การออกแบบและพัฒนาเทคนิคการเขียนโปรแกรมจาวาโดยใช้ตัวแปรเมทอด

นาย วชิราวุธ ธรรมวิเศษ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2543

ISBN 974-346-474-3

A DESIGN AND DEVELOPMENT OF A JAVA PROGRAMMING TECHNIQUE BY THE METHOD VARIABLE

MR. WACHIRAWUT THAMVISET

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2000

ISBN 974-346-474-3

หัวข้อวิทยานิพนธ์	การออกแบบและพัฒนาเทคนิคการเขียนโปรแกรมจาวาโดยใช้ตัวแปรเมทอด
โดย	นาย วชิราวุธ ธรรมวิเศษ
ภาควิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษา	รองศาสตราจารย์ ดร.วันชัย ริ้วไพบูลย์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

…….. คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สมศักดิ์ ปัญญาแก้ว)

คณะกรรมการสอบวิทยานิพนธ์

…….. ประธานกรรมการ
(อาจารย์ ดร.พรศิริ หมื่นไชยศรี)

…….. อาจารย์ที่ปรึกษา
(รองศาสตราจารย์ ดร.วันชัย ริ้วไพบูลย์)

…….. กรรมการ
(อาจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา)

…….. กรรมการ
(ผู้ช่วยศาสตราจารย์ วิวัฒน์ วัฒนาวุฒิ)

วชิราวุธ ธรรมวิเศษ : การออกแบบและพัฒนาเทคนิคการเขียนโปรแกรมจาวาโดยใช้ตัวแปรเมทอด. (A DESIGN AND DEVELOPMENT OF A JAVA PROGRAMMING TECHNIQUE BY THE METHOD VARIABLE) อ. ที่ปรึกษา : รศ.ดร.วันชัย ริ้วไพบูลย์, 86 หน้า. ISBN 974-346-474-3.

ตัวแปรเมทอดถูกพัฒนาด้วยจาวา โดยเป็นตัวแปรแบบออบเจคต์ที่สามารถเก็บข้อมูลเพื่ออ้างถึงเมทอดภายในออบเจคต์ใดๆ ตัวแปรเมทอดจะไม่พิจารณาชนิดและคลาสของออบเจคต์ แต่จะพิจารณาที่ชนิดของพารามิเตอร์ของเมทอดที่ถูกอ้างถึง ซึ่งเมทอดนั้นสามารถถูกเรียกให้ทำงานได้ผ่านทางฟังก์ชันที่กำหนดในตัวแปรเมทอด ทำให้การเรียกใช้เมทอดมีความยืดหยุ่นสูง สามารถนำตัวแปรเมทอดไปใช้ในการสร้างส่วนควบคุมเหตุการณ์และใช้ในการเชื่อมต่อระหว่างคอมโพเนนท์ในแบบจำลองคอมโพเนนท์ ในงานวิจัยนี้นำเสนอการใช้ตัวแปรเมทอดเพื่อพัฒนาเทคนิคการเขียนโปรแกรมจาวาให้สามารถสนับสนุนการใช้งานตัวแปรเมทอดในแบบต่าง ๆ ได้แก่ ตัวแปรเมทอดแบบเดี่ยว ตัวแปรเมทอดแบบหลายจำนวน การเก็บข้อมูลตัวแปรเมทอดในไฟล์และตัวแปรเมทอดสำหรับออบเจคต์ในฐานข้อมูลเชิงวัตถุ

ในงานวิจัยได้สร้างโครงร่างพัฒนาโปรแกรมโดยใช้ตัวแปรเมทอดสำหรับจาวา ซึ่งจะประกอบด้วย เครื่องมือสำหรับสร้างคลาสตัวแปรเมทอด กลุ่มของคลาสตัวแปรสำหรับการสร้างเหตุการณ์ และกลุ่มของคลาสคอมโพเนนท์ จากการทดสอบพบว่าการเขียนโปรแกรมในกรณีเดียวกันด้วยเทคนิคตัวแปรเมทอดจะมีการเขียนโปรแกรมที่สั้นกว่าการเขียนโปรแกรมด้วยวิธีการของจาวามาตรฐาน 1.1

ภาควิชา .....วิศวกรรมคอมพิวเตอร์.....	ลายมือชื่อนิสิต .....
สาขาวิชา.....วิทยาศาสตร์คอมพิวเตอร์....	ลายมือชื่ออาจารย์ที่ปรึกษา .....
ปีการศึกษา .....2543.....	ลายมือชื่ออาจารย์ที่ปรึกษาร่วม .....

4170493421 : MAJOR COMPUTERSCIENCE

KEY WORD: OBJECT / METHOD VARIABLE/ DELEGATION/ COMPONENT / EVENT

WACHIRAWUT THAMVISET: THESIS TITLE. (A DESIGN AND DEVELOPMENT OF A JAVA PROGRAMMING TECHNIQUE BY THE METHOD VARIABLE) THESIS ADVISOR: ASSOC. PROF. WANCHAI RIVEPIBOON, Ph.D, 86 pp.

ISBN 974-346-474-3.

A Method Variable is developed with Java. The Method Variable can store reference to any object method. Method Variable does not consider a class of object. But it considers the type of parameters of that method. The referenced method can be invoked passing a function of method variable. So the method invocation will be more flexibility. A programmer can use method variable to implement the event handling and component composition in the component model. This research presents the method variable technique for Java programming to support single method variable, multiple method variable, storing method variable in file and using method variable for object in an Object Orient Database.

This research constructs an Application Development Framework for Java by using method variable. This framework includes tool for construct a class of method variable, group of method variable classes for event handling and group of component classes. In the same application domain of the experiments, the source codes created by method variable technique are shorter than standard Java 1.1.

Department …..Computer Engineering…. Student's signature …….

Field of study….Computer Science…….. Advisor's signature …….

Academic Year……2000……. Co-advisor's signature ……

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ได้สำเร็จลุล่วงไปได้ ด้วยความช่วยเหลืออย่างดียิ่งของ รองศาสตราจารย์ ดร. วันชัย ริ้วไพบูลย์ อาจารย์ที่ปรึกษา และ ขอขอบคุณ อ.ดร. พรศิริ หมื่นไชยศรี อ.ดร. ทวิตีย์ เสนีวงศ์ ณ อยุธยา และ ผศ. วิวัฒน์ วัฒนาวุฒิ กรรมการวิทยานิพนธ์ที่กรุณาเสียสละเวลาให้คำแนะนำ ตรวจและแก้ไขต้นฉบับวิทยานิพนธ์

ขอขอบคุณ อ.พิชโยทัย มหัทธนาภิวัฒน์ และ เพื่อน ๆ ในห้องปฏิบัติการวิศวกรรมซอฟต์แวร์ที่เสียสละเวลาในการให้คำปรึกษา และ ช่วยตรวจสอบผลการวิจัยที่ได้ พร้อมทั้งให้กำลังใจและข้อเสนอแนะต่าง ๆ และ ขอขอบคุณท่านอื่น ๆ ที่มีส่วนช่วยในการทำวิทยานิพนธ์ที่ไม่ได้กล่าวนามมา ณ โอกาสนี้ด้วย

สุดท้ายนี้ ผู้วิจัยใคร่ขอกราบขอบพระคุณ บิดา มารดาที่สนับสนุนในด้านต่าง ๆ และให้กำลังใจแก่ผู้วิจัยเสมอมา

วชิราวุธ ธรรมวิเศษ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญภาพ

รูปที่ **หน้า**

สารบัญภาพ (ต่อ)

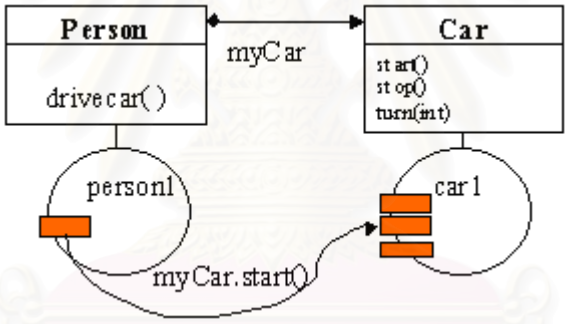
รูปที่ **หน้า**

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

แนวคิดของการเขียนโปรแกรมเชิงวัตถุจำลองมาจากคุณลักษณะของวัตถุในโลกความเป็นจริง ออบเจคต์(Object) จะประกอบด้วย ข้อมูล (Data) และ วิธีการทำงานเรียกว่า “เมทอด” (Method) ภายในระบบเชิงวัตถุจะประกอบด้วยออบเจคต์ต่างๆที่มีความสัมพันธ์กัน และระบบจะทำงานจากการสื่อสารระหว่างออบเจคต์ภายในระบบ โดยการเรียกใช้เมทอดของกันและกัน ซึ่งออบเจคต์จะสามารถสื่อสารกันได้จะต้องมีความสัมพันธ์กันในระดับคลาส ดังรูปที่ 1.1 จะเห็นว่าคลาส Person และ คลาส Car มีความสัมพันธ์กันทำให้ออบเจคต์ person1 สามารถเรียกใช้เมทอดของออบเจคต์ car1 ได้ โดย person1 รู้จัก car1 ในชื่อของ myCar

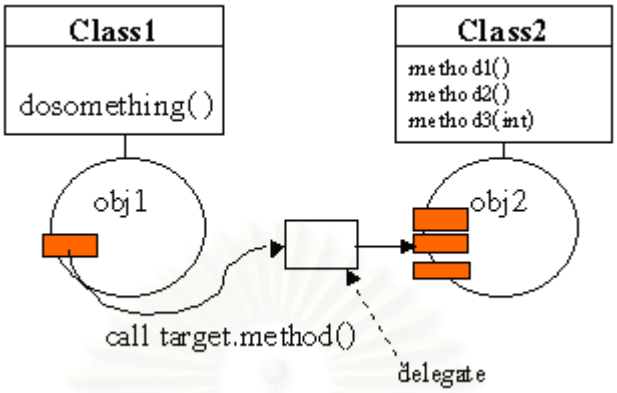


รูปที่ 1.1 การสื่อสารระหว่างออบเจคต์

แต่ในกรณีที่ออบเจคต์ไม่มีความสัมพันธ์กันในระดับคลาส จะทำอย่างไรจึงจะสามารถทำให้ออบเจคต์สามารถสื่อสารกันได้ ซึ่งวิธีการแก้ปัญหาคือ การสร้างตัวเชื่อมต่อ หรือ การสร้างตัวแทน ซึ่งสามารถให้ออบเจคต์ต้นทางสามารถเรียกเมทอดของออบเจคต์ปลายทางผ่านทางตัวแทนที่สร้างขึ้นดังรูปที่ 1.2 ซึ่งเราจะเรียกเทคนิคนี้ว่า “ดีเลเกชัน” (Delegation)

เทคนิคดีลีเกชันช่วยให้สร้างออบเจคต์ที่สามารถเปลี่ยนพฤติกรรมการทำงานได้ โดยไม่ต้องสร้างคลาสใหม่ โดยการจัดเตรียมตัวแทนการทำงานที่ยังไม่เกิดขึ้นไว้ก่อนเพื่อให้ส่งต่อการเรียกไปยังเมทอดปลายทางที่จะถูกกำหนดภายหลัง ซึ่งเทคนิคนี้จะถูกใช้ในการสร้างตัวจัดการเหตุการณ์ โดยการออกแบบคลาสจะมีการสร้างตัวแทน(Delegate)สำหรับแต่ละเหตุการณ์ไว้ และเขียน

โปรแกรมเรียกใช้ตัวแทนเหล่านั้นตามเหตุการณ์ เมื่อสร้างออบเจคต์จากคลาสและต้องการควบคุมเหตุการณ์ใด ก็สามารถกำหนดให้ตัวแทนเหตุการณ์นั้นให้เรียกเมทอดที่เขียนขึ้น



รูปที่ 1.2 การเรียกใช้เมทอดผ่านตัวแทน

ซึ่งวิธีการที่ใช้ในการสร้างตัวแทนมีได้หลายวิธีแต่ที่นิยมใช้กันคือการใช้ “ตัวชี้เมทอด” (Method Pointer) ซึ่งเป็นข้อมูลแบบพอยน์เตอร์ประเภทหนึ่งที่สามารถอ้างถึงเมทอดของออบเจคต์ และสามารถเรียกให้เมทอดที่ถูกอ้างถึงทำงานได้ ตัวชี้เมทอดจะพบในภาษาโปรแกรมบางภาษา เช่น ซีพลัสพลัส (C++) และ ออบเจคต์ปาสคาล(Object Pascal) ซึ่งจะมีคำสั่งสำหรับตัวชี้เมทอดโดยเฉพาะเช่น การสร้าง การกำหนดค่า การเรียกใช้ เป็นต้น แต่ในภาษาจาวาจะไม่มีความสามารถนี้ แต่เป็นที่ยอมรับว่าเทคนิคตัวชี้เมทอดมีความจำเป็นต่อการเขียนโปรแกรม[4][7] เพราะการที่ ออบเจคต์สามารถเรียกใช้เมทอดของออบเจคต์อื่นผ่านตัวแปรได้ ทำให้สามารถติดต่อไปยังออบเจคต์อื่นๆ ที่ไม่รู้จักมาก่อนได้ เช่น ในการจัดการเหตุการณ์สำหรับเขียนโปรแกรมติดต่อกับผู้ใช้แบบกราฟฟิก (GUI) ตัวอย่างเช่น ในโปรแกรมเมื่อปุ่มคำสั่งถูกกด ปุ่มคำสั่งนั้นจะทำการส่งสัญญาณไปเรียกให้เมทอดที่ทำหน้าที่ตอบสนองต่อเหตุการณ์นี้ทำงาน ในโปรแกรมสามารถมีปุ่มได้จำนวนมาก แต่ละปุ่มสามารถมาจากคลาสเดียวกัน ดังนั้นถ้าเมทอดเป้าหมายอยู่ในรูปของตัวแปร แต่ละปุ่มก็สามารถส่งสัญญาณเรียกใช้เมทอดสำหรับตอบสนองการทำงานที่ต่างกันได้

การที่ภาษาจาวานั้นไม่มีตัวชี้เมทอดนั้นบริษัทซันไมรโครซิสเต็ม[1] ได้กล่าวใน [1][4]ว่าไม่ควรจะมีคำสั่งพิเศษเพื่อจัดการกับตัวชี้เมทอดในภาษาจาวาเนื่องจากจะทำให้เกิดผลกระทบต่อภาษาและ แบบจำลองเชิงวัตถุของจาวา แต่ในการประชุมจาวาครั้งที่ 1 [4] มีการบันทึกถึงความจำเป็นที่ต้องมีวิธีที่สามารถใช้แทนตัวชี้เมทอดในจาวา ดังนั้นจึงได้มีการพัฒนาวิธีการสำหรับใช้แทนตัวชี้เมทอดในจาวาขึ้นซึ่งมีหลายวิธี แต่วิธีที่มีใช้ในปัจจุบันคือ วิธี “Adapter Class” [7][8] พัฒนาโดย

[1] Sun Microsystem, Inc เป็นบริษัทผู้สร้างภาษาจาวา

บริษัทซันไมรโครซิสเต็มเป็นวิธีที่ใช้ในจาวามาตราฐาน และวิธี “Delegate” ในวิชวลเจเวอร์ชัน 6.0 (Visual J++6.0) [2][3] พัฒนาโดยบริษัทไมโครซอพท์ วิธีนี้ใช้ได้กับ วิชวลเจเวอร์ชัน 6.0 เท่านั้น

สำหรับในวิทยานิพนธ์นี้จะเป็นการสร้างเทคนิคเพื่อใช้แทนตัวชี้เมทอดบนจาวาเช่นเดียวกัน โดยเลือกใช้วิธีการที่คล้ายกับ วิธี “Delegate” โดยใช้แนวคิดการเก็บข้อมูลอ้างถึงเมทอดไว้ในตัวแปรแบบออบเจคต์และเรียกใช้เมทอดผ่านทางตัวแปรนั้นได้ แต่วิธี “Delegate” มีการนิยามคำสั่งใหม่ในภาษาจาวา จึงมีการถกเถียงกันถึงแนวคิดของ “Delegate” ว่าขัดต่อแบบจำลองเชิงวัตถุของจาวา (Java Object Oriented Model) หรือไม่ ซึ่งบริษัทซันไมโครซิสเต็ม และ บริษัทไมโครซอพท์มีความเห็นที่ขัดแย้งกันและยังไม่มีข้อสรุป[3][4] ดังนั้นในงานวิจัยนี้จะไม่ขอกล่าวว่า เทคนิคที่ใช้ในงานวิจัยนี้ขัดต่อแบบจำลองเชิงวัตถุของจาวาหรือไม่ ผู้วิจัยเพียงหวังว่าเทคนิคนี้จะเป็นอีกทางเลือกหนึ่งในการเขียนโปรแกรมบนจาวา

ในงานวิจัยนี้เรียกเทคนิคนี้ว่า “Method Variable” หรือ ตัวแปรเมทอด ซึ่งหมายถึง ตัวแปรที่เก็บเมทอด เหตุที่ไม่ใช่คำว่า “Method Pointer” เนื่องจากตัวแปรเมทอดเป็นออบเจคต์ และที่ไม่ใช้คำว่า “delegate” เพราะซ้ำกับ “Delegation Model” อาจจะทำให้สับสน ถึงแม้ว่าแนวคิดที่ใช้ในงานวิจัยนี้จะเป็นแนวทางเดียวกับ วิธี “Delegate” ในวิชวลเจเวอร์ชัน 6.0 ก็ตาม แต่การพัฒนาตัวแปรเมทอดจะไม่มีการเพิ่มเติมและแก้ไขคำสั่งในภาษาจาวา เพื่อให้โปรแกรมสามารถคอมไพล์ด้วยตัวแปลภาษาจาวามาตรฐานได้

การออกแบบและสร้างตัวแปรเมทอดจะมุ่งเน้นให้ใช้งานง่าย รูปแบบการใช้มีความยืดหยุ่นเหมาะสำหรับใช้สร้างเหตุการณ์สำหรับคอมโพเนนท์ โดยจะมีการสร้างตัวแปรเมทอดแบบต่างๆ เช่น ตัวแปรเมทอดแบบเดี่ยว ตัวแปรเมทอดแบบหลายจำนวน เพื่อให้ผู้ใช้สามารถเลือกใช้งานตามความเหมาะสม นอกจากนี้ยังมีการวิจัยในส่วนของการจัดเก็บการเชื่อมต่อเหตุการณ์ระหว่างคอมโพเนนท์ซึ่งสร้างด้วยตัวแปรเมทอดไว้ในไฟล์ และการใช้ตัวแปรเมทอดในฐานข้อมูลเชิงวัตถุ โดยใช้ฐานข้อมูล Object Store[2] สำหรับจาวา

[2] Object Store เป็นระบบจัดการฐานข้อมูลเชิงวัตถุของบริษัท eXcelon Corporation

1.2 วัตถุประสงค์

เพื่อออกแบบและพัฒนาวิธีการสร้างตัวแปรเมทอดให้สามารถจัดเก็บเมทอดของออบเจคต์ในหน่วยความจำและเมทอดของออบเจคต์ในฐานข้อมูลเชิงวัตถุ และเรียกใช้เมทอดที่ถูกเก็บให้ทำงานผ่านทางตัวแปรเมทอด สำหรับใช้ในการพัฒนาโปรแกรมในจาวา

1.3 ขอบเขตการวิจัย

1. สามารถสร้างกลไกที่สามารถเก็บเมทอดของออบเจคต์และเรียกใช้เมทอดผ่านตัวแปรในภาษาจาวาได้ โดยสนับสนุนทั้งตัวแปรเมทอดแบบเดี่ยวและตัวแปรเมทอดแบบหลายจำนวน
2. สามารถสร้างโครงร่างพัฒนาโปรแกรมเบื้องต้น สำหรับการเขียนโปรแกรมติดต่อผู้ใช้แบบกราฟฟิกสำหรับจาวา โดยพัฒนาจากต้นแบบคอมโพเนนท์ในไลบรารี่ java.awt ของชุดพัฒนาโปรแกรมจาวา(Java Development Kit) และใช้ตัวแปรเมทอดในการจัดการเหตุการณ์แทนกลไกที่มีอยู่เดิม
3. สามารถจัดเก็บตัวแปรเมทอดในไฟล์ และ เรียกกลับคืนมาใช้ได้
4. สามารถเก็บและเรียกใช้เมทอดของออบเจคต์ถาวรในฐานข้อมูลเชิงวัตถุได้ โดยจะทำการทดลองกับระบบจัดการฐานข้อมูลเชิงวัตถุสำเร็จรูป ที่ใช้ได้กับภาษาจาวาในระบบงานตัวอย่างหนึ่งประเภท

1.4 ขั้นตอนการวิจัย

1. ศึกษาวิธีการจัดการเหตุการณ์ในเครื่องมือพัฒนาโปรแกรมต่างๆ ได้แก่ Delphi Visual Component Libary, Java Beans และ COM (Component Object Model)
2. ศึกษาโครงสร้างและ คลาส ต่างๆ ในชุดพัฒนาโปรแกรมจาวา
3 สร้างตัวแปรเมทอดในภาษา จาวา
4. ออกแบบและสร้างคลาสของตัวแปรเมทอดแบบหลายจำนวน
5. ทดลองใช้งานตัวแปรเมทอดในจาวา
6. สร้างโครงร่างพัฒนาโปรแกรมโดยใช้ตัวแปรเมทอดในการจัดการเหตุการณ์และสร้างโปรแกรมทดสอบการใช้งาน
7. ใช้ตัวแปรเมทอดในฐานข้อมูลเชิงวัตถุ
8. ศึกษาวิธีการจัดเก็บและเรียกคืนตัวแปรเมทอดในออบเจคต์ถาวรในระบบที่ไม่ซับซ้อน และ สร้างโปรแกรมทดสอบการทำงาน

9. ศึกษาวิธีการจัดเก็บและเรียกคืนตัวแปรเมทอดในออบเจคต์ถาวรในฐานข้อมูลเชิงวัตถุ และ สร้างโปรแกรมทดสอบการทำงาน
10. สรุปผลงานวิจัย
11. จัดทำรายงานวิทยานิพนธ์

1.5 ประโยชน์ที่จะได้รับ

กลไกการจัดการตัวแปรเมทอดสำหรับภาษาจาวาทำให้สามารถเก็บและทำการเรียกเมทอดผ่านทางตัวแปรได้ จะมีประโยชน์ต่อการพัฒนาโปรแกรมด้วยจาวาดังต่อไปนี้

1 สามารถใช้ตัวแปรเมทอดเป็นกลไกจัดการเหตุการณ์ ในแบบจำลองคอมโพเนนท์บนจาวา
2. สามารถนำตัวแปรเมทอดไปใช้ในการออกแบบคลาสในภาษาจาวาที่มีความยืดหยุ่นและช่วยลดการผูกติดระหว่างคลาสเมื่อใช้ตัวแปรเมทอดสำหรับเชื่อมต่อระหว่างออบเจคต์ ทำให้สามารถนำคลาสกลับมาใช้ใหม่ได้ง่าย
3 สามารถใช้ในเครื่องมือพัฒนาโปรแกรมแบบวิชวล (Visual Programming Tool) สำหรับภาษาจาวาเพราะไม่ต้องสร้างคลาสใหม่ในขณะออกแบบโปรแกรมเพื่อเชื่อมต่อเหตุการณ์ของคอมโพเนนท์ นอกจากนี้ยังสามารถเก็บการเชื่อมต่อที่อยู่ในตัวแปรเมทอดในไฟล์ และ อ่านขึ้นมาเพื่อเรียกใช้ได้ ทำให้ลดการสร้างโค้ดโปรแกรมสำหรับการเชื่อมต่อคอมโพเนนท์
4. การจัดเก็บเมทอดของออบเจคต์ในฐานข้อมูลเชิงวัตถุ สามารถใช้ในการสร้างกลไกของการจัดการเหตุการณ์ ซึ่งสามารถใช้ในการสร้างระบบฐานข้อมูลพร้อมทำงาน (Active Database System)

บทที่ 2

แนวคิดทฤษฎีและงานที่เกี่ยวข้อง

แนวคิดของตัวแปรเมทอดในงานวิจัยนี้ พัฒนามาจากความสามารถของตัวชี้ฟังก์ชัน และตัวชี้เมทอด ที่พบในภาษาโปรแกรมเช่น ซีพลัสพลัส และ ปาสคาล แต่ไม่มีในภาษาจาวา จึงได้มีการพัฒนาเทคนิคเพื่อใช้แทนความสามารถของตัวชี้เมทอดในจาวา โดยในบทนี้จะกล่าวถึงรายละเอียดของตัวชี้ฟังก์ชันและตัวชี้เมทอด และทำไมตัวชี้เมทอดจึงมีความสำคัญในการเขียนโปรแกรม และ มีเทคนิคใดบ้างที่ถูกพัฒนาเพื่อใช้แทนตัวชี้เมทอดในภาษาจาวา

2.1 ตัวชี้ฟังก์ชัน และ ตัวชี้เมทอด

2.1.1 "ตัวชี้ฟังก์ชัน" (Function Pointer)

คือตัวแปรแบบพอยน์เตอร์ที่สามารถใช้อ้างถึงที่อยู่ของฟังก์ชันและยอมให้สามารถเรียกใช้ฟังก์ชันผ่านตัวแปรได้ จะพบการใช้งานตัวชี้ฟังก์ชันในภาษาแบบโครงสร้าง เช่น ซี,ปาสคาล ตัวชี้ฟังก์ชันมีประโยชน์ในการเขียนโปรแกรมมาก การที่ยอมให้มีการเรียกใช้ฟังก์ชันผ่านตัวแปร หรือส่งที่อยู่ของฟังก์ชันเป็นพารามิเตอร์ในการเรียกใช้งานฟังก์ชันอื่น ในลักษณะของฟังก์ชันเรียกกลับ (Call-Back Function) ทำให้เราสามารถเขียนโปรแกรมที่ยืดหยุ่นสูงได้

ตัวอย่าง:การใช้ตัวชี้ฟังก์ชัน ในภาษาปาสคาลจะยอมให้เราสามารถสร้างชนิดข้อมูลที่เรียกว่า Procedure Type หรือ Fuction Type เป็นชนิดข้อมูลแบบพอยน์เตอร์ประเภทหนึ่งที่สามารถชี้ไปที่ฟังก์ชันได้ การสร้างชนิดของตัวชี้ฟังก์ชันกำหนดด้วยชนิดของพารามิเตอร์ที่ใช้เรียกฟังก์ชัน ซึ่งเรียกว่าเป็นลายเซ็นของฟังก์ชัน

```
Type   CompareFunction = function (p1,p2 :Pointer) :Integer;
Var  func1 :  CompareFunction;
```

จากตัวอย่างเป็นการสร้างชนิดฟังก์ชันสำหรับการเปรียบเทียบข้อมูล โดยจะรับพารามิเตอร์ 2 ตัว และ คืนค่าเป็นตัวเลข การใช้งานตัวชี้ฟังก์ชันจะคล้ายกับตัวแปรชนิดอื่นๆ แต่ค่าที่เก็บในตัวแปรคือที่อยู่ในหน่วยความจำของฟังก์ชัน การกำหนดฟังก์ชันให้ตัวแปรจะถูกตรวจสอบความถูกต้องในระหว่างการคอมไพล์ ซึ่งเหมือนกับการตรวจสอบชนิดของตัวแปรประเภทอื่น โดย

ฟังก์ชันที่จะกำหนดให้ตัวแปรได้จะต้องมีลายเซ็นตรงกัน และตัวชี้ฟังก์ชันยังสามารถสร้างเป็นตัวแปรหรือส่งเป็นพารามิเตอร์ของฟังก์ชันปกติได้

```
procedure  Sort (var items: Array of Pointer;  c: CompareFunction);
var  i, n1,n2 :integer;
Begin
....
   If  c( items[n1], items[n2] ) < 0 Then
      // n1 < n2
   else
      //  n2 >= n1
      .....
End;
```

ตัวอย่าง เป็นการสร้างฟังก์ชัน Sort สำหรับเรียงข้อมูล สิ่งสำคัญของการเรียงข้อมูลคือการเปรียบเทียบข้อมูลแต่ละตัวที่จะเอามาเรียง เราสามารถกำหนดให้วิธีการเปรียบเทียบเป็นตัวชี้ฟังก์ชันได้ ดังนั้นฟังก์ชัน Sort นี้จึงสามารถใช้กับข้อมูลหลากหลาย การใช้งานผู้ใช้ต้องกำหนดข้อมูลที่ต้องการเรียงด้วย อะเรย์ของพอยน์เตอร์และฟังก์ชันที่ใช้สำหรับเปรียบเทียบข้อมูลแต่ละตัว โดยการเรียงจะดูจากผลลัพธ์ของการเปรียบเทียบ

คืนค่า น้อยกว่า 0 ถ้า p1 น้อยกว่า p2

เท่ากับ 0 ถ้า p1 เท่ากับ p2

มากกว่า 0 ถ้า p1 มากกว่า p2

ตัวอย่างการใช้งาน : ให้ข้อมูลที่ต้องการเรียงเป็นข้อมูลนักเรียน

```
Type  Student = Record
                   ID :String[10];
                   Name :String[40];
                   Majar :String[20];
                End;
PStudent = ^Student;   // Student Pointer
Var  StudentData  :Array[0..100] of PStudent;
```

สร้างฟังก์ชันการเปรียบเทียบชื่อนักเรียน

```
Function  StdNameCompare(S1,S2 :Pointer) :Integer;
   Begin
      If  Student(S1).Name < Student(S2).Name then  Result := -1
      else If Student(S1).Name > Student(S2).Name then  Result := 1
      else  Result := 0;
   End;
```

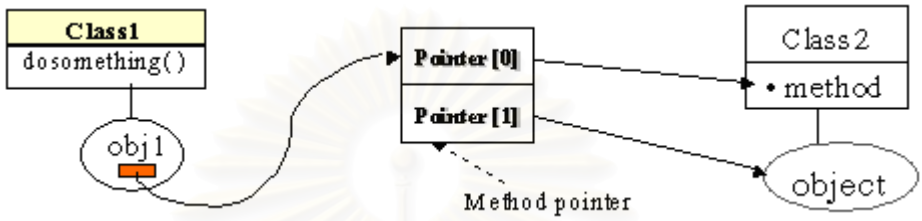
การเรียกใช้ฟังก์ชัน Sort

```
Sort (StudentData , StdNameCompare);
```

ถ้าต้องการเรียงข้อมูลวิธีใหม่ก็สามารถสร้างฟังก์ชัน เปรียบเทียบใหม่ โดยไม่ต้องแก้ไขโปรแกรมใน ฟังก์ชัน Sort จะพบว่าการใช้ตัวชี้ฟังก์ชันช่วยในการเขียนโปรแกรมมาก

