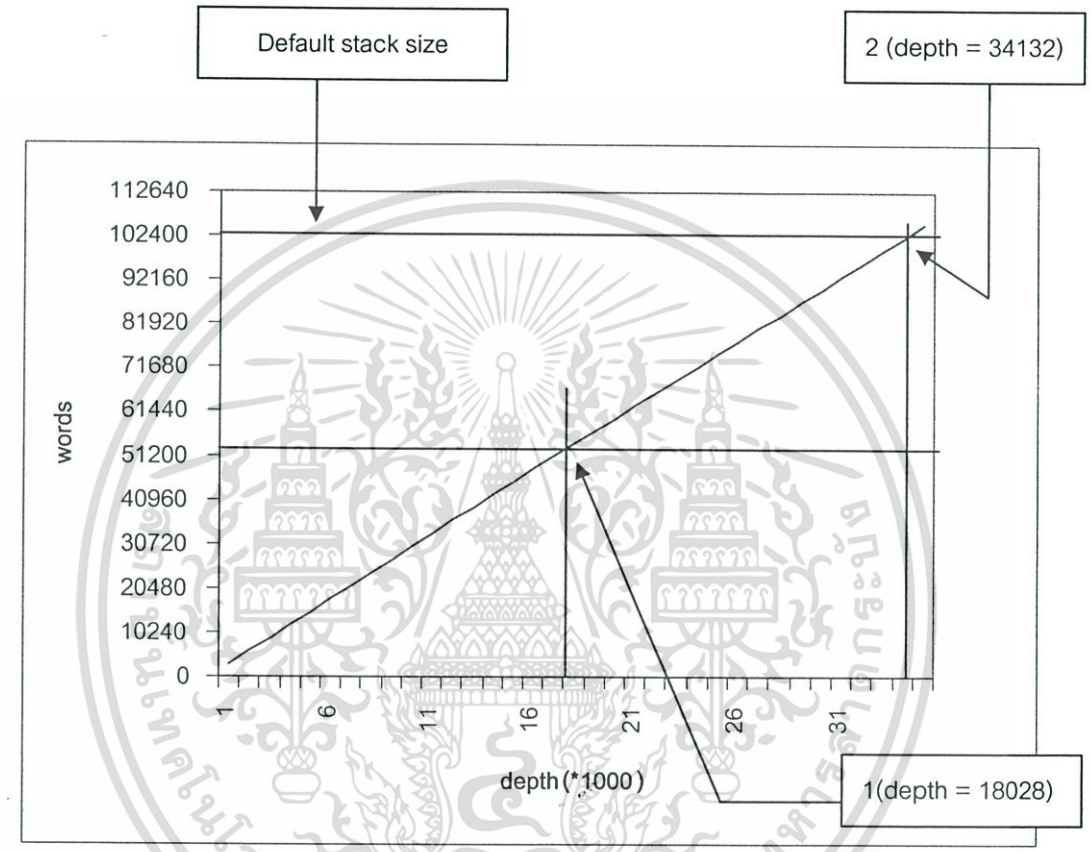
และมีค่าของ Operand Stack และ Local Variable เป็น

รูปที่ 4.10 Bytecode Instruction ของ Class ที่ใช้ในการทดลองที่ 4

และ Expression ของ Stackarea ของ Method fac คือ

```
<stackarea>max((testforfact).instance.fac(I:argument[0]+1;).mresource.stackarea) + 2
+ 1</stackarea>
```

โดยการทดลองจะป้อนค่าตัวแปรเพื่อการ Recursive และ เพิ่มค่าตัวแปรไปเรื่อย ๆ จนกว่าจะเกิด StackOverflow ซึ่งจะได้ผลการทดลองดังรูป



รูปที่ 4.11 กราฟแสดงการใช้ Stack ของ Class ที่มีลักษณะ Recursive

จากรูปเป็นกราฟแสดงการใช้ Stack ของ Class ที่มีลักษณะ Recursive โดยการทดลองจะเป็นการเปรียบเทียบเมื่อใช้การทำนายตาม Algorithm ที่นำเสนอ โดยจะเป็นการบวกค่าของ Operand Stack และ Local Variable ตามจำนวนของการ Recursive (เนื่องจาก Invoke แต่ละครั้งจะมีการ Allocate พื้นที่ของ Stack Frame) โดยค่า Default ของพื้นที่ Stack ที่สามารถใช้ได้คือ 400 kilobyte[16] ดังนั้นเมื่อใช้ Algorithm ตามที่นำเสนอจะพบว่าจะสามารถ Recursive ได้เมื่อ depth เท่ากับ 34132 (จุดตัดที่ 2) ในขณะที่การ Run ตามปกติจะ Recursive ได้เพียง depth เท่ากับ 18028 เท่านั้น (จุดตัดที่ 1) ก็จะเกิด StackOverflow จะเห็นได้ว่าเนื่องจาก Stack มีการใช้ส่วนของ Data Frame(ซึ่งไม่ได้วัดในงานวิจัยนี้) จึงทำให้ผลลัพธ์ไม่ตรงกัน

4.5 การทดลองที่ 5

การทดลองนี้เป็นการทดลองเพื่อหาว่าเวลาที่ใช้ในการ Compile และ Evaluate เพื่อหาว่ามีปัจจัยใดบ้างที่มีผลต่อการใช้เวลาในการ Compile และ Evaluate โดยการทดลองจะทดลองกับ Class และ Method ที่ใช้ในการทดลองที่ 2 และ 3 และเวลาที่ได้จากทดลองนี้เกิดจากการดึงเวลาของระบบโดยใช้ Method currentTimeMillis จาก Class java.lang.System และทำการเปรียบเทียบความแตกต่างของเวลาที่ได้จากก่อนและหลังการ Compile และ Evaluate ดังรูป

```
long start = System.currentTimeMillis();
Compile / Evaluate;
long stop = System.currentTimeMillis();
time = stop – start;
```

รูปที่ 4.12 การใช้คำสั่งเพื่อหาเวลาในการ Compile และ Evaluate

4.5.1 การ Compile

การทดลองเพื่อหาเวลาในการ Compile Class ต่าง ๆ จะทดลองกับ Class ทั้งหมด 132 Class แต่เนื่องจากแต่ละ Class มีลักษณะการทำงานที่ต่างกันรวมถึงจำนวน bytcode ต่างกัน ดังนั้นในส่วนแรกนี้จะเป็นผลการทดลองของ Class จำนวน 10 Class ซึ่งผลที่ได้ดังตาราง

ตารางที่ 4.6 ผลการทดลองการใช้เวลาในการ Compile ของ Class 10 Class

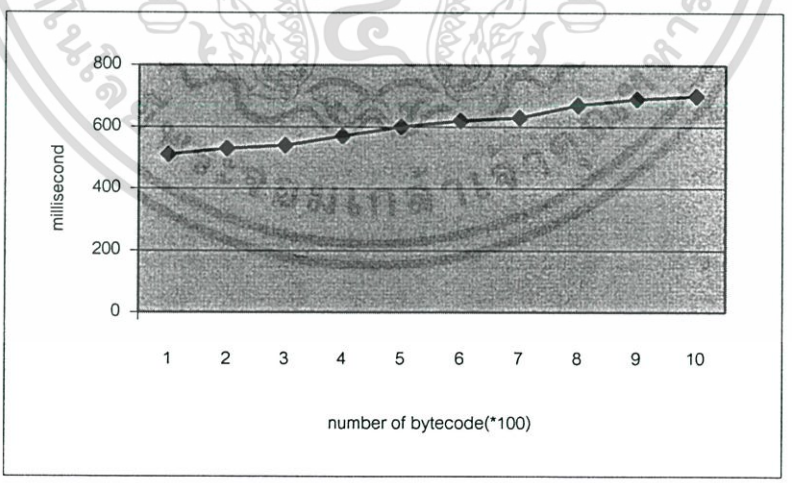
Class	เวลา (milli second)	จำนวน bytecode	จำนวน invokemethod bytecode	จำนวน array bytecode	จำนวน new bytecode
1	500	26	0	0	0
2	500	30	0	0	0
3	680	25	8	0	0
4	520	29	2	0	0
5	510	22	3	0	0
6	520	22	4	0	4
7	530	23	2	0	2
8	520	28	3	2	3
9	540	24	6	1	1
10	510	59	0	0	0

จากตารางจะพบว่าเมื่อจำนวน Bytecode ใกล้เคียงกัน(Class 1 – 9) จะพบว่าเมื่อ Class มี Bytecode ที่เกี่ยวกับการ Invoke จะมีการใช้เวลาในการทำงานเพิ่มขึ้นอย่างชัดเจน ส่วนการใช้คำสั่งสำหรับการสร้าง Object และ Array จะมีผลต่อการใช้เวลาไม่มากนัก เนื่องจาก Algorithm ในการ Compile ที่นำเสนอเมื่อมีการพบ Bytecode ที่เป็นการ Invokemethod , การสร้าง object และการสร้าง Array จะต้องมีการทำงานเพื่อหารายละเอียดของคำสั่งนั้นเพื่อ Append ลงใน Expression ของทั้ง Heap และ Stack ในขณะที่ Class ที่ 1 , 2 และ 10 ซึ่งเป็น Class ที่ไม่มีการใช้คำสั่งทั้ง 3 ประเภทนี้เมื่อมีการเพิ่มจำนวนของ Bytecode จะมีผลต่อการใช้เวลาค่อนข้างน้อย

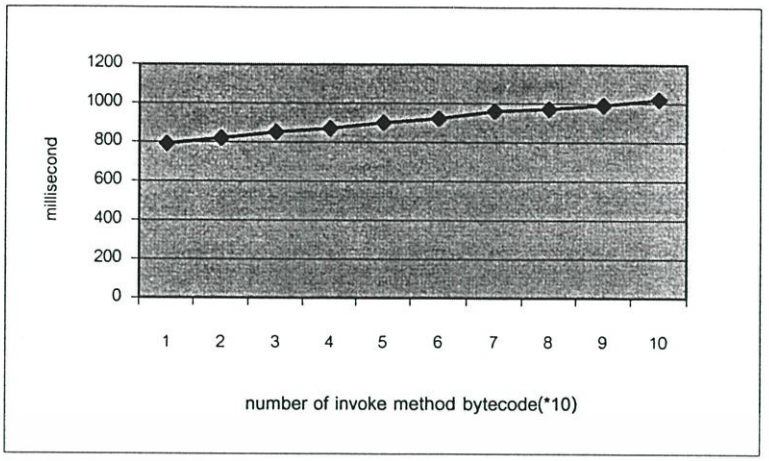
ดังนั้นในการทดลองในขั้นตอนต่อไปจะเป็นการทดลองการใช้คำสั่งของ Bytecode ทั้ง 3 ประเภทและประเภทอื่น ๆ เพื่อหาว่ามีผลกระทบต่อการใช้เวลาในการ Compile อย่างไร โดยในการทดลองเป็นการสร้าง Class ขึ้นมา 1 Class และเพิ่มจำนวน Bytecode ในแต่ละประเภทขึ้นเรื่อย ๆ ซึ่งจะประกอบด้วย

1. เป็นการทดลองโดยเพิ่มจำนวนของ Bytecode ประเภทอื่น ๆ
2. เป็นการทดลองโดยเพิ่มจำนวนของ Bytecode Invoke Method
3. เป็นการทดลองโดยเพิ่มจำนวนของ Bytecode new
4. เป็นการทดลองโดยเพิ่มจำนวนของ Bytecode array

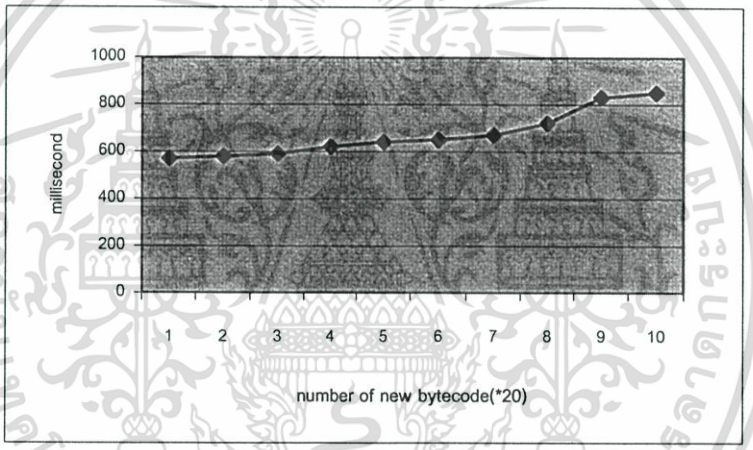
ซึ่งผลการทดลองเป็นดังรูปที่ 4.13 , 4.14 , 4.15 และ 4.16



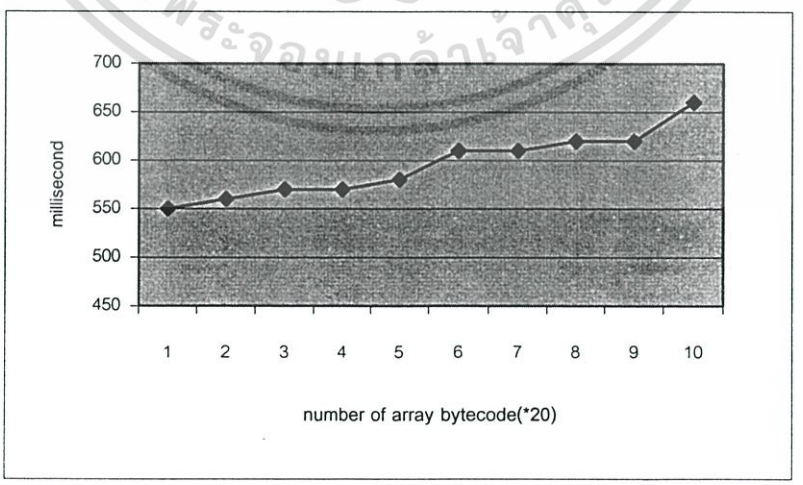
รูปที่ 4.13 กราฟแสดงการใช้เวลาในการ Compile เมื่อเพิ่มจำนวน Bytecode ทั่วไป



รูปที่ 4.14 กราฟแสดงการใช้เวลาในการ Compile เมื่อเพิ่มจำนวน Bytecode Invokemethod



รูปที่ 4.15 กราฟแสดงการใช้เวลาในการ Compile เมื่อเพิ่มจำนวน Bytecode New



รูปที่ 4.16 กราฟแสดงการใช้เวลาในการ Compile เมื่อเพิ่มจำนวน Bytecode Newarray

จากกราฟสามารถสรุปผลได้ดังนี้

1. เมื่อจำนวน Bytecode เพิ่มขึ้นจะมีการใช้เวลาในการ Compile Class เพิ่มขึ้น

2. เมื่อเพิ่มจำนวนของ Bytecode ที่เกี่ยวกับการ Invokemethod จะมีการใช้เวลาในการ Compile เพิ่มขึ้นอย่างชัดเจน ในขณะที่เมื่อมีการเพิ่ม Bytecode ที่เกี่ยวกับการสร้าง Object และ Array จะมีผลต่อการใช้เวลาน้อยกว่าการ Invokemethod

3. การใช้เวลาเพิ่มขึ้นจะเป็นการเพิ่มแบบไม่คงที่

4.5.2 การ Evaluate

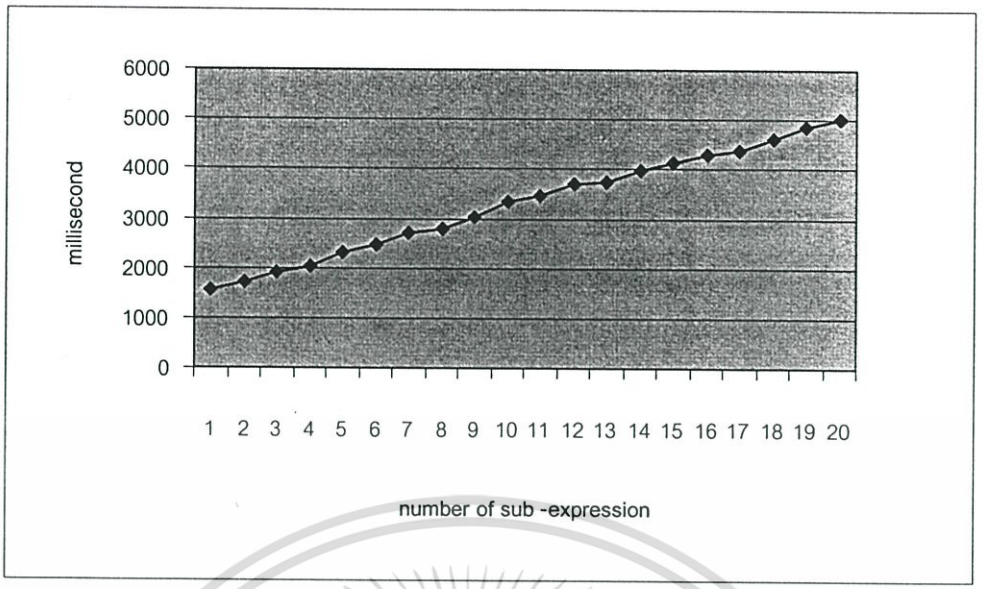
การทดลองการใช้เวลาสำหรับการ Evaluate ได้ทดลองกับ Method จำนวน 5 Method เพื่อเปรียบเทียบผล ซึ่งได้ผลการทดลองดังตาราง

ตารางที่ 4.7 ผลการทดลองการใช้เวลาในการ Evaluate ของ Method 5 Method

Method	เวลา (milli second)	จำนวน Expression ย่อย	จำนวน Expression invokemethod	จำนวน Expression newobject	จำนวน Expression array
1	1210	0	0	0	0
2	1530	1	1	0	0
3	1980	1	0	1	0
4	1270	1	0	0	1
5	2150	3	1	1	1

