ระบบรับชำระการเคหะแห่งชาติ

National Housing Authority Payment System

โดย

นายวรพันธ์ วิสุทธิ

รหัส 41067084

อาจารย์ที่ปรึกษา

ผศ. บรรจง ปิยธำรง

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ภาคเรียนที่ 2 ปีการศึกษา 2542

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	ระบบรับชำระการเดหะแห่งชาติ
นักศึกษา	นายวรพันธ์ วิสุทธิ
อาจารย์ที่ปรึกษา	ผศ. บรรจง ปิยธำรง
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2542

บทคัดย่อ

การพัฒนาระบบเชิงวัตถุ (Object Oriented Systems Development) เป็นแนวทางใหม่ในการพัฒนาระบบซึ่งสามารถรองรับความซับซ้อนของระบบที่เกิดขึ้น และยังสามารถรองรับเทคนิคการพัฒนาซอฟต์แวร์ เครื่องมือ รวมถึงเทคโนโลยีที่เปลี่ยนแปลงได้อย่างรวดเร็วได้ ประโยชน์หลักๆ ของการพัฒนาระบบเชิงวัตถุ คือเพิ่มความน่าเชื่อถือและเพิ่มผลผลิตของผู้พัฒนาให้สูงขึ้นได้

โครงการพัฒนาระบบนี้เป็นการพัฒนาระบบเชิงวัตถุ ของระบบรับชำระการเดหะแห่งชาติ โดยใช้ UML (Unified Modeling Language) เป็นเครื่องมือในการสร้างแบบจำลองของระบบ และจะพัฒนาจนถึงขั้นการสร้างโปรแกรมของระบบขึ้นมา

Title	National Housing Authority Payment System
Student	Mr. Vorapun Visuthi
Advisor	Asst. Prof. Banjong Piyatamrong
Level of Study	Master of Science in Information Technology
Major	Information Science
Academic Year	1999

ABSTRACT

Object-oriented systems development is new approach that can support complexity of system and support software development techniques, tools and technologies that are changing rapidly. The main benefits of object-oriented systems development are improved reliability and enhanced developer productivity.

This project is a system development project for the National Housing Authority Payment System using an object-oriented development approach. Using UML (Unified Modeling Language) as a tool for building model of a system and developing up to implement phase.

สารบัญ

หน้า

สารบัญภาพ

หน้า

รูปที่

บทที่ 1

บทนำ

1.1 คำอธิบายโครงการ

โครงการนี้เป็นโครงการ การพัฒนาระบบรับชำระของการเคหะแห่งชาติ ซึ่งจะใช้การพัฒนาระบบเชิงวัตถุ (Object-Oriented System Development) และใช้ UML (Unified Modeling Language) ช่วยในการสร้างแบบจำลอง (Model) ของระบบรับชำระขึ้น และจะสร้างระบบโดยใช้ DBMS เป็น Oracle เวอร์ชัน 8 ซึ่งทำหน้าที่เป็น Back End และพัฒนา Front End ด้วย Oracle Developer เวอร์ชัน 6

ระบบรับชำระเป็นระบบหนึ่งภายในหน่วยงานรับชำระของการเคหะแห่งชาติ ซึ่งจะครอบคลุมถึงการรับชำระเงินสดและการรับชำระผ่านธนาคาร รวมถึงการออกรายงานต่างๆ ของหน่วยงานรับชำระ

1.1.1 วัตถุประสงค์

1. เพื่อให้ข้อมูลการรับชำระมีความถูกต้อง รวดเร็ว และทันสมัยตลอดเวลา
2. เพื่อให้สามารถทราบผลการรับชำระได้อย่างถูกต้อง

1.1.2 แผนการดำเนินงาน

1. ศึกษาความต้องการของระบบ
2. วิเคราะห์และออกแบบระบบโดยใช้ UML ช่วยในการสร้างแบบจำลอง
3. พัฒนาระบบตามที่ได้ออกแบบไว้

1.1.3 ผลที่คาดว่าจะได้รับ

1. ข้อมูลมีความถูกต้อง รวดเร็ว และทันสมัยตลอดเวลา
2. การดำเนินงานมีความรวดเร็วและถูกต้องมากกว่าเดิม
3. สามารถทราบผลการชำระของลูกหนี้ได้อย่างรวดเร็ว ถูกต้อง และทันทีเมื่อต้องการ
4. สามารถนำข้อมูลที่ถูกต้อง มาวางแผนสำหรับแก้ปัญหาของลูกหนี้ได้อย่างรวดเร็ว ถูกต้องและทันต่อเหตุการณ์

1.2 โครงสร้างเนื้อหาของเอกสาร

1.2.1 บทที่ 1 บทนำ

เป็นบทที่กล่าวถึงภาพรวมของโครงการ ซึ่งประกอบไปด้วยวัตถุประสงค์ แผนการดำเนินงาน ผลที่คาดว่าจะได้รับ และความเป็นมาเกี่ยวกับการพัฒนาระบบเชิงวัตถุและ UML

1.2.2 บทที่ 2 ทฤษฎีที่เกี่ยวข้อง

เป็นบทที่กล่าวถึงทฤษฎีที่เกี่ยวข้องกับโครงการนี้ซึ่งจะประกอบไปด้วยกระบวนการของการวิเคราะห์และออกแบบเชิงวัตถุ และยังรวมถึงเนื้อหาของ UML (Unified Modeling Language) ที่ใช้ภายในโครงการ

1.2.3 บทที่ 3 การวิเคราะห์และออกแบบ

เป็นบทที่อธิบายการวิเคราะห์และออกแบบหลังจากผ่านการศึกษาระบบมาแล้ว ซึ่งจะเรียงลำดับขั้นตอนในการนำเสนอตามเนื้อหาของทฤษฎีที่เกี่ยวข้อง และจะแสดง UML เป็นแบบจำลองของระบบรับชำระในมุมมองต่างๆ

1.2.4 บทที่ 4 การสร้างระบบ

เป็นบทที่กล่าวถึงวิธีการพัฒนาซอฟต์แวร์ระบบรับชำระ ซึ่งจะเป็นการสร้างซอฟต์แวร์ตามสิ่งที่ได้จากการวิเคราะห์และออกแบบ โดยจะเป็นบทที่ลงรายละเอียดของเครื่องมือที่ใช้งาน

1.2.5 บทที่ 5 สรุป

เป็นบทที่กล่าวถึงปัญหาที่พบในการพัฒนาและแสดงการเปรียบเทียบระหว่างการพัฒนาระบบเชิงวัตถุกับการพัฒนาระบบแบบเดิม แล้วยังรวมถึงแนวทางของการพัฒนาระบบในอนาคต

1.2.6 บทที่ 6 คู่มือระบบ

เป็นบทสำหรับผู้ดูแลรักษาระบบซอฟต์แวร์หรือผู้จะพัฒนาเพิ่มเติมระบบรับชำระ

1.2.7 บทที่ 7 คู่มือผู้ใช้งาน

เป็นบทสำหรับผู้ใช้งานระบบซอฟต์แวร์ของระบบรับชำระ

1.2.8 ภาคผนวก

เป็นบทที่แสดงรูปแบบรายงานต่างๆ ของหน่วยงานรับชำระการเคหะแห่งชาติ

1.3 ความเป็นมาของการพัฒนาเชิงวัตถุและ UML

ในอดีตที่ผ่านมา ซอฟต์แวร์มีความซับซ้อนเพิ่มมากขึ้น ซึ่งมีผลทำให้แอปพลิเคชั่นในปัจจุบันต้องซับซ้อนมากและพัฒนาขึ้นจากความต้องการที่มากกว่าเดิม ด้วยเทคนิคการพัฒนาซอฟต์แวร์ เครื่องมือ และเทคโนโลยีที่เปลี่ยนแปลงได้อย่างรวดเร็ว จึงต้องหาวิธีการในการพัฒนาระบบที่สามารถรองรับกับการเปลี่ยนแปลงดังกล่าวได้ ซึ่งก็คือ การพัฒนาระบบเชิงวัตถุ (Object

Oriented Systems Development) และประโยชน์หลักๆ ของการพัฒนาเชิงวัตถุ คือเพิ่มความน่าเชื่อถือและเพิ่มผลผลิตของผู้พัฒนาให้สูงขึ้นได้

การพัฒนาระบบเชิงวัตถุเพิ่มความน่าเชื่อถือได้เพราะแต่ละ Object เปรียบเสมือนกล่องดำ (Black Box) ที่ภายนอก Object จะต้องมาติดต่อสื่อสารด้วย โครงสร้างข้อมูลและวิธีดำเนินการภายในสามารถจะเปลี่ยนแปลงแก้ไขได้โดยไม่มีผลกระทบกับส่วนที่อยู่ภายนอกของระบบเลย ซึ่งต่างกับการพัฒนาระบบแบบเดิม คือไม่สามารถคาดเดาถึงผลกระทบที่จะเกิดขึ้นเมื่อมีการเปลี่ยนแปลงส่วนหนึ่งส่วนใดของโค้ดนั้น นั่นก็คือเทคโนโลยีของ Object ช่วยให้ผู้พัฒนาจัดการกับความซับซ้อนภายในการพัฒนาระบบได้ และการพัฒนาระบบเชิงวัตถุสามารถเพิ่มผลผลิตของผู้พัฒนาให้สูงขึ้นได้เพราะคลาสของ Object สามารถนำกลับมาใช้ใหม่ (Reuse) ได้ เช่น ซับคลาส (Subclass) หรือการถ่ายทอดคุณลักษณะของ Object ที่คลาสสามารถใช้โค้ดโปรแกรมเดียวกันได้ การเพิ่มผลผลิตของผู้พัฒนาเป็นการเพิ่มความสัมพันธ์ระหว่าง Object ของระบบกับ Object ที่มีในโลกความเป็นจริงให้มากขึ้น ทำให้การพัฒนาแอปพลิเคชั่นใช้ระยะเวลาที่สั้นลงเนื่องจาก Object ที่ได้มา มีการรู้จักมาครั้งหนึ่งแล้ว ลักษณะของ Object ที่ได้นั้นมีความเป็นธรรมชาติมากขึ้น เพราะเก็บทั้งข้อมูลและโปรแกรมไว้ที่เดียวกัน และแต่ละ Object ที่ได้นั้นมีหน้าที่เป็นของตนเองด้วย พร้อมยังเกิดความสะดวกสบายในการที่จะนำ Object นั้นไปแทนที่ ปรับปรุงและนำกลับมาใช้ใหม่ได้

ภายใต้การพัฒนาระบบเชิงวัตถุ มีวิธีการเชิงวัตถุ (Object Oriented Method) เกิดขึ้นมากมายหลายแบบ เพื่อมารองรับการพัฒนาระบบเชิงวัตถุ วิธีการเชิงวัตถุนั้นประกอบไปด้วย 2 ส่วนใหญ่ๆ คือกระบวนการ (Process) ซึ่งเป็นกิจกรรมที่ใช้ในส่วนต่างๆ ของการพัฒนาระบบ และภาษาที่ใช้ในการสร้างแบบจำลอง (Modeling Language) ซึ่งประกอบไปด้วยสัญลักษณ์ (Notation) ที่ใช้ในแบบจำลองและกฎเกณฑ์ต่างๆ ในการใช้สัญลักษณ์เหล่านั้น

แต่ละวิธีการเชิงวัตถุก็จะมีกระบวนการและภาษาที่ใช้ในการสร้างแบบจำลองเป็นของผู้คิดค้นวิธีการเชิงวัตถุแต่ละท่าน และลักษณะโดยทั่วไปแล้วกระบวนการในการพัฒนาระบบเชิงวัตถุของแต่ละวิธีการเชิงวัตถุ จะประกอบไปด้วย 5 ขั้นตอนคือ การวิเคราะห์ศึกษาถึงความต้องการ (Requirement Analysis) การวิเคราะห์ (Analysis) การออกแบบ (Design) การสร้างโปรแกรม (Programming) และการทดสอบ (Test) และผลของการที่มีวิธีการเชิงวัตถุเกิดขึ้นมากมายนั้น ทำให้มีภาษาที่ใช้ในการสร้างแบบจำลองเกิดขึ้นมากมายเช่นกัน ซึ่งเป็นเรื่องยุ่งยากในการพัฒนาระบบ

ในที่สุดได้มีการสร้างภาษาที่ใช้ในการสร้างแบบจำลองขึ้นมาใหม่ ชื่อว่า UML (Unified Modeling Language) และมีความเป็นไปได้ที่จะเป็นภาษาที่ใช้ในการสร้างแบบจำลองที่เป็นมาตรฐานในอนาคต

บทที่ 2

ทฤษฎีที่เกี่ยวข้อง

2.1 UML (Unified Modeling Language)

UML เป็นภาษาที่ใช้ในการสร้างแบบจำลอง (Modeling Language) และกระบวนการต่างๆ สำหรับการพัฒนาระบบที่ใช้เทคโนโลยีของ Object-Oriented โดยการรวมวิธีการของ Grady Booch (Booch method) James Rumbaugh (OMT-2 : Object Modeling Technique) และ Ivar Jacobson (OOSE/Objectory method) เป็นพื้นฐานของ UML และยังรวมวิธีการที่ดีของคนอื่นๆ เข้าไปด้วย เช่น State Chart (ของ David Harel) ใช้เป็น State Diagram ใน UML เป็นต้น

โดยปกติแล้วการสร้างแบบจำลองนั้นจะใช้กับระบบซอฟต์แวร์เท่านั้นแต่ UML สามารถใช้ได้กับระบบอื่นๆ ดังนี้

- ระบบสารสนเทศ (Information System) เช่น แสดงความสัมพันธ์ระหว่างข้อมูลกับผู้ใช้
- ระบบเทคนิค (Technical System) เช่น การสื่อสาร ระบบทางการทหาร หรือกระบวนการทางอุตสาหกรรม (Industrial Process) เป็นต้น
- ระบบที่เป็นส่วนหนึ่งของการทำงานแบบเรียวไทม์ (Embedded Real-time System) เช่น โทรศัพท์มือถือ รถยนต์ เป็นต้น
- ระบบที่มีการประมวลผลแบบกระจาย (Distributed System) เช่น CORBA DCOM หรือ RMI เป็นต้น
- ซอฟต์แวร์ระบบ (System Software) เช่น ระบบปฏิบัติการ (Operating System)
- ระบบธุรกิจ (Business System) เช่น กระบวนการของธุรกิจ (Business process)

UML สามารถใช้สำหรับการสร้างแบบจำลองเพื่ออธิบายระบบต่างๆ ได้อย่างกว้างขวาง แล้วยังสามารถใช้ได้กับทุกๆ ขั้นตอนของการพัฒนาระบบคือตั้งแต่การกำหนดความต้องการจนถึงการตรวจสอบระบบ

UML ประกอบด้วยส่วนหลักๆ ดังนี้

- มุมมอง (View) เป็นการแสดงมุมมองในด้านต่างๆ ของระบบ ซึ่งจะประกอบไปด้วยไดอะแกรมต่างๆ ในแต่ละมุมมอง

- ไดอะแกรม (Diagram) เป็นรูปภาพที่ใช้อธิบายมุมมองของระบบ ซึ่ง UML มีไดอะแกรมทั้งหมด 9 ไดอะแกรม ต่อไปนี้

  1. Use-Case Diagram
  2. Class Diagram
  3. Object Diagram
  4. State Diagram
  5. Sequence Diagram
  6. Collaboration Diagram
  7. Activity Diagram
  8. Component Diagram
  9. Deployment Diagram

- ส่วนประกอบของแบบจำลอง (Model element) เป็นส่วนที่ใช้ภายในไดอะแกรม เช่น คลาส (Class) ออบเจ็กต์ (Object) หรือความสัมพันธ์ เป็นต้น
- วิธีทั่วไป (General mechanism) ใช้ในการเพิ่มประสิทธิภาพให้กับ UML เช่นการเพิ่มหมายเหตุ (Note) หรือการตกแต่ง (Adornment) ส่วนประกอบของแบบจำลองในไดอะแกรม

ธรรมดาแล้วระบบทุกระบบจะมีโครงสร้างแบบสแตติก (Static Structure) และพฤติกรรมแบบ ไดนามิก (Dynamic Behavior) ลักษณะของภาษาที่ใช้ในการสร้างแบบจำลองบางภาษาอาจไม่ครอบคลุม แต่ UML สามารถรองรับได้ทั้งโครงสร้างแบบสแตติกและพฤติกรรมแบบไดนามิกของระบบได้ ซึ่งใช้ Class Diagram และ Object Diagram อธิบายโครงสร้างแบบสแตติกของระบบและใช้ State Diagram Sequence Diagram Collaboration Diagram และ Activity Diagram อธิบายพฤติกรรมแบบไดนามิกของระบบ และด้วยเหตุนี้เองจึงทำให้ในปัจจุบัน UML ได้เป็นที่นิยมของผู้วิเคราะห์ระบบและออกแบบระบบในเชิงวัตถุ

2.1.1 Static Modeling

UML มี Class Diagram สำหรับรองรับโครงสร้างแบบสแตติกของระบบต่างๆ Class Diagram จะอธิบายถึงสิ่งที่มีอยู่ภายในระบบและความสัมพันธ์ระหว่างสิ่งเหล่านั้น และ Object Diagram จะเป็นมุมมองตัวอย่างของ Class Diagram แต่ไดอะแกรมเหล่านั้นไม่อธิบายถึงวิธีการที่สิ่งเหล่านั้นมีการทำกิจกรรมอะไรภายในระบบหรือกล่าวอีกอย่างหนึ่งเพื่อความเข้าใจคือ Class Diagram และ Object Diagram จะบอกว่าระบบทำอะไร แต่ไม่ได้บอกว่าทำอย่างไร

ไดอะแกรมอีกแบบหนึ่งที่อธิบายถึงโครงสร้างของระบบคือ Use-Case Diagram ซึ่งเป็นไดอะแกรมที่ต่างกับ Class Diagram และ Object Diagram ตรงที่มี Use-Case Diagram เป็นไดอะแกรมที่มีการสร้างแบบจำลองในมุมมองจากภายนอกระบบ ส่วน Class Diagram และ Object Diagram นั้นเป็นไดอะแกรมที่มีการสร้างแบบจำลองในมุมมองที่อยู่ภายในระบบ Use-Case Diagram เป็นไดอะแกรมที่สามารถช่วยในการออกแบบ Class Diagram รวมถึง Object Diagram อีกด้วย เพื่อการแสดงโครงสร้างของระบบโดยสมบูรณ์ จึงใช้ไดอะแกรมทั้ง 3 ไดอะแกรมช่วยในการแสดงโครงสร้างแบบของระบบ

2.1.1.1 Use-Case Diagram

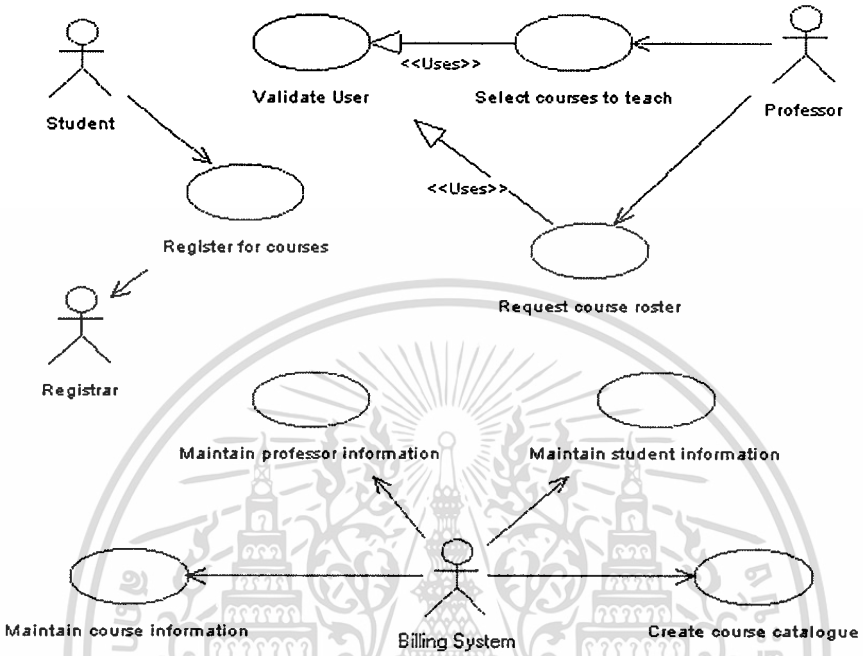
Use-Case Diagram เป็นมุมมองรูปภาพของความสัมพันธ์ระหว่างสิ่งที่อยู่ภายนอกกับสิ่งที่อยู่ภายในระบบ ทำหน้าที่เป็นตัวกลางในการสื่อสารระหว่างผู้พัฒนากับผู้ใช้งานเป็นหลัก

ความจริงในทางปฏิบัติจะมีการอธิบายส่วนของ Use-Case ในรูปแบบตัวอักษรอยู่ส่วนของคุณสมบัติเอกสาร (Document Property) ของ Use-Case

ส่วนประกอบของ Use-Case Diagram

1. Actor จะใช้สัญลักษณ์แทนด้วยรูปคน (Stickman) โดยมีชื่อของ Actor อยู่ใต้รูป ดังแสดงในรูปที่ 2.1 ซึ่งเป็นส่วนประกอบที่แทนด้วยสิ่งอื่นๆ หรือคนอื่นๆ ซึ่งอยู่ภายนอกระบบและมีการโต้ตอบ (Interact) กับระบบ อาจเป็นเพียงส่งอินพุตเข้าสู่ระบบ หรือเป็นเพียงการรับเอาท์พุตจากระบบ หรือแม้แต่เป็นการทั้งอินพุตและเอาท์พุตให้กับระบบก็เป็นได้
2. Use case คือลำดับของกิจกรรมของระบบที่ต้องให้ผลลัพธ์ที่มีตัวตนตรงกับความต้องการของ Actor จะใช้สัญลักษณ์รูปวงรี โดยมีชื่อของ Use-Case อยู่ภายในวงรีหรือใต้รูปวงรีก็ได้ (ดังแสดงในรูปที่ 2.1) เป็นส่วนประกอบที่แทนด้วยหน้าที่หนึ่งๆ ที่สมบูรณ์ที่ระบบมีการทำงาน
3. ความสัมพันธ์ภายใน Use-Case Diagramจะมีความสัมพันธ์ 3 ประเภท คือ ระหว่าง Use-Cases ระหว่าง Actors และระหว่าง Actor กับ Use-Case ดังแสดงในรูปที่ 2.1
   - ความสัมพันธ์ระหว่าง Use-Cases ด้วยกันจะมี 2 ลักษณะคือ Use และ Extend ทั้ง 2 เป็นความสัมพันธ์แบบ Generalization ความสัมพันธ์แบบ Use จะแทนด้วยลูกศรหัวปิดที่ชี้ไปยัง Use Case ที่จะใช้พร้อมกับมี <<Uses>> กำกับ เป็นการอธิบายว่า Use Case นั้นเป็นส่วนหนึ่งของ Use Case ที่ใช้งาน ส่วนความสัมพันธ์แบบ Extend จะแทนด้วยลูกศรหัวปิดที่ชี้ไปยัง Use Case ที่เป็นจุดเริ่ม (Base) พร้อมกับ

มี <<Extends>> กำกับ ส่วนมากความสัมพันธ์แบบ Extend จะใช้ในกรณีของทางเลือกที่มีเงื่อนไขที่ชัดเจน



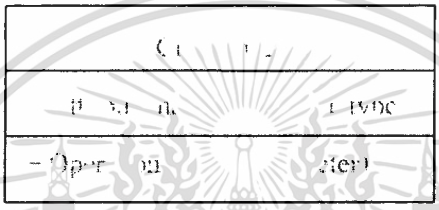
รูปที่ 2.1 ตัวอย่าง Use-Case Diagram

- ความสัมพันธ์ระหว่าง Actor มีเพียงความสัมพันธ์แบบ Generalization จะแทนด้วยลูกศรหัวปิดที่ชี้ไปยัง Actor ที่เป็นลักษณะของซุปเปอร์คลาส (Super Class) เป็นการถ่ายทอด (Inherit) คุณลักษณะของซุปเปอร์คลาสมา
- ความสัมพันธ์ระหว่าง Actor กับ Use Case มีเพียงความสัมพันธ์แบบ Association ซึ่งแทนด้วยลูกศรหัวเปิด (หรือไม่มีหัวลูกศรในกรณีของทั้ง Actor และ Use Case มีการความสัมพันธ์แบบ 2 ทาง) ซึ่งปลาย (หาง) ลูกศรที่ผู้เริ่มการติดต่อสื่อสาร

2.1.1.2 Class Diagram

เป็นรูปภาพที่อธิบายถึงสิ่งที่อยู่ภายในระบบในรูปแบบของคลาส และความสัมพันธ์ระหว่างคลาสเหล่านั้นแล้ว Class Diagram ยังเป็นพื้นฐานในการสร้าง Diagram อื่นๆ ด้วย ส่วนประกอบของ Class Diagram

