การศึกษาเชิงเปรียบเทียบของการพัฒนาโปรแกรม แบบเชิงอ็อบเจกต์ คอมโพเนนท์ และ แอสเปค

A COMPARATIVE STUDY ON OBJECT COMPONENT AND ASPECT ORIENTED PROGRAMMING

มนูญ เปรมรัตนวงศ์
สุพจน์ บุญชำนาญ

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต
ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2548

A COMPARATIVE STUDY ON OBJECT COMPONENT AND ASPECT ORIENTED PROGRAMMING

MANOON PREMRATTANAWONG

SUPHOT BOONCHAMNAN

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2005

ปัญหาพิเศษ	การศึกษาเชิงเปรียบเทียบของการพัฒนาโปรแกรมแบบเชิงอ็อบเจกต์คอมโพเนนท์และแอสเปค A COMPARATIVE STUDY ON OBJECT COMPONENT AND ASPECT ORIENTED PROGRAMMING
ชื่อนักศึกษา	นายมนูญ เปรมรัตนวงศ์ 45050505 นายสุพจน์ บุญชำนาญ 45050538
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สาขาวิชา	วิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษา	ผศ.ดร. ศรัณย์ อินทโกสุม

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2548

คณะกรรมการสอบ		ลายมือชื่อ
ประธานกรรมการ	อ.สันธนะ อู่อุดมยิ่ง	
กรรมการ	อ.ธีระ ฟักอ่อน	
กรรมการและอาจารย์ที่ปรึกษา	ผศ.ดร.ศรัณย์ อินทโกสุม	

(รองศาสตราจารย์ ดร.วีระ บุญจริง)
หัวหน้าภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

ลิขสิทธิ์ของภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

กิตติกรรมประกาศ

การทำปัญหาพิเศษหัวข้อ การศึกษาเชิงเปรียบเทียบของการพัฒนาโปรแกรมแบบเชิงอ็อบเจกต์ คอมโพเนนท์ และ แอสเปค สามารถสำเร็จลุล่วงไปด้วยดี คณะผู้จัดทำต้องขอขอบพระคุณบุคคลดังต่อไปนี้

1. บิดา มารดา ผู้ซึ่งมีพระคุณอย่างมากที่ได้ให้กำเนิดเลี้ยงดู อบรม ส่งเสริมให้ได้รับและกระทำในสิ่งที่ดีมาโดยตลอด รวมทั้งเป็นกำลังใจในทุกๆ เรื่องเสมอมา

2. ผู้ช่วยศาสตราจารย์ ดร.ศรัณย์ อินทโกสุม ซึ่งเป็นอาจารย์ที่ปรึกษาปัญหาพิเศษที่กรุณาให้คำปรึกษาและแนะนำในด้านการศึกษาปัญหา การให้แนวคิดและแนวทางการแก้ปัญหา รวมถึงการตรวจสอบและแก้ไขการเขียนรายงานปัญหาพิเศษเล่มนี้

3. อาจารย์สันธนะ อู่อุดมยิ่งและอาจารย์ธีระ ฟักอ่อนซึ่งเป็นกรรมการสอบปัญหาพิเศษที่กรุณาให้คำแนะนำอย่างดีในการแก้ไขปัญหาพิเศษ

4. อาจารย์ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ทุกๆท่าน ที่ได้ประสาทวิชาความรู้ทั้งในภาคทฤษฎีและภาคปฏิบัติแก่ผู้จัดทำตลอดเวลาทั้ง 4 ปี จนกระทั่งปัญหาพิเศษสัมฤทธิ์ผลได้ด้วยดีทุกประการ

5. คุณชุติกาญจน์ ตันยะสิทธิ์ เจ้าหน้าที่ฝ่ายธุรการประจำภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์ที่กรุณาให้ข้อมูลและคำแนะนำต่างๆที่มีความจำเป็นในการทำปัญหาพิเศษ

6. เพื่อนๆ ทุกคน ที่คอยให้คำแนะนำและกำลังใจมาตลอด

ผู้จัดทำ

มีนาคม 2549

หัวข้อปัญหาพิเศษ	การศึกษาเชิงเปรียบเทียบของการพัฒนาโปรแกรมแบบเชิงอ็อบเจกต์ คอมโพเนนท์และแอสเปค
ชื่อนักศึกษา	นายมนูญ เปรมรัตนวงศ์ 45050505
	นายสุพจน์ บุญชำนาญ 45050538
ปริญญา	วิทยาศาสตรบัณฑิต
ภาควิชา	คณิตศาสตร์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์
สาขาวิชา	วิทยาการคอมพิวเตอร์
ปีการศึกษา	2548
อาจารย์ที่ปรึกษา	ผศ.ดร.ศรัณย์ อินทโกสุม

บทคัดย่อ

การโปรแกรมเชิงแอสเปค เป็นวิธีการพัฒนาโปรแกรมแนวใหม่ที่ได้รับการคาดหมายว่าจะช่วยให้การโปรแกรมเชิงอ็อบเจกต์ และการโปรแกรมเชิงคอมโพเนนท์มีประสิทธิภาพมากขึ้น การโปรแกรมเชิงแอสเปคมีจุดมุ่งหมายหลักคือการทำให้การพัฒนาโปรแกรมมีความเป็นโมดูลมากขึ้น ด้วยการจัดเตรียมกลไกในการควบคุมและจัดการโค้ดที่กระจายในระบบ มาไว้ในโมดูลเดียวกันที่เรียกว่าแอสเปค อย่างไรก็ตามการโปรแกรมเชิงแอสเปคสำหรับประเทศไทยแล้ว จัดว่าเป็นเรื่องที่ใหม่อยู่มาก ดังนั้นงานวิจัยนี้จึงมีจุดประสงค์เพื่อแสดงให้เห็นอย่างเป็นรูปธรรมว่าการโปรแกรมเชิงแอสเปคคืออะไร และจะแก้ปัญหาของการโปรแกรมเชิงอ็อบเจกต์และเชิงคอมโพเนนท์ได้อย่างไร การวิจัยนี้เริ่มต้นจากการพัฒนาบางส่วนของระบบงานประกันภัยโดยใช้หลักการโปรแกรมเชิงอ็อบเจกต์ จากนั้นจึงพัฒนาระบบเดียวกันโดยใช้การโปรแกรมเชิงคอมโพเนนท์ และแสดงให้เห็นว่าทั้งการโปรแกรมเชิงออบเจกต์และเชิงคอมโพเนนท์ยังไม่สามารถแก้ปัญหาของการที่มีการกระจายของโค้ดในระบบได้ จากนั้นจึงแสดงให้เห็นว่าการโปรแกรมเชิงแอสเปคจะช่วยแก้ปัญหาได้อย่างไร

Special Project Title A COMPARATIVE STUDY ON OBJECT COMPONENT AND ASPECT ORIENTED PROGRAMMING

Students Mr. Manoon Premrattanawong 45050505

Mr. Suphot Boonchamnan 45050538

Degree Bachelor of Science

Department Mathematics and Computer Science,Faculty of Science

Programme Computer Science

Academic Year 2005

Special Project Advisor Asst.Prof. Dr.Sarun Intakosum

ABSTRACT

Aspect Oriented Programming technique is a modern program development method that can be considered as promising solution to make object-oriented and component-oriented programming techniques more efficient. The major purpose of aspect-oriented programming technique is to make a program to be more modular, by providing the mechanism to control and manage codes those are scattered through out the system into modules, called aspects. However, the concept of aspect-oriented programming is relatively new in Thailand. Therefore, the purpose of this research is to give the concrete idea on what aspect-oriented programming is, and how it can fulfill the problems of object-oriented and component-oriented programming. The steps in conducting this research are as follows; firstly object-oriented programming technique is used to implement parts of an insurance system. Secondly the component-oriented programming technique is used to develop the same system. Thirdly, show that both techniques are still suffered from the problem of code scattering. Finally, the aspect-oriented programming is applied to solve the problem.

สารบัญ

หน้า

บทคัดย่อปัญหาพิเศษภาษาไทย.....I

บทคัดย่อปัญหาพิเศษภาษาอังกฤษ.....II

กิตติกรรมประกาศ.....III

สารบัญ.....IV

สารบัญรูป.....VI

บทที่ 1 บทนำ.....1

1.1 ความสำคัญและที่มาของปัญหา.....1

1.2 วัตถุประสงค์.....1

1.3 ขอบเขตของปัญหา.....2

1.4 ประโยชน์ที่คาดว่าจะได้รับ.....2

1.5 ขั้นตอนในการดำเนินการ.....2

1.6 อุปกรณ์ที่ใช้ในการทำปัญหาพิเศษ.....2

บทที่ 2 ทฤษฎีและวรรณกรรมที่เกี่ยวข้อง.....4

2.1 การพัฒนาโปรแกรมเชิงอ็อบเจกต์.....4

2.1.1 ยูเอ็มแอลไดอะแกรม.....5

2.1.2 ปัญหาของการพัฒนาโปรแกรมเชิงอ็อบเจกต์.....11

2.2 การพัฒนาโปรแกรมเชิงคอมโพเนนท์.....14

2.2.1 แบบจำลองของคอมโพเนนท์.....14

2.2.2 ดีพลอยเมนท์.....15

2.2.3 การนำคอมโพเนนท์ไปใช้งาน.....15

2.2.4 ความแตกต่างระหว่างการพัฒนาโปรแกรมเชิงอ็อบเจกต์กับคอมโพเนนท์.....15

2.3 การพัฒนาโปรแกรมเพื่อแก้ไขปัญหาครอสคัตติ้งคอนเซิน.....16

2.3.1 การพัฒนาโปรแกรมแบบเอ็กพลิซิต.....16

2.4 การพัฒนาโปรแกรมเชิงแอสเปค.....17

2.4.1 แอสเปคเจ.....18

2.4.2 กลไกการทำงานของแอสเปคเจ.....18

สารบัญ (ต่อ)

2.4.3 รูปแบบภาษาของแอสเปคเจ.....19
2.4.4 ตัวอย่างโปรแกรมที่พัฒนาด้วยแอสเปคเจ.....24
บทที่ 3 การทดสอบและเปรียบเทียบการพัฒนาโปรแกรม.....26
3.1 การพัฒนาโปรแกรมโดยวิธีการเชิงอ็อบเจกต์.....29
3.2 การพัฒนาโปรแกรมโดยวิธีการเชิงคอมโพเนนท์.....38
3.3 การพัฒนาโปรแกรมโดยวิธีการเชิงแอสเปค.....39
บทที่ 4 สรุปผลการวิจัยและข้อเสนอแนะ.....43
4.1 ผลการวิจัยและพัฒนา.....43
4.2 ข้อเสนอแนะ.....44

บรรณานุกรม.....45
ภาคผนวก ก. การติดตั้งโปรแกรมแอสเปคเดฟซุดเซ็ตอัพ1.4.....46
ภาคผนวก ข. รายละเอียดคลาสในระบบประกันภัย.....56

สารบัญรูป

รูปที่ หน้า

2.1 แสดงการติดต่อกันของอ็อบเจกต์ โดยใช้การส่งเมสเสจ....4

2.2 แสดงตัวอย่างของคลาส....6

2.3 ตัวอย่างแสดงความสัมพันธ์กันระหว่างคลาส....7

2.4 ตัวอย่างคลาสที่เกิดจากความสัมพันธ์กันของคลาส....7

2.5 ตัวอย่างคลาสที่มีการการสืบทอดคุณสมบัติ....8

2.6 ตัวอย่างคลาสแอกกริเกชัน....8

2.7 ตัวอย่างคลาสคอมโพซิชัน....9

2.8 ตัวอย่างซีเควนซ์ไดอะแกรม....10

2.9 แสดงตัวอย่างคอมโพเนนท์ไดอะแกรม....10

2.10 แสดงตัวอย่างดีพลอยเมนท์ไดอะแกรม....11

2.11 แสดงปัญหาของครอสคัตติ้งคอนเซิน....13

2.12 แสดงแผนภาพคอมโพเนนท์....14

2.13 แสดงปัญหาครอสคัตติ้งคอนเซิน....16

2.14 แสดงการพัฒนาโปรแกรมแบบเอ็กพลิซิต....17

2.15 แสดงถึงกลไกการทำงานของแอสเปคเจ....18

2.16 แสดงซีเควนซ์ไดอะแกรมที่แสดงให้เห็นถึงจุดต่างๆ ของจอยพอยท์....20

2.17 แสดงตัวอย่างโปรแกรมแอสเปคเจ....25

3.1 แสดงแพคเกจและความสันพันธ์กันระหว่างแพคเกจ....27

3.2 แสดงคลาสไดอะแกรมของระบบประกัน....28

3.3 แสดงโค้ดที่ทำให้คลาสขาดความเป็นอิสระที่เกิดขึ้นในคลาส NewCustomer....29

3.4 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส Policy....30

3.5 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส LifePolicy....31

3.6 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส HousePolicy....31

3.7 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส AutoPolicy....32

3.8 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส ComputerPolicy....33

สารบัญรูป (ต่อ)

รูปที่ หน้า

3.9 แสดงการแก้ไขโค้ดเมื่อเมท็อด notifyListeners เกิดการเปลี่ยนแปลงที่เกิดขึ้น.....33
ภายในคลาส ComputerPolicy

3.10 แสดงการกระจายของโค้ดที่เกิดขึ้นในเมท็อด addCustomer ของคลาส NewCustomer.....34

3.11 แสดงการกระจายของโค้ดที่เกิดขึ้นในเมท็อด searchCustomer ของคลาส.....34
FindAndEditCustomer

3.12 แสดงการกระจายของโค้ดที่เกิดขึ้นในเมท็อด updateCustomer ของคลาส 35
FindAndEditCustomer

3.13 แสดงการกระจายของโค้ดที่เกิดขึ้นในเมท็อด deleteCustomer ของคลาส.....35
FindAndEditCustomer

3.14 แสดงการจัดการกับข้อผิดพลาดของโปรแกรมในเมท็อด addCustomer ของคลาส 36
NewCustomer

3.15 แสดงการจัดการกับข้อผิดพลาดของโปรแกรมในเมท็อด updateCustomer ของคลาส.....37
FindAndEditCustomer

3.16 แสดงการจัดการกับข้อผิดพลาดของโปรแกรมในเมท็อด addHousePolicy ของคลาส.....37
AddHousePolicy

3.17 แสดงการจัดการกับข้อผิดพลาดของโปรแกรมในเมท็อด addClaim ของคลาส AddClaim.....38

3.18 แสดงการส่งเหตุการณ์ออกไปให้กับคอมโพเนนท์อื่นโดยใช้เมท็อด.....39
firePropertyChange ในคลาส NewCustomer

3.19 แสดงส่วนหนึ่งของโค้ดของแอสเปคเจที่แก้ปัญหาที่เกิดจากการสืบทอดคลาส.....40

3.20 แสดงโค้ดของแอสเปคเจเมื่อเมท็อด notifyListeners เกิดการเปลี่ยนแปลง.....41

3.21 แสดงส่วนหนึ่งของโค้ดของแอสเปคเจที่แก้ปัญหาการติดต่อกับฐานข้อมูล.....41

3.22 แสดงโค้ดของแอสเปคเจที่แก้ปัญหาการจัดการกับข้อผิดพลาดที่เกิดขึ้นในโปรแกรม.....42

ก-1 แสดงหน้าต่างแรกสำหรับการติดตั้งเจดีเค 1.4.2_08.....47

ก-2 แสดงหน้าต่างสำหรับเลือกฟีเจอร์ที่ต้องการติดตั้ง 47

ก-3 แสดงหน้าต่างสำหรับระบุไดเรคทอรีที่ต้องการติดตั้งโปรแกรม.....48

ก-4 แสดงหน้าต่างสำหรับเลือกรีจิตเตอร์ปลั๊กอินของจาวาลงในเบราเซอร์.....48

ก-5 หน้าต่างแสดงการติดตั้งโปรแกรม.....49
ก-6 หน้าต่างแสดงการติดตั้งโปรแกรม.....49
ก-7 แสดงหน้าต่างแรกสำหรับการติดตั้งแอสเปคเคฟชุดเซ็ตอัพ1.4 50
ก-8 แสดงหน้าต่างสำหรับระบุไดเรคทอรีที่ต้องการติดตั้งโปรแกรม.....50
ก-9 แสดงหน้าต่างแสดงไดเรคทอรีที่จะทำการติดตั้งโปรแกรม.....51
ก-10 หน้าต่างแสดงการติดตั้งโปรแกรม.....51
ก-11 หน้าต่างแสดงการเสร็จสิ้นการติดตั้งโปรแกรม.....52
ก-12 หน้าต่างแสดงการติดตั้งเอเจดีเค.....52
ก-13 แสดงหน้าต่างระบุไดเรคทอรีเจดีเค.....53
ก-14 หน้าต่างแสดงการติดตตั้งโปรแกรม.....53
ก-15 หน้าต่างแสดงการติดตั้งโปรแกรมเสร็จแล้ว.....54
ก-16 หน้าต่างแสดงการเสร็จสิ้นการติดตั้งเอเจดีเค.....54

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหา

ในปัจจุบัน ประสิทธิภาพของเครื่องคอมพิวเตอร์ได้พัฒนาสูงขึ้นมากจนสามารถรองรับการทำงานของการพัฒนาโปรแกรมเชิงอ็อบเจกต์และคอมโพเนนท์ได้อย่างมีประสิทธิภาพ จึงทำให้การพัฒนาโปรแกรมทั้ง 2 แบบ ได้รับการยอมรับและเริ่มมีการนำมาใช้ในระบบงานจริงมากขึ้น แต่อย่างไรก็ตาม การพัฒนาโปรแกรมโดยใช้วิธีการทั้ง 2 นี้ยังไม่เหมาะสมกับปัญหาที่มีความซับซ้อนบางประการ เช่น ระบบธนาคารจะมีทรานแซคชั่น (Transaction) ที่มีความสัมพันธ์กันเป็นปริมาณมาก การพัฒนาโปรแกรมโดยใช้วิธีการทั้ง 2 จะมีความยุ่งยากในการพัฒนาระบบ คือ ทุกเหตุการณ์ที่ต้องมีกิจกรรมเกี่ยวกับทรานแซคชั่นเข้ามาเกี่ยวข้อง ไม่ว่าจะเป็นการเพิ่มข้อมูล การเปลี่ยนแปลงข้อมูล การลบข้อมูล และการค้นหาข้อมูล จะต้องมีโค้ด (Code) ที่คอยจัดการกับกิจกรรมเหล่านั้นอยู่ในทุกๆเหตุการณ์ของโปรแกรม ซึ่งจะทำให้โค้ดส่วนดังกล่าวกระจายไปทั่วโปรแกรม ซึ่งจะทำให้การดูแลรักษาโปรแกรมทำได้ไม่สะดวกนัก และโปรแกรมไม่มีความเป็นโมดูล (Module) ถ้าระบบยิ่งมีความซับซ้อนมากการพัฒนาก็จะยิ่งทำได้ยากตามไปด้วย จากปัญหาดังกล่าว จึงมีผู้คิดค้นแนวคิดของการพัฒนาโปรแกรมเชิงแอสเปคขึ้นเพื่อช่วยแก้ไขปัญหาดังกล่าว โดยใช้การแยกส่วนของโค้ดที่มีหน้าที่เหมือนกันที่กระจายอยู่ในหลายๆคลาสให้มารวมอยู่ที่โมดูลเดียว

แต่เนื่องจากการเขียนโปรแกรมเชิงแอสเปคนี้ยังเป็นเรื่องที่ใหม่อยู่มากในประเทศไทย จึงทำให้นักพัฒนาโปรแกรมอาจจะยังมองไม่เห็นถึงแนวคิด และลักษณะของพัฒนาโปรแกรมเชิงแอสเปค รวมไปถึงวิธีการแก้ปัญหาของการพัฒนาโปรแกรมเชิงแอสเปค ด้วยเหตุนี้ผู้จัดทำจึงได้จัดทำปัญหาพิเศษนี้เพื่อมุ่งที่จะตอบคำถามดังกล่าว โดยทำการศึกษาเปรียบเทียบถึงคุณลักษณะที่เหมือนกันและแตกต่างกันของการพัฒนาโปรแกรมเชิงอ็อบเจกต์ คอมโพเนนท์และแอสเปค จากนั้นแสดงการเปรียบเทียบโดยพัฒนาระบบที่ด้วยวิธีการทั้ง 3 และแสดงให้เห็นถึงปัญหาของการพัฒนาโปรแกรมเชิงอ็อบเจกต์ และวิธีการแก้ปัญหาโดยการพัฒนาโปรแกรมเชิงคอมโพเนนท์และแอสเปค

1.2 วัตถุประสงค์

ศึกษาเปรียบเทียบแนวคิดของการพัฒนาโปรแกรมเชิงอ็อบเจกต์ คอมโพเนนท์ และแอสเปค เพื่อแสดงให้เห็นถึงวิธีการพัฒนาโปรแกรม ข้อแตกต่างของการพัฒนาโปรแกรม ปัญหา และวิธีการแก้ปัญหาของการพัฒนาโปรแกรม

1.3 ขอบเขตของปัญหา

นำเสนอหลักการพัฒนาโปรแกรมเชิงแอสเปค ในส่วนของเนื้อหาที่นำมาช่วยลดความซับซ้อนของ การพัฒนาโปรแกรมเชิงอ็อบเจกต์ และ คอมโพเนนท์ โดยนำเอาหลักการพัฒนาโปรแกรมเชิงแอสเปคเข้ามาใช้ทำให้การพัฒนาโปรแกรมเพื่อแก้ปัญหาที่เกิดจาความซับซ้อนของโปรแกรม โดยที่จะพัฒนาระบบ 1 ระบบคือระบบประกันภัย และพัฒนาระบบด้วยวิธีการทั้ง 3 จากนั้นทำการทดสอบเปรียบเทียบเพื่อแสดงให้เห็นถึงความแตกต่างของการพัฒนาโปรแกรมด้วยวิธีทั้ง 3 ในส่วนของภาษาที่ใช้ในการพัฒนาจะใช้ภาษาจาวา (Java) ในการพัฒนาโปรแกรมเชิงอ็อบเจกต์ และการพัฒนาโปรแกรมเชิงคอมโพเนนท์ และใช้แอสเปคเจ (AspectJ) ในการพัฒนาโปรแกรมเชิงแอสเปค

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1. แสดงเห็นถึงความแตกต่างของการพัฒนาโปรแกรมโดยใช้ เทคนิคการเขียนโปรแกรมเชิงอ็อบเจกต์ คอมโพเนนท์ และ แอสเปค
2. แสดงให้เห็นถึงแนวทางในการนำการพัฒนาโปรแกรมเชิงคอมโพเนนท์และแอสเปคเข้ามาช่วยในการแก้ปัญหา

1.5 ขั้นตอนในการดำเนินการ

1. ศึกษาหลักการพัฒนาโปรแกรมเชิงอ็อบเจกต์และเชิงคอมโพเนนท์
2. ศึกษาหลักการพัฒนาโปรแกรมเชิงแอสเปค ในส่วนที่จะนำมาแก้ปัญหาที่มีความซับซ้อนยุ่งยากของการพัฒนาระบบในข้อ 1
3. กำหนดขอบเขตที่จะทำการทดสอบเปรียบเทียบ
4. เลือกระบบที่มีการทำงานอยู่ภายใต้ขอบเขตที่จะทำการทดสอบเปรียบเทียบในข้อ 3
5. พัฒนาระบบดังกล่าวโดยใช้ การพัฒนาโปรแกรมทั้ง 3 วิธี
6. เปรียบเทียบแนวทางในการพัฒนาโปรแกรมทั้ง 3 วิธี
7. จัดทำรายงาน

1.6 อุปกรณ์ที่ใช้ในการทำปัญหาพิเศษ

1. รายละเอียดทางด้านอุปกรณ์คอมพิวเตอร์
    1.1 ฮาร์ดดิสก์ ขนาด 20 กิกะไบต์
    1.2 แรม 256 เมกกะไบต์
    1.3 ซีพียู ความเร็ว 1.3 กิกะเฮิรตซ์ขึ้นไป
2. รายละเอียดทางด้านโปรแกรม
    2.1 แอสเปคเดฟสูทเซ็ตอัพ1.4 (AspectJDevSuiteSetup1.4)
    2.2 เจดีเค 1.4 (Java Development Kit Version 1.4 : JDK 1.4)
    2.3 บีดีเค 1.1 (Bean Developer Kit 1.1 : BDK 1.1)