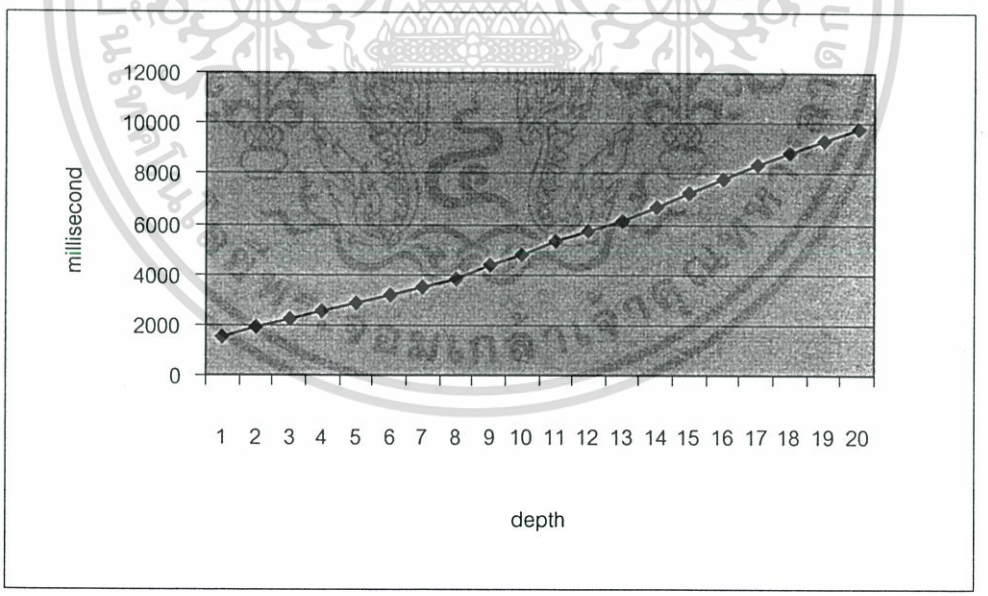
จากตารางพบว่าเวลาที่ใช้ในการ Evaluate ขึ้นอยู่กับ Expression ของ Heap และ Stack ซึ่งเมื่อค่าใน Expression เป็นค่าคงที่(ไม่มี Expression ย่อย) จะใช้เวลาในการ Evaluate น้อยกว่าในกรณีที่ Expression มี Expression ย่อย ซึ่งจะต้องทำการ Evaluate เพื่อหาค่าของแต่ละ Expression ย่อยนั้น

การทดลองต่อไปจะเป็นการทดลองโดยการเพิ่มจำนวนของ Expression ย่อยเพื่อหาแนวโน้มของการใช้เวลาในการ Evaluate ซึ่งผลการทดลองแสดงดังรูป



รูปที่ 4.17 กราฟแสดงการใช้เวลาในการ Evaluate เมื่อเพิ่มจำนวน Expression ย่อย

นอกจากนี้ในกรณีที่มีจำนวนของ Expression ย่อยเท่ากัน จำนวนความลึกของการ Evaluate ก็จะมีผลต่อการ Evaluate เมื่อ Method มีการ Evaluate ลึกลงไปเรื่อย ๆ ซึ่งผลการทดลองแสดงได้ดังรูป



รูปที่ 4.18 กราฟแสดงการใช้เวลาในการ Evaluate เมื่อเพิ่มจำนวนความลึกในการ Evaluate

จากผลการทดลองสรุปได้ว่า

1. เวลาที่ใช้ในการ Evaluate ขึ้นอยู่กับ Expression ของ Heap และ Stack Expression
2. เมื่อจำนวน Expression ย่อยเพิ่มขึ้นจะใช้เวลาในการ Evaluate เพิ่มขึ้น
3. เมื่อจำนวนความลึกในการ Evaluate เพิ่มขึ้นจะใช้เวลาในการ Evaluate เพิ่มขึ้น

4.6 วิเคราะห์ผลการทดลอง

4.6.1 Heaparea

จากผลการทดลองจะพบว่า

1. การใช้พื้นที่ของ Heaparea จะประกอบด้วย 3 ส่วน คือ การสร้าง Object การสร้าง Array และ การ Invoke Method(ซึ่งใน Method นั้นอาจจะประกอบด้วยทั้ง 3 ส่วนนี้เช่นกัน)

2. ในการสร้าง Object และ Array จะมีการใช้พื้นที่ของ Heaparea ตาม Type และ จำนวนหรือความยาวของ Array

3. ในการทำงานจริงเมื่อมีการ Invoke Method ไปยัง Class อื่นจะต้องมีการ Initialize Class นั้นจึงทำให้เกิด Overhead จำนวนหนึ่ง ดังนั้นการวัดในที่นี้จะเป็นการวัดโดยไม่รวมการใช้พื้นที่ของการ Initialize

4. ผลลัพธ์ที่ได้จากการวัดการใช้ Heaparea ของ Method (โดยวัดจากการ Run ครั้งที่ 2 เป็นต้นไป) ในส่วนของ Method ที่มีลักษณะ Branch จะให้ผลที่ไม่ถูกต้อง 100% แต่ถ้าเป็น Method ที่ไม่มีลักษณะ Branch จะให้ผลลัพธ์ที่ถูกต้อง 100%

4.6.2 Stackarea

จากผลการทดลองจะพบว่าทั้งในการวัดในส่วนของ Operand Stack และ Local Variable จากวิธีการที่นำเสนอให้ผลลัพธ์ที่ตรงกันกับการใช้ คำสั่ง javap –c ซึ่งเป็นการดูข้อมูลของ Method ภายใน Class ซึ่งในการทำงานจริง JVM จะ Allocate พื้นที่นี้เพื่อใช้ในการทำงานของ Method

4.6.3 การใช้เวลาในการ Compile และ Evaluate

จากผลการทดลองจะพบว่าเวลาในการ Compile จะขึ้นอยู่กับจำนวน Bytecode ของ Class File ที่ Compile นั้น และการ Evaluate จะขึ้นอยู่กับ Expression ของ Heap และ Stack Expression ซึ่งในการทดลองการเพิ่มขึ้นจะไม่เป็นแบบคงที่เนื่องจากทำงานกับคำสั่งแต่ละคำสั่งอาจจะรายละเอียดที่แตกต่างกัน

บทที่ 5
วิเคราะห์และสรุปผล

งานวิจัยนี้ได้นำเสนอวิธีการในการวัดและประเมินค่าการใช้ทรัพยากรของ Java Object ในส่วนของ Heaparea และ Stackarea เพื่อแก้ปัญหาที่เกิดจากการวัดการใช้ทรัพยากรในขณะ Runtime โดยแบ่งกระบวนการในการวัดเป็น 2 ขั้นตอนหลัก คือ

1. การ Compile เพื่อให้ได้ Resource Description File(RDF) ของ Class File เพื่อนำข้อมูลที่ได้จากการ RDF ไปใช้ในการวัดค่าการใช้ทรัพยากรต่อไป

2. การ Evaluate เป็นการนำข้อมูลที่ได้จาก RDF มาเข้าสู่กระบวนการเพื่อได้เป็นผลลัพธ์ในการใช้ทรัพยากร

ซึ่งจากวิธีการที่นำเสนอและผลการทดลองสามารถสรุปได้ดังนี้

5.1 วิเคราะห์ผลการวิจัย

จากผลการทดลองสรุปได้ว่า

1. Heaparea ซึ่งจะเป็นการทดลองที่เปรียบเทียบผลลัพธ์ที่ได้จาก Algorithm ที่นำเสนอ(จากการ Run ครั้งที่ 2) กับการใช้พื้นที่จริงขณะ Run พบว่า

- ในกรณีของ Class ที่ไม่มีการทำงานในลักษณะทางเลือกผลการทดลองจะถูกต้อง
- ในกรณีของ Class ที่มีการทำงานในลักษณะทางเลือกผลการทดลองจะมีความคลาดเคลื่อนเนื่องจากการวัดจะเป็นแบบ Maximum ซึ่งเมื่อมีการทดลองในทุกกรณีจะไม่มีการใช้ Heaparea เกินจากค่าที่วัดได้

การเปรียบเทียบผลต้องเปรียบเทียบกับผลกับ Runtime ครั้งที่ 2 เนื่องจากในการทำงานครั้งแรก System จะมีการใช้พื้นที่ในการเริ่มต้นการทำงานของ Class

2. Stackarea ในส่วนของ Operand Stack และ Local Variaable จะเปรียบเทียบผลลัพธ์ที่ได้จาก Algorithm ที่นำเสนอกับการใช้คำสั่ง javap ซึ่งผลการทดลองจะมีความถูกต้อง 100 เปอร์เซนต์ไม่ว่า Class File จะอยู่ในรูปแบบใดก็ตาม ส่วนการทดลองเพื่อวิเคราะห์หาจำนวนรอบของการ Recursive เพื่อหาการ StackOverflow จะยังมีความคลาดเคลื่อนเนื่องจากการใช้ Data Frame

5.2 สรุปผลการวิจัย

แนวคิดของการวัดการใช้ทรัพยากรสามารถนำใช้ในการจัดการทรัพยากรให้เกิดประสิทธิภาพ โดยงานวิจัยนี้เป็นการวัดการใช้ทรัพยากร 2 ประเภท คือ Heap และ Stack(Operand Stack และ Local Variable)โดยนำเสนอ Framework สำหรับวัดการใช้ทรัพยากรของ Java objects ที่มีรูปแบบตามที่กำหนด ซึ่ง Framework ประกอบด้วยการทำงาน 2 ขั้นตอน คือ

1. การ Compile เพื่อสร้าง Resource Description File(RDF)
2. การ Evaluate เพื่อประเมินค่าการใช้ทรัพยากร

ซึ่งจากผลการทดลองที่ได้จะพบว่าวิธีการที่นำเสนอสามารถสร้าง RDF File และ Evaluate RDF File นั้นได้ และงานวิจัยนี้นำเสนอ Algorithm ของการวัด (Algorithm รวมอยู่ใน Framework)

1. Heap ซึ่งจากผลการทดลอง

- กรณี Method ที่มีทางเลือกผลลัพธ์ที่ได้จะเป็นค่า maximum ของทุกทางเลือกและเมื่อทดลองกับการทุกกรณีแล้วมีการใช้ทรัพยากรไม่เกินจำนวนของ Maximum
- กรณี method ไม่มีทางเลือกผลลัพธ์จะถูกต้อง

โดยผลลัพธ์ที่ได้เปรียบเทียบกับการใช้ Method freeMemory ใน Class java.lang.Runtimeและในงานวิจัยนี้จะไม่วัดในส่วนของ Overhead ในการทำงานของ System ในการ Initial Class ในการ invoke method ครั้งแรก

2. Stack ซึ่งการวัดจะประกอบด้วย

- Operad Stack ให้ผลลัพธ์ที่ถูกต้อง
- Local Variable ให้ผลลัพธ์ที่ถูกต้อง

โดยผลลัพธ์ที่ได้เปรียบเทียบกับการใช้คำสั่ง javap –v และ ในงานวิจัยนี้ไม่ได้วัดในส่วนของ Data Frame

จากผลการทดลองจะพบว่าลักษณะการวัดในงานวิจัยนี้เป็นการวัดในลักษณะของ Upper Bound ซึ่งเป็นการวัดการใช้ทรัพยากรที่คาดว่า Object จะใช้เป็นปริมาณสูงสุดจากการทำงานในทุกกรณี

นอกจากนี้ในการทดสอบการใช้เวลาในการทำงานของ Algorithm ที่นำเสนอสามารถสรุปได้ดังนี้

1. การ Compile

เวลาที่ใช้ในการ Compile จะขึ้นอยู่กับจำนวนของ Bytecode ของ Class File ที่จะ Compile โดยเมื่อจำนวน Bytecode มากขึ้นจะใช้เวลาในการประมวลผลเพิ่มขึ้น โดยเฉพาะ Bytecode ที่มีผลต่อการสร้าง Expression คือ invokemethod , new และ array จะมีผลต่อการเพิ่มเวลาในการประมวลผลที่มากขึ้นอย่างชัดเจน

2. การ Evaluate

เวลาที่ใช้ในการ Evaluate จะขึ้นอยู่กับ Expression ของทั้ง Heap และ Stack ซึ่งหาก Expression มีความยาวมากขึ้น(จำนวน Expressin ย่อยใน Expression นั้นมากขึ้น) ก็จะใช้เวลาในการประมวลผลเพิ่มขึ้น รวมทั้งหาก Expression มีระดับความลึกในการ Evaluate มากขึ้นก็จะมีการใช้เวลาเพิ่มขึ้นตามระดับความลึกนั้น

5.3 ความคิดเห็นและข้อเสนอแนะ

เนื่องจากงานวิจัยนี้เป็นการวัดการใช้ทรัพยากรจากข้อมูลที่ได้จากการ Compile ดังนั้นผลลัพธ์ที่ได้จึงสามารถใช้ได้กับการวัดในส่วนของการทำงานจริงซึ่งไม่เกี่ยวกับการทำงานที่ขึ้นอยู่กับ System ซึ่งแบ่งเป็น

1. Heaparea จะมีส่วนที่เป็น Overhead ที่ใช้สำหรับการ Initial Class รวมทั้งการตรวจสอบความถูกต้องและการเตรียมพร้อมของ class เพื่อการทำงานจริง

2. Stackarea โดยปกติเมื่อมีการ Invoke Method ใดๆ จะมีการ Allocate พื้นที่ในการทำงานให้กับ method นั้น ซึ่งเรียกว่า Stack Frame ซึ่งประกอบด้วย Operand Stack , Local Variable นอกจากนี้ในการทำงานยังมีส่วนของ Data Frame ซึ่งจะเป็นพื้นที่ที่อยู่ระหว่าง Operand Stack กับ Local Variable ซึ่งส่วนของ Data Frame นี้จะทำหน้าที่เกี่ยวกับ Dynamic Linking เพื่อติดต่อกับ Constant Pool หรือ จัดการเกี่ยวกับ Exception รวมทั้ง Debugging ด้วย ซึ่งจะเป็นส่วนที่จะเกิดขึ้นในขณะ Run และขึ้นอยู่กับการ Implement ของ JVM

นอกจากนี้ Class File ที่นำมาใช้ในการทดลองนี้ยังมีข้อจำกัดในส่วนของการทำงานที่เป็นลักษณะ Dynamic เช่น การใช้ค่าจากการ Invoke Method

ดังนั้นงานวิจัยในอนาคตควรจะสามารถรองรับการทำงานได้มากขึ้น คือ

- Class ที่มีลักษณะของการรับค่าแบบ Dynamic จะต้องมีกลไกที่รองรับการทำงานในลักษณะนี้ เช่น การตรวจสอบว่ามีการใช้ค่าจากการ Invoke Method อาจจะมีส่วนที่ต้องทำการประมวลผลเพื่อให้ได้มาซึ่งค่าของ Method นั้น แล้วนำค่าที่ได้นั้นมาเข้าสู่กระบวนการของการประเมินค่าการใช้ทรัพยากร

- Heaparea จะต้องพิจารณาถึงกลไกของการเตรียมพร้อมของ Class เพื่อเริ่มการทำงานและพิจารณาถึงกลไกในการ Load Class และการใช้ System Class เพื่อเตรียมพร้อมในการทำงาน โดยจะต้องดูว่าเมื่อมีการ Load Class Class หนึ่งเข้ามาทำงานจะต้องมีการ Load Class อื่นที่เกี่ยวข้องเข้ามาในลักษณะใดบ้าง- Stackarea จะต้องพิจารณาในส่วนของการใช้พื้นที่ของ Data Frame ว่าการทำงานของ Method นั้นมีลักษณะเฉพาะตัวอย่างไร มีสิ่งใดที่บ่งบอกถึงพฤติกรรมในการใช้ Data Frame ของ Method ได้บ้าง

- นอกจากนี้ในงานวิจัยนี้การทำงานของ Object จะอยู่บนพื้นฐานของการทำงานบน Thread เดียว ซึ่งในการทำงานของ JVM จะมีการใช้พื้นที่ในการทำงานของ Java Stack ของแต่ละ Thread แยกกัน ดังนั้นงานต่อไปเพื่อสนับสนุนการทำงานในลักษณะของ Multithreading จะต้องมีการตรวจสอบการสร้าง Thread ใหม่ขึ้นมาเพื่อแยกการนับการใช้ Stack ในแต่ละ Thread เพื่อความถูกต้องในการวัดค่าการใช้ Stack

ซึ่งหากมีการปรับปรุงรูปแบบงานตามที่กล่าวมานี้การวัดการใช้ทรัพยากรก็จะมีความถูกต้องมากขึ้น เพื่อประสิทธิภาพในการทำงานและการใช้ทรัพยากรต่อไป

บรรณานุกรม

[1] วีระศักดิ์ ซึ่งถาวร. **JAVA PROGRAMMING Volume I** พ.ศ.2543 . กรุงเทพมหานคร. ซีเอ็ดยูเคชั่น. 2543.

[2] Akharin Khunkitti and Ruttikorn Varakulsiripunth , "SERVICE ENHANCEMENT USING FLEXIBLE SYSTEM FOR NETWORK MANAGEMENT",In **Preceeding of the 2000 Symposium on Theory and Applications and Information Technology** , August ,2000.

[3] Bill Venners . **Inside the JAVA Virtual Machine** . McGraw-Hill , 1998.

[4] Grzegorz Czajkowski and Thorsten von Eicken , "Jres : A Resource Accounting Interface for Java",In **Preceeding of the 1998 ACM OOPSLA Conference Vancouver** , BC ,October,1998.

[5] Gary B. Shelly , Thomas J. Cashman , Joy L.Starks.**Java Programming(Complet Concepts and Techniques)** . 2nd ED.THOMSON COURSE TECHNOLOGY , 2004.

[6] Markus Dahm ."Byte Code Engineering with the JavaClass API".**Freie University Berlin** . July,1999.

[7] Nicholas Chase . **XML and Java from scratch** .Que ,2001.

[8] Tim Lindholm and Frank Yellin.**The Java Virtual Machine Specification : The Java Series** . 2nd ED . Sun Microsystem,Inc,1999.