*2.1.2 "**ตัวชี้เมทอด**" (Method Pointer)*

คือ ตัวแปรแบบพอยน์เตอร์ เช่นเดียวกับตัวชี้ฟังก์ชันแต่ต่างที่ตัวชี้เมทอดหนึ่งตัวจะประกอบด้วย พอยน์เตอร์ 1 คู่ เพื่อใช้อ้างถึงตำแหน่งในหน่วยความจำของเมทอดและของ ออบเจคต์ เพื่อทำให้เรียกเมทอดของออบเจคต์ผ่านตัวแปรได้ ดังรูปที่ 2.1 ซึ่งมีภาษาที่สนับสนุนตัวชี้เมทอด เช่น ซีพลัสพลัส และ ออบเจคต์ปาสคาล เป็นต้น



รูปที่ 2.1 ตัวชี้เมทอด

รูปแบบการใช้งานจะเหมือนกับการใช้งานตัวชี้ฟังก์ชัน การจัดการตัวชี้เมทอดจะเป็นหน้าที่ของคอมไพเลอร์ ตัวอย่างการใช้ตัวชี้เมทอดในภาษาออบเจคต์ปาสคาล

```
Type    TNotifyEvent = Procedure (sender :TObject) of Object;
```

การประกาศชนิดของตัวชี้เมทอดจะมีประโยค of Object เพื่อบอกว่าเป็นตัวชี้ฟังก์ชันของออบเจคต์ และฟังก์ชันที่จะกำหนดค่าให้ตัวแปรชนิดนี้ได้ ต้องเป็นเมทอดของออบเจคต์ที่มีพารามิเตอร์เข้ากันได้เท่านั้น

```
var  onClick : TNotifyEvent;
Begin
    onClick :=  form1.OkButtonClick;
End;
```

การเรียกใช้ คำสั่ง onClick(OkButton); จะเท่ากับ form1.OkButtonClick(OkButton) เพื่อให้สามารถเรียกใช้ตัวชี้เมทอดแบบนี้ได้ ในตัวชี้เมทอดจึงต้องประกอบด้วย พอยน์เตอร์จำนวน 2 ตัว โดยตัวแรกชี้ไปที่ตำแหน่งของฟังก์ชัน(ซึ่งอยู่ในคลาส) ตัวที่สองชี้ไปที่ตำแหน่งของออบเจคต์ เมื่อคอมไพล์คำสั่งที่มีการเรียกใช้ตัวชี้เมทอด คอมไพเลอร์จะทำการแปลงคำสั่งดังกล่าวให้

ในเครื่องมือพัฒนาโปรแกรมเดลไฟ(Delphi) จะใช้ตัวชี้เมทอดในการสร้างเหตุการณ์ของคอมโพเนนท์ โดยใช้เทคนิคดีลีเกชัน (Delegation) ซึ่งจะใช้ตัวชี้เมทอดเก็บเมทอดที่ต้องการใช้คอมโพเนนท์เรียกใช้ เมื่อมีเหตุการณ์ตามที่กำหนดเกิดกับคอมโพเนนท์

2.2 **แบบจำลองดีลีเกชัน** (Delegation Model)

ดีลีเกชันเป็นแนวคิดของการเขียนโปรแกรมเชิงวัตถุที่ทำให้สามารถเปลี่ยนแปลงการทำงานหรือพฤติกรรมของวัตถุได้ โดยไม่ต้องแก้ไขคลาสหรือสร้างคลาสใหม่ด้วยการสืบทอดคลาส (Inheritance) โดยปกติเมื่อต้องการเปลี่ยนแปลงพฤติกรรมของวัตถุให้แตกต่างจากเดิม ผู้เขียนจะต้องสร้างคลาสใหม่โดยสืบทอดคลาสเดิมแล้วทำการ เขียนทับ(Override) เมทอดของคลาสเดิม

แนวคิดของดีลีเกชันจะทำการออกแบบโครงสร้างของคลาส เพื่อทำให้ออบเจคต์ของคลาสนั้นสามารถไปเรียกใช้ส่วนของโปรแกรมในออบเจคต์หรือในคลาสอื่นที่ยังไม่มีให้ขณะที่สร้างคลาสนั้น เพื่อให้สามารถทำเช่นนั้นได้ ออบเจคต์จะทำการสร้างตัวแทน(Delegate) ของส่วนโปรแกรมหรือ เมทอดของออบเจคต์ที่ต้องการเรียกใช้ ตัวแทนนี้สามารถเป็นอะไรก็ได้ขึ้นอยู่กับเทคนิคที่ใช้ด้วยวิธีการสร้างตัวแทนเมทอด เป้าหมายที่ออบเจคต์ต้องการเรียกจึงยังไม่จำเป็นต้องมีตัวตนจริงในขณะออกแบบคลาส แต่เป้าหมายจะถูกกำหนดเมื่อใช้งานออบเจคต์ทำให้เราสามารถเปลี่ยนแปลงพฤติกรรมของออบเจคต์ได้

วิธีดีลีเกชันนำไปใช้ในการทำส่วนจัดการเหตุการณ์(Event) ในแบบจำลองคอมโพเนนท์ เพราะการสร้างเหตุการณ์คือการเรียกใช้เมทอดของคอมโพเนนท์เมื่อมีเหตุการณ์เกิดขึ้น ซึ่งเป้าหมายของเหตุการณ์จะยังไม่ถูกกำหนดในขั้นตอนการสร้างคอมโพเนนท์ แต่จะถูกกำหนดการทำงานเมื่อนำเอาคอมโพเนนท์นั้นไปใช้ เทคนิคการทำดีลีเกชันในเครื่องมือพัฒนาโปรแกรมแต่ละภาษาจะแตกต่างกัน ขึ้นอยู่กับแนวคิดและข้อจำกัดของแต่ละภาษาด้วย สำหรับจาวาเองในปัจจุบันจะมีวิธีการทำดีลีเกชัน 2 วิธี สำหรับแนวคิดตัวแปรเมทอดในงานวิจัยนี้ก็เป็นเทคนิคหนึ่งที่สามารถใช้ในการทำดีลีเกชันในภาษาจาวา

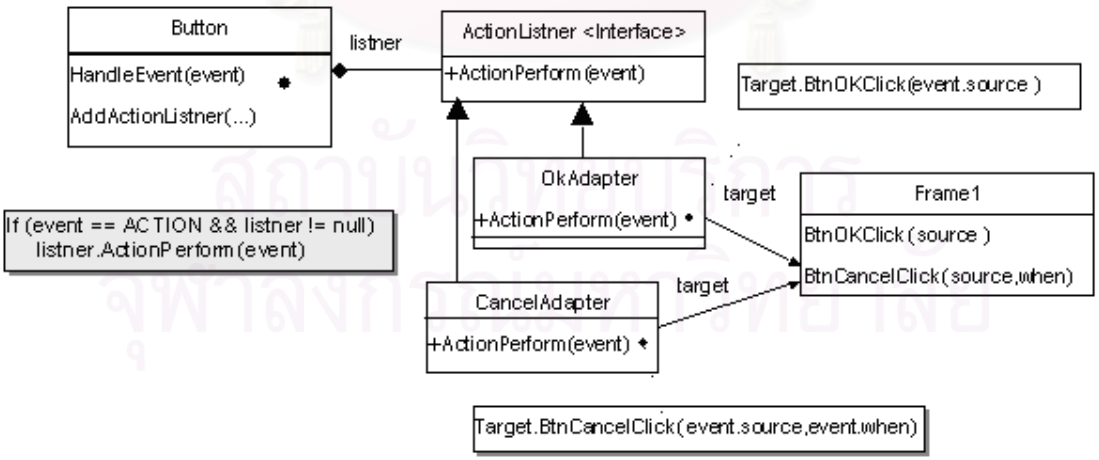
2.3 **เทคนิคที่ใช้แทนตัวชี้เมทอดบนภาษาจาวา**

จากที่กล่าวมาแล้วข้างต้น การพัฒนาโปรแกรมในปัจจุบันจะใช้แนวคิดของแบบจำลองคอมโพเนนท์ และ ใช้เทคนิค “ดีเลเกชัน (Delegation)” เพื่อช่วยในการเชื่อมต่อคอมโพเนนท์และเขียนโปรแกรมควบคุมเหตุการณ์ เครื่องมือพัฒนาโปรแกรมแต่ละภาษาจะมีการใช้เทคนิคที่แตกต่างกัน ขึ้นอยู่รูปแบบภาษาที่ใช้ด้วย เช่น ภาษาซีพลัสพลัส และ ออบเจคต์ปาสคาล จะใช้ “ตัวชี้เมทอด” (Method Pointer) ในการทำดีเลเกชัน แต่ภาษาจาวานั้นไม่มีตัวชี้เมทอดจึงต้องสร้างเทคนิคที่ใช้แทนตัวชี้เมทอด ซึ่งปัจจุบันมี 2 แบบได้แก่ “Adapter Class” ในจาวามาตรฐาน 1.1 และ “Delegate” ในไมโครซอฟท์วิชวลเจเวอร์ชัน 6.0

2.3.1 "Adapter Class" ในจาวามาตรฐาน 1.1

วิธี คลาสอะเดปเตอร์ จะสร้างตัวแทนด้วย "คลาสอินเตอร์เฟซ" (Interface Class) ดังนั้นออบเจคต์สามารถเรียกเมทอดของออบเจคต์จากคลาสที่ต่างกันได้ ถ้าคลาสเหล่านั้นมีอินเตอร์เฟซตรงกับที่ออบเจคต์รู้จัก ดังตัวอย่างในรูปที่ 2.2 มีการสร้างปุ่ม 2 ปุ่มจากคลาส Button คือปุ่ม OK และ ปุ่ม Cancel การเขียนคำสั่งเพื่อควบคุมการทำงานของแต่ละปุ่ม จะต้องสร้างคลาสอะเดปเตอร์ ที่สืบทอดจากคลาสอินเตอร์เฟซที่กำหนด คือคลาส ActionListner ซึ่งจะกำหนดให้เขียนคำสั่งควบคุมเหตุการณ์ ไว้ในเมทอด 'ActionPerform' เช่น คลาส OkAdapter จะเขียนคำสั่งเพื่อส่งต่อการทำงานไปยังเมทอดเป้าหมายคือ target.btnOKClick(…) จากนั้นจึงสร้างออบเจคต์จากคลาส OkAdapter นี้เพื่อส่งไปเป็นผู้รับฟังเหตุการณ์ (Listener) ดังนั้นเมื่อเกิดเหตุกาณ์ เช่นปุ่มถูกกด จะทำให้เมทอด 'ActionPerform' ของผู้รับฟังจะถูกเรียกทำให้การเรียกถูกส่งต่อไปยังเมทอดปลายทางได้

ด้วยวิธีนี้การเขียนโปรแกรมเพื่อควบคุมเหตุการณ์ ผู้เขียนจะต้องสร้างคลาสใหม่ที่สืบทอดจากคลาสอินเตอร์เฟซที่กำหนดไว้สำหรับสร้างออบเจคต์ตัวแทน และเขียนการทำงานเพื่อจัดการเหตุการณ์ในเมทอดตามที่กำหนดไว้ ดังนั้นการจะเชื่อมเหตุการณ์กับเมทอดเป้าหมายที่มีอยู่แล้วจะต้องมีการสร้างคลาสอะเดปเตอร์ใหม่ และ สร้างออบเจคต์จากคลาสนั้นเพื่อเป็นตัวเชื่อมทำให้วิธีนี้ไม่เหมาะกับการเชื่อมเหตุการณ์ในขณะรันโปรแกรม เพราะต้องมีการคอมไพล์คลาสใหม่ทุกครั้งที่สร้างการเชื่อมต่อ และนอกจากถ้ามีการเชื่อมต่อจำนวนมากก็จะเกิดคลาสอะเดปเตอร์จำนวนมากตามไปด้วย



รูปที่ 2.2 ตัวอย่าง แบบจำลองเหตุการณ์ ในจาวามาตรฐาน

2.3.2 "Delegate" ในไมโครซอฟท์วิชวลเจเวอร์ชัน 6.0

วิธี ดีลีเกท(Delegate) หรือ "bound method references" จะสร้างตัวแทนจากออบเจคต์ที่สามารถเก็บเมทอดของออบเจคต์อื่นได้ ซึ่งเป็นแนวคิดตรงมาจากตัวชี้เมทอด ตัวแทนจะต้องสร้างจากคลาสที่สืบทอดมาจากคลาส "Delegate" หรือ "MulticastDelegate" การสร้างเหตุการณ์จะเป็นการนิยามรูปแบบส่งพารามิเตอร์ โดยต้องสร้างคลาสสำหรับรูปแบบการส่งพารามิเตอร์แต่ละแบบ เหตุการณ์ที่มีการส่งพารามิเตอร์เหมือนกันสามารถใช้คลาสเดียวกันได้ ด้วยวิธีนี้การเขียนโปรแกรมเพื่อควบคุมเหตุการณ์จะเป็นการสร้างเมทอดในคลาสที่นำคอมโพเนนท์ไปใช้ เมทอดต้องมีรูปแบบพารามิเตอร์ตรงกับที่กำหนด แล้วสร้างออบเจคต์ตัวแทนจากคลาส "Delegate" ที่ใช้สำหรับรูปแบบพารามิเตอร์นั้น และให้ตัวแทนนั้นเก็บข้อมูลที่อ้างถึงเมทอดที่เราสร้างขึ้น ในวิธีนี้เพื่อให้สามารถสร้าง คลาส "Delegate" ได้ง่าย และ ให้สามารถใช้ลักษณะการเขียนอ้างถึงเมทอดของออบเจคต์คล้ายกับการใช้ตัวชี้เมทอด จึงมีการกำหนดรูปแบบคำสั่งใหม่สำหรับภาษาจาวาขึ้น ซึ่งคำสั่งดังกล่าวใช้ได้กับ ไมโครซอฟท์วิชวลเจเวอร์ชัน 6.0 เท่านั้น ดังตัวอย่าง

```
Delegate long IntOp(int a, int b);
```
:- คำสั่งนิยาม Delegate จะได้ IntOp เป็นคลาสแบบ Delegate ที่สามารถอ้างถึงเมทอดที่มีรูปแบบพารามิเตอร์เป็น (int a, int b)

```
long add(int a,int b){return a+b; }
void test() {
    IntOp op = new IntOp(this.add);
}
```
:- คำสั่งการสร้างออบเจคต์ จาก IntOp สามารถอ้างถึงเมทอดที่ต้องการได้เลย เช่น **this.add** ซึ่งจะถูกแปลงเป็น this , "add" ในขั้นการคอมไพล์

```
int  x = op.invoke(10,20);
```
:- เป็นคำสั่งการเรียกใช้เมทอดที่ถูกอ้างถึงในตัวแปร **op** จากตัวอย่าง this.add(10,20) จะถูกเรียกให้ทำงานต่อไป และตัวแปร **x** จะมีค่าเป็น 30

จากการศึกษาพบว่าเทคนิค "Delegate" ของไมโครซอฟท์วิชวลเจเวอร์ชัน 6.0 สำหรับการเชื่อมต่อคอมโพเนนท์มีความยืดหยุ่นสูงกว่า "Adapter class" ในจาวามาตรฐาน เนื่องจากไม่ต้อง

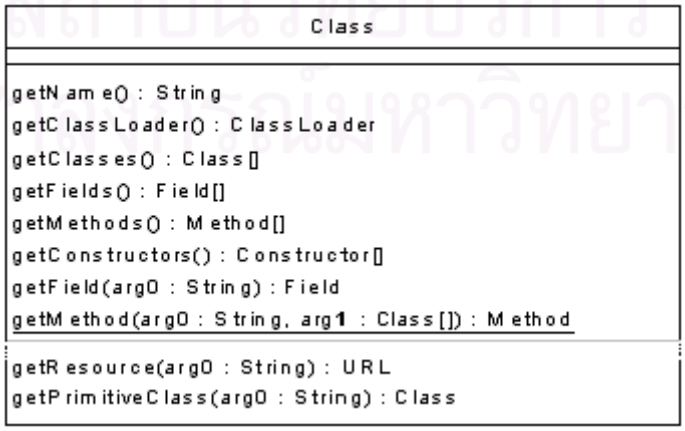
สร้างคลาสใหม่ การเชื่อมต่อจะคล้ายกับเทคนิคของตัวชี้เมทอดในเดลไฟซึ่งไม่สนใจอินเตอร์เฟสหรือคลาสเพราะเป็นการเชื่อมต่อที่เมทอดของออบเจคต์ ถ้ามีพารามิเตอร์ที่ตรงกันก็สามารถเชื่อมต่อกันได้

แนวคิดของตัวแปรเมทอดในงานวิจัยนี้จะใช้เทคนิคในแนวทางเดียวกับ “Delegate” แต่จะต่างที่การออกแบบและลักษณะการทำงาน ในไมโครซอฟท์วิชวลเจเวอร์ชัน 6.0 มีการนิยามคำสั่ง *delegate* และการใช้รูปแบบคำสั่ง *object.method* เพื่อใช้แทนชื่อเมทอดของออบเจคต์ ซึ่งไม่ใช่คำสั่งในภาษาจาวามาตรฐานจึงทำให้ไม่สามารถคอมไพล์ด้วยตัวแปรภาษาจาวาอื่นได้ ดังนั้นงานวิจัยนี้จะใช้คำสั่งของจาวามาตรฐานเพื่อให้สามารถใช้คอมไพล์ด้วยตัวแปรภาษาจาวามาตรฐานได้

2.4. การเข้าถึงข้อมูลเมทอดในจาวา

วิธีการของตัวแปรเมทอดคือการเก็บข้อมูลของเมทอดไว้ทำให้เรียกใช้เมทอดที่เก็บในตัวแปรได้ การจะเข้าถึงข้อมูลเมทอดในจาวา จะใช้ความสามารถของ “Reflection” ซึ่งจะอยู่ในกลุ่มคลาส “java.lang.reflect” สามารถศึกษาเพิ่มใน [1] และคู่มือภาษาจาวา อื่นๆ

ในภาษาจาวาข้อมูลของโปรแกรมทั้งหมดจะอยู่ในรูปของออบเจคต์ ซึ่งรวมถึงข้อมูลของคลาส และ เมทอด จะอยู่ในรูปของออบเจคต์ที่สามารถเขียนคำสั่งเข้าถึงได้ เมื่อรันโปรแกรมจาวาตัว "Java Virtual Machine (JVM)" จะทำการสร้างออบเจคต์ "Class" ของแต่ละคลาสที่มีการอ้างถึงในโปรแกรมจากไฟล์ ".class" ตามชื่อของคลาสนั้น และ ในไลบรารี่มาตรฐานของจาวาจะมีส่วนที่ให้เราสามารถสืบค้นสารสนเทศของ คลาสของออบเจคต์ในขณะรันได้ ซึ่งจะอยู่ในกลุ่มคลาส "java.lang.refect" โดยทุกออบเจคต์ในจาวาจะมีเมทอด "getClass( )" ที่จะคืนออบเจคต์ "Class" ของออบเจคต์นั้น คลาส "Class" มี โครงสร้าง ดังต่อไปนี้

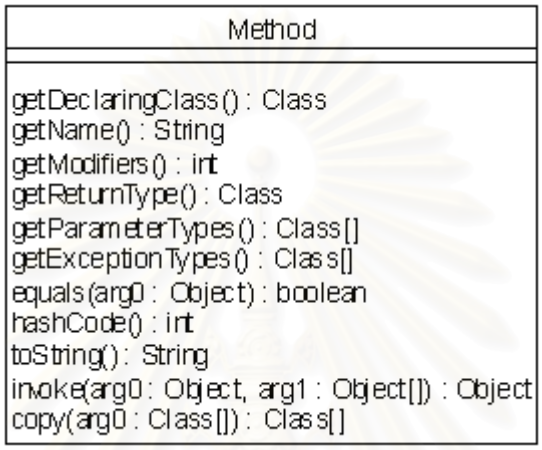


รูปที่ 2.3 อินเตอร์เฟสของคลาส “Class”

ฟังก์ชัน "getName" ทำให้ทราบชื่อของคลาส ส่วน "getMethods" และ "getMethod" จะได้เมทอดของคลาสนั้น ซึ่งเมทอดที่ได้เป็นออบเจคต์ของคลาส "Method" เราสามารถเรียกเมทอดผ่านทางออบเจคต์นี้ได้ ด้วยฟังก์ชัน "invoke" โดยอาร์กิวเมนต์ที่ส่งไปได้แก่ 'arg0' เป็นออบเจคต์ที่จะถูกเรียกและ 'arg1' เป็นอะเรย์ของอาร์กิวเมนต์ตามพารามิเตอร์ของเมทอดนั้น

รูปแบบการเรียกใช้เมทอดผ่านออบเจคต์ “Method”

method.invoke(object, parameters);



รูปที่ 2.4 อินเตอร์เฟสของคลาส “Method"

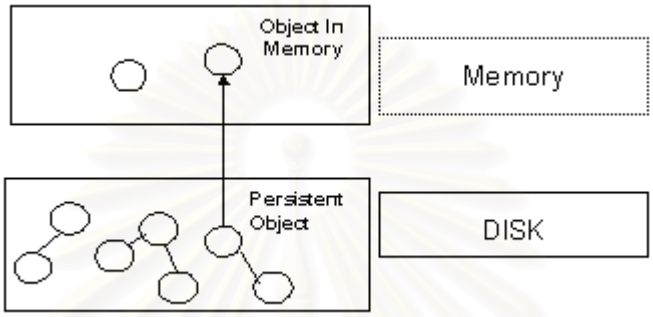
ดังนั้นจะพบว่าการเรียกใช้เมทอดผ่านทาง ออบเจคต์ "Method" นั้นจะต้องทำการรวมเอาพารามิเตอร์ไว้ในตัวแปรแบบอะเรย์(Array)ก่อน ซึ่งพารามิเตอร์ที่มีชนิดข้อมูลเป็นแบบดั้งเดิม (Primitive Type) เช่น ตัวเลข int, จำนวนจริง float จะไม่สามารถเก็บในอะเรย์รวมกับตัวแปรที่เป็นออบเจคต์ ดังนั้นจะต้องแปลงตัวแปรเหล่านี้ให้เป็นตัวแปรออบเจคต์ก่อน ด้วยการแปลงให้เป็นออบเจคต์ของคลาสสำหรับชนิดข้อมูลนั้นๆ เช่น ตัวแปร int ให้แปลงเป็นออบเจคต์ของคลาส "Integer" ส่วน float ก็ใช้คลาส "Float" ซึ่งจะพบว่าการเรียกใช้เมทอดผ่านออบเจคต์ “Method” นี้มีความยุ่งยากพอสมควร ดังนั้นในงานวิจัยนี้จะได้พยายามออกแบบ ตัวแปรเมทอดให้มีการใช้งานที่ง่ายขึ้น

2.5 ทฤษฎีที่เกี่ยวกับวัตถุถาวรและฐานข้อมูลเชิงวัตถุ

ในงานวิจัยนี้จะศึกษาการใช้งานตัวแปรเมทอดสำหรับวัตถุถาวรในระบบฐานข้อมูลเชิงวัตถุด้วย โดยจะทำให้การเชื่อมต่อของออบเจคต์แบบถาวรด้วยตัวแปรเมทอดสามารถคงอยู่ได้เมื่อปิดโปรแกรม

2.5.1 ออบเจคต์ถาวร และ ฐานข้อมูลเชิงวัตถุ

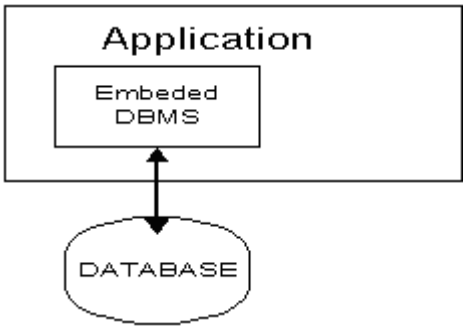
ปกติเมื่อโปรแกรมจบการทำงาน ข้อมูลของออบเจคต์ที่อยู่ในหน่วยความจำจะถูกลบทิ้งไป การจะให้ข้อมูลของออบเจคต์คงอยู่ทำได้โดยเก็บข้อมูลเหล่านั้นไว้ในหน่วยเก็บข้อมูลสำรอง เช่น เก็บลงไฟล์ ในเทคโนโลยีเชิงวัตถุมีแนวคิดของการทำ ออบเจคต์ถาวร(Persistent Object) คือ ออบเจคต์ที่สามารถคงอยู่ได้หลังจากโปรแกรมจบการทำงาน [11][12] ดังรูปที่2.5



รูปที่ 2.5 ออบเจคต์ถาวร (Persistent Object)

ข้อมูลสถานะ(State)ของออบเจคต์ถาวรสามารถที่จะถูกเก็บลงในหน่วยเก็บข้อมูลถาวรได้ ดังนั้นเมื่อโปรแกรมจบการทำงานข้อมูลเหล่านั้นก็ยังคงอยู่ และ เมื่อรันโปรแกรมใหม่อีกครั้ง ออบเจคต์นั้นก็สามารถที่จะโหลดสถานะเดิมของตนขึ้นมาใหม่และทำงานต่อเนื่องไปได้ การจัดเก็บข้อมูลของออบเจคต์ในไฟล์บนจาวาทำได้โดยใช้คุณสมบัติ “Serializability” [1] ซึ่งทำให้ ออบเจคต เก็บสถานะของตนลงไฟล์ หรือส่งผ่านระบบเครือข่ายไปยังโปรแกรมอื่นได้ แต่ก็สามารถใช้งานได้จำกัด เพราะการจัดการออบเจคต์จำนวนมากผู้เขียนต้องจะจัดการเอง ซึ่งอาจจะทำได้โดยการเก็บข้อมูลของออบเจคต์ลงในฐานข้อมูลเชิงสัมพันธ์(RDBMS) แต่ปัญหาคือข้อมูลในระบบเชิงวัตถุนั้นมีความซับซ้อน ซึ่งยากในการเก็บลงในฐานข้อมูลเชิงสัมพันธ์

จากแนวคิดของการทำออบเจคต์ถาวรได้นำไปสู่การสร้างฐานข้อมูลเชิงวัตถุ(OODBMS) ที่สามารถจัดเก็บออบเจคต์และความสัมพันธ์ระหว่างออบเจคต์ที่ซับซ้อนได้ และมีความสามารถของระบบจัดการฐานข้อมูลเหมือนกับฐานข้อมูลเชิงสัมพันธ์ เช่น สามารถค้นหา ออบเจคต์ จนถึงการสืบค้น(Query) ภายในฐานข้อมูล, การใช้ข้อมูลออบเจคต์ร่วมกัน เป็นต้น



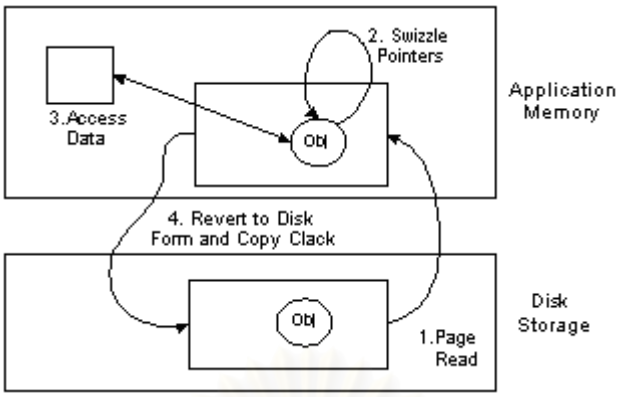
รูปที่ 2.6 ฐานข้อมูลแบบฝังตัว (Embedded DBMS)

ระบบจัดการฐานข้อมูลเชิงวัตถุส่วนหนึ่งจะอยู่ในรูปแบบของ Class Library ที่ใช้สำหรับเขียนโปรแกรมการจัดการฐานข้อมูลเชิงวัตถุ ดังนั้นปกติแล้วฐานข้อมูลเชิงวัตถุจะสร้างมาเฉพาะสำหรับภาษาที่ใช้ เช่น ฐานข้อมูลเชิงวัตถุสำหรับภาษาซีพลัสพลัส ก็สามารถใช้งานกับภาษา ซีพลัสพลัส เท่านั้น ดังนั้นฐานข้อมูลประเภทนี้ส่วนมากจึงอยู่ในรูปของ ระบบจัดการฐานข้อมูลแบบฝังตัว (Embedded DBMS) ดังแสดงในรูปที่ 2.6

ปกติระบบจัดการฐานข้อมูลเชิงวัตถุนั้นตัวจัดการฐานข้อมูลจะถูกรวมเข้าเป็นส่วนหนึ่งภายในแอปพลิเคชันเลย แต่ในฐานข้อมูลเชิงวัตถุบางตัวที่สามารถสนับสนุนการทำงานแบบไคล์เอนต์/เซิร์ฟเวอร์ (Client/Server) ก็จะมีส่วนที่แยกออกมาเพื่อทำหน้าที่จัดการข้อมูลของ ออบเจคต์ภายในเซิร์ฟเวอร์ แล้วส่งข้อมูลผ่านทางระบบเครือข่าย

2.5.2 การระบุออบเจคต์และพอยน์เตอร์สวิซซ์ลิง (Object Identify and Pointer Swizzling)

ปัญหาสำคัญอย่างหนึ่งของระบบจัดการออบเจคต์ถาวรคือ การเรียกใช้ออบเจคต์ในฐานข้อมูล ซึ่ง ออบเจคต์หนึ่งจะต้องมีข้อมูลเพียงที่เดียวในฐานข้อมูล และ การตรวจสอบ เมื่อมีการเรียกใช้ ออบเจคต์นั้นว่าพร้อมทำงานหรือไม่ ด้วยคุณสมบัติของออบเจคต์ถาวร ออบเจคต์ควรจะต้องพร้อมทำงานตลอดเวลา แต่ในความเป็นจริงถ้าระบบที่มี ออบเจคต์จำนวนมากเราไม่สามารถโหลดข้อมูลของออบเจคต์ทั้งหมดมาไว้บนหน่วยความจำได้ ดังนั้นออบเจคต์บางตัวจึงอาจจะยังไม่อยู่ในหน่วยความจำ



รูปที่ 2.7 เทคนิค Pointer Swizzle

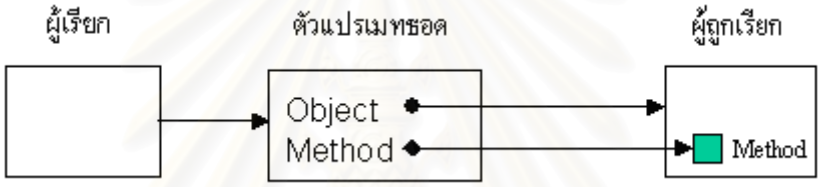
แต่ระบบจะต้องทำเสมือนว่าทุกออบเจคต์พร้อมทำงาน เมื่อมีการเรียกใช้งานออบเจคต์ที่ยังไม่อยู่ในหน่วยความจำ ระบบจำเป็นต้องมีการตรวจสอบสถานะของออบเจคต์ก่อน ถ้ายังไม่พร้อมก็ทำการโหลดออบเจคต์นั้นจากฐานข้อมูลเข้าสู่หน่วยความจำโดยอัตโนมัติ ดังนั้น ออบเจคต์ถาวรจะมีที่อยู่ 2 แห่ง คือ ที่อยู่ในหน่วยความจำและที่อยู่ในฐานข้อมูล ออบเจคต์แต่ละตัวต้องมีที่อยู่ไม่ซ้ำกัน เทคนิคของ ‘Pointer Swizzling’ [11][12] ใช้สำหรับแปลงระหว่างที่อยู่ในหน่วยควาจำและที่อยู่ในฐานข้อมูล ทำให้ทุกออบเจคต์สามารถทำงานได้เหมือนทั้งหมดอยู่บนหน่วยความจำ