1. คลาสซึ่งจะใช้รูปสี่เหลี่ยมแบ่งออกเป็น 3 ส่วนคือส่วนบนจะเป็นส่วนของชื่อ แสดงเป็นตัวหนาจัดอยู่กึ่งกลาง ส่วนกลางจะเป็นส่วนของคุณลักษณะ (Attribute) พร้อมด้วยการกำหนดประเภทและความสามารถในการมอง ซึ่งจะใช้ (+) แทน Public (-) แทน Private และ (#) แทน Protect และส่วนล่างจะเป็นส่วนของพฤติกรรม (Operation) พร้อมด้วยพารามิเตอร์และประเภทของค่าที่มีการส่งคืน และยังรวมถึงแสดงความสามารถในการมองด้วย หรืออาจจะแสดงในลักษณะของชื่ออย่างเดียว โดยแสดงดังรูปที่ 2.2 และรูปที่ 2.3 ตามลำดับ

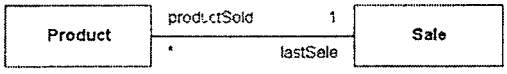


รูปที่ 2.2 แสดงรูปแบบของคลาสที่แบ่งออกเป็น 3 ส่วน

รูปที่ 2.3 แสดงรูปแบบของคลาสที่ไม่แบ่งส่วน

ความสัมพันธ์ระหว่างคลาสที่ใช้กันสามารถแบ่งได้เป็น 4 ความสัมพันธ์หลักๆ คือ

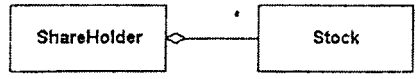
- Association เป็นความสัมพันธ์แบบสองทางระหว่างคลาส แสดงถึงความเกี่ยวข้องกันระหว่างคลาส ใช้สัญลักษณ์เส้นตรงและมีชื่อของความสัมพันธ์กำกับด้วย ดังแสดงในรูปที่ 2.4



รูปที่ 2.4 ตัวอย่างแสดงความสัมพันธ์แบบ Association ภายใน Class Diagram

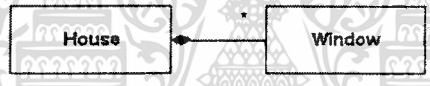
ความสัมพันธ์แบบ Association แบบหนึ่งที่แสดงถึงความสัพพันธ์แบบคลาสหนึ่งเป็นส่วนหนึ่ง ของอีกคลาสหนึ่ง (Whole-Part) ซึ่งเป็นความสัมพันธ์ที่

เรียกว่า ความสัมพันธ์แบบ Aggregation โดยใช้สัญลักษณ์ลูกศรที่มีหัวเป็นรูปสี่เหลี่ยมขนมเปียกปูนโปร่งอยู่ทางด้านที่เป็นทั้งหมด (Whole) ดังรูปที่ 2.5



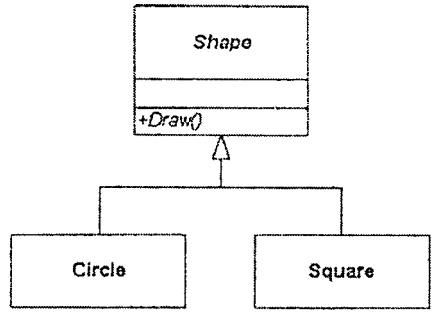
รูปที่ 2.5 ตัวอย่างแสดงความสัมพันธ์แบบ Aggregation ภายใน Class Diagram

ยังมีความสัมพันธ์แบบ Aggregation แบบ Composition Aggregation ซึ่งเป็นลักษณะความสัมพันธ์ที่มีความเกี่ยวข้องกันมากระหว่าง 2 ส่วน คือส่วนทั้งหมดและส่วนที่เป็นบางส่วน (Part) ใช้สัญลักษณ์ลูกศรที่มีหัวเป็นรูปสี่เหลี่ยมขนมเปียกปูนทึบอยู่ทางด้านที่เป็นทั้งหมด แสดงถึงว่าถ้าไม่มีคลาสทางด้านบางส่วน (part) จะไม่สามารถมีคลาสทางด้านทั้งหมด (Whole) ได้ โดยแสดงดังรูปที่ 2.6



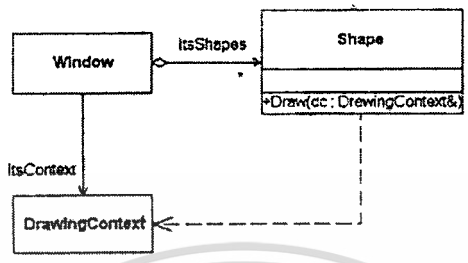
รูปที่ 2.6 ตัวอย่างแสดงความสัมพันธ์แบบ Composition Aggregation ภายใน Class Diagram

- Generalization เป็นความสัมพันธ์ที่แบ่งระหว่าง ซุปเปอร์คลาสกับซับคลาส (Superclass/Subclass) หรือเป็นความสัมพันธ์ที่บางครั้งเรียกว่าความสัมพันธ์แบบ "is-a relationship" ใช้สัญลักษณ์ลูกศรหัวปิดที่ชี้ไปยังซุปเปอร์คลาส แสดงถึงการถ่ายทอดทั้งคุณลักษณะและพฤติกรรมจากซุปเปอร์คลาสทั้งหมดไปยังซับคลาส โดยแสดงดังรูปที่ 2.7



รูปที่ 2.7 ตัวอย่างความสัมพันธ์แบบ Generalization ภายใน Class Diagram

- Dependency เป็นความสัมพันธ์ในลักษณะที่มีการขึ้นตรงต่อกัน คือเมื่อมีการแก้ไขสิ่งหนึ่งจะมีผลกับอีกสิ่งหนึ่งที่ขึ้นต่อกัน (Dependent) ใช้สัญลักษณ์เป็นเส้นประและลูกศรหัวเปิด ดังแสดงในรูปที่ 2.8



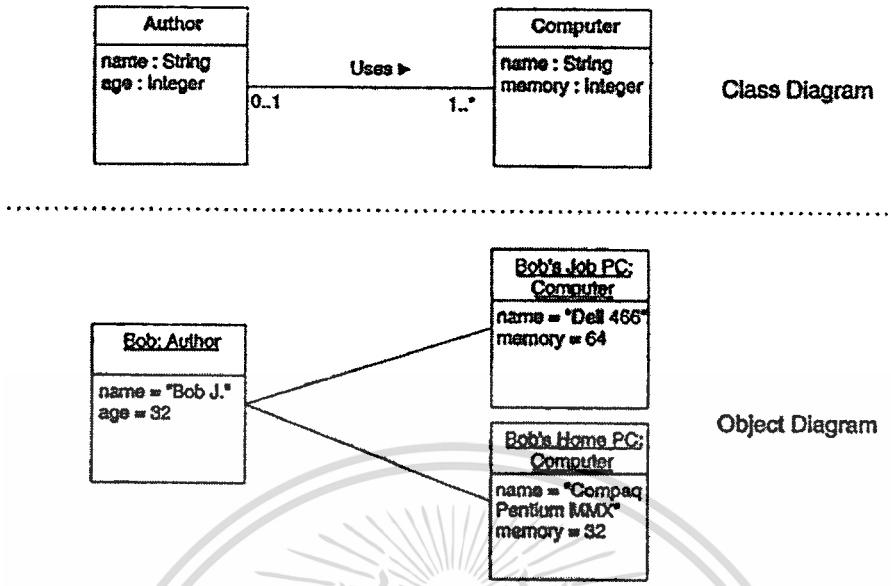
รูปที่ 2.8 ตัวอย่างความสัมพันธ์แบบ Dependency ภายใน Class Diagram

- Refinement เป็นความสัมพันธ์ของรูปแบบการอธิบาย 2 แบบซึ่งอธิบายในสิ่งๆ เดียวกัน แต่คนละมิติกัน ใช้สัญลักษณ์เป็นเส้นประและลูกศรหัวปิด ดังแสดงในรูปที่ 2.9

รูปที่ 2.9 ตัวอย่างความสัมพันธ์แบบ Refinement ภายใน Class Diagram

2.1.1.3 Object Diagram

เป็นรูปภาพที่แสดงข้อมูลของคลาสและความสัมพันธ์ระหว่างข้อมูลขณะเวลาใดเวลาหนึ่ง ซึ่ง Object Diagram สามารถเป็นมุมมองตัวอย่างของ Class Diagram ได้และยังสามารถเป็นตัวอย่างของ Class Diagram ที่ซับซ้อนเพื่อช่วยให้เกิดความเข้าใจได้ง่ายขึ้น สัญลักษณ์ที่ใช้เหมือนกับของ Class Diagram แต่ ชื่อของ Object ต้องขีดเส้นใต้หรือเป็นชื่อที่นำหน้าชื่อของคลาสและคั่นด้วย ":" จะได้รูปแบบคือ "ชื่อ Object : ชื่อคลาส" ดังแสดงในรูปที่ 2.10 ซึ่งเป็นรูปที่แสดงเปรียบเทียบระหว่าง Class Diagram กับ Object Diagram



รูปที่ 2.10 แสดงการเปรียบระหว่าง Class Diagram กับ Object Diagram

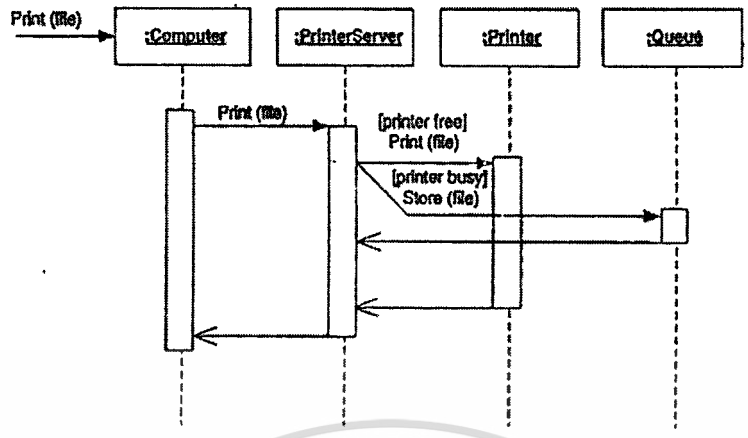
2.1.2 Dynamic Model

เป็น Model ที่สามารถอธิบายพฤติกรรมของระบบโดยการนำเสนอตัวอย่างของเหตุการณ์ที่เกิดขึ้นภายในระบบนั้นๆ รวมถึงเป็นการนำเสนอวิธีการของสิ่งที่มีอยู่ภายในระบบว่ามีการติดต่อสื่อสารกันอย่างไร โดยมีการส่งข้อความ (Message) ให้กันและกัน ซึ่ง Dynamic Model นี้คล้ายกับเป็นการเพิ่มรายละเอียดของระบบเพื่อให้เหมาะสมกับการสร้างหรือพัฒนาต่อไป

2.1.2.1 Sequence Diagram

เป็นรูปภาพที่อธิบายถึงความสัมพันธ์ของ Object ในมิติของเวลา กล่าวคือจะมีเรียงลำดับ Object ตามแนวนอน และส่วนในแนวตั้งจะมีลำดับการส่งผ่านข้อความจาก Object หนึ่งไปยัง Object หนึ่ง ที่เกิดขึ้นตามลำดับของเหตุการณ์ และมีหมายเลขกำกับการส่งผ่านข้อความเหล่านั้น

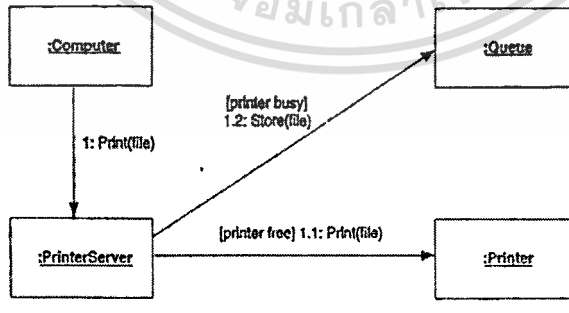
ใน Sequence Diagram จะใช้สัญลักษณ์สี่เหลี่ยมผืนผ้าแทน Object หรือ Class และสัญลักษณ์เส้นประในแนวตั้งเรียกว่าเส้นชีวิตของ Object (object's lifeline) ซึ่งเป็นการกำหนดขอบเขตการทำงานของ Object นั้น ดังแสดงตัวอย่างดังรูปที่ 2.11



รูปที่ 2.11 แสดงตัวอย่างของ Sequence Diagram

2.1.2.2 Collaboration Diagram

เป็นรูปภาพที่อธิบายถึง Object และความสัมพันธ์กันระหว่าง Object ในลักษณะ การส่งข้อความไปมาระหว่างผู้ส่ง (sender) และผู้รับ (suplier) โดยไม่มีลำดับ (space) สัญลักษณ์ที่ใช้เหมือนกับ Sequence Diagram ในการทำงาน Collaboration Diagram จะคล้ายกับ Sequence Diagram ซึ่งสามารถเลือกแสดงได้ในรูปแบบใดรูปแบบหนึ่ง กล่าวคือทั้งสองแผนผังสามารถเปลี่ยนแปลงรูปแบบระหว่างกันได้ ซึ่งสามารถเรียกแผนผังทั้งสองอย่างได้อีกอย่างหนึ่งว่า Interaction Diagram โดย Collaboration Diagram เหมาะกับส่วนการออกแบบระบบ ตัวอย่างแสดงได้ดังรูปที่ 2.12



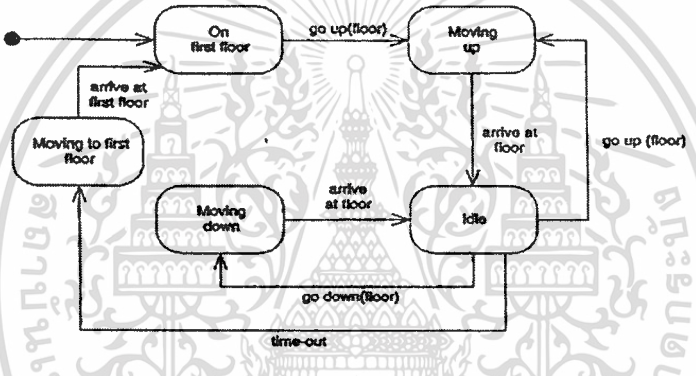
รูปที่ 2. 12 แสดงตัวอย่างของ Collaboration Diagram

โดยทั่วไปแล้วอาจใช้ Sequence Diagram หรือ Collaboration Diagram ในการแสดงเหตุการณ์หนึ่งๆ ของ Use Case ได้ ขึ้นอยู่กับความเหมาะสมและสถานการณ์มากกว่า

2.1.2.3 State Diagram

เป็นรูปภาพที่อธิบายถึงวงจรชีวิต (life cycle) ของ Object ว่ามีสถานะอย่างไรบ้าง โดยมากมักจะเขียน State Diagram เฉพาะคลาสที่มีความซับซ้อน เพื่อเป็นการแสดงสถานะของ Object ให้ชัดเจน และแสดงเหตุการณ์ที่ทำให้เกิดการเปลี่ยนแปลงของสถานะแต่ละสถานะ

ใน State Diagram ใช้สัญลักษณ์วงกลมทึบแทนจุดเริ่มต้น หรือใช้แทนความหมายว่า Object นั้นถูกสร้าง และใช้สัญลักษณ์วงกลมทึบที่มีวงกลมล้อมรอบแทนจุดสิ้นสุด (ไม่จำเป็นจะต้องมีทุกไดอะแกร) ใช้ลูกศรแสดงถึงการเปลี่ยนแปลงระหว่างสถานะ หรือแสดงเหตุการณ์ที่ทำให้สถานะนั้นเปลี่ยน และได้แสดงตัวอย่างของ State Diagram ไว้ดังรูปที่ 2.13

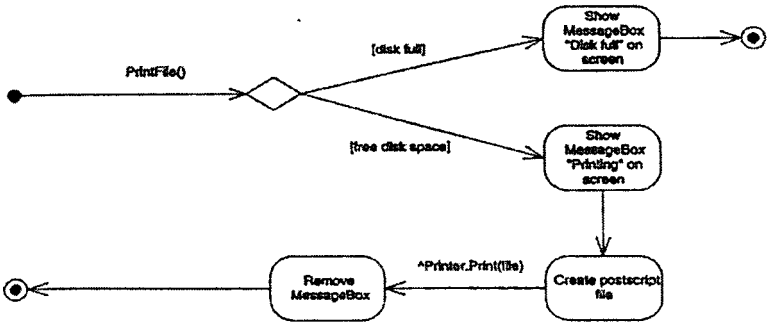


รูปที่ 2. 13 แสดงตัวอย่างของ State Diagram

2.1.2.4 Activity Diagram

เป็นรูปภาพที่มีลักษณะคล้ายกับ State Diagram แต่ต่างกันตรงที่ Activity Diagram จะเน้นที่การแสดงการกระทำต่างๆ และผลลัพธ์ที่เกิดขึ้นในรูปแบบของสถานะของ Object

การใช้สัญลักษณ์เดียวกับ State Diagram และมีการใช้สัญลักษณ์เพิ่มขึ้นมาคือสัญลักษณ์ สี่เหลี่ยมขนมเปียกปูนที่แสดงถึงการตัดสินใจของการกระทำนั้นๆ ซึ่งได้แสดงตัวอย่างไว้ดังรูปที่ 2.14 ยังรวมถึงรูปแบบในการแบ่งส่วนของการกระทำออกเป็นกลุ่มๆ เรียกว่า Swim lane ซึ่งจะมีการกำหนดชื่อให้สำหรับแต่ละกลุ่มด้วย



รูปที่ 2. 14 แสดงตัวอย่างของ Activity Diagram

2.1.3 โครงสร้างภายนอก

โครงสร้างภายนอก (Physical Architecture) ซึ่งใช้อธิบายถึงการแบ่งส่วนของซอฟต์แวร์และฮาร์ดแวร์ เป็นการแปลง (Mapping) จากโครงสร้างภายใน (Logical Architecture) เป็นโครงสร้างภายนอก ซึ่งจะเป็นการเปลี่ยนรูปแบบของคลาสให้อยู่ในรูปแบบของ Components, Process และคอมพิวเตอร์ในโครงสร้างภายนอก โดยสามารถแบ่งแผนผังได้ 2 แผนผังคือ

2.1.3.1 Component Diagram

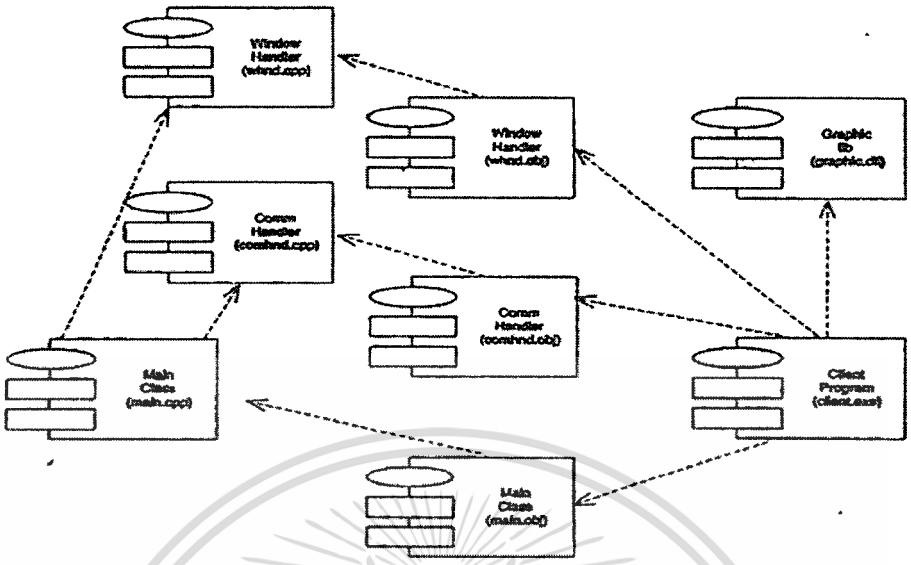
Component คือการแสดงถึงสิ่งที่ถูกสร้างขึ้นในโครงสร้างภายนอกตามหน้าที่ที่ถูกกำหนดภายในโครงสร้างภายใน เช่นคลาส Object หรือเป็นการทำงานร่วมกันระหว่าง Object เป็นต้น

Component Diagram จะแสดงถึงไฟล์ที่มีในระบบในสภาวะแวดล้อมของการพัฒนาระบบ ใช้สัญลักษณ์รูปสี่เหลี่ยมผืนผ้าที่มีรูปวงรีและสี่เหลี่ยมผืนผ้าเล็กๆสองอันอยู่ทางด้านซ้าย แทน Component และใช้สัญลักษณ์เส้นประที่มีลูกศรหัวเปิดแทนการเชื่อมต่อกันระหว่าง Component

Component สามารถเป็นสิ่งต่างๆ ได้ดังนี้

- Source Component เป็นโค้ดไฟล์ของโปรแกรม
- Binary Component เป็น Object โค้ดไฟล์ (อาจใช้แทน Library ไฟล์)
- Executable Component เป็น Execute ไฟล์

ได้แสดงตัวอย่าง Component Diagram ไว้ดังรูปที่ 2.15

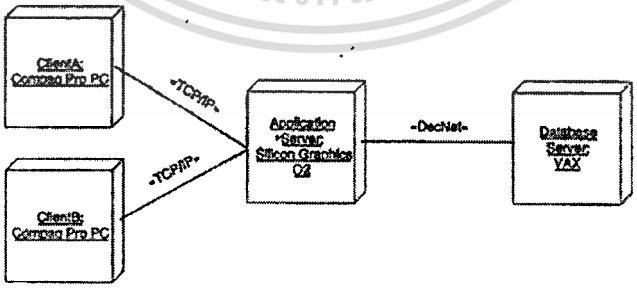


รูปที่ 2. 15 แสดงตัวอย่างของ Component Diagram

2.1.3.2 Deployment Diagram

เป็นรูปภาพที่อธิบายถึงสถาปัตยกรรมของระบบ ขณะโปรแกรมกำลังทำงาน (Run-Time) รวมถึงการติดต่อสื่อสารกันระหว่างโหนด (Node) ซึ่งเป็นโครงสร้างที่รวมโปรเซสเซอร์ อุปกรณ์ต่างๆ และซอฟต์แวร์ ต่างๆไว้ด้วยกัน

ใช้สัญลักษณ์ที่ใช้กล่องสี่เหลี่ยม แทนโหนด และใช้สัญลักษณ์เส้นทึบแทนการเชื่อมโยงกันระหว่างโหนด โดยมีป้ายกำกับบอกถึงความสัมพันธ์ในการเชื่อมโยง ซึ่งอาจแสดงเป็นโปรโตคอล ดังแสดงตัวอย่างในดังรูปที่ 2.16



รูปที่ 2.16 แสดงตัวอย่างของ Deployment Diagram

2.2 การวิเคราะห์เชิงวัตถุ

วัตถุประสงค์หลักของการวิเคราะห์คือการกำหนดความต้องการของระบบที่สมบูรณ์ (Complete) ชัดเจน (Unambiguous) และถูกต้อง (Consistent) รวมถึงสิ่งที่ระบบต้องทำตามความจำเป็นและความต้องการของผู้ใช้งาน การที่จะได้สิ่งดังกล่าวมานั้นจะต้องทำอาศัยการสร้างแบบจำลองหลายๆ แบบจำลองขึ้นมา ซึ่งจะเป็นเพียงการอธิบายถึงสิ่งที่ระบบทำงานมากกว่าวิธีที่ระบบควรจะทำ ดังนั้นควรจะเป็นมุมมองของผู้ใช้งานมากกว่าจะเป็นมุมมองจากระบบการทำงาน

การวิเคราะห์เป็นกระบวนการของการเปลี่ยนปัญหาต่างๆ ที่ไม่ชัดเจนให้อยู่ในรูปของความต้องการของระบบที่เกี่ยวข้องกันอย่างชัดเจน หรือเป็นการกำหนดความแน่นอนของความต้องการ (Requirement Determination)

การที่จะเข้าใจความต้องการของผู้ใช้งานได้นั้น จะต้องค้นหาว่าผู้ใช้งานเหล่านั้น "ใช้" ระบบอย่างไร และวิธีที่จะได้สิ่งเหล่านั้นมา คือการพัฒนา Use case เพราะ Use case นั้นเป็นการอธิบายสถานการณ์ต่างๆ ให้เกิดความเข้าใจความต้องการของระบบได้ โดย Use case นั้นเป็นการโต้ตอบระหว่างผู้ใช้งานกับระบบ และ Use case ยังสามารถจะกำหนดเป้าหมายของผู้ใช้งานและหน้าที่ของระบบ ซึ่งแต่ละ Use case จะแทนด้วยสิ่งที่ผู้ใช้งานต้องการทำ

แบบจำลอง Use case สามารถเป็นเครื่องมือในการพัฒนาโครงการ การวางแผน และเป็นเอกสารในการอธิบายความต้องการของระบบได้ ซึ่งแบบจำลอง Use case เป็นการอธิบายการใช้ระบบและแสดงเหตุการณ์ที่มีการปฏิบัติงาน เป็นการแสดงระบบและวิธีที่ตัวระบบจะถูกใช้งานในมุมมองของผู้ใช้เอง และนั่นก็คือหน้าที่ของระบบที่จะต้องรองรับผู้ใช้ได้

กระบวนการการวิเคราะห์เชิงวัตถุนั้นประกอบด้วยขั้นตอนต่างๆ ดังนี้

1. พัฒนาแบบจำลองของกระบวนการทางธุรกิจแบบง่ายๆ โดยใช้ UML Activity Diagram
2. กำหนด Actor
3. พัฒนา Use case
4. การสร้าง Interaction Diagram
5. การแบ่งประเภท (Classification)
6. การทำซ้ำ (Iterate) และการแก้ไขให้ดีขึ้น (Refine)