[9] Walter Binder , Jarle G. Hulaas and Alex Villazon,"Portable Resource Control in Java : The J-SEAL2 Approach",**ACM Conference on Object-Oriented Programming, Systems , Languages and Applications(OOPSLA-2001)**,Tampa Bay , Florida , USA, October 14 – 18 ,2001.

[10] http://h71033.www7.hp.com/nsswdocs/nsj/nsj_1_6/toolsref/java.htm

[11] www.javaworld.com/javaworld/javatips/jw-javatip130.html

[12] http://jraf2.org

[13] http://tt-bytecode.sourceforge.net/javadoc/com/techtrader/modules/tools/bytecode/

[14] http://lts.online.fr/java/matheval.javadoc/com/primalworld/math/MathEvaluator.html

ภาคผนวก ก

รายละเอียดของ bytecode instruction

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
Instruction that push a constant onto the stack	aconst_null	1(0x01)	Push null object reference	-		Null	+1
	iconst_m1	2(0x02)	Push int constant –1	-		-1	+1
	iconst_0	3(0x03)	Push int constant 0	-		0	+1
	iconst_1	4(0x04)	Push int constant 1	-		1	+1
	iconst_2	5(0x05)	Push int constant 2	-		2	+1
	iconst_3	6(0x06)	Push int constant 3	-		3	+1
	iconst_4	7(0x07)	Push int constant 4	-		4	+1
	iconst_5	8(0x08)	Push int constant 5	-		5	+1
	lconst_0	9(0x09)	Push long constant 0	-		<0>-word1,<0>-word2	+2
	lconst_1	10(0x0a)	Push long constant 1	-		<1>-word1,<1>-word2	+2
	fconst_0	11(0x0b)	Push float constant 0.0	-		<0.0>	+1
	fconst_1	12(0x0c)	Push float constant 1.0	-		<1.0>	+1
	fconst_2	13(0x0d)	Push float constant 2.0	-		<2.0>	+1
	dconst_0	14(0x0e)	Push double constant 0.0	-		<0.0>-word1,<0.0>-word2	+2
	dconst_1	15(0x0f)	Push double constant 1.0	-		<1.0>-word1,<1.0>-word2	+2
	bipush	16(0x10)	Push 8-bit signed integer	-		value	+1
	sipush	17(0x11)	Push 16-bit signed integer	-		value	+1

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	ldc	18(0x12)	Push item form constant pool	-		item	+1
	ldcw	19(0x13)	Push item from constant pool(wide)	-		item	+1
	ldc2_w	20(0x14)	Push long or double from constant pool	-		item.word1,item.word2	+2
Instruction that load a local variable onto stack	iload	21(0x15)	Load int from local variable	-		value	+1
	lload	22(0x16)	Load long from local variable	-		value.word1,value.word2	+2
	fload	23(0x17)	Load float from local variable	-		value	+1
	dload	24(0x18)	Load double from local variable	-		value.word1,value.word2	+2
	aload	25(0x19)	Load reference from local variable	-		value	+1
	iload_0	26(0x1a)	Load int from local variable 0	-		value	+1
	iload_1	27(0x1b)	Load int from local variable 1	-		value	+1
	iload_2	28(0x1c)	Load int from local variable 2	-		value	+1
	iload_3	29(0x1d)	Load int from local variable 3	-		value	+1
	lload_0	30(0x1e)	Load long from local variable 0	-		value.word1,value.word2	+2
	lload_1	31(0x1f)	Load long from local variable 1	-		value.word1,value.word2	+2

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	lload_2	32(0x20)	Load long from local variable 2	-		value.word1,value.word2	+2
	lload_3	33(0x21)	Load long from local variable 3	-		value.word1,value.word2	+2
	fload_0	34(0x22)	Load float from local variable 0	-		value	+1
	fload_1	35(0x23)	Load float from local variable 1	-		value	+1
	fload_2	36(0x24)	Load float from local variable 2	-		value	+1
	fload_3	37(0x25)	Load float from local variable 3	-		value	+1
	dload_0	38(0x26)	Load double from local variable 0	-		value.word1,value.word2	+2
	dload_1	39(0x27)	Load double from local variable 1	-		value.word1,value.word2	+2
	dload_2	40(0x28)	Load double from local variable 2	-		value.word1,value.word2	+2
	dload_3	41(0x2a)	Load double from local variable 3	-		value.word1,value.word2	+2
	aload_0	42(0x2b)	Load referane from local variable 0	-		value	+1
	aload_1	43(0x2b)	Load referane from local variable 1	-		value	+1
	aload_2	44(0x2c)	Load referane from local variable 2	-		value	+1
	aload_3	45(0x2d)	Load referane from local variable 3	-		value	+1
	iaload	46(0x2e)	Load int from array	-	arrayref,index	value	-1
	laload	47(0x2f)	Load long from array	-	arrayref,index	value.word1,value.word2	0
	faload	48(0x30)	Load float from array	-	arrayref,index	value	-1
	daload	49(0x31)	Load double from array	-	arrayref,index	value.word1,value.word2	0

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	aaload	50(0x32)	Load reference from array	-	arrayref,index	value	-1
	baload	51(0x33)	Load byte or boolean from array	-	arrayref,index	value	-1
	caload	52(0x34)	Load character from array	-	arrayref,index	value	-1
	saload	53(0x35)	Load short from aray	-	arrayref,index	value	-1
Instruction that store a value from the stack into a local variable	istore	54(0x36)	Store int into local variable	1	value		-1
	lstore	55(0x37)	Store long into local variable	1	value.word1, value.word2		-2
	fstore	56(0x38)	Store float into local variable	1	value		-1
	dstore	57(0x39)	Store double into local variable	1	value.word1, value.word2		-2
	astore	58(0x3a)	Store referenc into local variable	1	value		-1
	istore_0	59(0x3b)	Store int into local variable 0	-	value		-1
	istore_1	60(0x3c)	Store int into local variable 1	-	value		-1
	istore_2	61(0x3d)	Store int into local variable 2	-	value		-1
	istore_3	62(0x3e)	Store int into local variable 3	-	value		-1
	lstore_0	63(0x3f)	Store long into local variable 0	-	value.word1, value.word2		-2

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	lstore_1	64(0x40)	Store long into local variable 1	-	value.word1, value.word2		-2
	lstore_2	65(0x41)	Store long into local variable 2	-	value.word1, value.word2		-2
	lstore_3	66(0x42)	Store long into local variable 3	-	value.word1, value.word2		-2
	fstore_0	67(0x43)	Store float into local variable 0	-	value		-1
	fstore_1	68(0x44)	Store float into local variable 1	-	value		-1
	fstore_2	69(0x45)	Store float into local variable 2	-	value		-1
	fstore_3	70(0x46)	Store flaot into local variable 3	-	value		-1
	dstore_0	71(0x47)	Store double into local variable 0	-	value.word1, value.word2		-2
	dstore_1	72(0x48)	Store double into local variable 1	-	value.word1, value.word2		-2
	dstore_2	73(0x49)	Store double into local variable 2	-	value.word1, value.word2		-2
	dstore_3	74(0x4a)	Store double into local variable 3	-	value.word1, value.word2		-2

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	astore_0	75(0x4b)	Store reference or retutn address into local variable	-	value		-1
	astore_1	76(0x4c)	Store reference or retutn address into local variable 1	-	value		-1
	astore_2	77(0x4d)	Store reference or retutn address into local variable 2	-	value		-1
	astore_3	78(0x4e)	Store reference or retutn address into local variable 3	-	value		-1
	iastore	79(0x4f)	Store into int array	-	arrayref,index, value		-3
	lastore	80(0x50)	Store into long array	-	arrayref,index, value.word1, value.word2		-4
	fastore	81(0x51)	Store into float array	-	arrayref,index, value		-3
	dastore	82(0x52)	Store into double array	-	arrayref,index, value.word1, value.word2		-4

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	aastore	83(0x53)	Store into reference array	-	arrayref,index, value		-3
	bastore	84(0x54)	Store into byte or boolean array	-	arrayref,index, value		-3
	castore	85(0x55)	Store into character array	-	arrayref,index, value		-3
	sastore	86(0x56)	Store into short array	-	arrayref,index, value		-3
Wide instruction	wide	196(0xc4)	Extend a local variable index with additional bytes	-			
Generic stack operation	nop	0(0x00)	Do nothing	-			0
	pop	87(0x57)	Pop top stack word	-	word		-1
	pop2	88(0x58)	Pop top two stack word	-	word1,word2		-2
	dup	89(0x59)	Duplicate top stack word	-	word	word,word	+1
	dup_x1	90(0x5a)	Duplicate top stack word and put two down	-	word2,word1	word1,word2,word1	+1
	dup_x2	91(0x5b)	Duplicate top stack word and put three down	-	word3,word2, word1	word1,word3,word2, word1	+1

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	dup2	92(0x5c)	Duplicate top two stack words	-	word2,word1	word2,word1,word2, word1	+2
	dup2_x1	93(0x5d)	Duplicate top two stack words and put two down	-	word3,word2, word1	word2,word1,word3, word2,word1	+2
	dup2_x2	94(0x5e)	Duplicate top two stack words and put three down	-	word4,word3, word2,word1	word2,word1,word4, word3,word2,word1	+2
	swap	95(0x5f)	Swap top two stack words	-	word1,word2	word2,word1	0
Integer arithmetic	iadd	96(0x60)	Add ints	-	Value1,value2	result	-1
	ladd	97(0x61)	Add longs	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	isub	100(0x64)	Subtract ints	-	value1,value2	result	-1
	lsub	101(0x65)	Subtract longs	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	imul	104(0x68)	Multiply ints	-	Value1,value2	result	-1

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	lmul	105(0x69)	Multiply longs	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	idiv	108(0x6c)	Divide ints	-	Value1,value2	result	-1
	ldiv	109(0x6d)	Divide longs	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	irem	112(0x70)	Calculate remainder of division of ints	-	Value1,value2	result	-1
	lrem	113(0x71)	Calculate remainder of division of longs	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	ineg	116(0x74)	Negate int	-	value	Value	0
	lneg	117(0x75)	Negete long	-	value.word1, value.word2	value.word1,value.word2	0

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	iinc	132(0x84)	Increment int local variable by constant	2			0
Floating-point arithmetic	fadd	98(0x62)	Add floats	-	Value1,value2	result	-1
	dadd	99(0x63)	Add doubles	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	fsub	102(0x66)	Subtract floats	-	Value1,value2	result	-1
	dsub	103(0x67)	Subtract doubles	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	fmul	106(0x69)	Multiply floats	-	Value1,value2	result	-1
	dmul	107(0x6a)	Multiply doubles	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	fdiv	110(0x6e)	Divide floats	-	Value1,value2	result	-1

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	ddiv	111(0x6f)	Divide doubles	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	frem	114(0x72)	Calculate remainder of division of floats	-	value1,value2	result	-1
	drem	115(0x73)	Calculate remainder of division of doubles	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,2	-2
	fneg	118(0x76)	Negate floats	-	value	value	0
	dneg	119(0x77)	Negete doubles	-	value.word1, value.word2	value.word1,value.word2	0
Logic shift operation	ishl	120(0x78)	Perform left shift on int	-	value1,value2	result	-1
	lshl	121(0x79)	Perform left shift on long	-	value1.word1, value1.word2, value2	result.word1,result.word2	-1
	ishr	122(0x7a)	Perform arithmetic right shift on int	-	value1,value2	result	-1

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	lshr	123(0x7b)	Perform arithmetic right shift on long	-	value1.word1, value1.word2, value2	result.word1,result.word2	-1
	iushr	124(0x7c)	Perform logical right shift on int	-	value1,value2	result	-1
	lushr	125(0x7d)	Perform logical right shift on long	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,result.word2	-1
Bitwise boolean operation	iand	126(0x7e)	Perform boolean and on ints	-	value1,value2	result	-1
	land	127(0x7f)	Perform boolean and on longs	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,result.word2	-2
	ior	128(0x80)	Perform boolean or on ints	-	value1,value2	result	-1
	lor	129(0x81)	Perform boolean or on longs	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,result.word2	-2

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	ixor	130(0x82)	Perform boolean xor on ints	-	value1,value2	result	-1
	lxor	131(0x83)	Perform boolean xor on longs	-	value1.word1, value1.word2, value2.word1, value2.word2	result.word1,result.word2	-2
Type conversion	i2l	133(0x85)	Convert int to long	-	value	result.word1,result.word2	+1
	i2f	134(0x86)	Convert int to float	-	value	result	0
	i2d	135(0x87)	Convert int to double	-	value	result.word1,result.word2	+1
	l2i	136(0x88)	Convert long to int	-	value1.word1, value1.word2	result	-1
	l2f	137(0x89)	Convert long to float	-	value1.word1, value1.word2	result	-1
	l2d	138(0x8a)	Convert long to double	-	value1.word1, value1.word2	result.word1,result.word2	0
	f2i	139(0x8b)	Convert float to int	-	value	result	0
	f2l	140(0x8c)	Convert float to long	-	value	result.word1,result.word2	+1
	f2d	141(0x8d)	Convert float to double	-	value	result.word1,result.word2	+1
	d2i	142(0x8e)	Convert double to int	-	value1.word1, value1.word2	result	-1

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	d2l	143(0x8f)	Convert double to long	-	value1.word1, value1.word2	result.word1,2	0
	d2f	144(0x90)	Convert double to float	-	value1.word1, value1.word2	result	-1
	i2b	145(0x91)	Convert int to byte	-	value	result	0
	i2c	146(0x92)	Convert int to char	-	value	result	0
	i2s	147(0x93)	Convert int to short	-	value	result	0
Comparison instructions	lcmp	148(0x94)	Conpare longs	-	value1.word1, value1.word2, value2.word1, value2.word2	result	-3
	fcmpl	149(0x95)	Compare floats(-1 on NaN)	-	value1,value2	result	-1
	fcmpg	150(0x96)	Conpare floats(1 on NaN)	-	value1,value2	result	-1
	dcmpl	151(0x97)	Compare doubles(-1 in NaN)	-	value1.word1, value1.word2, value2.word1, value2.word2	result	-3

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	dcmpg	152(0x98)	Compare doubles(1 on NaN)	-	value1.word1, value1.word2, value2.word1, value2.word2	result	-3
Control flow instruction	ifeg	153(0x99)	Branch if equal to 0	2	value		-1
	ifne	154(0x9a)	Branch if not equal to 0	2	value		-1
	iflt	155(0x9b)	Branch if less than 0	2	value		-1
	ifge	156(0x9c)	Branch if greater than or equal to 0	2	value		-1
	ifge	157(0x9d)	Branch if greater than 0	2	value		-1
	ifle	158(0x9e)	Branch if less than or equal to 0	2	value		-1
	if_icmpeg	159(0x9f)	Branch if ints equal	2	value1,value2		-2
	if_icmpne	160(0xa0)	Branch if ints not equal	2	value1,value2		-2
	if_icmplt	161(0xa1)	Branch if int less than other int	2	value1,value2		-2
	if_icmpge	162(0xa2)	Branch if int greater than or equal to other int	2	value1,value2		-2
	if_icmpgt	163(0xa3)	Branch if int greater than other int	2	value1,value2		-2
	if_icmple	164(0xa4)	Branch if int less than or equal to other int	2	value1,value2		-2

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	if_acmpeg	165(0xa5)	Branch if object reference are equal	2	value1,value2		-2
	if_acmpne	166(0xa6)	Branc if object reference not equal	2	value1,value2		-2
	Ifnull	198(0xc6)	Branch if null	2	value		-1
	ifnonnull	199(0xc7)	Branch if not null	2	value		-1
	goto	167(0xa7)	Branch always	2			0
	goto_w	200(0xc8)	Branch always (wide)	4			0
	tableswitch	170(0xaa)	Access jump table by index and jump	>12	index		-1
	lookup switch	171(0xab)	Access jump table by key match and jump	>12	key		-1
Method return instruction	ireturn	172(0xac)	Return int from method	-	value	[empty]	-1
	lreturn	173(0xad)	Return long from method	-	value.word1, value.word2	[empty]	-2
	freturn	174(0xae)	Return float from method	-	value	[empty]	-1
	dreturn	175(0xaf)	Return double from method	-	value.word1, value.word2	[empty]	-2
	areturn	176(0xb0)	Return reference form method	-	value	[empty]	-1
	return	177(0xb1)	Return from method	-		[empty]	0

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
Finally clause	jsr	168(0xa8)	Jump to subroutine	2		address	+1
	jsr_w	201(0xc9)	Jump to subroutine(wide)	4		address	+1
	ret	169(0xa9)	Return from subroutine	1			0
Method invocation instruction	Invokevirtual	182(0xb6)	Invoke instance method	2	objectref, [arg1,arg2,...]		-(number of arg.+1)
	invoke special	183(0xb7)		2	objectref, [arg1,arg2,...]		-(number of arg.+1)
	invokestatic	184(0xb8)	Invoke a class(static) method	2	[arg1,arg2,...]		-(number of arg.)
	invoke interface	185(0xb9)	Invoke interface method	4	objectref, [arg1,arg2,...]		-(number of arg.+1)
Objects and array object	new	187(0xbb)	Create new object	2		objectref	+1
	checkcast	192(0xc0)	Make sure object is of given type	2	objectref	objectref	0
	getfield	180(0xb4)	Fetch field from object	2	objectref	value or value.word1,value.word2	+1 or +2
	putfield	181(0xb5)	Set field in object	2	objectref,value or objectref, value.word1, value.word2		-2 or -3