ดังนั้น การสร้างตัวแปรเมทอดสำหรับออบเจคต์ถาวรในฐานข้อมูลเชิงวัตถุนั้นจำเป็นต้องระมัดระวังในการใช้งานและจะต้องมีกลไกในการตรวจสอบสถานะที่แท้จริงของออบเจคต์ถาวร และ การเก็บที่อยู่ของออบเจคต์ก็จะต้องมีการเก็บที่อยู่ทั้งบนหน่วยความจำและในฐานข้อมูลด้วย นอกจากนี้การใช้ตัวแปรเมทอดจะต้องสามารถใช้กับทั้งออบเจคต์ปกติและออบเจคต์ถาวร ซึ่งจำเป็นต้องมีกลไกในการตรวจสอบชนิดของออบเจคต์ด้วย เพื่อป้องกันข้อผิดพลาด และกลไกการทำงานต่างๆ เหล่านี้จะต้องซ่อนความซับซ้อนไว้ภายใน เพื่อให้สามารถนำไปใช้ได้ง่าย

บทที่ 3

ตัวแปรเมทอดเบื้องต้น

แนวคิดของตัวแปรเมทอดสร้างขึ้นมาเพื่อใช้แทนตัวชี้เมทอดในจาวา โดยใช้เทคนิคที่คล้ายกับ “Delegate” ใน Microsoft Visual J++ 6.0 แต่ในงานวิจัยนี้จะใช้วิธีการออกแบบและเขียนโปรแกรมที่ต่างออกไป และศึกษาถึงการใช้ตัวแปรเมทอดในรูปแบบต่างๆ ทั้งตัวแปรเมทอดแบบเดี่ยวที่อ้างถึงเมทอดเดียว ตัวแปรเมทอดแบบหลายจำนวนที่อ้างถึงเมทอดเป้าหมายได้พร้อมๆกันหลายเมทอด การเรียกใช้เมทอดผ่านตัวแปรเมทอดในแบบต่างๆ การจัดเก็บและเรียกใช้ข้อมูลตัวแปรเมทอดในไฟล์ และการใช้ตัวแปรเมทอดกับออบเจคต์ถาวรในฐานข้อมูลเชิงวัตถุ



รูปที่ 3.1 หน้าที่ของตัวแปรเมทอด

3.1 คำนิยาม ที่เกี่ยวกับ ตัวแปรเมทอด

“ตัวแปรเมทอด” (Method Variable) คือ ตัวแปรแบบวัตถุประเภทหนึ่งที่สามารถเก็บข้อมูลที่อ้างถึงเมทอดของวัตถุ และ สามารถเรียกให้เมทอดของวัตถุที่อ้างถึงทำงานได้ผ่านทางตัวแปรชนิดนี้ ตัวแปรเมทอดจะไม่พิจารณาที่คลาสหรือชนิดของวัตถุที่อ้างถึง แต่จะพิจารณาที่รูปแบบพารามิเตอร์ของเมทอดเท่านั้น ซึ่งตัวแปรเมทอดจะสามารถอ้างถึงเมทอดของออบเจคต์ที่มีการกำหนดพารามิเตอร์ที่เข้ากันได้เท่านั้น *ตัวแปรเมทอดในงานวิจัยนี้ใช้ได้กับภาษาจาวาเท่านั้น*

“คลาสตัวแปรเมทอด” (Method Variable Class) คือ คลาสที่สืบทอดมาจากคลาส “com.wtv.MethodVar” ซึ่งเป็นคลาสรากของคลาสตัวแปรเมทอดทั้งหมด ซึ่งชนิดของเมทอด และคุณสมบัติต่างๆของตัวแปรเมทอดจะถูกกำหนดในคลาสตัวแปรเมทอด การสร้างคลาสตัวแปรเมทอดจะต้องทำตามลักษณะโครงร่างที่กำหนดไว้ และ ตัวแปรเมทอดก็คือวัตถุที่สร้างขึ้นจากคลาสนี้

3.2 การใช้งานตัวแปรเมทอดเบื้องต้น

ในส่วนนี้จะอธิบายภาพรวมของตัวแปรเมทอดลักษณะการใช้งานตัวแปรเมทอดในเบื้องต้นก่อน ซึ่งจะทำให้สามารถเข้าใจถึงรายละเอียดและข้อกำหนดของตัวแปรเมทอดที่จะกล่าวในบทต่อไปได้ง่ายยิ่งขึ้น การใช้งานตัวแปรเมทอดจะมี 4 ขั้นตอน ได้แก่

1. การสร้างคลาสตัวแปรเมทอด(Declaration)
2. การสร้างตัวแปรเมทอด (Instantiation)
3. การกำหนดเมทอดให้กับตัวแปร (Assignation)
4. การเรียกใช้เมทอดผ่านตัวแปร (Invocation)

3.2.1 การสร้างคลาสตัวแปรเมทอด

ทำเพื่อกำหนดคุณสมบัติของตัวแปรเมทอดซึ่งได้แก่ ลายเซ็นเมทอด(Method Signature) หรือ รูปแบบพารามิเตอร์ของเมทอดที่ตัวแปรเมทอดจะเก็บได้ เช่น (int, int) คือ เมทอดที่รับพารามิเตอร์เป็น ตัวเลขจำนวนเต็ม 2 ตัว

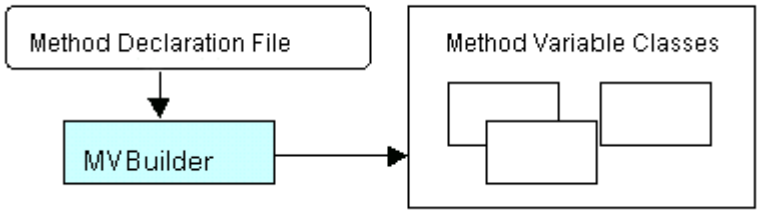
การสร้างคลาสของตัวแปรเมทอดจะต้องสืบทอดมาจากคลาสที่กำหนดไว้แล้วคือ com.wtv.MethodVar สำหรับออบเจคต์ปกติ และ com.wtv.OSMethodVar สำหรับ ออบเจคต์ถาวร โดยต้องสร้างคลาสตามรูปแบบที่กำหนดไว้ซึ่งมีความซับซ้อนพอสมควร ดังนั้นเพื่อให้สร้างคลาสของตัวแปรเมทอดได้สะดวก ในงานวิจัยนี้ได้สร้างภาษาและ เครื่องมือสำหรับนิยามคลาสของตัวแปรเมทอดขึ้น

```
package com.wtv
[IntMethod]
  //  Sample Integer method variable
  :void invoke
       int a
       int b

[StringMethod]
  :void invoke
       String  s
```

รูปที่ 3.2 ตัวอย่างไฟล์ นิยามคลาสของตัวแปรเมทอด

จากตัวอย่างเป็นการนิยามคลาสตัวแปรเมทอด 2 คลาสคือ IntMethod มีรูปแบบพารามิเตอร์ (int a, int b) และ StringMethod มีรูปแบบพารามิเตอร์ (String s) เมื่อสร้างไฟล์นิยามคลาสของตัวแปรเมทอด แล้วให้เรียกใช้โปรแกรม “Method Variable Builder” ซึ่งแบ่งเป็น 2 โปรแกรมคือ “MVBuilder” สำหรับออบเจคต์ปกติ และ “OSMVBuilder” สำหรับออบเจคต์ถาวร เพื่อคอมไพล์ให้เป็นคลาสในภาษาจาวา



รูปที่ 3.3 การสร้างคลาสตัวแปรเมทอด

ตัวอย่างการเรียก >

MVBuilder -c sample.def

จากตัวอย่างรูปที่ 3.2 เมื่อสร้างคลาสแล้วจะได้ไฟล์ทั้งหมด 4 ไฟล์ได้แก่

- IntMethod.java และ StringMethod.java เป็นไฟล์ภาษาจาวาที่ยังไม่คอมไพล์
- IntMethod.class และ StringMethod.class เป็นไฟล์ที่คอมไพล์แล้ว

ในชุดไลบรารีของตัวแปรเมทอดผู้วิจัยได้มีการสร้างคลาสของตัวแปรเมทอดสำหรับใช้เพื่อสร้างเหตุการณ์(Event) มาตรฐานไว้แล้ว ซึ่งหากไม่ต้องการสร้างเหตุการณ์ใหม่ที่มีรูปแบบของพารามิเตอร์ต่างออกไป ก็ไม่จำเป็นต้องสร้างคลาสตัวแปรเมทอดขึ้นมาใหม่

3.2.2 การสร้างตัวแปรเมทอด

เมื่อได้คลาสตัวแปรเมทอดที่มีคุณสมบัติตามแบบที่ต้องการแล้ว ก็สามารถสร้างตัวแปรจากคลาสนั้นได้ เหมือนการสร้างตัวแปรจากคลาสอื่นๆโดยทั่วไป ด้วยคำสั่งเช่น

```
IntMethod  im = new IntMethod()
StringMethod sm = new StringMethod ()
```

3.2.3 การกำหนดข้อมูลเมทอดให้กับตัวแปร

สำหรับการกำหนดเมทอดให้กับตัวแปรนั้นทำได้โดยเรียกใช้ฟังก์ชัน **set** ซึ่งมีรูปแบบดังนี้

boolean set (**Object** obj , **String** methodname)

obj คือ วัตถุเป้าหมาย

methodname คือ ชื่อของเมทอดที่ต้องการ ให้ตัวแปรเมทอดอ้างไป

การทำงานของฟังก์ชัน “set” จะใช้ชื่อเมทอดให้การค้นหาที่อยู่ของเมทอดในวัตถุเป้าหมาย ซึ่งถ้าหากเมทอดเป้าหมายเข้ากับชนิดของตัวแปรเมทอดนั้นได้จะคืนค่า true ถ้าเข้ากันไม่ได้ จะคืนค่าเป็น false

ตัวอย่างการใช้

```
Class  Test {
     void  show(int x,int y) {
     System.out.println(x+y);
     }
     void  print(String s) ;
}
  ….
IntMethod  im = new IntMethod();
Test  t1 = new Test();

im.set(t1,"show");       // return true
im.set(t1,"print");      // return false
```

ซึ่งจะเห็นว่า เมื่อกำหนด t1.**show** ให้กับ ตัวแปร im จะคืนค่าเป็นจริง เพราะว่า เมทอด show มีรูปแบบพารามิเตอร์เป็น (int x,int y) ซึ่งสามารถเข้ากันได้กับ รูปแบบพารามิเตอร์ของ IntMethod แต่ เมื่อกำหนด t1.**print** ให้กับตัวแปร im จะได้ผลลัพธ์เป็นเท็จ และ im จะไม่เก็บ t1.print ไว้ เพราะ print มีรูปแบบพารามิเตอร์เป็น (String s) ไม่สามารถเข้ากับรูปแบบพารามิเตอร์ของ IntMethod ได้ ดังนั้นผลลัพธ์จากคำสั่งทั้ง 2 บรรทัด จะมีเพียงเมทอด **show** เท่านั้นที่ถูกเก็บไว้ในตัวแปร **im**

3.2.4 การเรียกใช้เมทอดผ่านตัวแปร

การเรียกใช้เมทอดจะทำผ่านทางฟังก์ชัน “invoke” ของตัวแปรเมทอดนั้น ซึ่งรูปแบบการเรียก “invoke” จะถูกกำหนดไว้ในคลาส

ตัวอย่าง : จาก 3.2.3 ถ้า *im* เก็บข้อมูลอ้างอิงไปยังเมทอด *t1.show*

im.invoke(5,3); ***//*** t1.show(5,3) จะถูกเรียก และ จะได้ผลลัพธ์เป็น 8

โดยสรุปการใช้งานตัวแปรเมทอดมี 4 ขั้นตอนดังที่กล่าวมา สำหรับการนำตัวแปรเมทอดไปใช้ในการสร้างการควบคุมเหตุการณ์ (Event) จะพบว่า ขั้นตอนที่ 1 การสร้างคลาสตัวแปร เมทอด, ขั้นตอนที่ 2 การสร้างตัวแปรเมทอด และ ขั้นที่ 4 การเรียกใช้ตัวแปรเมทอด จะถูกใช้ในขั้นตอนการออกแบบคอมโพเนนท์ เป็นขั้นตอนที่ผู้ผลิตคอมโพเนนท์ต้องเข้าใจ ส่วนผู้ใช้งานคอมโพเนนท์จะได้ใช้งานเฉพาะในขั้นที่ 3 คือการกำหนดค่าเมทอดเป้าหมายให้กับเหตุการณ์เพื่อเขียนเมทอดควบคุมเหตุการณ์เท่านั้น ดังนั้นสิ่งที่ผู้ใช้คอมโพเนนท์ต้องเข้าใจคือ มีเหตุการณ์ใดในคอมโพเนนท์บ้าง และเหตุการณ์นั้นมีรูปแบบพารามิเตอร์อย่างไร เพื่อจะได้กำหนดเมทอดเพื่อควบคุมเหตุการณ์เหล่านั้นได้อย่างถูกต้อง

บทที่ 4

การออบแบบตัวแปรเมทอด

แนวคิดของตัวแปรเมทอดมาจากการพยายามหาวิธีการเพื่อใช้แทนเทคนิคตัวชี้เมทอด (Function Method) สำหรับภาษาจาวา ซึ่งตัวชี้เมทอดมีการใช้ในหลาย ๆ ภาษาเช่น ซีพลัสพลัส และ ออบเจคต์ปาสคาล ซึ่งมีการใช้ตัวชี้เมทอดในเขียนโปรแกรมเพื่อสร้างฟังก์ชันเรียกกลับ (CallBack-Function) และในการเขียนโปรแกรมควบคุมเหตุการณ์ในเครื่องมือพัฒนาโปรแกรมสมัยใหม่

การเขียนโปรแกรมเชิงวัตถุฟังก์ชันจะอยู่ในรูปของเมทอดซึ่งเป็นส่วนหนึ่งของวัตถุ ดังนั้นตัวแปรเมทอดต้องสนับสนุนคุณสมบัติของเมทอดตามแนวคิดของโปรแกรมเชิงวัตถุ การเรียกเมทอดผ่านทางตัวแปรเมทอดจึงเป็นเหมือนการส่งข่าวสารไปยังวัตถุ ตัวแปรเมทอดทำหน้าที่เป็นผู้ส่งข่าวสารหรือตัวกลางระหว่างผู้รับและผู้ส่ง โดยที่ทั้งผู้รับและผู้ส่งไม่จำเป็นต้องมีความสัมพันธ์กันมาก่อน การออกแบบตัวแปรเมทอดจะมุ่งเน้นให้สามารถใช้งานได้ง่ายและมีความยืดหยุ่นในการใช้งานสูง

4.1 คุณสมบัติของตัวแปรเมทอด

ในวิทยานิพนธ์นี้ ได้กำหนดให้ตัวแปรเมทอดมีคุณสมบัติดังต่อไปนี้

4.1.1 สามารถกำหนดชนิดของเมทอดที่จะเก็บได้ โดยระบุไว้ในคลาสของตัวแปรเมทอด โดยสิ่งที่ใช้สำหรับสร้างชนิดของเมทอดได้แก่ ชนิดพารามิเตอร์ที่ใช้ในการเรียก และชนิดผลลัพธ์ ตัวแปรเมทอดจะสามารถอ้างถึงไปยังเมทอดที่มีชนิดของพารามิเตอร์และผลลัพธ์ ที่เข้ากันได้กับที่กำหนดไว้เท่านั้น

4.1.2 ตัวแปรเมทอดเป็นออบเจคต์ที่สร้างจากคลาสตัวแปรเมทอดและสามารถเก็บข้อมูลในการอ้างถึงเมทอดของออบเจคต์ใดๆ โดยไม่สนใจคลาสหรืออินเตอร์เฟต (Interface) โดยสนใจเฉพาะลายเซนต์ของเมทอดที่มีชนิดตรงกับที่กำหนดตามข้อ 4.1.1 โดยสิ่งที่ใช้ในการกำหนดค่าให้ตัวแปรเมทอด คือ ออบเจคต์ และ ชื่อของเมทอดที่ต้องการให้เก็บ

4.1.3 สามารถเรียกใช้เมทอดผ่านตัวแปรนั้นได้ โดยเรียกผ่านฟังก์ชัน “invoke” โดยวิธีการส่งพารามิเตอร์จะเป็นไปตามที่กำหนดไว้ในข้อ 4.1.1 ซึ่งตัวแปรเมทอดจะส่งต่อการเรียกนั้น ไปยัง เมทอดของออบเจคต์ที่ตัวแปรมีอ้างอิงถึง และ คืนค่าผลลัพธ์กลับมายังผู้เรียก โดยอัตโนมัติ

4.1.4 สามารถตรวจสอบสถานะและสารสนเทศของตัวแปรเมทอดระหว่างรัน ดังต่อไปนี้

4.1.4.1 ตัวแปรเมทอดมีข้อมูลอ้างถึง ออบเจคต์และเมทอดใดหรือไม่

4.1.4.2 ชนิดของตัวแปรเมทอด ได้แก่ ชนิดของพารามิเตอร์ และ ชนิดผลลัพธ์

4.1.5 สามารถจัดเก็บข้อมูลของตัวแปรเมทอดลงในไฟล์ ซึ่งได้แก่ ข้อมูลอ้างถึงออบเจคต์ และ เมทอดที่เก็บในตัวแปรและสามารถอ่านข้อมูลตัวแปรเมทอดจากไฟล์มาใช้งานในขณะรันโปรแกรม

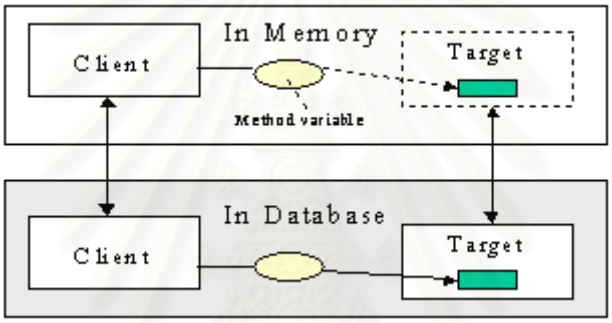
4.1.6 ตัวแปรเมทอดสำหรับฐานข้อมูลเชิงวัตถุสามารถเก็บและเรียกใช้เมทอดของออบเจคต์แบบถาวร ในฐานข้อมูลเชิงวัตถุ

4.2 ประเภทของตัวแปรเมทอด

4.2.1 แบ่งตามการใช้งาน ได้แก่

4.2.1.1 ตัวแปรเมทอดสำหรับใช้กับออบเจคต์แบบปกติ อยู่เฉพาะในหน่วยความจำ

4.2.1.2 ตัวแปรเมทอดสำหรับใช้กับออบเจคต์แบบถาวร ในฐานข้อมูลเชิงวัตถุ



รูปที่ 4.1 ตัวแปรเมทอดกับออบเจคต์ถาวรในฐานข้อมูล

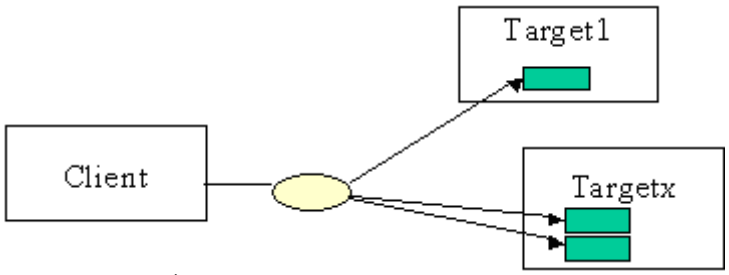
4.2.2 แบ่งตามจำนวนเมทอดที่สามารถเก็บได้ ได้แก่

4.2.2.1 ตัวแปรเมทอดแบบเดี่ยว (Single Method Variable)

คือตัวแปรเมทอดที่สามารถอ้างถึง เมทอดปลายทางได้เพียงหนึ่งเมทอดในเวลาเดียว การกำหนดเมทอดปลายทางใหม่ให้กับตัวแปร จะเป็นการเขียนทับข้อมูลเดิม การเรียกตัวแปรเมทอดแบบเดี่ยวสามารถให้คืนค่าผลลัพธ์ได้

4.2.2.2 ตัวแปรเมทอดแบบหลายจำนวน (Multiple Method Variable)

คือตัวแปรเมทอดที่สามารถอ้างถึงเมทอดปลายทางได้หลายเมทอดพร้อมกัน เมื่อกำหนดเมทอดปลายทางให้กับตัวแปร จะเป็นการเพิ่มข้อมูลการอ้างอิงไปยังเมทอดใหม่ด้วย โดยข้อมูลการอ้างถึงเมทอดที่มีอยู่เดิมยังอยู่ การเรียกให้ตัวแปรเมทอดทำงานจะมีผลให้เมทอดทั้งหมดที่ตัวแปรอ้างถึงทำงาน การเรียกตัวแปรเมทอดแบบหลายจำนวนจะไม่สามารถคืนผลลัพธ์ได้

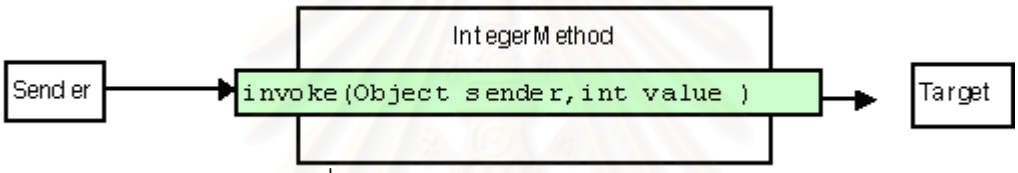


รูปที่ 4.2 ตัวแปรเมทอดแบบหลายจำนวน

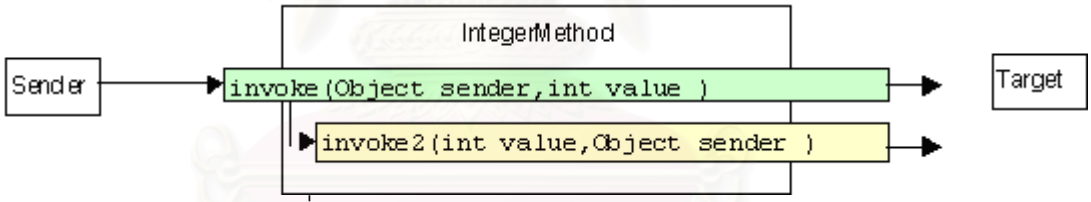
4.2.3 <u>แบ่งตามรูปแบบพารามิเตอร์(ลายเซ็นเมทอด)</u> ได้แก่

4.2.3.1 ตัวแปรเมทอดแบบรูปเดียว (Single Signature)

คือตัวแปรเมทอดที่มีรูปแบบของพารามิเตอร์ เพียงรูปแบบเดียว เช่น (Object , int) เมทอดที่ตัวแปรสามารถเก็บได้จะต้องมีพารามิเตอร์เป็น (Object,int) เหมือนกัน



รูปที่ 4.3 ตัวอย่างตัวแปรเมทอดรูปเดียว



รูปที่ 4.4 ตัวอย่างตัวแปรเมทอดแบบหลายรูป

4.2.3.2 ตัวแปรเมทอดแบบหลายรูป (Multiple Signature)

คือตัวแปรเมทอดที่มีการกำหนดรูปแบบของพารามิเตอร์ไว้มากกว่า 1 รูปแบบ โดยตัวแปรเมทอดแบบหลายรูปจะต้องมีรูปแบบพารามิเตอร์หลักหรือรูปแบบปกติเหมือนกับตัวแปรเมทอดรูปเดียว และ สามารถมี *รูปแบบสำรอง* เพิ่มได้อีก ซึ่งรูปแบบสำรองเป็นรูปแบบพารามิเตอร์ที่มีการนิยามเพิ่มเติม โดยนิยามด้วยการสร้างเมทอดที่มีชื่อไม่ซ้ำกับแบบหลัก เพื่อยอมให้เมทอดปลายทางที่มีพารามิเตอร์ไม่ตรงกับรูปแบบหลักสามารถใช้รูปแบบสำรองในการเชื่อมต่อได้ เช่น ในรูปที่ 4.4

invoke (Object , int) เป็นรูปแบบหลัก

invoke2 (int, Object) เป็นรูปแบบสำรอง

โดยการนิยามรูปแบบสำรองนั้น จะต้องกำหนดให้พารามิเตอร์ของรูปแบบสำรองสามารถรับพารามิเตอร์จากรูปแบบหลักได้ด้วย และ ลำดับของพารามิเตอร์จะต้องไม่ซ้ำซ้อนกับรูปแบบหลัก เมื่อกำหนดเมทอดปลายทางแล้วตัวแปรเมทอดจะเลือกรูปแบบพารามิเตอร์ที่เหมาะสมกับเมทอดปลายทางที่มันอ้างอิงถึง เพื่อให้การส่งพารามิเตอร์ทำได้อย่างถูกต้อง สามารถดูตัวอย่างการสร้างคลาสตัวแปรเมทอดแบบหลายรูปได้ใน ภาคผนวก ข

การที่ตัวแปรเมทอดสามารถมีรูปแบบพารามิเตอร์ได้หลายรูปแบบจะทำให้ตัวแปรเมทอดสามารถเชื่อมต่อกับเมทอดปลายทางได้หลายรูปแบบมากขึ้น การเรียกให้เมทอดปลายทางทำงานผ่านตัวแปรเมทอดจะต้องเรียกผ่านทางรูปแบบหลัก ซึ่งปกติจะนิยามไว้ในเมทอด invoke() และ ภายในเมทอด invoke จะมีกลไกเพื่อส่งต่อพารามิเตอร์ไปยังรูปแบบสำรองที่เหมาะสมกับเมทอดปลายทาง

4.3 ข้อกำหนดสำหรับการตรวจสอบชนิดของตัวแปรเมทอด

คลาสของตัวแปรเมทอดจะมีการกำหนดรูปแบบพารามิเตอร์ของเมทอดที่ตัวแปรเมทอดนี้จะสามารถอ้างถึงได้ หัวข้อนี้จะเป็นการกำหนดกฎเกณฑ์สำหรับตรวจสอบชนิดของเมทอดก่อนที่จะเก็บในตัวแปร

<u>กำหนดให้</u>

- Class mtTypes[] เป็นชนิดของตัวแปรเมทอดซึ่งเป็นอะเรย์ของชนิดพารามิเตอร์ ในภาษา จาวาชนิดของพารามิเตอร์จะเป็นออบเจคต์ที่มาจากคลาส “Class”
- Class targetTypes[] เป็นอะเรย์ชนิดพารามิเตอร์ของ เมทอดเป้าหมายที่ต้องการเก็บในตัวแปรเมทอด
- size(mtTypes) เป็นจำนวนพารามิเตอร์ของชนิดตัวแปรเมทอด
- size(targetTypes) เป็นจำนวนพารามิเตอร์ของเมทอดเป้าหมาย
- Class mtResult เป็นชนิดของผลลัพธ์ ของตัวแปรเมทอด
- Class targetResult เป็นชนิดของผลลัพธ์ของเมทอดเป้าหมาย

วิธีการตรวจสอบชนิดจะมี 2 วิธีคือ การตรวจสอบแบบสมบรูณ์ (Full Type Checking) และการตวรจสอบชนิดแบบบางส่วน (Partial Type Checking) โดยในการตรวจสอบของทั้งสองวิธีจะขึ้นกับประเภทของตัวแปรเมทอดว่ามีรูปเดียวหรือหลายรูป โดยผลการตรวจสอบตัวแปรเมทอดจะสามารถเก็บข้อมูลอ้างถึงเมทอดเป้าหมายได้เมื่อเงื่อนไขที่กำหนดทุกข้อเป็นจริง

4.3.1 <u>เงื่อนไขการตรวจสอบชนิด แบบสมบูรณ์ (Full Type Checking)</u>

1. size(mtTypes) = size(targetTypes)
2. targetTypes[n] ≈ mtTypes[n] เมื่อ n = 0 ถึง size(mtTypes)-1
   แต่ละสมาชิกใน targetTypes[] สามารถถูก กำหนดค่าได้จาก สมาชิกของ mtTypes[] ในตำแหน่งเดียวกัน
   a ≈ b (a สามารถถูกกำหนดค่าจาก b) เมื่อ ตรงกับเงื่อนไขข้อใดข้อหนึ่ง
   - 2.1 b เป็นคลาสเดียวกับ a
   - 2.2 b เป็นคลาสลูกของ a
   - 2.3 a เป็น Interface เดียวกับ b หรือ b เป็นคลาสที่ Implement a
   - 2.4 b เป็น Primitive type เช่น int, long, short, char, boolean, byte, float, double แล้ว a เป็น คลาสคู่ของ b เช่น int คู่กับ Integer, char คู่กับ Charecter
3. mtResult ≈ targetResult ถ้า mtResult ไม่ใช่ void (มีการคืนค่าผลลัพธ์)
   สำหรับตัวแปรเมทอดแบบหลายจำนวนจะไม่มีการคืนค่าผลลัพธ์ ไม่ต้องตรวจสอบชนิดผลลัพธ์ของเมทอดเป้าหมาย

4.3.2 <u>เงื่อนไขการตรวจสอบ แบบบางส่วน (Partial Type Checking)</u>

เป็นเงื่อนไขพิเศษสำหรับตัวแปรเมทอดที่ยอมให้ พารามิเตอร์ของเมทอดเป้าหมายมีจำนวนน้อยกว่า จำนวนพารามิเตอร์ที่กำหนดในตัวแปรเมทอด

1. size(mtTypes) ≥ size(targetTypes)
2. targetTypes[n] ≈ mtTypes[n] เมื่อ n = 0 ถึง size(targetTypes)-1
   แต่ละสมาชิกใน targetTypes[] สามารถถูก กำหนดค่าได้จาก สมาชิกของ mtTypes[] ในตำแหน่งเดียวกัน
   a ≈ b (a สามารถถูกกำหนดค่าจาก b) เมื่อ ตรงกับเงื่อนไขข้อใดข้อหนึ่ง
   - 2.1. b เป็นคลาสเดียวกับ a
   - 2.2. b เป็นคลาสลูกของ a
   - 2.3. a เป็น Interface เดียวกับ b หรือ b เป็นคลาสที่ Implement a
   - 2.4. b เป็น Primitive type เช่น int, long, short, char, boolean, byte, float, double แล้ว a เป็น คลาสคู่ของ b เช่น int คู่กับ Integer, long คู่กับ Long, char คู่กับ Charector