2.4 เอเจดีที 1.5 (AspectJ Development Tool 1.5 : AJDT 1.5)

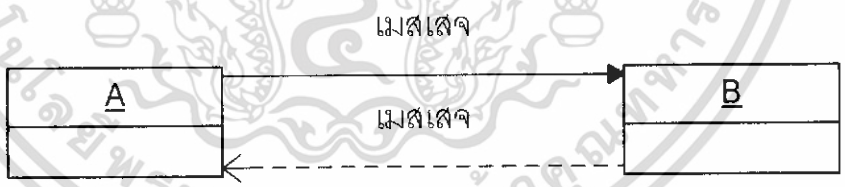
2.5 ไมโครซอฟต์แอกเซส (Microsoft Access)

บทที่ 2
ทฤษฎีและวรรณกรรมที่เกี่ยวข้อง

2.1 การพัฒนาโปรแกรมเชิงอ็อบเจกต์ (Object Oriented Programming : OOP)

อ็อบเจกต์ (Object) หรืออาจจะเรียกอีกอย่างว่า "วัตถุ" คือ ข้อมูลรูปแบบหนึ่งซึ่งมีองค์ประกอบ 2 อย่าง ได้แก่ แอททริบิวต์ (Attribute) ซึ่งเป็นสิ่งที่แสดงถึงความเป็นอ็อบเจกต์นั้นๆ และ เมท็อด (Method) ซึ่งเป็นพฤติกรรมที่อ็อบเจกต์นั้นสามารถทำได้ เช่น อ็อบเจกต์นักศึกษา จะมีคุณสมบัติที่แสดงความเป็นนักศึกษาดังต่อไปนี้ รหัสนักศึกษา ชื่อ นามสกุล ที่อยู่ เกรดเฉลี่ย เป็นต้น และมีพฤติกรรมดังต่อไปนี้ เดิน วิ่ง กิน เรียน สอบ อ่านหนังสือ เป็นต้น

การพัฒนาโปรแกรมเชิงอ็อบเจกต์ มีแนวคิดหลักคือมองการพัฒนาระบบ เหมือนกับการมองโลกด้วยความเป็นจริง คือมองสิ่งต่างๆ เป็นอ็อบเจกต์ ซึ่งเป็นหน่วยที่เล็กที่สุดของหลักการพัฒนาโปรแกรมเชิงอ็อบเจกต์ ดังนั้นการพัฒนาโปรแกรมจะประกอบขึ้นจากอ็อบเจกต์ หลายๆอ็อบเจกต์ และรวมกันขึ้นเป็นแอพพลิเคชัน (Application) เช่น ระบบการเช่าหนังสือ ประกอบไปด้วยอ็อบเจกต์ของคนเช่าหนังสือ อ็อบเจกต์หนังสือ และอ็อบเจกต์การเช่าหนังสือประกอบกันเป็นแอพพลิเคชัน โดยที่แต่ละอ็อบเจกต์ สามารถติดต่อสื่อสารกันได้โดยใช้วิธีการส่งเมสเสจ (Message Passing) ดังรูป 2.1



รูป 2.1 แสดงการติดต่อกันของอ็อบเจกต์ โดยใช้การส่งเมสเสจ

จากรูป 2.1 มีการทำงานโดยอ็อบเจกต์ A ทำการส่งเมสเสจไปให้กับอ็อบเจกต์ B เมื่ออ็อบเจกต์ B ได้รับเมสเสจแล้วจะทำการเรียกเมท็อดขึ้นมาทำงาน เมื่อทำงานเสร็จแล้วอ็อบเจกต์ B จะส่งเมสเสจกลับไปให้อ็อบเจกต์ A หรือไม่ส่งก็ได้

แนวคิดหลักในการพัฒนาโปรแกรมเชิงอ็อบเจกต์ มีดังต่อไปนี้

- คลาส (Class) และ อินสแตนซ์ (Instance)

คลาสเป็นแม่แบบที่เกิดจากการกำหนดลักษณะของอ็อบเจกต์ โดยคลาสนั้นเป็นเพียงแม่แบบ ไม่สามารถใช้ในการดำเนินกิจกรรมได้ ดังนั้นเราจึงจำเป็นต้องสร้างตัวดำเนินการขึ้นมาเพื่อใช้ในการดำเนินกิจกรรม โดยเรียกว่าอินสแตนซ์ ซึ่งอินสแตนซ์ก็คืออ็อบเจกต์นั่นเอง

- การห่อหุ้มข้อมูล (Encapsulation)

เป็นการนำแอททริบิวต์ และ เมท็อดมารวมกันเป็นโครงสร้างเพียงหน่วยเดียว ซึ่งเมื่อสร้างโครงสร้างนี้ขึ้นมาใช้งาน ก็จะทำให้มีแอททริบิวต์และเมท็อดสำหรับการทำงานต่างๆ

- การซ่อนข้อมูล (Information Hiding)

เป็นความสามารถที่จะจำกัดการมองเห็นข้อมูลที่อยู่ภายในอ็อบเจกต์ เพื่อไม่ให้อ็อบเจกต์อื่นสามารถเรียกใช้งานข้อมูลเหล่านั้นได้โดยตรง ถ้าอ็อบเจกต์หนึ่งต้องการใช้งานข้อมูลของอีกอ็อบเจกต์หนึ่ง จะต้องทำการเรียกใช้งานผ่านทางเมท็อดที่อ็อบเจกต์นั้นกำหนดไว้เท่านั้น ซึ่งก็คือเมท็อดของอ็อบเจกต์นั้น และส่วนข้อมูลดังกล่าวหมายถึงแอททริบิวต์ของ อ็อบเจกต์นั่นเอง

- การสืบทอด (Inheritance)

เป็นความสามารถที่ทำให้ คลาสหนึ่งสามารถถ่ายทอดคุณสมบัติไปให้กับคลาสอื่นได้ โดยคลาสที่สืบทอดมาจะได้รับแอททริบิวต์และเมท็อดของคลาสที่ถ่ายทอดมาทั้งหมด ซึ่งคลาสที่ทำการถ่ายทอดเรียกว่าซุปเปอร์คลาส (Super Class) ส่วนคลาสได้รับการถ่ายทอดเรียกว่าซับคลาส (Sub Class)

- การเชื่องโยงแบบไดนามิก (Dynamic Binding)

เป็นการกระทำที่ตัวแปลภาษาจะจัดการกับตัวแปรต่างๆของโปรแกรม ที่อยู่ในตารางสัญลักษณ์ (Symbol Table) ในระหว่างที่โปรแกรมทำงาน โดยมีผลให้ตัวแปรที่ถูกอ้างถึงในขอบเขตหนึ่งของโปรแกรมนั้น มีขอบเขตการมองเห็นที่เปลี่ยนแปลงได้ระหว่างที่โปรแกรมทำงาน

- การทำงานหลายพฤติกรรม (Polymorphism)

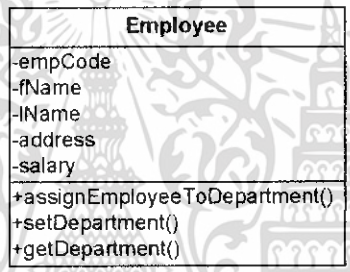
เป็นคุณสมบัติที่ทำให้สามารถส่งเมสเสจออกไปให้กับอ็อบเจกต์หลายๆอ็อบเจกต์ แล้วแต่ละอ็อบเจกต์มีการตอบสนองหรือมีการทำงานต่อเมสเสจนั้นแตกต่างกันออกไป โดยที่การทำงานจะเป็นอย่างไรนั้นขึ้นอยู่กับอ็อบเจกต์ที่รับเมสเสจนั้น

2.1.1 ยูเอ็มแอลไดอะแกรม (UML Diagram)

ยูเอ็มแอลไดอะแกรม เป็นไดอะแกรมมาตรฐานที่ใช้ในการแสดงองค์ประกอบความสัมพันธ์ และลำดับการทำงานของคลาสและอ็อบเจกต์ ซึ่งยูเอ็มแอลไดอะแกรมนี้แบ่ง

ออกเป็น 9 ประเภท ได้แก่ ยูสเคสไดอะแกรม (Use Case Diagram) คลาสไดอะแกรม (Class Diagram) อ็อบเจกต์ไดอะแกรม (Object Diagram) ซีเควนซ์ไดอะแกรม (Sequence Diagram) คอลลาโบเลชั่นไดอะแกรม (Collaboration Diagram) สเตทไดอะแกรม (State Diagram) แอ็คทิวิตี้ไดอะแกรม (Activity Diagram) คอมโพเนนท์ไดอะแกรม (Component Diagram) และดีพลอยเมนท์ไดอะแกรม (Deployment Diagram) แต่ในบทนี้จะอธิบายเฉพาะบางไดอะแกรมที่จะใช้ภายในโครงงานนี้เท่านั้น ซึ่งได้แก่ คลาสไดอะแกรม ซีเควนซ์ไดอะแกรม คอมโพเนนท์ไดอะแกรม และดีพลอยเมนท์ไดอะแกรม ส่วนไดอะแกรมอื่นๆถ้าต้องการรายละเอียดเพิ่มเติมสามารถหาเอกสารอ้างอิงเพิ่มเติมได้ เช่น [3] เป็นต้น

คลาสไดอะแกรม (Class Diagram) คือ ไดอะแกรมที่แสดงถึงรายละเอียดของคลาสซึ่งประกอบด้วยชื่อคลาส แอททริบิวต์ และเมท็อด แสดงได้ดังรูป 2.2

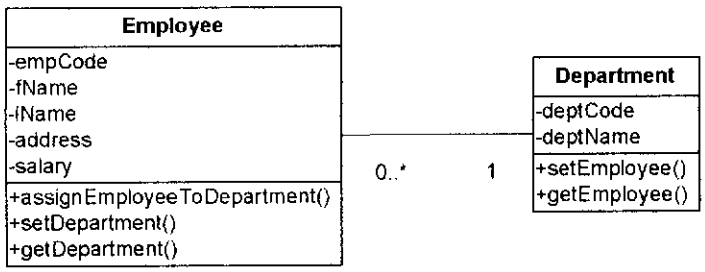


รูป 2.2 แสดงตัวอย่างของคลาส

จากรูป 2.2 คลาสไดอะแกรมแบ่งออกเป็น 3 ส่วน ได้แก่ ส่วนบนคือชื่อคลาส โดยในตัวอย่างนี้มีชื่อว่า Employee ส่วนกลางคือแอททริบิวต์ ซึ่งประกอบไปด้วย 5 แอททริบิวต์ ดังนี้ empCode , fName , lName , address และ salary และในส่วนล่างคือเมท็อด ซึ่งประกอบไปด้วย 3 เมท็อด ดังนี้ assignEmployeeToDepartment , setDepartment และ getDepartment

และนอกจากนี้คลาสไดอะแกรม ยังแสดงถึงความสัมพันธ์ของคลาสต่างๆในระบบงาน ซึ่งความสัมพันธ์ระหว่างคลาสมีหลายลักษณะ ซึ่งแบ่งได้ดังต่อไปนี้

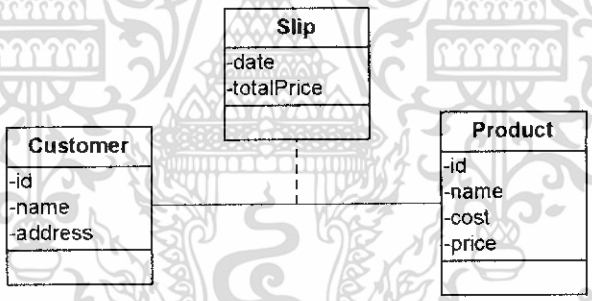
- ความสัมพันธ์ (Association) เป็นการแสดงการเชื่อมโยงกันระหว่างคลาส ว่าคลาสใดมีความสัมพันธ์กันบ้าง และยังสามารถแสดงถึงจำนวนของอ็อบเจกต์ที่เกิดจากความสัมพันธ์ของคลาสได้อีกด้วย ดังรูป 2.3



รูป 2.3 ตัวอย่างแสดงความสัมพันธ์กันระหว่างคลาส

จากรูป 2.3 เป็นการแสดงความสัมพันธ์กันระหว่างคลาสพนักงาน (Employee) และคลาสแผนก (Department) โดยที่ตัวเลขที่เส้นเชื่อมความสัมพันธ์มีความหมายว่าแผนก 1 แผนก อาจจะมีพนักงานหรือไม่มีก็ได้ และถ้ามีสามารถมีพนักงานได้หลายคน แต่พนักงาน 1 คน สามารถทำงานได้เพียงแผนกเดียวเท่านั้น

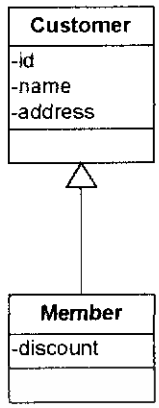
นอกจากนี้ความสัมพันธ์ในบางกรณีอาจทำให้เกิดคลาสใหม่ขึ้นมาได้ โดยเรียกคลาสดังกล่าวว่า คลาสความสัมพันธ์ (Association Class) ดังรูป 2.4



รูป 2.4 ตัวอย่างคลาสที่เกิดจากความสัมพันธ์กันของคลาส

จากรูป 2.4 เรียกคลาส Slip ว่าเป็นคลาสที่เกิดจากความสัมพันธ์กันระหว่างคลาส Customer และคลาส Product โดยที่คลาส Slip นั้นจะเกิดขึ้นได้ก็ต่อเมื่อคลาสทั้ง 2 นั้นเกิดความสัมพันธ์กันขึ้น

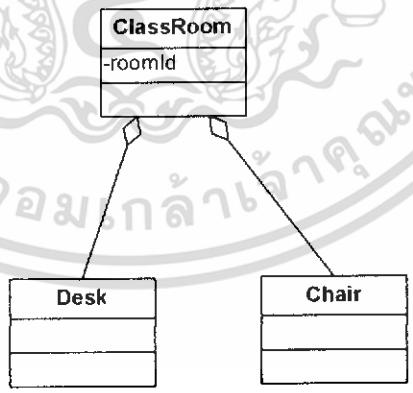
- การสืบทอดคุณสมบัติ การเขียนไดอะแกรมนั้นหัวลูกศรจะชี้ที่ซุปเปอร์คลาส และหางลูกศรจะชี้ที่ซับคลาส ดังรูป 2.5



รูป 2.5 ตัวอย่างคลาสที่มีการการสืบทอดคุณสมบัติ

จากรูปที่ 2.5 คลาส Member ได้รับการสืบทอดคลาสมาจากคลาส Customer ทำให้คลาส Member จะมีแอททริบิวต์ทั้ง 3 ตัวของคลาส Customer ด้วย

- แอกกริเกชัน (Aggregation) เป็นการแสดงความสัมพันธ์กันระหว่างคลาส โดยที่คลาสหนึ่งจะเข้าไปเป็นแอททริบิวต์ของอีกคลาสหนึ่งซึ่งเป็นคลาสหลัก แต่เมื่อใดก็ตามที่อ็อบเจกต์ของคลาสหลักจบการทำงานลง คลาสที่เป็นแอททริบิวต์ของคลาสหลักไม่จำเป็นต้องจบการทำงานตามไปด้วย การเขียนไดอะแกรมใช้สัญลักษณ์เส้นตรงและมีรูปข้ามหลามตัดสีขาวตรงปลายเส้น โดยที่ด้านหัวที่เป็นข้าวหลามตัดจะชี้ไปที่คลาสที่เป็นคลาสหลัก ส่วนด้านปลายจะชี้ไปที่คลาสที่เป็นส่วนประกอบภายในคลาสหลัก ดังรูป 2.6

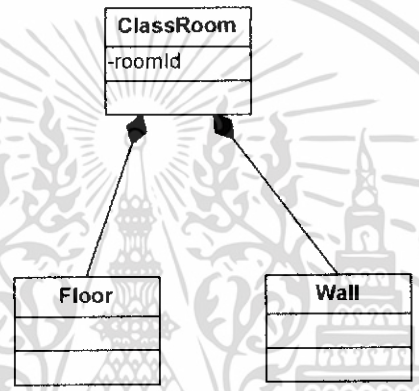


รูป 2.6 ตัวอย่างคลาสแอกกริเกชัน

จากรูป 2.6 เป็นการแสดงตัวอย่างคลาสที่เกิดการรวมกลุ่มกัน โดยที่อ็อบเจกต์ห้องเรียน (ClassRoom) นั้นเมื่อถูกสร้างขึ้นแอททริบิวต์จะประกอบไปด้วยหมายเลขห้อง อ็อบเจกต์โต๊ะ

(Desk) และอ็อบเจกต์เก้าอี้ (Chair) แต่ถ้าอ็อบเจกต์ห้องเรียนจบการทำงานลงอ็อบเจกต์โต๊ะและอ็อบเจกต์เก้าอี้ไม่จำเป็นต้องจบการทำงานตามไปด้วย

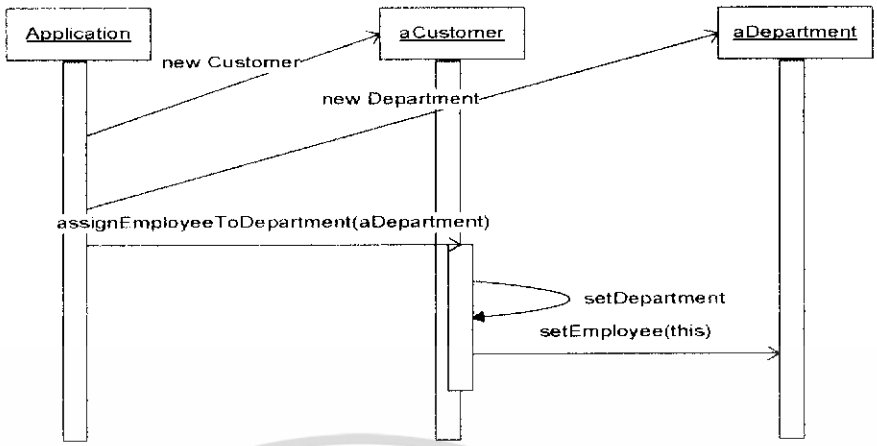
- คอมโพซิชัน (Composition) เป็นความสัมพันธ์กันระหว่างคลาส โดยที่คลาสหนึ่งจะเข้าไปเป็นแอททริบิวต์ของอีกคลาสหนึ่งซึ่งเป็นคลาสหลัก แต่จะต่างกันกับแอกกริเกชันตรงที่เมื่อใดก็ตามที่อ็อบเจกต์ของคลาสหลักจบการทำงานลง คลาสที่เป็นแอททริบิวต์ของคลาสหลักจำเป็นต้องจบการทำงานตามไปด้วย การเขียนไดอะแกรมใช้สัญลักษณ์เส้นตรงและมีรูปข้ามหลามตัดสีดำตรงปลายเส้น โดยที่ด้านหัวที่เป็นข้าวหลามตัดจะชี้ไปที่คลาสที่เป็นคลาสหลัก ส่วนด้านปลายจะชี้ไปที่คลาสที่เป็นองค์ประกอบภายในคลาสหลัก ดังรูป 2.7



รูป 2.7 ตัวอย่างคลาสคอมโพซิชัน

จากรูป 2.7 เป็นการแสดงตัวอย่างคลาสที่คอมโพซิชัน โดยที่อ็อบเจกต์ห้องเรียน (ClassRoom) นั้นเมื่อถูกสร้างขึ้นแอททริบิวต์จะประกอบไปด้วยหมายเลขห้อง อ็อบเจกต์กำแพง (Wall) และอ็อบเจกต์พื้น (Floor) แต่ถ้าอ็อบเจกต์ห้องเรียนจบการทำงานลง อ็อบเจกต์กำแพงและอ็อบเจกต์พื้นก็ต้องจบการทำงานตามไปด้วย

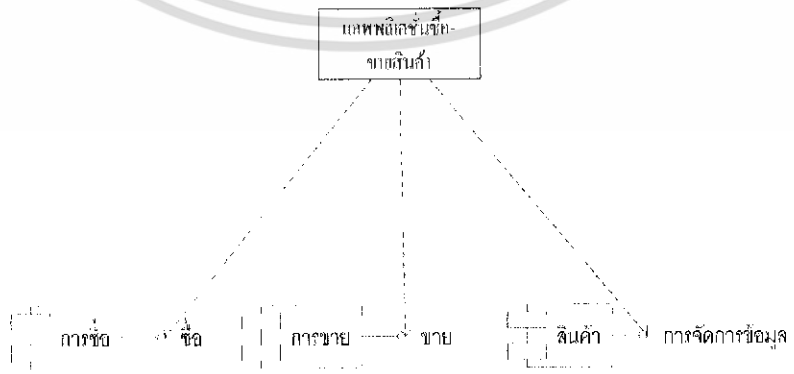
ซีเควนซ์ไดอะแกรม (Sequence Diagram) คือ ไดอะแกรมที่แสดงถึงลำดับการทำงานของโปรแกรม โดยจะเริ่มจากซ้ายไปขวาและจากบนล่าง และแสดงถึงการติดต่อกันของอ็อบเจกต์โดยการส่งเมสเสจ ดังรูป 2.8



รูป 2.8 ตัวอย่างซีเควนซ์ไดอะแกรม

จากซีเควนซ์ไดอะแกรม ในรูป 2.8 มีการทำงานโดย เริ่มต้นจาก Application สร้างอ็อบเจกต์ aCustomer ของ คลาส Customer และอ็อบเจกต์ aDepartment ของคลาส Department ขึ้นมา และส่งเมสเสจผ่านทางเมท็อด assignEmployeeToDepartment ของอ็อบเจกต์ aCustomer จากนั้นเมท็อด assignEmployeeToDepartment ของอ็อบเจกต์ aCustomer ก็จะถูกเรียกให้ทำงานต่อไป

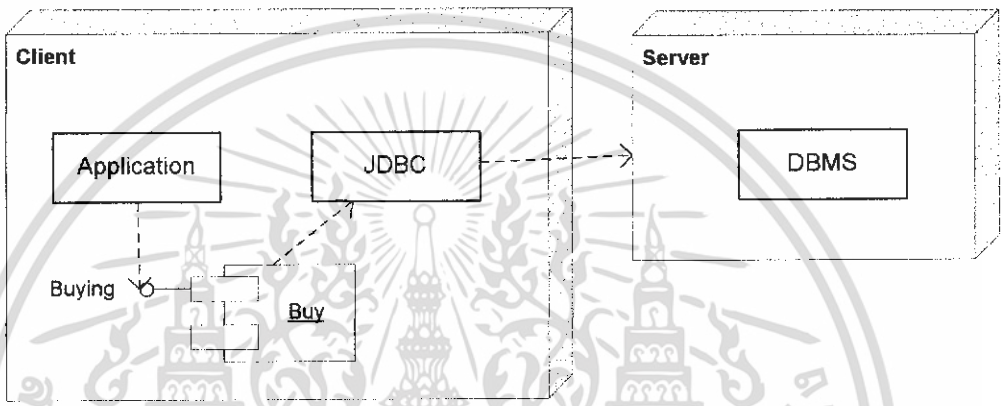
คอมโพเนนท์ไดอะแกรม (Component Diagram) เป็นไดอะแกรมซึ่งแสดงถึงโครงสร้างของซอฟต์แวร์ว่าประกอบด้วยคอมโพเนนท์ใดบ้าง และแต่ละคอมโพเนนท์มีการติดต่อกับคอมโพเนนท์อื่นโดยผ่านอินเตอร์เฟซใด โดยรูปกล่องสี่เหลี่ยมแทนคอมโพเนนท์ เส้นตรงมีวงกลมที่ปลาย 1 ด้านแทนอินเตอร์เฟซ และใช้เส้นประมีหัวลูกศร 1 ด้าน หมายถึงการเรียกใช้งานคอมโพเนนท์ โดยที่ด้านที่มีหัวลูกศรหมายถึงคอมโพเนนท์ที่ถูกเรียกใช้ ส่วนด้านปลายหมายถึงคอมโพเนนท์ที่เรียกใช้งาน ดังรูป 2.9