Type	Mnemonic	Opcode	Description	Argument	Stack		Stackchange
					Before	After	
	getstatic	178(0xb2)	Fetch static field from class	2		value or value.word1,value.word2	+1 or +2
	putstatic	179(0xb3)	Set static filed in class	2	value or value.word1, value.word2		-1 or -2
	instanceof	193(0xc1)	Determine whether an object is of given type	2	objectref	result	0
	newarray	188(0xbc)	Allocate new array of primitive type components	1	count	arrayref	0
	anewarray	189(0xbd)	Allocate new array of reference type components	2	count	arrayref	0
	arraylength	190(0xbe)	Get length of array	-	arrayref	length	0
	multi anewarray	197(0xc5)	Allocate a new mutidimensional array	3	count, [count2,...]	arrayref	
Thread Synchroni-zation	monitorenter	194(0xc2)	Enter and acquire object monitor	-	objectref		-1
	monitorexit	195(0xc3)	Release and exit object monitor	-	objectref		-1
Exception	athrow	191(0xbf)	Throw exception or error	-	objectref	objectref	0

ภาคผนวก ข

รายละเอียดของ Code ที่ใช้ในการทดลอง

code ของ Vladimir Roubtsov (“Java Tip 130 : Do you know your data size” , http :// www.javaworld.com/javaworld/javatips/jw-javatip130.html)

```
//Sizeof.java for evaluate size of object
public class Sizeof{
        public static void main (String [] args) throws Exception{
                // Warm up all classes/methods we will use
                runGC ();
                usedMemory ();
                // Array to keep strong references to allocated objects
                final int count = 10000;
                Object [] objects = new Object [count];

                long heap1 = 0;

                // Allocate count+1 objects, discard the first one
                for (int i = -1; i < count; ++ i){
                        Object object = null;
                        object = new test();
                        if (i >= 0)
                                objects [i] = object;
                        else{
                                object = null; // Discard the warm up object
                                runGC ();
                                heap1 = usedMemory (); // Take a before heapsnapshot
                        }
                }
```

```
                runGC ();
                long heap2 = usedMemory (); // Take an after heap snapshot:

                final int size = Math.round (((float)(heap2 - heap1))/count);
                System.out.println ("'before' heap: " + heap1 + ", 'after' heap: " + heap2);
                 System.out.println ("heap delta: " + (heap2 - heap1) + ", {" + objects
[0].getClass () + "} size = " + size + " bytes");

                for (int i = 0; i < count; ++ i) objects [i] = null;
                objects = null;
        }

        private static void runGC () throws Exception{
                // It helps to call Runtime.gc()
                // using several method calls:
                for (int r = 0; r < 4; ++ r) _runGC ();
        }

        private static void _runGC () throws Exception{
                long usedMem1 = usedMemory (), usedMem2 = Long.MAX_VALUE;
                for (int i = 0; (usedMem1 < usedMem2) && (i < 500); ++ i){
                        s_runtime.runFinalization ();
                        s_runtime.gc ();
                        Thread.currentThread ().yield ();
                        usedMem2 = usedMem1;
                        usedMem1 = usedMemory ();
                }
        }
```

```
        private static long usedMemory (){
                return s_runtime.totalMemory () - s_runtime.freeMemory ();
        }

        private static final Runtime s_runtime = Runtime.getRuntime ();

} // End of class
```

code ที่ 2 สำหรับการหาพื้นที่ของ heaparea ในการ invoke method

```
class GcTest{
        public static final Runtime r = Runtime.getRuntime();
        public static void main(String args[])throws Exception{
                final int count = 10;
                long h1 = 0;
                long h2 = 0;
                long size = 0;
                long Sumsize = 0;
                long Result = 0;
                for(int i = 0;i<count;i++){
                        if(i==0){
                                runGC();
                        }else{
                                r.gc();
                        }
                        h1 = r.freeMemory();
                        test z = new test();
                        h2 = r.freeMemory();
                        System.out.println("freememory at " + i + " : "+h2);
                        size = h1-h2;
                        System.out.println("size at " + i + "          : "+size);
                        Sumsize = Sumsize + size;
                        System.out.println("sumsize at " + i + "     : "+Sumsize);
                        System.out.println();
                }
                Result = Sumsize/count;
                System.out.println("size : "+Result);
                r.gc();
        }
```

```
        private static void runGC () throws Exception{
                for (int r = 0; r < 4; ++ r) _runGC ();
        }
        private static void _runGC () throws Exception{
                long usedMem1 = usedMemory (), usedMem2 = Long.MAX_VALUE;
                for (int i = 0; (usedMem1 < usedMem2) && (i < 500); ++ i){
                        r.runFinalization ();
                        r.gc ();
                        Thread.currentThread ().yield ();
                                usedMem2 = usedMem1;
                        usedMem1 = usedMemory ();
                }
        }
        private static long usedMemory (){
                return r.totalMemory () - r.freeMemory ();
        }
}
```

ภาคผนวก ค

ตัวอย่างของ Class File ที่ใช้ในการทดลอง

```
class IntegerConstant{
        public static void main(String args[]){
                System.out.println(Byte.MAX_VALUE);
                System.out.println(Byte.MIN_VALUE);
                System.out.println(Short.MAX_VALUE);
                System.out.println(Short.MIN_VALUE);
                System.out.println(Integer.MAX_VALUE);
                System.out.println(Integer.MIN_VALUE);
                System.out.println(Long.MAX_VALUE);
                System.out.println(Long.MIN_VALUE);
        }
}
```

```
class AriAssOp{
        public static void main(String args[]){
                int x = 1;
                x += 1;System.out.println(x);
                x *= 2;System.out.println(x);
                x -= 3;System.out.println(x);
                x /= 4;System.out.println(x);
                float y = 1f;
                y += 1;System.out.println(y);
                y *= 2;System.out.println(y);
                y -= 3;System.out.println(y);
                y /= 4;System.out.println(y);
        }
}
```

```
class IntegerBit{
        public static void main(String args[]){
                Binary.print(-2 , 32);
                Binary.print(-1 , 32);
                Binary.print(0 , 32);
                Binary.print(1 , 32);
                Binary.print(2  , 32);
        }
}
```

```
class Test20{
        public static void main(String args[]){
                int A = 1234, B = 9876;
                System.out.print("A=");Binary.print(A , 32);
                System.out.print("B=");Binary.print(B , 32);
                System.out.print("A|B=");Binary.print(A|B , 32);
                System.out.print("A^B=");Binary.print(A^B , 32);
        }
}
```

```
class CondOp{
        public static void main(String args[]){
                int a = Integer.parseInt(args[0]);
                int b = Integer.parseInt(args[1]);
                System.out.println((a > b)? a : b);
        }
}
```

```
class Cast{
        public static void main(String args[]){
                double d = 257.234;
                int i = (int) d;
                System.out.println(i);
                byte b = (byte) i;
                System.out.println(b);
        }
}
```

```
class Test24{
        public static void main(String args[]){
                int s = Integer.parseInt(args[0]);
                char g =        (s >= 85) ? 'A' :
                                (s >= 75) ? 'B' :
                                (s >= 65) ? 'C' :
                                (s >= 55) ? 'D' : 'F' ;
                System.out.println(g);
        }
}
```

```
class UnaryPromotion{
        public static void main(String args[]){
                byte b = 10;
                char c = '\u0001';
                int  a[] = new int[b];
                a[0] = -1;
                a[c] = 1;
                a[2] = -c;
                a[3] = ~b;
                a[4] = b << 4L;
        }
}
```

```
class BinaryPromotion{
        public static void main(String args[]){
                int i;
                char c = 2;
                long l = 3L;
                float f = 1.5f;
                i = (int)(c + l * f - 3);
                System.out.println((c + l * f - 3));
                System.out.println(i);
        }
}
```

```
class SwitchBlock{
        public static void main(String args[]){
                int x = 1;
                switch(x){
                        case 0 :    int a;
                                    System.out.println(0);
                                    break;
                        case 1 :    int b;
                                    System.out.println(1);
                                    break;
                        default :   System.out.println(2);
                }
        }
}
```

```
class Test30{
        public static void main(String args[]){
                here : {
                        System.out.println("Hello");
                        he : {
                                System.out.println("Hi");
                        }
                }
        }
}
```

```
class If1{
        public static void main(String args[]){
                int a = Integer.parseInt("1");
                int b = Integer.parseInt("2");
                if(b == 0){
                        System.out.println("Error:dividebyzero!");
                        System.exit(0);
                }
                System.out.println(a/b);
        }
}
```

```
class If3{
        public static void main(String args[]){
                int s = Integer.parseInt(args[0]);
                if(s >= 85)
                        System.out.println('A');
                else if(s >= 75)
                        System.out.println('B');
                else if(s >= 65)
                        System.out.println('C');
                else if(s >=55)
                        System.out.println('D');
                else
                        System.out.println('F');
        }
}
```

```
class Switch3{
        public static void main(String args[]){
                char g = args[0].charAt(0);
                switch(g){
                        case 'A' :
                        case 'B' :
                        case 'C' : System.out.println("Passed");
                                break;
                        case 'D' :
                        case 'F' : System.out.println("Fails");
                                break;
                        default : System.out.println("Invalid");
                }
        }
}
```

```
class Continue1{
        public static void main(String args[]){
                for(int i = 0;i < 5; i++){
                        if( i == 3)
                                continue;
                        System.out.println(i);
                }
                System.out.println("End");
        }
}
```

```
class MyVar{
        static int x = 1;
}
class StaticVar{
        public static void main(String args[]){
                for(int i = 1; i < 5;i++)
                        System.out.println(MyVar.x++);
        }
}
```

```
class MyVar2{
        public int x = 1;
}
class MemberVar{
        public static void main(String args[]){
                for(int i = 0 ; i < 3 ;i++){
                        MyVar2 m  = new MyVar2();
                        System.out.println(m.x++);
                }
        }
}
```

```
class MyVar3{
        int p(){
                int x  =0;
                return x++;
        }
}
class LocalVar{
        public static void main(String args[]){
                MyVar3 v = new MyVar3();
                for(int i = 0;i < 3;i++){
                        System.out.println(v.p());
                }
        }
}
```

```
class Test32{
        public static void main(String args[]){
                int n = Integer.parseInt(args[0]);
                switch(n%8){
                        case 0 : System.out.println("0");
                                 break;
                        case 7 :  System.out.println("7");
                                 break;
                        case 6 :  System.out.println("6");
                                 break;
                        case 5 : System.out.println("5");
                                 break;
                        case 4 :  System.out.println("4");
                                 break;
                        case 3 :  System.out.println("3");
                                 break;
                        case 2 :  System.out.println("2");
                                 break;
                        case 1 :  System.out.println("1");
                }
        }
}
```

```
class Overload{
        static int sq(int n){return n*n;}
        static double sq(float n){return (double)n*n;}
        static double sq(double n){return n*n;}
        public static void main(String args[]){
                int x = sq(10);
                double y = sq(11.0);
        }
}
```

```
class StaticDynamic{
        static int x = 1;
        static void print(){
                System.out.println(x);
        }
        public static void main(String args[]){
                int x = 2;
                print();
        }
}
```

```
class Student2{
        private int id;
        private String name;
        Student2() {id = 0;name = null;}
        Student2(int i,String n){id = i;name = n;}
        Student2(Student2 s) {id = s.id;name = s.name;}
        public void setId(int i){id = 1;}
        public int getId() {return id;}
        public void setName(String n){name = n;}
        public String getName() {return name;}
}
```

```
class StudentTest2{
        public static void main(String args[]){
                Student2 x = new Student2();
                Student2 y= new Student2(123 , "JohnRambo");
                Student2 z = new Student2(y);
        }
}
```

```
class MyInteger{
        private int i;
        MyInteger (int x){i = x;}
        public void inc() {i++;}
        public int getI() {return i;}
}
class ParaPass{
        static void f(int n){n++;}
        static void g(MyInteger n){n.inc();}
        public static void main(String args[]){
                int a = 0;
                f(a);
                System.out.println(a);
                MyInteger b = new MyInteger(0);
                g(b);
                System.out.println(b.getI());
        }
}
```

```
class MyMath{
        static public double sq(double x){return x*x;}
}
class StaticMethod{
        public static void main(String args[]){
                System.out.println(MyMath.sq(4.0));
        }
}
```

```
class MyClass{
        private int x;
        MyClass (int x){this.x = x;}
        public int getX() {return x;}
}
class LoadClass{
        public static void main(String args[]){
                MyClass a = new MyClass(1);
                MyClass b = new MyClass(2);
        }
}
```

```
class MyClass2{
        static public int n = 0;
        private int x;
        MyClass2(int x){this.x = x;n++;}
}
class StaticData{
        public static void main(String args[]){
                MyClass2 a = new MyClass2(1);
                MyClass2 b = new MyClass2(2);
        }
}
```

```
class MyClass4{
        static{
                System.out.println("Hello");
        }
        static{
                System.out.println("Hello , again");
        }
}
class Test9{
        public static void main(String args[]){
                new MyClass4();
        }
}
```

```
class AA{int a = 1;}
class BB extends AA{int b = 2;}
final class CC extends BB{int c = 3;}
class Inherit2{
        public static void main(String args[]){
                CC z = new CC();
                System.out.print(z.a + z.b + z.c);
        }
}
```

```
class ArrayInit{
        public static void main(String args[]){
                int i[] = {1 , 2 , 3 , 5 , 7 , 7+4};
                char c[] = {'H' , 'e' , 'l' , 'l' , 'o'};
                String m[] = {"how" , "do" , "you" , "do" , "?"};
                Student2 s[] = {    new Student2(123 , "JohnRambo") ,
                                    new Student2(555 , "MikeTyson") ,
                                    new Student2(666 , "JackRipper")};
        }
}
```

```
class A{
        void printA() {System.out.println('A');}
}
class B extends A{
        void printB() {System.out.println('B');}
}
class Inherit1{
        public static void main(String args[]){
                A x = new A();
                x.printA();
                B y = new B();
                y.printA();
                y.printB();
        }
}
```

```
class A{int x = 1;}
class B extends A{int y =2;}
class Inherit3{
        public static void main(String args[]){
                A a = new A();
                System.out.println(a.x);
                B b = new B();
                b.x--;
                a = b;
                System.out.println(a.x);
        }
}
```

```
class A{void print() {System.out.println('A'); } }
class B extends A {void print() {System.out.println('B'); }}
class DynamicBinding{
        static void test(A a) {a.print();}
        public static void main(String args[]){
                test(new A());
                test(new B());
        }
}
```

```
class A{
        int a;
        void print(){System.out.println(a);}
}
class B extends A{
        int a;
        B(int x , int y) {super.a = x;this.a = y;}
        void print(){
                super.print();
                System.out.println(a);
        }
}
class Super1{
        public static void main(String args[]){
                B b = new B(1,2);
                b.print();
        }
}
```

```
class BB extends A{
        void print(){
                System.out.print(a);
        }
}
class C extends BB{
        int a;
        C(int x,int y) {super.a = x; this.a = y;}
        void print(){
                super.print();
                System.out.print(a);
        }
}
class Super2{
        public static void main(String args[]){
                C c = new C(1,2);
                c.print();
        }
}
```

```
class A{public int x = 2;}
class B extends A{ public int x = 2; }
class StaticBinding{
        static void test(A a){
                System.out.println(a.x);
        }
        public static void main(String args[]){
                test(new A());
                test(new B());
        }
}
```

```
class Figure{
        double width , height;
        String name;
        Figure(double x , double y , String n){
                width = x; height = y; name = n;
        }
        String getName() {return name;}
        double getArea() {return 0.0;}
}
class Rectangle extends Figure{
        Rectangle (double w , double h){super(w,h,"rectangle");}
        double getArea() { return width * height;}
}
class Triangle extends Figure{
        Triangle(double w , double h) {super (w,h,"triangle");}
        double getArea(){return 0.5 * width * height;}
}
class PolyEx{
        static void compute(Figure x){
                System.out.println(x.getName());
                System.out.println(x.getArea());
        }
        public static void main(String args[]){
                compute(new Figure(1,1,"undefined"));
                //compute(new Triangle(1,1));
                //compute(new Rectangle(1,1));
        }
}
```

```
abstract class One{
        abstract void greet1();
        abstract void greet2();
}
abstract class Two extends One{
        void greet1() {System.out.println("Hello");}
}
class Three extends Two{
        void greet2() {System.out.println("Hi");}
}
class Abstract3{
        static public void main(String args[]){
                Three t = new Three();
                t.greet1();
                t.greet2();
        }
}
```

```
interface Greeting{
        static final String s = "Hello";
        abstract void greet(int s);
}
class Hello implements Greeting{
        public void greet(){System.out.println(s);}
}
class Interface1{
        public static void main(String args[]){
                new Hello().greet();
        }
}
```

```
//use interface greeting from Greeting
//use class Hello from Interface1
class Hi implements Greeting{
        public void greet(){System.out.println("Hi");}
        void moreGreet(){System.out.println("Hoe do you do?");}
}
class Interface2{
        static void test(Greeting g){g.greet();}
        public static void main(String args[]){
                test(new Hello());
                test(new Hi());
        }
}
```

```
interface Greeting{void greet();}
interface Farewell{void bye();}
class MyClass implements Greeting , Farewell{
        public void greet(){ System.out.println("Hello");}
        public void bye() {System.out.println("Goodbye");}
}
class MulInterface{
        static void test1(Greeting g){g.greet();}
        static void test2(Farewell f) {f.bye();}
        public static void main(String args[]){
                MyClass s = new MyClass();
                test1(s);
                test2(s);
        }
}
```

```
interface Greeting{
        void hi();
        void hello();
}
abstract class NotYesGreet implements Greeting{
        public void hi() {System.out.println("Hi");}
}
class Greet extends NotYesGreet{
        public void hello(){System.out.println("Hello");}
}
class PartInterface{
        public static void main(String args[]){
                Greet g = new Greet();
                g.hi();
                g.hello();
        }
}
```

```
interface Greeting{void hi();}
interface MoreGreeting extends Greeting{void hello();}
class Greet implements MoreGreeting{
        public void hi() { System.out.println("Hi");}
        public void hello() {System.out.println("Hello");}
}
class ExtInterface{
        public static void main(String args[]){
                Greet g = new Greet();
                g.hi();
                g.hello();
        }
}
```

```
interface MyConstants{
        int YES = 1;
        double PI = 3.1415;
        String GREET = "Hello";
}
class ConsInterface implements MyConstants{
        public static void main(String args[]){
                System.out.println(YES);
                System.out.println(PI);
                System.out.println(GREET);
        }
}
```

```
class Printer{
        protected int c =0;
        public int getCount(){return c;}
        public void print(){
                c++;
                System.out.println("Printer");
        }
}
class MyApplication{
        Printer p;
        void setPrinter(Printer p){this.p = p;}
        void printOut(){p.print();}
}
class MyApplicationTest{
        public static void main(String args[]){
                MyApplication a = new MyApplication();
                a.setPrinter(new XPrinter());
                a.printOut();
        }
}
```

```
class A{
        static class B{
                void print(){
                        System.out.println("B");
                }
        }
        void print(){
                System.out.println("A");
        }
}
class NestedClassTest1{
        public static void main(String args[]){
                A a = new A();
                a.print();
                A.B b = new A.B();
                b.print();
        }
}
```

```
class A{
        int w;
        static private int v = 0;
        static int x = 1;
        static class B{
                int y = 2;
                static int z = 3;
                void print(){
                        System.out.println(x);
                }
        }
}
class NestedClassTest3{
        public static void main(String args[]){
                System.out.println(A.B.z);
                new A.B().print();
        }
}
```

```
class testotherlength{
        static int w;
        public static void main(String args[]){
                int a = 3;
                int b = 2;
                int c = a + b;
                tt f = new tt();
                System.out.println(f.x);
                int y[] = new int[f.x];
        }
}
class tt{
        int x;
        void ttt(){
                x = 5;
        }
}
```

```
class testcompile1{
        public static void main(String args[]){
                for(int i = 0;i < 3;i++){
                        for(int j = 0;j <2;j++){
                                int x[] = new int[7];
                        }
                        int w[] = new int[4];
                }
                int y[] = new int[3];
        }
}
```

```
class testdeep{
        public static void main(String args[]){
                int x = 5;
                int y = 3;
                int z = a(x,y);
        }
        static int a(int r,int p){
                int s;
                s = b(r)+b(p);
                return s;
        }
        static int b(int m){
                m = t.o(m);
                return m;
        }
}
class t{
        static int o(int c){
                return c;
        }
}
```

```
class testtest1{
        public static void main(String args[]){
                int y = 8;
                int c[] = new int[8];
                if(y < 8){
                        cal();
                }else{
                        testtest t = new testtest();
                }
        }
        public static void cal(){
                int x = 9;
        }
}
```

```
class testadd6{
        public static void main(String args[]){
                int x[] = new int[10];
                testadd6 c = new testadd6();
                c.cal(x);
        }
        void cal(int[] s){
                char c[] = new char[s.length];
        }
}
```

```
class testloop22{
        int x;
        public static void main(String args[]){
                int a = 3;
                int b = 5;
                testloop22 v = new testloop22();
                v.cal(a);
        }
        void cal(int a){
                for(int i = 0;i < a;i++){
                        System.out.println(i);
                }

        }
}
```

```
class invoke18{
        public static void main(String args[]){
                int n = 5;
                int x = fac(n);
        }
        public static int fac(int n){
                if(n==0){
                        System.out.println(n);
                        return n;
                }else{
                        System.out.println(n);
                        n = n-1;
                        return fac(n);
                }
        }
}
```

```
class invoke2{
        public static void main(String args[]){
                int a = 0;
                int b = 1;
                invoke2 f = new invoke2();
                int c = f.getA(a,b);
                getB(c);
        }
        public  int getA(int a,int b){
                return a +b;
        }
        public static void getB(int c){
        }
}
```

```
class invoke22{
        public static void main(String args[]){
                int x = 5;
                int d = getLength(4,x);
        }
        public static int getLength(int y , int x){
                int z = 0;
                switch(x){
                        case 0 : z = 0;
                                System.out.println(z);
                                break;
                        case 1 :  z = 1;
                                System.out.println(z);
                                break;
                        case 2 :  z = 2;
                                System.out.println(z);
                                break;
                        case 3 :  z = getLength(4,x-1);
                                break;
                        case 4 : z = getLength(4,x-1);
                                break;
                        case 5 :  z = getLength(3,x-2);
                                break;
                }
                return z;
        }
}
```

```
class invoketest4{
        public static void main(String args[]){
                Student x = new Student();
                x.id = 123;
                x.setName("JohnRambo");
                //x.setGpa(4.00);
                test4 y = new test4();
                y.print("John");
        }
}
class test4{
        int x;
        void print(String g){
                System.out.println(g);
        }
}
```

```
class Test8{
        public static void main(String args[]){
                String a = "Hello how do you?";
                System.out.println(a.indexOf('o'));
                System.out.println(a.lastIndexOf('o'));
                System.out.println(a.indexOf("do"));
                System.out.println(a.lastIndexOf("do"));
                System.out.println(a.indexOf('o',5));
                System.out.println(a.lastIndexOf('o',15));
                System.out.println(a.indexOf("do",11));
                System.out.println(a.lastIndexOf("do",11));
        }
}
```

```
class Replace{
        public static void main(String args[]){
                String a = "Hello";
                String b = a.replace('l','L');
                String c = a.toLowerCase();
                String d = a.toUpperCase();
                System.out.println(a);
                System.out.println(b);
                System.out.println(c);
                System.out.println(d);
        }
}
```

```
class Greet{
        void sayHello(String s){
                if(s.equals("John"))
                        System.exit(0);
                System.out.println(s);
        }
}
class ExitTest1{
        public static void main(String args[]){
                Greet g = new Greet();
                g.sayHello(args[0]);
                System.out.println(args[0]);
        }
}
```

```
import java.io.*;
class PrintStreamTest{
        public static void main(String args[])throws IOException{
                PrintStream pout = new PrintStream(System.out);
                pout.println(12345);
                pout.println(true);
                pout.println((short)65);
                pout.println('A');
                pout.println("Hello");
        }
}
```

```
class SoundTrack{
        static public float track_time[] = {3.24f,3.28f,2.58f,4.01f,3.17f,3.59f,2.24f,3.22f};
        static float getLongTrack(){
                float max = 0.0f;
                for(int i =0;i<8;i++)
                        if(max<track_time[i])
                                max = track_time[i];
                return max;
        }
        public static void main(String args[]){
                System.out.print(getLongTrack());
        }
}
```

```
public class stackDepth5{
        static int depth;
        public static void main(String args[]){
                try {
                        double d1 = 0.0;
                        double d2 = 0.0;
                        double d3 = 0.0;
                        double d4 = 0.0;
                        double d5 = 0.0;
                        depth = 0;
                        test2(args);
                } catch (StackOverflowError err) {
                        System.out.println(err);
                }
                System.out.println("depth  : "+depth);
        }
}
```

```
class Flight1{
        String captain_name;
        int qty_hostess,qty_seat;
        void FlightTeam(String n,int h,int s){
                captain_name = n;
                qty_hostess = h;
                qty_seat = s;
        }
        void FlightTeam(String n){
                captain_name = n;
        }
        void FlightTeam(int s,int h,String n){
                captain_name = n;
                qty_hostess = h;
                qty_seat = s;
        }
        void printInfo(){
                System.out.println(captain_name);
                System.out.println(qty_seat);
                System.out.println(qty_hostess);
        }
}
class AirLine{
        public static void main(String args[]){
                Flight1 f1 = new Flight1();
                Flight1 f2 = new Flight1();
                Flight1 f3 = new Flight1();
                f1.FlightTeam("PeterAnderson",5,50);
                f2.FlightTeam("SompongN.Lumpang");
                f3.FlightTeam(120,10,"AlenTangee");
                f1.printInfo();
                f2.printInfo();
                f3.printInfo();
        }
}
```