3. mtResult ≈ targetResult ถ้า mtResult ไม่ใช่ void (มีการคืนค่าผลลัพธ์) สำหรับตัวแปรเมทอดแบบหลายจำนวนจะไม่มีการคืนค่าผลลัพธ์ ไม่ต้องตรวจสอบชนิดผลลัพธ์ของเมทอดเป้าหมาย

ทั้งตัวแปรเมทอดแบบรูปเดียวที่มีพารามิเตอร์ 1 แบบ และตัวแปรเมทอดแบบหลายรูปที่มีรูปแบบพารามิเตอร์มากกว่า 1 แบบ จะใช้การวิธีการตรวจสอบเงื่อนไขเหมือนกัน แต่จะต่างที่ตัวแปรเมทอดแบบหลายรูปจะมี รูปแบบพารามิเตอร์แบบสำรองเพิ่มมาจากเดิมมีรูปแบบหลักเพียงรูปแบบเดียว โดยรูปแบบสำรองจะอยู่ในรูปของลิสต์ (List) ของ Class mtTypes[] เงื่อนไขการตรวจสอบเมทอดเป้าหมายจะต้องเข้ากับรูปแบบพารามิเตอร์แบบใดแบบหนึ่ง โดยการตรวจสอบจะเริ่มจากรูปแบบหลัก ก่อนถ้าไม่สามารถเข้ากับรูปแบบหลักก็ตรวจสอบกับรูปแบบสำรองในลิสต์ที่ละรูปแบบ ไปจนกว่าจะพบรูปแบบที่เหมาะสม ถ้าไม่มีรูปแบบใดเหมาะสมเลยก็ให้แจ้งข้อผิดพลาด แสดงว่ารูปแบบพารามิเตอร์ของเมทอดเป้าหมายไม่สามารถเข้ากับของตัวแปรเมทอดได้ ตัวแปรเมทอดก็จะไม่สามารถเก็บข้อมูลอ้างถึงเมทอดนั้นได้

ตัวอย่างการตรวจสอบ

1. **ตัวแปรเมทอดแบบรูปเดียว**

เมื่อ mtTypes[] = [int , Object] , mtResult = int

ในตารางที่ 4.1 จะแสดงตัวอย่างเมทอดที่สามารถเข้ากับ mtTypes ได้เมื่อทำการตรวจสอบชนิดแบบสมบูรณ์ ส่วนตารางที่ 4.2 จะแสดงตัวอย่างเมทอดที่สามารถเข้ากับ mtTypes ได้เมื่อทำการตรวจสอบชนิดแบบบางส่วน

ตารางที่ 4. 1 ตัวอย่างเมทอดเป้าหมายที่ผ่านการตรวจสอบแบบสมบูรณ์

TargetTypes	targetResult
ตัวแปรเมทอดแบบเดี่ยว [int,Object] [Integer,Object]	int Integer
ตัวแปรเมทอดแบบหลายจำนวน [int,Object] [Integer,Object]	any type

ตารางที่ 4. 2 ตัวอย่างเมทอดเป้าหมายที่ผ่านการตรวจสอบแบบบางส่วน

TargetTypes	targetResult
ตัวแปรเมทอดแบบเดี่ยว [int,Object] [int] [Integer,Object] [Integer] []	 int Integer
ตัวแปรเมทอดแบบหลายจำนวน [int,Object] [int] [Integer,Object] [Integer] []	 any type

2. **ตัวแปรเมทอดแบบหลายรูป**

mtTypes = { [int , Object] , [Object,int] } , mtResult = int

สำหรับตัวแปรเมทอดแบบหลายรูป mtTypes สามารถมีรูปแบบได้มากกว่า 1 รูปแบบ ในตารางที่ 4.3 และ 4.4 จะแสดงเมทอดที่สามารถเข้ากับรูปแบบใน mtTypes โดยการตรวจสอบชนิดแบบสมบูรณ์ และ แบบบางส่วนตามลำดับ

ตารางที่ 4. 3 เมทอดเป้าหมายที่ผ่านการตรวจสอบแบบสมบูรณ์ สำหรับตัวแปรเมทอดแบบหลายรูป

TargetTypes	targetResult
[int,Object] [Integer,Object] [Object,int] [Object,Integer]	int Integer

ตารางที่ 4. 4 เมทอดเป้าหมายที่ผ่านการตรวจสอบแบบบางส่วน สำหรับตัวแปรเมทอดแบบหลายรูป

TargetTypes	targetResult
[int,Object] [Integer,Object] [Object,int] [Object,Integer] [int] [Integer] [Object] []	Int Integer

จากตัวอย่างจะพบว่า ตัวแปรเมทอดแบบหลายรูปจะสามารถเชื่อมต่อไปยังเมทอดเป้าหมายได้รูปแบบมากกว่า และ การใช้วิธีตรวจสอบแบบบางส่วนจะเพิ่มจำนวนรูปแบบเมทอดเป้าหมายที่สามารถเข้ากับตัวแปรเมทอดมากขึ้น แต่การใช้วิธีตรวจสอบแบบบางส่วนจะไม่สามารถมั่นใจได้ว่าเมทอดปลายทางทั้งหมดจะเป็นเป้าหมายที่แท้จริง การตรวจสอบแบบสมบูรณ์เปรียบได้กับการคัดเลือกผู้รับว่าจะต้องรับทุกอย่างที่ส่งให้ ส่วนแบบบางส่วนผู้รับสามารถเลือกรับข้อมูลเพียงบางส่วนที่สนใจไปเท่านั้น โดยที่ผู้ส่งจะส่งพารามิเตอร์ไปเท่ากับจำนวนที่ผู้รับต้องการ

4.4 ข้อกำหนดสำหรับนิยามคลาสตัวแปรเมทอด

เพื่อให้สามารถสร้างคลาสตัวแปรเมทอดได้สะดวกในวิทยานิพนธ์นี้จึงได้ออกแบบภาษาสำหรับการนิยามคลาสตัวแปรเมทอด และ สร้างเครื่องมือสำหรับแปลงข้อมูลการนิยามคลาสตัวแปรเมทอด ให้เป็นภาษาจาวาโดยอัตโนมัติ ซึ่งภาษานี้จะสนับสนุนการนิยามคลาสตัวแปรเมทอดประเภทต่างๆ ตามหัวข้อที่ 4.2

4.4.1 รูปแบบของภาษา

โครงสร้างของภาษานิยามจะแบ่งเป็น 5 ส่วนได้แก่

1. ข้อมูลส่วนหัวสากล (Global Header) อยู่บนสุดของไฟล์ เพราะสามารถนิยามตัวแปรเมทอดหลายๆ คลาสพร้อมกันในไฟล์เดียวกันได้ ดังนั้นข้อมูลในส่วนนี้จะถูกนำไปเขียนไว้ในไฟล์ตัวแปรเมทอดของแต่ละคลาส ซึ่งปกติส่วนนี้ใช้เขียนคำสั่ง **import, package** เป็นต้น
2. ชื่อคลาสตัวแปรเมทอด (Method Variable Class Name) การระบุชื่อของคลาสจะใช้รูปแบบ **[**ClassName**]** เช่น [IntMethod] ชื่อคลาสจะเป็นตัวระบุจุดเริ่มต้นข้อมูลนิยามตัวแปรเมทอดแต่ละคลาส
3. ส่วนหัวเฉพาะคลาส (Class Header) คล้ายกับส่วนที่ 1 แต่จะถูกนำไปเขียนในไฟล์ของคลาสเดียวเท่านั้น
4. ชื่อเมทอดระบุชนิด (Type Name) สำหรับกำหนดชื่อเมทอดที่จะใช้เป็นชนิดของตัวแปรเมทอด ในตัวแปรเมทอดแบบหลายรูปสามารถมีหลายชื่อได้ โดยชื่อแรกจะเป็น เมทอดหลัก รูปแบบใช้เครื่องหมาย : นำหน้าชื่อที่ต้องการเช่น :invoke ซึ่งในวิทยานิพนธ์นี้จะกำหนดให้เมทอดที่เป็นรูปแบบหลักควรใช้ชื่อว่า **invoke** เพื่อให้ง่ายในการจดจำ แต่ผู้ใช้ก็สามารถตั้งชื่อเมทอดหลักเป็นชื่ออื่นได้ โดยเมทอดหลักจะเป็นเมทอดที่ประกาศไว้เป็น

อันดับแรก สำหรับเมทอดลำดับถัดไปจะเป็นเมทอดสำรอง ซึ่งการตั้งชื่อเมทอดทั้งหมดในคลาสเดียวกันจะต้องไม่ใช้ชื่อซ้ำกัน

5.รายการพารามิเตอร์ (Parameters List) สำหรับระบุชื่อและชนิดของพารามิเตอร์ สำหรับตัวแปรเมทอดที่มีหลายรูปแบบ รายการพารามิเตอร์ในรูปแบบสำรองจะต้องระบุ วิธีการแปลงพารามิเตอร์ที่มาจากรูปแบบหลักด้วย โดยใช้เครื่องหมายเท่ากับ (=) เช่น int value = e.getValue() เมื่อ value เป็นพารามิเตอร์ในเมทอดสำรอง และ e เป็นพารามิเตอร์ในเมทอดหลัก

ตัวอย่าง (ตัวเลขในวงเล็บด้านซ้าย แสดงหมายเลขส่วน)

```
(1)  // Method Varible class for AWT Event Model
          package com.wtv.method
          import java.awt.*
          import java.awt.event.*
(2)  [TadjustmentEvent]
(3)  // TadjustmentEvent for Scrollbar component
     //  onValueChanged,  onValueChanged,   onUnitIncrement
     //  onUnitDecrement, onBlockIncrement, onBlockDecrement, onTrack;
(4)   :void invoke
(5)    AdjustmentEvent e
(4)   :void invoke1
(5)    int value           = e.getValue()
       int type            = e.getAdjustmentType()
       Adjustable sender   = e.getAdjustable()
(4)   :void invoke2
       Adjustable sender   = e.getAdjustable()
       int value           = e.getValue()
       int type            = e.getAdjustmentType()
(2)  [TItemEvent]
(3)  // TItemEvent for List component
     //   onChanged
(4)   :void invoke
(5)      ItemEvent e
(4)   :void invoke1
(5)      Object item      = e.getItem()
         int state        = e.getStateChange()
         Object sender    = e.getSource()
(4)   :void invoke2
(5)      Object item      = e.getItem()
         Object sender    = e.getSource()
         int state        = e.getStateChange()
(4)   :void invoke3
(5)      int state        = e.getStateChange()
         Object item      = e.getItem()
         Object sender    = e.getSource()
```

จากตัวอย่างมีการนิยามคลาสตัวแปรเมทอด 2 คลาสคือ TAdjustmentEvent และ TItemEvent เมื่อนำไปประมวลผลจะได้ไฟล์ 2 ไฟล์คือ TAdjustmentEvent.java และ TItemEvent.java สำหรับคำสั่ง

```
:void invoke1
  int value           = e.getValue()
  int type            = e.getAdjustmentType()
  Adjustable sender   = e.getAdjustable()
```

จะระบุให้ทราบว่า invoke1 เป็นรูปแบบสำรอง ซึ่งพารามิเตอร์ของ invoke1 จะถูกส่งมาจากพารามิเตอร์ของรูปแบบหลัก (invoke) เช่น value ของ invoke1 คือ e.getValue() ของ invoke ซึ่งจะทำให้ตัวแปรเมทอดแบบหลายรูปสามารถส่งต่อพารามิเตอร์ได้อย่างถูกต้อง สำหรับตัวอย่างไฟล์นิยามที่สมบูรณ์สามารถดูได้ในภาคผนวก ข.

ในตัวอย่าง สำหรับคลาส `TAdjustmentEvent` จะมีรูปแบบพารามิเตอร์ทั้งหมด 3 รูปแบบคือ

```
void invoke(AdjustmentEvent e);
void invoke1(int value,int type,Adjustable sender);
void invoke2(Adjustable sender,int value,int type);
```

การเรียกใช้ผู้เรียกจะต้องเรียกผ่านทางเมทอด invoke และกลไกภายใน invoke จะเป็นตัวเลือกว่าจะส่งต่อพารามิเตอร์ไปยังปลายทางด้วยรูปแบบใด เช่น ae เป็น AdjustmentEvent ถ้าเรียก *ae.invoke(e)* แล้ว เมทอดปลายทางมีรูปแบบพารามิเตอร์เป็น *(int,int,Object)* แล้ว *ae.invoke1(e.getValue(),e.getAdjustmentType(),e.getAdjustable())* จะถูกเรียกให้ทำงานต่อไป ซึ่งพารามิเตอร์ก็จะถูกส่งไปยังเมทอดปลายทางได้อย่างถูกต้อง

การนิยามรูปแบบสำรองต้องระมัดระวังในการกำหนดวิธีการแปลงพารามิเตอร์จากเมทอดหลัก ค่าที่ใช้กำหนดให้กับพารามิเตอร์แต่ละตัวในรูปแบบสำรองจะต้องเป็นนิพจน์ที่มีการเขียนตามหลักภาษาจาวา ผลลัพท์จากนิพนธ์นั้นจะต้องเป็นข้อมูลชนิดเดียวกับชนิดของพารามิเตอร์ ถ้ามีการกำหนดผิด เมื่อนำไปสร้างคลาสตัวแปรเมทอดเป็นไฟล์ .java ก็จะไม่สามารถนำไปคอมไพล์ได้ เช่น ตัวอย่างการประกาศรูปแบบเมทอดสำรองที่ผิด

```
1 :void invoke1
2   int value           = e.getValue(
3   int type            = e
4   Adjustable sender   = e.getAdjustable();
```

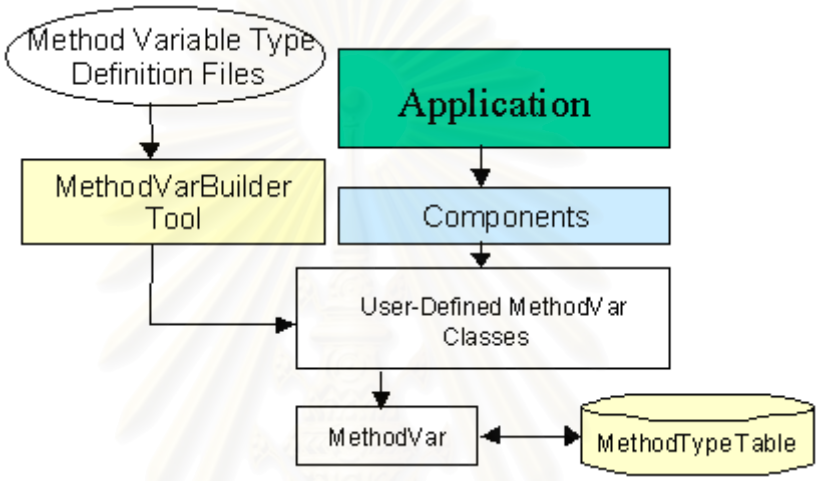
บรรทัดที่ 2 ผิดเพราะเครื่องหมายวงเล็บไม่ครบ
บรรทัดที่ 3 ผิดเพราะ e ไม่ใช่ข้อมูลแบบ int จึงไม่สามารถแทนค่าได้
บรรทัดที่ 4 ผิดที่เครื่องหมาย ; (simicolon)

บทที่ 5

การพัฒนาตัวแปรเมทอด

5.1. โครงสร้างของระบบ

5.1.1 ส่วนประกอบ



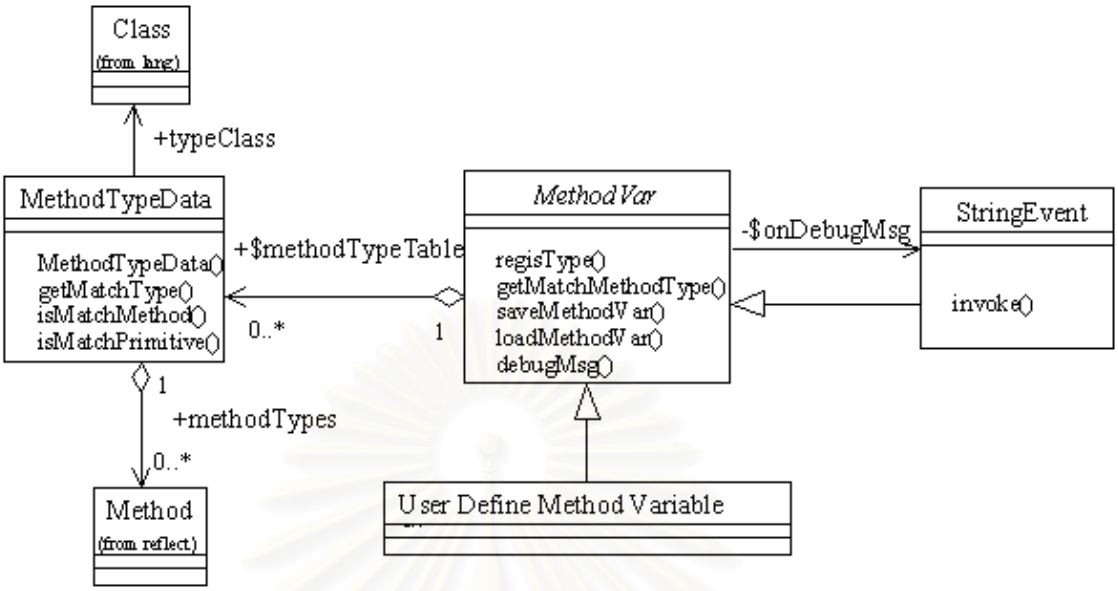
รูปที่ 5.1 แบบจำลองตัวแปรเมทอด

การใช้งานตัวแปรเมทอดจะมีส่วนประกอบดังรูปที่ 5.1 โปรแกรมประยุกต์และคอมโพเนนท์ (Component) จะใช้งานตัวแปรเมทอดในการสร้างส่วนจัดการเหตุการณ์ โดยต้องมีการสร้างคลาสตัวแปรเมทอด (User-Defined MethodVar Class) ซึ่งจะต้องสืบทอดมาจาก คลาส “MethodVar” ซึ่งจะจัดเตรียมฟังก์ชัน และ ข้อมูลสำหรับใช้งานตัวแปรเมทอดไว้ และเพื่อให้ระบบรู้จักคลาสตัวแปรเมทอดที่ผู้ใช้สร้างขึ้น คลาสนั้นจะต้องถูกลงทะเบียนไว้ใน “MethodType Table” ส่วนการนิยามตัวแปรเมทอด ผู้ใช้สามารถนิยามไว้ในไฟล์นิยามตัวแปรเมทอดตามรูปแบบในบทที่ 4. และใช้เครื่องมือสร้างตัวแปรเมทอด(MethodVar Builder) เพื่อสร้างเป็นคลาสตัวแปรเมทอด

5.1.2 แผนภาพคลาส (Class Diagram)

การทำงานของตัวแปรเมทอดแบ่งเป็น 2 ส่วน ได้แก่

1. ส่วนจัดการข้อมูล เป็นศูนย์กลางสำหรับจัดเตรียมข้อมูลและฟังก์ชันการทำงานเพื่อจัดการข้อมูลคลาสตัวแปรเมทอด และ สนับสนุนการทำงานของส่วนที่ 2
2. ส่วนกลไกตัวแปรเมทอด จะเป็นส่วนการทำงานของตัวแปรเมทอดแต่ละตัว



รูปที่ 5.2 แผนภาพคลาสของตัวแปรเมทอดส่วนจัดการข้อมูล

5.1.2.1 <u>ส่วนจัดการข้อมูล</u> สำหรับการลงทะเบียนคลาสตัวแปรเมทอด(regisType) จัดการข้อมูลคลาสตัวแปรเมทอด ตรวจสอบชนิดของเมทอด แสดงข้อผิดพลาดของการทำงาน (Debug) ประกอบด้วยคลาสต่อไปนี้

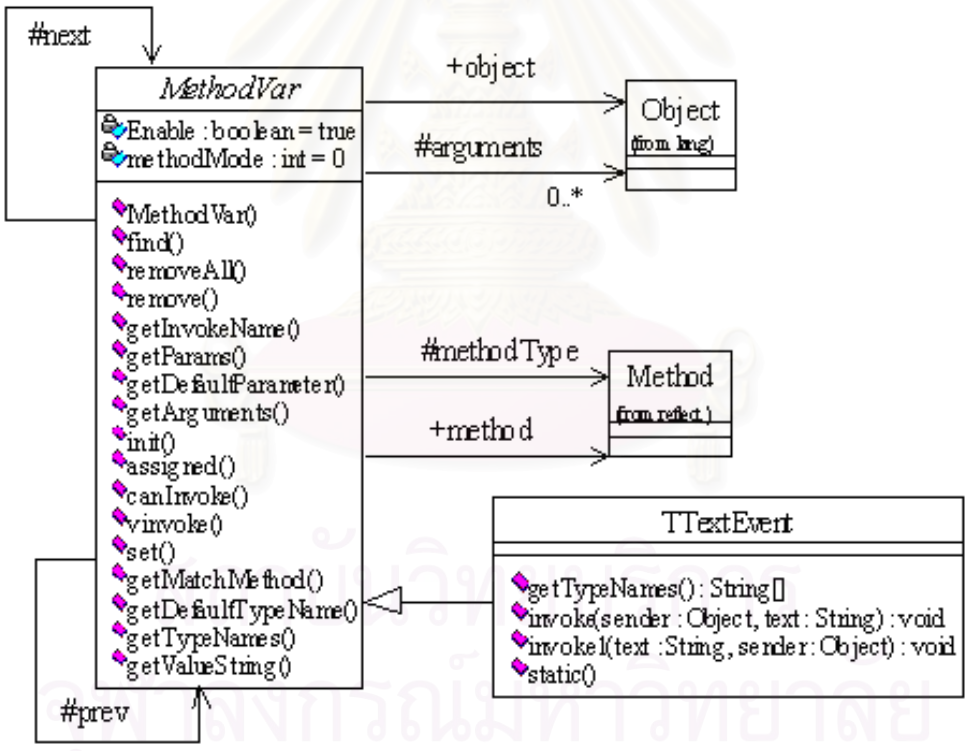
1.) MethodVar การทำงานส่วนนี้จะเป็นแบบ Static

- static public void **regisType**(MethodVar aMethodVar)
  สำหรับลงทะเบียนคลาสตัวแปรเมทอด คลาสที่จะใช้งานได้จะต้องลงทะเบียนก่อน ข้อมูลตัวแปรเมทอดได้แก่ ชื่อคลาส และ รูปแบบพารามิเตอร์ จะสร้างเป็นออบเจคต์ของคลาส MethodTypeData แล้วเก็บไว้ใน methodTypeTable
- static public Method **getMatchMethodType**(MethodVar aMethodVar, Method withMethod)จะคืนค่าชนิดของตัวแปรเมทอด (aMethodVar) ที่สามารถเข้ากับเมทอด (withMethod) ที่กำหนดให้ได้
- static public void **saveMethodVar** (Container root, String filename)
  static public void **loadMethodVar** (Container root,String filename)
  สำหรับเก็บข้อมูล และอ่านข้อมูล การเชื่อมต่อของตัวแปรเมทอดใน ออบเจคต์ root ในไฟล์

- static public void **debugMsg**(String s,int level)ใช้สำหรับแสดงข้อมูลการดีบัก (Debug) เมื่อเรียกใช้เมทอดนี้จะเกิดเหตุการณ์ onDebugMsg มีชนิดเป็น StringEvent ซึ่งหากต้องการให้ debugMsg ส่งข้อความไปที่เมทอดใด ก็สามารถกำหนดให้ onDebugMsg ชี้ไปที่เมทอดนั้นได้

2.) MethodTypeData ใช้เก็บข้อมูลชนิดของตัวแปรเมทอดจะมีข้อมูลที่อ้างถึงไปยังคลาสตัวแปรเมทอด (typeClass) และ ข้อมูลชนิดของเมทอด methodTypes ซึ่งถ้าเป็นตัวแปรเมทอดแบบหลายรูปคลาสตัวแปรเมทอดสามารถมีหลายชนิดได้

- public Method getMatchType (Method m,int typeMode)
  ใช้ตรวจสอบว่ามีเมทอดที่กำหนด เข้ากับตัวแปรเมทอดนั้นได้หรือไม่ และจะคืน Method ซึ่งเป็นชนิดเมทอดที่เข้ากันได้



รูปที่ 5.3 แผนภาพคลาสของตัวแปรเมทอดส่วนกลไกตัวแปรเมทอด

5.1.2.2 ส่วนกลไกตัวแปรเมทอด เป็นส่วนข้อมูลและการทำงานของตัวแปรเมทอดแต่ละตัว โดยจะอยู่ในคลาสเดียวกับส่วนที่ 1 แต่ส่วนที่ 1 จะเป็นเมทอดแบบ สถิต (Static) ซึ่งเป็นคลาสเมทอด ส่วนที่ 2 จะทำงานที่ออบเจคต์ หน้าที่ของส่วนนี้ได้แก่ เก็บข้อ

มูลอ้างถึงเมทอดปลายทาง มีฟังก์ชันสำหรับกำหนดคุณสมบัติ การเรียกใช้เมทอดที่อ้างถึง การทำงานหลักจะอยู่ในคลาส MethodVar และคลาสตัวแปรเมทอดจะต้องสืบทอดมาจากคลาสนี้ ข้อมูล *object, method* ใช้อ้างถึงเมทอดปลายทาง ซึ่งถูกกำหนดมาจากการเรียกใช้เมทอด **set**(object,"methodname")

arguments เป็นอะเรย์ของ Object ใช้เพื่อสำหรับเรียกเมทอดปลายทาง

methodType เป็นรูปแบบเมทอดที่ใช้ติดต่อกับเมทอดเป้าหมาย และ

next,prev เป็นข้อมูลที่ใช้สำหรับเมทอดแบบหลายจำนวน เพื่อสร้างเป็นลิสต์ของตัวแปรเมทอด

ในคลาส MethodVar จะมีเมทอดที่สำคัญดังต่อไปนี้

- public MethodVar find(Object obj,String methodname) สำหรับตัวแปรเมทอดแบบหลายจำนวนใช้ค้นหาว่าในลิสต์ มีเมทอดของออบเจคต์ที่ระบุ เก็บไว้หรือไม่ obj,methodname คือ ออบเจคต์และชื่อเมทอดที่ต้องการสอบถาม และ จะคืนค่าผลลัพธ์เป็นตัวแปรเมทอดที่เก็บเมทอดของออบเจคต์ถ้ามีในลิสต์ และ คืนค่าเป็น NULL ถ้าไม่พบ
- public void removeAll() สำหรับลบข้อมูลเมทอดและออบเจคต์ทั้งหมดที่เก็บในตัวแปรเมทอด
- public void remove()

  public void remove(Object obj, String methodname)

  สำหรับลบข้อมูลเมทอดและออบเจคต์ตามที่ระบุใน obj และ methodname ถ้าไม่ระบุจะลบเฉพาะข้อมูลที่เก็บในตัวปัจจุบัน
- public String getInvokeName() สำหรับใช้อ่านชื่อของเมทอด Invoke ที่กำลังใช้เชื่อมต่อกับเมทอดปลายทาง เมื่อตัวแปรเมทอดเป็นแบบหลายรูป ซึ่งสามารถมีเมทอด Invoke ที่ใช้กำหนดรูปแบบพารามิเตอร์อยู่หลายเมทอด ดังนั้นฟังก์ชันนี้จะช่วยบอกให้ทราบว่า เมทอด Invoke ใดที่มีพารามิเตอร์ตรงกับเมทอดปลายทาง
- public Class[] getParams() จะคืนค่าชนิดของพารามิเตอร์ของเมทอดที่เก็บในตัวแปรเมทอด โดยผลลัพธ์เป็นอะเรย์ของ Class
- public Object[] getArguments() สำหรับใช้สร้างอะเรย์ของออบเจคต์เพื่อใช้ส่งพารามิเตอร์ไปยังเมทอดปลายทาง ดูตัวอย่างการใช้ในขั้นตอนการเรียกใช้ เมทอดในหัวข้อ 5.2.3

- public void init() เป็นเมทอดที่สร้างไว้ใช้สำหรับ คลาสตัวแปรเมทอดของผู้ใช้ให้สามารถเขียนคำสั่งที่ต้องการให้ถูกเรียกเมื่อตัวแปรเมทอดเริ่มทำงาน
- public boolean assigned() ใช้ตรวจสอบว่าตัวแปรเมทอดถูกกำหนดค่าแล้วหรือไม่ จะคืนค่าผลลัพธ์เป็น จริง(true) หรือ เท็จ (false)
- public boolan canInvoke() ใช้ตรวจสอบสถานะว่า ตัวแปรเมทอดพร้อมจะถูกเรียกให้ทำงานหรือยัง โดยดูจากว่ามีการกำหนดเมทอดปลายทางให้แล้วหรือไม่ และ มีค่า Enable เป็นจริง
- public Object vinvoke(Object argv[]) เป็นเมทอดที่ใช้สำหรับให้ฟังก์ชัน invoke ในคลาสตัวแปรเมทอดของผู้ใช้ส่งพารามิเตอร์มาเพื่อเรียกใช้งานเมทอดปลายทาง หน้าทีของ vinvoke จะทำการตรวจสอบความถูกต้องของพารามิเตอร์ และความพร้อมของตัวแปรเมทอด แล้วทำการส่งต่อพารามิเตอร์ไปยังเมทอดปลายทางต่อไป
- public boolean set(Object obj, String methodname) เป็นฟังก์ชันที่ใช้สำหรับกำหนดค่าเมทอดปลายทางให้กับตัวแปรเมทอด โดยส่งออบเจคต์ (obj) และ ชื่อของเมทอด (methodname) ฟังก์ชันนี้จะทำการตรวจสอบข้อมูลของเมทอดที่ส่งเข้ามาว่าถูกต้องและมีพารามิเตอร์ที่เข้ากับรูปแบบพารามิเตอร์ของตัวแปรเมทอดหรือไม่ ถ้าเข้ากันได้จะคืนค่าเป็น true ถ้าไม่ได้คืนค่าเป็น false
- public Method[] getMatchMethod(Object fromObj) ใช้สำหรับตรวจสอบว่ามีเมทอดใดบ้างในออบเจคต์ fromObj ที่สามารถเชื่อมต่อกับตัวแปรเมทอด ซึ่งจะคืนค่าผลลัพธ์เป็นอะเรย์ของเมทอดทั้งหมดใน fromObj ที่มีพารามิเตอร์ที่เข้ากับตัวแปรเมทอดได้
- public String getDefaultTypeName() ใช้ในการกำหนดชื่อรูปแบบพารามิเตอร์หลัก ซึ่งปกติจะชื่อว่า “invoke”
- public String[] getTypeNames() ใช้สำหรับกำหนดชื่อของรูปแบบพารามิเตอร์สำรองอื่นๆ ในกรณีที่วแปรเมทอดมีพารามิเตอร์หลายรูปแบบ