รูป 2.9 แสดงตัวอย่างคอมโพเนนท์ไดอะแกรม

จากรูป 2.9 แอพพลิเคชันของระบบซื้อขายสินค้าประกอบไปด้วยคอมโพเนนท์ของการซื้อการขาย และการจัดการข้อมูลสินค้า โดยที่แอพพลิเคชันหลักจะติดต่อกับคอมโพเนนท์ต่างๆ ผ่านทางอินเตอร์เฟซที่คอมโพเนนท์นั้นๆเตรียมไว้ให้

ดีพลอยเมนท์ไดอะแกรม (Deployment Diagram) เป็นการแสดงถึงสถาปัตยกรรมของฮาร์ดแวร์ (Hardware) และซอฟต์แวร์ (Software) ตลอดจนความสัมพันธ์ระหว่าง ฮาร์ดแวร์และซอฟต์แวร์ ที่แอพพลิเคชันจำเป็นต้องใช้เพื่อให้แอพพลิเคชันสามารถทำงานได้อย่างสมบูรณ์ ดังรูป 2.10



รูป 2.10 แสดงตัวอย่างดีพลอยเมนท์ไดอะแกรม

จากรูป 2.10 การทำงานของระบบประกอบไปด้วย 2 ส่วน คือ ส่วนของไคลเอนท์ (Client) และ เซิฟเวอร์ (Server) โดยส่วนไคลเอนท์จะต้องติดตั้งแอพพลิเคชันที่ใช้ติดต่อกับผู้ใช้งาน และ เจดีบีซี (JDBC) เพื่อใช้ในการติดต่อกับระบบจัดการฐานข้อมูล (DBMS) และในส่วนของเซิฟเวอร์ต้องติดตั้งระบบจัดการฐานข้อมูล

2.1.2 ปัญหาของการพัฒนาโปรแกรมเชิงอ็อบเจกต์

การพัฒนาโปรแกรมเชิงอ็อบเจกต์นั้นมีจุดเริ่มมาจากคลาส จากนั้นทำการสร้างอ็อบเจกต์ของคลาสขึ้นมาใช้งาน และอ็อบเจกต์แต่ละอ็อบเจกต์สามารถติดต่อสื่อสารกันระหว่างอ็อบเจกต์ได้โดยการส่งเมสเสจหากัน แต่การทำงานลักษณะนี้ก่อให้เกิดปัญหาขึ้นในการพัฒนาโปรแกรมหลายประการดังต่อไปนี้

1. การนำโค้ดกลับมาใช้ใหม่

การนำโค้ดกลับมาใช้ใหม่ของการพัฒนาโปรแกรมเชิงอ็อบเจกต์นั้น สามารถทำได้โดยการใช้คุณสมบัติการสืบทอดคลาส แต่การสืบทอดคลาสนั้นยังมีปัญหาในการใช้งานอยู่บ้าง เนื่องจากคลาสที่ได้รับการถ่ายทอดนั้นจะผูกติดกับคลาสที่ทำการถ่ายทอด คลาสที่ได้รับการถ่ายทอดนั้นจะได้รับแอททริบิวต์และเมท็อดจากคลาสที่ทำการถ่ายทอดมาทั้งหมด ดังนั้นถ้า

คลาสที่ทำการถ่ายทอดมีการเปลี่ยนแปลงไปจากเดิม ก็จะส่งผลกระทบกับคลาสที่ได้รับการถ่ายทอดด้วย

2. การเปลี่ยนแปลงเวอร์ชัน

การพัฒนาโปรแกรมเชิงอ็อบเจกต์นั้นไม่สนับสนุนการทำงานในส่วนของการเปลี่ยนแปลงเวอร์ชัน เนื่องจากเมื่อมีการเปลี่ยนแปลงเวอร์ชันของคลาส การทำงานบางส่วนอาจมีการเปลี่ยนแปลงไป และเมื่อคลาสอื่นมีการติดต่อกับคลาสที่มีการเปลี่ยนแปลงเวอร์ชัน บางครั้งอาจจะไม่สามารถทำงานได้ เนื่องจากการพัฒนาโปรแกรมเชิงอ็อบเจกต์ไม่สามารถตรวจสอบได้ว่าคลาสที่เข้ามาติดต่อนั้นติดต่อกับเวอร์ชันใด และไม่สามารถควบคุมให้มีการเลือกเวอร์ชันที่ถูกต้องขึ้นมาทำงานได้

3. ภาษาในการพัฒนาโปรแกรม

การพัฒนาโปรแกรมเชิงอ็อบเจกต์นั้นไม่สามารถทำงานข้ามภาษากันได้ ดังนั้นการพัฒนาแอพพลิเคชันขึ้นมาด้วยการพัฒนาโปรแกรมเชิงอ็อบเจกต์นั้น ภาษาที่ใช้ในการพัฒนาจำเป็นต้องใช้ภาษาเดียวกันในการพัฒนา ไม่สามารถพัฒนาโดยการใช้หลายๆภาษามารวมกันเป็นแอพพลิเคชันได้ ทำให้โปรแกรมเมอร์เกิดปัญหาในการเลือกใช้ภาษาสำหรับการพัฒนาโปรแกรมได้ เช่น ถ้าจำเป็นต้องใช้ภาษาจาวาในการพัฒนาโปรแกรม แต่มีบางคนในทีมพัฒนาไม่ถนัดการพัฒนาโปรแกรมด้วยจาวา แต่ถนัดการใช้ซีพลัสพลัส ทำให้ต้องทำการศึกษาจาวาก่อนจึงจะทำการพัฒนาโปรแกรมได้ ทำให้เสียเวลาในการพัฒนาโปรแกรม

4. โมดูลขาดความอิสระ

การที่อ็อบเจกต์หนึ่งจะส่งเมสเสจไปให้อีกอ็อบเจกต์หนึ่งได้นั้น คลาสของอ็อบเจกต์ที่ต้องการส่งเมสเสจจะต้องมีอ็อบเจกต์ของคลาสที่รับเมสเสจ ตัวอย่างเช่น ถ้าอ็อบเจกต์ A ต้องการส่งเมสเสจไปหาอ็อบเจกต์ B นั้น คลาสของอ็อบเจกต์ A จะต้องมีอ็อบเจกต์ของคลาส B รวมอยู่ด้วย เป็นต้น จากตัวอย่างจะเห็นได้ว่าคลาสขาดความเป็นอิสระจากกัน เนื่องจากภายในคลาส A มีอ็อบเจกต์ของคลาส B รวมอยู่ด้วย ดังนั้นการนำคลาสไปใช้งานนั้นจำเป็นต้องใช้ทั้ง 2 คลาส ไม่สามารถนำคลาส A หรือว่าคลาส B ไปใช้แยกต่างหากได้ ซึ่งบางครั้งอาจจะต้องการใช้งานเพียงคลาสใดคลาสหนึ่งเท่านั้นแต่ก็ไม่สามารถทำได้

5. ครอสคัตติ้งคอนเซิน (Cross-Cutting Concern)

คอนเซิน (Concern) คือ เหตุการณ์ต่างๆที่เราสนใจในการทำงานของโปรแกรม เช่น การจัดการกับข้อมูล ได้แก่ การเพิ่ม ลบ แก้ไข และค้นหาข้อมูล การตรวจจับข้อผิดพลาดของโปรแกรม เป็นต้น

ครอสคัตติ้งคอนเซิน คือ ปัญหาที่เกิดจากการที่คอนเซินต่างๆนั้นกระจายไปอยู่ในหลายที่ของโปรแกรม ทำให้ยากต่อการทำความเข้าใจ ดูแลและแก้ไข เช่น ถ้าเมท็อดที่เป็นคอนเซินมีการเปลี่ยนแปลงจำนวนพารามิเตอร์ โปรแกรมเมอร์จะต้องทำการแก้ไขจำนวนพารามิเตอร์

ของเมท็อดที่เป็นคอนเซินทุกๆที่ของโปรแกรมที่มีการเรียกใช้งานเมท็อดนี้ ซึ่งการพัฒนาโปรแกรมเชิงอ็อบเจกต์ที่ใช้อ็อบเจกต์เป็นหลักในการดำเนินการของโปรแกรมนั้น ไม่สามารถหลีกเลี่ยงปัญหานี้ได้ ดังรูป 2.11

```
public class Student {
        private String id ;
        private String name ;
        private double gpa ;
        public Student(String id,String name,double gpa) {
            setId(id) ;
            setName(name) ;
            setGpa(gpa) ;
        }
        public void setId(String id) {
            this.id = id ;
        }
        public void setName(String name) {
            this.name = name ;
        }
        public void setGpa(double gpa) {
            this.gpa = gpa ;
        }
        public String getId() {
            return id ;
        }
        public String getName() {
            return name ;
        }
        public double getGpa() {
            return gpa ;
        }
        public void add(String id,String name,double gpa) {
            String str = "insert into Customer values ('" + id + "','" + name +
                         "'," + gpa + ")" ;
            ConDB con = new ConDB() ;
            con.SQLExecute(str) ;
        }
}
public class Books {
        private String isbn ;
        private String name ;
        private double price ;
        public Books(String isbn,String name,double price) {
            setIsbn(id) ;
            setName(name) ;
            setPrice(price) ;
        }
        public void setIsbn(String isbn) {
            this.isbn = isbn ;
        }
        public void setName(String name) {
            this.name = name ;
        }
        public void setGpa(double price) {
            this.price = price ;
        }
        public String getIsbn() {
            return isbn ;
        }
        public String getName() {
            return name ;
        }
        public double getGpa() {
            return price ;
        }
        public void add(String isbn,String name,double price) {
            String str = "insert into book values ('" + isbn + "','" + name +
                         "'," + price + ")" ;
            ConDB con = new ConDB() ;
            con.SQLExecute(str) ;
        }
}
```

รูป 2.11 แสดงปัญหาของครอสคัตติ้งคอนเซิน

จากรูป 2.11 ส่วนที่เกิดปัญหาครอสคัตติ้งคอนเซิน คือส่วนที่อยู่ในกรอบสี่เหลี่ยม โดยที่คลาส Student และคลาส Books มีการเรียกใช้เมท็อด SQLExcute ของคลาส ConDB ซึ่งทำให้เกิดโค้ดที่ซ้ำซ้อน

2.2 การพัฒนาโปรแกรมเชิงคอมโพเนนท์ (Component Oriented Programming : COP)

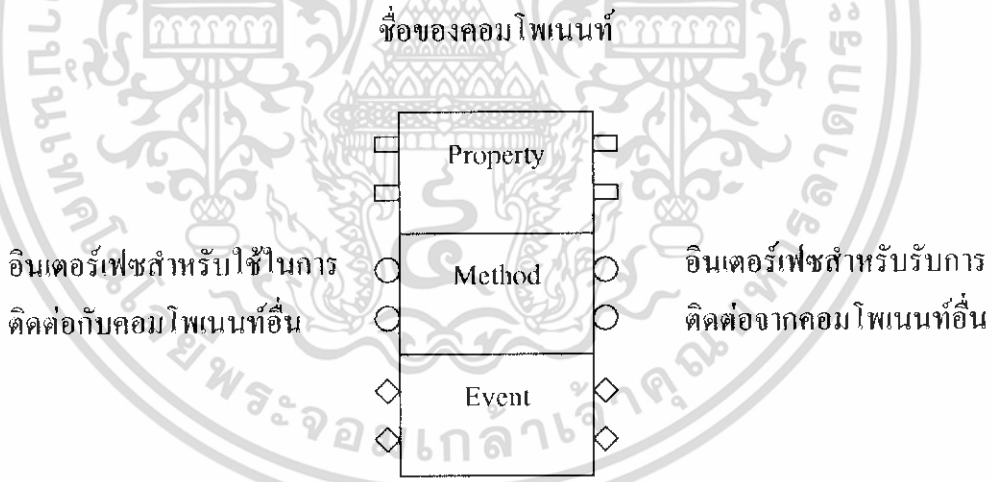
คอมโพเนนท์เป็นหน่วยที่ย่อยที่สุดของการพัฒนาโปรแกรมเชิงคอมโพเนนท์ โดยที่คอมโพเนนท์นั้น คือ กลุ่มของโปรแกรม 1 กลุ่ม ที่สามารถทำงานอย่างใดอย่างหนึ่งให้สำเร็จได้ด้วยตัวของมันเอง โดยที่ภายในคอมโพเนนท์อาจจะมีแบ่งการทำงานออกเป็นส่วนย่อยๆ หลายๆ ส่วนหรือมีเพียงส่วนเดียวก็ได้ และคอมโพเนนท์ต่างๆสามารถติดต่อกันได้โดยผ่านทางอินเตอร์เฟซ ซึ่งจากลักษณะดังกล่าวทำให้การพัฒนาแอพพลิเคชันสามารถทำได้โดยการนำคอมโพเนนท์หลายคอมโพเนนท์มาประกอบกันขึ้นเป็นแอพพลิเคชัน

2.2.1 แบบจำลองของคอมโพเนนท์ (Component Model)

คอมโพเนนท์ทุกคอมโพเนนท์จะต้องมีชื่อเพื่อใช้สำหรับระบุและอ้างถึงคอมโพเนนท์นั้นๆ และภายในคอมโพเนนท์จะประกอบด้วย 3 ส่วน ได้แก่

1. คุณสมบัติ (Property) คือ ข้อมูลที่อยู่ภายในคอมโพเนนท์
2. เมท็อด (Method) คือ พฤติกรรมของคอมโพเนนท์ที่สามารถทำได้
3. เหตุการณ์ (Event) คือ สิ่งที่กระตุ้นให้คอมโพเนนท์เกิดการทำงาน

เราสามารถแสดงแบบจำลองของคอมโพเนนท์ในรูปของรูปภาพได้ ซึ่งเรียกว่า แผนภาพคอมโพเนนท์ (Component Chart) ดังรูป 2.12



รูป 2.12 แสดงแผนภาพคอมโพเนนท์

จากรูป 2.12 แผนภาพคอมโพเนนท์แสดงถึงองค์ประกอบต่างๆของคอมโพเนนท์ โดยที่รูปสี่เหลี่ยมผืนผ้า วงกลม และ สี่เหลี่ยมจัตุรัสทั้งสองข้างแทนคุณสมบัติ เมท็อด และ เหตุการณ์ตามลำดับ

2.2.2 ดีพลอยเมนท์ (Depolyment)

ในคอมโพเนนท์ทุกคอมโพเนนท์จะต้องมีส่วนการทำงานของการดีพลอยเมนท์ ซึ่งดีพลอยเมนท์นั้นมีจุดประสงค์เพื่อ สร้างสภาพแวดล้อม (Environment) ต่างๆเพื่อทำให้คอมโพเนนท์สามารถทำงานได้ โดยที่ดีพลอยเมนท์นั้นจะประกอบไปด้วยกระบวนการสำหรับการเตรียมการรันคอมโพเนนท์ การติดตั้งคอมโพเนนท์ และการกำหนดคุณสมบัติ (Configuration) ของคอมโพเนนท์ให้เหมาะสมกับการทำงาน

2.2.3 การนำคอมโพเนนท์ไปใช้งาน

คอมโพเนนท์นั้นจะอยู่ในรูปของโปรแกรมที่พร้อมใช้งาน ซึ่งประกอบไปด้วย 2 ส่วน คือ ส่วนของโค้ดการทำงานและส่วนของการดีพลอยเมนท์ โดยที่คอมโพเนนท์สามารถทำงานที่ใดก็ได้ ที่มีการปรับสภาพแวดล้อมให้เหมาะสมกับการทำงานของคอมโพเนนท์ และเมื่อทำการติดตั้งคอมโพเนนท์แล้ว คอมโพเนนท์ก็จะสามารถทำงานร่วมกับส่วนอื่นๆของแอพพลิเคชันได้โดยผ่านทางอินเตอร์เฟซ

2.2.4 ความแตกต่างระหว่างการพัฒนาโปรแกรมเชิงอ็อบเจกต์กับคอมโพเนนท์

การพัฒนาโปรแกรมทั้ง 2 วิธีนั้น มีความแตกต่างกันหลายประการ ดังต่อไปนี้

1. ความเป็นอิสระของโมดูล

เนื่องจากคอมโพเนนท์มีการทำงานผ่านทางอินเตอร์เฟซ ทำให้คอมโพเนนท์แต่ละคอมโพเนนท์มีความเป็นอิสระจากกัน และโปรแกรมมีความยืดหยุ่นในการใช้งาน

2. ภาษาในการพัฒนาโปรแกรม

การพัฒนาโปรแกรมเชิงคอมโพเนนท์นั้นเวลานำไปใช้งานจะอยู่ในรูปของไบนารีโค้ด ซึ่งเป็นรูปแบบภาษาที่คอมพิวเตอร์เข้าใจ ดังนั้นการพัฒนาแอพพลิเคชันหนึ่งสามารถประกอบด้วยคอมโพเนนท์ที่พัฒนาขึ้นมาจากหลายๆภาษาได้ เช่น ถ้าพัฒนาแอพพลิเคชันด้วยจาวา แต่เรามีคอมโพเนนท์ที่พัฒนาด้วยซีพลัสพลัสที่มีการทำงานตรงกับแอพพลิเคชัน เราก็สามารถนำคอมโพเนนท์นั้นมาใช้ร่วมกับส่วนอื่นของโปรแกรมได้

3. การนำคอมโพเนนท์กลับมาใช้ใหม่

การพัฒนาแอพพลิเคชันหลายๆแอพพลิเคชัน บางครั้งในแต่ละแอพพลิเคชันอาจมีหน้าที่และการทำงานบางอย่างที่เหมือนกัน ในการพัฒนาโปรแกรมเชิงคอมโพเนนท์นั้นสามารถนำคอมโพเนนท์เดิมที่มีอยู่ไปใช้ได้กับแอพพลิเคชันที่ต้องการได้ ถ้าสภาพแวดล้อมของการทำงานของแอพพลิเคชันเหมาะสมกับการทำงานของคอมโพเนนท์ ซึ่งแตกต่างจากการพัฒนาโปรแกรมเชิงอ็อบเจกต์ เนื่องจากคลาสที่นำไปใช้บางครั้งอาจจะผูกติดอยู่กับคลาสอื่น หรือภาษาที่ใช้อาจจะเป็นคนละภาษา ทำให้ไม่สามารถนำคลาสที่ต้องการไปใช้งานต่อได้

4. การเปลี่ยนแปลงเวอร์ชัน

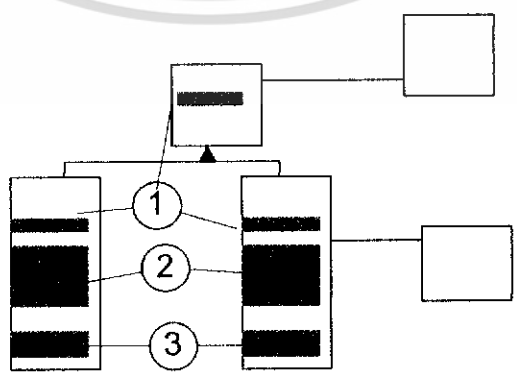
การพัฒนาโปรแกรมเชิงคอมโพเนนท์นั้น สนับสนุนการทำงานในส่วนของการเปลี่ยนแปลงเวอร์ชัน ถ้าคอมโพเนนท์มีการเปลี่ยนแปลงเวอร์ชัน การพัฒนาโปรแกรมเชิงคอมโพเนนท์สามารถตรวจสอบได้ว่า คอมโพเนนท์อื่นที่เข้ามาติดต่อนั้นติดต่อกับเวอร์ชันใด และทำการเลือกใช้คอมโพเนนท์เวอร์ชันที่ถูกต้องมาทำงานได้ เช่น แอพพลิเคชันของการให้บริการโทรศัพท์มือถือ ประกอบไปด้วยคอมโพเนนท์ของโปรโมชัน การจัดการข้อมูลเบอร์โทรศัพท์และลูกค้า เมื่อมีโปรโมชันใหม่ออกมาก็ต้องสร้างคอมโพเนนท์โปรโมชันมาใหม่ แต่ในขณะเดียวกันโปรโมชันเก่าก็ยังมีการใช้งานต่อไปด้วย ดังนั้นเมื่อลูกค้ามีการใช้โทรศัพท์มือถือ คอมโพเนนท์ต้องสามารถตรวจสอบได้ว่าเบอร์โทรศัพท์นี้ใช้โปรโมชันใดอยู่ และต้องเลือกคอมโพเนนท์โปรโมชันที่ถูกต้องขึ้นมาทำงานได้

2.3 การพัฒนาโปรแกรมเพื่อแก้ไขปัญหาครอสคัตติ้งคอนเซิน

การพัฒนาโปรแกรมเชิงคอมโพเนนท์ สามารถแก้ไขปัญหาของการพัฒนาโปรแกรมเชิงอ็อบเจกต์ได้หลายประการ แต่ก็ยังไม่สามารถแก้ไขปัญหาครอสคัตติ้งคอนเซินได้ ดังนั้นจึงได้มีการคิดค้นการพัฒนาโปรแกรมขึ้นมาใหม่เพื่อการแก้ไขปัญหาครอสคัตติ้งคอนเซิน ซึ่งมีอยู่ด้วยกันหลายวิธี เช่น การพัฒนาโปรแกรมแบบเอ็กพลิซิต (Explicit Programming) และ การพัฒนาโปรแกรมเชิงแอสเปค เป็นต้น แต่ในโครงงานนี้จะใช้วิธีการพัฒนาโปรแกรมเชิงแอสเปคเนื่องจากเป็นวิธีที่ได้รับความนิยม ซึ่งจะอธิบายเนื้อหาอย่างละเอียดในหัวข้อถัดไป ส่วนวิธีการอื่นนั้นจะอธิบายหลักการในการพัฒนาโปรแกรมอย่างคร่าวๆ ดังต่อไปนี้

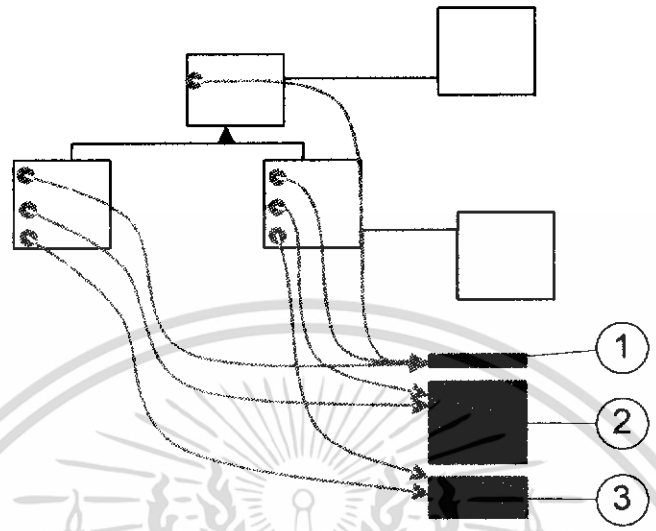
2.3.1 การพัฒนาโปรแกรมแบบเอ็กพลิซิต

การพัฒนาโปรแกรมด้วยวิธีนี้ จะทำการแยกโค้ดที่มีการทำงานซ้ำซ้อนกันออกมาไว้ในโมดูลที่สร้างขึ้นใหม่ และสร้างจุดที่จะเชื่อมการทำงานระหว่างโปรแกรมในส่วนของอ็อบเจกต์และโมดูลใหม่ที่สร้างขึ้น โดยที่เรียกจุดนี้ว่า โวแคบบูลารี (Vocabulary) ดังนั้นเมื่อโปรแกรมทำงานมาถึงจุดที่สร้างโวแคบบูลารีไว้ โวแคบบูลารีก็จะเรียกโมดูลที่เชื่อมการทำงานกับมันไว้ขึ้นมาทำงาน ดังรูป 2.13 และ 2.14



รูป 2.13 แสดงปัญหาครอสคัตติ้งคอนเซิน

จากรูป 2.13 หมายเลข 1 2 และ 3 ต่างๆคือโค้ดส่วนที่มีการทำงานซ้ำซ้อนกันซึ่งก่อให้เกิดปัญหา ครอสคัตติ้งคอนเซิน