การพัฒนาแบบจำลองของกระบวนการทางธุรกิจนั้นอาจเป็นการพัฒนาที่ใช้เวลามาก ดังนั้นจึงควรสร้างแบบจำลองพื้นฐานขึ้นมาเพื่อที่จะช่วยทำความคุ้นเคยกับระบบและความต้องการของผู้ใช้งานได้ แล้วยังสามารถมีส่วนช่วยในการพัฒนา Use case ได้ด้วย การพัฒนาแบบจำลองของกระบวนการทางธุรกิจโดยใช้ Activity Diagram นั้นนอกจากจะช่วยให้เกิดความเข้าใจความ

ต้องการของระบบได้ดีขึ้นแล้ว ยังสามารถทำให้ทราบถึงลำดับของกิจกรรมต่างๆ ที่ปฏิบัติได้อีกด้วย

แต่ละขั้นตอนของกระบวนการการวิเคราะห์เชิงวัตถุ จะมีรายละเอียดดังต่อไปนี้

2.2.1 พัฒนาแบบจำลองของกระบวนการทางธุรกิจ

เป็นการสร้าง Activity Diagram สำหรับกระบวนการทางธุรกิจเพื่อให้เกิดความเข้าใจในความต้องการต่างๆ ของผู้ใช้งาน

2.2.2 การกำหนด Actor

Actor ใช้แทนบทบาทของผู้ใช้งานที่เกี่ยวข้องกับระบบ ซึ่งไม่ใช่คนหรือตำแหน่งงาน เพราะคนหนึ่งคนอาจมีบทบาทกับระบบได้มากกว่าบทบาทเดียวก็ได้ การกำหนด Actor นั้นสามารถจะได้มาโดยการตอบคำถามดังต่อไปนี้ก็ได้

- ใครคือผู้ที่ใช้ระบบ ระบบส่งผลต่อใครหรือกลุ่มใดที่ต้องการความช่วยเหลือจากระบบให้งานให้
- ใครมีผลต่อระบบ กลุ่มใดที่ระบบจะทำงานให้ (ซึ่งจะรวมทั้งงานหลักและงานรองด้วย เช่นงานจัดการ)
- ฮาร์ดแวร์ภายนอกตัวหรือระบบอื่นๆ ใดที่ใช้ระบบ
- Application นี้แก้ปัญหาของใคร
- ผู้ใช้งานใช้ระบบนี้อย่างไร สิ่งใดที่ผู้ใช้งานเหล่านั้นทำกับระบบ

จากการตอบคำถามเหล่านี้ได้แล้วก็จะได้ Actor ขึ้นมาชุดหนึ่ง และควรจะใส่คำอธิบาย Actor แต่ละตัวที่ได้ในลักษณะของบทบาทที่ Actor นั้นๆ โต้ตอบกับระบบ และ Actor อื่นๆ อาจถูกกำหนดขึ้นในการทำซ้ำต่อไปก็ได้

2.2.3 พัฒนา Use Case

Use case จะแทนด้วยหน้าที่ ที่ระบบสามารถทำงาน โดยให้พิจารณาจากสิ่งที่ Actor ต้องการจะทำ และการหา Use case จะหาได้จากขั้นตอนต่อไปนี้คือ

1. หางานและหน้าที่ที่ Actor จะปฏิบัติ หรืองานที่ระบบต้องการให้ Actor ปฏิบัติ (สำหรับแต่ละ Actor) Use case ควรจะแทนด้วยเหตุการณ์ที่มีเป้าหมายที่ชัดเจน
2. กำหนดชื่อให้กับ Use Case โดยที่ชื่อควรจะแสดงสิ่งที่เกิดขึ้นเมื่อ Use case มีการปฏิบัติงาน และควรจะกำหนดชื่อ Use Case ให้ถูกต้องมั่นคง (Consistent)
3. อธิบาย Use Case อย่างย่อๆ โดยใช้คำที่ผู้ใช้งานคุ้นเคย เพราะจะทำให้คำอธิบายไม่ครุมเครือน้อยที่สุด

ขณะที่ทำการหา Use Case อาจจะต้องมีการเปลี่ยนแปลง Actor ได้ และการเปลี่ยนแปลงควรจะกระทำโดยระมัดระวังเพราะการเปลี่ยนแปลง Actor จะมีผลต่อ Use Case ด้วย เมื่อกำหนด Use Case ได้ก็อาจจะค้นพบการความแตกต่างบางอย่างของ Use Case และนั่นคือการที่สามารถจะนำ Use Case กลับมาใช้ใหม่ (Reuse) โดยอาศัยความสัมพันธ์แบบ Uses และ Extends ได้

จากนั้นก็จะเป็นการนำ Actor และ Use Case มาสร้างเป็นแบบจำลอง Use Case ในรูปแบบของ Use Case Diagram ได้ และยังสามารถจะใช้ Package เข้ามาช่วยในการแบ่งกลุ่ม Use Case เพื่อช่วยทำให้ระบุเหตุการณ์ของระบบให้แคบลงได้

2.2.4 การเตรียม Interaction Diagram

เป็น Interaction Diagram ที่อธิบายถึงวิธีการทำงานร่วมกัน ของกลุ่มของ Object เพื่อให้ได้งานหนึ่งๆ มา ซึ่ง Interaction Diagram จะแสดงถึงรูปแบบพฤติกรรมของ Object ที่โต้ตอบกันใน Use Case หนึ่ง

การเลือกใช้ Diagram สำหรับการอธิบาย Use Case หนึ่งๆ นั้นจะขึ้นอยู่กับ ความต้องการที่ให้แสดง หากต้องการแสดงการโต้ตอบของกลุ่มของ Object ตามลำดับเวลานั้นอาจจะต้องใช้ Sequence Diagram หรือหากต้องการแสดงความสัมพันธ์ระหว่างกลุ่มของ Object ก็อาจต้องใช้ Collaboration Diagram ซึ่งก็หมายความว่า อาจจะใช้ไดอะแกรมใดไดอะแกรมหนึ่งในการอธิบายเหตุการณ์ที่เกิดขึ้นของ Use Case หนึ่งๆ ก็ได้ หรืออาจใช้ทั้งสองไดอะแกรมร่วมกัน และอย่างน้อยในแต่ละ Use Case ควรจะอธิบายเหตุการณ์หลักๆ ไว้อย่างน้อยหนึ่งเหตุการณ์ ทั้งนี้ขึ้นอยู่กับความต้องการในการแสดงรายละเอียดของเหตุการณ์ที่เกิดขึ้น

2.2.5 การแบ่งประเภท (Classification)

การแบ่งประเภทจะเป็นการพัฒนา UML Class Diagram และการแบ่งประเภทในที่นี้คือการจำแนก Object ต่างๆ ให้อยู่ในรูปแบบของ Class ซึ่งจะอาศัยแบบจำลอง Use case ที่ได้สร้างมาในขั้นต้น ซึ่งได้ถูกอธิบายในรูปแบบของเหตุการณ์ในลักษณะทั้งตัวอักษรและรูปภาพ (Diagram) ที่กำหนดถึง Object ที่ต้องการในเหตุการณ์ที่เกิดขึ้น และนี่เป็นการเริ่มใช้สิ่งที่มีอยู่ให้เป็นประโยชน์

การตั้งชื่อให้ Class ซึ่งเป็นกิจกรรมที่สำคัญอีกประการหนึ่ง และอาจใช้แนวทางดังต่อไปนี้

- ชื่อของ Class ควรเป็นเอกพจน์ เพราะ Class ควรจะอธิบาย Object หนึ่งๆ เท่านั้น ซึ่งอาจเป็นคำนามหรือเป็นคำนามที่รวมกับคำที่ขยายคำนามก็ได้
- ควรใช้ชื่อของ Class ที่ผู้ใช้งานพอใจ ซึ่งอาจเป็นคำศัพท์ที่อยู่ในสายงานของผู้ใช้งานเอง
- ชื่อของ Class ควรจะสื่อความหมายด้วยตัวเอง และชื่อของ Class ควรจะสามารถอ่านแล้วเข้าใจได้

2.2.5.1 การกำหนดความสัมพันธ์ระหว่าง Class

โดยทั่วไปแล้วความสัมพันธ์ระว่าง Class จะแบ่งเป็น 3 ประเภท คือ Association, Super-Sub Structure และ Aggregation and a-part of structure

Association จะแทนด้วยการติดต่อแบบ Physical หรือ Conceptual ระหว่าง 2 Class หรือมากกว่า การกำหนดความสัมพันธ์แบบ Association จะเริ่มจากการวิเคราะห์หาการโต้ตอบระหว่าง Class หากความสัมพันธ์ที่พึ่งพากัน (Dependency Relationship) ระหว่างสอง Class หรือมากกว่าจะเป็นความสัมพันธ์แบบ Association ดังนั้นจึงต้องมีการทดสอบและหน้าที่ของ Class ที่พึ่งพากัน เช่นถ้าหากว่า Object มีหน้าที่สำหรับทำงานใดงานหนึ่งและขาดข้อมูลที่จำเป็นในการปฏิบัติงานแล้ว Object นั่นต้องมอบหน้าที่ให้ Object อื่นๆ ที่มีข้อมูลทำงานแทน

Wirfs-Brock, Wilkerson and Wiener ได้ให้คำถามที่สามารถช่วยในการกำหนดความสัมพันธ์แบบ Association ได้ดังนี้คือ

- Class สามารถทำงานให้เสร็จด้วยตัวเองได้หรือไม่
- ถ้าไม่ แล้ว Class นั้นต้องการอะไร
- และ Class อื่นใดๆ ที่มีสิ่งๆ นั่น

การตอบคำถามจากคำถามเหล่านี้จะช่วยให้สามารถกำหนดความสัมพันธ์แบบ Association ได้ ส่วนมากความสัมพันธ์แบบ Association จะตรงกับคำกริยา (Verb) หรือคำบุพบท (Preposition) เช่น Part of หรือ Next to เป็นต้น นอกจากความสัมพันธ์ที่พึ่งพากันแล้วยังมีความสัมพันธ์แบบการอ้าง (Reference) ถึงจาก Class หนึ่งไปยัง Class อื่นๆ ก็เป็นความสัมพันธ์แบบ Association เช่นกัน

พยายามตัดความสัมพันธ์แบบ Association ที่ไม่จำเป็นออกไปเช่น ความสัมพันธ์แบบ Ternary Association ซึ่งเป็นความสัมพันธ์ที่มีความซับซ้อน และหากเป็นไปได้ จงแปลง Ternary Association ให้เป็นความสัมพันธ์แบบ Binary Association และความสัมพันธ์ที่ไม่จำเป็นคือความสัมพันธ์แบบการอ้างถึง (Directed Action Association / Derived Association) ซึ่งควรจะกำหนดให้อยู่ในรูปแบบของความสัมพันธ์อื่นๆ เพราะความสัมพันธ์แบบอ้างถึงนั้นเป็นความสัมพันธ์ที่ซ้ำซ้อนกัน

Super-Sub Structure หรือที่ทราบกันเป็น Generalization Hierarchy เป็นความสัมพันธ์ที่แสดงถึงการความสัมพันธ์ที่ถ่ายทอดคุณลักษณะของ Class กล่าวคืออนุญาตให้ Object สร้างมาจาก Object อื่นๆ ได้ ซึ่งจะมี Class หนึ่งเป็น Parent (หรือที่เรียกว่าเป็น Base หรือ Super Class หรือ Ancestor)

รูปแบบและแนวทางสำหรับการกำหนดความสัมพันธ์แบบ Super-Sub Relationship (A Generalization)

- Top-Down เป็นลักษณะของการมองที่ชื่อของ Class โดยมองคำนามที่ประกอบไปด้วยคำขยายคำนาม
- Bottom-Up เป็นลักษณะของการมอง Attribute และ Method ของแต่ละ Class ที่เหมือนกัน แล้วรวมกลุ่มเป็น Common Attribute และ Common Method (ใน Super Class) และอาจจะต้องมีการเปลี่ยนแปลงการกำหนดเพียงเล็กน้อย
- Reusability เป็นการนำ Structure ที่ได้นั้นมากลับมาใช้ใหม่โดย การสร้างเป็น Sub-Class (หรือ Specialized Class) อย่าพยายามสร้าง Class ที่มีความเฉพาะเจาะจงไว้บนตำแหน่งบนของ Structure
- Multiple Inheritance เป็นรูปแบบความสัมพันธ์ที่ให้ Class สามารถมี Parent ได้มากกว่าหนึ่ง Class ซึ่งเป็นรูปแบบที่ควรหลีกเลี่ยงรูปแบบนี้ และควรใช้ความสัมพันธ์ในรูปแบบนี้ในกรณีที่เหมาะสมเท่านั้น เพราะเป็นรูปแบบที่มีความซับซ้อนและในทางปฏิบัติในหลาย Ancestor อาจมีชื่อของ Method เหมือนกันก็เป็นได้ ทางหนึ่งที่ใช้แก้คือ ให้ถ่ายทอดคุณลักษณะของ Class หนึ่งที่เหมาะสมที่สุดแล้ว ให้เพิ่ม Attribute ของอีก Class ลงไปแทน

Aggregation and a-part of structure เป็นความสัมพันธ์ที่แทนด้วย Class ที่ประกอบด้วย Class อื่นๆ หลาย Class ที่มีพฤติกรรมที่ไม่เหมือนกัน หรือแตกต่างกันอย่างมากเช่น Class รถยนต์ประกอบไปด้วย Class หลายๆ Class ซึ่งหนึ่งในนั้นอาจเป็น Class วิทยุ และจะเห็นอย่างชัดเจนว่าพฤติกรรมของ Class รถยนต์และ Class วิทยุนั้นเป็นพฤติกรรมที่ไม่เหมือนกันเลย

ความสัมพันธ์แบบ Aggregation เป็นกรณีพิเศษของความสัมพันธ์แบบ Association และยังมีลักษณะคล้ายกับความสัมพันธ์แบบ Association อย่างมาก ทั้งนี้ในการเลือกใช้จะขึ้นอยู่กับปัญหาของระบบเอง และความสัมพันธ์แบบ Aggregation จะมีลักษณะสำคัญหลักๆ อยู่ 2 ประการคือ

- Transitivity เป็นคุณลักษณะที่เกิดขึ้นคือ ถ้า A เป็นส่วนหนึ่งของ B และ B เป็นส่วนหนึ่งของ C แล้ว A ก็เป็นส่วนหนึ่งของ C ด้วย เช่น คาร์บูเรเตอร์เป็นส่วนหนึ่งของเครื่องยนต์ และเครื่องยนต์เป็นส่วนหนึ่งของรถยนต์ ดังนั้น คาร์บูเรเตอร์ก็เป็นส่วนหนึ่งของรถยนต์ด้วย
- Antisymmetry เป็นคุณลักษณะของความสัมพันธ์ที่บอกว่า ถ้า A เป็นส่วนหนึ่งของ B แล้ว B จะไม่เป็นส่วนหนึ่งของ A ตัวอย่างเช่น เครื่องยนต์เป็นส่วนหนึ่งของรถยนต์ แต่รถยนต์ไม่เป็นส่วนหนึ่งของเครื่องยนต์ เป็นต้น

Coad and Yourdon ได้ให้แนวทางในการกำหนดความสัมพันธ์แบบ Aggregation ไว้ดังนี้

- Assembly เป็นลักษณะของการรวมกันโดยการสร้างมาจากส่วนต่างๆ เช่น ซุปหัวหอมจะเป็นการรวมกันของ หัวหอม แป้ง เนยก้อน และอื่นๆ เป็นต้น
- Container เป็นลักษณะของการรวมกัน แต่ไม่ใช่ลักษณะที่ถูกสร้างขึ้นด้วยจากส่วนต่างๆ เช่น บ้านจะประกอบไปด้วย เฟอร์นิเจอร์ และเครื่องมือต่างๆ เป็นต้น
- Collection-Member เป็นลักษณะการรวมแบบแนวความคิด (Conceptual) เช่น ทีมฟุตบอลจะประกอบไปด้วยผู้เล่น เป็นต้น

2.2.5.2 การกำหนด Attribute

Attribute เป็นสิ่งที่ Object ต้องจำ เช่น สี, ราคา, โรงงานผู้ผลิต การกำหนด Attribute ของ Class ในระบบเป็นการเริ่มพร้อมกับการเข้าใจในหน้าที่ของระบบ ซึ่งจะเห็นได้ว่าหน้าที่ของระบบและลักษณะ Application ที่ต้องการนั้น สามารถจะพัฒนาได้ด้วย Use Case เช่นการกำหนดข้อมูลที่จำเป็นต่อระบบ ซึ่งอาจใช้คำถาม "ข้อมูลเกี่ยวกับอะไรที่เราต้องการเก็บและสืบค้น" ช่วยในการค้นหา Attribute ของ Class ได้

Attribute สามารถจะได้มาจากการทดสอบเหตุการณ์ที่เกิดขึ้นโดยอาศัยการวิเคราะห์ Use Case, Sequence / Collaboration, Activity และ State Diagrams ซึ่งจะทำให้เกิดความเข้าใจในหน้าที่ของ Class และวิธีที่ Class เหล่านั้นมีการตอบโต้กันในการปฏิบัติงาน การที่จะได้ Attribute มานั้นอาจใช้แนวทางต่างๆ ต่อไปนี้ ช่วยในการกำหนด Attribute ได้

- โดยปกติแล้ว Attribute ส่วนใหญ่จะตรงกับคำนามที่ตามด้วยคำบุพบท เช่น ราคาของซุป ซึ่งในบางกรณีก็อาจจะตรงกับคำขยายคำนาม (Adjective) หรือคำขยายกริยา (Adverb) ก็ได้
- พยายามให้ Class นั้นมีความเรียบง่าย คือกำหนด Attribute แค่สามารถกำหนดสถานะของ Object ได้ก็พอ
- ไม่กำหนด Attribute ที่เกิดมาจาก Attribute อื่นๆ (Derived Attribute) เช่น อายุ เพราะค่าของมันจะได้มาจากการคำนวณกับ Attribute วันเกิด ซึ่ง Derived Attribute นี้ควรจะถูกกำหนดเป็น Method
- อย่าพยายามกำหนด Attribute เกินความจำเป็น เพราะสามารถกำหนดเพิ่มได้ในการทำซ้ำต่อๆ ไปได้

2.2.5.3 การกำหนด Method

Object ไม่เพียงแต่จะอธิบายข้อมูลเพียงอย่างเดียวแต่มันยังให้บริการบางอย่างได้ด้วย ในสภาวะแวดล้อมเชิงวัตถุ (Object Oriented Environment) ข้อมูลทุกตัวหรือ Object จะถูกล้อมรอบ

ด้วยหน้าที่จำนวนมาก ซึ่งเรียกว่า Method ซึ่ง Method เหล่านี้เป็นตัวทำทุกๆ สิ่งตั้งแต่การพิมพ์จนถึงการเริ่มต้นตัวแปลด้วย

ทุกๆ Class มีหน้าที่สำหรับเก็บข้อมูลของระบบ ซึ่งเป็นเหมือนกับการบอกว่าทุกๆ Class มีหน้าที่ ที่ต้องปฏิบัติกับข้อมูลเหล่านั้นด้วยเช่นกัน ซึ่งก็คือเป็นเหมือนกับการกำหนดให้มี Method ที่จัดการกับข้อมูลที่มันมีด้วย และหน้าที่จัดการกับข้อมูลหรือค่าใน Attribute นั้นจะรวมถึง การสืบค้น (Query), การแก้ไขเปลี่ยนแปลง (Update), การอ่าน (Reading) และการเขียน (Write) และในบางกรณีอาจมีความต้องการชุดของการ Methods ที่ใช้ในการจัดการและเปลี่ยนแปลงค่าด้วย

2.2.6 การทำซ้ำ (Iterate) และการแก้ไขให้ดีขึ้น (Refine)

เป็นการทำซ้ำเพื่อเป็นการเพิ่มสิ่งที่ถูกต้องที่สุด ในทุกๆ ขั้นของการพัฒนานั้นสามารถจะทำซ้ำได้โดยการทำซ้ำนั้นจะเป็นการแก้ไขให้ถูกต้องและสิ่งที่เหมาะสมที่สุด และในการพัฒนาจะค่อยๆ เพิ่มความถูกต้องและความสมบูรณ์ขึ้นมาเรื่อยๆ

2.3 การออกแบบเชิงวัตถุ

การออกแบบนั้นจะใช้ Object ที่ค้นหาในช่วงการวิเคราะห์เป็นโครงในการออกแบบ การกำหนด Attribute, Method และความสัมพันธ์ของ Class ในขั้นการวิเคราะห์นั้น ต้องถูกออกแบบให้เหมาะสมกับประเภทของข้อมูลในภาษาที่จะใช้ในการสร้าง ซึ่งจะเป็นการเลื่อนจาก การกำหนดสิ่งที่ต้องการจะทำ ไปเป็น วิธีการทำงานแทน

ในระหว่างการออกแบบนั้นอาจจะมีความสัมพันธ์ในเชิงเทคนิค (เช่นอาจเกี่ยวข้องกับ User Interface หรือวิธีการเข้าถึงข้อมูล) มากกว่าความเป็นจริง (เช่น คนหรือพนักงาน) เพราะเป้าหมายของการออกแบบคือการออกแบบ Class ที่จำเป็นในการสร้างระบบ

ปกติแล้วการออกแบบที่ดีคือการทำให้ง่ายและชัดเจนในการสร้างและการบำรุงรักษาของโครงการ ซึ่งเวลาที่ใช้ในออกแบบนั้นมีผลกระทบกระเทือนกับความสำเร็จทั้งหมดของโครงการการพัฒนาซอฟต์แวร์อย่างมาก

ในกระบวนการการออกแบบเชิงวัตถุจะอาศัย Axiom Approach ซึ่งเป็นแนวทางของการออกแบบอย่างเป็นระบบระเบียบ และช่วยในการสร้างพื้นฐานทางวิทยาศาสตร์เกี่ยวกับกระบวนการการออกแบบเชิงวัตถุ และยังให้พื้นฐานที่สำคัญในการสร้างระบบอีกด้วย หากปราศจากหลักเกณฑ์ทางวิทยาศาสตร์แล้วการออกแบบอาจจะไม่เป็นขั้นเป็นตอนและจะยังคงเป็นเรื่องที่ยากที่จะปฏิบัติได้

กระบวนการการออกแบบเชิงวัตถุประกอบไปได้ขั้นตอนต่างๆ ดังนี้

1 ใช้ Axiom ช่วยในการออกแบบ Class, Attribute, Method, Association, Structure และ Protocol

- แก้ไขปรับปรุงให้ Class Diagram สมบูรณ์ขึ้นโดยการเพิ่มรายละเอียดลงไป
- ทำซ้ำและแก้ไขปรับปรุงให้ดีขึ้นอีกครั้ง

2 การออกแบบ Access Layer ซึ่งเป็นชั้นที่มีหน้าที่รับผิดชอบในการเข้าถึงข้อมูลของระบบ

- การสร้าง Class แบบ Mirror (Mirror Class)
- กำหนดสร้างความสัมพันธ์ใน Layer
- ทำให้ Class และความสัมพันธ์เรียบง่ายและชัดเจน อาจเป็นการหา Class ที่ซ้ำกันหรือเป็น Class ที่อยู่ในประเภท Method Class
- ทำซ้ำและแก้ไขปรับปรุงให้ดีขึ้นอีกครั้ง

3. การออกแบบ View Layer เป็นการออกแบบ Class ที่ทำหน้าที่เป็นตัวโต้ตอบระหว่างผู้ใช้งานกับระบบ (User Interface)

- การออกแบบ User Interface ในขั้น Macro Level
- การออกแบบ User Interface ในขั้น Micro Level
- ทดสอบการใช้งานและความพอใจของผู้ใช้งาน
- ทำซ้ำและแก้ไขปรับปรุงให้ดีขึ้นอีกครั้ง

4. การทำซ้ำและแก้ไขปรับปรุงให้ดีขึ้นการออกแบบทั้งหมด

ทุกขั้นตอนของการพัฒนาระบบซอฟต์แวร์สามารถจะปฏิบัติแบบค่อยๆ เพิ่มความถูกต้องเข้าไปได้ ดังนั้นการตัดสินใจในครั้งแรกทุกทั้งไม่จำเป็นต้องถูกต้องเสมอก็ได้ การออกแบบเชิงวัตถุที่ดีนั้นควรจะต้องมีการทำซ้ำ (Iterative) หลายๆ ครั้ง นานเท่าที่คิดว่า Class ของ Object นั้นได้ผ่านการเรียนรู้เรียบร้อย และเต็มใจกับการออกแบบ หรือจนกระทั่งรู้สึกพอใจกับกระบวนการที่ผ่านมา