5.2 การทำงานของตัวแปรเมทอดแบบรูปเดียว

ส่วนนี้จะเป็นการแสดงรายละเอียดการทำงานของตัวแปรเมทอดแบบพื้นฐานที่ไม่ซับซ้อน คือมีรูปแบบพารามิเตอร์เดียว และเป็นตัวแปรเมทอดแบบจำนวนเดียว ที่ทำงานกับออบเจคต์บนหน่วยความจำ ในหัวข้อต่อไปจึงจะแสดงการทำงานของตัวแปรเมทอดที่ซับซ้อนมากยิ่งขึ้น

5.2.1 การสร้างและเก็บข้อมูลคลาสตัวแปรเมทอด

คลาสตัวแปรเมทอดจะถูกลงทะเบียนไว้ใน ตารางข้อมูลชนิดเมทอด (MethodTypeTable) จะมีการทำงานดังส่วนของโปรแกรมต่อไปนี้

```
public  class TTextEvent extends MethodVar {
  // Regis Invoke method
  static {
     MethodVar.regisType(new TTextEvent());
    }
```

คำสั่งการลงทะเบียนคลาสจะถูกเขียนไว้ใน เมทอด static ของแต่ละคลาส ซึ่งจะถูกเรียกให้ทำงานโดยอัตโนมัติเมื่อคลาสถูกโหลดเข้ามาในหน่วยความจำ หลังจากนั้นข้อมูลของคลาสจะถูกเก็บไว้ใน methodTypeTable โดยใช้ชื่อคลาสเป็นคีย์ ดังส่วนของโปรแกรมจาวาต่อไปนี้

```
static public void regisType(MethodVar aMethodVar){
  String key=aMethodVar.getClass().getName();
  if (methodTypeTable.get(key)==null)
     methodTypeTable.put(key,new MethodTypeData(aMethodVar));
  else
    System.out.println(" Method type aready registered.");
  }
```

ซึ่งข้อมูลดังกล่าวจะถูกใช้เมื่อมีการสร้างตัวแปรเมทอด กลไกในตัวแปรเมทอดจะทำการอ่านข้อมูลคลาสเมทอดของตนจาก methodTypeTable เพื่อนำไปใช้งานในขั้นต่อไป ดังส่วนโปรแกรมต่อไปนี้

```
    ...
public MethodVar (int typeMode, int  methodMode){
    fMethodTypeData = (MethodTypeData)
    methodTypeTable.get(getClass().getName());
    ...
```

5.2.2 การกำหนดค่าเมทอดและการตรวจสอบความถูกต้อง

เมื่อมีการกำหนดเมทอดให้กับตัวแปร ก็จะทำการตรวจสอบว่าเมทอดดังกล่าวสามารถเก็บในตัวแปรนี้ได้หรือไม่ การตรวจสอบชนิดเมทอดกำหนดไว้ในบทที่ 4 ซึ่งจะมีการทำงานดังส่วนของโปรแกรมต่อไปนี้

```
public boolean set(Object object, String methodName)
{
  Class paramTypes[] = fMethodType.getParameterTypes();
  Class returnType   = fMethodType.getReturnType();
  try{
   Method method = object.getClass().getMethod(methodName,paramTypes);
   if((returnType == Void.TYPE )||
      (returnType.isAssignableFrom(method.getReturnType()))){
           fMethod = method;
           fObject = object;
```

```
            return true;
      }
     else
       debugMsg("Incorrect Return Type. Can't set this method.",0);
     } catch (Exception e){
       debugMsg("Incorrect Params Type. Can't set this method.",0);
     }
  return false;
}
```

การเรียกใช้เมทอด set จะส่งออบเจคต์และชื่อเมทอด ถ้าข้อมูลถูกต้อง เมทอดที่ส่งมาเข้ากับรูปแบบพารามิเตอร์ และ การคืนผลลัพธ์ ตามที่กำหนดไว้ได้ ก็จะคือค่า จริง(true) และจะเก็บข้อมูลอ้างถึงเมทอดใน fMethod และ fObject แต่ถ้าเก็บไม่ได้จะคืนค่า เท็จ (false)

5.2.3 การเรียกใช้เมทอดที่อ้างถึงในตัวแปร

ทำได้โดยเรียกใช้เมทอด invoke(…) ของตัวแปรเมทอดนั้น (อาจจะเป็นชื่ออื่นก็ได้ ขึ้นอยู่กับการนิยาม) เช่น m.invoke(target,"Hello") จากนั้นเมทอด invoke จะต้องทำการส่งต่อพารามิเตอร์ไปที่เมทอดเป้าหมาย โดยการรวมพารามิเตอร์ทั้งหมดเป็นอะเรย์ แล้วส่งต่อไปที่ method vinvoke(..) ซึ่งจะเรียกต่อไปที่เมทอดเป้าหมายต่อไป ดังตัวอย่างโปรแกรม

```
public void invoke(Object sender,String text){
     Object arg[] = getArguments();
     arg[0] = sender;
     arg[1] = text;
     super.vinvoke(arg);
}
```

vinvoke เป็นเมทอดที่ใช้ในการส่งพารามิเตอร์ต่อไปยังเมทอดที่ตัวแปรอ้างถึง ซึ่งใน vinvoke จะมีการตรวจสอบข้อมูลและสถานะของตัวแปรเมทอดเพื่อป้องกันข้อผิดพลาด เช่น การเรียกตัวแปรเมทอดขณะที่ ยังไม่มีการกำหนดค่าให้ หรือ เมทอดมีการ setEnable (false) ไว้ซึ่งตัวแปรเมทอดจะไม่พร้อมที่จะให้เรียก แต่ผู้ใช้ก็สามารถตรวจสอบสถานะของตัวแปรเมทอดได้จากฟังก์ชัน

public boolean canInvoke()

จะคืนจริง(true) ถ้าตัวแปรเมทอดนั้นพร้อมถูกเรียก และคืนค่าเท็จ(false) ถ้ายังพร้อมถูกเรียกซึ่งฟังก์ชัน vinvoke จะมีการเขียนดังตัวอย่างโปรแกรมต่อไปนี้

```
public Object vinvoke (Object args[])
 {
  Object result = null;
  if ((args == null)&&(arguments == null)) return null;
  if (invokeDelay < 0) {
     setDefaultParameter(args);
     return null;
```

```
    }
    if( ! canInvoke()) return null;

    try {
       int n = fMethod.getParameterTypes().length;
       Object xArg[] = new Object[n];
       if (args == null)
          for(int i = 0;i < n;i++) xArg[i] = arguments[i];
       else
          for(int i = 0;i < n;i++) {
             xArg[i] = args[i];
        debugMsg("Argument :"+i+" = "+args[i], DEBUG_MULTIPLE);
       }

    if (invokeDelay > 0 || methodMode == MULTIPLE_THREAD) {
        MethodThread mt = new
                   MethodThread(fObject, fMethod, xArg, invokeDelay);
        mt.start();
    } else {
        result = fMethod.invoke(fObject , xArg);
    }
    } catch (Exception e) {
     debugMsg("Exception can not invoke method "+fObject+"."+fMethod, 0);
     e.printStackTrace(debugStream);
    }
    return result;
}
```

คำสั่ง result = fMethod.invoke(fObject ,xArg) เป็นคำสั่งที่ใช้เรียกเมทอดที่ถูกอ้างถึงให้ทำงาน โดย fMethod จะอ้างถึงเมทอด และ fObject จะอ้างถึงออบเจคต์ที่เจ้าของเมทอดนั้น ส่วน xArg เป็นอากิวเมนต์ที่จะถูกส่งไป ซึ่งพารามิเตอร์ทั้งหมดจะถูกเก็บไว้ใน xArg ซึ่งเป็นอะเรย์ของออบเจคต์ เมื่อเมทอดทำงานเสร็จแล้วจะคืนผลลัพท์กลับมาที่ result

5.3. การทำงานตัวแปรเมทอดแบบหลายรูป

ส่วนนี้จะเป็นการแสดงรายละเอียดการทำงานของตัวแปรเมทอดที่ซับซ้อนขึ้น คือมีรูปแบบพารามิเตอร์มากกว่า 1 รูปแบบ แต่ยังเป็นตัวแปรเมทอดแบบจำนวนเดียวที่ทำงานกับออบเจคต์บนหน่วยความจำ

5.3.1 การสร้างและเก็บข้อมูลคลาสตัวแปรเมทอด

สำหรับตัวแปรเมทอดแบบหลายรูปจะมีรูปแบบพารามิเตอร์ได้มากกว่า 1 แบบดังนั้นในคลาสตัวแปรเมทอดจะต้องบอกให้ระบบทราบว่ามีเมทอดใดบ้างที่ใช้นิยามรูปแบบพารามิเตอร์โดยเขียนไว้ในเมทอด getTypeNames ดังตัวอย่าง

```
public String[] getTypeNames(){
   String tname[] = new String[2];
   tname[0]="invoke1";
   tname[1]="invoke2";
   return tname;
```

```
}
```

ซึ่งข้อมูลดังกล่าวจะถูกใช้ในการลงทะเบียนข้อมูลรูปแบบพารามิเตอร์ ดังส่วนของโปรแกรมต่อไปนี้

```
String typeNames[] = aMethodVar.getTypeNames();
Method ml[] = typeClass.getMethods();
for(int i=0;i<ml.length;i++){
  for(int j=0;j<typeNames.length;j++){
    if(ml[i].getName().compareTo(typeNames[j])==0){
     if(methodTypes.indexOf(ml[i])<0)
        methodTypes.addElement(ml[i]);
    }
 }
}
```

การทำงานจะอ่านรายชื่อเมทอดที่ใช้นิยามรูปแบบในฟังก์ชัน getTypeName แล้วทำการอ่านข้อมูลเมทอดทั้งหมดของคลาสตัวแปรเมทอดที่มีชื่ออยู่ในรายชื่อ เก็บลงในลิสต์ methodTypes

5.3.2 การกำหนดค่าเมทอดและการตรวจสอบความถูกต้อง

การตรวจสอบความถูกต้องของเมทอดแบบหลายรูปจะเพิ่มเติมจากเมทอดแบบรูปเดียว โดยจะต้องตรวจสอบรูปแบบมากกว่า 1 แบบ ตามที่ลงทะเบียนไว้ใน 3.1 ซึ่งมีการทำงานดังส่วนของโปรแกรมต่อไปนี้

```
public Method getMatchType(Method m,int typeMode){
 // Type Checking for defaultType.
 Class fParamTypes[];
 Class fReturnType;
 Method mth;

 for(int i=0;i<methodTypes.size();i++){
   mth = (Method)methodTypes.elementAt(i);
   fParamTypes = mth.getParameterTypes();
   fReturnType = mth.getReturnType();
   if(isMatchMethod(fParamTypes,fReturnType,m,typeMode))
     return mth;
 }
 return null;
}
```

ในฟังก์ชันนี้จะคืนค่ารูปแบบพารามิเตอร์ของตัวแปรเมทอดที่สอดคล้องกับ m โดย typeMode คือโหมดที่ใช้ในการตรวจสอบชนิดเป็นแบบ สมบูรณ์ (FULL) หรือ บางส่วน (PARTIAL) ซึ่งจะต่างกันที่จำนวนพารามิเตอร์ที่ใช้ตรวจสอบ

5.3.3 การเรียกใช้เมทอดผ่านตัวแปร

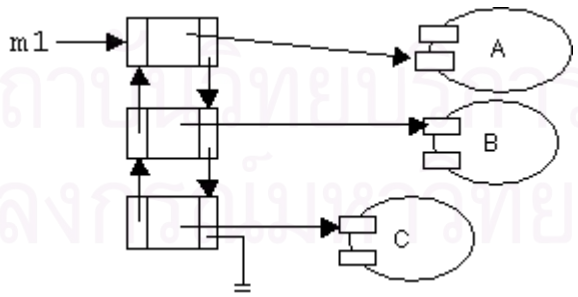
ผู้ใช้จะต้องเรียกใช้เมทอดผ่านทาง รูปแบบพารามิเตอร์หลักซึ่งปกติคือ **invoke** จากนั้นตัวแปรเมทอดจะเลือกรูปแบบที่ตรงกับเมทอดที่อ้างถึง ตามส่วนของโปรแกรมต่อไปนี้

```
public void invoke(Object sender,String text){
  if(getInvokeName().compareTo("invoke1")==0){
     invoke1(text,sender);
  }
  else{
     Object arg[] = getArguments();
     arg[0] = sender;
     arg[1] = text;
     super.vinvoke(arg);
  }
}
public void invoke1(String text,Object sender){
     Object arg[] = getArguments();
     arg[0] = text;
     arg[1] = sender;
     super.vinvoke(arg);
}
```

ตัวแปรเมทอดสามารถตรวจสอบรูปแบบที่กำลังใช้ติดต่อกับเมทอดเป้าหมายจากเมทอด **getInvokeName** เพื่อสร้างเงื่อนไขให้ส่งพารามิเตอร์ต่อไปได้อย่างถูกต้อง รายละเอียดการส่งต่อพารามิเตอร์ไปยังเมทอดปลายทางอธิบายไว้ในหัวข้อ 5.2.3

5.4 การทำงานของตัวแปรเมทอดแบบหลายจำนวน

ส่วนนี้จะเป็นการแสดงรายละเอียดการทำงานของตัวแปรเมทอดที่ซับซ้อนขึ้น คือสามารถเก็บเมทอดของออบเจคต์ได้มากกว่า 1 เมทอด ในงานวิจัยนี้จะใช้วิธีการจัดการแบบ *LinkList* ดังรูปที่ 5-4



รูปที่ 5.4 การทำงานของเมทอดแบบหลายจำนวน

จากรูปการเรียก m1.invoke(…) จะทำให้เมทอดของ A,B และ C ถูกเรียก ตัวแปรเมทอดแบบหลายจำนวนสามารถใช้ร่วมกับ ตัวแปรเมทอดแบบรูปเดียว หรือ หลายรูปก็ได้

5.4.1 การกำหนดค่าเมทอดและการตรวจสอบความถูกต้อง

ตัวแปรเมทอดที่จะเป็นแบบหลายจำนวนจะต้องกำหนดให้

methodMode = MULTIPLE หรือ MULTIPLE_THREAD ซึ่งทั้ง 2 แบบ จะแตกต่างในวิธีการเรียกใช้เมทอด การกำหนดค่าเมทอดและตรวจสอบเมทอดทำได้โดยสร้าง ตัวแปรเมทอดใหม่เพิ่มในท้ายของลิสต์ ดังส่วนของโปรแกรมต่อไปนี้

```
if (methodMode!=SINGLE && assigned()){
  if(next!=null)
    return next.set(object,methodName);
  else {
    try {
      MethodVar newMV = (MethodVar)(getClass().newInstance());
      newMV.methodMode = methodMode;
      next = newMV;
      newMV.prev = this;
      return newMV.set(object,methodName);
    } catch (Exception ex){
      return false;
    }
}
```

5.4.2 การเรียกใช้เมทอดผ่านตัวแปร

การเรียกเมทอดที่อ้างถึงในลิสต์ สามารถเรียกได้ 2 แบบคือ

1. <u>การเรียกใช้เมทอดแบบลำดับ</u>

   methodMode = MULTIPLE โดยเรียกให้เมทอดตัวแรกทำงานเสร็จก่อน แล้วค่อยเรียกตัวต่อไป ที่ละตัวตามลำดับ

2. <u>การเรียกใช้เมทอดแบบอิสระ</u>

   methodMode = MULTIPLE_THREAD โดยเรียกให้เมทอดตัวแรกทำงานแต่ไม่รอให้เสร็จ ก็เรียกตัวต่อไปทันทีตามลำดับ โดยไม่ต้องรอให้ตัวก่อนหน้าทำงานเสร็จก่อน วิธีนี้จะใช้ความสามารถของ Thread ในจาวาช่วยทำให้เมทอดทั้งหมดทำงานเป็นอิสระกันได้

รายละเอียดการส่งต่อพารามิเตอร์ไปยังเมทอดปลายทางอธิบายไว้ในหัวข้อ 5.2.3

5.5 การเก็บ และ อ่านข้อมูลตัวแปรเมทอด ในที่เก็บข้อมูลถาวร

ในงานวิจัยนี้จะเก็บข้อมูลของตัวแปรเมทอดที่ใช้สร้างเหตุการณ์ในคอมโพเนนท์ โดยข้อมูลตัวแปรเมทอดที่สามารถเก็บได้นั้นจะต้องอ้างถึงออบเจคต์ที่มีการกำหนดชื่อแล้วเท่านั้น เพื่อที่จะสามารถอ้างถึงออบเจคต์ได้ถูกต้อง โดยข้อมูลที่ใช้อ้างถึงเมทอดจะประกอบด้วย ชื่อ ออบเจคต์ และ ชื่อเมทอด

5.5.1 รูปแบบไฟล์ มีตัวอย่างดังต่อไปนี้

```
#Method Variable Data File
[startButton]
  OnAction=stopButton.enable
  OnAction=startButton.disable
  OnAction=timer1.enable
[stopButton]
  OnAction=stopButton.disable
  OnAction=startButton.enable
  OnAction=timer1.disable
[timer1]
  OnAction=.stepImage
```

รูปแบบของไฟล์จะเหมือนกับไฟล์ INI ในวินโดวส์ ข้อความหลัง # เป็นหมายเหตุ ข้อมูลเหตุการณ์ของคอมโพเนนท์แต่ละตัวแยกกันด้วยชื่อคอมโพเนนท์ในเครื่องหมาย [ ] จากตัวอย่าง OnAction เหตุการณ์ของ startButton เป็นตัวแปรเมทอดแบบหลายจำนวซึ่งจะอ้างถึงเมทอดพร้อมกัน 3 เมทอด

5.5.2 การเก็บข้อมูลตัวแปรเมทอด

สามารถทำได้โดยเรียกใช้เมทอด MethodVar.saveMethodVar(…) ดังตัวอย่าง

MethodVar.saveMethodVar(applet1, “c:\applet1.event”);

ข้อมูลเหตุการณ์ทั้งหมดที่อยู่ใน applet1 จะถูกเขียนลงในไฟล์ที่ระบุ

5.5.3 การอ่านข้อมูลตัวแปรเมทอด

สามารถทำได้โดยเรียกใช้เมทอด MethodVar.loadMethodVar(…) ดังตัวอย่าง

MethodVar.loadMethodVar(applet1, “c:\applet1.event”);

ข้อมูลเหตุการณ์ทั้งหมดในไฟล์ที่ระบุจะถูกนำมาสร้างการเชื่อมต่อเหตุการณ์ใน applet1

ดังนั้นจะพบว่าความสามารถของการอ่านข้อมูลตัวแปรเมทอดจากไฟล์มาสร้างการเชื่อมต่อเหตุการณ์ในขณะรันโปรแกรม แสดงให้เห็นว่าตัวแปรเมทอดสามารถใช้ในการเชื่อมต่อคอมโพเนนท์ระหว่างรันได้เพราะการเชื่อมต่อถูกสร้างขึ้นหลังจากการอ่านข้อมูลจากไฟล์ รูปแบบไฟล์ข้อมูลเป็นไฟล์ตัวอักษรที่สามารถแก้ไขได้ง่าย และสามารถแก้ไขไฟล์ข้อมูลแล้วอ่านเข้ามาใหม่ได้โดยไม่ต้องปิดโปรแกรม และไม่ต้องคอมไพล์โปรแกรมใหม่ เราอาจจะแยกข้อมูลการเชื่อมต่อคอมโพ

เนนท์ไว้ในไฟล์ข้อมูล โดยไม่ต้องเขียนไว้ในโปรแกรมก็ได้ ทำให้สามารถแก้ไขการทำงานของโปรแกรมได้โดยไม่ต้องคอมไพล์ใหม่ ซึ่งสามารถดูตัวอย่างการทดสอบในบทที่ 6

5.6 ตัวแปรเมทอดที่ใช้กับออบเจคต์ในฐานข้อมูลเชิงวัตถุ (OODBMS)

ฐานข้อมูลเชิงวัตถุที่ใช้ในงานวิจัยนี้คือ Object Store: Personal Storage Edition (PSE)/PSE Pro Release 6.0 for Java การสร้างออบเจคต์ที่จะสามารถเก็บในฐานข้อมูล จะต้องเป็นออบเจคต์ของคลาสถาวร (Persistence-Capable) ซึ่งการทำให้คลาสมีความสามารถเป็นออบเจคต์ถาวรได้ จะต้องใช้เครื่องมือ Class File Postprocesser (CFP) เพื่อแปลงคลาสนั้นก่อน ตัวอย่างคำสั่ง

```
c:\>osjcfp -dest . -inplace Person.class
```

จะทำให้คลาส Person เป็นคลาสถาวร ดังนั้นตัวแปรเมทอดที่จะสามารถเก็บในฐานข้อมูลได้จะต้องถูกแปลงด้วย CFP เช่นกัน และเมทอดที่จะเก็บในตัวแปรได้จะต้องอ้างไปยัง ออบเจคต์แบบถาวรเช่นเดียวกัน

5.6.1 การสร้างและเก็บข้อมูลคลาสตัวแปรเมทอด

ตัวแปรเมทอดสำหรับ Object Store จะถูกสร้างแยกเป็นพิเศษจากตัวแปรเมทอดปกติ แต่มีกลไกการทำงานที่คล้ายกัน เพียงแต่การเก็บข้อมูลของเมทอดเป้าหมายจะต้องมีการจัดการพิเศษสำหรับออบเจคต์แบบถาวรด้วย ซึ่งคลาสตัวแปรเมทอดจะต้องสืบทอดมาจากคลาส com.wtv.odi.OSMethodVar การนิยามทำเช่นเดียวกับตัวแปรเมทอดปกติ แต่การคอมไพล์จะต้องใช้โปรแกรม OSMVBuilder ดังตัวอย่าง

```
>java  com.wtv.util.OSMVBuilder –c method.def
```

โปรแกรมจะทำการสร้างไฟล์ .java และ คอมไพล์เป็น .class แล้วเรียก CFP ทำการแปลงคลาสให้เป็นคลาสแบบถาวร ซึ่งพร้อมที่จะทำงานกับ Object Store ได้ทันที คลาสตัวแปรเมทอดจะต้องลงทะเบียนใน com.wtv.odi.MethodVar ด้วยคำสั่ง regisType(..)

```
com.wtv.odi.MethodVar.regisType(new PIntEvent());
```

โดยคำสั่งนี้จะเขียนไว้ในเมทอด static ของคลาสตัวแปรเมทอด

5.6.2 การกำหนดค่าเมทอดและการตรวจสอบความถูกต้อง

การทำงานในส่วนนี้จะเหมือนกับตัวแปรเมทอดปกติ แต่จะเพิ่มส่วนของการตรวจสอบชนิดของออบเจคต์ว่าเป็น ออบเจคต์ถาวรหรือไม่ ดังนี้

```
if (object != null && object instanceof IPersistent) {
     ptargetObject = (IPersistent)object;
     targetMethodName = methodName;
}
```

ถ้าเป็นออบเจคต์ถาวรจะทำการเก็บข้อมูลเพิ่มเติมคือ

IPersistent ptargetObject;

String targetMethodName;

เพราะเมื่อตัวแปรเมทอดถูกเก็บลงฐานข้อมูล ptargetObject และ targetMethod-Name จะถูกเก็บไว้โดยอัตโนมัติ ถ้า ptargetObject อ้างถึงออบเจคต์ที่ไม่ใช่ออบเจคต์ถาวรก็จะเกิด ERROR เพราะ Object Store ไม่สามารถเก็บออบเจคต์นั้นได้ ดังนั้นจึงต้องตรวจสอบชนิดของออบเจคต์ก่อนการกำหนดค่าให้ ptargetObject

5.6.3 การเรียกใช้เมทอดผ่านตัวแปร

การเรียกใช้ตัวแปรเมทอดในออบเจคต์ถาวรจะต้องมีความระวังเป็นพิเศษเพราะว่าในบางครั้งออบเจคต์ที่ตัวแปรเมทอดอ้างถึงอาจจะยังไม่อยู่ในหน่วยความจำและไม่พร้อมทำงาน ดังนั้นจึงต้องตรวจสอบก่อนการเรียกใช้เมทอด

```
public boolean assigned (){
   if( ptargetObject != null &&
       ptargetObject != fObject &&
       targetMethodName != null) {
       set(ptargetObject,targetMethodName);
    }
    return (fMethodType != null)&&
           (fMethod != null ) &&
           (fObject != null);
}
```

จะเป็นการตรวจสอบว่าตัวแปรเมทอดพร้อมที่จะถูกเรียกหรือยัง เพราะเมื่อตัวแปรถูกเก็บในฐานข้อมูลจะมีเพียง ptargetObject และ targetMethodName เท่านั้นข้อมูลอื่นๆจะไม่ถูกเก็บ ดังนั้นก่อนการเรียกใช้งานจึงต้องกำหนดค่าให้กับข้อมูลอื่นๆก่อน

5.7 ตัวอย่างคลาสตัวแปรเมทอด

5.7.1 สำหรับออบเจคต์ทั่วไป

ส่วนแรกเป็นตัวอย่างการนิยามคลาสตัวแปรเมทอด ส่วนที่ 2 เป็นไฟล์คลาสตัวแปรเมทอดที่ถูกสร้างขึ้นตามที่ถูกนิยามได้โดยใช้เครื่องมือ MVBuilder ดังรูปที่ 5.5

```
// Method Varible Define File
package com.wtv.method
import java.awt.*
import java.awt.event.*
[TtextEvent]
  :void invoke
     Object sender
     String text
  :void invoke1
     String text      = text
     Object sender    = sender
```

```
FILE : TTextEvent.Java
/*******************************************
 Method Variable Class
 Class Name  : TTextEvent
 Generate by : Method Builder 1.0
 Wachirawut Tamviset
*******************************************/
package com.wtv.method;
import java.awt.*;
import java.awt.event.*;
public  class TTextEvent extends MethodVar {
  // Regis Invoke method
  static {
     MethodVar.regisType(new TTextEvent());
  }
  // override getTypeNames for regis mutiple type invoke method
  public String[] getTypeNames(){
     String tname[] = new String[1];
     tname[0]="invoke1";
     return tname;
  }
  public void invoke(Object sender,String text){
     if(getInvokeName().compareTo("invoke1")==0){
         invoke1(text,sender);
     }
     else{
       Object arg[] = getArguments();
       arg[0] = sender;
       arg[1] = text;
       super.vinvoke(arg);
     }
     if(getMethodMode()!=SINGLE && next != null){
        ((TTextEvent)next).invoke(sender,text);
     }
  }
  public void invoke1(String text,Object sender){
       Object arg[] = getArguments();
       arg[0] = text;
       arg[1] = sender;
       super.vinvoke(arg);
  }
} // end of TTextEvent
```

รูปที่ 5.5 ตัวอย่างคลาสตัวแปรเมทอดสำหรับออบเจคต์ทั่วไป

5.7.2 สำหรับออบเจคต์ถาวร

ส่วนแรกเป็นตัวอย่างการนิยามคลาสตัวแปรเมทอด ส่วนที่ 2 เป็นไฟล์คลาสตัวแปรเมทอดที่ถูกสร้างขึ้นตามที่ถูกนิยามได้โดยใช้เครื่องมือ OSMVBuilder เพื่อสร้างคลาสตัวแปรเมทอดสำหรับออบเจคต์ถาวร ดังรูปที่ 5.6

```
// Method Variable Define File for ObjectStore
package com.wtv.odi
[PintEvent]
  :void invoke
     Object sender
     int value
  :void invoke1
     int value = value
     Object sender = sender
```

```
FILE : PIntEvent.java
/*******************************************
 Method Variable Class for Object Store
 Class Name  : PIntEvent
 Generate by : Method Builder 1.0
 Wachirawut Tamviset
*******************************************/
package com.wtv.odi;
import com.odi.*;
public  class PIntEvent extends OSMethodVar {
  // Regis Invoke method
  static {
     OSMethodVar.regisType(new PIntEvent());
  }
  public PIntEvent(){super();}
  // override getTypeNames for regis mutiple type invoke method
  public String[] getTypeNames(){
     String tname[] = new String[1];
     tname[0]="invoke1";
     return tname;
  }
  public void invoke(Object sender,int value){
     if(getInvokeName().compareTo("invoke1")==0){
         invoke1(value,sender);
     }
     else{
       Object arg[] = getArguments();
       arg[0] = sender;
       arg[1] = new Integer(value);
       super.vinvoke(arg);
     }
     if(getMethodMode()!=SINGLE && next != null){
         ((PIntEvent)next).invoke(sender,value);
     }
  }

  public void invoke1(int value,Object sender){
       Object arg[] = getArguments();
       arg[0] = new Integer(value);
       arg[1] = sender;
       super.vinvoke(arg);
  }
} // end of PIntEvent
```

รูปที่ 5.6 ตัวอย่างคลาสตัวแปรเมทอดสำหรับออบเจคต์ถาวร

การสร้างตัวแปรเมทอดมีความซับซ้อนมากพอสมควร เพราะจะต้องสร้างคลาสให้ถูกต้องตามข้อกำหนด ดังแสดงตัวอย่างโปรแกรมในรูปที่ 5.5 และ รูปที่ 5.6 ในการเรียกเมทอดปลายทางผ่านทางคลาส Method จะต้องทำการรวมพารามิเตอร์ไว้ในอะเรย์ก่อน นอกจากนี้สำหรับเมทอดแบบหลายรูปที่มีรูปแบบพารามิเตอร์ จะมีความซับซ้อนยิ่งขึ้นเพราะต้องทำเลือกรูปแบบพารามิเตอร์สำหรับส่งต่อไปยังเมทอดปลายทางให้ถูกต้องแต่การใช้เครื่องมือสร้างคลาสตัวแปรเมทอดสามารถทำให้ขั้นตอนการสร้างตัวแปรเมทอดสะดวกขึ้น ซึ่งผู้ใช้ไม่จำเป็นต้องเข้าใจกลไกภายในของตัวแปรเมทอดก็สามารถสร้างคลาสตัวแปรเมทอดได้ด้วยตัวเอง

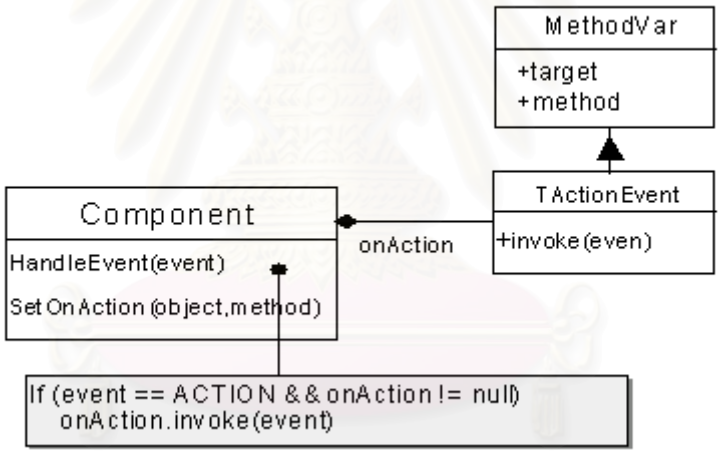
ในบทต่อไปจะแสดงการนำเทคนิคตัวแปรเมทอดไปใช้ในการสร้างโปรแกรม และ ทดสอบการใช้งานตัวแปรเมทอด