รูป 2.14 แสดงการพัฒนาโปรแกรมแบบเอ็กพลิซิต

จากรูป 2.14 วงกลมแทนโวแคบบูลารีที่สร้างขึ้นในส่วนของโปรแกรมที่พัฒนาด้วย อ็อบเจกต์ และหมายเลข 1 2 และ 3 คือโค้ดส่วนที่มีการทำงานซ้ำซ้อนกันซึ่งแยกออกมาสร้างไว้ในโมดูลใหม่ และทำการเชื่อมการทำงานระหว่างโวแคบบูลารีและโมดูลที่สร้างขึ้น

ELIDE คือภาษาที่ใช้ในการพัฒนาโปรแกรมแบบเอ็กพลิซิตที่ทำงานร่วมกับภาษาจาวา โดยภาษานี้จะใช้หลักการสร้างโวแคบบูลารีขึ้นในแอททริบิวต์ หรือเมท็อดที่มีการทำงานซ้ำซ้อนกัน โดยเรียกว่า โมดิฟายเออร์ (Modifier) และแยกโค้ดที่มีการซ้ำซ้อนกันออกมาไว้ในโมดูลใหม่

2.4 การพัฒนาโปรแกรมเชิงแอสเปค (Aspect Oriented Programming : AOP)

การพัฒนาโปรแกรมเชิงแอสเปค เป็นรูปแบบของการพัฒนาโปรแกรมที่มีการทำงานร่วมกับการพัฒนาโปรแกรมเชิงอ็อบเจกต์ โดยมีจุดประสงค์เพื่อขจัดปัญหาครอสคัตติ้งคอนเซินและแทงเกิ้ลคอนเซิน ของการพัฒนาโปรแกรมเชิงอ็อบเจกต์ โดยการแยกส่วนของโค้ดที่ซ้ำซ้อนกันในโปรแกรมออกมาเป็นโมดูลใหม่ซึ่งเรียกว่าแอสเปค (Aspect)

การแยกส่วนของเหตุการณ์ต่างๆที่ซ้ำซ้อนกันในโปรแกรม ออกมาเป็นหน่วยที่แน่นอนของการพัฒนาโปรแกรมเชิงแอสเปคนั้น จะต้องมีการกำหนดการทำงานให้กับแอสเปค ซึ่งเรียกว่า จอยพอยท์ (Join Point) โดยจะอธิบายรายละเอียดในหัวข้อถัดไป และเมื่อโปรแกรม

ทำงานมาถึงจุดที่จอยพอยท์กำหนดไว้ แอสเปคจะถูกเรียกขึ้นมาทำงาน และเมื่อแอสเปคทำงานเสร็จแล้วการทำงานก็จะกลับมาอยู่ในส่วนของอ็อบเจกต์เช่นเดิม

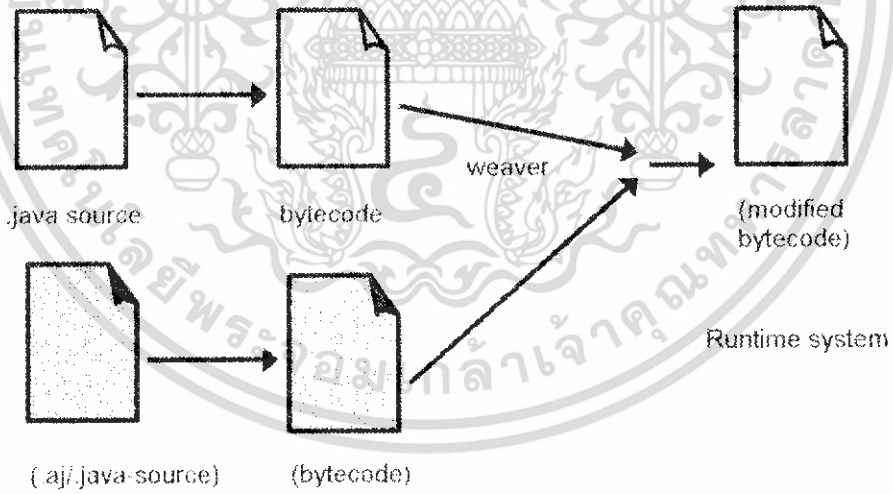
2.4.1 แอสเปคเจ (AspectJ)

แอสเปคเจ คือ ภาษาที่สนับสนุนการพัฒนาโปรแกรมเชิงแอสเปค ซึ่งจะทำงานร่วมกับภาษาจาวา ดังนั้น แอสเปคเจจึงถูกออกแบบมาให้เข้ากันได้ (Compatible) กับภาษาจาวาอย่างสมบูรณ์ โดยแบ่งความเข้ากันได้ออกเป็น 3 ประเภท ดังนี้

1. สิ่งที่ถูกต้องตามกฎของจาวาต้องถูกต้องใน แอสเปคเจโปรแกรม
2. แอสเปคเจโปรแกรมต้องรันบนเจวีเอ็มได้
3. ไวยากรณ์ภาษาของแอสเปคเจต้องมีลักษณะคล้ายกับจาวา เพื่อให้ผู้ใช้สามารถศึกษาได้ง่าย

2.4.2 กลไกการทำงานของแอสเปคเจ (AspectJ Mechanisms)

เริ่มต้นด้วยการคอมไพล์ไฟล์ .java ของจาวาเป็นไบต์โค้ด (Bytecode) และคอมไพล์ไฟล์ .aj ของแอสเปคเจเป็นไบต์โค้ด แล้วจะนำไบต์โค้ดที่ได้ทั้ง 2 รวมกัน โดยเวียร์เวอร์ (Weaver) ซึ่งเวียร์เวอร์นี้จะมีหน้าที่ในการเชื่อมความสัมพันธ์ระหว่างจาวากับแอสเปคเจให้สามารถทำงานร่วมกันได้ โดยจะได้เป็นไบต์โค้ดที่ถูกปรับปรุง (Modified Bytecode) ดังรูป 2.15



รูป 2.15 แสดงถึงกลไกการทำงานของแอสเปคเจ

2.4.3 รูปแบบภาษาของแอสเปคเจ (AspectJ Language)

แอสเปคเจนั้นประกอบไปด้วยโครงสร้างหลักๆ 5 ส่วน ได้แก่ จอยพอยท์ , พอยท์คัต (Pointcut) , แอดไวซ์ (Advice) , การประกาศประเภทข้อมูลร่วมกันระหว่างคลาส (Inter-Type Declaration) และแอสเปค ซึ่งแต่ละส่วนมีรายละเอียดดังต่อไปนี้

1. จอยพอยท์ คือ จุดของเหตุการณ์ต่างๆ ที่เกิดขึ้นในขั้นตอนการทำงานของโปรแกรม (Control Flow of Program) โดยที่แอสเปคเจได้แบ่งเหตุการณ์ที่ทำให้เกิด จอยพอยท์ ดังต่อไปนี้

- การเรียกใช้คอนสทรัคเตอร์ (Constructor Call) ซึ่งเป็นจอยพอยท์ที่เกิดขึ้น เมื่อคอนสทรัคเตอร์เมท็อดถูกเรียกใช้งาน

- การเรียกใช้เมท็อด (Method Call) ซึ่งเป็นจอยพอยท์ที่เกิดขึ้นเมื่อเมท็อดที่ไม่ใช่คอนสทรัคเตอร์เมท็อดถูกเรียกใช้งาน

- การทำงานของคอนสตรัคเตอร์ (Constructor Execution) ซึ่งเป็นจอยพอยท์ที่เกิดขึ้นเมื่อคอนสทรัคเตอร์เมท็อดมีการทำงาน

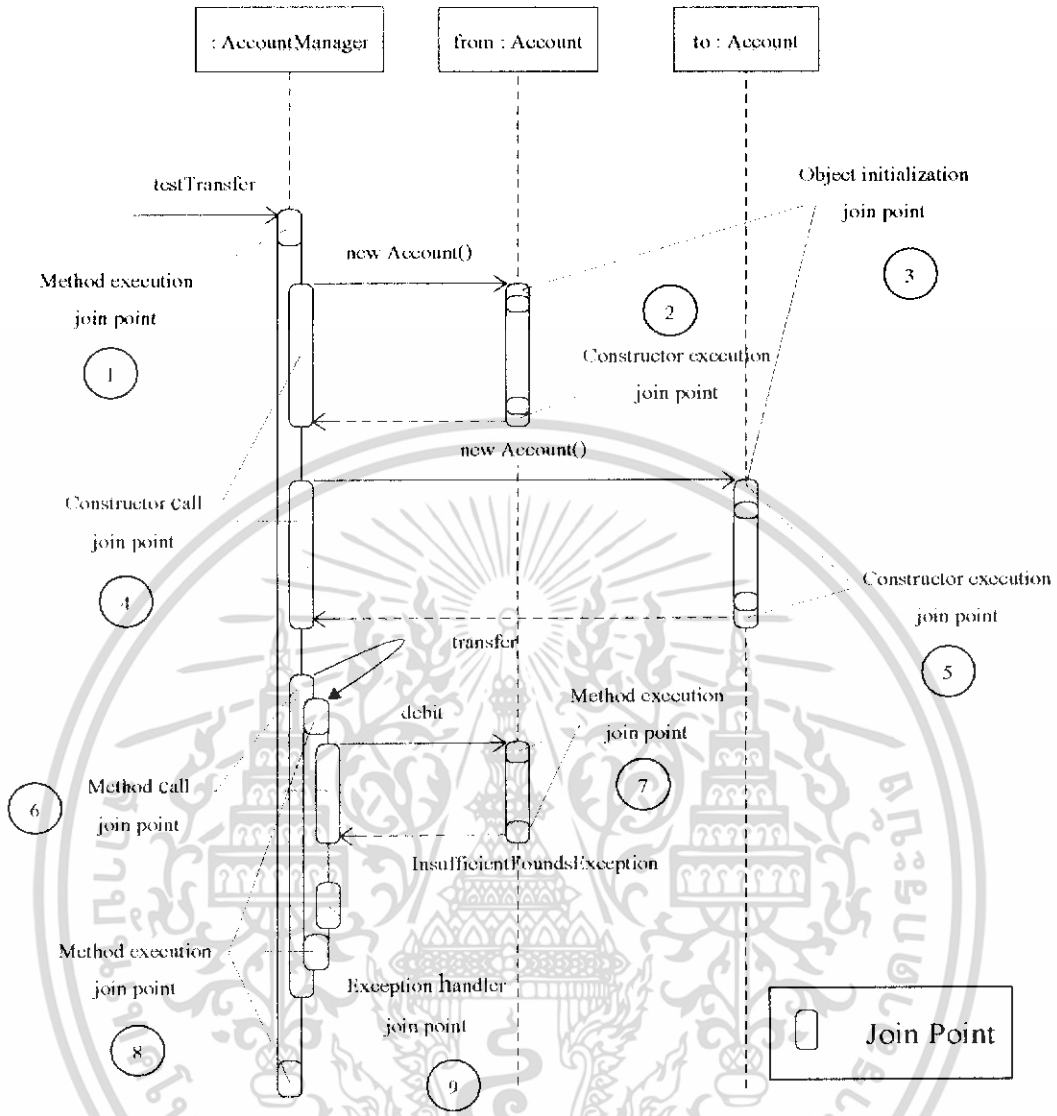
- การทำงานของเมท็อด (Method Execution) ซึ่งเป็นจอยพอยท์ที่เกิดขึ้นเมื่อเมท็อดที่ไม่ใช่คอนสทรัคเตอร์เมท็อดมีการทำงาน

- การเข้าถึงแอททริบิวต์ (Attribute Access) ซึ่งเป็นจอยพอยท์ที่เกิดขึ้นเมื่อมีการกำหนดหรือเรียกดูค่าของแอททริบิวต์

- การกำหนดค่าของแอททริบิวต์เริ่มต้นของอ็อบเจกต์ (Object Initialization) ซึ่งเป็นจอยพอยท์ที่เกิดขึ้นเมื่อมีการกำหนดค่าเริ่มต้นของแอททริบิวต์ในคอนสทรัคเตอร์เมท็อด

- การทำงานของการจัดการข้อผิดพลาด (Exception Handler Execution) เป็นจอยพอยท์ที่เกิดขึ้นเมื่อมีการทำงานในส่วนของการจัดการกับข้อผิดพลาดที่เกิดขึ้นในโปรแกรม

โดยที่จอยพอยท์ต่างๆสามารถแสดงโดยซีเควนซ์ไดอะแกรม ดังรูป 2.16



รูป 2.16 แสดงซีเควนซ์ไดอะแกรมที่แสดงให้เห็นถึงจุดต่างๆ ของจอยพอยท์

จากรูป 2.16 เป็นซีเควนซ์ไดอะแกรมที่แสดงการทำงานของเมท็อด testTransfer ของคลาส AccountManager โดยที่การทำงานของเมท็อด testTransfer ก่อให้เกิดจอยพอยท์ขึ้นในการทำงานของโปรแกรม โดยจอยพอยท์จะแทนด้วยสี่เหลี่ยมมนสีเทา ซึ่งในซีเควนซ์ไดอะแกรมนี้ประกอบด้วยจอยพอยท์ต่างๆ 6 จอยพอยท์ ได้แก่ การเรียกใช้คอนสทรัคเตอร์ การทำงานของคอนสตรัคเตอร์ การเรียกใช้เมท็อด การทำงานของเมท็อด การกำหนดค่าของ แอททริบิวต์เริ่มต้นของอ็อบเจกต์ และทำงานของการจัดการข้อผิดพลาด โดยที่จอยพอยท์แต่ละประเภทจะเกิดขึ้นในจุดต่างๆ ดังนี้

การเรียกใช้คอนสทรัคเตอร์ ซึ่งในซีเควนซ์ไดอะแกรมนี้เกิดการเรียกใช้คอนสทรัคเตอร์ตรงหมายเลข 4 และ 5 ของรูป 2.16 ซึ่งเป็นเหตุการณ์ของการสร้างอ็อบเจกต์ from และ to ของคลาส Account

การทำงานของคอนสทรัคเตอร์ ซึ่งในซีเควนซ์ไดอะแกรมนี้เกิดการทำงานของคอนสทรัคเตอร์ตรงหมายเลข 2 ของรูป 2.16 ซึ่งเป็นเหตุการณ์ที่คอนสทรัคเตอร์ของอ็อบเจกต์ from และ to ของคลาส Account มีการทำงาน

การกำหนดค่าของแอททริบิวต์เริ่มต้นของอ็อบเจกต์ ซึ่งในซีเควนซ์ไดอะแกรมนี้เกิดการกำหนดค่าของแอททริบิวต์เริ่มต้นของอ็อบเจกต์ตรงหมายเลข 3 ของรูป 2.16 ซึ่งเป็นเหตุการณ์ที่คอนสทรัคเตอร์ของอ็อบเจกต์ from และ to ของคลาส Account เริ่มมีการกำหนดค่าของแอททริบิวต์

การเรียกใช้เมท็อด ในซีเควนซ์ไดอะแกรมนี้เกิดการเรียกใช้เมท็อดตรงหมายเลข 6 ของรูป 2.16 ซึ่งเป็นเหตุการณ์ที่อ็อบเจกต์ AccountManaer เรียกเมท็อด transfer และอ็อบเจกต์ Account เรียกเมท็อด debit

การทำงานของเมท็อด ในซีเควนซ์ไดอะแกรมนี้เกิดการทำงานของเมท็อดตรงหมายเลข 1 7 และ 8 ของรูป 2.16 ซึ่งเป็นเหตุการณ์ที่เมท็อด testTranfer ของคลาส AccountManager เมท็อด transfer ของคลาส AccountManager และเมท็อด debit ของคลาส Account มีการทำงาน

การทำงานของการจัดการข้อผิดพลาด ในซีเควนซ์ไดอะแกรมนี้เกิดการทำงานของการจัดการข้อผิดพลาดตรงหมายเลข 9 ของรูป 2.16 ซึ่งเป็นเหตุการณ์ที่เมท็อด tranfer ของอ็อบเจกต์Account มีการทำงานในส่วนของการจัดการข้อผิดพลาดของโปรแกรมที่เกิดจากการทำงานของเมท็อด debit ของอ็อบเจกต์ Account

2. พอยท์คัต คือ โครงสร้างที่ใช้สำหรับเลือกเหตุการณ์ของจอยพอยท์เพื่อกำหนดการทำงานของโปรแกรมเมื่อเกิดเหตุการณ์ที่ได้เลือกไว้ขึ้น โดยที่การประกาศพอยท์คัตมีลักษณะดังนี้

|Modifier| pointcut Name (ParameterList) : PointcutExpression ;

ซึ่งมีรายละเอียดดังนี้

- Modifier เป็นการกำหนดการเข้าถึงหรือการมองเห็นของพอยท์คัต ซึ่งมีลักษณะเหมือนกันกับจาวา โดยโมดิฟายเออร์ของพอยท์คัตสามารถแบ่งออกได้ดังนี้ public private และ protected

- pointcut เป็นคีย์เวิร์ด (Key word) ที่ใช้ในการประกาศพอยท์คัต

- Name คือชื่อของพอยท์คัต
- ParameterList คือ กลุ่มของพารามิเตอร์ที่รับเข้ามาเพื่อทำงานในพอยท์คัต
- PointcutExpression เป็นส่วนที่ระบุว่าพอยท์คัตไปทำงานในเหตุการณ์ใดของจอยพอยท์

ตัวอย่างของพอยท์คัต

```
pointcut jdbcCall() ;
        call (* java.sql..(..)) ||
        call (* javax.sql..(..)) ;
```

จากตัวอย่าง ได้สร้างพอยท์คัตชื่อ jdbcCall โดยไม่รับพารามิเตอร์ใดเลย โดยพอยท์คัต jdbcCall นี้จะทำงานทุกครั้งที่มีการเรียกเมท็อดทุกเมท็อดที่อยู่ในแพคเกจของ java.sql หรือ javax.sql

3. แอดไวซ์ คือ ส่วนที่ระบุเนื้อหา (Context) ว่าจะให้โปรแกรมมีการทำงานอย่างไร ในแต่ละพอยท์คัตอาจจะเปรียบแอดไวซ์ได้ว่ามีลักษณะคล้ายกับเมท็อดในจาวา โดยที่แอดไวซ์แบ่งออกเป็น 3 ประเภท ได้แก่

- Before เป็นแอดไวซ์ที่จะทำงานก่อนที่เมท็อดของพอยท์คัตนั้นจะทำงาน
- After เป็นแอดไวซ์ที่จะทำงานหลังจากที่เมท็อดของพอยท์คัตนั้นทำงานเสร็จแล้ว
- Around เป็นแอดไวซ์ที่จะทำงานทั้งก่อนและหลังจากที่เมท็อดของพอยท์คัตนั้นทำงาน

การประกาศแอดไวซ์มีลักษณะดังนี้

```
Type ([ParameterList]) : PointcutExpression { Body }
```

ซึ่งมีรายละเอียดดังนี้

- Type คือประเภทของแอดไวซ์ซึ่งประกอบด้วย 3 ประเภทได้แก่ Before After และ Around
- ParameterList คือ กลุ่มของพารามิเตอร์ที่รับเข้ามาเพื่อทำงานในแอดไวซ์
- PointcutExpression เป็นส่วนที่ระบุชื่อพอยท์คัตเพื่อบอกกับแอดไวซ์ว่าจะมีการทำงานร่วมกับพอยท์คัตใด
- Body คือ การทำงานของแอดไวซ์

ตัวอย่างของแอดไวซ์

```
pointcut jdbcCall() ;
        call (* java.sql..(..)) ||
        call (* javax.sql..(..)) ;
```

```
before()  :  jdbcCall() { System.out.println("This is berfor advice") ; }
```

จากตัวอย่าง แอดไวซ์นี้จะทำงานทุกๆครั้งก่อนที่ เมท็อดทุกเมท็อดที่อยู่ในแพคเกจของ java.sql หรือ javax.sql จะถูกเรียกขึ้นมาทำงานโดยจะพิมพ์ข้อความว่า This is berfor advice ออกทางหน้าจอ

4. การประกาศประเภทข้อมูลร่วมกันระหว่างคลาส เป็นการประกาศประเภทข้อมูลได้แก่ แอททริบิวต์และเมท็อดภายในแอสเปค ทำให้แอสเปคสามารถเรียกใช้งานอ็อบเจกต์หรือเมท็อดของคลาสที่พัฒนาจากจาวาภายในแอสเปคได้ ซึ่งการประกาศมีลักษณะการประกาศเช่นเดียวกับจาวา

ตัวอย่างของการประกาศประเภทข้อมูลร่วมกันระหว่างคลาส

```
aspect Test  {
    private int count ;
    public void setCount(int c) {
        count = c;
    }
    public int getCount() {
        return c;
    }
}
```

จากตัวอย่าง มีการประกาศแอททริบิวต์ count และ เมท็อด setCount และ getCount ไว้ใช้งานภายในแอสเปค

5. แอสเปค คือ โมดูลมาตรฐานของการพัฒนาโปรแกรมเชิงแอสเปค เพื่อสนับสนุนการทำงานการแยกส่วนของเหตุการณ์ต่างๆที่ซ้ำซ้อนกันในโปรแกรม โดยที่ภายในแอสเปค จะประกอบด้วยพอยท์คัต แอดไวซ์และ การประกาศประเภทข้อมูลร่วมกันระหว่างคลาสซึ่งอาจเปรียบเทียบได้กับคลาสในจาวา การประกาศแอสเปคมีลักษณะดังนี้

```
[Modifier] aspect Name { Body }
```

ซึ่งมีรายละเอียดดังนี้

- Modifier เป็นการกำหนดการเข้าถึงหรือการมองเห็นของแอสเปค ซึ่งมีลักษณะเหมือนกันกับจาวา โดย Modifier ของแอสเปคสามารถแบ่งออกได้ดังนี้ public private และ protected
- aspect เป็นคีย์เวิร์ดที่ใช้ในการประกาศแอสเปค
- Name คือชื่อของแอสเปค
- Body คือ การทำงานต่างๆในแอสเปค

ตัวอย่างของแอสเปค

```
aspect Test  {
    private int count ;
    public void setCount(int c) {
        count = c;
    }
    public int getCount() {
        return c;
    }
    pointcut jdbcCall() :
        call (* java.sql..(..)) ||
        call (* javax.sql..(..)) ;
}
```

จากตัวอย่าง แสดงการสร้างโมดูลแอสเปคชื่อ Test ซึ่งภายในประกอบด้วยพอยท์คัตชื่อ jdbcCall และ การประกาศประเภทข้อมูลร่วมกันระหว่างคลาสโดยมีแอททริบิวต์ชื่อ count และเมท็อดชื่อ setCount และ getCount

2.4.4 ตัวอย่างโปรแกรมที่พัฒนาด้วยแอสเปคเจ

แอสเปคเจนั้นจะทำงานร่วมกับภาษาจาวา ซึ่งโปรแกรมที่พัฒนาขึ้นมานั้นจะแบ่งเป็น 2 ส่วน ได้แก่ ส่วนของจาวาและส่วนของแอสเปคเจ ตัวอย่างของโปรแกรมแสดงได้ดังรูป 2.17

```
class PlayList{
  private String name;
  private List<Song> songs = new ArrayList<Song>();
  public void play() {
    for (Song song : songs) {
      song.play();
    }
  }
}
class Song{
  private String name;
  public void play() {
    // play song
  }
  public void showLyrics(){
    // show lyrics
  }
}
aspect BillingPolicy {
  pointcut useTitle(Playable playable) :
     this(playable) && execution(public void Playable.play())
     ||  execution(public void Song.showLyrics());

  after(Playable playable) returning :   useTitle(playable) {
      BillingService.generateCharge(playable);
  }
}
```