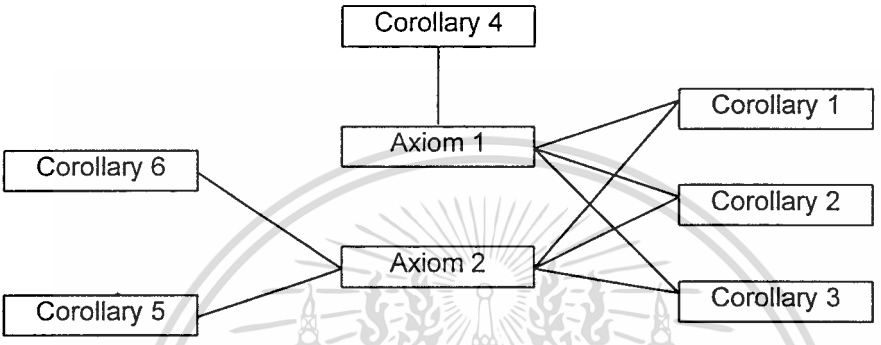
2.3.1 Object-Oriented Design Axioms

Axiom คือความจริงที่ไม่ต้องพิสูจน์ซึ่งไม่มีข้อยกเว้น ในการออกแบบครั้งนี้นั้นจะใช้ Axiom 2 ข้อ ซึ่งข้อ 1 เป็นข้อที่จัดการกับความสัมพันธ์ระหว่างส่วนประกอบต่างๆ ของระบบ (เช่น Class, Software Component และ Requirement) และข้อที่ 2 เป็นข้อที่จัดการกับความซับซ้อนของการออกแบบ ซึ่งโดยสรุปคือ

1. The independence axiom เป็นการดูแลรักษาความเป็นอิสระของส่วนประกอบต่างๆ

2. The information axiom เป็นการทำให้ข้อมูลของการออกแบบน้อยที่สุด

จาก Axiom ในการออกแบบทั้ง 2 อาจนำไปสู่ที่มาของ Corollary ดังรูปที่ 2.17 ซึ่ง Corollary คือส่วนขยายที่ต่อมาจาก Axiom หรือส่วนขยายที่มาจากส่วนอื่นๆ ที่ได้ผ่านการพิสูจน์มาแล้ว Corollary อาจมีส่วนช่วยในการตัดสินใจในการออกแบบด้วย เพราะ Corollary เหล่านั้นได้นำมาใช้ในสถานการณ์จริงซึ่งใช้ง่ายกว่า Axiom หรืออาจเรียก Corollary ว่าเป็นกฎในการออกแบบ แต่ทั้งหมดนั้นมีที่มา มาจาก Axiom ทั้ง 2 นั่นเอง



รูปที่ 2.17 แสดงความสัมพันธ์ระหว่าง Corollary กับ Axiom

ใช้ Corollary เหล่านี้เป็นหลักเกณฑ์ในการใช้ออกแบบ Class

Corollary 1 ออกแบบให้มีข้อมูลน้อยที่สุดและลักษณะเป็น Uncoupled ซึ่งมีจุดประสงค์หลักคือการทำให้ภายใน Object และ Software Component มีลักษณะเกี่ยวข้องกันมากที่สุด

Corollary 2 ให้มีวัตถุประสงค์เดียว ซึ่งแต่ละ Class ต้องมีวัตถุประสงค์เดียวที่ชัดเจน ซึ่งอาจจะอธิบายอย่างง่ายๆ ได้โดยใช้ไม่กี่ประโยค

Corollary 3 ให้จำนวน Class ที่มีความธรรมดา (Simple) มากๆ เพื่อที่จะนำกลับมาใช้ใหม่ได้ (Reusability)

Corollary 4 ให้มีการแทนกันอย่างลงตัว (Strong Mapping) เป็นการบอกถึงความสัมพันธ์ระหว่าง Object ที่ได้มาจากการวิเคราะห์ กับ Object ที่ใช้ในการออกแบบ ซึ่งในการวิเคราะห์และออกแบบเชิงวัตถุนั้นอยู่บนพื้นฐานของการสร้างแบบจำลองเดียวกัน ซึ่งในขั้นการออกแบบจะเป็นเพียงการเพิ่มรายละเอียดและยังคงรักษาความจำเป็นที่เหมือนกันเอาไว้

Corollary 5 ให้เป็นมาตรฐาน เป็นการสนับสนุนให้มีมาตรฐานในการออกแบบ ซึ่งใช้ในการติดต่อสื่อสารระหว่าง Component และการนำ Class หรือ Component กลับมาใช้ใหม่ (Reusing)

Corollary 6 ให้การออกแบบร่วมกับการถ่ายทอดไปด้วยกัน ซึ่งก็คือ Common Method ต้องย้ายเป็นอยู่ใน Super-Class เพราะอาจมีผลต่อคำสั่งของโปรแกรมได้

แต่ละขั้นตอนของกระบวนการการออกแบบเชิงวัตถุ จะมีรายละเอียดดังต่อไปนี้

2.3.2 การออกแบบ Class

เป็นการนำ Class ที่ได้มาจากขั้นการวิเคราะห์แก้ไขปรับปรุงให้ดีขึ้น โดยการกำหนดความสามารถในการมองเห็น (Visibility) ทั้ง Attribute และ Method กำหนดประเภทข้อมูลและขนาดของ Attribute ที่ตรงกับภาษาที่ใช้ในการพัฒนาระบบ รวมถึงอาจมีการกำหนดค่าเริ่มต้นให้กับ Attribute บางกรณีอาจต้องมีการเพิ่ม บาง Attribute ขึ้นมาสำหรับรองรับการสร้างระบบด้วย

ในส่วนของ Method จะมีการกำหนดรูปแบบการเรียกใช้ (Protocol) ซึ่งจะรวมถึงการกำหนดความสามารถในการมองเห็น กำหนด Algorithm ในแต่ละ Method โดยอาจใช้ Activity Diagram หรือ State Diagram ช่วยในการแสดง Algorithm ดังกล่าว และจะใช้ Corollary 1 เข้ามาช่วยในการออกแบบ Method ด้วยเพื่อต้องการให้ Method มีความเป็นหนึ่งเดียวมากที่สุด (คือทำงานเพียงหน้าที่เดียว) และลดความซับซ้อนของ Message และจำนวนของการรับ-ส่ง Message ด้วย.

บางกรณีอาจมีการใช้ Package เพื่อเป็นการแบ่งกลุ่มของ Class ที่ทำหน้าที่เกี่ยวข้องกัน โดยปกติแล้ว Package สามารถใช้เป็นที่รวมกันของ Model Element แม้กระทั้ง Package อื่นๆ อีก เพื่อเป็นการอธิบายระบบในระดับที่สูงขึ้น

2.3.3 การออกแบบ Access Layer

เป็นการออกแบบ Class ที่ทำหน้าที่เป็นผู้เข้าถึงข้อมูลของระบบ ซึ่งอาจใช้ขั้นตอนดังนี้

- นำทุก Class มาสร้างเป็น Class ใหม่ (Mirror Class) ที่ทำหน้าที่เป็นผู้เข้าถึงข้อมูลของระบบ เช่น มี Class1 และอาจสร้าง Class ใหม่เป็น Class1DB เป็นต้น
- กำหนดความสัมพันธ์ของ Class ที่สร้างขึ้นมาใหม่ ซึ่งอาจใช้ความสัมพันธ์ในลักษณะเดิมก็ได้
- ทำให้ Class และความสัมพันธ์เรียบง่ายและชัดเจน โดยการตัด Class ที่ซ้ำกันทิ้ง หรือ Class ที่ไม่จำเป็น หรือแม้แต่โครงสร้างที่ไม่จำเป็น เช่น Class ที่ให้บริการเหมือนกัน เป็นต้น
- ทำซ้ำและแก้ไขปรับปรุงให้ดีขึ้น

2.3.4 การออกแบบ View Layer

เป็นการออกแบบ Class ที่ทำหน้าที่เป็น User Interface ระหว่างผู้ใช้งานกับตัวระบบ ซึ่งอาจใช้แนวทางตามขั้นตอนดังนี้

การออกแบบ User Interface ในขั้น Macro Level เป็นการกำหนด Class ที่มีหน้าที่โต้ตอบระหว่างผู้ใช้งานกับระบบ สามารถหาได้จากการวิเคราะห์ Use Case ที่ถูกพัฒนาขึ้นในขั้นการวิเคราะห์ และ Sequence Diagram หรือ Collaboration Diagram สามารถช่วยในการกำหนด Class ด้วย หลังจากได้ Class แล้วจึงมากำหนดความสัมพันธ์ระหว่าง Class ที่ได้มาใหม่ และอาจมีการทำซ้ำแก้ไขปรับปรุงก็ได้

การออกแบบ User Interface ในขั้น Micro Level อาจจะใช้หลักการต่อไปนี้ เป็นแนวทางในการช่วยออกแบบ User Interface ที่ได้มา

- ทำให้ชัดเจนเรียบง่าย (ประยุกต์ Corollary 2) ทำให้ User Interface เรียบง่ายและชัดเจน จนผู้ที่ไม่รู้เรื่องเกี่ยวกับเครื่องมือและกลไกของระบบสามารถทำงานได้
- ทำให้เป็นธรรมชาติ (ประยุกต์ Corollary 4) ผู้ใช้งานปกติจะสามารถคาดเดาเหตุการณ์ที่จะเกิดขึ้นกับ User Interface ได้ โดยเสมือนไม่ได้ทำงานกับระบบอยู่
- ทำให้ผู้ใช้งานเป็นผู้ควบคุม Software (ประยุกต์ Corollary 1) อาจเป็นการเตรียมงานที่เป็นอัตโนมัติ เพื่ออำนวยความสะดวกให้กับผู้ใช้งาน หรืออาจเป็นการเพิ่มคำแนะนำหากผู้ใช้มีการทำผิดพลาด เป็นต้น ยังรวมถึงทำให้ User Interface มีลักษณะเหมือนกันทั้งระบบ

หลังจากสร้าง User Interface จึงมาถึงขั้นการทดสอบการใช้งานและทดสอบความพอใจของผู้ใช้งานต่อไป ซึ่งจะเป็นการนำ User Interface ที่ได้มาให้ผู้ใช้งานของระบบเป็นผู้ประเมิน และสุดท้ายทำซ้ำแก้ไขปรับปรุงให้ดีขึ้น

2.3.5 การทำซ้ำและแก้ไขปรับปรุงให้ดีขึ้นการออกแบบทั้งหมด

การออกแบบเชิงวัตถุเป็นกระบวนการที่ทำซ้ำ (Iterative Process) ซึ่งหลังจากผ่านแต่ละขั้นของการออกแบบอาจจะเป็นการค้นพบสิ่งที่เกี่ยวกับการการสร้างอย่างมากก็ได้

บทที่ 3

การวิเคราะห์และออกแบบ

3.1 คำอธิบายระบบ

หน่วยงานรับชำระของการเคหะแห่งชาติ เป็นหน่วยงานที่มีหน้าที่รับผิดชอบในเรื่องของการรับชำระของลูกหนี้ในการเคหะ ทั้งรับชำระด้วยเงินสดและรับชำระผ่านธนาคาร รวมถึงการออกรายงานสรุปผลรวมการรับชำระของการเคหะ

3.1.1 ความต้องการของระบบ

ลูกหนี้สามารถชำระหนี้ได้โดยการชำระด้วยเงินสด หรือชำระผ่านธนาคาร ซึ่งการชำระผ่านธนาคารนั้นจะเป็นการให้ทางธนาคารเป็นผู้หักเงินจากบัญชีของลูกหนี้

รายละเอียดการรับชำระของลูกหนี้แต่ละรายจะต้องถูกบันทึกเก็บไว้ได้ทั้งชำระด้วยเงินสดและชำระผ่านธนาคาร ลูกหนี้ที่ชำระหนี้ในแต่ละงวดจะได้รับใบเสร็จรับเงินเพื่อเป็นการยืนยันการชำระ

ลูกหนี้จะชำระจำนวนเงินเท่าที่สัญญาระบุไว้ต่องวด (ต้องไม่น้อยกว่า) และลูกหนี้สามารถชำระหนี้ได้มากกว่าจำนวนเงินที่กำหนดไว้ต่องวด ซึ่งอาจเป็นการชำระหนี้ที่เหลือทั้งหมดหรือเป็นการชำระเพียงบางส่วน ดังนั้นต้องมีการคำนวณยอดหนี้คงเหลือได้ และหากลูกหนี้มีการชำระมากกว่าจำนวนเงินที่กำหนดไว้ต่องวด (เป็นกรณีที่ชำระเพียงบางส่วน) สัญญาเดิมของลูกหนี้ที่ทำไว้และต้องมีการแก้ไขข้อมูลภายในสัญญาใหม่ เพื่อเป็นการกำหนดจำนวนหนี้ทั้งหมดและจำนวนเงินที่จ่ายต่องวดใหม่ ดังนั้นจึงต้องมีการแก้ไขสถานะของสัญญาเพื่อเป็นการบอกหน่วยงานที่รับผิดชอบทราบและดำเนินการต่อไป

ระบบสามารถออกรายงานประมาณการรายรับและรายงานการรับจริงได้ สรุปการรับชำระประจำวัน สรุปการรับชำระประจำเดือน สรุปการหักบัญชีได้-ไม่ได้ของแต่ละธนาคาร สรุปลูกหนี้คงค้าง และรายงานลูกหนี้คงเหลือ

ข้อกำหนด

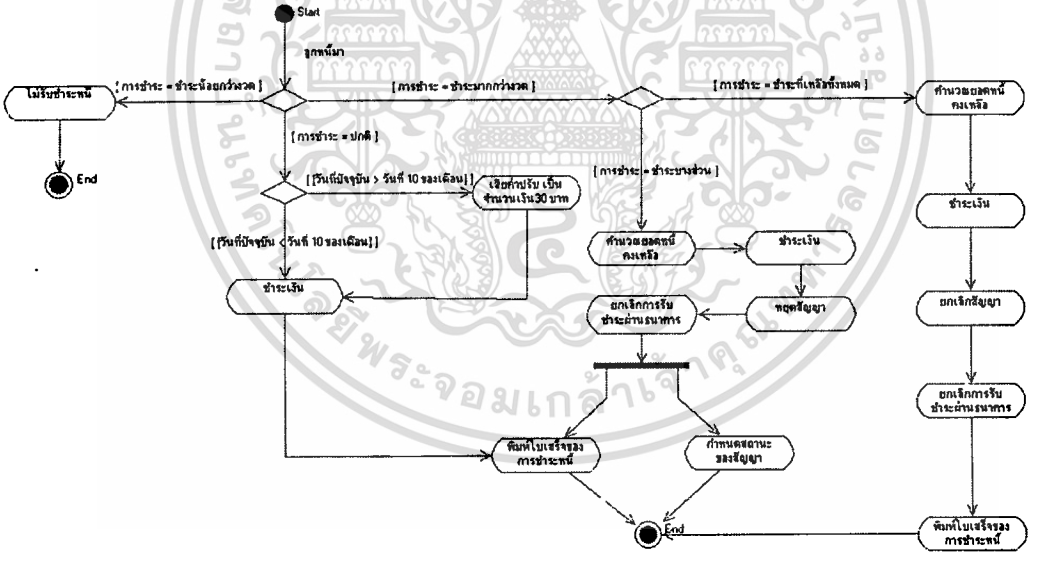
- ลูกหนี้จะไม่สามารถเปลี่ยนรูปแบบการชำระหนี้ได้ตามใจ จะต้องมีการแจ้งให้ทางการเคหะรับทราบก่อนทุกครั้งที่ต้องการเปลี่ยนรูปแบบการชำระ
- ลูกหนี้ไม่สามารถชำระหนี้น้อยกว่าจำนวนเงินที่กำหนดไว้ต่องวด

- กรณีที่ลูกหนี้มีการชำระหนี้ผ่านทางธนาคาร และลูกหนี้ต้องการจะชำระหนี้มากกว่าจำนวนเงินที่กำหนดไว้ เมื่อชำระแล้วจะถือว่าความประสงค์ที่แจ้งไว้เพื่อการขอชำระผ่านธนาคารที่ผ่านมานั้นจะถูกยกเลิกโดยทันที
- หากลูกหนี้ชำระเงินหลังจากวันที่ 10 ของทุกเดือน ลูกหนี้จะต้องเสียค่าปรับเป็นจำนวนเงิน 30 บาท

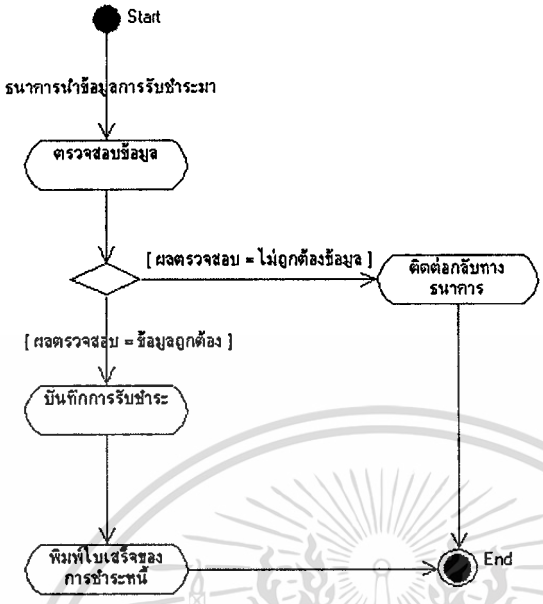
3.2 แบบจำลองของกระบวนการทางธุรกิจ

พัฒนาแบบจำลองของกระบวนการการรับชำระ (ในเชิงธุรกิจ) จะใช้ Activity Diagram ช่วยในการสร้างแบบจำลอง ซึ่งจะได้ไดอะแกรมดังรูปที่ 3.1 และ รูปที่ 3.2 ซึ่งเป็นรูปที่แสดง Activity Diagram ของระบบรับชำระ

ไดอะแกรมที่ได้เป็นการใช้ CASE Tool ชื่อ Rational Rose เวอร์ชัน 2000 เข้ามาช่วยในการสร้าง และจะใช้ CASE Tool ตัวนี้ช่วยในการสร้างแบบจำลองต่างๆ ของทั้งระบบ



รูปที่ 3.1 แสดง Activity Diagram ของระบบรับชำระ (รับชำระเงินสด)



รูปที่ 3.2 แสดง Activity Diagram ของระบบรับชำระ (รับชำระผ่านธนาคาร)

3.3 แบบจำลอง Use-Case

การที่จะได้ Actor และ Use Case มานั้นต้องผ่านการศึกษาความต้องการของระบบ และผ่านการทำซ้ำหลายต่อหลายรอบ ซึ่งก็เป็นการศึกษาความต้องการของระบบ พร้อมกับกำหนด Actor และ Use Case ไปด้วยกัน และในการศึกษารอบต่อไปอาจจะต้องมีการปรับปรุงเปลี่ยนแปลง Actor และ Use Case ที่ค้นพบด้วย จนในที่สุดได้ Actor และ Use Case ดังนี้

3.3.1 Actor

Bank (ธนาคาร)	ส่งข้อมูลที่ลูกหนี้ชำระผ่านธนาคาร
Debtor (ลูกหนี้)	ชำระหนี้ด้วยเงินสดและทำการขอชำระผ่านธนาคาร
Official (พนักงานรับชำระ)	ทำงานกับระบบรับชำระ

3.3.2 Use Case

AllTermCashPayment (รับชำระเงินสดที่เหลือทั้งหมด)

ผู้เริ่มต้น : ลูกหนี้

หน้าที่ : บันทึกการรับชำระด้วยเงินสดที่เหลือทั้งหมดของลูกหนี้

BankPayment (รับชำระผ่านธนาคาร)

ผู้เริ่มต้น : ธนาคาร

หน้าที่ : บันทึกการรับชำระผ่านธนาคารของลูกหนี้

CalculateRemainDebt (คำนวณยอดหนี้คงเหลือ)

ผู้เริ่มต้น : พนักงานรับชำระ

หน้าที่ : คำนวณยอดหนี้คงเหลือของลูกหนี้

CashPayment (รับชำระเงินสด)

ผู้เริ่มต้น : ลูกหนี้

หน้าที่ : บันทึกการรับชำระด้วยเงินสดของลูกหนี้

MoreTermCashPayment (รับชำระเงินสดมากกว่าจำนวนต่องวด)

ผู้เริ่มต้น : ลูกหนี้

หน้าที่ : บันทึกการรับชำระด้วยเงินสดที่มากกว่าจำนวนเงินต่องวดของลูกหนี้ (ยังคงเหลือหนี้อยู่)

Receipt (พิมพ์ใบเสร็จ)

ผู้เริ่มต้น : พนักงานรับชำระ

หน้าที่ : พิมพ์ใบเสร็จสำหรับผู้ที่ชำระหนี้

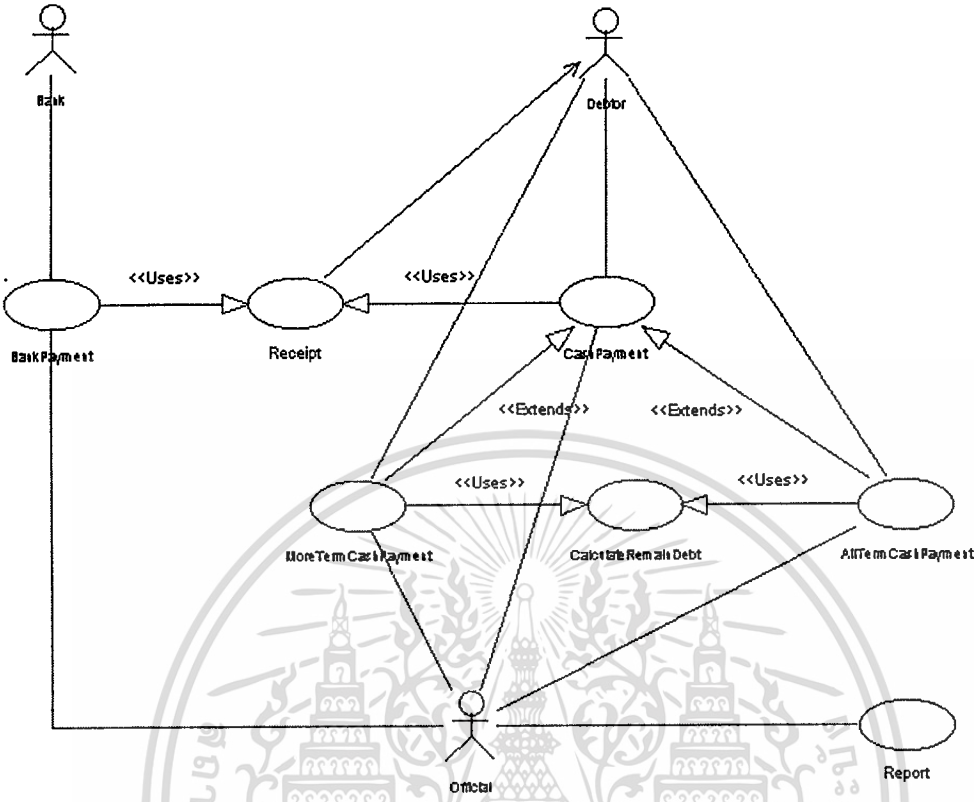
Report (พิมพ์รายงาน)

ผู้เริ่มต้น : พนักงานรับชำระ

หน้าที่ : พิมพ์รายงานต่างๆ ของการรับชำระ

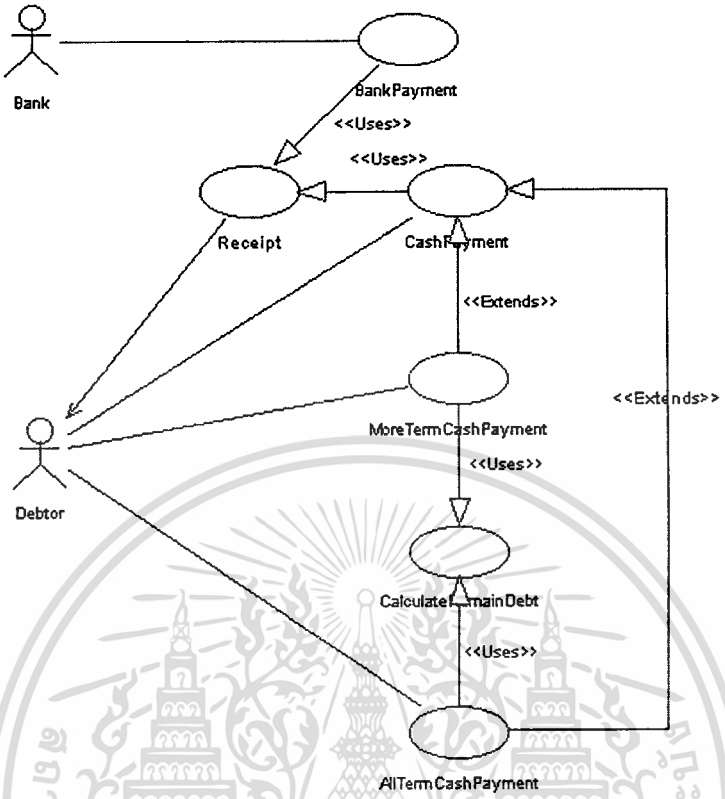
3.3.3 Use case Diagram

จาก Actor และ Use case ทั้งหมดที่ได้สามารถนำมาสร้างเป็นไดอะแกรมซึ่งแสดงไว้ดังรูปที่ 3.3