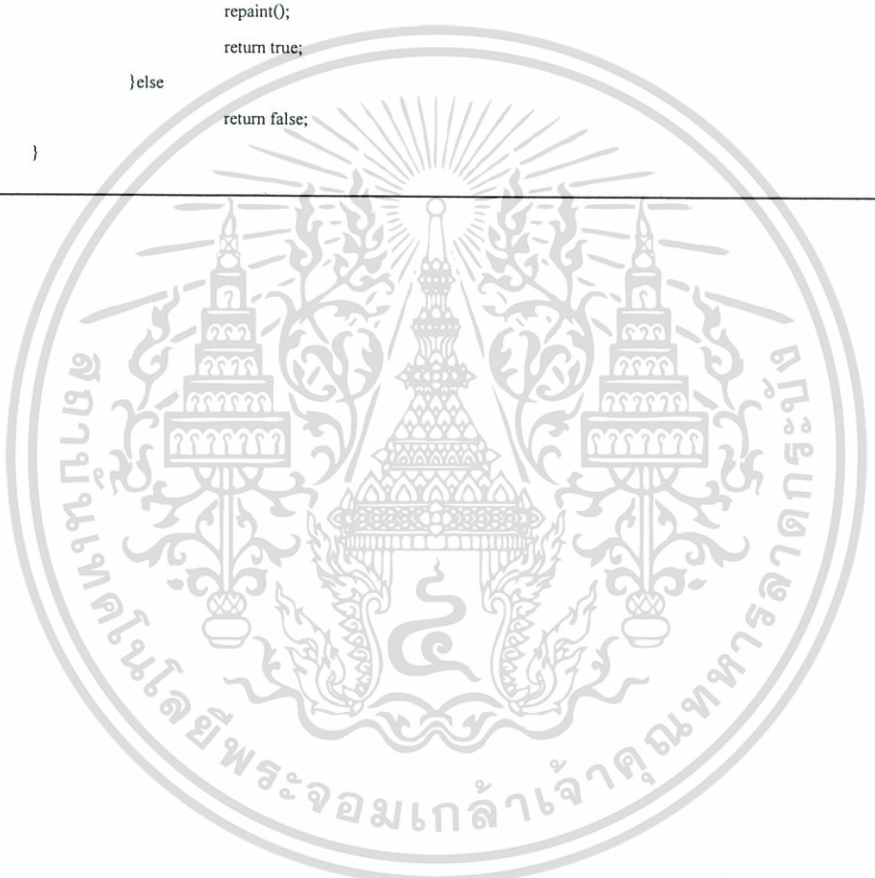
```
class Baby{
        String name;
        char gender;
        void setName(String n){
                name = n;
        }
        String getName(){
                return name;
        }
        void setGender(char g){
                gender = g;
        }
        char getGender(){
                return gender;
        }
}
class MyBaby{
        public static void main(String args[]){
                Baby Kid = new Baby();
                Kid.setName(args[0]);
                Kid.setGender(args[1].charAt(0));
                if(Kid.getGender()=='M')
                        System.out.print("MySonis");
                else
                        System.out.print("MyDaughteris");
                System.out.print(Kid.getName());
        }
}
```

```
//ScrollbarDemo.java
import java.awt.*;
public class ScrollbarDemo extends Frame{
        Image image;
        Scrollbar horiz = new Scrollbar(Scrollbar.HORIZONTAL,0,400,0,500);
        Scrollbar vert = new Scrollbar(Scrollbar.VERTICAL,0,400,0,500);
        public static void main(String args[]){
                ScrollbarDemo win = new ScrollbarDemo();
        }
        public ScrollbarDemo(){
                super("Scrollbar");
                addMenus();
                loadImage();
                add("South",horiz);
                add("East",vert);
                pack();
                resize(400,400);
                show();
        }
        void addMenus(){
                MenuBar menubar = new MenuBar();
                Menu file = new Menu("File");
                file.add("Quit");
                menubar.add(file);
                setMenuBar(menubar);
        }
        void loadImage(){
                Toolkit toolkit = getToolkit();
                image = toolkit.getImage("A3041477-11.gif");
                MediaTracker tracker = new MediaTracker(this);
                tracker.addImage(image,7);
                try{
                        tracker.waitForID(7);
                }catch(InterruptedException ex){
                }
        }
        public void paint(Graphics g){
                g.drawImage(image,0-horiz.getValue(),0-vert.getValue(),this);
        }
```

(ต่อ)

(ต่อ)

```
        public boolean handleEvent(Event event){
                if(event.id == Event.WINDOW_DESTROY){
                        System.exit(0);
                        return true;
                }else if(event.id == Event.ACTION_EVENT&&event.target instanceof MenuItem){
                        if("Quit".equals(event.arg)){
                                System.exit(0);
                        }else{
                                return false;
                        }
                }else if(event.target instanceof Scrollbar){
                        repaint();
                        return true;
                }else
                        return false;
        }
```



```
import java.text.DecimalFormat;

public class Complex1{
   private double real;
   private double imaginary;

   public Complex1(){
            this(0.0, 0.0);
   }
   public Complex1(double real, double imaginary){
            this.real     = real;
            this.imaginary = imaginary;
   }
   public double get_real(){
            return this.real;
   }
   public double get_imaginary(){
            return this.imaginary;
   }
   public void c_add(double add_real, double add_imaginary){
            this.real     += add_real;
            this.imaginary += add_imaginary;
   }
   public void c_sub(double sub_real, double sub_imaginary){
            this.real     -= sub_real;
            this.imaginary -= sub_imaginary;
   }
   public void c_mul(double mul_real, double mul_imaginary){
            double real, imaginary;

            real     = this.real * mul_real - this.imaginary * mul_imaginary;
            imaginary = this.real * mul_imaginary + this.imaginary * mul_real;
            this.real     = real;
            this.imaginary = imaginary;
   }
   public void c_display(){
            DecimalFormat fmt = new DecimalFormat("+#,##0.0;-#,##0.0");
            String result = "";
            /* Use this if you are not using the DecimalFormat object above.
            result += this.real;
            if (this.imaginary >= 0)
              result += "+ ";
            result += this.imaginary + "i";
            */
```

(ต่อ)

(ต่อ)

```
            result += fmt.format(this.real);
            result += fmt.format(this.imaginary);
            result += "i";

            System.out.println(result);
    }
}
public class complex1_example{
        public static void main(String s[]){
                int SIZE = 5;

                if (s.length == 1)
                        SIZE = Integer.parseInt(s[0]);

                Complex1 C[] = new Complex1[SIZE];

                for (int i=0; i<SIZE; i++) {
                        C[i] = new Complex1(i+1,i+1);
                        C[i].c_display();
                }

                Complex1 sum     = new Complex1(C[0].get_real(), C[0].get_imaginary());
                Complex1 diff    = new Complex1(C[0].get_real(), C[0].get_imaginary());
                Complex1 product = new Complex1(C[0].get_real(), C[0].get_imaginary());

                for (int i=1; i<SIZE; i++) {
                        sum.c_add(C[i].get_real(), C[i].get_imaginary());
                        diff.c_sub(C[i].get_real(), C[i].get_imaginary());
                        product.c_mul(C[i].get_real(), C[i].get_imaginary());
                }

                System.out.println("The results are: ");
                System.out.print(" sum     = ");
                sum.c_display();
                System.out.print(" diff    = ");
                diff.c_display();
                System.out.print(" product = ");
                product.c_display();

                for (int i=0; i<SIZE; i++)
                        C[i].c_display();

        System.exit(0);
        }
}
```

ภาคผนวก ง
บทความที่ได้รับการตีพิมพ์

การประชุมทางวิชาการทางวิศวกรรมไฟฟ้า (EECON-26)

การประชุมทางวิชาการ ISCIT 2003

การประเมินค่าการใช้ Heap แบบคงที่ของ Java Object

Java's Object Fixed Heap Resources Evaluation

ชริยา นนทกาญจน์ และ อัครินทร์ คุณกิตติ
คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ถนนฉลองกรุง เขตลาดกระบัง กรุงเทพฯ 10520
E-mail : annchariya@yahoo.com , akharin@it.kmitl.ac.th

บทคัดย่อ

การจัดการทรัพยากร(Resource Management) เป็นงานหลักที่สำคัญในระบบคอมพิวเตอร์ จึงได้มีการนำแนวคิดของการประเมินค่าการใช้ทรัพยากรเข้ามาช่วยในการแก้ปัญหาที่เกิดจากการจัดการและการใช้ทรัพยากร ซึ่งงานที่ผ่านมาจะเป็นการวัดการใช้ทรัพยากรในขณะ runtime ซึ่งยังคงมีปัญหาหลายประการ บทความนี้นำเสนอ Framework สำหรับประเมินค่าการใช้ทรัพยากรก่อนการ run (หลังจาก compile) เพื่อหาปริมาณของทรัพยากรในแต่ละประเภทก่อนที่จะทำงานจริง โดยจะเป็นการวัดการใช้ทรัพยากรของ Java object และนำเสนอวิธีการเบื้องต้นและผลการทดลองในการวัดพื้นที่ของ Heap แบบคงที่ (Fixed) โดยนำข้อมูลจาก class fileในส่วนที่เกี่ยวข้องกับการใช้ทรัพยากรมาวิเคราะห์เพื่อประมาณค่าการใช้ทรัพยากรของหน่วยความจำและผลที่ได้จากการวัดตรงกับการวัดขณะ runtime

คำสำคัญ : Java Object, การจัดการทรัพยากร, Framework สำหรับประเมินค่าการใช้ทรัพยากร

Abstract

Resource management is a principle role of computer system. Evaluating amount of resources is a concept that can be used by the system for basic information of resource management. The previous works have focused on resource evaluation at runtime which can cause many problems. This paper proposes a framework for resource evaluation by evaluating resource before runtime and focus on Java objects. This paper also present a basic algorithm and experimental result of fixed evaluating of fixed heap area for Java objects. The algorithm use information that correspond to resource usage from class file and experimentation prove that the result from algorithm has the same as result from runtime tool.