รูป 2.17 แสดงตัวอย่างโปรแกรมแอสเปคเจ

จากรูป 2.17 โปรแกรมแบ่งออกเป็น 2 ส่วน ได้แก่ ส่วนของจาวา ซึ่งประกอบไปด้วยคลาส PlayList และ Song และอีกส่วนหนึ่งคือส่วนของแอสเปคเจ ซึ่งประกอบไปด้วยแอสเปค BillingPolicy โดยที่ 2 ส่วนนี้มีการทำงานร่วมกันคือ ภายในแอสเปค BillingPolicy ซึ่งแอสเปคนี้จะทำงานก็ต่อเมื่อเมท็อด play ในคลาส Playlist หรือเมท็อด play ในคลาส Song หรือเมท็อด showLyrids ในคลาส Song เกิดการทำงานของเมท็อดขึ้น และเมื่อเมท็อดทำงานเสร็จแอสเปคจะเข้ามาทำงาน ซึ่งในรูป 2.17 จะมีการทำงานคือเรียกเมท็อด generateCharge ของคลาส BillingService ให้ทำงาน

บทที่ 3
การทดสอบและเปรียบเทียบการพัฒนาโปรแกรม

ในโครงงานนี้ทำการทดสอบเปรียบเทียบวิธีการพัฒนาโปรแกรม 3 วิธี ได้แก่การพัฒนาโปรแกรมเชิงอ็อบเจกต์ คอมโพเนนท์และแอสเปค โดยการเลือกระบบขึ้นมา 1 ระบบ และทำการพัฒนาระบบด้วยวิธีการทั้ง 3 ซึ่งระบบที่จะนำมาใช้ในการทดสอบและเปรียบเทียบ คือ ระบบบริษัทประกันภัย โดยที่ระบบมีรายละเอียด ดังต่อไปนี้

ระบบประกันภัย จะมีการประกันภัยอยู่ 3 ประเภท ได้แก่ ประกันภัยชีวิต ประกันภัยที่อยู่อาศัย และ ประกันภัยรถยนต์ โดยการประกันภัยจะมี ราคาประกันภัย การผ่อนชำระเงิน ซึ่งสามารถเลือกจ่ายได้ตามตารางการผ่อนเงิน และค่าสินไหมทดแทนต่างกันในแต่ละครั้งที่ทำการประกันภัย โดยที่การประกันภัยแต่ละครั้งอาจมีค่าสินไหมทดแทนมากกว่า 1 ค่าสินไหมทดแทนได้ การประกันภัยชีวิตแต่ละครั้งจะทำได้ต่อบุคคลเดียวเท่านั้น การประกันภัยที่อยู่อาศัยแต่ละครั้งจะทำได้ต่อที่อยู่อาศัยเดียวเท่านั้น และการประกันภัยรถยนต์แต่ละครั้งจะทำได้ต่อรถยนต์คันเดียวเท่านั้น แต่ลูกค้า 1 คนสามารถทำประกันภัยได้มากกว่า 1 ครั้ง และสามารถทำประกันภัยได้หลายประเภท โดยมีการเก็บข้อมูลดังนี้

ลูกค้า ประกอบด้วย ชื่อ นามสกุล วันเกิด ที่อยู่ และ รหัสลูกค้า

ตารางการชำระเงิน ประกอบด้วย จำนวนงวดที่ต้องผ่อนชำระ จำนวนเงินที่ต้องผ่อนชำระแต่ละงวด และวันแรกที่ผ่อนชำระ

ค่าสินไหมทดแทน ประกอบด้วย วันที่เริ่มทำค่าสินไหมทดแทน จำนวนเงินค่าสินไหมทดแทน สถานะ รายละเอียด รหัสค่าสินไหมทดแทน

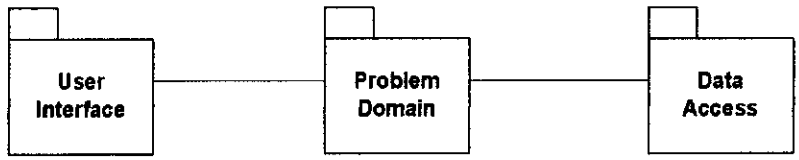
ที่อยู่อาศัย ประกอบด้วย บ้านเลขที่ ถนน เมือง รหัสไปรษณีย์ และ เบอร์โทรศัพท์

รถยนต์ ประกอบด้วย ยี่ห้อ รุ่น สี เลขทะเบียน และ ราคารถยนต์

การประกันภัย ประกอบด้วย วันเริ่มการประกันภัย วันสิ้นสุดการประกันภัย ราคาของการประกันภัย ตารางการชำระเงิน และ รหัสการประกันภัย และข้อมูลเฉพาะของการประกันภัยในแต่ละประเภท โดยแต่ละประเภทมีรายละเอียดดังนี้

- การประกันภัยชีวิต จะเก็บข้อมูลของผู้ทำประกันดังนี้ เป็นคนสูบบุหรี่หรือไม่ เป็นคนดื่มเครื่องดื่มแอลกอฮอล์หรือไม่ และเป็นคนสุภาพแข็งแรงหรือไม่
- การประกันภัยที่อยู่อาศัย จะเก็บข้อมูลของที่อยู่อาศัยดังนี้ มูลค่าของที่อยู่อาศัย และ ที่อยู่อาศัย
- การประกันภัยรถยนต์ จะเก็บข้อมูลของรถยนต์ดังนี้ เคยมีการเรียกร้องค่าสินไหมทดแทนหรือไม่ และจำนวนเงินส่วนเกิน

จากนั้นทำการออกแบบแพคเกจสำหรับการพัฒนาโปรแกรม โดยแพคเกจนั้นจะแบ่งออกเป็น 3 ส่วน ได้แก่ ส่วนการติดต่อกับผู้ใช้ ส่วนความสัมพันธ์ของระบบ และส่วนของการติดต่อกับฐานข้อมูล แสดงได้ดังรูป 3.1



รูป 3.1 แสดงแพคเกจและความสันพันธ์กันระหว่างแพคเกจ

จากรูป 3.1 แสดงแพคเกจที่จะใช้ในการพัฒนาโปรแกรม ซึ่งถูกแบ่งออกตามหน้าที่การทำงาน และแต่ละแพคเกจจะมีการติดต่อกันดังรูป 3.1 คือ แพคเกจส่วนการติดต่อกับผู้ใช้จะรับหรือส่งข้อมูลผ่านทางแพคเกจความสัมพันธ์ของระบบเท่านั้น และแพคเกจความสัมพันธ์ของระบบจะทำหน้าที่ติดต่อกับแพคเกจส่วนของการติดต่อกับฐานข้อมูล ในการอ่านข้อมูลหรือเก็บข้อมูลในฐานข้อมูล

รายละเอียดของแต่ละแพคเกจมีดังต่อไปนี้

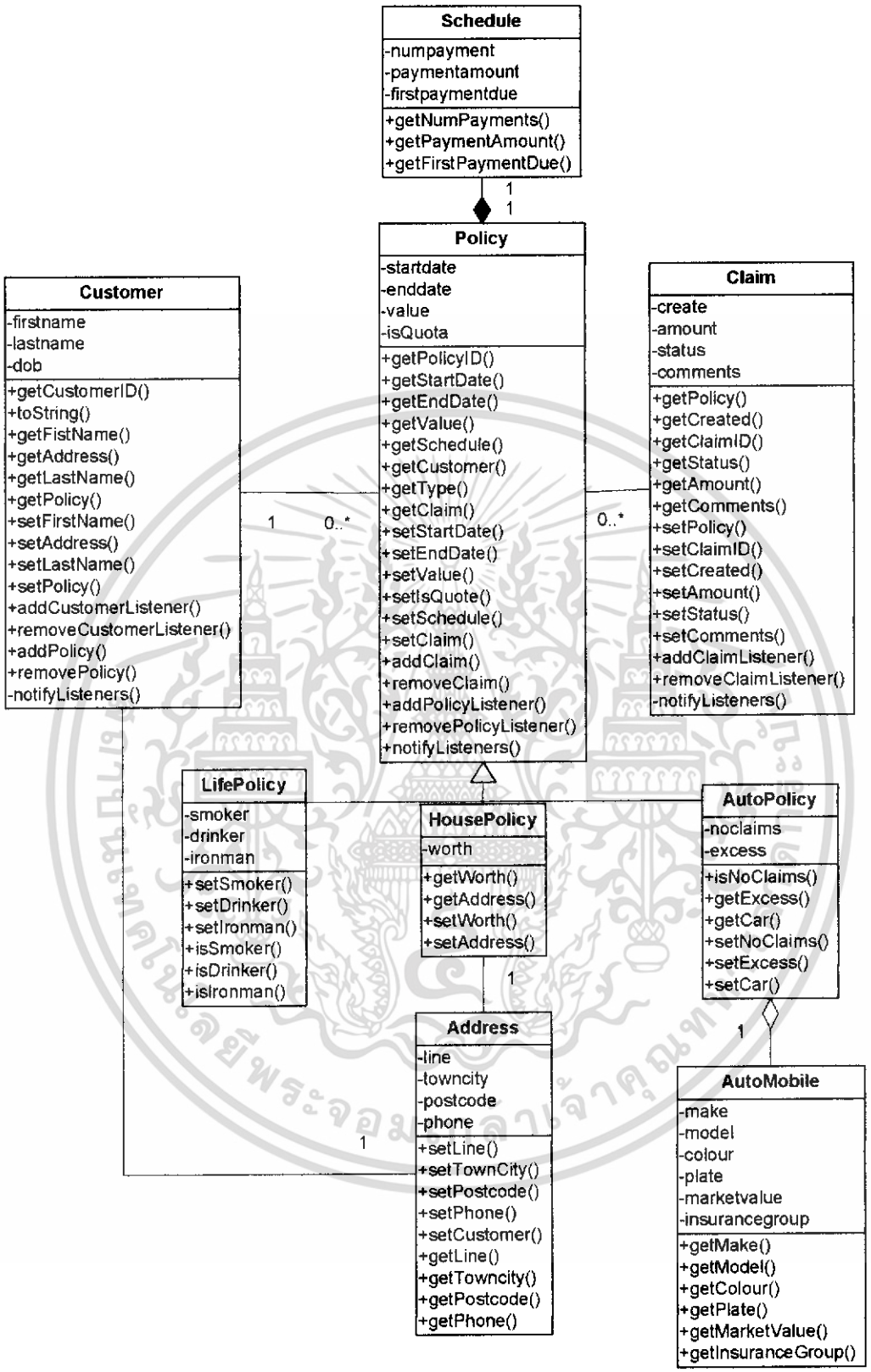
1. แพคเกจส่วนการติดต่อกับผู้ใช้ ในแพคเกจนี้จะประกอบไปด้วยคลาสที่ทำหน้าที่ในการติดต่อกับผู้ใช้ โดยคลาสที่สำคัญที่จะนำมาแสดงการเปรียบเทียบในแพคเกจนี้ มีดังต่อไปนี้

- คลาส AddHousePolicy เป็นคลาสที่ทำหน้าที่เพิ่มข้อมูลของการประกันบ้าน
- คลาส AddClaim เป็นคลาสที่ทำหน้าที่เพิ่มข้อมูลของค่าสินไหมทดแทน
- คลาส NewCustomer เป็นคลาสที่ทำหน้าที่เพิ่มข้อมูลของลูกค้า
- คลาส FindAndEditCustomer เป็นคลาสที่ทำหน้าที่ค้นหาและแก้ไขข้อมูลของลูกค้า

2. แพคเกจการติดต่อกับฐานข้อมูล แพคเกจนี้จะประกอบไปด้วยคลาสที่ทำหน้าที่ในการติดต่อกับฐานข้อมูล โดยคลาสที่สำคัญที่จะนำมาแสดงการเปรียบเทียบในแพคเกจนี้ มีดังต่อไปนี้

- คลาส CustomerDao เป็นคลาสที่ทำหน้าที่เป็นตัวกลางในการจัดการข้อมูลระหว่างคลาส Customer และฐานข้อมูล

3. แพคเกจความสัมพันธ์ของระบบ แพคเกจนี้จะประกอบไปด้วยคลาสที่แสดงถึงความสัมพันธ์ที่เกิดจากระบบประกันภัย ซึ่งสามารถนำมาเขียนเป็นคลาสไดอะแกรมได้ดังรูป 3.2



รูป 3.2 แสดงคลาสไดอะแกรมของระบบประกัน

จากรูป 3.2 คลาสไดอะแกรมของระบบประกันซึ่งประกอบไปด้วย 9 คลาส ได้แก่ คลาส Customer Schedule Policy Claim LifePolicy HousePolicy AutoPolicy Address และ AutoMobile ซึ่งแต่ละคลาสเป็นต้นแบบของอ็อบเจกต์ที่สามารถเกิดขึ้นได้ในระบบการประกันภัย

3.1 การพัฒนาโปรแกรมโดยวิธีการเชิงอ็อบเจกต์

จากการพัฒนาแอพพลิเคชันของระบบประกันภัยด้วยวิธีการเชิงอ็อบเจกต์พบว่า เกิดปัญหาขึ้นในการพัฒนาหลายประการตามที่กล่าวไว้ในบทที่ 2 ซึ่งปัญหาต่างๆ สามารถแสดงได้ดังนี้

1. โมดูลขาดความเป็นอิสระ ในระบบประกันภัยคลาสแต่ละคลาสนั้นผูกติดกัน ทำให้โปรแกรมขาดความเป็นอิสระ แสดงได้ดังรูป 3.3 ซึ่งเป็นโค้ดของเมท็อด addCustomer ในคลาส NewCustomer โดยที่เมท็อดนี้ทำหน้าที่เพิ่มข้อมูลของลูกค้าและที่อยู่ลงในฐานข้อมูล ซึ่งโค้ดภายในเมท็อดนี้จะมีการสร้างอ็อบเจกต์ของลูกค้าและที่อยู่ เพิ่มเก็บข้อมูลจากที่ผู้ใช้ได้กรอกเข้ามาทางส่วนติดต่อกับผู้ใช้ ซึ่งคลาส NewCustomer นั้น มีการทำงานคนละหน้าที่กับคลาส Customer และ Address ทำให้เกิดโมดูลขาดความเป็นอิสระ

```
private void addCustomer() {

    if (tCustomerId.getText().equals("") || tName.getText().equals("") ||
     tSurName.getText().equals("") || tAddress.getText().equals("") ||
     tTown.getText().equals("") || tPostcode.getText().equals("") ||
     tPhone.getText().equals(""))

     JOptionPane.showMessageDialog(this,
     "Please Enter All Data","Error",JOptionPane.ERROR_MESSAGE);
    else {
     if(JOptionPane.showConfirmDialog(null,"Are you want to add Customer"
             ,"Confirm",JOptionPane.YES_NO_OPTION) == 0)
     {
         Address address = new Address(null,tAddress.getText(),
                           tTown.getText(),tPostcode.getText(),tPhone.getText()) ;

         Customer customer = new Customer(tCustomerId.getText(),tName.getText(),
                           tSurName.getText(),address) ;

         address.setCustomer(customer) ;

         try {
             CustomerDao.connect() ;
             customer.addCustomerListener();
             JOptionPane.showMessageDialog(this,"Add customer already") ;
             CustomerDao.close() ;
             clearInput() ;
         }
         catch(CheckDataException e)
         {
             JOptionPane.showMessageDialog(this,e.toString(),"error",JOptionPane.INFORMATI
         }
     }
    }
}
```

รูป 3.3 แสดงโค้ดที่ทำให้คลาสขาดความเป็นอิสระที่เกิดขึ้นในคลาส NewCustomer

จากรูป 3.3 แสดงให้เห็นถึงปัญหาโมดูลขาดความเป็นอิสระ เนื่องจากคลาส NewCutomer นั้นเป็นคลาสที่มีการทำงานอยู่ในส่วนของการติดต่อกับผู้ใช้ แต่ต้องทำการสร้างอ็อบเจกต์ Address และ Customer ซึ่งเป็นคลาสในส่วนของความสัมพันธ์ของระบบประกันภัย ซึ่งทั้ง 2 ส่วนนี้เป็นการทำงานคนละหน้าที่ ดังนั้นคลาสไม่ควรที่จะผูกติดกัน

2. โปรแกรมที่มีลักษณะการสืบทอดคลาส ในระบบประกันภัยคลาสของการประกันนั้น คลาสของการประกันภัยมีลักษณะเป็นแบบการสืบทอดคลาส และในซุปเปอร์คลาสนั้นมีเมท็อด notifyListeners ที่ถูกเรียกใช้ทั้งในซุปเปอร์คลาสเองและจากซับคลาส ทำให้เกิดการกระจายของโปรแกรมขึ้น ดังรูป 3.4 3.5 3.6 และ 3.7

```
public void setValue(double value) {
    this.value = value;
    notifyListeners();
}

public double getValue() {
    return value;
}

public void setIsQuote(boolean isQuote) {
    this.isQuote = isQuote;
    notifyListeners();
}

public boolean isQuote() {
    return isQuote;
}

public void setSchedule(Schedule schedule) {
    this.schedule = schedule;
    notifyListeners();
}

protected void notifyListeners() {
  policyDao.update(this);
}
```

รูป 3.4 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส Policy

```
public String toString() {
    return getPolicyID() + " [Life]";
}
public void setSmoker(boolean smoker) {
    this.smoker = smoker;
    notifyListeners();
}
public boolean isSmoker() {
    return smoker;
}
public void setDrinker(boolean drinker) {
    this.drinker = drinker;
    notifyListeners();
}
public boolean isDrinker() {
    return drinker;
}
```

รูป 3.5 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส LifePolicy

```
public String toString() {
    return getPolicyID() + " [House]";
}
public void setWorth(double worth) {
    this.worth = worth;
    notifyListeners();
}
public double getWorth() {
    return worth;
}
public void setAddress(Address address) {
    this.address = address;
    notifyListeners();
}
public Address getAddress() {
    return address;
}
```

รูป 3.6 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส HousePolicy

```
public void setNoClaims(boolean noClaims) {
    this.noClaims = noClaims;
    notifyListeners();
}
public boolean isNoClaims() {
    return noClaims;
}
public void setExcess(double excess) {
    this.excess = excess;
    notifyListeners();
}
public double getExcess() {
    return excess;
}
public void setCar(Automobile car) {
    this.car = car;
    notifyListeners();
}
public Automobile getCar() {
    return car;
}
```

รูป 3.7 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส AutoPolicy

จากรูป 3.4 3.5 3.6 และ 3.7 เป็นโค้ดส่วนหนึ่งของคลาส Policy HousePolicy LifePolicy และ AutoPolicy ตามลำดับ โดยที่คลาสทุกๆคลาสมีการเรียกใช้เมท็อด notifyListeners ซึ่งเป็นเมท็อดของคลาส Policy ทำให้เกิดการทำงานที่ซ้ำซ้อนกัน ก่อให้เกิดปัญหาครอสคัตติ้งคอนเซินขึ้นในคลาส

นอกจากปัญหาที่ได้กล่าวมานี้ โค้ดลักษณะนี้ยังก่อให้เกิดปัญหาในการปรับปรุงและแก้ไขโปรแกรมอีกด้วย เช่น ถ้าต้องการเพิ่มการประกันคอมพิวเตอร์เข้าไปในระบบประกันภัย โค้ดของคลาสการประกันคอมพิวเตอร์ก็จะเกิดปัญหาของการกระจายของเมท็อด notifyListeners ขึ้นเช่นกัน ดังรูป 3.8

```
public ComputerPolicy (Customer c,String policyId) {
    super(c,policyId);
    this.type = PolicyType.COMPUTER;
}
public ComputerPolicy(String startDate,
                      String endDate,
                      double value,
                      boolean isQuote,
                      Schedule schedule,
                      Customer customer,
                      Computer computer,
                      String policyId) {
    super(startDate,endDate,value,isQuote,schedule,customer,policyId);
    this.computer = computer;
    this.type = PolicyType.COMPUTER;
}
public void setComputer(Computer computer) {
    this.computer = computer;
    notifyListeners();
}
public Computer getComputer() {
    return computer;
}
```

รูป 3.8 แสดงการกระจายของเมท็อด notifyListeners ที่เกิดขึ้นภายในคลาส ComputerPolicy

จากรูป 3.8 เป็นโค้ดส่วนหนึ่งของคลาส ComputerPolicy โดยจะเห็นว่าถ้าระบบมีการเพิ่มการประกันภัยใหม่เข้ามาก็จะก่อให้เกิดปัญหาการกระจายของโค้ดมากขึ้นตามมาด้วย

นอกจากนี้ถ้าในอนาคตโปรแกรมมีการเปลี่ยนแปลงโค้ดของเมท็อด notifyListeners โดยการรับพารามิเตอร์ 1 ตัว จะเกิดปัญหาในการแก้ไขโปรแกรม เพราะต้องแก้ไขโปรแกรมทุกที่ที่มีการเรียกใช้เมท็อด notifyListeners เช่นถ้าเมท็อด notifyListeners ต้องการพารามิเตอร์ 1 ตัว คือ ค่าที่บอกว่าจะอัพเดตข้อมูลหรือไม่ มีผลทำให้ต้องแก้ไขโปรแกรมทุกที่ที่มีการเรียกใช้เมท็อด notifyListeners ดังรูป 3.9

```
public void setValue(double value,boolean isUpdate) {
    this.value = value;
    notifyListeners(isUpdate);
}

public double getValue() {
    return value;
}

public void setIsQuote(boolean isQuoteboolean isUpdate) {
    this.isQuote = isQuote;
    notifyListeners(isUpdate);
}

public boolean isQuote() {
    return isQuote;
}

public void setSchedule(Schedule schedule,boolean isUpdate) {
    this.schedule = schedule;
    notifyListeners(isUpdate);
}

protected void notifyListeners(boolean isUpdate) {
    policyDao.update(this,isUpdate) ;
}
```

รูป 3.9 แสดงการแก้ไขโค้ดเมื่อเมท็อด notifyListeners เกิดการเปลี่ยนแปลง ที่เกิดขึ้นภายในคลาส ComputerPolicy

จากรูป 3.9 แสดงให้เห็นถึงการแก้ไขโค้ดเมื่อเมท็อด notifyListeners เกิดการเปลี่ยนแปลง จะเห็นว่าโปรแกรมเมอร์ต้องแก้ไขโค้ดทุกๆที่ที่มีการเรียกใช้เมท็อด notifyListeners ซึ่งในคลาส LifePolicy HousePolicy AutoPolicy และ ComputerPolicy ก็ต้องแก้ไขโค้ดลักษณะเดียวกันกับคลาส ComputerPolicy

3. การติดต่อกับฐานข้อมูล เมื่อโปรแกรมต้องการใช้ข้อมูลที่เก็บอยู่ในฐานข้อมูล โปรแกรมจะต้องทำการติดต่อกับฐานข้อมูล และเข้าใช้งานฐานข้อมูล เมื่อทำงานเสร็จก็ต้องทำการยกเลิกการติดต่อกับฐานข้อมูล ซึ่งในระบบประกันภัยนั้นมีคลาสที่มีการติดต่อกับฐานข้อมูลอยู่หลายคลาส เช่น คลาส NewCustomer และคลาส FindAndEditCustomer ทำให้โค้ดของการใช้งานฐานข้อมูลกระจายไปอยู่ในหลายๆที่ของโปรแกรม ดังรูป 3.10 3.11 3.12 และ 3.13

```
private void addCustomer() {

    if (tCustomerId.getText().equals("") || tName.getText().equals("") ||
        (tSurName.getText().equals("") || tAddress.getText().equals("") ||
        tTown.getText().equals("") || tPostcode.getText().equals("") ||
        tPhone.getText().equals(""))

        JOptionPane.showMessageDialog(this,
        "Please Enter All Data", "Error",JOptionPane.ERROR_MESSAGE);
    else {
        if(JOptionPane.showConfirmDialog(null,"Are you want to add Customer"
                ,"Confirm",JOptionPane.YES_NO_OPTION) == 0)
        {
            Address address = new Address(null,tAddress.getText(),
                        tTown.getText(),tPostcode.getText(),tPhone.getText());

            Customer customer = new Customer(tCustomerId.getText(),tName.getText(),
                        tSurName.getText(),address);

            address.setCustomer(customer);

            try {
                CustomerDao.connect();
                customer.addCustomerListener();
                JOptionPane.showMessageDialog(this,"Add Customer already");
                CustomerDao.close();
                clearInput();
            }
            catch(CheckDataException e)
            {
                JOptionPane.showMessageDialog(this,e.toString(),"error",JOptionPane.INFORMATI
            }
        }
    }
}
```