บทที่ 6

การนำไปใช้และการทดสอบตัวแปรเมทอด

ในบทนี้จะเป็นการทดสอบตัวแปรเมทอด โดยการสร้างโครงร่างพัฒนาโปรแกรม โดยใช้ตัวแปรเมทอดในสร้างกลไกเพื่อจัดการเหตุการณ์ของคอมโพเนนท์ กรณีศึกษาในงานวิจัยนี้จะเป็นการปรับปรุงคลาสไลบรารี่ AWT ของจาวามาตรฐานโดยเพิ่มการจัดการเหตุการณ์จากเดิมที่ใช้วิธีคลาสอะเดปเตอร์ ให้ใช้วิธีตัวแปรเมทอด และ ทำการทดลองสร้างโปรแกรมเพื่อเปรียบเทียบการเขียนโปรแกรมทั้ง 2 วิธี และทดสอบการใช้งานตัวแปรเมทอดในฐานข้อมูลเชิงวัตถุ

6.1 การใช้ตัวแปรเมทอดในการสร้างส่วนจัดการเหตุการณ์



รูปที่ 6.1 แบบจำลองการจัดการเหตุการณ์ด้วยตัวแปรเมทอด

แบบจำลองที่ใช้เป็นแบบจำลองดีเลเกชันเช่นเดียวกับวิธีการจัดการของจาวามาตรฐาน แต่เปลี่ยนจากการใช้คลาสอะเดปเตอร์ ในการสร้างตัวแทนเหตุการณ์มาใช้ตัวแปรเมทอด ในรูปที่ 6.1 คอมโพเนนท์สามารถเลือกใช้ตัวแปรเมทอดเพื่อสร้างเหตุการณ์ได้ เช่น onAction เมื่อ ฟังก์ชัน HandleEvent ของคอมโพเนนท์ถูกเรียกให้ทำงาน ก็จะตรวจสอบเหตุการณ์ที่เกิดขึ้น (event) และถ้า onAction มีการกำหนดเมทอดปลายทาง (not null) แล้ว onAction.invoke(..) จะถูกเรียกให้ทำงาน ซึ่งเมทอดปลายทางที่ตอบสนองต่อเหตุการณ์นั้นก็จะถูกเรียกให้ทำงาน

ตัวอย่าง : คอมโพเนนท์ Timer

ในตัวอย่างนี้จะเป็นการสร้างคอมโพเนนท์ Timer ซึ่งเป็นคอมโพเนนท์สำหรับจับเวลา Timer จะมีเหตุการณ์ onAction ซึ่งจะถูกเรียกทุกๆ รอบของเวลาที่กำหนดเช่น กำหนดให้ Timer มีรอบการจับเวลาเป็น 1 วินาทีจะทำให้เกิดเหตุการณ์ onAction ทุกๆ 1 วินาที การสร้าง Timer จะใช้ความสามารถของ Thread ในจาวา ดังโปรแกรมต่อไปนี้

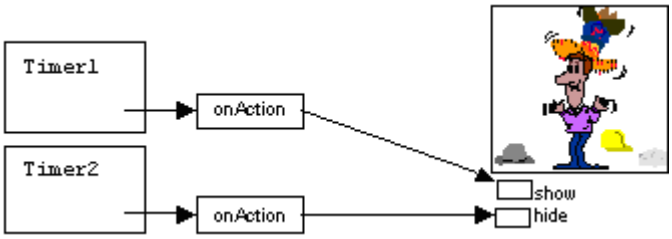
```
public  class Timer extends Thread {
private int interval=1000;
  private boolean flag=false;
  public TNotifyEvent onAction = new TNotifyEvent();
  public void run(){
   flag = true;
   while (flag){
    try{
       sleep(interval);
       if(onAction != null) onAction.invoke(this);
        }catch (Exception E) {
       E.printStackTrace(MethodVar.debugStream);
     }
   }
  }
  public int getInterval(){ return interval; }
  public void setInterval(int v){
    interval = v;
  }
  public stop(){ flag = false; }
}
```

ในเมทอด ‘run’ จะเป็นการวนลูปไปจนกว่า Timer จะถูกส่งให้ stop ในทุกๆรอบการทำงานจะมีการตั้งเวลาด้วยคำสั่ง sleep เมื่อถึงเวลาที่กำหนดในค่า interval แล้วและเหตุการณ์ onAction มีการกำหนดค่า onAction.invoke(..) จะถูกเรียกให้ทำงาน การทดลองใช้งาน Timer แสดงในส่วนของโปรแกรมต่อไปนี้

```
public class Test{
  ...
  public void start{
   Timer t = new Timer();
   t.setInterval(500);   // 500 milisecond
   t.onAction.set(this,"doTimer");
   t.start();
  }
  public void doTimer(){
    System.out.println("Do Timer");
  }
}
```

เมื่อรันโปรแกรม Timer จะทำงานแล้วทุกๆ ครึ่งวินาที (500 milisecond) จะพิมพ์คำว่า “Do Timer” จากตัวอย่างนี้ Timer สามารถนำไปใช้ในการสร้างโปรแกรมแบบมีการเคลื่อนไหว (Animation) ได้เช่น การทำภาพกระพริบ สามารถใช้ Timer 2 ตัว โดยตั้งเวลาห่างกันให้พอดี แล้ว

ให้ Timer1.onAction ชี้ไปที่ เมทอด show() ของภาพ Timer2.onAction ให้ชี้ที่ เมทอด hide() เมื่อ Timer ทั้ง 2 ทำงานภาพก็จะกระพริบ ดังรูปที่ 6.2



รูปที่ 6.2 ตัวอย่างโปรแกรมสร้างภาพกระพริบ

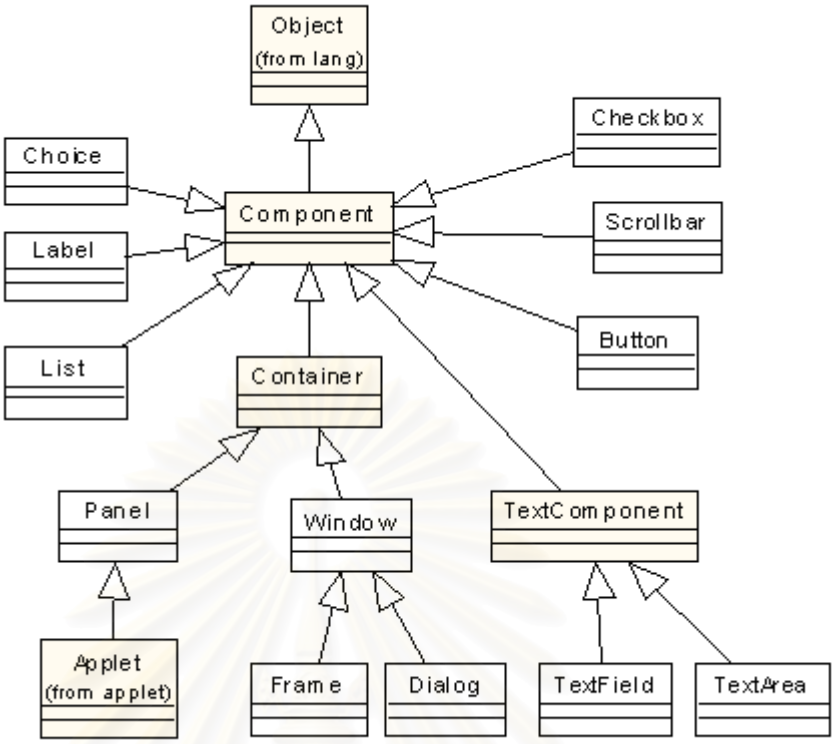
6.2 กรณีศึกษาการสร้างโครงร่างพัฒนาโปรแกรม

ในงานวิจัยนี้ได้ทำการทดลองสร้างคอมโพเนนท์โดยแนวคิดของตัวแปรเมทอดในการสร้างเหตุการณ์ โดยอาศัยต้นแบบจากคลาสไลบรารี AWT ของจาวา

6.2.1 เกี่ยวกับไลบรารี่ AWT (Abstract Window Toolkit)

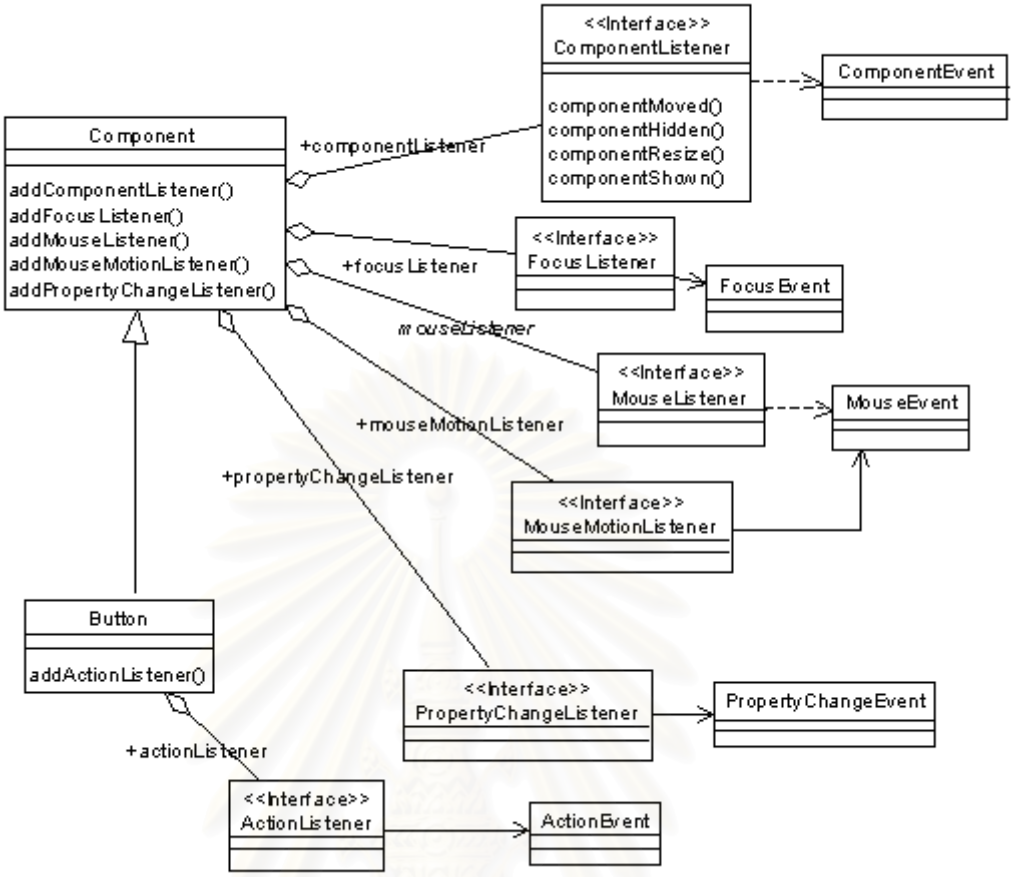
ไลบรารี่ AWT จะประกอบด้วยคอมโพเนนท์เกี่ยวกับการสร้างโปรแกรมติดต่อผู้ใช้แบบกราฟิกซึ่งจะประกอบด้วยคอมโพเนนท์ต่างๆเช่น ปุ่ม เมนู กรอบข้อความ และมีคำสั่งสำหรับการแสดงผลรูปภาพกราฟฟิกต่างๆ ด้วย การสร้างโปรแกรมจะเป็นการเขียนโปรแกรมเพื่อนำคอมโพเนนท์ต่างๆ มาประกอบกัน และ ควบคุมการทำงานของคอมโพเนนท์ด้วยการเขียนโปรแกรมควบคุมเหตุการณ์ที่เกิดกับคอมโพเนนท์เหล่านั้น ไลบรารี่ AWT ถือว่าเป็นไลบรารี่มาตรฐานสำหรับจาวา เครื่องมือพัฒนาโปรแกรมสำหรับภาษาจาวาทุกตัวจะสนับสนุนคอมโพเนนท์ใน AWT

ในไลบรารี่ AWT จะประกอบด้วยคลาสที่เป็นคอนโทรลต่างๆเช่น ปุ่ม กรอบข้อความ ซึ่งสืบทอดมากจากคลาส Component ส่วน คอมโพเนนท์ที่สามารถวางคอมโพเนนท์อื่นภายในได้ จะสืบทอดจากคลาส Container ดังรูปที่ 6.3



รูปที่ 6.3 แผนภาพคลาสคอมโพเนนท์ใน AWT

คลาส Component จะมีเหตุการณ์ที่เกี่ยวข้องจำนวนมาก เหตุการณ์ต่างๆจะถูกจัดกลุ่มรวมกันไว้ในอินเตอร์เฟสคลาส เช่น ComponentListener จะรวมเหตุการณ์ที่เกี่ยวกับการทำงานของคอมโพเนนท์ ได้แก่ componentMoved จะถูกเรียกเมื่อคอมโพเนนท์มีการย้ายตำแหน่ง และ componentHidden จะถูกเรียกเมื่อคอมโพเนนท์ถูกทำให้หายไป เป็นต้น ในรูปที่ 6.4 จะแสดงเหตุการณ์ต่างๆที่อยู่ใน Component

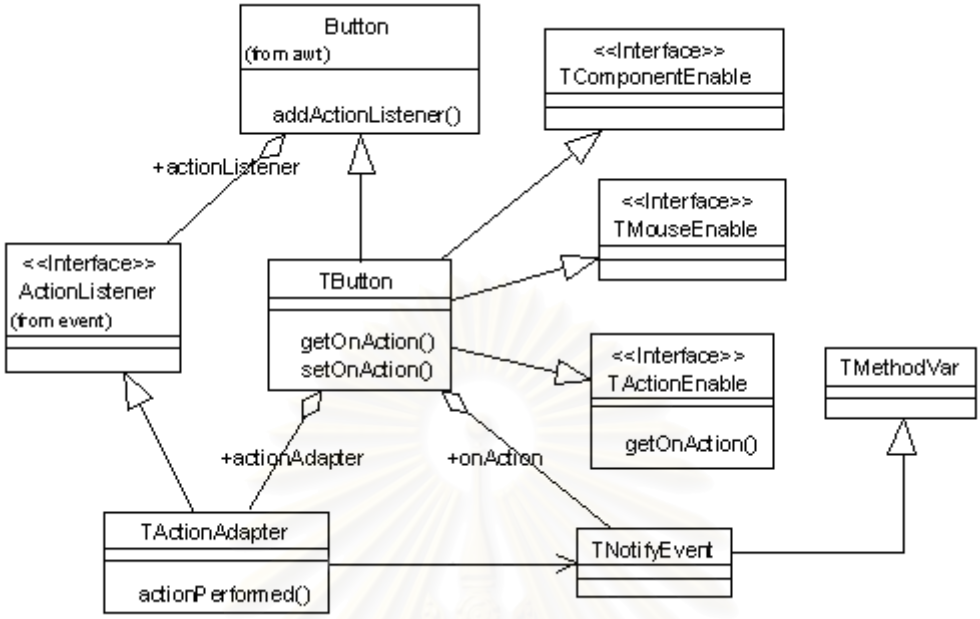


รูปที่ 6.4 แผนภาพคลาสแสดงแบบจำลองเหตุการณ์ใน AWT

คลาสที่สืบทอดจากคลาส Component ก็จะมีความสามารถของเหตุการณ์ใน Component ด้วย และสามารถเพิ่มเหตุการณ์ของตนเองได้ เช่น คลาส Button จะมี ActionListener ซึ่งจะถูกเรียกเมื่อ ผู้ใช้กดปุ่ม

การเขียนโปรแกรมควบคุมเหตุการณ์ของคอมโพเนนท์ ทำได้โดยการสร้างคลาสที่สืบทอดจากอินเตอร์เฟสคลาสของเหตุการณ์นั้นและทำการเขียนโปรแกรมในเมทอดของเหตุการณ์ที่ต้องการ แล้วเรียกใช้เมทอด add_Listener เพื่อให้คอมโพเนนท์ส่งเหตุการณ์มายังเมทอดที่เขียนขึ้นได้

6.2.2 วิธีการจัดการเหตุการณ์ด้วยตัวแปรเมทอดในคอมโพเนนท์ AWT



รูปที่ 6.5 แผนภาพคลาสแสดงเหตุการณ์ใน TButton

ในรูปที่ 6.5 คลาส TButton สืบทอดจากคลาส Button เพื่อสร้างการจัดการเหตุการณ์ใหม่ด้วยตัวแปรเมทอด ในตัวอย่างแสดงการสร้างเหตุการณ์ onAction ซึ่งจะเกิดเมื่อปุ่มถูกกด โดย onAction เป็นออบเจคต์ของคลาส TNotifyEvent ซึ่งเป็นตัวแปรเมทอด เนื่องจากการทำงานของ AWT เดิมใช้วิธีการของ คลาสอะเดปเตอร์ ดังนั้นจึงต้องสร้าง TActionAdapter เพื่อกำหนดให้ เหตุการณ์ปุ่มถูกกดส่งมาที่ เมทอด actionPerform และ actionPerform จะเรียกให้ onAction ทำงานต่อไป ดังส่วนข้อมูลโปรแกรม ต่อไปนี้

```
public class TButton extends Button
  implements TMouseEnable,TComponentEnable,
   TKeyEnable,TActionEnable {
public TButton (String label)  {
     super(label);
     new TComponentAdapter(this);
     new TMouseAdapter(this);
     new TKeyAdapter(this);
     new TActionAdapter(this);
  }
  ...
}
```

เมื่อออบเจคต์ของ TButton ถูกสร้างมันจะทำการสร้างออบเจคต์อะเดปเตอร์ สำหรับเหตุการณ์ต่างๆ เช่นคลาส TActionAdapter สำหรับใช้กับ ActionEvent หลังจากนั้นตัวออบเจคต์อะเดปเตอร์ ที่สร้างขึ้นจะทำการลงทะเบียนตัวเองเป็นผู้ฟังเหตุการณ์ของคอมโพเนนท์นั้น ซึ่งจะทำให้เมื่อ

เกิดเหตุการณ์กับคอมโพเนนท์นั้นเช่น ปุ่มถูกกดจะทำให้เมทอด actionPerformed ของออบเจคต์อะเดปเตอร์ จะถูกเรียกให้ทำงาน ดังส่วนของโปรแกรม

```
public class TActionAdapter implements ActionListener
{
  public TActionEnable target;

  public TActionAdapter(TActionEnable aTarget){
    target = aTarget;
    target.addActionListener(this);
  }
  public void actionPerformed(ActionEvent e){
    if (target.getOnAction()!=null)
       target.getOnAction().invoke(target);
  }
}
```

ในเมทอด actionPerformed จะทำการเรียก target.getOnAction().invoke ถ้ามีการกำหนดเมทอดเป้าหมายให้กับ onAction แล้วเมทอดเป้าหมายก็จะถูกเรียกให้ทำงาน

6.3 การทดสอบสร้างโปรแกรม

6.3.1 สภาวะแวดล้อมที่ใช้พัฒนาและทดสอบ

1.) การพัฒนาโปรแกรม

- เครื่องมือพัฒนาโปรแกรมภาษาจาวา ที่รองรับมาตรฐานเวอร์ชัน 1.1 ขึ้นไป ตัวแปรภาษาจาวา ที่ทดสอบได้แก่
  - JAVAC.EXE ใน Sun JDK
  - MJC.EXE ใน Microsoft JDK
  - JIKES.EXE จากบริษัทไอบีเอ็ม

2.) การทดสอบการทำงานของโปรแกรม

- สามารถทำงานบนตัวรันโปรแกรมจาวา(Java Virtual Machine) ที่รองรับมาตรฐานจาวาเวอร์ชัน 1.1 ขึ้นไป ตัวรันโปรแกรมจาวาที่ทดสอบได้แก่
  - โปรแกรม JAVA.EXE ใน Sun Java Runtime Environment 1.1
  - โปรแกรม JVIEW.EXE ใน Microsoft Java Virtual Machine
- สำหรับโปรแกรมที่เป็นแอปเพล็ต(Applet)ซึ่งแสดงผลบนเวบบราวเซอร์ทำการทดสอบกับเวบบราวเซอร์ ได้แก่
  - แอปเพล็ตวิวเวอร์ (AppletViewer) ผลการทดสอบไม่มีปัญหา

- ไมโครซอฟท์อินเตอร์เนทเอ็กซ์พลอเรอร์(Microsoft Internet Explorer) เวอร์ชัน 4.0 และ 5.0 ผลการทดสอบไม่มีปัญหา
- เนทสเคปเนวิเกเตอร์ (Netscape Navigator) เวอร์ชัน 4.07 ผลการทดสอบสามารถรันได้ แต่มีปัญหาการแสดงผล และ การอ่านข้อมูลจากไฟล์

3.) ฐานข้อมูลเชิงวัตถุที่ใช้ทดสอบ

ออบเจคต์สตอร์ (Object Store) สำหรับจาวา เวอร์ชัน 6.0

รุ่น "Personal Storage Edition (PSE)/PSE Pro Release 6.0 for Java"

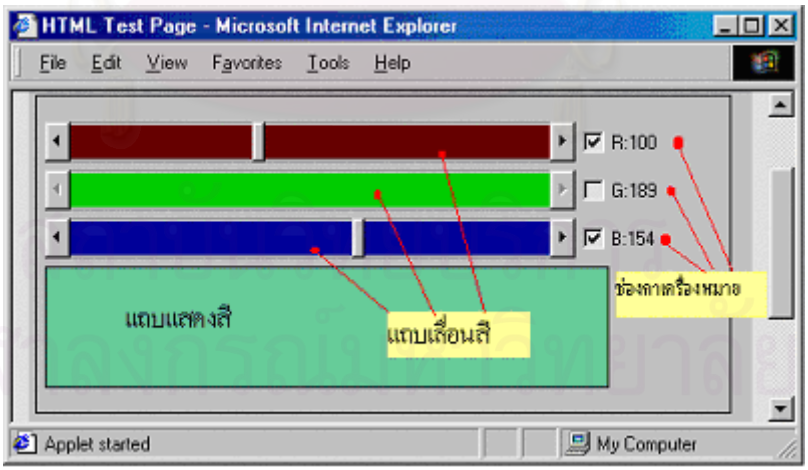
4.) ระบบปฏิบัติการที่ใช้ทดสอบ

ไมโครซอฟท์วินโดวส์ 95/98 และ NT 4.0

สำหรับระบบปฏิบัติการอื่นที่สนับสนุนจาวายังไม่มีการทดสอบ

6.3.2 การสร้างโปรแกรมแถบสี

หลังจากทำการสร้างคลาสคอมโพเนนท์เพื่อเป็นโครงร่างพัฒนาโปรแกรมใน 6.2 หัวข้อนี้จะเป็นการทดสอบ สร้างโปรแกรมจากโครงร่างพัฒนาโปรแกมดังกล่าว และเปรียบเทียบกับการเขียนโปรแกรมลักษณะเดียวกันโดยใช้วิธีมาตราฐานของจาวา สำหรับรูปแบบการเขียนโปรแกรมจะแตกต่างกันเฉพาะในส่วนของการจัดการเหตุการณ์ สำหรับส่วนอื่นๆ เช่น ส่วนของการสร้างคอมโพเนนท์ จะเขียนเหมือนกัน



รูปที่ 6.6 หน้าจอโปรแกรมแถบสี

โปรแกรมตัวอย่างที่ใช้ทดสอบ คือโปรแกรมแถบแสดงสี (Color Panel) โดยในหน้าจอโปรแกรมจะประกอบด้วย แถบเลื่อน (Scollbar) 3 ตัวเพื่อใช้ในการกำหนดค่าของแม่สี 3 สีได้แก่ สี

แดง (R) สีเขียว (G) และ สีน้ำเงิน (B) และ แผ่น (Panel) สำหรับแสดงสีที่ได้จากการผสมของ RGB และมีช่องกาเครื่องหมาย (Check Box) 3 อันเพื่อใช้ควบคุมแถบเลื่อนสี และแสดงค่าตัวเลขของสีแต่ละค่าด้วย ดังรูปที่ 6.6

<u>การทำงานของโปรแกรม</u>

- เมื่อผู้ใช้เปลี่ยนแปลงค่าของแถบเลื่อนสี สิ่งที่เกิดขึ้นได้แก่ สีของแถบเลื่อนสีจะเปลี่ยนตามค่าของแถบเลื่อน แถบสีแดงจะแสดงสีแดง แถบสีเขียวจะแสดงสีเขียว แถบสีน้ำเงินจะแสดงสีน้ำเงิน จากนั้นแผ่นแสดงสีจะแสดงสีที่เกิดจากการผสมของทั้ง 3 สี (RGB) และค่าของสีแตละสีจะแสดงที่ ช่องกาเครื่องหมาย ประจำสีนั้นๆ
- เมื่อผู้ใช้กาเครื่องหมาย ในช่องกาเครื่องหมายประจำสี ถ้ากาเครื่องหมายถูก ☑ แถบเลื่อนสีประจำช่องนั้นจะสามารถเลื่อนได้ ถ้าเอาเครื่องหมายถูกออกแถบประจำช่องนั้นจะไม่สามารถเลื่อนได้ ดังรูปช่อง สีเขียว(G) ไม่มีเครื่องหมายถูก แถบสีเขียวจะเลื่อนไม่ได้

ส่วนแรกของโปรแกรมสำหรับสร้างคอมโพเนนท์ และกำหนดตำแหน่งแสดงบนหน้าจอ ของทุกวิธีจะเหมือนกัน ดังโปรแกรมต่อไปนี้

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import com.wtv.*;
import com.wtv.awt.*;

public class ColorTest extends Applet {
  TPanel      colorPanel  = new TPanel();
  TScrollbar  redBar   = new TScrollbar(Scrollbar.HORIZONTAL,0,0,0,256);
  TScrollbar  greenBar = new TScrollbar(Scrollbar.HORIZONTAL,0,0,0,256);
  TScrollbar  blueBar  = new TScrollbar(Scrollbar.HORIZONTAL,0,0,0,256);
  TCheckbox rCheck =new TCheckbox("R:0",true);
  TCheckbox gCheck =new TCheckbox("G:0",true);
  TCheckbox bCheck =new TCheckbox("B:0",true);

//Initialize the applet
  public void init(){
    try  {
      setForeground(Color.black);
      setBackground(Color.lightGray);
      setLayout(null);
      colorPanel.setBackground(Color.black);
      redBar.setBackground(Color.black);
      greenBar.setBackground(Color.black);
      blueBar.setBackground(Color.black);
      add(colorPanel);
      add(redBar);
     add(greenBar);
      add(blueBar);
      add(rCheck);
      add(gCheck);
      add(bCheck);
```

```
      colorPanel.reshape(10,115,350,75);
      redBar.reshape(10,25,330,25);
      greenBar.reshape(10,55,330,25);
      blueBar.reshape(10,85,330,25);
      rCheck.reshape(345,25,80,25);
      gCheck.reshape(345,55,80,25);
      bCheck.reshape(345,85,80,25);

      event_Init();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
 private void event_Init(){
 …
 }
}
```

ส่วนต่อไปเป็นการเขียนโปรแกรมจัดการเหตุการณ์ จะเขียนในเมทอด “event_Init” ซึ่ง รูปแบบการเขียนของแต่ละวิธีจะแตกต่างกัน

6.3.2.1 สร้างด้วยวิธีจาวามาตรฐาน ในตัวอย่างนี้จะใช้คลาสภายใน (Inner Class) [7] ในการสร้าง คลาสอะเดปเตอร์ ซึ่งเป็นวิธีมาตรฐานของจาวา จะได้ดังโปรแกรมต่อไปนี้

```
1. private void event_Init() throws Exception {
2.    redBar.addAdjustmentListener(new java.awt.event.AdjustmentListener(){
3.      public void adjustmentValueChanged(AdjustmentEvent e) {
4.      redBar_adjustmentValueChanged(e);
5.     }});

6.    greenBar.addAdjustmentListener(new java.awt.event.AdjustmentListener(){
7.      public void adjustmentValueChanged(AdjustmentEvent e) {
8.      greenBar_adjustmentValueChanged(e);
9.     }});

10.   blueBar.addAdjustmentListener(new java.awt.event.AdjustmentListener(){
11.     public void adjustmentValueChanged(AdjustmentEvent e) {
12.     blueBar_adjustmentValueChanged(e);
13.    }});

14.   rCheck.addItemListener(new java.awt.event.ItemListener() {
15.     public void itemStateChanged(ItemEvent e) {
16.     redBar.setEnabled(rCheck.getState());
17.   }});

18.   gCheck.addItemListener(new java.awt.event.ItemListener() {
19.     public void itemStateChanged(ItemEvent e) {
20.     greenBar.setEnabled(gCheck.getState());
21.   }});

22.   bCheck.addItemListener(new java.awt.event.ItemListener() {
23.     public void itemStateChanged(ItemEvent e) {
24.     blueBar.setEnabled(gCheck.getState());
25.   }});
26. } // end of initialization Event

27. void redBar_adjustmentValueChanged(AdjustmentEvent e) {
28.    int v = e.getValue();
29.    rCheck.setLabel("Red:"+Integer.toString(v));
30.    redBar.setColorRed(v);
31.    colorPanel.setColorRed(v);
32. }
```

```
33. void greenBar_adjustmentValueChanged(AdjustmentEvent e) {
34.    int v = e.getValue();
35.    gCheck.setLabel("Green:"+Integer.toString(v));
36.    greenBar.setColorGreen(v);
37.    colorPanel.setColorGreen(v);
38. }

39. void blueBar_adjustmentValueChanged(AdjustmentEvent e) {
40.    int v = e.getValue();
41.    bCheck.setLabel("Blue:"+Integer.toString(v));
42.    blueBar.setColorBlue(v);
43.    colorPanel.setColorBlue(v);
44. }
```

เนื่องจากวิธีของจาวามาตรฐานจะต้องทำการสร้างคลาสอะเดปเตอร์ เพื่อใช้ควบคุมเหตุการณ์ เมื่อมีเหตุการณ์ที่ต้องควบคุมมาก ก็จะมีคลาสเกิดขึ้นมากตามไปด้วย ซึ่งในตัวอย่างจะต้องเขียนโปรแกรมควบคุมเหตุการณ์อย่างน้อย 6 เหตุการณ์ ทำให้เกิดคลาสอะเดปเตอร์ 6 คลาส

เมื่อทำการคอมไพล์ จะได้ไฟล์คลาสดังต่อไปนี้

ColorTest.class, ColorTest$1.class, ColorTest$2.class, ColorTest$3.class, ColorTest$4.class, ColorTest$5.class และ ColorTest$6.class รวม 7 ไฟล์ มีขนาดรวมกันทั้งหมด 6,867 ไบต์

<u>หมายเหตุ :</u> ColorTest$1-$6 เป็นคลาสภายใน (Inner Class)

<u>6.3.2.2 สร้างด้วยวิธีของตัวแปรเมทอดแบบเดี่ยวและตรวจสอบชนิดแบบสมบูรณ์</u>