Keywords : Java Object , Resource Management , Resource Evaluation Framework

1. บทนำ

จากความเปลี่ยนแปลงและพัฒนาที่เกิดขึ้นกับระบบคอมพิวเตอร์ ทำให้ปัจจุบันเทคโนโลยีต่าง ๆ ของระบบคอมพิวเตอร์และระบบเครือข่ายมีการเปลี่ยนแปลงและพัฒนาไปสู่รูปแบบใหม่ ๆ อย่างรวดเร็ว ดังนั้นระบบจึงต้องหาวิธีการที่จะนำมาใช้จัดการทรัพยากรให้เหมาะสมกับความเปลี่ยนแปลงที่เกิดขึ้นเพื่อสนองตอบต่อความต้องการของผู้ใช้ โดยเทคโนโลยีทางด้าน Object เป็นเทคโนโลยีที่มีการนำมาใช้งานอย่างกว้างขวางเนื่องจากเหตุผลหลายประการ และ Java เป็นภาษาที่มีการนำมาใช้กับเทคโนโลยีเหล่านี้ โดยเฉพาะอย่างยิ่งเทคโนโลยีทางด้าน Internet เนื่องจาก คุณสมบัติที่สำคัญของ Java เช่น portability และ security โดยเทคโนโลยีที่เข้ามาเกี่ยวข้อง คือ mobile object และ active network ตัวอย่างของการนำ Java มาใช้ เช่น Web Applet , Servlet เป็นต้น ซึ่งมีการใช้งานอย่างแพร่หลายในเครือข่าย Internet แต่การนำ Java มาใช้กับเทคโนโลยีเหล่านี้ยังคงประสบกับปัญหา คือ ไม่มีระบบการจัดการทรัพยากรที่ชัดเจน และเนื่องจากการที่ระบบจะต้องมีการรองรับงานที่มีลักษณะหลากหลายมากขึ้นอาจทำให้มี code หรือ application ที่ไม่เหมาะสมถูกส่งผ่านเข้ามาสู่ระบบ ดังนั้นหากไม่มีการจัดการที่มีประสิทธิภาพเพียงพออาจจะก่อให้เกิดปัญหาขึ้นได้ เช่น การใช้ทรัพยากรไม่เหมาะสมหรือการใช้ทรัพยากรมากเกินไป งานวิจัยที่ผ่านมาได้มีการนำเสนอวิธีการต่าง ๆ ที่จะนำมาใช้แก้ปัญหาในส่วนนี้ เช่น Jres[5] , J-SEAL[6] โดยทั้งสองได้เสนอวิธีการในการควบคุมและจัดการการใช้ทรัพยากร โดยการกำหนดข้อจำกัดในการใช้ทรัพยากรและวัดการใช้ทรัพยากรในแต่ละประเภทของ application เพื่อเปรียบเทียบว่ากำลังจะมีการใช้ทรัพยากรเกินกว่าที่จำกัดไว้หรือไม่ และลักษณะการวัดจะเป็นการวัดการใช้ทรัพยากรในขณะ runtime ซึ่งการวัดในลักษณะนี้ยังอาจทำให้เกิดปัญหา คือ

1. ระบบจะต้องทำการวัดทรัพยากรทุกครั้งที่มีการทำงาน ไม่ว่า application นั้นจะถูกส่งผ่านเข้าสู่ระบบกี่ครั้งก็ตาม ทำให้ต้องสิ้นเปลืองทรัพยากรที่จะนำมาใช้วัดทุกครั้ง

2. ในขณะที่มีการวัดการใช้ทรัพยากรจะมีการใช้ทรัพยากรใน 2 ส่วน คือ ใช้ทรัพยากรเพื่อการทำงานจริงและใช้ทรัพยากรเพื่อการวัด ซึ่งอาจทำให้เกิดการใช้ทรัพยากรเกินไปจากที่กำหนดไว้

3. ในขณะที่ run หากพบว่ากำลังจะมีการใช้ทรัพยากรเกินไปจากที่กำหนดไว้ ระบบอาจจะมีการหยุดการทำงานและเกิดการสูญเปล่าของทรัพยากรที่ใช้มาก่อนหน้า

จากปัญหาดังกล่าว บทความนี้จึงได้นำเสนอวิธีการในการวัดการใช้ทรัพยากรในอีกรูปแบบหนึ่ง ซึ่งเป็นการวัดการใช้ทรัพยากรของ application ก่อนที่จะมีการ run จริง โดยจะเป็นการวัดการใช้ทรัพยากรของ Java object และใช้ข้อมูลจาก class file เพื่อนำมาวิเคราะห์หาปริมาณของทรัพยากรที่จะใช้ในการทำงานบน JVM

การวัดจะเป็นการดึงข้อมูลจาก class file ซึ่งได้จากการ compile Java application นำมาวิเคราะห์เพื่อสร้าง Resource Object และสร้าง Resource Description File(RDF) ที่สอดคล้องกับ Java object แต่ละตัวและนำข้อมูลที่ได้จาก RDF มาใช้เพื่อการประเมินและวัดค่าเพื่อให้ได้ปริมาณของทรัพยากรในแต่ละประเภทที่ application นั้น ใช้ในการทำงาน โดยในขณะที่มีการวัดค่าซึ่งใช้ข้อมูลจาก RDF นั้น ตัว application จริงจะไม่มีการ run ดังนั้นระบบจึงสามารถหาปริมาณของทรัพยากรที่ application นั้นได้ก่อนที่จะมีการ run จริง และนำข้อมูลที่ได้ไปช่วยให้การจัดการทรัพยากรของระบบมีประสิทธิภาพและลดปัญหาที่เกี่ยวกับการใช้ทรัพยากรของระบบและสามารถรองรับความต้องการของผู้ใช้ได้อย่างเหมาะสม

บทความนี้จะเป็นการนำเสนอภาพรวม(Framework) ในกระบวนการของการ evaluation เพื่อประเมินค่าการใช้ทรัพยากรเพื่อนำเสนอกระบวนการในการวัดการใช้ทรัพยากรก่อนการ run จริงของ application และกำหนดการแบ่งประเภทของทรัพยากรที่สอดคล้องกับ Java Object รวมทั้งผลการทดลองเบื้องต้นเพื่อนำผลที่ได้ไปปรับปรุงงานในขั้นตอนต่อไป

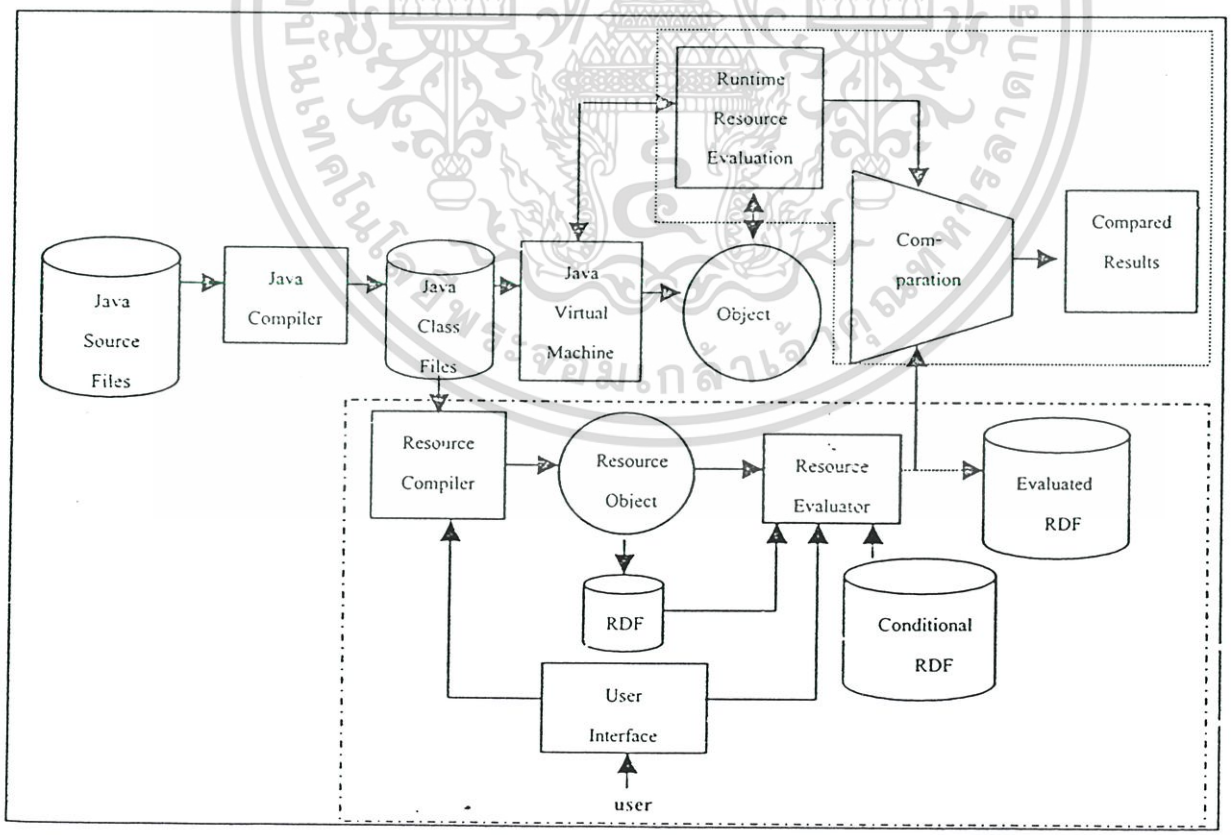
2. Java Resource Evaluation Framework(JREF)

กระบวนการของการวัดและประเมินค่าการใช้ทรัพยากรจะประกอบด้วยส่วนต่าง ๆ ที่สำคัญ ดังต่อไปนี้

2.1 ภาพรวมของ JREF

Java Resource Evaluation Framework(JREF) (รูปที่ 1) เป็นกระบวนการในการประเมินค่าการใช้ทรัพยากรใด ๆ โดย Framework ประกอบด้วยการทำงานตามขั้นตอนหลัก 2 ขั้นตอน คือ

1. Compilation : ขั้นตอนนี้เป็นการนำ class file ที่ได้จาก Java compiler มา compile เพื่อสร้าง Resource Object และ Resource Description File(RDF) ซึ่งจะเก็บรายละเอียดต่าง ๆ ที่เกี่ยวข้องกับการใช้ทรัพยากรในแต่ละประเภทของ object โดย class file จะประกอบด้วยข้อมูลต่าง ๆ ที่สามารถบอกถึงการใช้ทรัพยากรของ object และ byte code instruction ซึ่งมีอยู่ในทุก method ของ class file จะเป็นส่วนที่บอกถึงพฤติกรรมในการใช้ทรัพยากรของ object ซึ่งสามารถนำข้อมูลทั้งหมดไปใช้ในการประเมินค่าการใช้ทรัพยากร นอกจากนี้ในขั้นตอนของ



รูปที่ 1 Java Resource Evaluation Framework

compiler จะประกอบด้วยขั้นตอนย่อย คือ

1.1 ตรวจสอบ RDF ซึ่งจะทำการตรวจสอบว่า class file ที่กำลังวัดมี RDF แล้วหรือไม่ และ RDF นั้นเป็น RDF ที่สอดคล้องกับ class file หรือไม่ ซึ่งถ้าหาก class file มี RDF แล้วก็ไม่ต้องเข้าสู่กระบวนการ compile และข้ามไปทำงานในขั้นตอนของ Evaluation ต่อไป

1.2 Resource Compiler จะสร้าง Resource Object ซึ่งจะเป็น object ที่มีการ run จริงในขณะที่มีการ evaluate ดังนั้นตัว object จริงจึงไม่จำเป็นต้องมีการ run นอกจากนี้ Resource Compiler สร้าง Resource Description File(RDF) ที่สอดคล้องกับ Resource Object เพื่อที่ object ที่จะประเมินค่าการใช้ทรัพยากรไม่จำเป็นที่จะต้องเข้าสู่กระบวนการของการ compile ทุกครั้ง และนำ RDF ที่มีอยู่ไปใช้ในกระบวนการของ Evaluation ได้เลย

2. Evaluation : Resource Evaluator นำ RDF มาเข้าสู่กระบวนการเพื่อให้ได้มาซึ่งค่าปริมาณของทรัพยากรในแต่ละประเภทที่จะใช้ และจะทำงานควบคู่ไปกับ Resource Object ซึ่งทำให้ในขณะที่มีการ evaluate ตัว class file นั้นไม่ต้องทำการ run และในขั้นตอนนี้หาก Object นั้นมีการเรียกใช้ object อื่น ก็จะต้องมีการเรียกใช้ RDF ของ Object ที่ถูกเรียกใช้นั้นมาเพื่อการประเมินเพื่อให้ได้ผลลัพธ์ด้วย นอกจากนี้อาจจะต้องมีการดึงข้อมูลจาก Conditional RDF ซึ่งเก็บค่าเริ่มต้นสำหรับตัวแปรต่าง ๆ ที่จะนำไปใช้ในการ evaluate

การทำงานทั้งใน 2 ขั้นตอนจะถูกควบคุมโดย User Interface ซึ่งจะทำหน้าที่เสมือนเป็น Framework Controller เพื่อควบคุมให้ Framework ทำงานตามที่กำหนด

2.2 Resource Object

Resource Object เป็น object ที่จะ run ไปพร้อม ๆ กับที่มีการดึงข้อมูลจาก RDF ในกระบวนการของ Evaluation โดย Resource Object จะมีโครงสร้างสอดคล้องกับประเภทของทรัพยากร โดยบทความนี้นำเสนอการวัดทรัพยากรประเภทหน่วยความจำซึ่งสามารถแบ่งได้ดังนี้

1. MethodArea : เป็นส่วนที่เก็บข้อมูลต่าง ๆ ของ class
2. HeapArea : เป็นพื้นที่ของ object
3. StackArea : เป็นพื้นที่ในการทำงานของ method

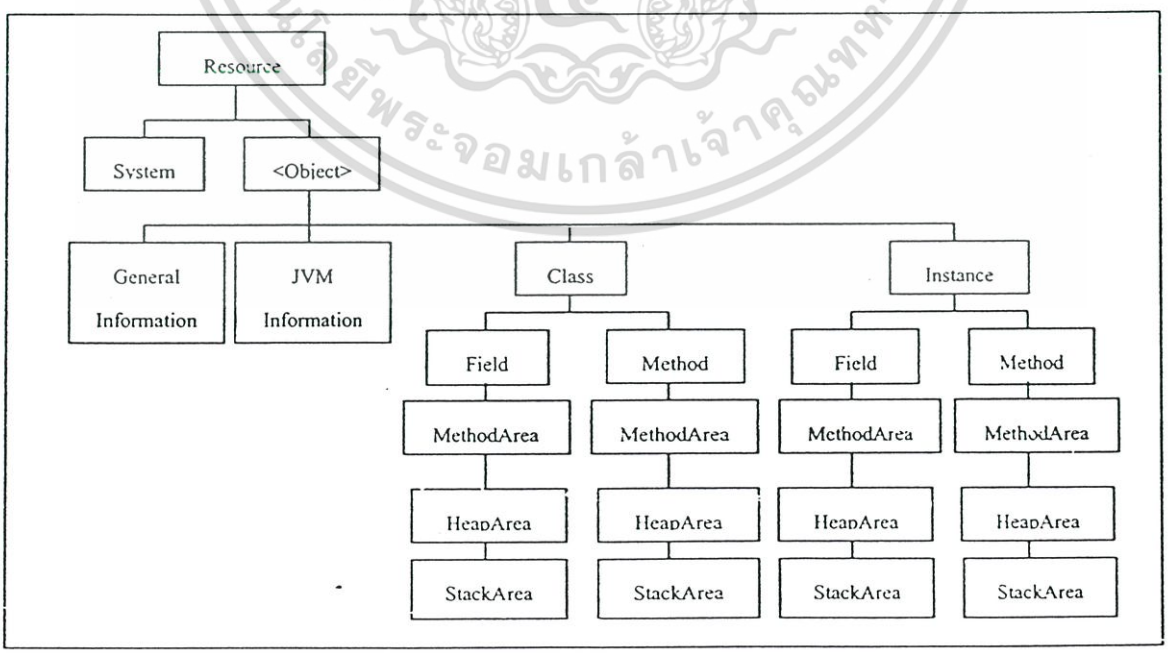
โดย Resource Object จะมีโครงสร้างดังรูป 2

2.3 Resource Description File(RDF)

เป็น file ผลลัพธ์จากการ Compile ซึ่งเก็บรายละเอียดของทรัพยากรเพื่อ object โดยภายใน RDF อาจจะประกอบด้วยค่าต่าง ๆ ที่เป็นค่าคงที่ที่ได้จาก class file หรืออาจจะอยู่ในรูปแบบของ resource expression ที่ต้องมีการติดค่าของตัวแปรไว้ และจะต้องทำการประมวลผลเพื่อหาผลลัพธ์ ซึ่งโดยส่วนใหญ่แล้วค่าของทรัพยากรต้องการวัดจะอยู่ในรูปของ resource expression ที่จะต้อง evaluate เพื่อได้ค่าการใช้ทรัพยากรที่แท้จริงที่สอดคล้องกับเงื่อนไขในการทำงานของ object โดย RDF ถูกสร้างขึ้นมาจาก Resource Object ที่ได้จากการ compile และจะเขียนด้วย XML และจะมีโครงสร้างดังรูปที่ 2

3. การวัดการใช้พื้นที่ของ Heap แบบคงที่ (Fixed)

สำหรับส่วนนี้จะเสนอ algorithm สำหรับวัดการใช้พื้นที่ heap ของ Java object ซึ่งจะเป็นการวัดแบบคงที่(Fixed) โดย Java object ที่มี object 2 ประเภท คือ



รูปที่ 2 Resource Information Classification

1. พื้นที่ของ object จะคำนวณจากผลรวมของพื้นที่ของ field ที่ modifier ไม่เป็น static (Instance variable) ทั้งหมดของ class ของ Object และ overhead (8 bytes)(ใช้ขนาดจาก ตารางที่1)โดยมี algorithm

```
heaparea = 0
for i = 1 to n do /*n is number of fields of class */
    if(modifier of field is not "static") then
        x = primitive area of field
        heaparea = heaparea + x
    endif
endfor
heaparea = heaparea + 8
```