รูป 3.10 แสดงการกระจายของโค้ดที่เกิดขึ้นใน
เมท็อด addCustomer ของคลาส NewCustomer

```
private void searchCustomer() {

    try
    {
        CustomerDao.connect() ;
        Customer sCustomer = CustomerDao.search(tSearchCus.getText());
        CustomerDao.close() ;
```

รูป 3.11 แสดงการกระจายของโค้ดที่เกิดขึ้นใน
เมท็อด searchCustomer ของคลาส FindAndEditCustomer

```
private void updateCustomer() {

    if (JOptionPane.showConfirmDialog(null,"Are you sure to edit data"
            ,"Confirm",JOptionPane.YES_NO_OPTION) == 0) {

        try
        {
            CustomerDao.connect() ;

            Customer customer = CustomerDao.search(tSearchCus.getText());

            Address address = new Address(customer,tAddress.getText(),
                                          tTown.getText(),tPostcode.getText(),
                                          tPhone.getText()) ;

            if (!customer.getFirstName().equals(tName.getText()))
                customer.setFirstName(tName.getText()) ;
            if (!customer.getLastName().equals(tSurName.getText()))
                customer.setLastName(tSurName.getText()) ;
            if (!customer.getAddress().equals(address))
                customer.setAddress(address) ;

            CustomerDao.close() ;
```

รูป 3.12 แสดงการกระจายของโค้ดที่เกิดขึ้นใน
เมท็อด updateCustomer ของคลาส FindAndEditCustomer

```
private void deleteCustomer() {
    if (JOptionPane.showConfirmDialog(null,"Are you sure to delete data"
            ,"Confirm",JOptionPane.YES_NO_OPTION) == 0) {

        try
        {
            CustomerDao.connect();
            Customer customer = CustomerDao.search(tSearchCus.getText());
            customer.removeCustomerListener();
            CustomerDao.close();
            JOptionPane.showMessageDialog(this,"Delete customer already") ;
        }
        catch(NotFoundException e)
        {
            JOptionPane.showMessageDialog(this,e.toString(),"error",JOptionPane.INFORMATION_MESSAGE);
        }

        clearInput();
    }
}
```

รูป 3.13 แสดงการกระจายของโค้ดที่เกิดขึ้นใน
เมท็อด deleteCustomer ของคลาส FindAndEditCustomer

จากรูป 3.10 เป็นโค้ดส่วนหนึ่งของคลาส NewCustomer รูป 3.11 3.12 และ 3.13 เป็นโค้ดส่วนหนึ่งของคลาส FindAndEditCustomer โดยที่คลาสทุกๆคลาสมีการติดต่อกับฐานข้อมูล โดยในรูปจะเห็นว่าการทำงานมีการเรียกใช้เมท็อด connect ที่ทำการเชื่อมต่อกับฐานข้อมูลและเมท็อด close ที่ทำการยกเลิกการติดต่อกับฐานข้อมูลของคลาส CustomerDao ซึ่งก่อให้เกิดปัญหาครอสคัตติ้งคอนเซินขึ้นในคลาส

นอกจากปัญหาที่ได้กล่าวมานี้ โค้ดลักษณะนี้ยังก่อให้เกิดปัญหาในการปรับปรุงและแก้ไขโปรแกรมเช่นเดียวกับโค้ดที่มีการสืบทอดคลาส เช่น ถ้าต้องการเพิ่มการประกันคอมพิวเตอร์เข้าไปในระบบประกันภัย ก็จะต้องเพิ่มคลาสส่วนของการติดต่อกับผู้ใช้ในการเพิ่มการประกัน

คอมพิวเตอร์เข้ามาด้วย ซึ่งก็จะก่อให้เกิดการกระจายของโค้ดในการติดต่อและยกเลิกการติดต่อฐานข้อมูล เช่นเดียวกับคลาส NewCustomer และ คลาส FindAndEditCustomer

4. การตรวจจับข้อผิดพลาด การพัฒนาโปรแกรมเชิงอ็อบเจกต์นั้นมีการตรวจจับข้อผิดพลาดของโปรแกรมโดยใช้คำสั่ง try catch แต่บางครั้งข้อผิดพลาดที่เกิดขึ้นในหลายๆจุดโปรแกรมอาจจะมีการจัดการกับข้อผิดพลาดที่เกิดขึ้นเหมือนๆกัน ดังนั้นทำให้การตรวจจับข้อผิดพลาดของการทำงานเกิดการตรวจจับที่ซ้ำซ้อนกัน ซึ่งในระบบประกันภัยมีการตรวจจับข้อผิดพลาดที่ซ้ำซ้อนกัน ดังรูป 3.14 3.15 3.16 และ 3.17

```
private void addCustomer() {

    if (tCustomerId.getText().equals("") || tName.getText().equals("") ||
        tSurName.getText().equals("") || tAddress.getText().equals("") ||
        tTown.getText().equals("") || tPostcode.getText().equals("") ||
        tPhone.getText().equals(""))

        JOptionPane.showMessageDialog(this,
        "Please Enter All Data","Error",JOptionPane.ERROR_MESSAGE);
    else {
        if(JOptionPane.showConfirmDialog(null,"Are you want to add Customer"
                ,"Confirm",JOptionPane.YES_NO_OPTION) == 0)
        {
            Address address = new Address(null,tAddress.getText(),
                            tTown.getText(),tPostcode.getText(),tPhone.getText()) ;

            Customer customer = new Customer(tCustomerId.getText(),tName.getText(),
                            tSurName.getText(),address) ;

            address.setCustomer(customer) ;

            try {
                CustomerDao.connect() ;
                customer.addCustomerListener();
                JOptionPane.showMessageDialog(this,"Add customer already") ;
                CustomerDao.connect() ;
                clearInput() ;
            }
            catch(CheckDataException e)
            {
                JOptionPane.showMessageDialog(this,e.toString(),"error",JOptionPane.INFORMATION_MESSAGE);
            }
        }
    }
}
```

รูป 3.14 แสดงการจัดการกับข้อผิดพลาดของโปรแกรมใน
เมท็อด addCustomer ของคลาส NewCustomer

```
private void updateCustomer() {

    if (JOptionPane.showConfirmDialog(null,"Are you sure to edit data"
            ,"Confirm",JOptionPane.YES_NO_OPTION) == 0) {

        try
        {
            CustomerDao.connect() ;

            Customer customer = CustomerDao.search(tSearchCus.getText());

            Address address = new Address(customer,tAddress.getText(),
                                          tTown.getText(),tPostcode.getText(),
                                          tPhone.getText()) ;

            if (!customer.getFirstName().equals(tName.getText()))
                customer.setFirstName(tName.getText()) ;
            if (!customer.getLastName().equals(tSurName.getText()))
                customer.setLastName(tSurName.getText()) ;
            if (!customer.getAddress().equals(address))
                customer.setAddress(address) ;

            CustomerDao.connect() ;

            JOptionPane.showMessageDialog(this,"Update customer already") ;
        }
        catch(NotFoundException e)
        {
            JOptionPane.showMessageDialog(this,e.toString(),"error",JOptionPane.INFORMATION_MESSAGE);
        }
    }
}
```

รูป 3.15 แสดงการจัดการกับข้อผิดพลาดของโปรแกรมใน

เมท็อด updateCustomer ของคลาส FindAndEditCustomer

```
private void addHousePolicy() {
    if (tCusId.getText().equals("") || tsYear.getText().equals("") || teYear.getText().equals("") ||
        tPolicyID.getText().equals("") || tValue.getText().equals("") || tFirstPay.getText().equals("") ||
        tPayAmount.getText().equals("") || tNumPay.getText().equals("") || tWorth.getText().equals("") ||
        tAddress.getText().equals("") || tTown.getText().equals("") || tPostcode.getText().equals("") ||
        tPhone.getText().equals(""))

        JOptionPane.showMessageDialog(this,
        "Please Enter All Data","Error",JOptionPane.ERROR_MESSAGE);
    else {
        try {
            CustomerDao.connect() ;
            PolicyDao.connect() ;
            ClaimDao.connect() ;
            String sDate = tsYear.getText()+"/"+(csMonth.getSelectedIndex()+1)+"/"+(csDate.getSelectedIndex()+1)
            String eDate = teYear.getText()+"/"+(ceMonth.getSelectedIndex()+1)+"/"+(ceDate.getSelectedIndex()+1)

            Schedule schedule = new Schedule(Integer.parseInt(tNumPay.getText()),
                                             Double.parseDouble(tPayAmount.getText()),
                                             tFirstPay.getText()) ;

            Customer customer = CustomerDao.search(tCusId.getText()) ;

            HousePolicy housePolicy = new HousePolicy(sDate,eDate,Double.parseDouble(tValue.getText()),
                                                      true,schedule,customer,Double.parseDouble(tWorth.getText()),
                                                      customer.getAddress(),tPolicyID.getText()) ;

            housePolicy.addPolicyListener();
            JOptionPane.showMessageDialog(this,"Add HousePolicy already") ;
            CustomerDao.close() ;
            PolicyDao.close() ;
            ClaimDao.close() ;
            clearInput() ;
        }
        catch(CheckDataException e)
        {
            JOptionPane.showMessageDialog(this,e.toString(),"error",JOptionPane.INFORMATION_MESSAGE);
        }
        catch(NotFoundException ee)
        {
            JOptionPane.showMessageDialog(this,ee.toString(),"error",JOptionPane.INFORMATION_MESSAGE);
        }
    }
}
```

รูป 3.16 แสดงการจัดการกับข้อผิดพลาดของโปรแกรมใน

เมท็อด addHousePolicy ของคลาส AddHousePolicy

```
private void addClaim() {

    if (tPolicyID.getText().equals("") || tClaimID.getText().equals("") || tAmount.getText().equals("") ||
        tStatus.getText().equals("") || tComments.getText().equals(""))

        JOptionPane.showMessageDialog(this,
        "Please Enter All Data","Error",JOptionPane.ERROR_MESSAGE);
    else {
        try {
            CustomerDao.connect();
            Policy.connect();
            ClaimDao.connect();
            Policy policy = PolicyDao.search(tPolicyID.getText());

            Claim claim = new Claim(policy,tClaimID.getText(),addDate(),
                                    Double.parseDouble(tAmount.getText()),
                                    tStatus.getText(),tComments.getText()) ;

            policy.addClaim(claim) ;

            if(policy.getType().toString().equals("auto"))
                {((AutoPolicy)policy).setNoClaims(false) ;}

            claim.addClaimListener() ;
            JOptionPane.showMessageDialog(this,"Add Claim already") ;
            CustomerDao.close();
            Policy.close();
            ClaimDao.close();
            clearInput() ;
        }
        catch(CheckDataException e)
        {
            JOptionPane.showMessageDialog(this,e.toString(),"error",JOptionPane.INFORMATION_MESSAGE);
        }
        catch(NotFoundException ee)
        {
            JOptionPane.showMessageDialog(this,ee.toString(),"error",JOptionPane.INFORMATION_MESSAGE);
        }
    }
}
```

รูป 3.17 แสดงการจัดการกับข้อผิดพลาดของโปรแกรมใน
เมท็อด addClaim ของคลาส AddClaim

จากรูป 3.14 3.15 3.16 และ 3.17 เป็นโค้ดส่วนหนึ่งของคลาส NewEditCustomer FindAndEditCustomer AddHousePolicy และ AddClaim ตามลำดับ โดยทุกคลาสนั้นมีการทำงานของการจัดการกับข้อผิดพลาดที่เกิดขึ้นเหมือนๆกัน ทำให้เกิดการทำงานที่ซ้ำซ้อนกัน

นอกจากปัญหาที่ได้กล่าวมานี้ โค้ดลักษณะนี้ยังก่อให้เกิดปัญหาในการปรับปรุงและแก้ไขโปรแกรมเช่นเดียวกับโค้ดที่มีการสืบทอดคลาส เช่น ถ้าต้องการเพิ่มการประกันคอมพิวเตอร์เข้าไปในระบบประกันภัย ก็จะต้องเพิ่มคลาสส่วนของการติดต่อกับผู้ใช้ในการเพิ่มการประกันคอมพิวเตอร์เข้ามาด้วย ซึ่งก็จะก่อให้เกิดการกระจายของโค้ดในส่วนของการจัดการกับข้อผิดพลาดของโปรแกรม เช่นเดียวกับคลาส NewCustomer คลาส FindAndEditCustomer คลาส AddHousePolicy และคลาส AddClaim

3.2 การพัฒนาโปรแกรมโดยวิธีการเชิงคอมโพเนนท์

จากการพัฒนาระบบประกันภัยด้วยวิธีการเชิงคอมโพเนนท์นั้น สามารถช่วยแก้ไขปัญหาของความเป็นอิสระของโมดูล โดยใช้วิธีการส่งเหตุการณ์ออกไปเพื่อติดต่อกับคอมโพเนทท์อื่น ซึ่งในจาวาบีนนั้นจะส่งเหตุการณ์ออกไปโดยการใช้เมท็อด firePropertyChange ของคลาส PropertyChangrSupport โดยที่เมท็อด firePropertyChange นั้นรับพารามิเตอร์ 3 ตัว โดยที่พารามิเตอร์ตัวแรกคือชื่อของเหตุการณ์ ตัวที่สองคือค่าแอททริบิวต์เก่าที่คอมโพเนนท์เก็บไว้ และตัวสุดท้ายคือค่าแอททริบิวต์ใหม่ที่คอมโพเนนท์ได้รับ และเมท็อด firePropertyChange นั้นจะทำ

การส่งเหตุการณ์ออกไปให้คอมโพเนนท์อื่นก็ต่อเมื่อค่าเก่าและค่าใหม่นั้นแตกต่างกัน ถ้าค่าเก่าและค่าใหม่เป็นค่าเดียวกันเมท็อด firePropertyChange จะไม่ทำการส่งเหตุการณ์ออกไป

การทำงานนี้ช่วยแก้ปัญหาของระบบประกันภัยได้ ดังรูป 3.18 ซึ่งเป็นรูปที่แสดงส่วนหนึ่งของโค้ดระบบประกันภัย โดยในรูปแสดงเมท็อด addCustomer ของคลาส NewCustomer ซึ่งเป็นโค้ดที่ทำหน้าที่เพิ่มข้อมูลลูกค้า ซึ่งคลาส NewCustomer นี้ทำหน้าที่อยู่ในส่วนของการติดต่อกับผู้ใช้ และเมื่อต้องการส่งข้อมูลที่ได้จากผู้ใช้ไปไว้ในคลาสของ Customer และคลาส Address คลาส NewCustomer ก็ทำการส่งเหตุการณ์ออกไปให้กับคลาสทั้ง 2

```
private void addCustomer() {

    if (tCustomerId.getText().equals("") || tName.getText().equals("") ||
        tSurName.getText().equals("") || tAddress.getText().equals("") ||
        tTown.getText().equals("") || tPostcode.getText().equals("") ||
        tPhone.getText().equals(""))

        JOptionPane.showMessageDialog(this,
        "Please Enter All Data","Error",JOptionPane.ERROR_MESSAGE);
    else {
        if(JOptionPane.showConfirmDialog(null,"Are you want to add Customer"
                ,"Confirm",JOptionPane.YES_NO_OPTION) == 0)
        {

            Vector v = new Vector() ;
            v.add(tAddress.getText()) ;
            v.add(tTown.getText()) ;
            v.add(tPostcode.getText()) ;
            v.add(tPhone.getText()) ;

            pcs.firePropertyChange("New Adr",new Integer(0),v) ;

            v = new Vector() ;
            v.add(tCustomerId.getText()) ;
            v.add(tName.getText()) ;
            v.add(tSurName.getText()) ;

            pcs.firePropertyChange("New Cus",new Integer(0),v) ;
            pcs.firePropertyChange("add Cus",new Integer(0),new Integer(1)) ;

            JOptionPane.showMessageDialog(this,"Add customer already") ;
            clearInput() ;

        }
    }
}
```

รูป 3.18 แสดงการส่งเหตุการณ์ออกไปให้กับคอมโพเนนท์อื่นโดยใช้เมท็อด firePropertyChange ในคลาส NewCustomer

จากรูป 3.18 คลาส NewCustomer ส่งเหตุการณ์ออกไปให้กับคอมโพเนนท์ที่เป็นผู้รับเหตุการณ์โดยใช้เมท็อด firePropertyChange จะเห็นว่าการทำงานไม่ต้องมีการสร้างอ็อบเจกต์ของคลาสอื่นไว้ในคลาส NewCustomer เหมือนอย่างการทำงานของการพัฒนาโปรแกรมเชิงอ็อบเจกต์ ทำให้คอมโพเนนท์ต่างๆ และการทำงานแต่ละหน้าที่สามารถแยกออกจากกันได้อย่างอิสระ

3.3 การพัฒนาโปรแกรมโดยวิธีการเชิงแอสเปค

การพัฒนาโปรแกรมเชิงแอสเปคนั้น สามารถช่วยแก้ไขปัญหาครอสคัตติ้งคอนเซินของการพัฒนาโปรแกรมเชิงอ็อบเจกต์ ซึ่งจุดต่างๆที่แอสเปคเข้ามาแก้ไขปัญหาในโปรแกรมนั้น สามารถแสดงได้ดังนี้

1. โปรแกรมที่มีลักษณะการสืบทอดคลาส จากปัญหาที่เกิดขึ้นในคลาสของการประกันภัยในการพัฒนาโปรแกรมเชิงอ็อบเจกต์ การพัฒนาโปรแกรมเชิงแอสเปคนั้นจะเข้ามาแก้ปัญหาดังกล่าวโดยการแยกเมท็อด notifyListeners ออกจากซุปเปอร์คลาสไปไว้ในแอสเปค โดยนำเมท็อดดังกล่าวสร้างไว้เป็นการประกาศประเภทข้อมูลร่วมกันระหว่างคลาส และย้ายโค้ดของการเรียกใช้เมท็อด notifyListeners ภายในคลาส Policy และคลาสที่สืบทอดมาจากคลาส Policy มาไว้ในแอสเปคด้วย และกำหนดการทำงานให้ทุกครั้งที่เมท็อดใดๆที่ต้องการเรียก เมท็อด notifyListeners แอสเปคจะเข้ามาทำงานแทน ดังรูป 3.19

```
public aspect PolicyChangeNotification {

      pointcut policyStateUpdate(Policy aPolicy) :
        (execution(* Policy+.set*(..)))
         && target(Policy+) && this(aPolicy);

      after(Policy policy) returning :
        policyStateUpdate(policy) {
          notifyListeners(policy);
      }
      private static PolicyDao policyDao = new PolicyDao() ;
      private void notifyListeners(PolicyImpl policy) {
        policyDao.update(policy) ;
      }
```

รูป 3.19 แสดงส่วนหนึ่งของโค้ดของแอสเปคเจที่แก้ปัญหาที่เกิดจากการสืบทอดคลาส

จากรูป 3.19 แสดงส่วนหนึ่งของโค้ดแอสเปคที่ชื่อ PolicyChangeNotification ซึ่งภายในแอสเปคประกอบไปด้วย พอยท์คัต แอดไวซ์ และการประกาศประเภทข้อมูลร่วมกันระหว่างคลาส โดยที่ พอยท์คัตมีชื่อว่า policyStateUpdate ซึ่งรับพารามิเตอร์ 1 ตัว โดยที่ชนิดของข้อมูลคือ Policy และพอยท์คัตนี้จะจับเหตุการณ์ของโปรแกรมในจุดที่เมท็อดทุกๆเมท็อดที่มีชื่อขึ้นต้นด้วยคำว่า set ที่อยู่ภายในคลาส Policy และคลาสทุกคลาสที่สืบทอดมาจากคลาส Policy ส่วนแอดไวซ์นั้นมีการทำงานคือ เรียกเมท็อด notifiyListeners ขึ้นมาทำงาน โดยจะทำงานก็ต่อเมื่อเมท็อดที่ถูก พอยท์คัตจับมีการทำงานเสร็จแล้ว และการประกาศประเภทข้อมูลร่วมกันระหว่างคลาสนั้นเป็นส่วนของการประกาศ แอททริบิวต์ที่มีชนิดของข้อมูลคือ PolicyDao ชื่อ policyDao และเมท็อดชื่อ notifyListeners ภายในแอสเปค

เมื่อสร้างโมดูลแอสเปคดังกล่าวขึ้นมาแล้วทำให้โค้ดของคลาส Policy และคลาสทุกคลาสที่สืบทอดมาจากคลาส Policy ที่แสดงในรูป 3.4 3.5 3.6 และ 3.7 ได้มีโค้ดการทำงานที่เปลี่ยนแปลงไป โดยที่โค้ดของเมท็อด notifyListeners ในคลาส Policy นั้นจะถูกย้ายไปอยู่ในแอสเปคแทน และการเรียกใช้เมท็อด notifyListeners ในคลาส Policy และคลาสทุกคลาสที่สืบทอดมาจากคลาส Policy ก็จะถูกย้ายออกไปอยู่ในแอสเปคเช่นเดียวกัน

เมื่อทำการสร้างแอสเปคดังกล่าวขึ้นมาแล้ว นอกจากจะแก้ไขปัญหาของการกระจายของโค้ดแล้ว ยังทำให้มีความยืดหยุ่นในการแก้ไขโปรแกรมในภายหลังอีกด้วย เช่น ถ้ามีการเพิ่มคลาสการประกันคอมพิวเตอร์ ก็สามารถทำได้ง่ายโดยไม่ต้องมีการเรียกใช้เมท็อด notifyListeners ภายในคลาสการประกันคอมพิวเตอร์ เนื่องจากแอสเปคได้จับการทำงานของคลาสทุกคลาสที่มีการสืบทอดมาจากคลาส Policy และถ้าโปรแกรมมีการแก้ไขพารามิเตอร์ในเมท็อด notifyListeners ก็สามารถแก้ไขได้ง่าย เนื่องจากโค้ดของการเรียกเมท็อด notifyListeners ได้ย้ายมาอยู่ในแอสเปค ทำให้การแก้ไขโค้ดนั้นไปแก้ไขที่แอสเปคเพียงที่เดียวเท่านั้น ดังรูป 3.20

```
public aspect PolicyChangeNotification {

      pointcut policyStateUpdate(Policy aPolicy,boolean isUpdate) :
        (execution(* Policy+.set*(..,boolean)))
         && target(Policy+) && this(aPolicy) && args(isUpdate);

      after(Policy policy,boolean) returning :
        policyStateUpdate(policy,isUpdate) {
          notifyListeners(policy,isUpdate);
      }
      private static PolicyDao policyDao = new PolicyDao() ;
      private void notifyListeners(PolicyImpl policy,boolean isUpdate) {
        if(isUpdate) {policyDao.update(policy) ;}
      }
```

รูป 3.20 แสดงโค้ดของแอสเปคเจเมื่อเมท็อด notifyListeners เกิดการเปลี่ยนแปลง

จากรูป 3.20 แสดงให้เห็นว่าการแก้ไขโค้ดเมื่อเมท็อด notifyListeners มีการเปลี่ยนแปลงนั้นสามารถทำได้ง่ายเนื่องจากมาแก้ไขที่แอสเปคเพียงที่เดียวเท่านั้น

2. การติดต่อกับฐานข้อมูล จากปัญหาที่เกิดขึ้นในการติดต่อกับฐานข้อมูลในระบบประกันภัย การพัฒนาโปรแกรมเชิงแอสเปคนั้นจะเข้ามาแก้ปัญหาดังกล่าวโดยการสร้างแอสเปคขึ้นมาซึ่งภายในแอสเปคประกอบด้วยพอยท์คัตชื่อ atCusDataBase และ 2 แอดไวซ์ โดยที่ พอยท์

```
public aspect CloseDataBase {

    pointcut atCusDataBase() :
        (execution(void addCustomer()));

    before() : atCusDataBase() {
        CustomerDao.connect();
    }

    after() returning : atCusDataBase() {
        CustomerDao.close();
    }
```

omer และแอดไวซ์ตัวแรกจะทำงานก่อนที่ ตัวที่ 2 จะทำงานเมื่อเมท็อด atCusDataBase

รูป 3.21 แสดงส่วนหนึ่งของโค้ดของแอสเปคเจที่แก้ปัญหาการติดต่อกับฐานข้อมูล

จากรูป 3.21 เป็นโค้ดของการเชื่อมต่อและยกเลิกการติดต่อกับฐานข้อมูล ที่ถูกแยกมาไว้ในแอสเปค เพื่อแก้ปัญหาของครอสคัตติ้งคอนเซิน เมื่อสร้างโมดูลแอสเปคดังกล่าวขึ้นทำให้โค้ดในส่วนของอ็อบเจกต์ที่กล่าวมาในรูป 3.10 3.11 3.12 และ 3.13 จะมีการเปลี่ยนแปลงไปคือ ในเมท็อดดังกล่าวไม่ต้องมีโค้ดในการติดต่อและยกเลิกการติดต่อกับฐานข้อมูลอยู่ในคลาส