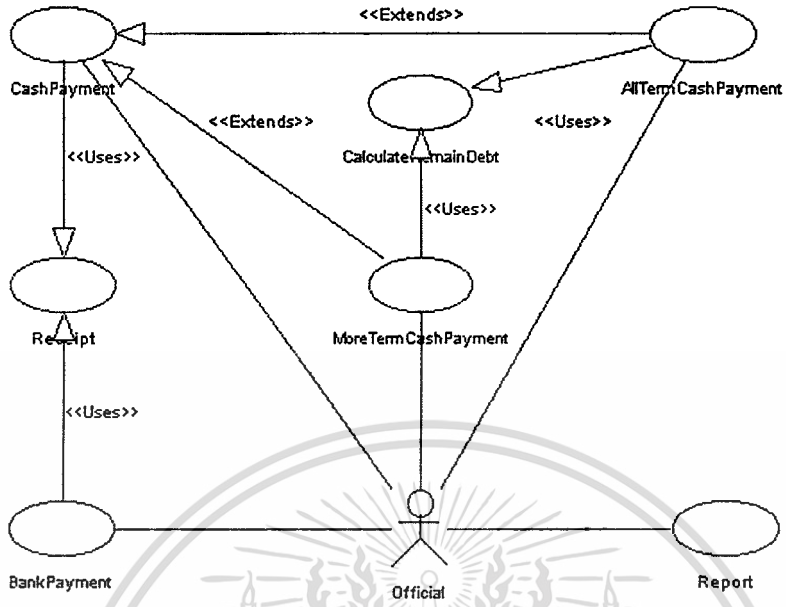
รูปที่ 3.3 Use Case Diagram ของระบบรับชำระ

จากรูปที่ 3.3 นั้นเป็น Use Case Diagram ของระบบรับชำระทั้งหมด ซึ่งอาจเป็นการยากในการทำความเข้าใจจึงสร้าง Use Case Diagram ที่เป็นมุมมองของในกรณีของบุคคลภายนอกว่ามีส่วนเกี่ยวข้องอย่างไรกับระบบงานรับชำระจะแสดงไว้ดังรูปที่ 3.4



รูปที่ 3.4 แสดง Use Case Diagram ของระบบรับชำระในมุมมองของบุคคลภายนอกระบบ

และมีการสร้าง Use Case Diagram เพื่อให้ง่ายในการทำความเข้าใจระบบในมุมมองของบุคคลภายใน หรือเป็นผู้ที่ทำงานกับระบบซึ่งแสดงไว้ดังรูปที่ 3.5

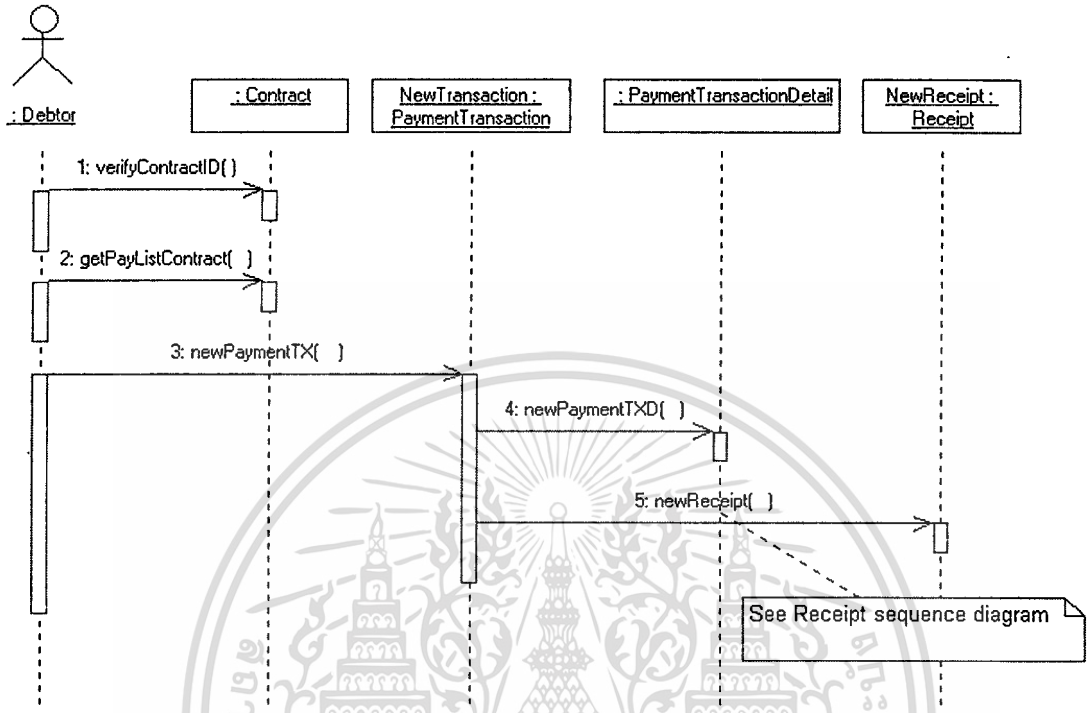


รูปที่ 3.5 แสดง Use Case Diagram ของระบบรับชำระในมุมมองของบุคคลภายในระบบ

3.4 Interaction Diagram

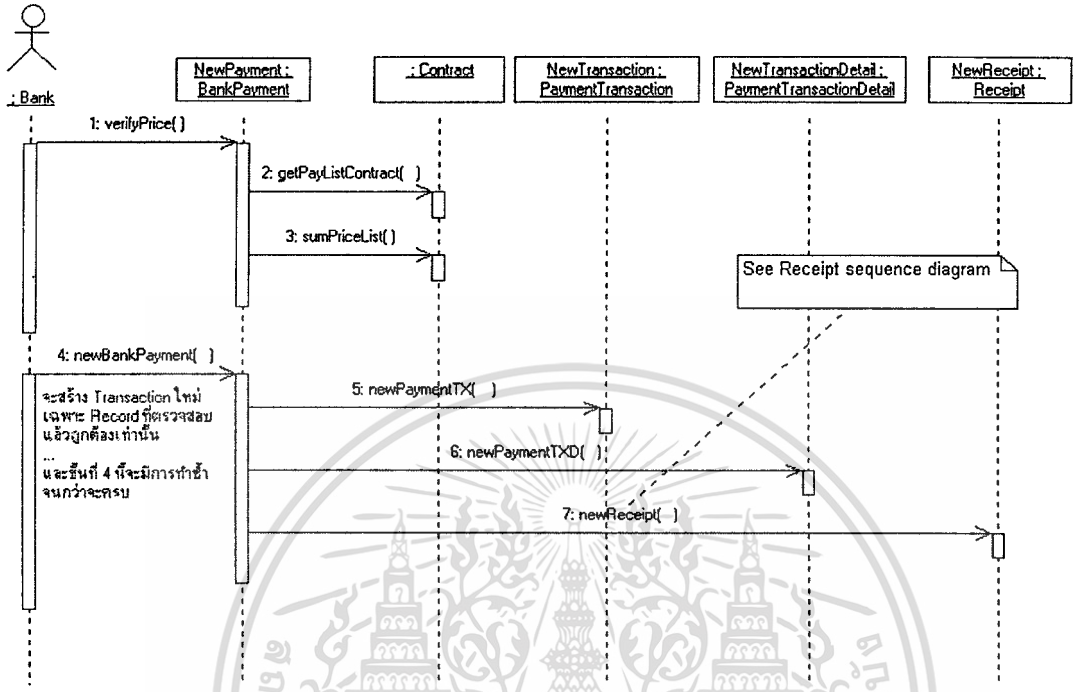
ซึ่งจะใช้ Sequence Diagram เป็นหลักในการแสดงเหตุการณ์ที่เกิดขึ้นเพื่อต้องการแสดงให้เห็นถึงลำดับเหตุการณ์ที่เกิดขึ้น ในแต่ละ Use Case โดยจะเรียงลำดับการแสดงเหตุการณ์ของแต่ละ Use Case ตามลำดับความสำคัญของระบบ

3.4.1 Sequence Diagram ของ CashPayment Use Case



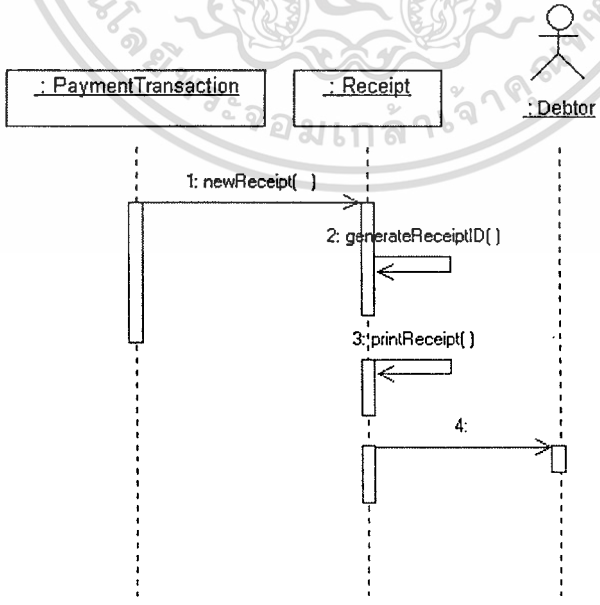
รูปที่ 3. 6 แสดง Sequence Diagram ของ CashPayment Use Case

3.4.2 Sequence Diagram ของ BankPayment Use Case



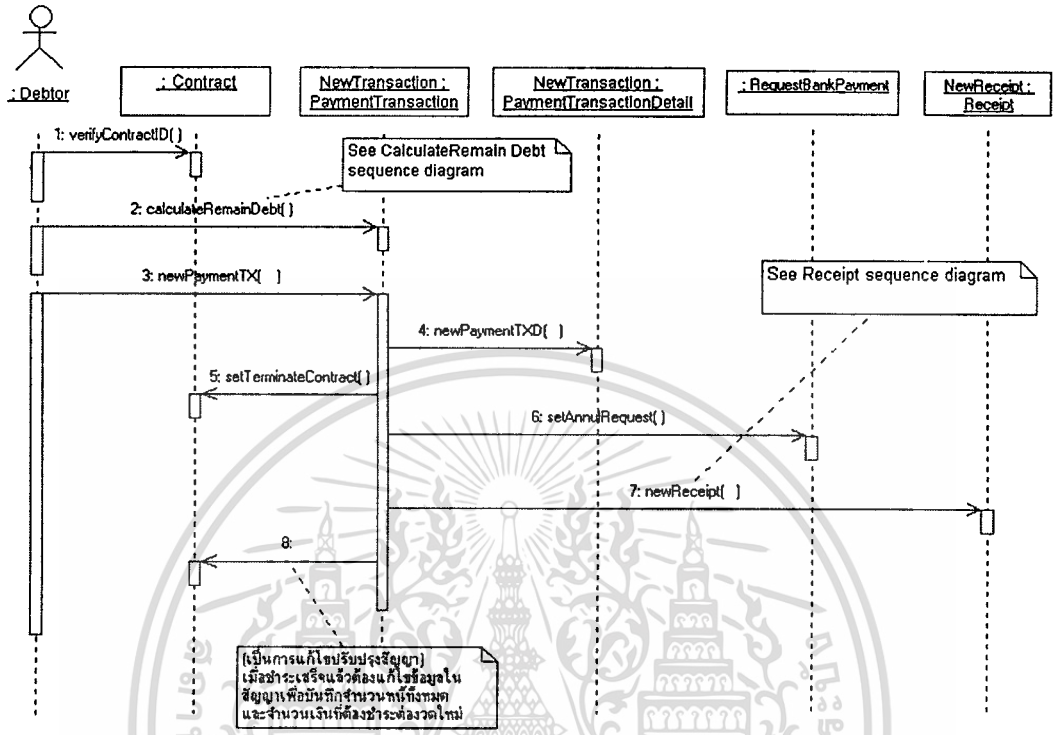
รูปที่ 3. 7 แสดง Sequence Diagram ของ BankPayment Use Case

3.4.3 Sequence Diagram ของ Receipt Use Case



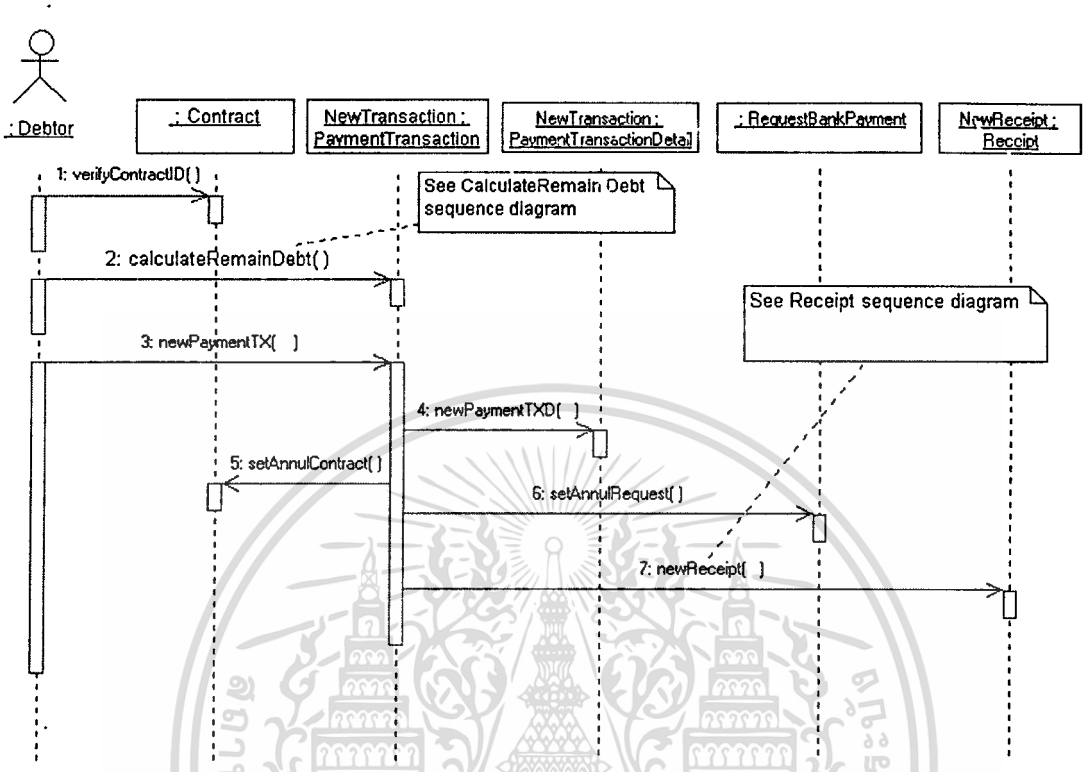
รูปที่ 3. 8 แสดง Sequence Diagram ของ Receipt Use Case

3.4.4 Sequence Diagram ของ MoreTermCashPayment Use Case



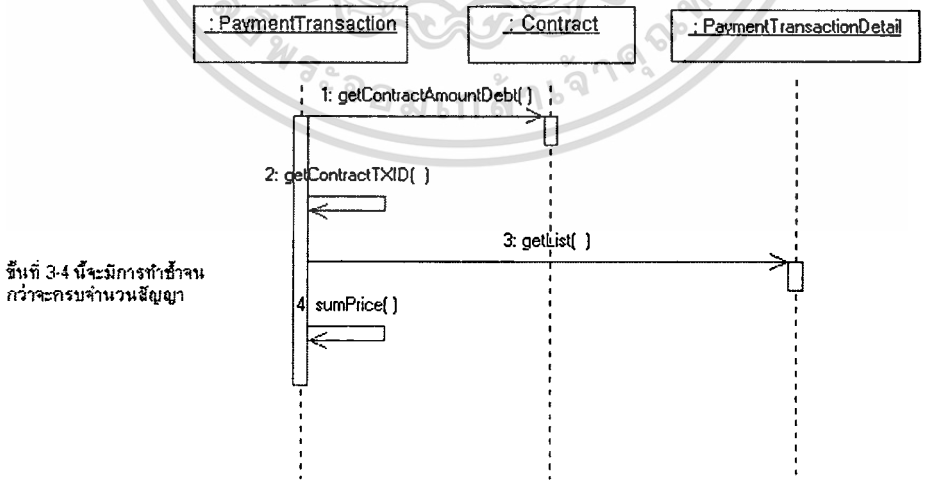
รูปที่ 3. 9 แสดง Sequence Diagram ของ MoreTermCashPayment Use Case

3.4.5 Sequence Diagram ของ AllTermCashPayment Use Case



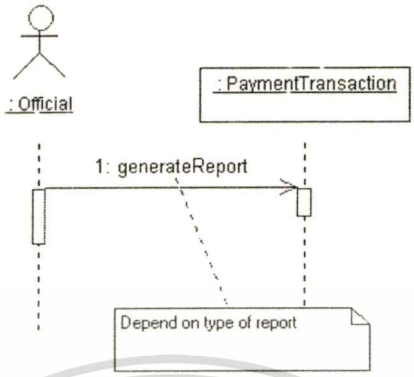
รูปที่ 3. 10 แสดง Sequence Diagram ของ AllTermCashPayment Use Case

3.4.6 Sequence Diagram ของ CaculationRemainDebt



รูปที่ 3. 11 แสดง Sequence Diagram ของ CaculationRemainDebt

3.4.7 Sequence Diagram ของ Report Use Case



รูปที่ 3. 12 แสดง Sequence Diagram ของ Report Use Case

3.5 Class Diagram

3.5.1 กำหนด Class

จากการวิเคราะห์และแบบจำลอง Use case และ Interaction Diagram ที่ได้มาในตอนต้น สามารถจำแนกคลาสได้ดังตารางที่ 1

ตารางที่ 3.1 แสดงคลาสของระบบรับชำระในขั้นการวิเคราะห์

ชื่อ	วัตถุประสงค์
AccountInformation	เป็นรายการบัญชีที่ใช้ระบบรับชำระ ซึ่งเป็นข้อมูลที่ต้องการในการบันทึกการรับชำระแต่ละรายการ
BankInformation	เป็นธนาคารที่ลูกหนี้ชำระหนี้ผ่านให้กับการเคหะ
BankPayment	เป็นการเก็บรายละเอียดของการรับชำระผ่านธนาคารของลูกหนี้
BranchInformation	เป็นสาขาใดของหน่วยงานห้างร้าน ซึ่งในกรณีนี้จะเป็นสาขาของธนาคารที่ลูกหนี้ชำระหนี้ผ่านให้กับการเคหะ
Contract	เป็นสัญญาของลูกหนี้ ที่ได้ทำไว้กับทางการเคหะ ซึ่งเป็นข้อมูลที่ต้องการในการรับชำระ
PaymentTransaction	เป็น Transaction การรับชำระของระบบ ซึ่งเป็นการเก็บรายละเอียดการรับชำระในแต่ละครั้ง
PaymentTransactionDetail	เป็นรายละเอียดของแต่ละ Transaction ของการรับชำระ
Receipt	เป็นใบเสร็จสำหรับลูกหนี้ที่ชำระหนี้ในแต่ละครั้ง

จากตารางที่ 3.1 Class Contract เป็น Class ตรงกับ Class ที่อยู่ภายในระบบสัญญา และภายในระบบสัญญานี้ก็มีโครงสร้างของ Class ที่ประกอบไปด้วย Class จำนวนหนึ่ง ระบบรับชำระเป็นระบบที่ต้องการข้อมูลจากระบบสัญญา จึงต้องมีการติดต่อกับ Class ภายในระบบสัญญา ดังนั้นจะขอแสดง Class Contract เพียงบางส่วนซึ่งจะแทนด้วยจำนวนข้อมูลที่ต้องการ

3.5.2 ความสัมพันธ์ของ Class

3.5.2.1 Contract กับ PaymentTransaction

สัญญา (Contract) ของลูกหนี้ทุกสัญญาที่มีการทำไว้กับทางการเคหะ ลูกหนี้ที่ถือสัญญานั้นๆ จะต้องมีการจ่ายเงิน (PaymentTransaction) เกิดขึ้น และสัญญาหนึ่งๆ จะต้องมีการจ่ายเงินหนึ่งครั้งหรือมากกว่าหนึ่งครั้งขึ้นไป ดังนั้นจึงเกิดความสัมพันธ์ระหว่าง Contract กับ PaymentTransaction และเป็นความสัมพันธ์แบบ Association

3.5.2.2 PaymentTransaction กับ Receipt

การที่ลูกหนี้มาจ่ายเงินหนึ่งครั้งนั้นลูกหนี้จะได้รับใบเสร็จ (Receipt) หนึ่งใบ ภายในใบเสร็จหนึ่งใบนี้จะประกอบไปด้วยการจ่ายเงินครั้งนั้นๆ ทำให้เกิดความสัมพันธ์แบบ Aggregation ระหว่าง Receipt กับ PaymentTransaction ซึ่ง Receipt จะเป็นฝั่งทั้งหมด (Whole) และ Payment-Transaction จะเป็นฝั่งส่วนประกอบ (Part) ซึ่งเป็นความสัมพันธ์ชนิด Container เพราะปกติตามหลักความเป็นจริงใบเสร็จใบหนึ่งๆ อาจไม่ได้ประกอบไปด้วยการชำระเสมอไป

3.5.2.3 PaymentTransactionDetail กับ PaymentTransaction

ในหนึ่งครั้งของการชำระเงินของลูกหนี้จะมีรายละเอียดการชำระเงิน (Payment-TransactionDetail) เกิดขึ้นเพราะการชำระเงินนั้นอาจมีการชำระได้ตั้งแต่หนึ่งรายการขึ้นไปภายในหนึ่งครั้ง จึงทำให้เกิดความสัมพันธ์ระหว่าง PaymentTransactionDetail กับ PaymentTransaction และเป็นความสัมพันธ์แบบ Aggregation ชนิด Composition เพราะการชำระเงินหนึ่งรายการหรือหลายๆ รายการนั้นจะเกิดขึ้นภายในครั้งหนึ่งๆ เท่านั้นและหากไม่เกิดขึ้น (ในหนึ่งครั้งคือไม่เกิดเลย) ก็จะไม่มีรายละเอียดรายการที่ชำระเลย ดังนั้น PaymentTransactionDetail จึงเป็นด้านทั้งหมด (Whole) และ PaymentTransaction เป็นด้านส่วนประกอบ (Part)

3.5.2.4 PaymentTransactionDetail กับ AccountInformation

เนื่องจากระบบรับชำระนี้เป็นส่วนหนึ่งของระบบบัญชีลูกหนี้ ดังนั้นในการชำระแต่ละรายการที่ชำระก็จะมีการบ่งบอกถึงชนิดบัญชี (AccountInformation) ที่ลูกหนี้ชำระมาซึ่งรายการหนึ่งก็จะบอกถึงชนิดบัญชีหนึ่งเท่านั้น จึงทำให้เกิดความสัมพันธ์แบบ Association ระหว่าง PaymentTransactionDetail กับ AccountInformation

3.5.2.5 BankPayment กับ PaymentTransaction

ในกรณีที่ลูกหนี้ชำระผ่านธนาคารนั้น การชำระผ่านธนาคาร (BankPayment) หนึ่งครั้งก็จะประกอบไปด้วยการจ่ายเงินหนึ่งครั้ง แต่หากลูกหนี้ไม่จ่ายเงินก็จะถือว่าไม่ได้ทำการชำระผ่านธนาคารเช่นกัน ดังนั้นจึงเกิดความสัมพันธ์แบบ Aggregation ชนิด Composition ระหว่าง BankPayment กับ PaymentTransaction ซึ่ง BankPayment จะเป็นส่วนทั้งหมดและ Payment-Transaction จะส่วนประกอบ

3.5.2.6 BranchInformation กับ BankInformation

โดยทั่วไปแล้วคำว่าสาขา (BranchInformation) นั้นจะแทนด้วยพื้นที่ที่อยู่ตำแหน่งหนึ่งๆ แล้วแต่ละสาขาจะประกอบหน่วยงานหรือห้างร้าน (BankInformation) ดังนั้นจึงเกิดความสัมพันธ์แบบ Aggregation แบบ Container ระหว่าง BranchInformation กับ BankInformation ซึ่ง Branch-Information จะเป็นส่วนทั้งหมดและ BankInformation จะเป็นส่วนประกอบ

3.5.2.7 BankPayment กับ BranchInformation

สำหรับการชำระเงินผ่านธนาคารนั้น ลูกหนี้จะถูกหักบัญชีที่สาขาของธนาคารใดธนาคารหนึ่ง ซึ่งหมายความว่าเป็นการชำระเงินผ่านสาขาของธนาคาร ดังนั้นการหักบัญชี (การชำระเงินผ่านธนาคาร) จะประกอบไปด้วยสาขาของธนาคารใดธนาคารหนึ่ง และหากสาขานั้นๆ ไม่หักเงินจากบัญชีดังนั้นก็ไม่มีการหักอะไรเกิดขึ้นทั้งนั้น ซึ่งหมายความว่าจะไม่มีการชำระเงินเกิดขึ้น จึงทำให้เกิดความสัมพันธ์แบบ Aggregation แบบ Composition ระหว่าง BankPayment กับ Branch-Information ส่วน BankPayment จะเป็นส่วนทั้งหมดและ BranchInformation จะเป็นส่วนประกอบ

3.5.3 Attribute ของแต่ละ Class

โดยจะแสดงชื่อ Attribute พร้อมคำอธิบายของแต่ละ Attribute ดังนี้

AccountInformation	รายการบัญชี
accountID	รหัสบัญชี เช่น 0520
accountName	ชื่อบัญชี เช่นค่าเช่า, เช่าซื้อ (Install), ค่าบริการ (Service), ค่าประกัน (Insurance), ภาษี (Tax), ค่าที่ดินส่วนเกิน (Ex-land), สินเชื่อ (Loan), ค่าปรับ (Re-Schedule), อื่นๆ (Other)
accountDescription	คำอธิบายเพิ่มเติมแต่ละชื่อบัญชี