```
// Event Connection
1. private void event_Init() throws Exception {
2. MethodVar.defaultMode = MethodVar.SINGLE;
3. MethodVar.typeMode    = MethodVar.FULL_TYPE;

4. redBar.setOnValueChanged(this,"redBar_adjustmentValueChanged");
5. greenBar.setOnValueChanged(this,"greenBar_adjustmentValueChanged");
6. blueBar.setOnValueChanged(this,"blueBar_adjustmentValueChanged ");

7. rCheck.setOnChanged(this,"rCheck_Changed");
8. gCheck.setOnChanged(this,"gCheck_Changed");
9. bCheck.setOnChanged(this,"bCheck_Changed");
10. }
11. public void redBar_adjustmentValueChanged(AdjustmentEvent e) {
12. int v = e.getValue();
13. rCheck.setLabel("Red:"+Integer.toString(v));
14. redBar.setColorRed(v);
15. colorPanel.setColorRed(v);
16. }

17. public void greenBar_adjustmentValueChanged(AdjustmentEvent e) {
18. int v = e.getValue();
19. gCheck.setLabel("Green:"+Integer.toString(v));
20. greenBar.setColorGreen(v);
21. colorPanel.setColorGreen(v);
22. }

23. public void blueBar_adjustmentValueChanged(AdjustmentEvent e) {
24. int v = e.getValue();
```

```
25. bCheck.setLabel("Blue:"+Integer.toString(v));
26. blueBar.setColorBlue(v);
27. colorPanel.setColorBlue(v);
28. }
29. public void rCheckBox_Changed(Object sender, boolean state){
30. redBar.setEnabled(state);
31. }

32. public void gCheckBox_Changed(Object sender, boolean state){
33. greenBar.setEnabled(state);
34. }

35. public void bCheckBox_Changed(Object sender, boolean state){
36. blueBar.setEnabled(state);
37. }
```

การเชื่อมต่อเหตุการณ์ด้วยตัวแปรเมทอดไม่จำเป็นต้องสร้างคลาสอะเดปเตอร์ เพียงกำหนดให้เหตุการณ์ที่ต้องการควบคุมการทำงานซึ่งเป็นตัวแปรเมทอดชี้ไปที่เมทอดที่เขียนขึ้นเท่านั้น และการใช้ตัวแปรเมทอดแบบจำนวนเดียวนั้นทำให้ไม่สามารถกำหนดเมทอดปลายทางให้กับเหตุการณ์หลายๆเมทอดพร้อมกัน และ เมื่อใช้วิธีการตรวจสอบแบบสมบูรณ์ทำให้ เหตุการณ์จากคอมโพเนนท์ต้นทางเช่น เหตุการณ์การเปลี่ยนค่าของ redBar ไม่สามารถเชื่อมต่อไปยัง เมทอดของคอมโพเนนท์เป้าหมายเช่น เมทอด setColorRed ใน colorPanel ได้โดยตรง เพราะรูปแบบพารามิเตอร์ไม่ตรงกัน จึงทำให้ต้องเขียนโปรแกรม **redBar_adjustmentValueChanged** เพื่อรับเหตุการณ์จาก redBar และส่งคำสั่งต่อไปที่เมทอด setColorRed ใน colorPanel
เมื่อทำการคอมไพล์ จะได้ไฟล์คลาสดังต่อไปนี้ ColorTest.class มีขนาด 3,093 ไบต์

6.3.2.3 สร้างด้วยวิธีของตัวแปรเมทอดแบบหลายจำนวนและตรวจสอบแบบบางส่วน

```
 // Event Connection
1.  private void event_Init() throws Exception {
2.  MethodVar.typeMode    = MethodVar.PARTIAL_TYPE;
3.  MethodVar.defaultMode = MethodVar.MULTIPLE;

4.  redBar.setOnValueChanged(this,"doScroll");
5.  redBar.setOnValueChanged(colorPanel,"setColorRed");
6.  redBar.setOnValueChanged(redBar,"setColorRed");

7.  greenBar.setOnValueChanged(colorPanel,"setColorGreen");
8.  greenBar.setOnValueChanged(greenBar,"setColorGreen");
9.  greenBar.setOnValueChanged(this,"doScroll");

10. blueBar.setOnValueChanged(colorPanel,"setColorBlue");
11. blueBar.setOnValueChanged(blueBar,"setColorBlue");
12. blueBar.setOnValueChanged(this,"doScroll");

13. rCheck.setOnChanged(redBar,"setEnabled");
14. gCheck.setOnChanged(greenBar,"setEnabled");
15. bCheck.setOnChanged(blueBar,"setEnabled");
16. }
17. public void doScroll(Object sender,int value){
18. if(sender==redBar)        rCheck.setLabel("R:"+Integer.toString(value));
```

```
19. else if(sender==greenBar) gCheck.setLabel("G:"+Integer.toString(value));
20. else if(sender==blueBar)  bCheck.setLabel("B:"+Integer.toString(value));
21. }
```

การใช้ตัวแปรเมทอดแบบหลายจำนวนนั้นทำให้สามารถกำหนดเมทอดปลายทางให้กับเหตุการณ์พร้อมกันหลายเมทอด และ เมื่อใช้วิธีการตรวจสอบแบบบางส่วนก็จะทำให้ เหตุการณ์จากคอมโพเนนท์ต้นทางเช่น เหตุการณ์การเปลี่ยนค่าของ redBar สามารถเชื่อมต่อไปยังเมทอดของคอมโพเนนท์เป้าหมายเช่น เมทอด setColorRed ใน colorPanel ได้โดยตรง เพราะรูปแบบพารามิเตอร์ของเหตุการณ์ และเมทอดปลายทางไม่จำเป็นต้องตรงกันทั้งหมด จึงทำให้ไม่จำเป็นต้องเขียน **redBar_adjustmentValueChanged** เพื่อรับเหตุการณ์จาก redBar จึงทำให้สามารถลดการเขียนโปรแกรมในการควบคุมเหตุการณ์ได้

เมื่อทำการคอมไพล์ จะได้ไฟล์คลาสดังต่อไปนี้ ColorTest.class มีขนาด 2,671 Kb

6.3.2.4 สร้างด้วยวิธีของตัวแปรเมทอด โดยอ่านข้อมูลการเชื่อมต่อเหตุการณ์จากไฟล์

```
1.  private void event_Init() throws Exception {
2.  colorPanel.setName("colorPanel");
3.  redBar.setName("redBar");
4.  greenBar.setName("greenBar");
5.  blueBar.setName("blueBar");
6.  rCheck.setName("rCheck");
7.  gCheck.setName("gCheck");
8.  bCheck.setName("bCheck");

9.  MethodVar.typeMode    = MethodVar.PARTIAL_TYPE;
10. MethodVar.defaultMode = MethodVar.MULTIPLE;
11. try{
12. InputStream evstream = getClass().getResourceAsStream("ColorTest.event");
13.    MethodVar.loadMethodVar(this,evstream);
14.   }catch(Exception E) {
15.     E.printStackTrace(MethodVar.debugStream);
16.  }
17. }
18. public void doScroll(Object sender,int value){
19.  if(sender==redBar)    rCheck.setLabel("R:"+Integer.toString(value));
20.  else if(sender==greenBar)gCheck.setLabel("G:"+Integer.toString(value));
21.  else if(sender==blueBar) bCheck.setLabel("B:"+Integer.toString(value));
22. }
```

และในไฟล์ ColorTest.event ขนาด 414 ไบต์ จะข้อมูลดังต่อไปนี้

```
[redBar]
  OnValueChanged=.doScroll
  OnValueChanged=colorPanel.setColorRed
  OnValueChanged=redBar.setColorRed
[greenBar]
  OnValueChanged=.doScroll;colorPanel.setColorGreen;greenBar.setColorGreen
[blueBar]
  OnValueChanged=.doScroll;colorPanel.setColorBlue;blueBar.setColorBlue
[rCheck]
  OnChanged=redBar.setEnabled
[gCheck]
  OnChanged=greenBar.setEnabled
[bCheck]
  OnChanged=blueBar.setEnabled
```

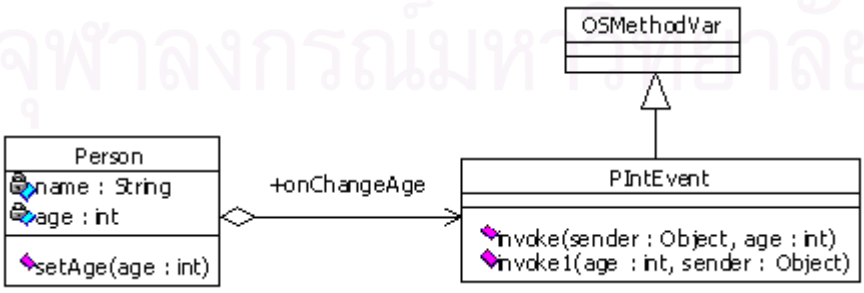
เมื่อทำการคอมไพล์ จะได้ไฟล์คลาสดังต่อไปนี้ ColorTest.class มีขนาด 2,824 Kb เหตุที่วิธีนี้จะต้องมีการกำหนดชื่อของคอมโพเนนท์ก่อนด้วยคำสั่ง setName เพราะจำเป็นต้องใช้ชื่อของคอมโพเนนท์สำหรับเชื่อมต่อเหตุการณ์ที่ระบุไว้ในไฟล์ `colortest.event` ซึ่งด้วยวิธีการนี้ทำให้ภายในโปรแกรม ไม่ต้องมีคำสั่งในการเชื่อมต่อคอมโพเนนท์ โดยข้อมูลการเชื่อมต่อเหตุการณ์จะถูกเก็บไว้ในไฟล์อื่น การแก้ไขไฟล์ที่เก็บเหตุการณ์สามารถทำให้โปรแกรมเปลี่ยนแปลงการทำงานได้ โดยที่ไม่ต้องคอมไพล์ใหม่

โปรแกรมทั้ง 4 ตัวอย่างจะทำงานได้เหมือนกัน แต่จะพบว่าแบบที่ 3 และ 4 จะเขียนโปรแกรมน้อยที่สุด และ แบบที่ 4 สามารถเปลี่ยนแปลงการทำงานของโปรแกรมได้ จากการแก้ไขไฟล์ colorTest.event โดยไม่ต้องคอมไพล์โปรแกรมใหม่ สำหรับโปรแกรมในตัวอย่างที่ 1 ซึ่งใช้วิธีการสร้างคลาสภายในเป็นคลาสอะเดปเตอร์ เมื่อคอมไพล์จะทำให้ได้จำนวนไฟล์ถึง 7 ไฟล์มีขนาดใหญ่กว่าโปรแกรมที่ใช้ ตัวแปรเมทอดประมาณ 4 กิโลไบต์ ดังตาราง 6.1

ตารางที่ 6.1 เปรียบเทียบการเขียนโปรแกรมด้วยวิธีต่างๆ

วิธีที่	จำนวนบรรทัด	จำนวนไฟล์หลังการคอมไพล์	ขนาดไฟล์รวม (ไบต์)
1. จาวามาตรฐาน	44	7	6,867
2. ตัวแปรเมทอด Single Mode, FullType	37	1	3,246
3. ตัวแปรเมทอด Multiple Mode, PartialType	21	1	2,671
4. ตัวแปรเมทอด Multiple Mode, PartialType, อ่านข้อมูลเหตุการณ์จากไฟล์	22	2	3,238

6.3.3 ทดสอบการใช้ตัวแปรเมทอดบนฐานข้อมูลเชิงวัตถุ



รูปที่ 6.7 ตัวอย่างการทดสอบตัวแปรเมทอดกับฐานข้อมูลเชิงวัตถุ

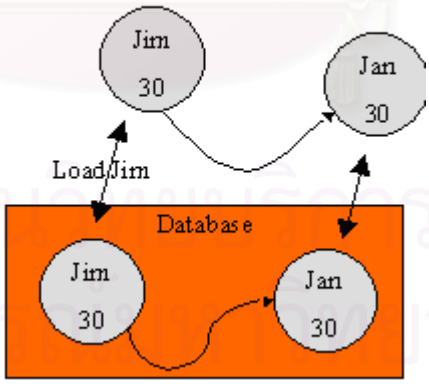
จากรูปที่ 6.7 เป็นการทดลองสร้างคลาสตัวแปรเมทอด PIntEvent เป็นเมทอดแบบหลายรูปและมีวิธีการตรวจสอบชนิดแบบบางส่วน และสืบทอดจากคลาส OSMethodVar และ คลาส Person ใช้ PIntEvent ในการสร้างเหตุการณ์ onChangeAge ซึ่งมีการเขียนเรียกใช้งานดังส่วนโปรแกรมต่อไปนี้

```
public void setAge(int a){
        age = a;
        if(onChangeAge != null)onChangeAge.invoke(this,age);
}
```

เมื่อเมทอด setAge ถูกเรียกใช้ ก็จะเกิดเหตุการณ์ onChangeAge เมทอดในเหตุการณ์นี้จะถูกเรียกให้ทำงาน ส่วนของโปรแกรมต่อไปจะเป็นการทดลองสร้างออบเจคต์ถาวรแล้วเชื่อมเหตุการณ์ดังนี้

```
Person jan = new Person("JAN",30);
Person jim = new Person("JIM",30);
jim.setOnChangeAge(jan,"setAge");
db.createRoot("JIM",jim);
db.createRoot("JAN",jan);
```

จากโปรแกรมสร้าง Jan และ Jim ทั้งสองมีอายุเท่ากันคือ 30 เรากำหนดให้ onChangeAge ของ jim อ้างถึงไปยังเมทอด setAge ของ jan จะทำให้ เมื่อเรียก jim.setAge(..) แล้ว jan.setAge(…) จะถูกเรียกด้วย จากนั้นทำการเก็บออบเจคต์ทั้งสองลงในฐานข้อมูล ข้อมูลของ onChangeAge จะถูกเก็บไว้โดยอัตโนมัติ ดังรูปที่ 6.8



รูปที่ 6.8 ตัวอย่างการเก็บตัวแปรเมทอดในฐานข้อมูล

ในอีกโปรแกรมหนึ่งทดสอบการเรียกใช้งานออบเจคต์ jim ที่อยู่ในฐานข้อมูล ดังนี้

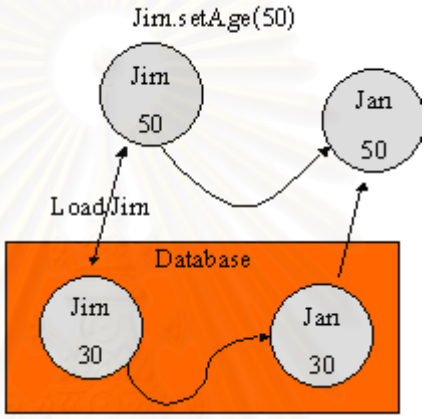
```
Person jim = (Person) db.getRoot("JIM");
jim.setAge(50);
Person jan = (Person) db.getRoot("JAN");
System.out.println("->"+jim);
System.out.println("->"+jan);
```

ผลลัพธ์ที่ได้คือ

-> jim 50

-> jan 50

จากเดิมข้อมูลของ jan คือ age = 30 แต่เมื่อรันโปรแกรม สร้างออบเจคต์ jim แล้วเรียก jim.setAge(50) จะทำให้ jan ถูกโหลดขึ้นมาจากฐานข้อมูลโดยอัตโนมัติ และ jan.setAge จะถูกเรียกด้วยทำให้ age ของ jan เป็น 50 ด้วย ดังรูปที่ 6.9



รูปที่ 6.9 ตัวอย่างการเรียกใช้ตัวแปรเมทอดในฐานข้อมูล

จะเห็นว่าตัวแปรเมทอดสามารถใช้อ้างถึงเมทอดของออบเจคต์ถาวรในฐานข้อมูลได้ เมื่อตัวแปรเมทอดถูกเรียก (invoke) จะทำให้ออบเจคต์ที่มันอ้างถึงถูกโหลดขึ้นมา สู่หน่วยความจำทำให้เมทอดเป้าหมายถูกเรียกให้ทำงานได้ ซึ่งกลไกดังกล่าวเกิดขึ้นโดยอัตโนมัติ แม้ในตัวอย่างจะเป็นการอ้างถึงออบเจคต์ในคลาสเดียวกัน แต่ก็ไม่มีปัญหาสำหรับออบเจคต์ของคลาสที่ต่างกัน เพราะตัวแปรเมทอดไม่สนใจชนิดของคลาสอยู่แล้ว

บทที่ 7

สรุปผลการวิจัย และข้อเสนอแนะ

7.1 สรุปผลการวิจัย

เทคนิคตัวแปรเมทอดที่พัฒนาขึ้นในงานวิจัยนี้ สามารถนำไปใช้ในการเขียนโปรแกรมเพื่อใช้แทนความสามารถของตัวชี้เมทอดบนจาวาได้ ซึ่งนำไปใช้ในสร้างส่วนการจัดการเหตุการณ์ ในการเชื่อมต่อระหว่างคอมโพเนนท์ได้ และด้วยความสามารถของตัวแปรเมทอดแบบหลายจำนวนใช้ร่วมกับตัวแปรเมทอดแบบหลายรูปและการตรวจสอบชนิดแบบบางส่วน เป็นการเพิ่มความยืดหยุ่นให้กับส่วนจัดการเหตุการณ์ ซึ่งสามารถช่วยลดการเขียนโปรแกรมสำหรับเชื่อมต่อเหตุการณ์ระหว่างคอมโพเนนท์และทำให้สามารถเชื่อมต่อเหตุการณ์ระหว่างรันโปรแกรมได้ ซึ่งนับเป็นจุดเด่นที่แตกต่างกับวิธีการของตัวชี้เมทอดในเครื่องมืออื่น ๆ เช่นใน “Delegate” ของ วิชวลเจเวอร์ชัน 6.0 หรือ การสร้างเหตุการณ์ในเดลไฟ ที่เหตุการณ์สามารถมีรูปแบบพารามิเตอร์ได้เพียงแบบเดียว ทำให้การเชื่อมต่อคอมโพเนนท์ระหว่างรันเป็นไปได้ยาก

จากการทดสอบการพัฒนาโปรแกรมด้วยตัวแปรเมทอดในบทที่ 6 จะพบว่าการใช้วิธีของตัวแปรเมทอดทำให้จำนวนบรรทัดของการเขียนน้อยกว่าวิธีของจาวามาตรฐาน และ เมื่อคอมไพล์โปรแกรมจะได้ขนาดของไฟล์ที่เล็กกว่าด้วย นอกจากนี้ในตัวอย่างการอ่านข้อมูลตัวแปรเมทอดซึ่งเก็บการเชื่อมต่อเหตุการณ์ระหว่างคอมโพเนนท์จากไฟล์ แสดงให้เห็นว่าเทคนิคตัวแปรเมทอดสามารถสร้างการเชื่อมต่อระหว่างคอมโพเนนท์ในระหว่างรันได้เป็นอย่างดี

7.2 สรุปข้อดีข้อเสียของตัวแปรเมทอด

โดยจะเปรียบเทียบกับวิธีการ “Adapter Class” ในจาวามาตรฐาน

1) **การนำกลับมาใช้ใหม่** (Reuseability)

จากตัวอย่างในบทที่ 6 จะเห็นว่าเหตุการณ์ต่างๆ สามารถใช้คลาสตัวแปรเมทอดเดียวกันได้ โดยไม่จำเป็นต้องสร้างคลาสตัวแปรเมทอดใหม่ยกเว้นต้องการเหตุการณ์ที่มรูปแบบพารามิเตอร์อื่น และ คลาสตัวแปรเมทอดที่สร้างขึ้นสามารถใช้ซ้ำได้อีก ซึ่งต่าจากวิธีของจาวามาตรฐานที่จะต้องสร้าง อะเดปเตอร์คลาสทุกครั้ง และคลาสเหล่านั้นจะไม่มการนำกลับมาใช้ใหม่อีก

2) **ขนาดของโปรแกรม**(Program Size)

เนื่องจากวิธีของตัวแปรเมทอดไม่ต้องสร้างคลาสในการเชื่อมต่อทำให้ จำนวนคลาสน้อยกว่าและขนาดของโปรแกรมโดยรวมจะเล็กกว่าวิธีของจาวามาตรฐาน เพราะโดยปกติอะเดปเตอร์คลาสจะมีขนาดอย่างน้อย 500 ไบต์ ยิ่งในโปรแกรมมีการเชื่อมต่อเหตุการณ์มาก วิธีของจาวามาตรฐานจะเกิดคลาสอะเดปเตอร์จำนวนมาก ขนาดของโปรแกรมจะใหญ่ตามไปด้วย

แต่ในทางกลับกันเมื่อรันโปรแกรมขนาดของหน่วยความจำที่ใช้สำหรับตัวแปรเมทอดจะมีขนาดใหญ่กว่า เพราะตัวแปรเมทอดจะมีข้อมูลภายในมากกว่า

3) **ความเร็วในการทำงาน** (Speed)

เนื่องจากกลไกการทำงานของตัวแปรเมทอดเป็นแบบพลวัต การเรียกใช้เมทอดผ่านตัวแปรจะมีการสร้างผ่านพารามิเตอร์หลายชั้น ทำให้ช้ากว่าการเรียกใช้ผ่านคลาสอะเดเตอร์ อย่างไรก็ตามในการทดสอบสร้างโปรแกรมในบทที่ 6 จะไม่เห็นความแตกต่างด้านความเร็ว เพราะซีพียูในปัจจุบันมีการทำงานที่เร็วมาก

เมื่อมองอีกมุมหนึ่งจาวามาตรฐานจะมีจำนวนคลาสมากกว่าและขนาดของไฟล์มากกว่าดังนั้นถ้าเป็นโปรแกรมที่อยู่บนอินเตอร์เนต โปรแกรมที่เขียนด้วยตัวแปรเมทอดมขนาดเล็กกว่าจะถูกดาวน์โหลดได้เร็วกว่า

4) **ความยืดหยุ่น** (Flexibility)

เนื่องจากการเชื่อมต่อด้วยตัวแปรเมทอดไม่ต้องสร้างคลาสใหม่ และ ตัวแปรเมทอดสามารถมีรูปแบบในการส่งพารามิเตอร์หลายแบบได้ทำให้มีความยืดหยุ่นในการใช้งานสูงขึ้น นอกจากนี้การเพิ่มเติมรูปแบบพารามิเตอร์ใหม่ให้กับตัวแปรเมทอดก็สามารถทำได้ง่าย ถ้าไม่มีการแก้ไขที่รูปแบบหลักก็จะไม่มีผลกระทบต่อคอมโพเนนท์อื่นๆ

5) **การตรวจสอบข้อผิดพลาด** (Error Checking)

การตรวจสอบข้อผิดพลาดของตัวแปรเมทอดจะเกิดในขณะรันโปรแกรม ซึ่งเป็นข้อเสียเพราะทำให้ไม่ทราบว่าจะเกิดข้อผิดพลาดก่อนการรันโปรแกรม

7.3 ข้อจำกัดของตัวแปรเมทอด

ในส่วนของการจัดเก็บและเรียกใช้เมทอดนั้น ตัวแปรเมทอดจะไม่สามารถจัดเก็บเมทอดของออบเจคต์ของคลาสที่ไม่ใช่ “Public” และอยู่ในเพจเกจ (Package) อื่นได้ เมทอดที่จะเก็บได้จะต้องเป็นเมทอดแบบ “Public” เนื่องจากตัวแปรเมทอดถือเป็นออบเจคต์ภายนอกทำให้ไม่สามารถเข้าถึงข้อมูลและเมทอดที่เป็นส่วนตัว (Private) ของออบเจคต์อื่นได้

ปัญหาของการเก็บเมทอดของออบเจคต์ปลายทางที่มีการตั้งชื่อซ้ำกัน (Overloading) เช่น ในออบเจคต์ปลายทางมีเมทอด print(int x) , print(int x , int y) เป็นต้น เพราะการกำหนดเมทอดให้ตัวแปรจะระบุด้วยออบเจคต์ และ ชื่อเมทอด ในกรณีนี้ ถ้าใช้วิธีการตรวจสอบชนิดพารามิเตอร์แบบสมบูรณ์ จะไม่เกิดปัญหาในการเลือกเมทอดปลายทาง แต่ถ้าใช้วิธีการตรวจสอบชนิดพารามิเตอร์แบบบางส่วนแล้ว จะทำให้ print(int x) และ print(int x,int y) มีโอกาสถูกเลือกได้ทั้ง 2 ตัว สำหรับวิธีที่ใช้อยู่ในงานวิจัยนี้ จะเลือกเอาเมทอดแรกที่พบว่าชนิดพารามิเตอร์เข้ากันได้

ข้อจำกัดของการจัดเก็บข้อมูลของตัวแปรเมทอดลงในไฟล์ และ อ่านกลับคืนจากไฟล์นั้น จะเก็บได้เฉพาะตัวแปรเมทอดที่เป็นเหตุการณ์ของคอมโพเนนท์ และคอมโพเนนท์ต้องมีการระบุชื่อก่อนด้วย

ข้อจำกัดในระบบแบบกระจาย (Distributed System) ตัวแปรเมทอดยังไม่สามารถเก็บเมทอดของออบเจคต์ ซึ่งออบเจคต์ทำงานอยู่ในเครื่องที่แตกต่างกันได้

7.4 ข้อเสนอแนะในการพัฒนาเพิ่มเติม

เนื่องจากในงานวิจัยนี้จะมุ่งเน้นในการพัฒนากลไกสำหรับสร้างและใช้งานตัวแปรเมทอดเพื่อใช้ในการสร้างส่วนจัดการเหตุการณ์เท่านั้น ยังไม่มีการทดสอบการใช้ในเครื่องมือพัฒนาโปรแกรมแบบวิชวล ซึ่งผู้วิจัยมีความเห็นว่ากลไกของตัวแปรเมทอดซึ่งทำให้คอมโพเนนท์สามารถเชื่อมต่อกันแบบพลวัตได้ จะทำให้การพัฒนาโปรแกรมทำได้ง่ายขึ้น

นอกจากการใช้ตัวแปรเมทอดสำหรับการสร้างเหตุการณ์ในคอมโพเนนท์แล้ว ยังเป็นไปได้ที่จะประยุกต์ใช้กับออบเจคต์พร้อมทำงาน(Active Object) เพราะจากความสามารถของตัวแปรเมทอด ทำให้ออบเจคต์สามารถตั้งเวลาให้ระบบเรียกใช้เมทอดของตนได้ โดยสร้างตัวแปรเมทอดให้เก็บเมทอดของตนไว้ เมื่อถึงเวลาที่กำหนดระบบก็จะสามารถเรียกเมทอดในตัวแปรนั้นได้

ปัจจุบันตัวแปรเมทอดยังไม่สามารถใช้ได้กับออบเจคต์ในระบบแบบกระจายได้ แต่หากมีการพัฒนาเพิ่มเติมให้ตัวแปรเมทอดเก็บและเรียกใช้เมทอดของออบเจคต์ที่อยู่ต่างเครื่องได้ จะมีประโยชน์ทำให้สามารถเชื่อมต่อคอมโพเนนท์แบบกระจายในลักษณะพลวัตได้สะดวกขึ้น

รายการอ้างอิง

[1] Sun Microsystems, Inc. Java(TM) 2 Platform, Standard Edition, v1.2.2 API Specification. http://www.javasoft.com/products/jdk/1.2/docs/api/index.html, Jun-1999.

[2] Microsoft Corporation. Visual J++ Technical Article: Delegates in Visual J++ 6.0. http://msdn.microsoft.com/visualj/technical/articles/delegates/default.asp, Sep-1998.

[3] Microsoft Corporation. Visual J++ Technical Article: The Truth about Delegates. http://msdn.microsoft.com/visualj/technical/articles/delegates/truth.asp, Sep-1998.

[4] Sun Microsystems. White Paper: About Microsoft's 'Delegates'. http://www.javasoft .com/docs/white/delegates.html, Sep-1998.

[5] Sun Microsystems, Inc. White Paper Sidebar: Implementing Delegates using Reflection. http://www.javasoft.com/docs/white/sidebar.html, Sep-1998.

[6] Fred Bulback. Programming Delphi Custom Components. New York: M&T Books, 1998.

[7] Sun Microsystems, Inc. Inner Classes Specification, http://java.sun.com/products/jdk/1.1/docs/, Mar-1997.

[8] Gamma, Helm, Johnson & Vlissides. Design Patterns. (n.p.):Addison-Wesley, 1995.

[9] Michael Morrison. Presenting JavaBeans. U.S.A: Sams Publishing, 1997.

[10] Charlie Calvert. Borland C++ Builder Unleashed. U.S.A: Sams Publishing, 1997.

[11] Bindu R. Rao. Object-Oriented Database Technology, Application, and Products. (n.p.): McGraw-Hill, 1994.

[12] Buchmann, A.P.,and Zimmermann, J. Building an integrated active OODBMS: requirements, architecture, and design decisions. Data Engineering,Proceedings of the Eleventh International Conference 1995 :117–128.

[13] McAuliffe, M.L.; Solomon, M.H. A trace-based simulation of pointer swizzling techniques. Data Engineering, Proceedings of the Eleventh International Conference 1995:52–61.

ภาคผนวก

ภาคผนวก ก.

ไฟล์ และ คลาส ในไลบรารี่ของตัวแปรเมทอด

ในส่วนนี้จะแสดงรายละเอียดของไฟล์ และ คลาสที่อยู่ในไลบรารี่ของตัวแปรเมทอด

ตารางที่ ก. 1 **รายการไฟล์ต่าง ๆในไลบรารี่ตัวแปรเมทอด**

ไดเรคทอรี่	**รายการไฟล์**
com\wtv\ เก็บไฟล์คลาสตัวแปรเมทอด	MethodVar.java MethodVarFilter.java
com\wtv**util** เก็บโปรแกรมเครื่องมือสร้าง คลาสตัวแปรเมทอด	MVBuilder.java OSMVBuilder.java
com\wtv**awt** เก็บไฟล์คอมโพเนนท์	awtevent.def TScrollbar.java TCheckbox.java TDialog.java TFrame.java TLabel.java TList.java TMenu.java TMenuItem.java TPanel.java TButton.java TTextField.java TWindow.java TAdjustmentEvent.java TWindowEnable.java TActionEnable.java TAdjustmentAdapter.java TAdjustmentEnable.java TComponentAdapter.java TComponentEnable.java TInputMethodAdapter.java TInputMethodEnable.java TItemAdapter.java TItemEnable.java TKeyAdapter.java TKeyEnable.java TMouseAdapter.java TMouseEnable.java TPropertyChangeAdapter.java TPropertyChangeEnable.java TStateAdapter.java TStateEnable.java TTextAdapter.java TTextEnable.java TWindowAdapter.java TActionAdapter.java TItemEvent.java TKeyEvent.java TMouseEvent.java TNotifyEvent.java TPropertyChangeEvent.java TStateEvent.java TTextEvent.java

com\wtv**odi** เก็บไฟล์ตัวแปรเมทอดสำหรับฐานข้อมูลเชิงวัตถุ	OSMethodVar.java odievent.def PNotifyEvent.java PStateEvent.java PTextEvent.java PIntEvent.java

เมื่อนำมาจัดแสดงเป็นโครงสร้างของคลาสจะได้ดังต่อไปนี้ โดยคลาสที่ขีดเส้นใต้คือคลาสที่สร้างในงานวิจัยนี้ ส่วนคลาสอื่นเป็นคลาสในไลบรารี่มาตรฐานของจาวา

Hierarchy For All Packages

Package Hierarchies:

com.wtv, com.wtv.awt, com.wtv.odi, com.wtv.util

Class Hierarchy

- class java.lang.Object
  - class java.awt.Component
    - class java.awt.Button
      - class com.wtv.awt.**TButton** (implements com.wtv.awt.TActionEnable, com.wtv.awt.TComponentEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable)
    - class java.awt.Checkbox
      - class com.wtv.awt.**TCheckbox** (implements com.wtv.awt.TComponentEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable, com.wtv.awt.TStateEnable)
    - class java.awt.Choice
      - class com.wtv.awt.**TChoice** (implements com.wtv.awt.TComponentEnable, com.wtv.awt.TItemEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable)
    - class java.awt.Label
      - class com.wtv.awt.**TLabel** (implements com.wtv.awt.TComponentEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable)
    - class java.awt.List
      - class com.wtv.awt.**TList** (implements com.wtv.awt.TActionEnable, com.wtv.awt.TComponentEnable, com.wtv.awt.TItemEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable)
    - class java.awt.Scrollbar
      - class com.wtv.awt.**TScrollbar** (implements com.wtv.awt.TAdjustmentEnable, com.wtv.awt.TComponentEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable)
    - class java.awt.TextComponent
      - class java.awt.TextField
        - class com.wtv.awt.**TTextField** (implements com.wtv.awt.TActionEnable, com.wtv.awt.TComponentEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable, com.wtv.awt.TTextEnable)
    - class java.awt.MenuComponent
      - class java.awt.MenuItem
        - class java.awt.Menu (implements java.awt.MenuContainer)
          - class com.wtv.awt.**TMenu** (implements com.wtv.awt.TActionEnable)
        - class com.wtv.awt.**TMenuItem** (implements com.wtv.awt.TActionEnable)
    - class java.awt.Container
      - class java.awt.Panel

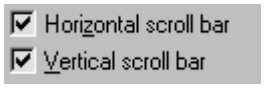
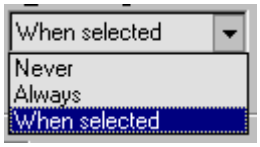
        - class com.wtv.awt.**TPanel** (implements com.wtv.awt.TComponentEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable)
        - class java.awt.Window
            - class java.awt.Dialog
                - class com.wtv.awt.**TDialog** (implements com.wtv.awt.TComponentEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable, com.wtv.awt.TWindowEnable)
            - class java.awt.Frame
                - class com.wtv.awt.**TFrame** (implements com.wtv.awt.TComponentEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable, com.wtv.awt.TWindowEnable)
            - class com.wtv.awt.**TWindow** (implements com.wtv.awt.TComponentEnable, com.wtv.awt.TKeyEnable, com.wtv.awt.TMouseEnable)
- class com.wtv.**MethodVar** (implements java.io.Serializable)
    - class com.wtv.awt.**TAdjustmentEvent**
    - class com.wtv.awt.**TItemEvent**
    - class com.wtv.awt.**TKeyEvent**
    - class com.wtv.awt.**TMouseEvent**
    - class com.wtv.awt.**TNotifyEvent**
    - class com.wtv.awt.**TPropertyChangeEvent**
    - class com.wtv.awt.**TStateEvent**
    - class com.wtv.awt.**TTextEvent**
- class com.wtv.odi.**MethodVar**
    - class com.wtv.odi.**OSMethodVar**
        - class com.wtv.odi.**PIntEvent**
        - class com.wtv.odi.**PNotifyEvent**
        - class com.wtv.odi.**PStateEvent**
        - class com.wtv.odi.**PTextEvent**
- class com.wtv.**MethodVarFilter**
- class com.wtv.util.**MVBuilder**
- class com.wtv.util.**OSMVBuilder**
- class com.wtv.awt.**TActionAdapter** (implements java.awt.event.ActionListener)
- class com.wtv.awt.**TAdjustmentAdapter** (implements java.awt.event.AdjustmentListener)
- class com.wtv.awt.**TComponentAdapter** (implements java.awt.event.ComponentListener, java.awt.event.FocusListener)
- class java.lang.Thread (implements java.lang.Runnable)
    - class com.wtv.**TTimer**
- class com.wtv.awt.**TInputMethodAdapter** (implements java.awt.event.InputMethodListener)
- class com.wtv.awt.**TItemAdapter** (implements java.awt.event.ItemListener)
- class com.wtv.awt.**TKeyAdapter** (implements java.awt.event.KeyListener)
- class com.wtv.awt.**TMouseAdapter** (implements java.awt.event.MouseListener, java.awt.event.MouseMotionListener)
- class com.wtv.awt.**TPropertyChangeAdapter** (implements java.beans.PropertyChangeListener)
- class com.wtv.awt.**TStateAdapter** (implements java.awt.event.ItemListener)
- class com.wtv.awt.**TTextAdapter** (implements java.awt.event.TextListener)
- class com.wtv.awt.**TWindowAdapter** (implements java.awt.event.WindowListener)

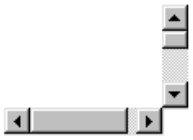
Interface Hierarchy

- interface java.awt.Adjustable
    - interface com.wtv.awt.**TAdjustmentEnable**
- interface com.wtv.awt.**TActionEnable**

- interface com.wtv.awt.**TComponentEnable**
- interface com.wtv.awt.**TInputMethodEnable**
- interface com.wtv.awt.**TItemEnable**
- interface com.wtv.awt.**TKeyEnable**
- interface com.wtv.awt.**TMouseEnable**
- interface com.wtv.awt.**TPropertyChangeEnable**
- interface com.wtv.awt.**TStateEnable**
- interface com.wtv.awt.**TTextEnable**
- interface com.wtv.awt.**TWindowEnable**

ตารางที่ ก. 2 คอมโพเนนท์ใน com.wtv.awt

คลาส	รายละเอียด
1. TButton 	สำหรับสร้างปุ่มคำสั่ง เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TActionEnable
2. TCheckbox 	แสดงข้อความ และ สร้างช่องสำหรับกาเครื่องหมาย เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TStateEnable
3. TChoice 	แสดงกล่องข้อความ พร้อมตัวเลือก เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TItemEnable
4. TDialog	สร้างหน้าต่างแบบไดอะล็อก เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TWindowEnable
5. TFrame	สร้างหน้าต่างแบบเฟรม เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TWindowEnable
6. TLabel	เป็นแถบแสดงข้อความ เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable
7. TList 	กล่องรายการสำหรับสร้างตัวเลือก เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TItemEnable, TActionEnable

8. TMenu	เป็นเมนูหลัก เหตุการณ์ที่สนับสนุนได้แก่ TActionEnable
9. TMenuItem	เป็นรายการของเมนู เหตุการณ์ที่สนับสนุนได้แก่ TActionEnable
10. TPanel	เป็นแผ่นสำหรับวางคอมโพเนนท์อื่นๆ เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TWindowEnable
11. TScrollbar 	เป็นแถบเลื่อน เพื่อเลือกค่า เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TAdjustmentEnable
12. TTextField 	เป็นช่องสำหรับแก้ไขข้อความ เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TTextEnable, TActionEnable
13. TWindow	สร้างหน้าต่างแบบปกติ เหตุการณ์ที่สนับสนุนได้แก่ TMouseEnable, TComponentEnable, TKeyEnable, TWindowEnable

รายละเอียดของเหตุการณ์ที่สนับสนุนแต่ละแบบ แสดงในตารางต่อไปนี้

ตารางที่ ก. 3 เหตุการณ์สำหรับ TAdjustmentEnable

เหตุการณ์	คลาส	รายละเอียด
1. OnBlockDecrement	TAdjustmentEvent	เมื่อปุ่มเลื่อน ถูกลากลง (หรือไปด้านซ้าย)
2. OnBlockIncrement	TAdjustmentEvent	เมื่อปุ่มเลื่อน ถูกเลื่อนขึ้น (หรือไปด้สนขวา)
3. OnMaximum	TNotifyEvent	เมื่อแถบถูกเลื่อนจนสูงสุด
4. OnMinimum	TNotifyEvent	เมื่อแถบถูกเลื่อนลงจนต่ำสุด
5. OnTrack	TAdjustmentEvent	เมื่อใช้เมาส์ลากที่ปุ่มเลื่อน
6. OnUnitDecrement	TAdjustmentEvent	เมื่อผู้ใช้ กดปุ่มที่ลูกศรด้านล่าง หรือ ด้านซ้าย
7. OnUnitIncrement	TAdjustmentEvent	เมื่อผู้ใช้ กดปุ่มที่ลูกศรด้านบน หรือ ด้านขวา
8. OnValueChanged	TAdjustmentEvent	เมื่อค่ามีการเปลี่ยนแปลง

คลาสที่ใช้ได้แก่ TScrollbar

ตารางที่ ก. 4 เหตุการณ์สำหรับ TComponentEnable

เหตุการณ์	คลาส	รายละเอียด
1. OnGotFocus	TNotifyEvent	เมื่อคอมโพเนนท์ได้รับการควบคุม
2. OnHide	TNotifyEvent	เมื่อคอมโพเนนท์ซ่อนตัว
3. OnLostFocus	TNotifyEvent	เมื่อคอมโพเนนท์เสียการควบคุม
4. OnMoved	TNotifyEvent	เมื่อคอมโพเนนท์ถูกย้ายตำแหน่ง
5. OnResized	TNotifyEvent	เมื่อคอมโพเนนท์มีการเปลี่ยนขนาด
6. OnShow	TNotifyEvent	เมื่อคอมโพเนนท์ปรากฏตัว

คลาสที่ใช้ได้แก่ TDialog, TLabel, TPanel, TWindow, TTextField, TFrame,TCheckbox, TList, TButton, TScrollbar, TChoice

ตารางที่ ก. 5 เหตุการณ์สำหรับ TItemEnable

เหตุการณ์	คลาส	รายละเอียด
1. OnChanged	TItemEvent	เมื่อผู้ใช้เปลี่ยนตัวเลือก

คลาสที่ใช้ได้แก่ TChoice, TList

ตารางที่ ก. 6 เหตุการณ์สำหรับ TKeyEnable

เหตุการณ์	คลาส	รายละเอียด
1. OnKeyDown	TKeyEvent	เมื่อคีย์ถูกกด
2. OnKeyTyped	TKeyEvent	เมื่อคีย์ที่พิมพ์เป็นตัวอักษร
3. OnKeyUp	TKeyEvent	เมื่อคีย์ถูกปล่อย

คลาสที่ใช้ได้แก่ TButton, TCheckbox, TChoice, TDialog, TFrame, TLabel, TList, TPanel, TScrollbar, TTextField, TWindow

ตารางที่ ก. 7 เหตุการณ์สำหรับ TMouseEnable

เหตุการณ์	คลาส	รายละเอียด
1. OnMouseClicked	TMouseEvent	เมื่อปุ่มเมาส์ถูกคลิ้ก (กดและปล่อยทันที)
2. OnMouseDblClicked	TMouseEvent	เมื่อปุ่มเมาส์ถูกคลิ้กซ้ำ 2 ครั้ง ติดต่อกัน
3. OnMouseDrag	TMouseEvent	เมื่อกดปุ่มเมาส์แล้วลาก
4. OnMouseEntered	TMouseEvent	เมื่อลูกศรของเมาส์เข้ามาในบริเวณคอมโพเนนท์

5. OnMouseExited	TMouseEvent	เมื่อลูกศรของเมาส์ออกจากบริเวณคอมโพเนนท์
6. OnMouseMove	TMouseEvent	เมื่อลูกศรของเมาส์มีการเคลื่อนที่
7. OnMousePressed	TMouseEvent	เมื่อปุ่มเมาส์ถูกกด
8. OnMouseReleased	TMouseEvent	เมื่อปุ่มเมาส์ถูกปล่อย

คลาสที่ใช้ได้แก่ TButton, TCheckbox, TChoice, TDialog, TFrame, TLabel, TList, TPanel, TScrollbar, TTextField, TWindow

ตารางที่ ก. 8 เหตุการณ์สำหรับ TStateEnable

เหตุการณ์	คลาส	รายละเอียด
1. OnChanged	TStateEvent	เมื่อค่าของเครื่องหมายในช่องถูกเปลี่ยน
2. OnDeselected	TStateEvent	เมื่อไม่ถูกเลือก (เอาเครื่องหมายเลือกออก)
3. OnSelected	TStateEvent	เมื่อถูกเลือก (กา เครื่องหมายเลือกในช่อง)

คลาสที่ใช้ได้แก่ TCheckBox

ตารางที่ ก. 9 เหตุการณ์สำหรับ TTextEnable

เหตุการณ์	คลาส	รายละเอียด
1. OnValueChanged	TTextEvent	ข้อความถูกแก้ไข

คลาสที่ใช้ได้แก่ TTextField

ตารางที่ ก. 10 เหตุการณ์สำหรับ TWindowEnable

เหตุการณ์	คลาส	รายละเอียด
1. OnWindowActivated	TNotifyEvent	หน้าต่างถูกเรียกให้ทำงาน
2. OnWindowClosed	TNotifyEvent	หน้าต่างถูกปิด
3. OnWindowClosing	TNotifyEvent	หน้าต่างกำลังปิด
4. OnWindowDeactivated	TNotifyEvent	เมื่อการทำงานย้ายไปหน้าต่างอื่น
5. OnWindowDeiconified	TNotifyEvent	หน้าต่างถูกขยายจากไอคอน
6. OnWindowIconified	TNotifyEvent	หน้าต่างถูกย่อให้เป็นไอคอน
7. OnWindowOpened	TNotifyEvent	หน้าต่างถูกเปิด

คลาสที่ใช้ได้แก่ TDialog, TFrame

ตารางที่ ก. 11 แสดงรายละเอียดการใช้งานสำหรับคลาสตัวแปรเมทอดแต่ละคลาส

คลาสตัวแปรเมทอด	รายละเอียด
TAdjustmentEvent	แทน java.awt.event.AdjustmentEvent สำหรับเหตุการณ์ onValueChanged, onUnitIncrement onUnitDecrement, onBlockIncrement, onBlockDecrement, onTrack ในคอมโพเนนท์ TScrollbar
TItemEvent	แทน java.awt.event.ItemEvent สำหรับเหตุการณ์ onChanged ในคอมโพเนนท์ TList
TKeyEvent	แทน java.awt.event.KeyEvent สำหรับเหตุการณ์ onKeyUp, onKeyDown, onKeyTyped ในทุกคอมโพเนนท์
TMouseEvent	แทน java.awt.event.MouseEvent สำหรับเหตุการณ์ onMouseClicked, onMouseDblClicked, onMouseEntered, onMouseExited, onMousePressed, onMouseReleased onMouseDrag, onMouseMove ในทุกคอมโพเนนท์
TNotifyEvent	สำหรับเหตุการณ์ onGotFocus, onLostFocus, onShow, onHide, onMoved, onResized ในทุกคอมโพเนนท์
TPropertyChangeEvent	แทน java.bean.PropertyChangeEvent
TStateEvent	แทน StateEvent สำหรับเหตุการณ์ onChanged, onSelected, onDeselected ใน Checkbox
TTextEvent	สำหรับส่งเหตุการณ์ข้อความ

ภาคผนวก ข.

ตัวอย่างของไฟล์ที่สำคัญ

ในส่วนนี้จะแสดงตัวอย่างของไฟล์ที่สำคัญในไลบรารี่ตัวแปรเมทอด

1. ไฟล์ข้อมูลนิยามตัวแปรเมทอดสำหรับเหตุการณ์ใน AWT

File : awtevent.def