3. การตรวจจับข้อผิดพลาด จากปัญหาการจัดการกับข้อผิดพลาดที่เกิดขึ้นในโปรแกรมของระบบประกันภัยในการพัฒนาโปรแกรมเชิงอ็อบเจกต์ การพัฒนาโปรแกรมเชิงแอสเปคนั้นจะเข้ามาแก้ปัญหาดังกล่าว โดยการแยกโค้ดของการตรวจจับข้อผิดพลาดที่มีการทำงานซ้ำๆกันมาไว้ในแอสเปค และภายในแอสเปคมีพอยท์คัต uiException ซึ่งจับเหตุการณ์ของการเกิดข้อผิดพลาด 2 อย่างคือ NotfoundException และ CheckDataException และมีกำหนดการทำงานโดยก่อนที่อ็อบเจกต์จะมีการทำงานในส่วนของคำสั่ง catch ให้แอสเปคมีการทำงานขึ้น ดังรูป 3.22

```
package ui;

import model.*;
import java.awt.*;
import javax.swing.JOptionPane;

public aspect CatchNotFoundAndCheckDataException {

    pointcut uiException(Exception ee,Component bb) :
        (handler(NotFoundException) ||
        handler(CheckDataException)) && args(ee) && this(bb);

    before(Exception e,Component b) : uiException(e,b) {
        //System.out.println("Test ERROR :"+e.toString()) ;
        JOptionPane.showMessageDialog(b,e.toString(),"error",JOptionPane.INFORMATION_MESSAGE);
    }
}
```

รูป 3.22 แสดงโค้ดของแอสเปคเจที่แก้ปัญหาการจัดการกับข้อผิดพลาดที่เกิดขึ้นในโปรแกรม

จากรูป 3.22 เป็นโค้ดของการจัดการกับข้อผิดพลาดที่เกิดขึ้นในโปรแกรม ที่ถูกแยกมาไว้ในแอสเปค เพื่อแก้ปัญหาของครอสคัตติ้งคอนเซิน เมื่อสร้างโมดูลแอสเปคดังกล่าวขึ้นทำให้โค้ดในส่วนของอ็อบเจกต์ที่กล่าวมาในรูป 3.14 3.15 3.16 และ 3.17 มีการเปลี่ยนแปลงไปจากเดิมคือในส่วนของคำสั่ง catch โค้ดการทำงานที่แสดงข้อความที่บอกถึงข้อผิดพลาดที่เกิดขึ้นในการทำงาน

บทที่ 4
สรุปผลการวิจัยและข้อเสนอแนะ

4.1 ผลการวิจัยและพัฒนา

จากการวิจัยวิธีการพัฒนาโปรแกรมทั้ง 3 สามารถสรุปรายละเอียดได้ดังต่อไปนี้

การพัฒนาโปรแกรมที่นิยมในปัจจุบันนี้ได้แก่ การพัฒนาโปรแกรมเชิงอ็อบเจกต์และเชิงคอมโพเนนท์ แต่การพัฒนาโปรแกรมทั้ง 2 วิธีนั้นยังมีปัญหาที่ไม่สามารถแก้ไขได้คือ ปัญหาครอสคัตติ้งคอนเซิน ดังนั้นจึงได้มีผู้คิดวิธีการพัฒนาโปรแกรมเชิงแอสเปคขึ้นมาเพื่อแก้ไขปัญหาดังกล่าว ดังนั้นผู้จัดทำจึงได้พัฒนาระบบขึ้นมาเพื่อใช้ศึกษาเปรียบเทียบให้เห็นเป็นรูปธรรมถึงความแตกต่างระหว่างการพัฒนาโปรแกรมทั้ง 3 วิธี โดยระบบที่ใช้ในการพัฒนาคือระบบประกันภัย เนื่องจากเป็นระบบที่แสดงให้เห็นถึงปัญหาของการพัฒนาโปรแกรมได้อย่างชัดเจน จากนั้นทำการพัฒนาโปรแกรมด้วยวิธีการเชิงอ็อบเจกต์ เพื่อแสดงให้เห็นถึงปัญหาที่เกิดขึ้นในการพัฒนาโปรแกรม และพัฒนาโปรแกรมด้วยวิธีการเชิงคอมโพเนนท์และแอสเปคเพื่อแสดงให้เห็นถึงวิธีการแก้ปัญหาของวิธีการพัฒนาโปรแกรมเชิงอ็อบเจกต์ ซึ่งผลลัพธ์ที่ได้จากการเปรียบเทียบแสดงได้ดังต่อไปนี้

1. ปัญหาโมดูลขาดความเป็นอิสระของการพัฒนาโปรแกรมเชิงอ็อบเจกต์นั้นทำให้คลาสต่างๆผูกติดกัน ทั้งๆที่คลาสบางคลาสมีหน้าที่การทำงานคนละส่วนจึงควรที่จะมีการทำงานแยกจากกัน ปัญหาดังกล่าวทำให้โปรแกรมขาดความยืดหยุ่น แต่การพัฒนาโปรแกรมเชิงคอมโพเนนท์นั้นสามารถเข้ามาแก้ปัญหาได้โดยการใช้การส่งเหตุการณ์เพื่อติดต่อกับคอมโพเนนท์อื่น และใช้อินเตอร์เฟซเป็นช่องทางที่ใช้ในการติดต่อ

2. ปัญหาครอสคัตติ้งคอนเซินของการพัฒนาโปรแกรมเชิงอ็อบเจกต์นั้นเกิดขึ้นได้จากหลายสาเหตุ ดังนี้ การเรียกใช้เมท็อดของซุปเปอร์คลาสในคลาสที่มีการสืบทอด การทำงานในส่วนของการติดต่อกับฐานข้อมูล การจัดการกับข้อผิดพลาดที่เกิดขึ้นในการทำงานของโปรแกรม ปัญหาดังกล่าวนี้สามารถแก้ไขได้ด้วยวิธีการพัฒนาโปรแกรมเชิงแอสเปค ซึ่งแอสเปคแก้ปัญหาโดยการสร้างโมดูลขึ้นมา และทำการย้ายโค้ดที่ทำให้เกิดปัญหาครอสคัตติ้งคอนเซินมาไว้ในแอสเปคโมดูล และกำหนดการทำงานให้กับแอสเปค ทำให้เมื่อการทำงานมาถึงจุดที่แอสเปคได้กำหนดการทำงานไว้แอสเปคก็จะเข้ามาทำงานโดยอัตโนมัติ

4.2 ข้อเสนอแนะ

1. การวิจัยนี้มุ่งเน้นไปที่การแก้ปัญหาครอสคัตติ้งคอนเซิน โดยใช้วิธีการพัฒนาโปรแกรมเชิงแอสเปค แต่ในความเป็นจริงยังมีเนื้อหาของการพัฒนาโปรแกรมเชิงคอมโพเนนท์ในส่วนของ

การควบคุมเวอร์ชัน การทำงานข้ามภาษา เป็นต้น ซึ่งอาจจะนำไปวิจัยเพื่อศึกษาเปรียบเทียบต่อไปได้

2. การแก้ปัญหาครอสคัตติ้งคอนเซินนั้น ยังมีวิธีอื่นนอกจากการพัฒนาโปรแกรมเชิงแอสเปค เช่น การพัฒนาโปรแกรมแบบเอ็กพลิซิต เป็นต้น ซึ่งอาจจะนำการพัฒนาโปรแกรมเชิงแอสเปคและการพัฒนาโปรแกรมแบบเอ็กพลิซิตมาศึกษาเปรียบเทียบกันเพื่อให้ทราบถึงข้อแตกต่างในการแก้ปัญหา

บรรณานุกรม

[1]. ดร.วีระศักดิ์ ซึงถาวร. 2546. จาวาโปรแกรมมิงเล่ม1. กรุงเทพฯ : ซีเอ็ด.

[2]. ดร.วีระศักดิ์ ซึงถาวร. 2546. จาวาโปรแกรมมิงเล่ม3. กรุงเทพฯ : ซีเอ็ด.

[3]. สุนทริน วงศ์ศิริกุล. พัฒนาโมเดลยุคใหม่ยูเอ็มแอลมาตรฐานการสร้างโมเดลระบบงาน. กรุงเทพ : ซัคเซสมีเดีย.

[4]. Adrian Colyer, Andy Clement, George Harley, Matthew Webster. 2005. Eclipse AspectJ. NJ, USA : Pearson Education.

[5]. Andy Ju An Wang, Kai Qian. 2005. Component-Oriented Programming. NJ, USA : John Wiley & Sons.

[6]. Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, Willaim G. Griswold. An Overview of AspectJ. CA, USA.

[7]. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin. 1997. Aspect-Oriented Programming. CA, USA : Springer-Verlag.

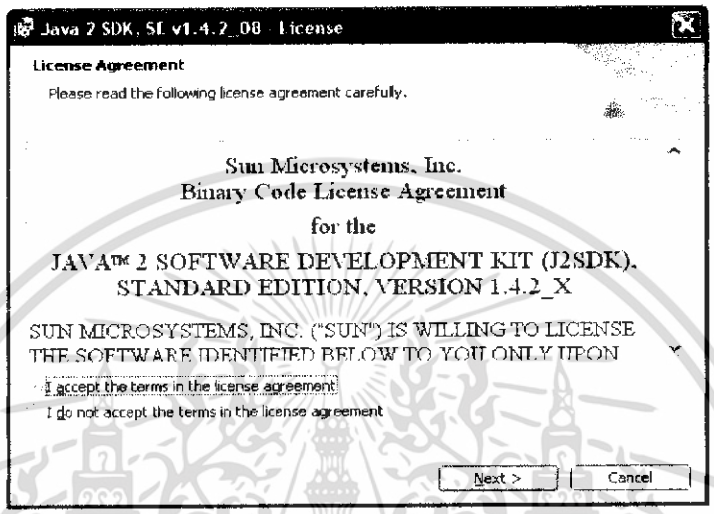
[8]. Robert E. Filman, Tzilla Elrad, Siobhan Clarke, Mehmet Aksit. 2005. Aspect-Oriented Software Development. NJ, USA : Pearson Education.

ภาคผนวก ก

การติดตั้งโปรแกรม

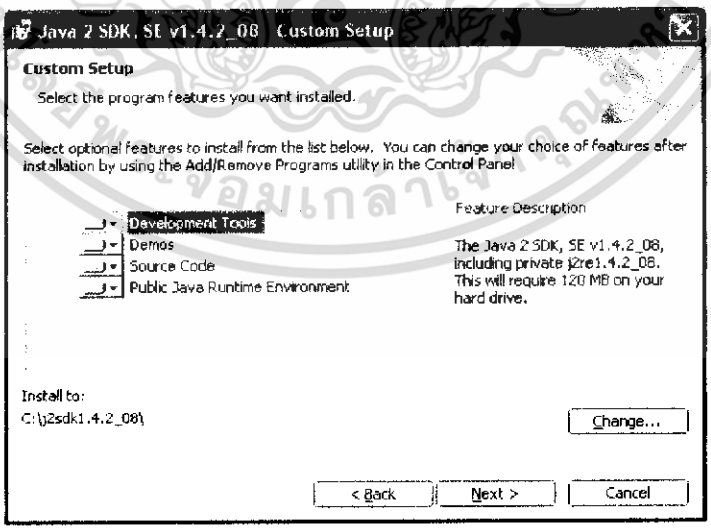
การติดตั้งโปรแกรมแอสเปคเดฟสูทเซ็ตอัพ1.4 (AspectJDevSuiteSetup1.4)

1. ต้องทำการติดตั้งโปรแกรมนั้นจำเป็นต้องทำการติดตั้งเจดีเคเวอร์ชั่น 1.4 ก่อน ซึ่งในโครงงานนี้ใช้เจดีเคเวอร์ชั่น 1.4.2_08 เมื่อทำการรันโปรแกรมจะมีช่องหน้าต่างดังรูป ก-1 จากนั้นเลือกหัวข้อแรกคือ I accept the term in the license agreement แล้วคลิกปุ่ม next>



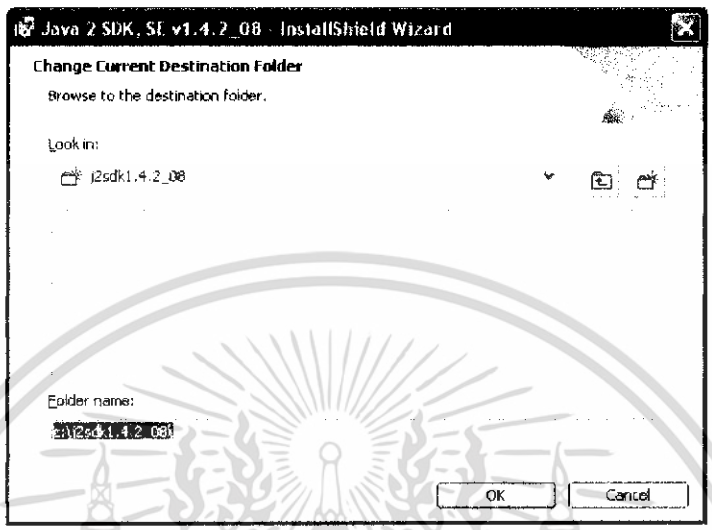
รูป ก-1 แสดงหน้าต่างแรกสำหรับการติดตั้งเจดีเค 1.4.2_08

2. หน้าจอจะแสดงหน้าต่างสำหรับเลือกฟีเจอร์ที่ต้องการติดตั้ง แล้วคลิกปุ่ม next> ดังรูป ก-2



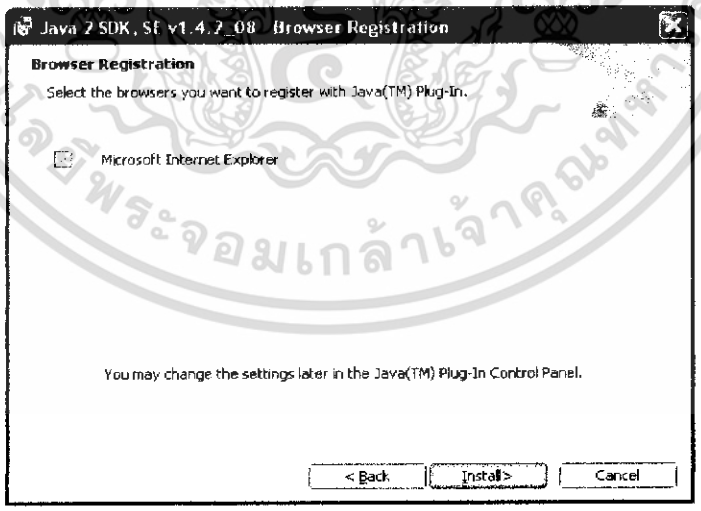
รูป ก-2 แสดงหน้าต่างสำหรับเลือกฟีเจอร์ที่ต้องการติดตั้ง

3. หน้าจอจะแสดงหน้าต่างสำหรับระบุไดเรคทอรีที่ต้องการติดตั้งโปรแกรม แล้วคลิกปุ่ม ok ดังรูป ก-3



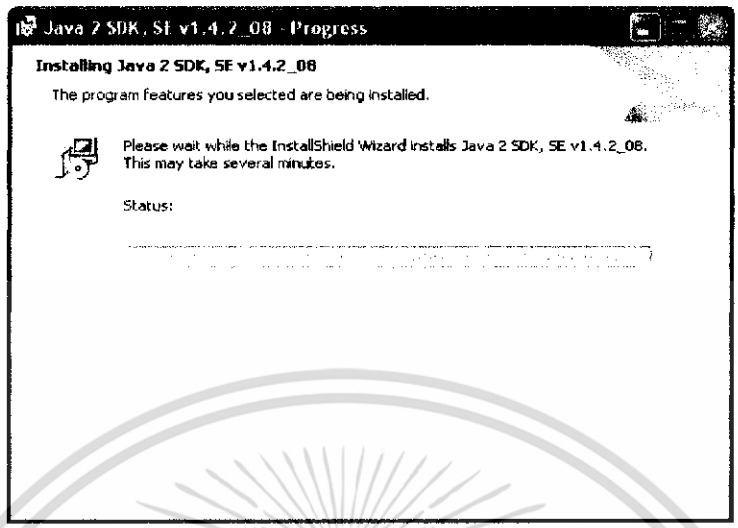
รูป ก-3 แสดงหน้าต่างสำหรับระบุไดเรคทอรีที่ต้องการติดตั้งโปรแกรม

4. หน้าจอจะแสดงหน้าต่างสำหรับเลือกรีจิสเตอร์ปลั๊กอินของจาวาลงในเบราเซอร์ แล้วคลิกปุ่ม Install> ดังรูป ก-4



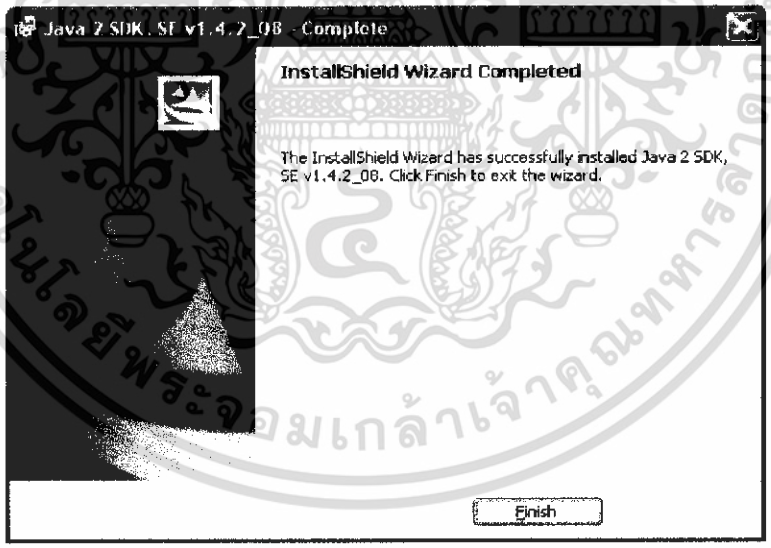
รูป ก-4 แสดงหน้าต่างสำหรับเลือกรีจิสเตอร์ปลั๊กอินของจาวาลงในเบราเซอร์

5. หน้าจอแสดงการติดตั้งโปรแกรม ดังรูป ก-5



รูป ก-5 หน้าต่างแสดงการติดตั้งโปรแกรม

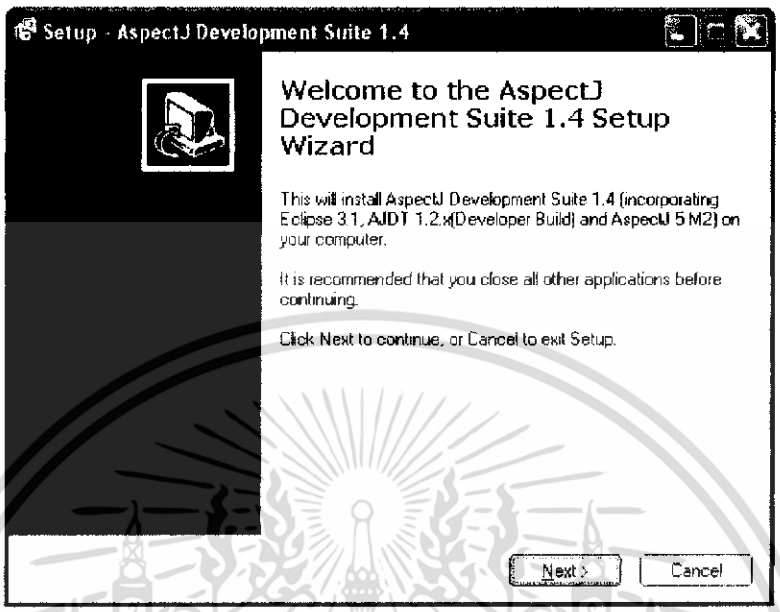
6. แสดงการติดตั้งโปรแกรมเสร็จสมบูรณ์ ดังรูป ก-6



รูป ก-6 แสดงการติดตั้งโปรแกรมเสร็จสมบูรณ์

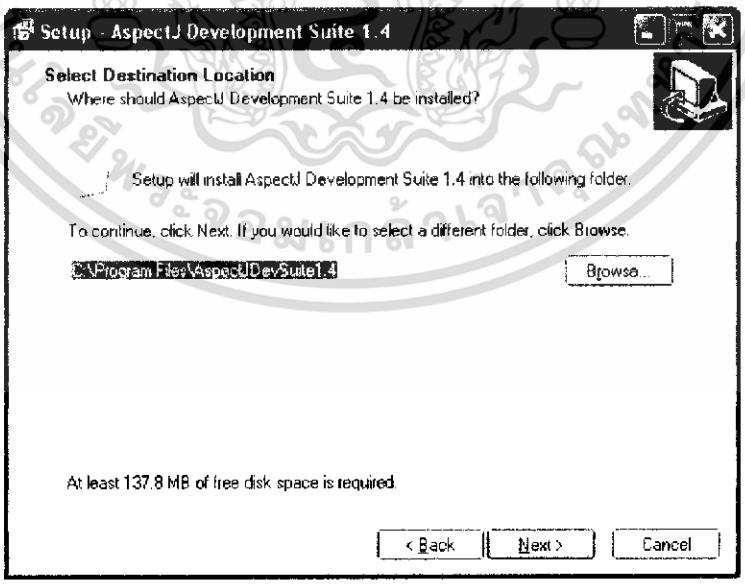
7. จากนั้น ทำการดาวน์โหลด (Download) ไฟล์ aspectj-1.5.0M2.jar จาก www.eclipse.org มาไว้ในไดเรคทอรี C:\

8. จากนั้นทำการรันโปรแกรมแอสเปคเดฟซุตเซ็ตอัพ1.4 จะมีช่องหน้าต่างดังรูป ก-7 จากนั้นคลิกปุ่ม Next



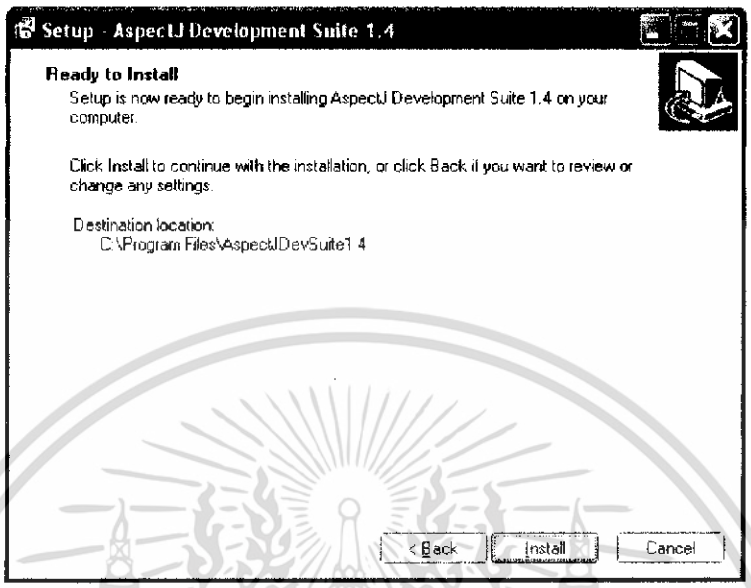
รูป ก-7 แสดงหน้าต่างแรกสำหรับการติดตั้งแอสเปคเดฟซุตเซ็ตอัพ1.4