BankInformation	ข้อมูลธนาคาร
bankCode	รหัสที่ใช้ธนาคาร เช่น SCB
bankName	ชื่อธนาคาร
bankDetail	เป็นคำอธิบายสำหรับแต่ละธนาคาร
BankPayment	การรับชำระผ่านธนาคาร
txID	รหัสสำหรับแต่ละ Transaction
branch	ชื่อสาขา
bankCode	รหัสที่ใช้ธนาคาร เช่น SCB
BranchInformation	ข้อมูลสาขา
branch	ชื่อสาขา
AddressID	บ้านเลขที่
Moo	หมู่
Soi	ซอย
Road	ถนน
Precinct	แขวง/ตำบล
District	เขต/อำเภอ
Province	จังหวัด
Country	ประเทศ
PostalCode	รหัสไปรษณีย์
Contract	สัญญา
contractID	เลขที่สัญญา
contractDate	วันที่ทำสัญญา
contractStatus	สถานะสัญญาใช้ U แทนสัญญาที่ใช้งาน, T แทนสัญญาที่รอการปรับปรุงเปลี่ยนแปลง และ A แทนสัญญาที่มีการชำระครบแล้ว
paymentType	ประเภทการชำระเงิน ใช้ C แทนชำระเงินสด และ B แทนชำระผ่านธนาคาร

	pricePerTerm	เงินที่ต้องจ่ายต่องวด
	dateFirstTerm	วันที่ทำการจ่ายเงินงวดแรก
	term	จำนวนงวดที่ต้องจ่าย
	contractType	ประเภทของสัญญาใช้ B แทนสัญญาเช่าซื้อ, R แทนสัญญาเช่า, L แทนสัญญากู้เงิน, CB แทนเช่าซื้อ-ประนอมหนี้ และ CL แทนเงินกู้-ประนอมหนี้
	customerID	รายละเอียดลูกหนี้
	debtorCode	รหัสลูกหนี้
PaymentTransaction		การชำระเงินของลูกหนี้
	txID	รหัสสำหรับแต่ละ Transaction
	paymentDate	วันที่ของ Transaction ที่รับชำระ
	paymentTerm	งวดที่ชำระ
	contractID	เลขที่สัญญา
PaymentTransactionDetail		รายละเอียดในการชำระ
	txID	รหัสสำหรับแต่ละ Transaction
	accountID	รหัสบัญชี เช่น 0520
	paymentPrice	จำนวนเงินที่ชำระต่อรายการบัญชี
Receipt		ใบเสร็จ
	receiptID	เลขที่ใบเสร็จ
	receiptDate	วันที่ออกใบเสร็จครั้งแรก
	receiptStatus	สถานะของใบเสร็จ ซึ่งใช้ U แทน Use และใช้ C แทน Cancel
	txID	รหัสสำหรับแต่ละ Transaction

3.5.4 Method ของแต่ละ Class

การสื่อความหมายจากชื่อของ Method โดยใช้คำขึ้นต้นของ Method เป็นตัวสื่อความหมาย คือ หากขึ้นต้นด้วย New จะหมายความว่าเป็น Method ที่ใช้ในการสร้าง Object หากขึ้นต้นด้วย Get จะหมายถึงการดึงข้อมูลขึ้นมาให้ (เป็นการส่งค่ากลับคืนยังผู้เรียก) หากขึ้นต้นด้วย Set จะหมายถึงเป็นการกำหนดค่า Attribute ของ Object หากขึ้นต้นด้วย Verify จะหมายถึงเป็นการตรวจสอบและจะส่งค่ากลับคืน หากขึ้นต้นด้วย Calculate จะหมายความว่าเป็นการคำนวณที่ซับซ้อนและจะส่งค่ากลับคืน และหากขึ้นต้นด้วย Sum จะหมายถึงผลรวมของสิ่งใดสิ่งหนึ่งพร้อมกับส่งค่าคืน

AccountInformation

getAccountName	ค้นหาชื่อบัญชี จากเลขที่บัญชี

BankPayment

getDataFile	นำข้อมูลจากไฟล์ที่ทางธนาคารส่งมาให้เข้าสู่โปรแกรม
newBankPayment	เพิ่ม Object การรับชำระผ่านธนาคาร
verifyPrice	ตรวจสอบข้อมูลที่ธนาคารส่งมาให้

Contract

verifyContractID	ตรวจสอบรหัสสัญญา ว่ามีหรือจริง ใช่ได้หรือไม่
getContractType	ค้นหาประเภทของสัญญา จากเลขที่สัญญา
getContractStatus	ค้นหาสถานะของสัญญา จากเลขที่สัญญา
getContractAmountTerm	ค้นหาจำนวนงวดที่ต้องชำระทั้งหมด จากเลขที่สัญญา
getContractAmountDebt	ค้นหาจำนวนหนี้ทั้งหมดของสัญญา จากเลขที่สัญญา
getContractInterestRate	ค้นหาอัตราดอกเบี้ยของสัญญา จากเลขที่สัญญา
getContractPaymentType	ค้นหาประเภทของการชำระเงิน จากเลขที่สัญญา
getContractPricePerTerm	ค้นหาจำนวนเงินที่ต้องชำระต่องวด จากเลขที่สัญญา
getContractDateFirstTerm	ค้นหาวันที่ชำระเงินวันแรก จากเลขทีสัญญา
getCustomerContractID	หา List (ชุดข้อมูล) ของเลขที่สัญญา จากข้อมูลลูกค้า
getDebtorContract	หา List (ชุดข้อมูล) ของเลขที่สัญญา จากรหัสลูกหนี้
getEstimatePrice	คำนวณจำนวนเงินทั้งหมดต่องวดที่สัญญาควรจะชำระ (ประมาณรายรับ)

getEstimateCount คำนวณจำนวนสัญญาทั้งหมดที่สัญญาควรจะชำระ (ประมาณจำนวนราย)

getPayListContract หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญาและงวดที่ชำระ

sumPayListContract คำนวณผลรวมของ List (ชุดข้อมูล) รายละเอียดการชำระจาก List ที่ส่งเข้ามา

setAnnulContract กำหนดสถานะของสัญญา จากเลขที่สัญญา := ยกเลิกสัญญา (กรณีชำระหนี้หมด)

setTerminateContract กำหนดสถานะของสัญญา จากเลขที่สัญญา := สิ้นสุดสัญญา (เพื่อให้หน่วยงานที่เกี่ยวข้องทำการเปลี่ยนแปลงแก้ไขข้อมูลในสัญญาสำหรับ กรณีมาชำระบางส่วนที่มากกว่าจำนวนงวด)

setPaymentType กำหนดสถานะของการชำระเงินให้เป็นการชำระแบบเงินสด โดยจะกำหนดจากเลขที่สัญญา

getPaylistContract_R หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเช่าใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญาและงวดที่ชำระ

getPaylistContract_B หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเช่าซื้อใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญาและงวดที่ชำระ

getPaylistContract_L หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเงินกู้ใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญา

getPaylistContract_CL หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเงินกู้-ประนอมหนี้ใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญา

getPaylistContract_CB หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเช่าซื้อ-ประนอมหนี้ใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญา

getCompromiseInterestRate ค้นหาอัตราดอกเบี้ยของสัญญาประนอมหนี้ จากเลขที่สัญญา

getBuyInterestRate ค้นหาอัตราดอกเบี้ยของสัญญาเช่าซื้อ จากเลขที่สัญญา

getLoanInterestRate ค้นหาอัตราดอกเบี้ยของสัญญาเงินกู้ จากเลขที่สัญญา

getCompromiseAmountDebt ค้นหาจำนวนหนี้ทั้งหมดของสัญญาประนอมหนี้ จากเลขที่สัญญา

getBuyAmountDebt ค้นหาจำนวนหนี้ทั้งหมดของสัญญาเช่าซื้อ จากเลขที่สัญญา

getLoanAmountDebt ค้นหาจำนวนหนี้ทั้งหมดของสัญญาเงินกู้ จากเลขที่สัญญา

PaymentTransaction

newPaymentTX เพิ่ม Object การรับชำระ

calculateTerm คำนวณงวดที่จะชำระ

calculateFine คำนวณค่าปรับ

calculateRemainDebt คำนวณจำนวนเงินที่ลูกหนี้ชำระไปแล้ว

getContractTXID หา List (ชุดข้อมูล) ของ TXID จากเลขที่สัญญา

generateDailyCashPaymentReport สร้างรายงานรับชำระประจำวัน

generateMonthlyCashPaymentReport สร้างรายงานรับชำระประจำเดือน

generateIncomeEstimateReport สร้างรายงานประมาณการรายรับ-รับจริง

generateRemainDebtorReport สร้างรายงานลูกหนี้คงเหลือ (เป็นรายคน ว่าเหลือจำนวนหนี้ทั้งหมดเป็นจำนวนเท่าใด)

generateRemainDebtReport สร้างรายงานลูกหนี้คงค้าง

generateYClearing สร้างรายงานการตัดบัญชีได้

generateNClearing สร้างรายงานการตัดบัญชีไม่ได้

PaymentTransactionDetail

newPaymentTXD เพิ่ม Object รายละเอียดการรับชำระ

getPaymentDetail หา List ของรายละเอียด จาก TXID

Receipt

newReceipt เพิ่ม Object ใบเสร็จใหม่

setStatus กำหนดสถานะของ Object ใบเสร็จนั้นๆ ให้ยกเลิก

printReceipt พิมพ์ใบเสร็จ

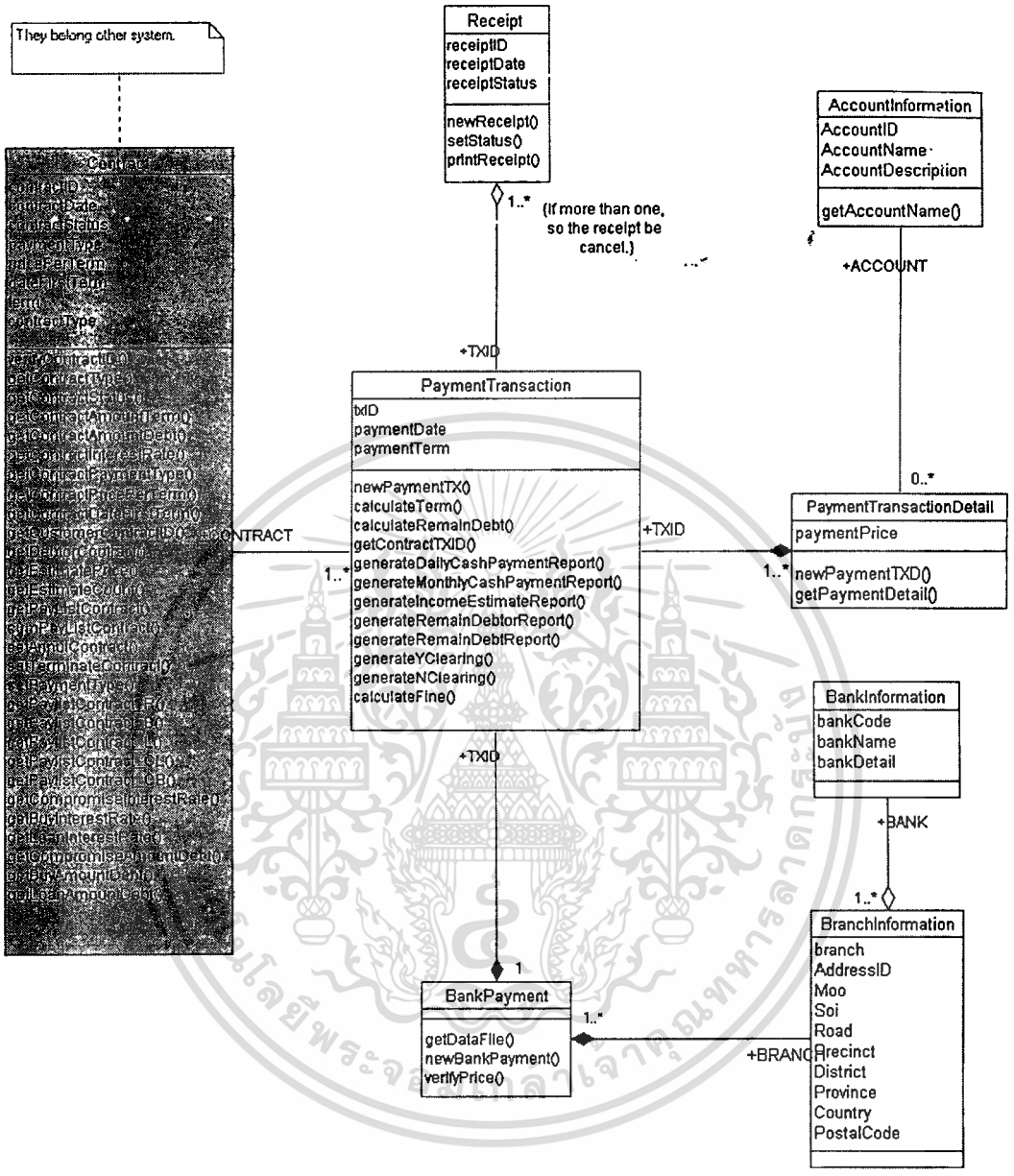
เนื่องจาก Class Contract เป็น Class ของระบบสัญญา แต่ Method ของ Class Contract ที่แสดงในที่นี้ เป็น Method ที่ทางระบบรับสร้างขึ้นมาเพื่อรองรับงานของระบบรับชำระ (จะไม่คำนึงถึง Method ที่อยู่ใน Class ภายในระบบสัญญาที่ถูกระบบสัญญาพัฒนาขึ้น)

จาก Attribute, Method, ความสัมพันธ์ของ Class ที่ได้มา สามารถจะวาดเป็นไดอะแกรมได้รูปที่ 3.13 (หน้าถัดไป)

จากรูปที่ 3.13 Class ที่มีสีทึกจะหมายถึง Class ที่ไม่ได้อยู่ในความรับผิดชอบของระบบ และจะเห็นว่า Attribute ที่กำหนดในตอนต้นนั้นนำมาแสดงไม่ใน Class ไม่หมด ในความเป็นจริง Class Diagram นี้ได้นำ Attribute ทั้งหมดมาสร้าง แต่ด้วยเครื่องมือที่ใช้นั้นมีการแสดงการอ้างถึงจาก Class หนึ่งไปยังอีก Class หนึ่งในรูปแบบที่เป็นของเครื่องมือนั้นเอง

นั่นคือ Rational Rose จะแสดง Attribute ที่เป็นการอ้างมาจาก Class อื่นในรูปแบบสัญลักษณ์ของ Role ที่ใช้ใน Class Diagram ดังนั้น Class ใดที่มีความสัมพันธ์กับ Class อื่น และมีบทบาท (Role) เป็นไปตามชื่อของบทบาท (ที่มีเครื่องหมาย + นำหน้า) นั้นก็จะแสดงว่า Role นั้นเป็น Attribute ตัวหนึ่งใน Class ที่มีความสัมพันธ์ด้วย

ตัวอย่างเช่น Class Receipt จะประกอบไปด้วย Attribute 4 ตัวซึ่งตัวหนึ่งก็คือ TXID ของ Class PaymentTransaction จากตัวอย่างนี้สามารถอธิบายความหมายได้คือ Class Receipt มีความสัมพันธ์กับ Class PaymentTransaction ในรูปแบบ Aggregation และ Class PaymentTransaction ทำหน้าที่ (มีบทบาท) เป็น TXID ของ Class Receipt



รูปที่.3.13 Class Diagram ของระบบรับชำระในขั้นการวิเคราะห์

จาก Class ที่ได้มาในขั้นการวิเคราะห์จะนำมาเป็นขอบข่ายในการออกแบบ โดยจะนำ Class ที่ได้มา มาทำการออกแบบให้เหมาะสมกับการสร้างระบบ ซึ่งระบบที่สร้างนี้จะทำงานอยู่บนระบบฐานข้อมูลแบบ Relational ของ Oracle เวอร์ชัน 8 และใช้ Oracle Developer 6 เข้ามาพัฒนาเป็น Application ของระบบรับชำระ

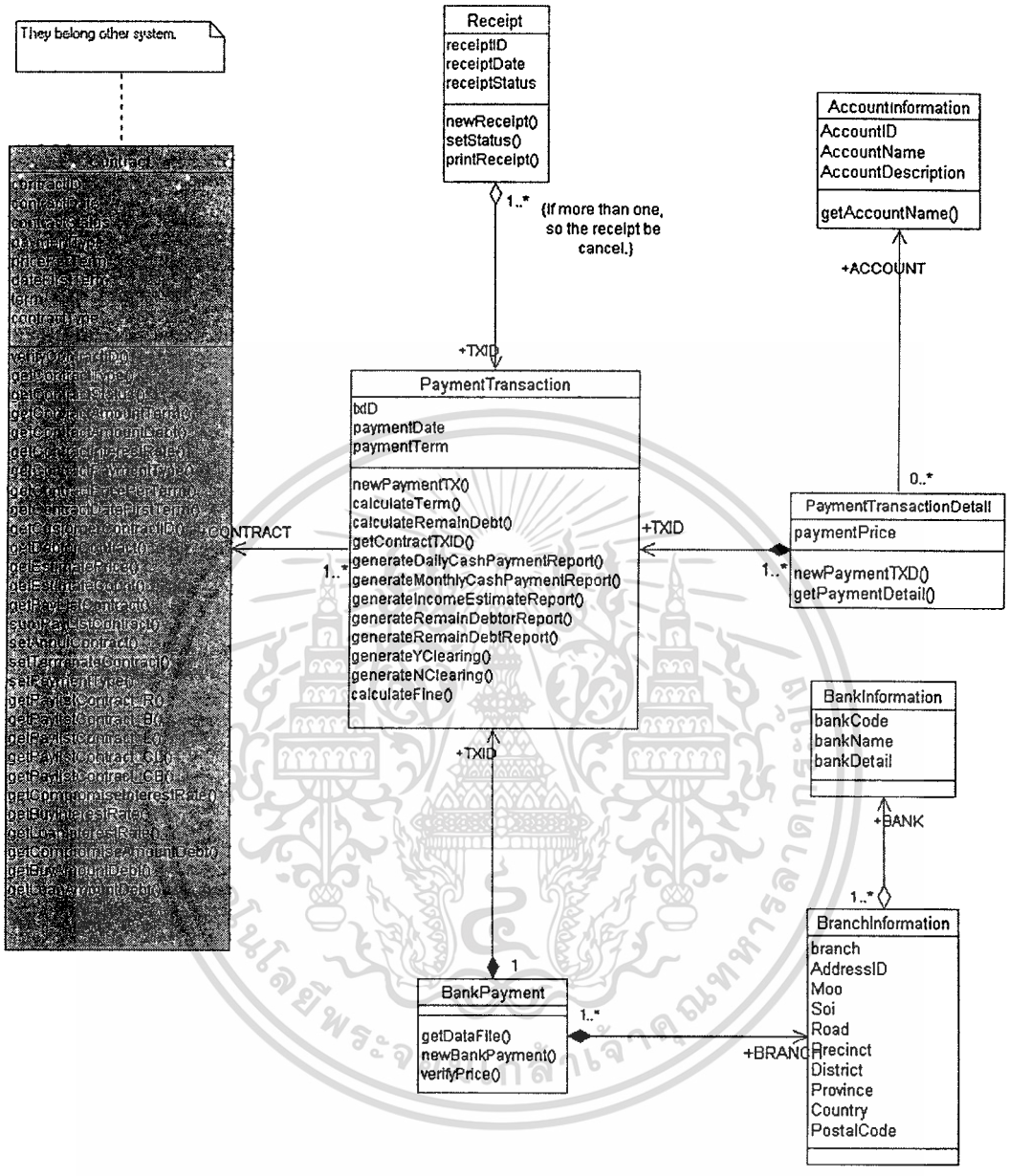
3.5.5 การออกแบบความสัมพันธ์

ความสัมพันธ์ของ Class Diagram จากรูปที่ 3.13 นั้นเป็นความสัมพันธ์แบบ Binary ซึ่งเป็นความสัมพันธ์แบบสองทาง หรือหมายความว่าระหว่างทั้ง 2 Class จะรู้ข้อมูลซึ่งกันและกันและในการ Implement นั้นจะต้องสร้าง Attribute ในการอ้างถึงไว้ทั้ง 2 ฝั่ง ซึ่งอาจไม่มีความจำเป็นเลย ทั้งนี้ขึ้นอยู่กับความต้องการของระบบด้วย ส่วนความสัมพันธ์อื่นๆ ในไดอะแกรมมีความชัดเจนแล้ว

จาก Class Diagram ในรูปที่ 3.13 นี้จะมีการกำหนดความสัมพันธ์ใหม่อีกครั้งเพื่อให้เหมาะสมกับการออกแบบ Attribute ต่อไป ความสัมพันธ์แบบ Binary ใดที่ไม่มีความจำเป็นให้ใช้ ความสัมพันธ์ทางเดียวแทน ซึ่งความสัมพันธ์แบบทางเดียวนี้สามารถอธิบายได้คือ Class ใด Class หนึ่งรู้ข้อมูลอีก Class ฝ่ายเดียว

ความสัมพันธ์ระหว่าง สัญญา (Contract) กับการชำระเงิน (PaymentTransaction) โดยปกติแล้วการชำระเงินต้องทราบว่าการชำระเงินครั้งนี้มาจากสัญญาใด แต่สัญญาไม่จำเป็นต้องทราบว่ามีการชำระอย่างไรเป็นแบบไหน ดังนั้นจึงสามารถเปลี่ยนความสัมพันธ์แบบสองทางไปเป็นความสัมพันธ์แบบทางเดียวได้ ส่วนในทางปฏิบัตินั้นสามารถจะทราบได้ทั้งสองกรณีที่ยกตัวอย่างมา อาจโดยการ Query ธรรมดาก็จะสามารถทราบได้

หลังจากการออกแบบความสัมพันธ์ครั้งใหม่นี้แล้วจะได้ Class Diagram ที่มีความชัดเจนมากขึ้นและสามารถทำให้ Rational Rose เข้าใจ Class Diagram ที่สร้างขึ้นมาได้ และได้แสดงไว้ดังรูปที่ 3.14



รูปที่ 3. 14 แสดง Class Diagram หลังจากการออกแบบความสัมพันธ์

3.5.6 การออกแบบ Attribute

เป็นการออกแบบ Attribute ให้เหมาะสมกับการพัฒนา ซึ่งจะหมายถึงการกำหนดประเภทข้อมูล ขนาดของข้อมูล ค่าเริ่มต้นและความสามารถในการเข้าถึงของแต่ละ Attribute อาจรวมถึงเพิ่มหรือลดจำนวน Attribute

AccountInformation

Protected Attributes:

AccountID : CHAR(4)

AccountName : VARCHAR2(30)

AccountDescription : VARCHAR2(255)

BankInformation

Protected Attributes:

bankCode : CHAR(4)

bankName : VARCHAR2(30)

bankDetail : VARCHAR2(255)

BankPayment

Protected Attributes:

txID : NUMBER(15)

branch : VARCHAR2(30)

bankCode : CHAR(4)

BranchInformation

Protected Attributes:

branch : VARCHAR2(30)

AddressID : VARCHAR2(20)

Moo : CHAR(2)

Soi : VARCHAR2(30)

Road : VARCHAR2(30)

Precinct : VARCHAR2(30)

District : VARCHAR2(30)

Province : VARCHAR2(30)

Country : VARCHAR2(30) = 'Thailand'

PostalCode : CHAR(5)

Contract

Protected Attributes:

contractID : CHAR(8)

contractDate : DATE

contractStatus : CHAR(1)

paymentType : CHAR(1)

pricePerTerm : NUMBER(11,2)

dateFirstTerm : DATE

term : NUMBER(5)

contractType : CHAR(1)

debtorCode : CHAR(14)

customerID : CHAR(8)

PaymentTransaction

Protected Attributes:

txID : NUMBER(15)

paymentDate : DATE

paymentTerm : NUMBER(5)

contractID : CHAR(8)

PaymentTransactionDetail

Protected Attributes:

txID : NUMBER(15)

AccountID : CHAR(4)

paymentPrice : NUMBER(11,2)

Receipt

Protected Attributes:

receiptID : CHAR(8)

receiptDate : DATE

receiptStatus : CHAR(1) = U

txID : NUMBER(15)

3.5.7 การออกแบบ Method

เป็นการกำหนด Protocol ในการติดต่อสื่อสารระหว่าง Class ซึ่งเป็นการกำหนด Visibility Parameter และค่าที่ส่งกลับคืนของ Method และอาจรวมถึงการออกแบบ Algorithm ในแต่ละ Method ด้วยก็ได้ ในบางกรณีอาจต้องมีการเพิ่ม Method ที่อำนวยความสะดวกในการสร้างระบบและการทำงานของระบบด้วย สำหรับ Method ที่เพิ่มขึ้นมาใหม่จะมีคำอธิบายประกอบ อยู่ข้างๆ