รูปที่ 3 Algorithm วัด heap สำหรับเก็บข้อมูลของ object

2. พื้นที่ของ array จะคำนวณจากความยาว , มิติ และ พื้นที่ของ primitive type ของ array (ตารางที่ 1) และ overhead (8 bytes)

```
check dimension , length and primitive type of array
heaparea = 0
for i = 1 to n do /* n is number of dimention of array*/
    x = calculate area of primitive and length  *area of each dimension*/
    heaparea = heaparea + x
endfor
heaparea =heaparea + 8
```

รูปที่ 4 Algorithm วัด heap สำหรับเก็บข้อมูลของ array

ตารางที่ 1 พื้นที่ของ primitive type[2]

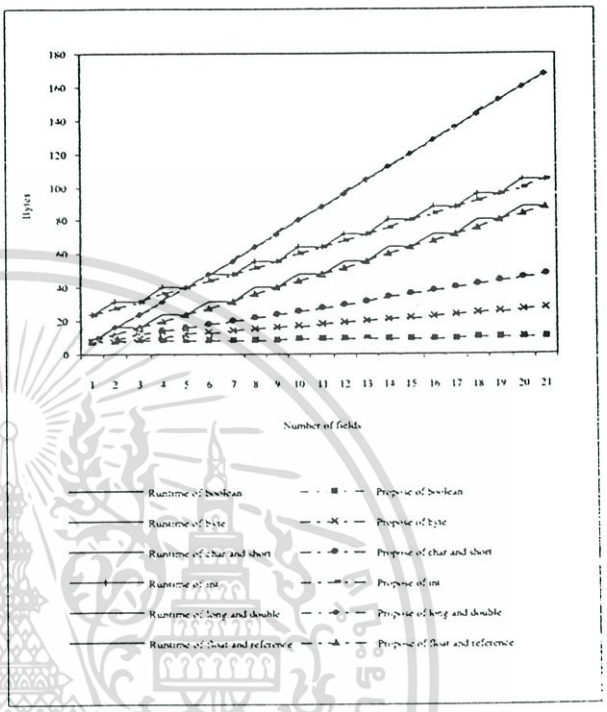
Type	Size
Boolean	1 bit
Byte	1 byte
Char	2 bytes
Short	2 bytes
Int	4 bytes
Long	8 bytes
Float	4 bytes
Double	8 bytes
reference	4 bytes

4. การทดลอง

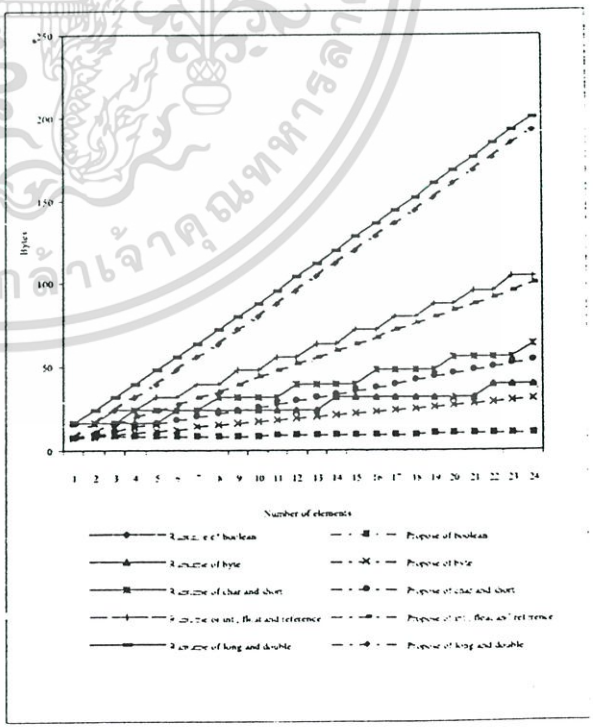
การทดลองเบื้องต้นจะเป็นการเปรียบเทียบค่าของทรัพยากรที่ได้จากการใช้ Tool ที่วัดการใช้ทรัพยากรขณะ runtime [8] ซึ่งใช้ method จาก package Java.lang.Runtime กับ algorithm ที่นำเสนอ โดยการทดลองจะเป็นการ run Java object บน J.D.K 1.3.0_02 ซึ่งเป็นการวัดในลักษณะคงที่ (Fixed) คือ ดึงค่าโดยตรงจาก class file และทำการ evaluate (โดยไม่ต้องเข้าสู่กระบวนการ compile) โดยในการวัดจะประกอบด้วย

1. object จะทดลองโดยกำหนด class และเพิ่ม field ชนิดต่าง ๆ ให้กับ class นั้นและสร้าง object จาก class

2. array ทดลองโดยสร้าง array ของ type ต่าง ๆ และเพิ่ม element ครั้งละ 1 element และจะได้ผลการทดลอง ดังรูปที่ 5 และ 6



รูปที่ 5 กราฟเปรียบเทียบการใช้heap สำหรับเก็บข้อมูลของ object



รูปที่ 6 กราฟเปรียบเทียบการใช้ heap สำหรับเก็บข้อมูลของ array

จากกราฟ จะสามารถสรุปได้ดังนี้

1. object (Instance ของ class)
   - มี overhead เริ่มต้น 8 bytes
   - การ allocate จะมีลักษณะ 8-bytes block
   - การ allocate สำหรับ field ที่มี primitive type น้อยกว่า 8 จะใช้พื้นที่ 2 fields ต่อ 1 byte
2. array
   - มี overhead เริ่มต้นที่ 16 bytes
   - การ allocate จะมีลักษณะ 8-bytes block คือ จะ allocate ครั้งละ 8 bytes ถึงแม้ว่า field นั้นจะไม่ได้มีขนาดเท่ากับ 8 bytes ก็ตาม เช่นเดียวกับ Object
   - หลังจาก allocate 8 bytes แล้วเมื่อมีการเพิ่ม element จะใช้พื้นที่ตามขนาดของ primitive type

ดังนั้นหากปรับ algorithm ให้สอดคล้องกับผลลัพธ์จาก Runtime Tool จะได้ algorithm ต่อไปนี้

```
heaparea = 0
count = 0
sum = 0
for i = 1 to n do /*n is number of fields of class */
    if(modifier of field is not "static") then
        x = primitive area of field
        sum = sum + x
        if (type is not long or double) then
            count = count + 1
        endif
        a = count % 2
        if(a > 0) then
            sum = sum + 4
        endif
    endif
    heaparea = heaparea - 8
endfor
```

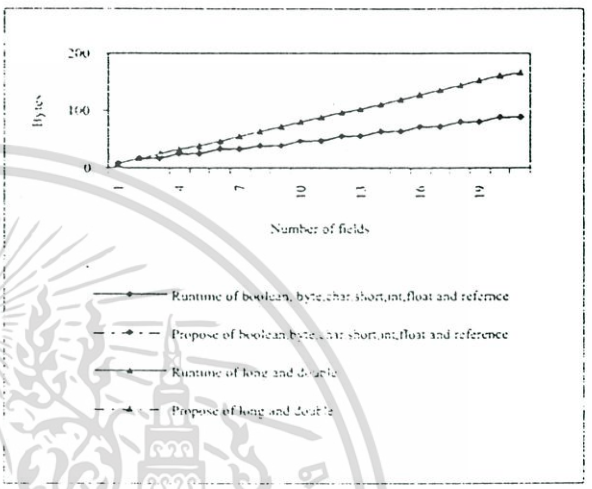
รูปที่ 7 Algorithm ที่ปรับปรุงสำหรับวัดheap สำหรับเก็บข้อมูลของ object

```
check dimension , length and primitive type of array
heaparea = 0
for i = 1 to n do /* n is number of dimention of array*/
    x = area of dimension /* calculate by check type and condition of each
        type and length */
    heaparea = heaparea + x
endfor
heaparea =heaparea + 16
```

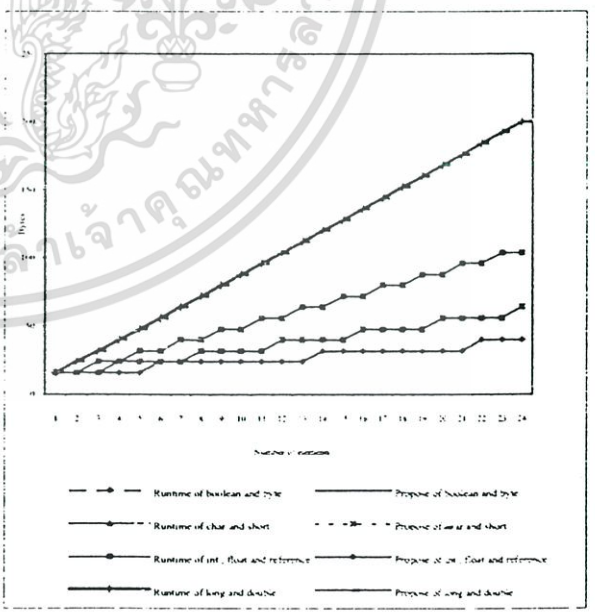
รูปที่ 8 Algorithm ที่ปรับปรุงสำหรับวัด heap สำหรับเก็บข้อมูลของ array

เมื่อทดลองโดยการใช้ algorithm ที่ปรับปรุงนี้ก็จะได้ผลดังรูปที่ 9 และ 10

จากรูปที่ 9 และ 10 จะเห็นได้ว่าเมื่อมีการปรับปรุง algorithm ก็จะสามารถวัดการใช้ทรัพยากรได้ตรงกับการวัดในขณะ runtime และสามารถแก้ปัญหาที่เกิดจากการวัดในขณะ runtime ได้ โดยการปรับ algorithm จะสอดคล้องกับเงื่อนไขต่าง ๆ ที่สรุปได้จากผลการทดลองข้างต้น



รูปที่ 9 กราฟเปรียบการใช้ heap สำหรับเก็บข้อมูลของ object หลังปรับ algorithm



รูปที่ 10 กราฟเปรียบการใช้ heap สำหรับเก็บข้อมูลของ array หลังปรับ algorithm

และจากผลการทดลองจะพบว่าสามารถปรับ algorithm ที่ใช้สำหรับวัดซึ่งให้ผลที่ตรงกันกับการวัดขณะ runtime ได้ จากจุดนี้จะเห็นได้ว่าการวัดค่าการใช้ทรัพยากรก่อนการ run สามารถให้ผลลัพธ์ออกมาตรงกับการวัดในขณะ runtime

5.สรุป

การจัดการทรัพยากรเป็นภาระหลักที่สำคัญของระบบคอมพิวเตอร์และระบบเครือข่ายเพื่อให้สามารถตอบสนองต่อความต้องการของผู้ใช้ได้อย่างเหมาะสม ระบบจึงมีความจำเป็นอย่างยิ่งที่จะต้องมีการจัดการทรัพยากรให้มีประสิทธิภาพและมีความปลอดภัยในการใช้ทรัพยากร ซึ่งการวัดปริมาณของทรัพยากรที่จะใช้ในการทำงานเป็นแนวคิดหนึ่งที่สามารถนำมาปรับใช้กับการจัดการทรัพยากรเพื่อนำข้อมูลที่ได้ไปใช้ในการจัดการทรัพยากรให้มีประสิทธิภาพได้

บทความนี้ได้นำเสนอแนวคิดของการวัดและประเมินค่าการใช้ทรัพยากรของ Java object โดยจะเป็นการวัดการใช้ทรัพยากรก่อนที่มีการ run จริงเพื่อแก้ปัญหาดังที่ได้กล่าวไว้เบื้องต้น และการวัดจะใช้ข้อมูลจาก class file มาสร้างเป็น Resource Object และ RDF และประเมินค่าจาก RDF เพื่อให้ได้ปริมาณของทรัพยากรที่ object ใด ๆ จะใช้ในการทำงาน โดยในบทความนี้ได้นำเสนอภาพรวมในการทำงานที่สามารถนำไปใช้วัดค่าทรัพยากรได้ทุกประเภท และการแบ่งประเภทของทรัพยากรที่สอดคล้องกับการทำงานของ object นอกจากนี้ได้นำเสนอวิธีการวัดการใช้พื้นที่ heap ของ object แบบคงที่และจากผลการทดลอง จะเห็นได้ว่าสามารถปรับปรุง algorithm ให้สอดคล้องกับการทำงานจริงในขณะ runtime ได้ ซึ่งงานในอนาคตจะเป็นการปรับปรุงการทำงานตาม Framework เพื่อให้การทำงานมีลักษณะ dynamic และ สามารถวัดปริมาณการใช้ทรัพยากรในแต่ละประเภทของ object ใด ๆ และนำผลที่ได้เปรียบเทียบกับผลที่ได้จากการวัดในขณะที่มีการ run จริงเพื่อหาปัญหาที่เกิดขึ้นเพื่อพัฒนาระบบงานต่อไป

เอกสารอ้างอิง

[1] John W. Satzinger and Tore U. Orvik.The Object-Oriented Approach concept modeling and system development. Boyd & fraser publishing company,1996.

[2] Tim Lindholm and Frank Yellin.The Java Virtual Machine Specification: The Java Series.Sun Microsystem, Inc, 1997.

[3] Bill Venners. Inside the JAVA Virtual Machine. McGraw-Hill , 1998.

[4] Markus Dahm. "Byte Code Engineering with the JavaClass API".Freie University Berlin .July,1999.

[5] Grzegorz Czajkowski and Thorsten von Eicken, "Jres: A Resource Accounting Interface for Java",In Preceeding of the 1998 ACM OOPSLA Conference Vancouver , BC ,October,1998.

[6] Walter Binder, Jarle G. Hulaas and Alex Villazon,"Portable Resource Control in Java: The J-SEAL2 Approach",CoCo Software Engineering Gmbll and the Swiss National Science Foundation grants.

[7] Mirza Beg and Mike Dablin, "A Memory Accounting Interface for The Java Programming Lenguage", The University of Taxas at Austin, Department of Computer Sciences. Technical Report CS TR 01 40.October, 2001.

[8] www.Javaworld.com/Javaworld/Javatips/jw-javatip130.html

อัครินทร์ คุณกิตติ : อาจารย์ประจำคณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

งานวิจัย : - Computer Systems
- Distributed Systems
- Data Communication and Network

ชริยา นนทกาญจน์ : จบปริญญาตรีสาขาคอมพิวเตอร์ธุรกิจ มหาวิทยาลัยสงขลานครินทร์ ปัจจุบัน นักศึกษาระดับปริญญาโท คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

งานวิจัย : Java Resource Evaluation

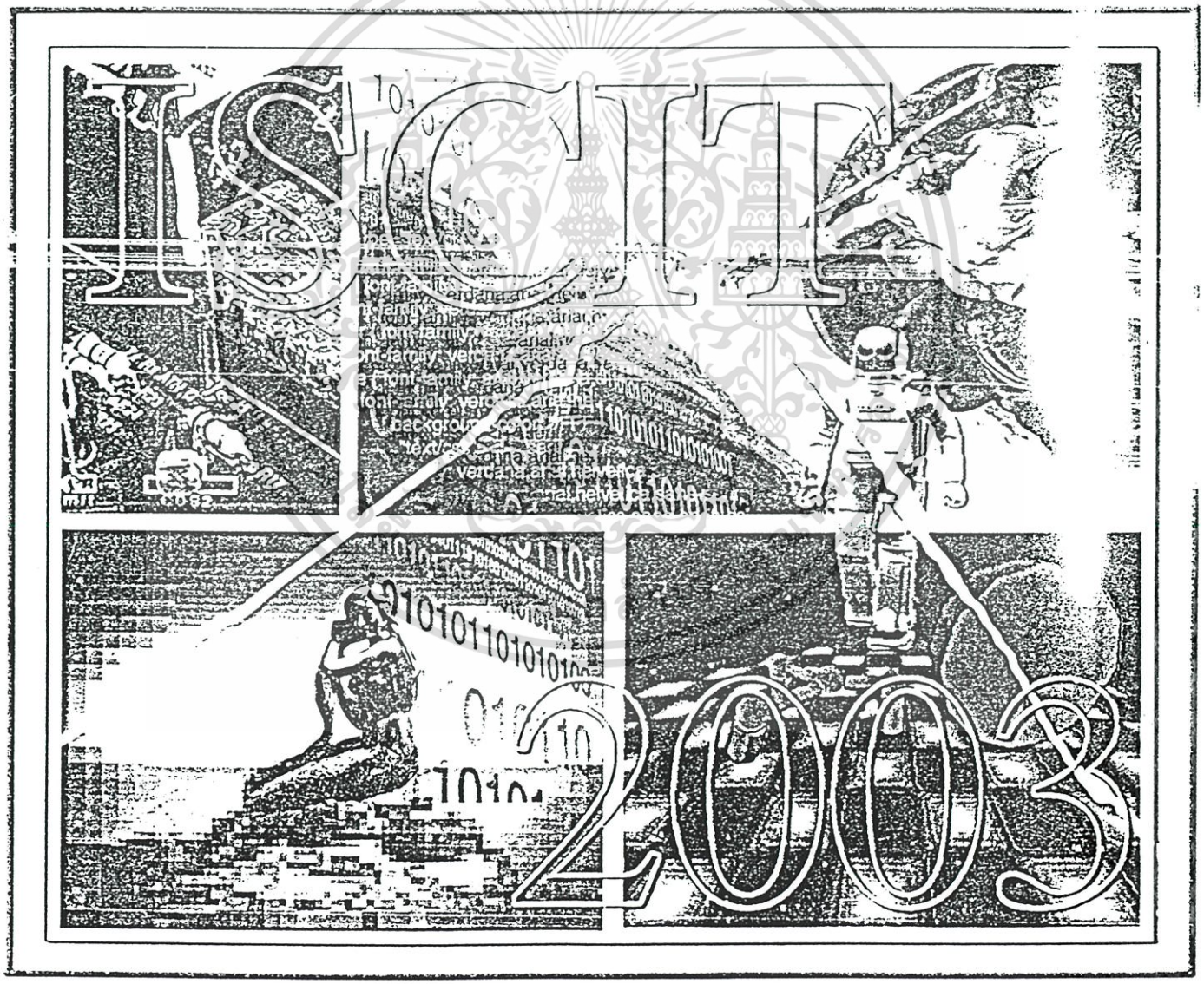
Volume I

Proceedings

The Third International Symposium on Communications and Information Technologie

September 3-5, 2003

BP Samila Beach Hotel and Resort, Songkhla, Thailand



Java's Object Resource Evaluation

Chariya Nonthakarn and Akharin Khunkitti

Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang
Ladkrabang , Bangkok 10520 , Thailland
Email: annchariya@yahoo.com , Akharin@it.kmitl.ac.th

Abstract

Resource management is a principle role of computer system. The system must use management strategies which can make the best quality of services. Evaluating amount of resources is a concept that can be used by the system for basic information of resource management. The system can use such information to make decision support. Previous works have focused on resource evaluation at runtime which may cause many problems. This paper proposes a framework for resource evaluation by evaluating resource before runtime for Java objects. This approach is analyzing of Java class file and evaluating resource that will be used on the Java Virtual Machine (JVM). The system can use resource information result for the efficiency of resource management.