9. จากนั้นจะแสดงหน้าต่างให้ระบุไดเรคทอรีที่ต้องการติดตั้งโปรแกรม ดังรูป ก-8 จากนั้นคลิกปุ่ม Next



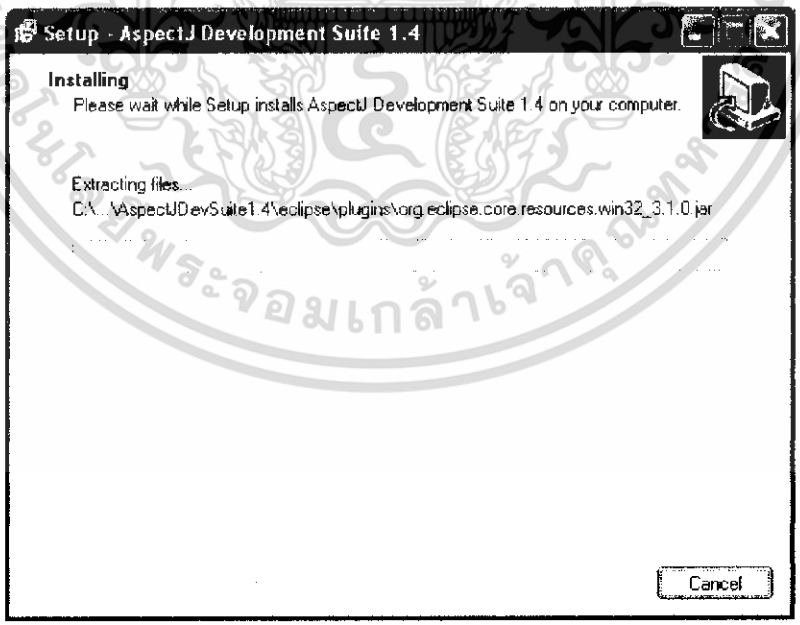
รูป ก-8 แสดงหน้าต่างสำหรับระบุไดเรคทอรีที่ต้องการติดตั้งโปรแกรม

10. จากนั้นหน้าต่างจะแสดงไดเรคทอรีที่ได้เลือกไว้ ดังรูป ก-9 จากนั้นคลิกปุ่ม Next



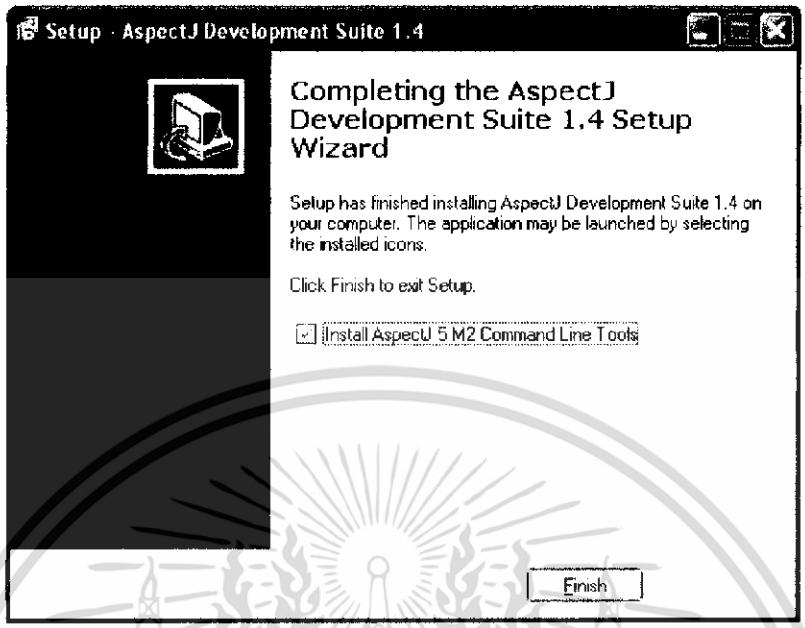
รูป ก-9 แสดงหน้าต่างแสดงไดเรคทอรีที่จะทำการติดตั้งโปรแกรม

11. จากนั้นจะเริ่มทำการติดตั้งโปแกรม ดังรูป ก-10



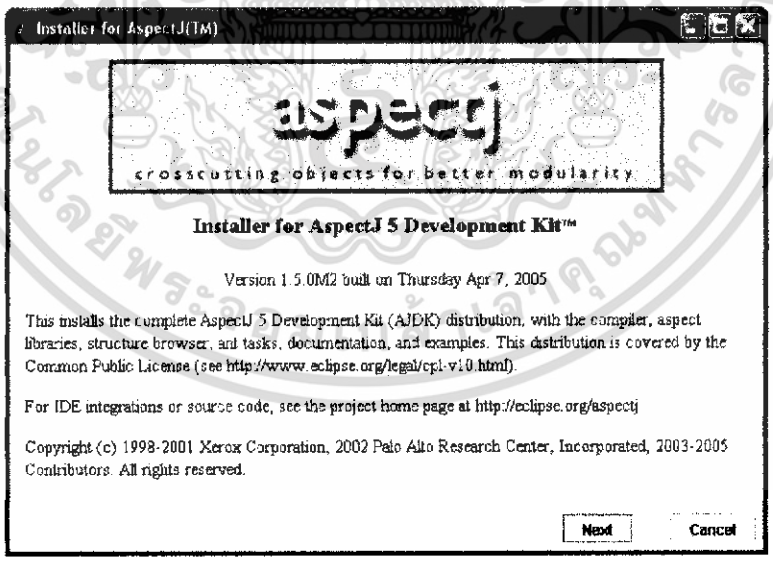
รูป ก-10 หน้าต่างแสดงการติดตั้งโปรแกรม

12. หน้าต่างจะแสดงการเสร็จสิ้นการติดตั้งโปรแกรม คลิกปุ่ม Finish ดังรูป ก-11



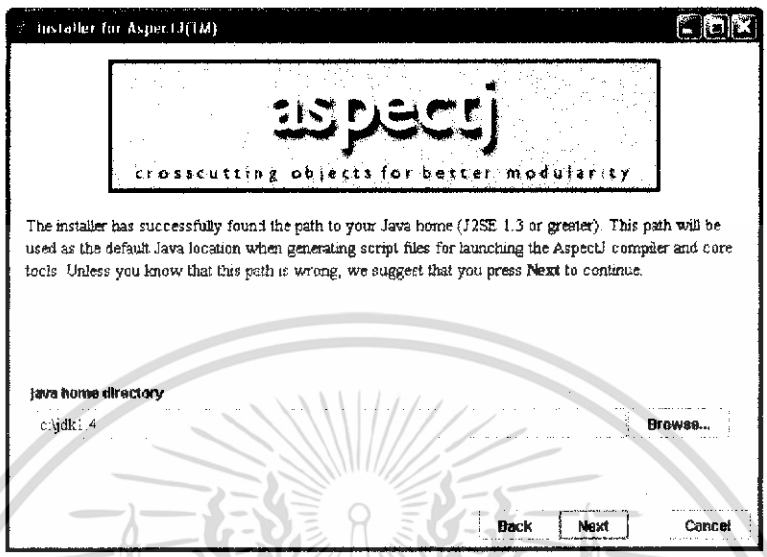
รูป ก-11 หน้าต่างแสดงการเสร็จสิ้นการติดตั้งโปรแกรม

13. จากนั้นจะมีหน้าต่างให้ติดตั้งเอเจดีเค (AJDK) ให้คลิกปุ่ม Next ดังรูป ก-12



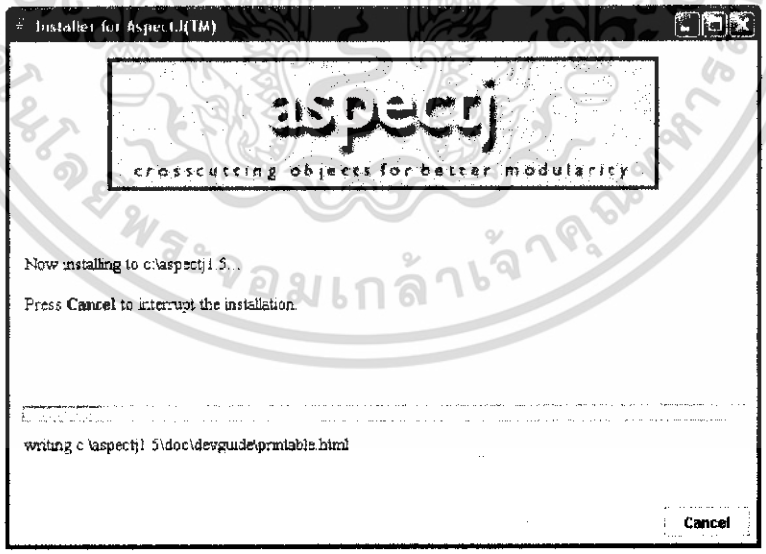
รูป ก-12 หน้าต่างแสดงการติดตั้งเอเจดีเค

14. จากนั้นจะแสดงหน้าต่างให้ระบุไดเรคทอรีของเจดีเค ดังรูป ก-13 ให้ทำการระบุไปที่เจดีเคที่ติดตั้งไว้ในตอนแรก จากนั้นคลิกปุ่ม Next



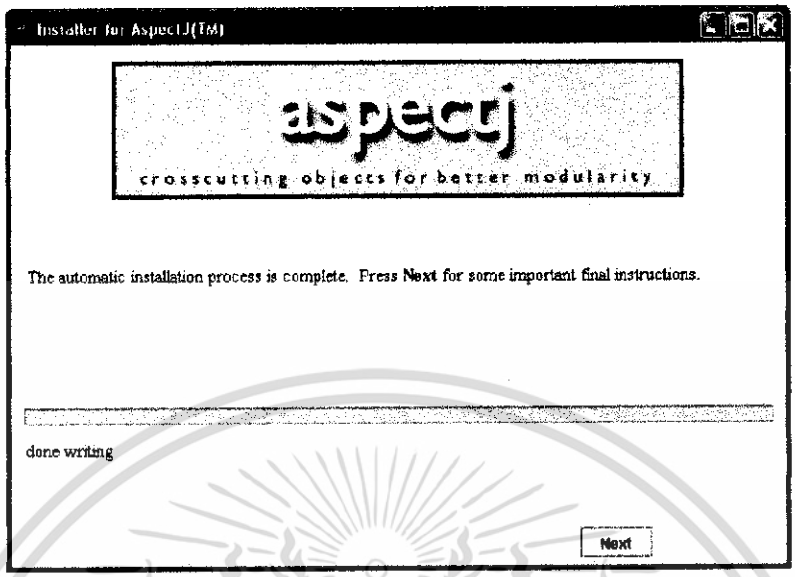
รูป ก-13 แสดงหน้าต่างระบุไดเรคทอรีเจดีเค

15. หน้าต่างแสดงการติดตั้งโปรแกรมลงไดเรคทอรี C:\ ดังรูป ก-14



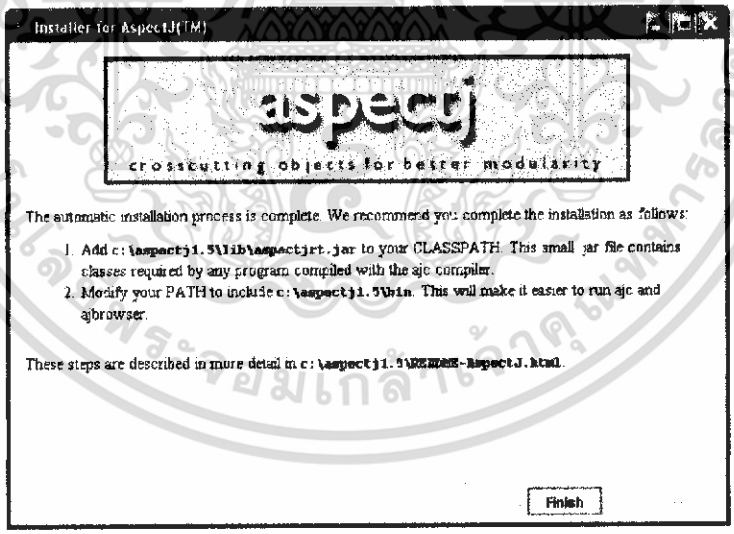
รูป ก-14 หน้าต่างแสดงการติดตั้งโปรแกรม

16. เมื่อโปรแกรมทำกรติดตั้งเสร็จแล้วให้คลิกปุ่ม Next ดังรูป ก-15



รูป ก-15 หน้าต่างแสดงการติดตั้งโปรแกรมเสร็จแล้ว

17. แสดงการเสร็จสิ้นการติดตั้งเอเจดีเค ดังรูป ก-16 จากนั้นคลิกปุ่ม Finish



รูป ก-16 หน้าต่างแสดงการเสร็จสิ้นการติดตั้งเอเจดีเค

การติดตั้งโปรแกรมบีดีเค

1. ดาวน์โหลดไฟล์ bdk1_1.zip มาจาก www.java.sun.com จากนั้นทำการคลายการบีบอัด (Unzip) ไฟล์มาไว้ที่ไดเรคอทอี C:\

2. ทำการเปิดไฟล์ run.bat ซึ่งอยู่ที่ไดเรคทอรี C:\beans\beanbox และทำการกำหนดค่าของตัวแปรที่ชื่อว่า PATH ให้ชี้ไปที่ไดเรคทอรีของเจดีเค โดยการเพิ่มคำสั่งเข้าไปในไฟล์ดังนี้

```
PATH c:\windows;C:\jdk1.4.1_08\bin;C:\beans\beanbox;
```

ภาคผนวก ข

รายละเอียดคลาสในระบบประกันภัย

แพคเกจการติดต่อกับผู้ใช้ประกอบด้วยคลาสดังต่อไปนี้

คลาส SimpleInsuranceApp เป็นคลาสที่เป็นเฟรม (Frame) หลักของแอพพลิเคชั่น ประกอบด้วยเมท็อดดังต่อไปนี้

main(String args[])

คลาส MainMenu เป็นคลาสที่สร้างเมนูของแอพพลิเคชั่น ประกอบด้วยเมท็อดดังต่อไปนี้

MainMenu(JFrame f,JPanel r)

actionPerformed(ActionEvent e)

Initial()

คลาส AddHousePolicy เป็นคลาสที่ทำหน้าที่เพิ่มข้อมูลของการประกันบ้าน ประกอบด้วยเมท็อดดังต่อไปนี้

AddHousePolicy()

actionPerformed(ActionEvent e)

addHousePolicy()

addDate()

clearInput()

คลาส AddLifePolicy เป็นคลาสที่ทำหน้าที่เพิ่มข้อมูลของการประกันชีวิต ประกอบด้วยเมท็อดดังต่อไปนี้

AddLifePolicy()

actionPerformed(ActionEvent e)

addLifePolicy()

addDate()

clearInput()

คลาส AddAutoPolicy เป็นคลาสที่ทำหน้าที่เพิ่มข้อมูลของการประกันรถยนต์ ประกอบด้วยเมท็อดดังต่อไปนี้

AddAutoPolicy()

actionPerformed(ActionEvent e)

addAutoPolicy()

addDate()

clearInput()

คลาส AddClaim เป็นคลาสที่ทำหน้าที่เพิ่มข้อมูลของค่าสินไหมทดแทน ประกอบด้วยเมท็อดดังต่อไปนี้

AddClaim()

actionPerformed(ActionEvent e)

void addClaim()

addDate()

clearInput()

คลาส EditHousePolicy เป็นคลาสที่ทำหน้าที่แก้ไขข้อมูลของการประกันบ้าน ประกอบด้วยเมท็อดดังต่อไปนี้

EditHousePolicy()

actionPerformed(ActionEvent e)

searchHousePolicy()

updateHousePolicy()

deleteHousePolicy()

addDate()

clearInput()

คลาส EditLifePolicy เป็นคลาสที่ทำหน้าที่แก้ไขข้อมูลของการประกันชีวิต ประกอบด้วยเมท็อดดังต่อไปนี้

EditLifePolicy()

actionPerformed(ActionEvent e)

searchLifePolicy()

updateLifePolicy()

deleteLifePolicy()

addDate()

clearInput()

คลาส EditAutoPolicy เป็นคลาสที่ทำหน้าที่แก้ไขข้อมูลของการประกันรถยนต์ ประกอบด้วยเมท็อดดังต่อไปนี้

EditAutoPolicy()

actionPerformed(ActionEvent e)

searchAutoPolicy()

updateAutoPolicy()

deleteAutoPolicy()

addDate()

clearInput()

คลาส NewCustomer เป็นคลาสที่ทำหน้าที่เพิ่มข้อมูลของลูกค้า ประกอบด้วยเมท็อดดังต่อไปนี้

NewCustomer()

actionPerformed(ActionEvent e)

addCustomer()

addDate()

clearInput()

คลาส FindAndEditCustomer เป็นคลาสที่ทำหน้าที่ค้นหาและแก้ไขข้อมูลของลูกค้า ประกอบด้วยเมท็อดดังต่อไปนี้

FindAndEditCustomer()

actionPerformed(ActionEvent e)

searchCustomer()

updateCustomer()

deleteCustomer()

addDate()

clearInput()

แพคเกจความสัมพันธ์ของระบบประกอบด้วยคลาสดังต่อไปนี้

คลาส Address เป็นคลาสที่เป็นตัวแทนของที่อยู่อาศัย ประกอบด้วยเมท็อดดังต่อไปนี้

Address (Customer customer , String line , String townCity , String postcode , String phone)

setLine(String line)

setTownCity(String townCity)

setPostcode(String postcode)

setPhone(String phone)

setCustomer(Customer customer)

getLine()

getTownCity()

getPostcode()

getPhone()

getCustomer()

คลาส AutoMobile เป็นคลาสที่เป็นตัวแทนของรถยนต์ ประกอบด้วยเมท็อดดังต่อไปนี้

Automobile (String make , String model , String colour , String plate , double marketValue , int insuranceGroup)

getMake()

getModel()

getColour()

getPlate()

getMarketValue()

getInsuranceGroup()

คลาส AutoPolicy เป็นคลาสที่เป็นตัวแทนของการประกันรถยนต์ ประกอบด้วยเมท็อดดังต่อไปนี้

AutoPolicy(Customer c , String policyId)

AutoPolicy(String startDate , String endDate , double value , boolean isQuote , Schedule schedule , Customer customer , Automobile car , double excess , boolean noClaims , String policyId)

toString()

setNoClaims(boolean noClaims)

setExcess(double excess)

setCar(Automobile car)

isNoClaims()

getExcess()

getCar()

คลาส Claim เป็นคลาสที่เป็นตัวแทนของค่าสินไหมทดแทน ประกอบด้วยเมท็อดดังต่อไปนี้

Claim(Policy policy , String claimed , String created , double amount , String status , String comments)

toString()

setPolicy (Policy policy)

setClaimID (String claimId)

setCreated(String created)

setAmount(double amount)

setStatus(String status)

setComments(String comments)

getPolicy ()

getClaimID ()

getCreated()

getAmount()

getStatus()

getComments()

addClaimListener()

removeClaimListener()

notifyListeners()

คลาส Customer เป็นคลาสที่เป็นตัวแทนของลูกค้า ประกอบด้วยเมท็อดดังต่อไปนี้

Customer (String customerId , String firstName , String lastName , Address address)

toString()

equals(Object obj)

equals(String cusId)

setFirstName(String firstName)

setLastName(String lastName)

setAddress(Address address)

setPolicy (Vector vPolicy)

getFirstName()

getLastName()

getAddress()

getPolicy()

addCustomerListener()

removeCustomerListener()

notifyListeners()

addPolicy(Policy policy)

removePolicy(Policy policy)

คลาส LifePolicy เป็นคลาสที่เป็นตัวแทนของการประกันชีวิต ประกอบด้วยเมท็อดดังต่อไปนี้

LifePolicy(Customer c,String policyId)

LifePolicy (String startDate , String endDate , double value , boolean isQuote , Schedule schedule , Customer customer , boolean smoker , boolean drinker , String policyId)

toString()

setSmoker(boolean smoker)

setDrinker(boolean drinker)

setIronman(boolean ironman)

isSmoker()

isDrinker()

isIronman()

คลาส Policy เป็นคลาสที่เป็นตัวแทนของการประกันภัย ประกอบด้วยเมท็อดดังต่อไปนี้

Policy(Customer customer,String policyId)

Policy (String startDate , String endDate , double value , boolean isQuote , Schedule schedule , Customer customer , String policyId)

equals(Object obj)

setStartDate(String startDate)

setEndDate(String endDate)

setValue(double value)

setIsQuote(boolean isQuote)

setSchedule(Schedule schedule)

setClaim(Vector claim)

getPolicyID()

getStartDate()

getEndDate()

getValue()

getValue()

isQuote()

getSchedule()

getCustomer()

getType()

getClaim()

addClaim(Claim claim)

removeClaim(Claim claim)

addPolicyListener()

removePolicyListener()

notifyListeners()

คลาส HousePolicy เป็นคลาสที่เป็นตัวแทนของการประกันที่อยู่อาศัย ประกอบด้วยเมท็อดดังต่อไปนี้

HousePolicy(Customer c,String policyId)

HousePolicy(String startDate , String endDate , double value , boolean isQuote , Schedule schedule , Customer customer , double worth , Address address , String policyId)

toString()

setWorth(double worth)

setAddress(Address address)

getWorth()

getAddress()

คลาส Schedule เป็นคลาสที่เป็นตัวแทนของตารางการจ่ายเงิน ประกอบด้วยเมท็อดดังต่อไปนี้

Schedule(long numPayments , double paymentAmount , String firstPaymentDue)

getNumPayments()

getPaymentAmount()

getFirstPaymentDue()

แพคเกจการติดต่อฐานข้อมูลของระบบประกอบด้วยคลาสดังต่อไปนี้

คลาส ConDB เป็นคลาสทำทำหน้าที่ในสร้างการติดต่อ ยกเลิก และทำงานกับฐานข้อมูล ประกอบด้วยเมท็อดดังต่อไปนี้

ConDB ()

connect()

SQLExecute(String query)

SQLRetrive(String query)

getFieldValue(ResultSet r , String fn , String ft)

close()

คลาส CustomerDao เป็นคลาสที่มีทำหน้าที่เป็นตัวกลางในการจัดการข้อมูลระหว่างคลาส Customer กับฐานข้อมูล ประกอบด้วยเมท็อดดังต่อไปนี้

connect()

close()

search(String id)

addNew(Customer aCustomer)

delete(Customer aCustomer)

update(Customer aCustomer)

คลาส PolicyDao เป็นคลาสที่มีทำหน้าที่เป็นตัวกลางในการจัดการข้อมูลระหว่างคลาส HousePolicy LifePolicy และ AutoPolicy กับฐานข้อมูล ประกอบด้วยเมท็อดดังต่อไปนี้

connect()

close()

getAll(Customer customer)

search(String policyId)

addNew (Policy policy)

update(Policy policy)

delete(Policy policy)

คลาส ClaimDao เป็นคลาสที่มีทำหน้าที่เป็นตัวกลางในการจัดการข้อมูลระหว่างคลาส Claim กับฐานข้อมูล ประกอบด้วยเมท็อดดังต่อไปนี้

connect()

close()

getAll(Policy policy)

search(String claimId)

addNew(Claim claim)

update(Claim claim)

delete(Claim claim)

นอกจากนี้ยังประกอบด้วยคลาสที่จัดการกับข้อผิดพลาดของโปรแกรมซึ่งประกอบด้วยคลาสดังต่อไปนี้

คลาส CheckDataException เป็นคลาสที่ทำหน้าที่ตรวจจับข้อผิดพลาดในการเพิ่มข้อมูลลงในฐานข้อมูลว่าข้อมูลที่ต้องการเพิ่มมีอยู่ในฐานข้อมูลแล้วหรือไม่ ประกอบด้วยเมท็อดดังต่อไปนี้

CheckDataException(String message)

คลาส NotFoundException เป็นคลาสที่ทำหน้าที่ตรวจจับข้อผิดพลาดในการค้นหาข้อมูลจากฐานข้อมูลว่าข้อมูลที่ต้องการค้นหามีอยู่ในฐานข้อมูลหรือไม่ ประกอบด้วยเมท็อดดังต่อไปนี้

NotFoundException(String message)