AccountInformation

Public Operations:

getAccountName (account_ID : CHAR) : VARCHAR2

BankPayment

Public Operations:

getDataFile (file : VARCHAR2)

newBankPayment (txID : NUMBER, bank : CHAR, branch : VARCHAR2) :

verifyPrice (contractID : CHAR) : BOOLEAN

Contract

Public Operations:

verifyContractID (contractID : CHAR) : BOOLEAN

getContractType (contractID : CHAR) : CHAR

getContractStatus (contractID : CHAR) : CHAR

getContractAmountTerm (contractID : CHAR) : NUMBER

getContractAmountDebt (contractID : CHAR) : NUMBER

getContractInterestRate (contractID : CHAR) : NUMBER

getContractPaymentType (aContactID : CHAR) : CHAR

getContractPricePerTerm (contractNum : CHAR) : NUMBER

getContractDateFirstTerm (contractNum : CHAR) : DATE

getCustomerContractID (customerID : VARCHAR2, tableOfContractID : TAB_TMP)

getDebtorContract (debtorCode : CHAR, tableOfContractID : TAB_TMP)

getEstimatePrice (con_type : CHAR) : NUMBER

getEstimateCount (con_type : CHAR) : NUMBER

getPayListContract (contractID : CHAR, term : NUMBER, tableOfList : TAB_TMP)

sumPayListContract (tableOfList : TAB_TMP) : NUMBER

setAnnulContract (contractStatus : CHAR = A)

setTerminateContract (contractStatus : CHAR = T)

setPaymentType (contractID : CHAR)

Protected Operations:

getPaylistContract_R (contractNum : CHAR, term : NUMBER, tableOfList : TAB_TMP)

getPaylistContract_B (contractNum : CHAR, term : NUMBER, tableOfList : TAB_TMP)

getPaylistContract_L (contractNum : CHAR, tableOfList : TAB_TMP)

getPaylistContract_CL (contractNum : CHAR, tableOfList : TAB_TMP)

getPaylistContract_CB (contractNum : CHAR, tableOfList : TAB_TMP)

getCompromiseInterestRate (contractNum : CHAR) : NUMBER

getBuyInterestRate (contractNum : CHAR) : NUMBER

getLoanInterestRate (contractNum : CHAR) : NUMBER

getCompromiseAmountDebt (contractNum : CHAR) : NUMBER

getBuyAmountDebt (contractNum : CHAR) : NUMBER

getLoanAmountDebt (contractNum : CHAR) : NUMBER

PaymentTransaction

Public Operations:

newPaymentTX (txID : NUMBER, date : DATE, term : NUMBER, contractID : CHAR)

calculateTerm (contractID : CHAR) : NUMBER

calculateFine (contractNum : CHAR, TableOfList : TAB_TMP)

calculateRemainDebt (contractID : CHAR) : NUMBER

getContractTXID (contractID : CHAR, tableOfTXID : TAB_TMP)

sumTXIDSet (txID_Set : TAB_TMP) : NUMBER

คำนวณผลรวมใน List (ชุดข้อมูล) ของ TXID

getRealPrice (con_type : CHAR, sdate : DATE, edate : DATE) : NUMBER

คำนวณจำนวนเงินทั้งหมดที่ลูกหนี้มาชำระ (คำนวณเงินที่รับจริง)

getRealCount (con_type : CHAR, sdate : DATE, edate : DATE) : NUMBER

คำนวณจำนวนรายทั้งหมดที่สัญญาทำการชำระ (คำนวณจำนวนที่รับจริง)

pcommit () :

บันทึกลงฐานข้อมูลจริง

generateDailyCashPaymentReport (date : DATE) :

generateMonthlyCashPaymentReport (startDate : DATE, endDate : DATE) :

generateIncomeEstimateReport (startDate : DATE, endDate : DATE) :

generateRemainDebtorReport (customerID : CHAR) :

generateRemainDebtReport (startDate : DATE, endDate : DATE) :

generateYClearing (startDate : DATE, endDate : DATE) :

generateNClearing (startDate : DATE, endDate : DATE) :

Protected Operations:

generateTXID () : NUMBER

สร้างเลขที่ของ Object การรับชำระ

PaymentTransactionDetail

Public Operations:

newPaymentTXD (txID : NUMBER, accountID : CHAR, price : NUMBER) :

getPaymentDetail (txID : NUMBER, tableOfList : VARCHAR2) :

sumPerAccount (acc_id : CHAR, sdate : DATE) : NUMBER

หาผลรวมของจำนวนเงินทั้งหมด จากเลขที่บัญชีและวันที่ที่กำหนด

sumPerAccount (acc_id : CHAR, sdate : DATE, edate : DATE) : NUMBER

หาผลรวมของจำนวนเงินทั้งหมด จากเลขที่บัญชีและช่วงของวันที่ที่กำหนด

countPerAccount (acc_id : CHAR, sdate : DATE) : NUMBER

หาผลรวมของจำนวนรายทั้งหมด จากเลขที่บัญชีและวันที่ที่กำหนด

countPerAccount (acc_id : CHAR, sdate : DATE, edate : DATE) : NUMBER

หาผลรวมของจำนวนรายทั้งหมด จากเลขที่บัญชีและช่วงของวันที่ที่กำหนด

Receipt

Public Operations:

newReceipt (receiptID : CHAR, receiptDate : DATE, txID : NUMBER) :

setStatus (receiptID : CHAR) :

printReceipt (txID : NUMBER) :

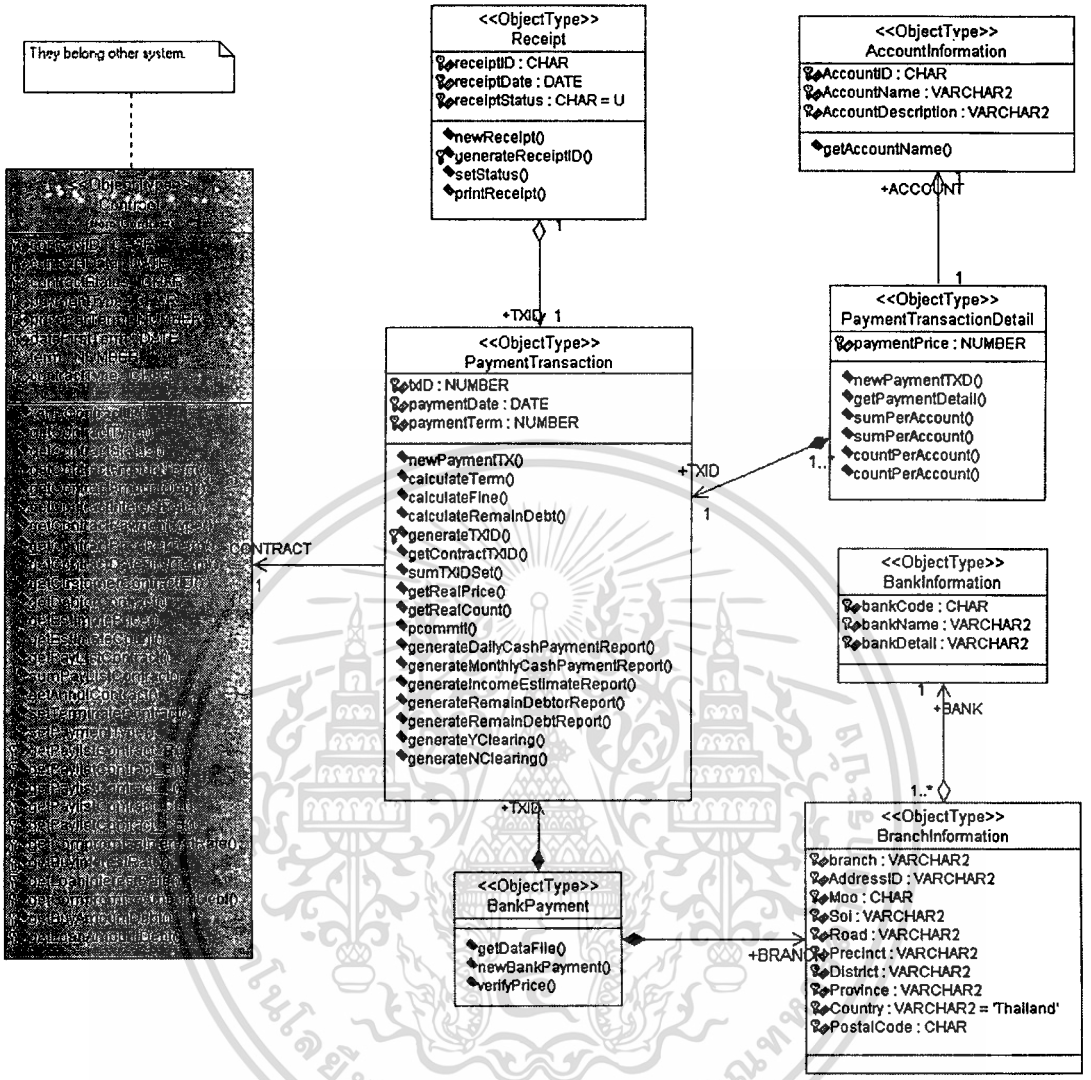
Protected Operations:

generateReceiptID (firstCharacter : CHAR = K) : CHAR

สร้างเลขที่ใบเสร็จใหม่

ประเภทข้อมูลของ Parameter ชื่อ TAB_TMP จะแทนด้วยตัวแปลที่เป็นลักษณะของ Array เพื่อเป็นทำให้ Method นั้นสามารถรับ-ส่งค่าได้มากกว่า 1 ค่าพร้อมกัน และส่วนของ Algorithm ของ Method ที่ซับซ้อนจะขอแสดงในบทที่ 6 (คู่มือระบบ) ข้อที่ 6.6.4 ซึ่งแสดง Method ของระบบ

หลังจากขั้นตอนการออกแบบ Class ใหม่ (การแก้ไขปรับปรุงให้มีความชัดเจนขึ้น) แล้วจะได้ Class Diagram ดังรูปที่ 3.15



รูปที่ 3. 15 แสดง Class Diagram ของระบบรับชำระในขั้นการออกแบบ

จากรูปที่ 3.15 จะเห็นได้ว่ามีการเพิ่มรายละเอียดลงไปใน Class ที่ได้จากการวิเคราะห์ และเป็นการสร้างแบบจำลองบน CASE Tool จึงมีความจำเป็นที่จะต้องใช้สัญลักษณ์ตาม CASE Tool ด้วย โดยส่วนมากแล้ว CASE Tool ที่ใช้นี้จะมีสัญลักษณ์เหมือนกับสัญลักษณ์ที่ใช้ใน UML แต่สัญลักษณ์ในส่วนที่เป็น Visibility นั้นจะแสดงในลักษณะดังตารางที่ 3.2

ตารางที่ 3. 2 แสดงสัญลักษณ์ Visibility ใน Class Diagram ตาม CASE Tool

สัญลักษณ์ที่ใช้	ความหมาย
	แทน Public
	แทน Private
	แทน Protect

3.6 Access Layer

ในขั้นตอนนี้เป็นการออกแบบ Class ที่ทำหน้าที่ในการติดต่อกับฐานข้อมูลจริงของระบบ และรวมถึงการสร้างฐานข้อมูลให้กับระบบอีกด้วย และในที่นี้จะเป็นการออกแบบเพื่อให้เหมาะสมกับระบบฐานข้อมูลแบบ Relational ของ Oracle ซึ่งเป็น DBMS ที่ใช้ในระบบรับชำระ

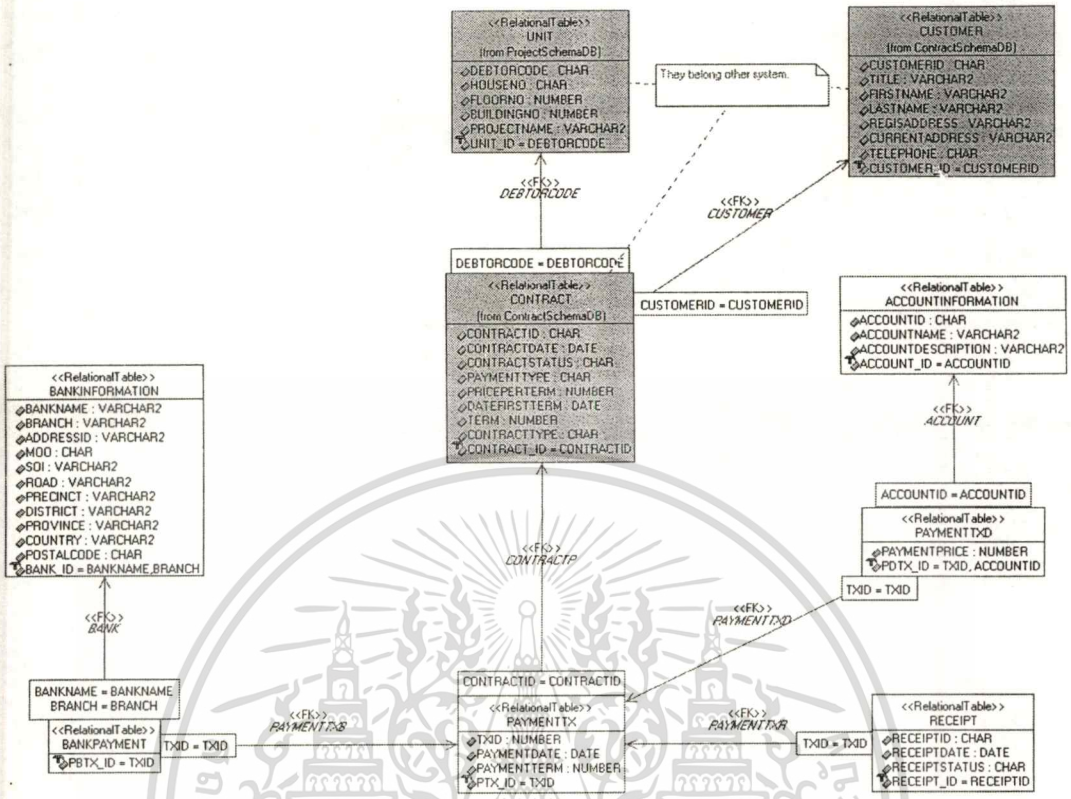
เนื่องจากระบบฐานข้อมูลที่ใช้ในการพัฒนาระบบเป็นแบบ Relational และการออกแบบที่ผ่านมานั้นเป็นการออกแบบเชิงวัตถุซึ่งมีทฤษฎีที่แตกต่างกัน เพราะระบบฐานข้อมูลแบบ Relational นั้นเป็นระบบฐานข้อมูลที่สามารถเข้าถึงข้อมูลได้โดยตรง (ในที่นี้จะไม่คำนึงถึงสิทธิของผู้ใช้งาน) แต่ทฤษฎีของ Object-Oriented นั้นโดยปกติแล้วจะไม่สามารถเข้าถึงข้อมูลได้โดยตรง จะเข้าถึงข้อมูลผ่านทาง Interface ของ Object

ในระบบฐานข้อมูลแบบ Relational นั้น Schema ต่างๆ จะถูกสร้างเป็น Table ซึ่ง Table จะประกอบไปด้วย Row (Tuple) และ Column และแต่ละ Column ก็จะมีชื่อและประเภทของข้อมูลประเภทเดียวกัน ส่วนในรูปแบบของ Object นั้นอาจเทียบได้ว่า Class เป็น Table, Object เป็น Row และ Method เป็น Store Procedure ในระบบฐานข้อมูลแบบ Relational

จากความแตกต่างดังกล่าวจึงต้องมีการปรับแบบจำลองที่ออกมาให้มีความเหมาะสมกับระบบฐานข้อมูลที่จะสร้างโดยอิงตามลักษณะของระบบฐานข้อมูล จะมีการปรับ Class ที่ได้ในการออกแบบคือจะสร้าง Class Diagram แสดง Class ที่มีแต่ Attribute อย่างเดียว กำหนดความสัมพันธ์ของ Class เหล่านั้นและทำการเพิ่ม Attribute ที่ทำหน้าที่เป็น Primary Key และ Foreign Key เพื่อเทียบให้เป็น Table ในระบบฐานข้อมูลแบบ Relation

ได้ทำการเพิ่ม Class บางตัวขึ้นมาเพื่อแสดงให้เห็นความสัมพันธ์ในการเชื่อมโยงข้อมูลกันอย่างชัดเจน สำหรับ Class ที่เพิ่มเข้ามาใหม่นั้นเป็น Class Unit และ Class Customer ซึ่ง Class ทั้งสองนี้อยู่ในความรับผิดชอบของหน่วยงานอื่น สำหรับ Class Unit นี้จะเป็น Class ที่แทนด้วยที่อยู่อาศัย ซึ่งจะเก็บรายละเอียดของตัวบ้านแต่ละหลัง ส่วน Class Customer จะเป็น Class ที่แทนด้วยลูกค้า (ลูกหนี้) ซึ่งจะเก็บรายละเอียดข้อมูลเกี่ยวกับลูกค้าเอง

มีการเปลี่ยนแปลง Class BankInfromation และ Class BranchInformation โดยจะทำการรวมทั้งสอง Class เข้าเป็น Class เดียวกันเพื่อความเหมาะสมในการสร้างและพัฒนาระบบ หลังจากมีการปรับ Class ต่างๆ ให้เหมาะสมกับสภาวะแวดล้อมของระบบฐานข้อมูลแบบ Relational ในที่สุดจะได้ผลลัพธ์ที่แสดงไว้ดังรูปที่ 3.16

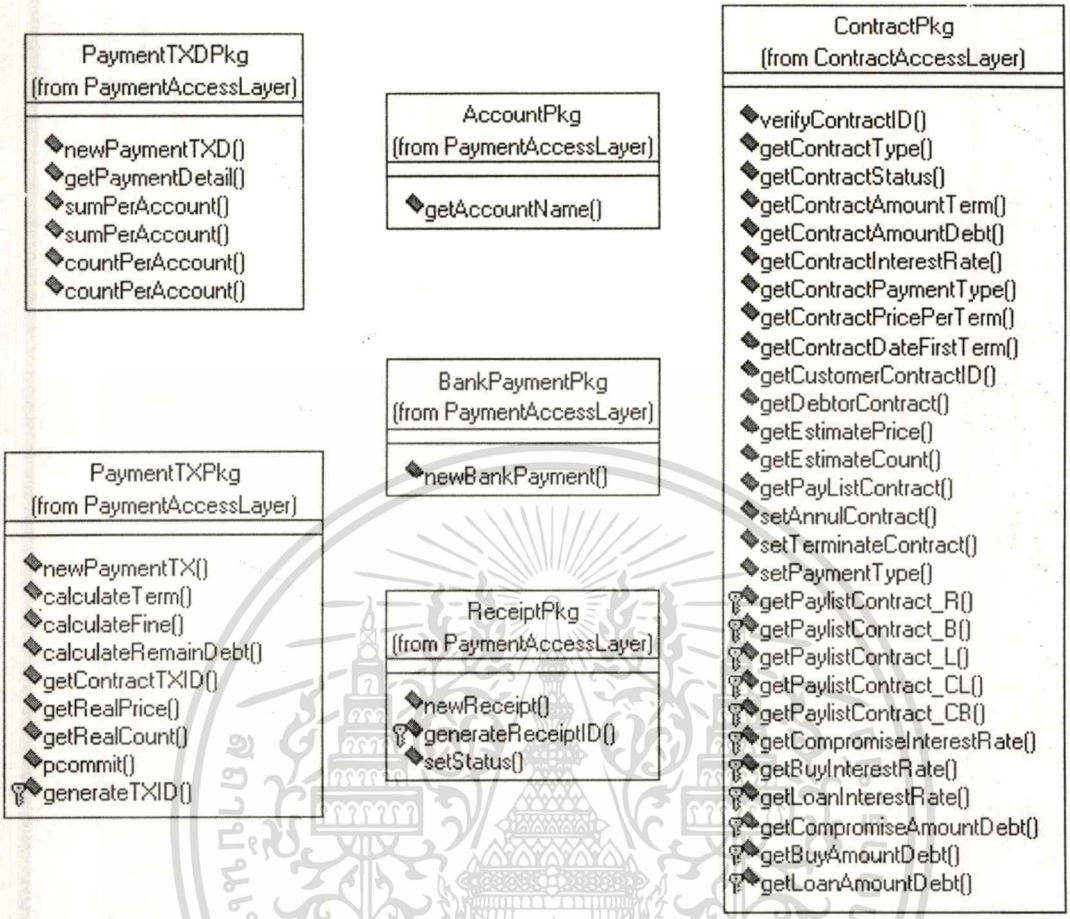


รูปที่ 3.16 แสดง Class ที่เหมาะสมกับระบบฐานข้อมูลแบบ Relational

จากรูปที่ 3.16 จะใช้สัญลักษณ์ Qualifier (ซึ่งเป็นสัญลักษณ์รูปสี่เหลี่ยมเล็กที่อยู่ติดกับตัว Class) ของ Class Diagram ใน UML แทนด้วย Foreign Key และแสดงชื่อของความสัมพันธ์ระหว่างแต่ละ Class พร้อมกับ Stereotype เป็น FK ด้วย สำหรับ Qualifier ที่มีการกำหนด Attribute มากกว่า 1 ตัวแสดงว่า Attribute เหล่านั้นเป็น Foreign Key เช่นกัน และยังมีการใช้สัญลักษณ์รูปที่คล้ายกับเครื่องมือในการขุด ( ) ซึ่งจะแสดงถึง Attribute สำหรับการสร้างและในที่นี้ให้กำหนดให้เป็น Index ที่ใช้ร่วมกับ Primary Key ใน Class

การทำงานสภาวะแวดล้อมใดๆ ก็ตามควรจะพยายามทำงานอยู่ในสภาวะแวดล้อมนั้นอย่างมีประสิทธิภาพ และเมื่อมีการทำงานอยู่ในสภาวะแวดล้อมของระบบฐานข้อมูลแบบ Relational จึงจำเป็นที่ต้องพิจารณา Class Diagram (หลังจากการปรับ) นั้นว่าเกิดปัญหาความซ้ำซ้อนของข้อมูลหรือไม่ หากเกิดปัญหาความซ้ำซ้อนของข้อมูลจะต้องมีการทำ Normalize ให้เหมาะสมด้วย สำหรับรูปที่ 3.16 นี้ไม่เกิดปัญหาความซ้ำซ้อนของข้อมูล

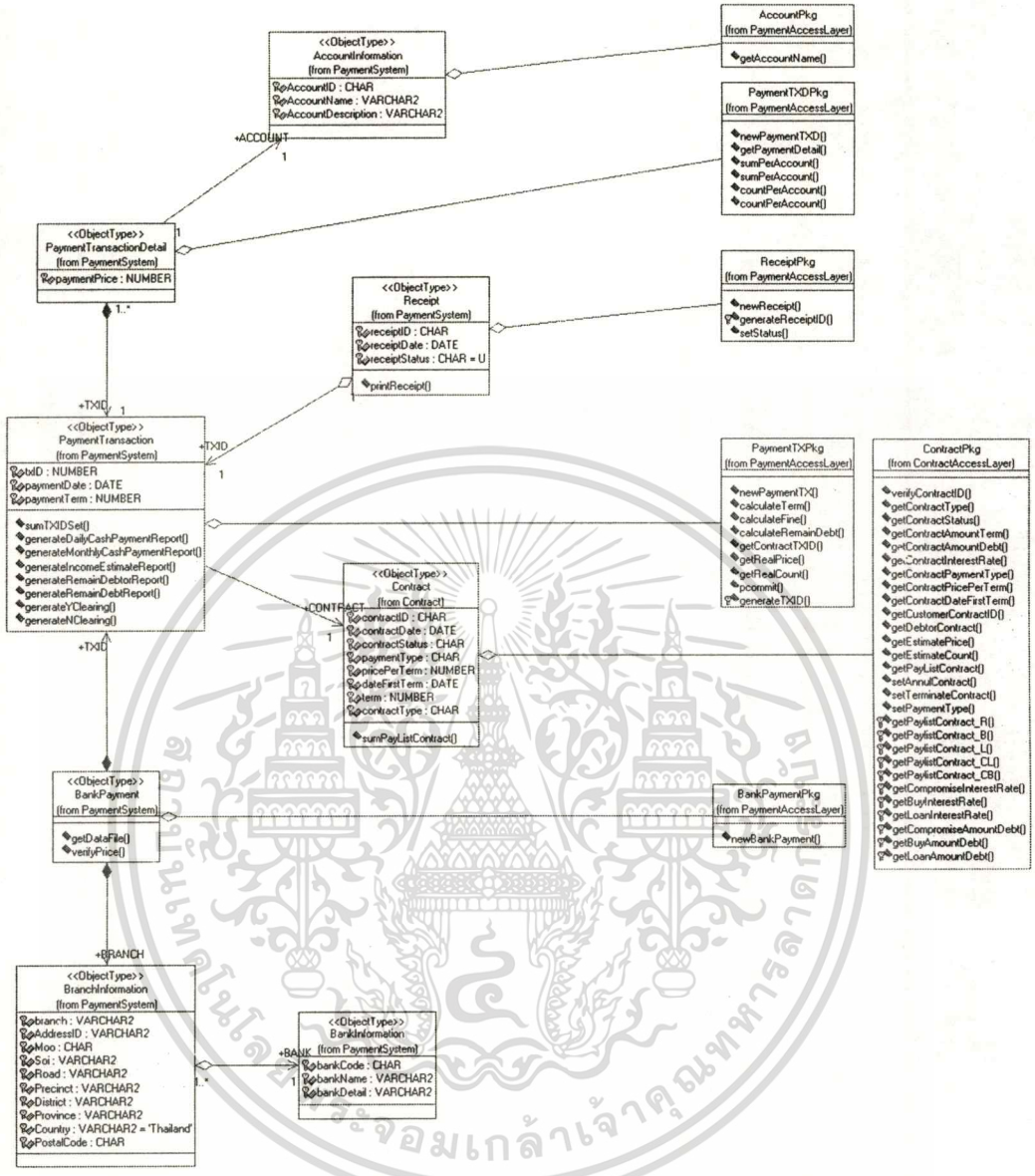
ในการสร้าง Class ที่มีหน้าที่รับผิดชอบในการเข้าถึงข้อมูลจะพิจารณาว่า Method ของ Class ใดที่เป็นการทำงานได้โดยไม่จำเป็นจะต้องใช้ข้อมูลที่เก็บอยู่ในฐานข้อมูลก็ยังคงไว้ที่เดิม นอกนั้นจะทำการย้าย Method ไปไว้ใน Class ที่สร้างขึ้นมาใหม่ สามารถแสดงได้ดังรูปที่ 3.17