1. Introduction

Recently, technology of computers and network systems has widely developed for new architectures and structures. Consequently the user's requirements have changed. The system must choose management strategies which appropriate to system's architecture and correspond to user's requirement. From this, many researchers have proposed many approaches and techniques which suitable for the variety of architectures and technologies. Those techniques have tried to solve problems of resource management for the best quality of services. But the resources have limitations and the requirement has dramatically increasing and changed to new methodologies. So, it is difficult to make the most appropriate strategy. The basic concept which make resource management more efficiency is evaluation of resources before they are used. This concept mean that if the system know the amount of resources will be used by current application, the system can make resource management work in appropriate way. For example, it can make decision that the system will accept an application or not because of the application will use resources succeed system available resources. Beside that the system can use information from evaluation to work in the way that correspond to user's requirement for the whole efficiency.

There are many previous works have studied and proposed approaches and techniques for resource evaluation. For example, Jres[7] , J-SEAL[8] , J-sprint[12]. These works have proposed approaches for evaluating resources at runtime by specifying the limitation of resources that can be used by system, keep current resources that is in used and evaluate resources of the current application .Then use evaluation results to compare with current resources and recognize the application will use resource succeed the limitation or not and make decision that the system will or will not accept that application. These approaches evaluate resources while the application is running. Although runtime resource evaluating can get amount of resources that really use in any given time but it continue encounter many problems such as :

1. System must evaluate resources every time that the same application enters to the system.
2. While evaluating resource ,the system will use 2 types of resources :
   2.1 resources for tasks of current application
   2.2 resources for evaluation
   These may cause suceeeding of resource limitation
3. When system recognize that the current application will use too large amount of resources .It may decided to disclaim such application and cause the system fail from continuing task.

From discuss above, this paper proposes a new approach for evaluating resource by evaluating resource before runtime of Java's object. Because Java has many properties which appropriate with several work areas. For example, Internet technology which uses Java in technology of Web Applet, Servlet, mobile object, active network, etc. This work will evaluate resources of Java's object that will be used on Java Virtual Machine (JVM). The evaluation will be made by analyzing Java class file for retrieving information to create Resource Description File (RDF) and Resource Object (RO) of evaluated object. Then use that RO and RDF for evaluating resources. The main idea is while evaluating resource object which being evaluated is not necessary to run.

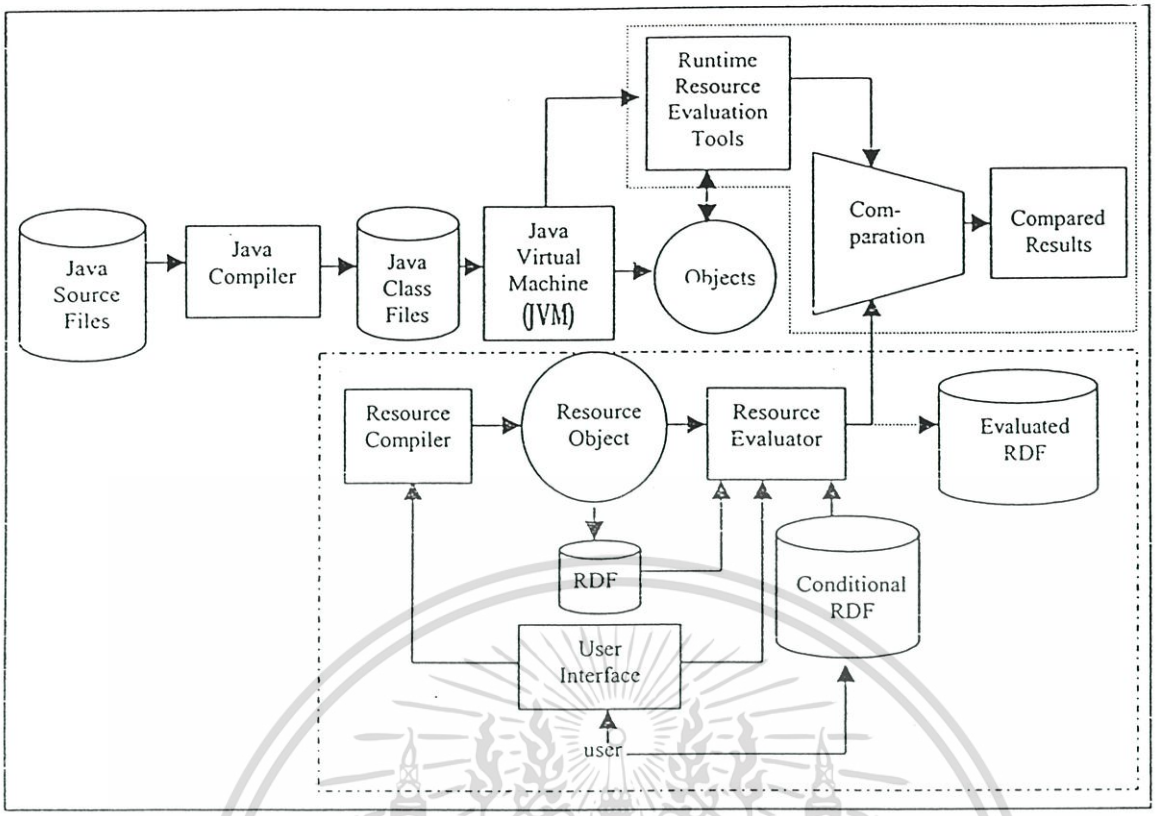


Figure1. Java's object resource evaluation framework

The rest of the paper will be structured as following. In the next section we propose a framework of Java's object resource evaluation. This is followed by structure of Resource Objects and Resource Description File (RDF). The last section summarizes paper and future work.

2. Design Strategy

This section will propose a framework of Java's object resource evaluation in details, resource classification according to structure of Resource Object (RO) , object coordination and Resource Description File (RDF).

2.1 A Framework of Java's object resource evaluation

This framework has been designed for evaluating any kinds of resources of Java's object (figure 1) and the results from our approach will be compared to the results from Runtime Resource Evaluation Tool. Our framework has 2 main phases :

Phase 1 Resource Compilation

The class file from Java compiler will work on Java Virtual Machine (JVM) and contain of many useful information, especially byte code instructions of every methods which tell us what behaviors of object that will occur at runtime.

We can use theses byte code instructions for prediction of what kind of the object will do or what kind of resource that will be used. In this phase, we use class file as input of resource compilation process which are:

1. The system must check if current object has RDF or not. It the object already has RDF and RDF is the current version of class file it is not has to compile for RDF, otherwise do next.
2. Resource Compiler creates Resource Object and Resource Description File (RDF) that keep information of class file according to the structure of Resource Object. Class file will be compiled in form of resource expression for further evaluation step.

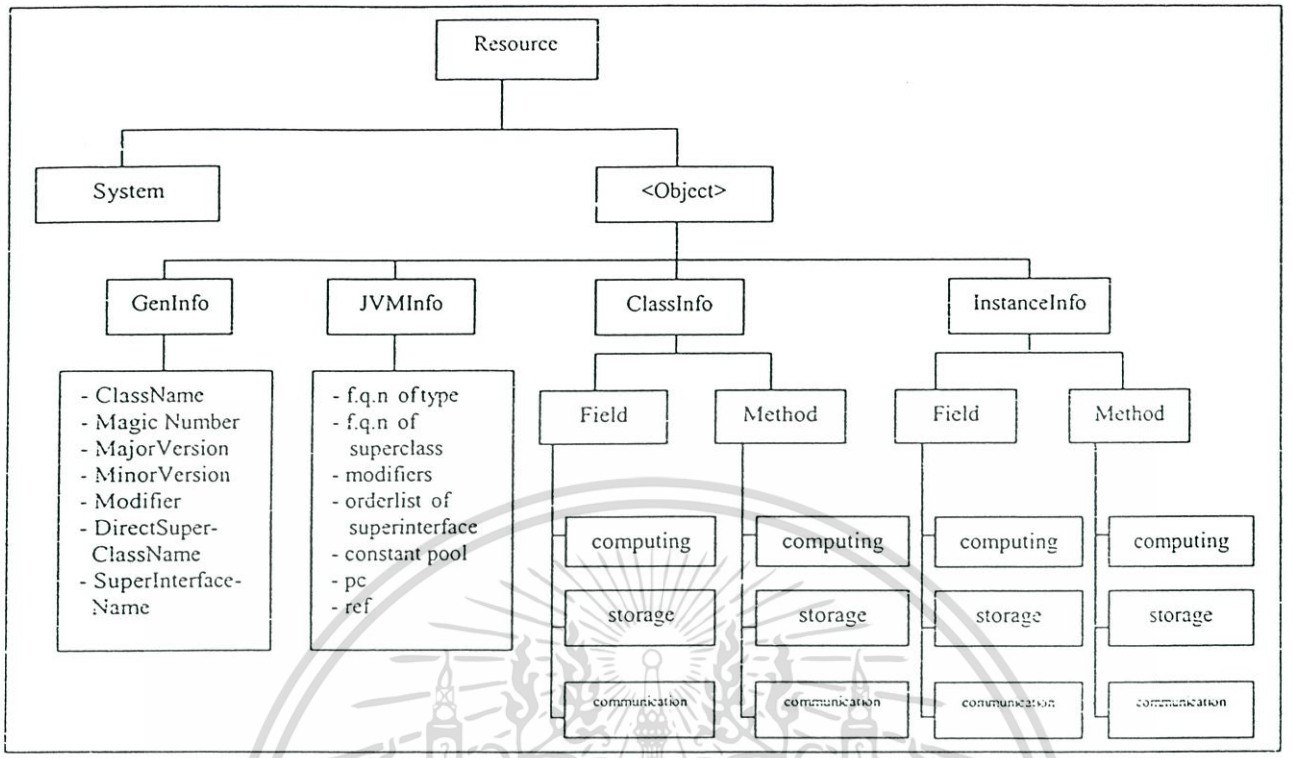


Figure2. Resource information classification

Phase2. Resource Evaluation

This step use information from RDF and work together with Resource Object by retrieving information from RDF which contain resource expressions.

Such expressions will be evaluated by Resource Evaluator. The process will use data (variables) from conditional RDF and other RDF which referenced by object as inputs. That is while evaluating the class files are not need to run and the running is Resource Object.

These 2 phases will be controlled by User Interface which make decision that the system will compile or evaluate (in case of object already has RDF) and create conditional RDF for evaluating process.

2.2 Resource Objects(RO)

Resource is a companion of object being evaluated and will work together with resource evaluator process. The resource compilation process will use RO to make Resource Description File (RDF) and resource evaluation process will load RDF and work with RO of that RDF (figure3)

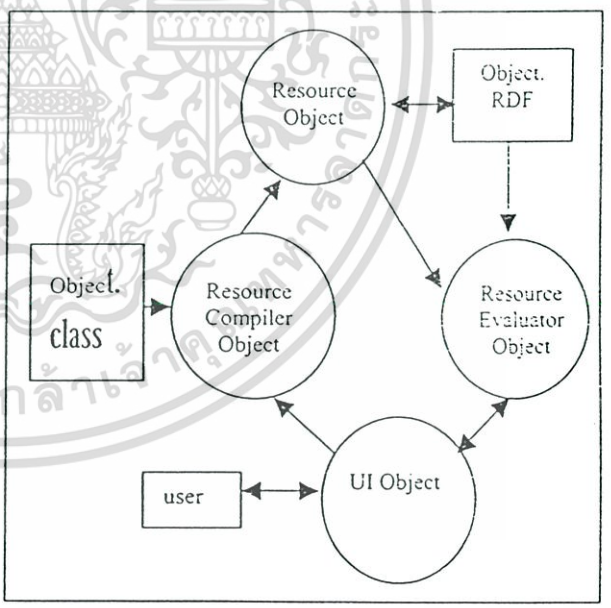


Figure 3 Object coordination

The RO can be structured according to the classification of any kind of resources.
In this propose we can classify resource as following (figure 2):

1. Computing Resource: resource that will be used for computing process e.g. processor.
2. Storage Resource: used for store information e.g. memory
3. Communication Resource: used for transition information e.g. network interface, links.

We can create Resource Object which has structure corresponding to resource classification .This mean that every Resource Object will have the deepest object as structured.
In evaluation process, Resource Object may reference to other Resource Object (as well as object reference to other objects) to retrieve information for final results of evaluating.

2.3 Resource Description File(RDF)

Resource Description File is the result file from compilation that keep details of object's resource that object not has to take into resource compilation process every times it enter to system . RDF may include of information in 2 formats:

1. Constants which directly get from class file .For example, name of class file.
2. Resource expressions; in this work all resource that will be evaluated is in form of expression which has variables and will be construct as conditions which detected by Resource Compiler. The variables will be in form which can be referenced. For example. <object>.x which mean variable x of object. The resource expressions are created during resource compilation and evaluated in evaluation process.

RDF will be written on XML and has structure as same as structure of Resource Object (see Figure4).

```
<?xml version="1.0" encoding="windows-874"?>

<Object>
    <GenInfo>
            <classname>...< classname>
            ...
    < GenInfo>

    <JVMInfo>
            <constantpool>...< constantpool>
            ...
    < JVMInfo>
```

```
    <ClassInfo>
        <Field>
            <PrimResource>
                <computing>...<\computing>
                <storage>...<\storage>
                <communicate>...<\communicate>
            <\PrimResource>
        <\Field>
        <Method>
            <PrimResource>
                <computing>...<\computing>
                <storage>...<\storage>
                <communicate>...<\communicate>
            <\PrimResource>
        <\Method>
    <\ClassInfo>

    <InstanceInfo>
            ...
    <InstanceInfo>
<\Object>
```

Figure 4. RDF example

3. Conclusions and future work

Resource management is principle role of computer system. The system must choose strategy that appropriate with system's architecture. The concept of evaluation resource can make resource management work more efficiency. That is system will use information from evaluation to determine that it will accept object or not. This result will make appropriate use of resources. This paper proposes a framework of Java's object resource evaluation which contain of 2 main phases: resource compilation and resource evaluation. Resource Object and Resource Description File are principle components of this framework. The future, we will work according to the framework proposed which can evaluate any styles of object and will focus on heap area and stack area of Java object and get the result to compare with the result from runtime tool to improve algorithm for the correctness of results.

4. References

[1] John W. Satzinger and Tore U. Orvik.The Object-Oriented Approach concept modeling and system development. Boyd & fraser publishing company,1996.

[2] Jan L. Harrington. Java Programming: An IS Perspective . John Wiley & Sons , Inc ,1998.

[3] Mary Campione and Kathy Walrath. The Java™ Tutorial: object-oriented programming for the internet. 1996.

[4] Tim Lindholm and Frank Yellin.The Java Virtual Machine Specification: The Java Series.Sun Microsystem, Inc, 1997.

[5] Bill Venners. Inside the JAVA Virtual Machine. McGraw-Hill , 1998.

[6] Markus Dahm. "Byte Code Engineering with the JavaClass API".Freie University Berlin .July,1999.

[7] Grzegorz Czajkowski and Thorsten von Eicken, "Jres: A Resource Accounting Interface for Java",In Preceeding of the 1998 ACM OOPSLA Conference Vancouver , BC ,October,1998.

[8] Walter Binder, Jarle G. Hulaas and Alex Villazon,"Portable Resource Control in Java: The J-SEAL2 Approach",CoCo Software Engineering GmbH and the Swiss National Science Foundation grants.

[9] Akharin Khunkitti and Ruttikorn Varakulsiripunth, "SERVICE ENHANCEMENT USING FLEXIBLE SYSTEM FOR NETWORK MANAGEMENT",In Preceeding of the 2000 Symposium on Theory and Applications and Information Technology, August, 2000.

[10] Silvano Maffeis, "Adding Group Communication and Fault-Tolarence to CORBA",In Preceeding of the USENIX Conference on Object Oriented Technologies,C.A,June , 1995.

[11] Mirza Beg and Mike Dablin, "A Memory Accounting Interface for The Java Programming Lenguage", The University of Taxas at Austin, Department of Computer Sciences. Technical Report CS TR 01 40.October, 2001.

[12] http://www.j-sprint.com.

ประวัติผู้เขียน

ชื่อ – นามสกุล	นางชริยา นนทกาญจน์
วัน เดือน ปีเกิด	1 กุมภาพันธ์ 2518
ที่อยู่	18/1 หมู่ 4 ตำบล ชัยบุรี อำเภอเมือง จังหวัด พัทลุง 93000 โทร.01-3087694
ประวัติการศึกษา	จบการศึกษาระดับปริญญาตรีจากคณะวิทยาการจัดการ(คอมพิวเตอร์ธุรกิจ) มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตหาดใหญ่ ปีการศึกษา 2539
ประสบการณ์ทำงาน	2539 – ปัจจุบัน เป็นอาจารย์ประจำแผนกคอมพิวเตอร์ธุรกิจ คณะวิชาเศรษฐศาสตร์และบริหารธุรกิจ สถาบันเทคโนโลยีราชมงคล วิทยาเขตนครศรีธรรมราช