```
// Method Varible class for AWT Event Model
package com.wtv.method
import java.awt.*
import java.awt.event.*
[TAdjustmentEvent]
  // TAdjustmentEvent for Scrollbar component
  //    onValueChanged,  onValueChanged,   onUnitIncrement
  //    onUnitDecrement, onBlockIncrement, onBlockDecrement, onTrack;
  :void invoke
       AdjustmentEvent e
  :void invoke1
       int value           = e.getValue()
       int type            = e.getAdjustmentType()
       Adjustable sender   = e.getAdjustable()
  :void invoke2
       Adjustable sender   = e.getAdjustable()
       int value           = e.getValue()
       int type            = e.getAdjustmentType()
[TItemEvent]
  // TItemEvent for List component
  //   onChanged
  :void invoke
     ItemEvent e
  :void invoke1
     Object item      = e.getItem()
     int state        = e.getStateChange()
     Object sender    = e.getSource()
  :void invoke2
     Object item      = e.getItem()
     Object sender    = e.getSource()
     int state        = e.getStateChange()
  :void invoke3
     int state        = e.getStateChange()
     Object item      = e.getItem()
     Object sender    = e.getSource()
[TKeyEvent]
  import java.lang.*
  // TKeyEvent for all components
  //   onKeyUp, onKeyDown, onKeyTyped
  :void invoke
     KeyEvent e
  :void invoke1
     Object sender     = e.getSource()
     int id            = e.getID()
     long when         = e.getWhen()
     int modifiers     = e.getModifiers()
     int keyCode       = e.getKeyCode()
     char keyChar      = e.getKeyChar()
  :void invoke2
     int keyCode       = e.getKeyCode()
     long when         = e.getWhen()
     int modifiers     = e.getModifiers()
     int id            = e.getID()
     char keyChar      = e.getKeyChar()
     Object sender     = e.getSource()
  :void invoke3
     char keyChar      = e.getKeyChar()
     long when         = e.getWhen()
     int modifiers     = e.getModifiers()
```

```
    int id            = e.getID()
    int keyCode       = e.getKeyCode()
    Object sender     = e.getSource()
[TMouseEvent]
  // TMouseEvent for all components
  //    onMouseClicked, onMouseDblClicked,  onMouseEntered
  //    onMouseExited,  onMousePressed,     onMouseReleased
  //    onMouseDrag,    onMouseMove;
  :void invoke
       MouseEvent e
  :void invoke1
       Object sender        = e.getSource()
       int x                = e.getX()
       int y                = e.getY()
       int modifiers        = e.getModifiers()
       int clickCount       = e.getClickCount()
       boolean popupTrigger = e.isPopupTrigger()
       long when            = e.getWhen()
  :void invoke2
       int x                = e.getX()
       int y                = e.getY()
       int modifiers        = e.getModifiers()
       int clickCount       = e.getClickCount()
       boolean popupTrigger = e.isPopupTrigger()
       long when            = e.getWhen()
       Object sender        = e.getSource()
[TNotifyEvent]
  // TNotifyEvent for all components
  //   onGotFocus, onLostFocus, onShow
  //   onHide,     onMoved,     onResized
  :void invoke
       Object sender
[TPropertyChangeEvent]
  import java.beans.PropertyChangeEvent
  :void invoke
     PropertyChangeEvent e
  :void invoke1
     Object source         = e.getSource()
     String propertyName   = e.getPropertyName()
     Object oldValue       = e.getOldValue()
     Object newValue       = e.getNewValue()
  :void invoke2
     String propertyName   = e.getPropertyName()
     Object oldValue       = e.getOldValue()
     Object newValue       = e.getNewValue()
     Object source         = e.getSource()
[TStateEvent]
  // TStateEvent for Checkbox component
  // onChanged,  onSelected, onDeselected
  :void invoke
     Object sender
     boolean state
  :void invoke1
     boolean state    = state
     Object sender = sender
[TTextEvent]
  :void invoke
     Object sender
     String text
  :void invoke1
     String text      = text
     Object sender    = sender
```

การคอมไพล์ด้วยคำสั่ง

c:\> java com.wtv.util.MVBuilder –c awtevent.def

2. ตัวอย่างไฟล์ TAdjustmentEvent.java

ซึ่งสร้างจาก com.wtv.util.MVBuilder

```
/*********************************************
 Method Variable Class
 Class Name  : TAdjustmentEvent
 Generate by :Method Builder 1.0
 Wachirawut Tamviset
*********************************************/
//Method Varible class for AWT Event Model;
package com.wtv.awt;
import com.wtv.MethodVar;
import java.awt.*;
import java.awt.event.*;
//TAdjustmentEvent for Scrollbar component;
// onValueChanged,  onValueChanged,   onUnitIncrement;
// onUnitDecrement, onBlockIncrement, onBlockDecrement, onTrack;;
public  class TAdjustmentEvent extends MethodVar {
 //Regis Invoke method
  static {
     MethodVar.regisType(new TAdjustmentEvent());
  }
 //override getTypeNames for regis mutiple type invoke method
  public String[] getTypeNames(){
     String tname[] = new String[2];
     tname[0]="invoke1";
     tname[1]="invoke2";
     return tname;
  }
  public void invoke(AdjustmentEvent e){
     if(getInvokeName().compareTo("invoke1")==0){
         invoke1(e.getValue(),e.getAdjustmentType(),e.getAdjustable());
     }
     else
     if(getInvokeName().compareTo("invoke2")==0){
         invoke2(e.getAdjustable(),e.getValue(),e.getAdjustmentType());
     }
     else{
       Object arg[] = getArguments();
       arg[0] = e;
       super.vinvoke(arg);
     }
     if(getMethodMode()!=SINGLE && next != null){

   ((TAdjustmentEvent)next).invoke(e);

     }
  }
  public void invoke1(int value,int type,Adjustable sender){
       Object arg[] = getArguments();
       arg[0] = new Integer(value);
       arg[1] = new Integer(type);
       arg[2] = sender;
       super.vinvoke(arg);
  }
  public void invoke2(Adjustable sender,int value,int type){
       Object arg[] = getArguments();
       arg[0] = sender;
       arg[1] = new Integer(value);
       arg[2] = new Integer(type);
       super.vinvoke(arg);
  }
} //end of TAdjustmentEvent
```

3. ตัวอย่างโค้ดโปรแกรมของคอมโพเนนท์ที่เขียนด้วยวิธีตัวแปรเมทอด

TButton ใช้แทน java.awt.Button โดยการเพิ่มเติมการจัดการเหตุการณ์ด้วยตัวแปรเมทอด

```
package com.wtv.awt;

import java.awt.*;
import java.awt.event.*;
import com.wtv.*;
public  class TButton extends Button
  implements TMouseEnable,TComponentEnable,TKeyEnable,TActionEnable {
  public TButton (){
     this("");
   }
  public TButton (String label){
    super(label);
    new TComponentAdapter(this);
    new TMouseAdapter(this);
    new TKeyAdapter(this);
    new TActionAdapter(this);
  }
 //Set Color Method
  public void setColorRed(int v){
    Color c =getBackground();
    setBackground(new Color(v,c.getGreen(),c.getBlue()));
  }
  public void setColorGreen(int v){
    Color c =getBackground();
    setBackground(new Color(c.getRed(),v,c.getBlue()));
  }
  public void setColorBlue(int v){
    Color c =getBackground();
    setBackground(new Color(c.getRed(),c.getGreen(),v));
  }

 //** Event Define **
  TNotifyEvent onAction;
  TMouseEvent  onMouseClicked;
  TMouseEvent  onMouseDblClicked;
  TMouseEvent  onMouseEntered;
  TMouseEvent  onMouseExited;
  TMouseEvent  onMousePressed;
  TMouseEvent  onMouseReleased;
  TMouseEvent  onMouseDrag;
  TMouseEvent  onMouseMove;
  TNotifyEvent onGotFocus;
  TNotifyEvent onLostFocus;
  TNotifyEvent onShow;
  TNotifyEvent onHide;
  TNotifyEvent onMoved;
  TNotifyEvent onResized;
  TKeyEvent    onKeyUp;
  TKeyEvent    onKeyDown;
  TKeyEvent    onKeyTyped;
 //** End Event Define **

//** <Event Set&Get Method>
  public TNotifyEvent getOnAction(){return onAction;};
  public boolean setOnAction(Object obj,String methodname){
    if(obj==null || methodname==""){
      onAction=null;
      return true;
    }
    if(onAction==null)onAction =new TNotifyEvent();
```

```
    return onAction.set(obj,methodname);
   }
  public TMouseEvent getOnMouseClicked(){return onMouseClicked;};
  public boolean setOnMouseClicked(Object obj,String methodname){
    if(obj==null || methodname==""){
      onMouseClicked=null;
      return true;
    }
    if(onMouseClicked==null)onMouseClicked =new TMouseEvent();
    return onMouseClicked.set(obj,methodname);
   }

  public TMouseEvent getOnMouseDblClicked(){return onMouseDblClicked;};
  public boolean setOnMouseDblClicked(Object obj,String methodname){
    if(obj==null || methodname==""){
      onMouseDblClicked=null;
      return true;
    }
    if(onMouseDblClicked==null)onMouseDblClicked =new TMouseEvent();
    return onMouseDblClicked.set(obj,methodname);
   }

  public TMouseEvent getOnMouseEntered(){return onMouseEntered;};
  public boolean setOnMouseEntered(Object obj,String methodname){
    if(obj==null || methodname==""){
      onMouseEntered=null;
      return true;
    }
    if(onMouseEntered==null)onMouseEntered =new TMouseEvent();
    return onMouseEntered.set(obj,methodname);
   }

  public TMouseEvent getOnMouseExited(){return onMouseExited;};
  public boolean setOnMouseExited(Object obj,String methodname){
    if(obj==null || methodname==""){
      onMouseExited=null;
      return true;
    }
    if(onMouseExited==null)onMouseExited =new TMouseEvent();
    return onMouseExited.set(obj,methodname);
   }

  public TMouseEvent getOnMousePressed(){return onMousePressed;};
  public boolean setOnMousePressed(Object obj,String methodname){
    if(obj==null || methodname==""){
      onMousePressed=null;
      return true;
    }
    if(onMousePressed==null)onMousePressed =new TMouseEvent();
    return onMousePressed.set(obj,methodname);
   }

  public TMouseEvent getOnMouseReleased(){return onMouseReleased;};
  public boolean setOnMouseReleased(Object obj,String methodname){
    if(obj==null || methodname==""){
      onMouseReleased=null;
      return true;
    }
    if(onMouseReleased==null)onMouseReleased =new TMouseEvent();
    return onMouseReleased.set(obj,methodname);
   }

  public TMouseEvent getOnMouseDrag(){return onMouseDrag;};
  public boolean setOnMouseDrag(Object obj,String methodname){
    if(obj==null || methodname==""){
      onMouseDrag=null;
      return true;
```

```
    }
    if(onMouseDrag==null)onMouseDrag=new TMouseEvent();
    return onMouseDrag.set(obj,methodname);
   }

  public TMouseEvent getOnMouseMove(){return onMouseMove;};
  public boolean setOnMouseMove(Object obj,String methodname){
    if(obj==null || methodname==""){
      onMouseMove=null;
      return true;
    }
    if(onMouseMove==null)onMouseMove =new TMouseEvent();
    return onMouseMove.set(obj,methodname);
   }

  public TNotifyEvent getOnGotFocus(){return onGotFocus;};
  public boolean setOnGotFocus(Object obj,String methodname){
   if(obj==null || methodname==""){
      onGotFocus=null;
      return true;
    }
    if(onGotFocus==null)onGotFocus =new TNotifyEvent();
    return onGotFocus.set(obj,methodname);
   }

  public TNotifyEvent getOnLostFocus(){return onLostFocus;};
  public boolean setOnLostFocus(Object obj,String methodname){
    if(obj==null || methodname==""){
      onLostFocus=null;
      return true;
    }
    if(onLostFocus==null)onLostFocus =new TNotifyEvent();
    return onLostFocus.set(obj,methodname);
   }

  public TNotifyEvent getOnShow(){return onShow;};
  public boolean setOnShow(Object obj,String methodname){
    if(obj==null || methodname==""){
      onShow=null;
      return true;
    }
    if(onShow==null)onShow =new TNotifyEvent();
    return onShow.set(obj,methodname);
   }

  public TNotifyEvent getOnHide(){return onHide;};
  public boolean setOnHide(Object obj,String methodname){
    if(obj==null || methodname==""){
      onHide=null;
      return true;
    }
    if(onHide==null)onHide =new TNotifyEvent();
    return onHide.set(obj,methodname);
   }

  public TNotifyEvent getOnMoved(){return onMoved;};
  public boolean setOnMoved(Object obj,String methodname){
    if(obj==null || methodname==""){
      onMoved=null;
      return true;
    }
    if(onMoved==null)onMoved =new TNotifyEvent();
    return onMoved.set(obj,methodname);
   }
  public TNotifyEvent getOnResized(){return onResized;};
  public boolean setOnResized(Object obj,String methodname){
    if(obj==null || methodname==""){
```

```
      onResized=null;
      return true;
    }
    if(onResized==null)onResized =new TNotifyEvent();
    return onResized.set(obj,methodname);
   }

  public TKeyEvent getOnKeyUp(){return onKeyUp;};
  public boolean setOnKeyUp(Object obj,String methodname){
    if(obj==null || methodname==""){
      onKeyUp=null;
      return true;
    }
    if(onKeyUp==null)onKeyUp =new TKeyEvent();
    return onKeyUp.set(obj,methodname);
   }

  public TKeyEvent getOnKeyDown(){return onKeyDown;};
  public boolean setOnKeyDown(Object obj,String methodname){
    if(obj==null || methodname==""){
      onKeyDown=null;
      return true;
    }
    if(onKeyDown==null)onKeyDown =new TKeyEvent();
    return onKeyDown.set(obj,methodname);
   }

  public TKeyEvent getOnKeyTyped(){return onKeyTyped;};
  public boolean setOnKeyTyped(Object obj,String methodname){
    if(obj==null || methodname==""){
      onKeyTyped=null;
      return true;
    }
    if(onKeyTyped==null)onKeyTyped =new TKeyEvent();
    return onKeyTyped.set(obj,methodname);
   }
//**<End Event Set&Get Method>
} //End of TButton.java
```

สำหรับคอมโพเนนท์อื่นๆ ก็จะมีรูปแบบการเขียนที่คล้ายคลึงกัน

ประวัติผู้เขียนวิทยานิพนธ์

นายวชิราวุธ ธรรมวิเศษ เกิดวันที่ 6 พฤษภาคม พ.ศ. 2518 ที่จังหวัดขอนแก่น สำเร็จการศึกษาระดับมัธยมศึกษาตอนปลายจาก โรงเรียนชุมแพศึกษา เมื่อปีการศึกษา 2535 และสำเร็จการศึกษาปริญญาวิทยาศาสตร์บัณฑิต สาขาวิทยาการคอมพิวเตอร์ จากมหาวิทยาลัยขอนแก่น เมื่อปีการศึกษา 2539 จากนั้นเข้าศึกษาต่อในหลักสูตรวิทยาศาสตร์มหาบัณฑิต สาขาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2541