รูปที่ 3.17 แสดง Class Diagram ที่อยู่ใน Access Layer

ส่วน Method ของ Class ใน Access Layer นี้ในการสร้างระบบจะสร้างเป็น Store Procedure ไว้ใน DBMS และเพื่อยังคงรักษาทฤษฎีของ Object-Oriented ไว้คือ Object มีการซ่อนข้อมูลของตัวมันไว้ภายในโดยที่ผู้อื่นไม่สามารถมองเห็น (Encapsulate) ดังนั้นจะสร้าง Store Procedure แบบที่เป็น Package และสร้างแต่ละ Package สำหรับ Class หนึ่ง Class

ความสัมพันธ์ระหว่าง Access Layer กับ Class Diagram ที่มีการย้าย Method แล้ว (จะเรียกว่า Business Layer) จะเป็นความสัมพันธ์แบบ Aggregation และจะแสดง Class ทั้งหมดที่ได้ในการออกแบบ (โดยไม่รวมโครงสร้างของฐานข้อมูล) ไว้ดังรูปที่ 3.18



รูปที่ 3. 18 แสดงความสัมพันธ์ระหว่าง Access Layer กับ Business Layer

3.7 View Layer

เป็นการออกแบบ Class ที่ทำหน้าที่เป็น Interface ของระบบ ซึ่งเป็น Interface ระหว่างผู้ใช้งานกับระบบ หรือเป็น User Interface (UI) ของระบบ และนำ Class เหล่านั้นมาพัฒนาต่อ อาจเป็นในรูปแบบของ Graphic User Interface (GUI) ก็ได้

เนื่องจากระบบที่พัฒนานี้เป็นการพัฒนาระบบซอฟต์แวร์ จึงจำเป็นต้องมีการสร้าง Class ขึ้นมาใหม่เพื่อทำหน้าที่เป็น User Interface ให้กับระบบเพราะที่ผ่านมานั้นมิได้คำนึงถึงในส่วนนี้

ในการออกแบบครั้งแรกคือจะเป็นการกำหนด Class ที่ทำหน้าที่เป็น Interface ของระบบก่อน ซึ่งอาจจะอาศัยจากการวิเคราะห์แบบจำลอง Use Case ที่ผ่านมาช่วยในการออกแบบได้ และในการออกแบบนี้จะอาศัย Use Case เป็นส่วนในการกำหนด Class ที่ทำหน้าที่เป็น User Interface โดยจะดูที่ Use Case ในมุมมองของผู้ใช้งานภายในระบบเป็นหลัก (รูปที่ 3.5) เพราะผู้ใช้งานกับระบบที่พัฒนาขึ้นนั้นเป็นผู้ที่ทำงานอยู่ในการเคหะแห่งชาติ

จากใน Use Case Diagram นี้สามารถบอกให้ทราบถึงงานต่างๆ ที่ผู้ใช้งานจะทำงาน ดังนั้นจึงสามารถกำหนด Class ที่ทำหน้าที่เป็น User Interface ได้ดังนี้คือ

- CashPaymentUI
- BankPaymentUI
- MorePaymentUI
- PaymentReportUI

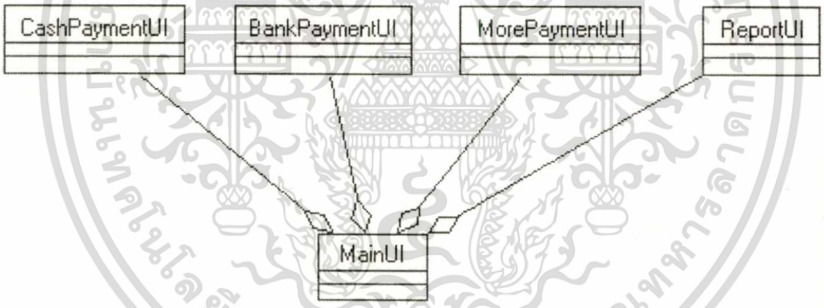
โดยที่ Class CashPaymentUI นี้จะเป็น Class ที่ทำหน้าที่เป็น User Interface สำหรับการรับชำระเงินสด Class BankPaymentUI จะเป็น Class ที่ทำหน้าที่เป็น User Interface สำหรับการรับชำระผ่านธนาคาร ส่วน Class MorePaymentUI นี้จะเป็น Class ที่ทำหน้าที่เป็น User Interface สำหรับการรับชำระเงินสดที่มีจำนวนเงินมากกว่าจำนวนเงินต่องวด ซึ่งจะใช้ในทั้งการหนี้ชำระบางส่วนและการชำระหนี้ทั้งหมด และส่วน Class PaymentReportUI นี้จะเป็น Class ที่ทำหน้าที่เป็น User Interface สำหรับการออกรายงานการรับชำระของระบบ

จาก Class ที่ได้ผ่านนั้นเป็น User Interface ที่รับหน้าที่ ที่ต่างกันไป หากนึกถึงในกรณีที่ผู้ใช้งานจะใช้ระบบจะต้องเข้ามาทำงานกับ Class เหล่านี้แยกกัน ทำให้เกิดความไม่สะดวกเกิดขึ้นดังนั้นจึงมีการสร้าง Class ขึ้นมาใหม่จะให้ชื่อว่า MainPaymentUI ที่ทำหน้าที่อำนวยความสะดวกให้แก่ผู้ใช้งานโดยเป็น User Interface หลักในของระบบ และให้ผู้ใช้งานเข้ามาทำงานกับ Class ในตอนเริ่มต้นใช้ระบบ เพราะ Class นี้จะมีทางเข้าไปสู่ Class ต่างๆ ได้อย่างสะดวก ฉะนั้นจะได้ Class ที่เป็น User Interface ทั้งหมดดังตารางที่ 3.3

ตารางที่ 3.3 แสดง Class ที่ทำหน้าที่เป็น User Interface ของระบบรับชำระ

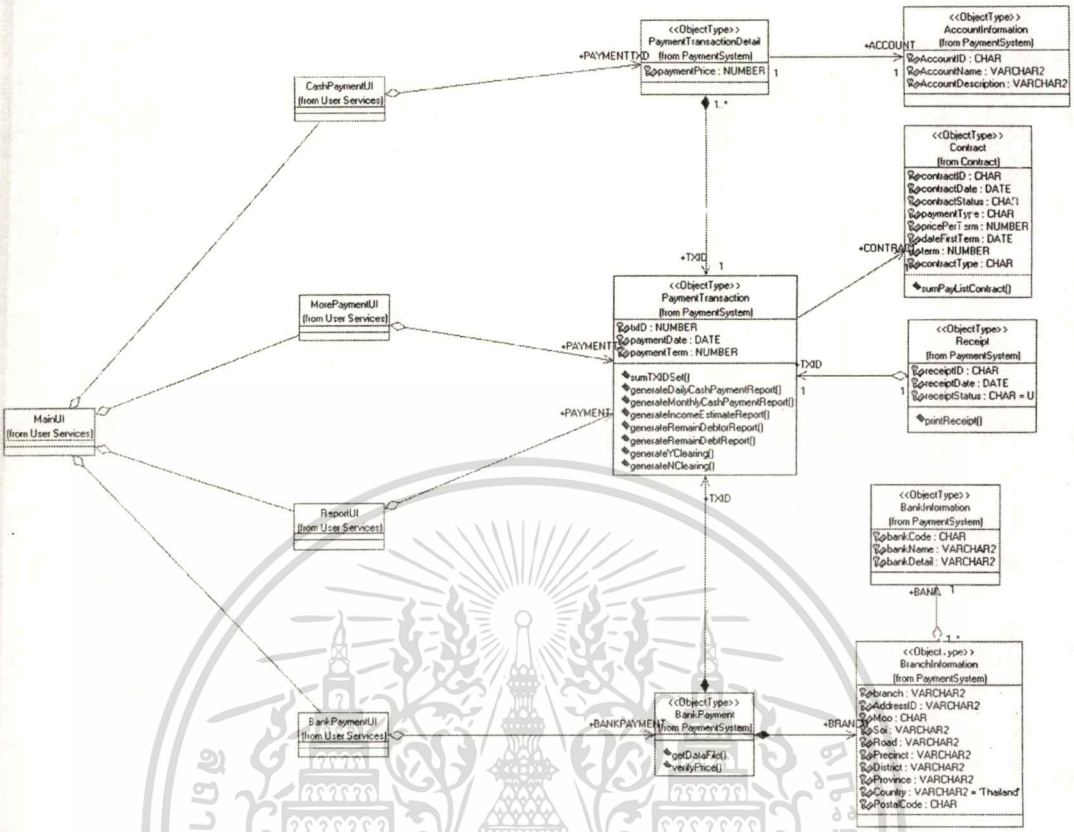
ชื่อ Class	คำอธิบาย
MainPaymentUI	ที่ทำหน้าที่เป็น User Interface หลักในการใช้ระบบ
CashPaymentUI	ที่ทำหน้าที่เป็น User Interface สำหรับการรับชำระเงินสด
BankPaymentUI	ที่ทำหน้าที่เป็น User Interface สำหรับการรับชำระผ่านธนาคาร
MorePaymentUI	ที่ทำหน้าที่เป็น User Interface สำหรับการรับชำระเงินสดที่จำนวนเงินมากกว่าจำนวนเงินต่องวด ทั้งชำระหนี้ที่เหลือเพียงบางส่วนและชำระหนี้ที่เหลือทั้งหมด
PaymentReportUI	ที่ทำหน้าที่เป็น User Interface สำหรับการออกรายงานในระบบรับชำระ

หลังจากได้ Class ที่เป็น User Interface แล้วต่อไปจะเป็นการกำหนดความสัมพันธ์ของ Class เหล่านี้เพื่อให้เห็นโครงสร้างของ User Interface ชัดขึ้น ซึ่งจะได้ดังรูปที่ 3.19



รูปที่ 3.19 แสดงความสัมพันธ์ระหว่าง Class ที่ทำหน้าเป็น User Interface

ความสัมพันธ์ระหว่าง Class ที่ได้ในขั้นการออกแบบ View Layer นี้กับ Class ที่ได้ในออกแบบช่วงต้นจะเป็นความสัมพันธ์แบบ Aggregation ของ Class ทำงานร่วมกันซึ่งแสดงไว้ในรูปที่ 3.20



รูปที่ 3. 20 แสดงความสัมพันธ์ระหว่าง View Layer กับ Business Layer

จากความสัมพันธ์และ Class ที่ได้จะใช้ในการออกแบบและสร้าง Prototype ขึ้นมาเป็นแบบรูปภาพ (GUI) โดยสิ่งที่จะแสดงใน GUI นี้อาจจะพิจารณาจาก Attribute ที่อยู่ใน Class ที่มีความสัมพันธ์กันใน Business Layer ช่วยก็ได้ ซึ่ง Prototype ที่ได้มานั้นอาจเป็นต้นแบบในการช่วยพัฒนาต่อไปได้

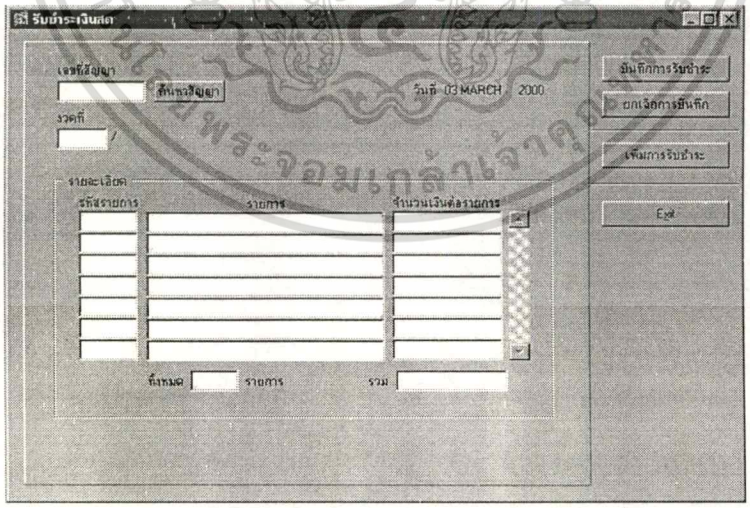
การสร้าง Prototype ในช่วงการออกแบบนี้จะใช้ Oracle Developer 6 ในการสร้างซึ่งจะได้ GUI ดังนี้

Class MainPaymentUI หลังจากการสร้างเป็น GUI จะได้ดังรูปที่ 3.21

รูปที่ 3.21 แสดง Graphic User Interface หลักของระบบ

ใน GUI นี้จะเป็นส่วนของหน้าจอหลักในการทำงานของผู้ใช้งาน โดยมีการทำงานหลักเป็นเพียงการเรียกให้ GUI อื่นขึ้นมาเท่านั้น ดังนั้นจึงไม่มีการบันทึกหรือแสดงข้อมูลใดๆ ที่อยู่ในระบบฐานข้อมูล

Class CashPaymentUI หลังจากการสร้างเป็น GUI จะได้ดังรูปที่ 3.22



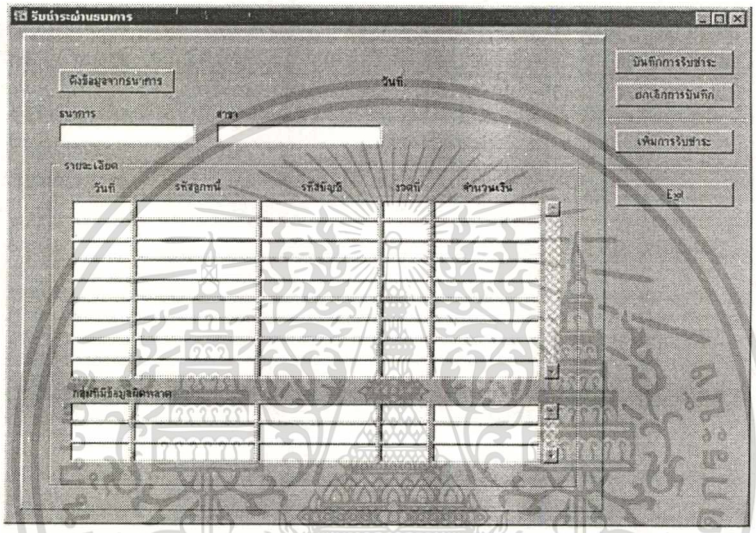
รูปที่ 3.22 แสดง Graphic User Interface ของการรับชำระเงินสดแบบปกติ

GUI นี้จะเป็นส่วนของหน้าจอที่ทำหน้าที่รับชำระเงินสดแบปกติของลูกนี้โดยที่จะทำหน้าที่ในการแสดงรายการที่ลูกหนี้จะต้องชำระตามที่ในสัญญาระบุไว้ โดยเริ่มจากให้ผู้ใช้งานใส่เลขที่สัญญาลงไป หรือทำการค้นหาเพื่อให้ได้เลขที่ของลูกหนี้คนนั้น จากนั้นภายใน GUI นี้จะทำการ

คำนวณงวดในครั้งนี้ให้โดยการใช้ Store Procedure และจะไปเรียก Store Procedure ที่ทำงานเกี่ยวกับการนำรายการที่สัญญานี้ต้องชำระมาแสดงบน GUI นี้

เมื่อลูกหนี้ชำระเงินให้ ผู้ใช้งานจะทำการบันทึกข้อมูลลงไปในระบบฐานข้อมูลพร้อมกับพิมพ์ใบเสร็จรับเงินให้ลูกหนี้ และภายใน GUI นี้จะนำข้อมูลที่แสดงขึ้นมาไปบันทึกลงไปในระบบฐานข้อมูลและจะบันทึกข้อมูลในส่วนของใบเสร็จรับเงิน โดยอาศัย Store Procedure ที่สร้างขึ้นมา

Class BankPaymentUI หลังจากการสร้างเป็น GUI จะได้ดังรูปที่ 3.23

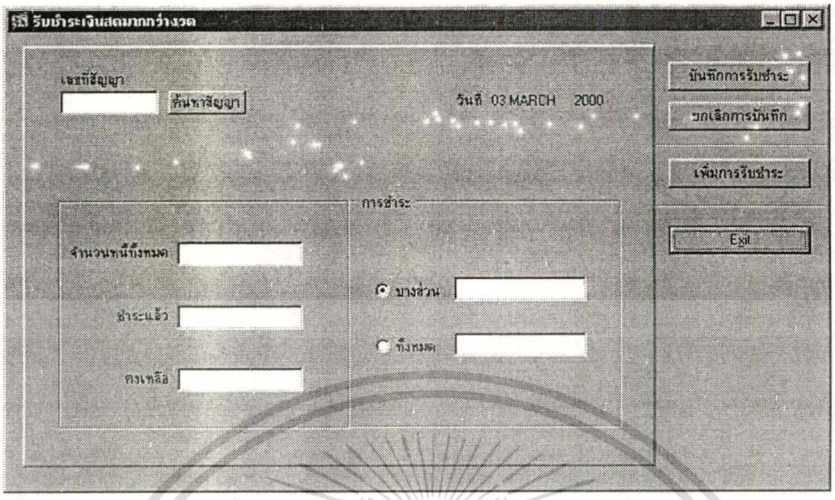


รูปที่ 3.23 แสดง Graphic User Interface ของการรับชำระผ่านธนาคาร

GUI นี้จะเป็นส่วนหน้าจอที่ทำหน้าที่เกี่ยวกับการรับชำระผ่านธนาคาร ซึ่งทางธนาคารจะส่งแผ่นดิสก์ ที่เป็นข้อมูลการหักบัญชีมาให้ ผู้ใช้งานจะต้องนำแผ่นดิสก์ที่ทางธนาคารส่งมาให้เป็นข้อมูลในส่วน Input ขณะที่มีการนำข้อมูลในแผ่นดิสก์ขึ้นมาแสดงใน GUI นี้นั้นก็จะมีการตรวจสอบความถูกต้องของข้อมูลพร้อมไปด้วย และแสดงข้อมูลในแผ่นดิสก์ขึ้นมาใน GUI นี้ในที่สุด

เมื่อผู้ใช้งานทราบว่าข้อมูลชุดใดมีปัญหาอาจทำการติดต่อกลับไปยังธนาคารเพื่อทำการตรวจสอบอีกครั้ง และผู้ใช้งานสามารถบันทึกข้อมูลที่แสดงอยู่ใน GUI นี้ลงไปในระบบฐานข้อมูลได้ (จะบันทึกเฉพาะข้อมูลที่ไม่มีปัญหาจะถูกบันทึกลงระบบฐานข้อมูลเท่านั้น) และจะทำการพิมพ์ใบเสร็จรับเงินให้ผู้ที่ถูกบันทึกเรียบร้อยแล้ว โดยการเรียก Store Procedure ช่วยในการบันทึกข้อมูลที่อยู่บน GUI นี้

Class MorePaymentUI หลังจากการสร้างเป็น GUI จะได้ดังรูปที่ 3.24

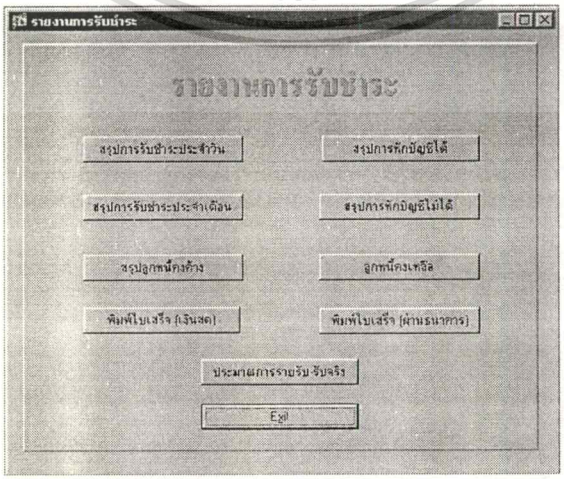


รูปที่ 3.24 แสดง Graphic User Interface ของการรับชำระเงินสดมากกว่างวด

เป็น GUI ที่ใช้ในกรณีพิเศษคือกรณีที่ลูกหนี้มาชำระเงินที่เป็นจำนวนมาก (มากกว่างวด) จะเริ่มต้นโดยให้ผู้ใช้ใส่เลขที่สัญญาของลูกหนี้ที่มีความประสงค์ และภายใน GUI จะมีการไปเรียก Store Procedure ที่ทำหน้าที่ในการคำนวณจำนวนเงินหนี้ที่ค้างอยู่ และจะแสดงเปรียบเทียบกับจำนวนหนี้ทั้งหมด เมื่อผู้ใช้งานทราบว่าลูกหนี้รายนี้มีการชำระเป็นอย่างไรก็เลือกการรับชำระให้ถูกต้องกับการชำระของลูกหนี้

หากลูกหนี้มีการชำระเงินแล้ว ผู้ใช้งานสามารถจะทำการบันทึกและพิมพ์ใบเสร็จรับเงินให้กับลูกหนี้ โดยอาศัย Store Procedure ที่สร้างขึ้นมา

Class PaymentReportUI หลังจากการสร้างเป็น GUI จะได้ดังรูปที่ 3.25

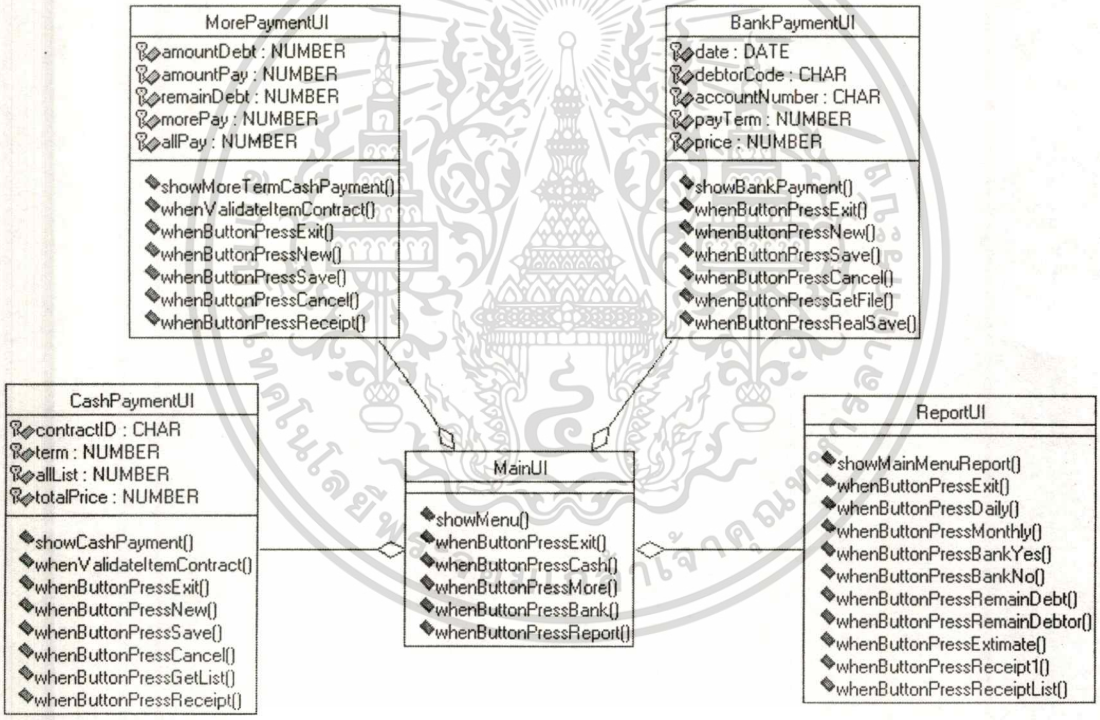


รูปที่ 3.25 แสดง Graphic User Interface ของการเลือกพิมพ์รายงาน

GUI นี้ทำหน้าที่เป็นช่องทางในการเลือกพิมพ์รายงานของระบบรับชำระ ดังนั้นผู้ใช้งานจะทำการพิมพ์รายงานของระบบรับชำระได้ โดยอาศัยปุ่มต่างๆ ที่อยู่ใน GUI นี้ ซึ่งภายใน GUI นี้จะมีการเรียก Store Procedure ในการสร้างรายงานขึ้นมา และ GUI นี้จะไม่มีการบันทึกข้อมูลเข้าสู่ระบบฐานข้อมูลเลย

จาก Class ทั้งหมดที่ทำหน้าที่เป็น User Interface จะมีหน้าที่หลักคือแสดงข้อมูลให้ผู้ใช้งาน และทำการเรียก Store Procedure ให้ทำงาน ดังนั้นจะมี Method ภายใน Class เป็นลักษณะของการแสดงผล ส่วนการเรียก Store Procedure ให้ทำงานนั้นเป็นลักษณะของการส่ง Message

ส่วน Attribute นั้นจะเป็นสิ่งที่แสดงอยู่ใน GUI นั้นๆ และเมื่อนำส่วนต่างๆ ทั้งหมดมาสร้างเป็น Class Diagram จะได้ดังรูปที่ 3.26



รูปที่ 3.26 แสดง Class Diagram ในขั้นการออกแบบ View Layer

3.8 Physical Architecture

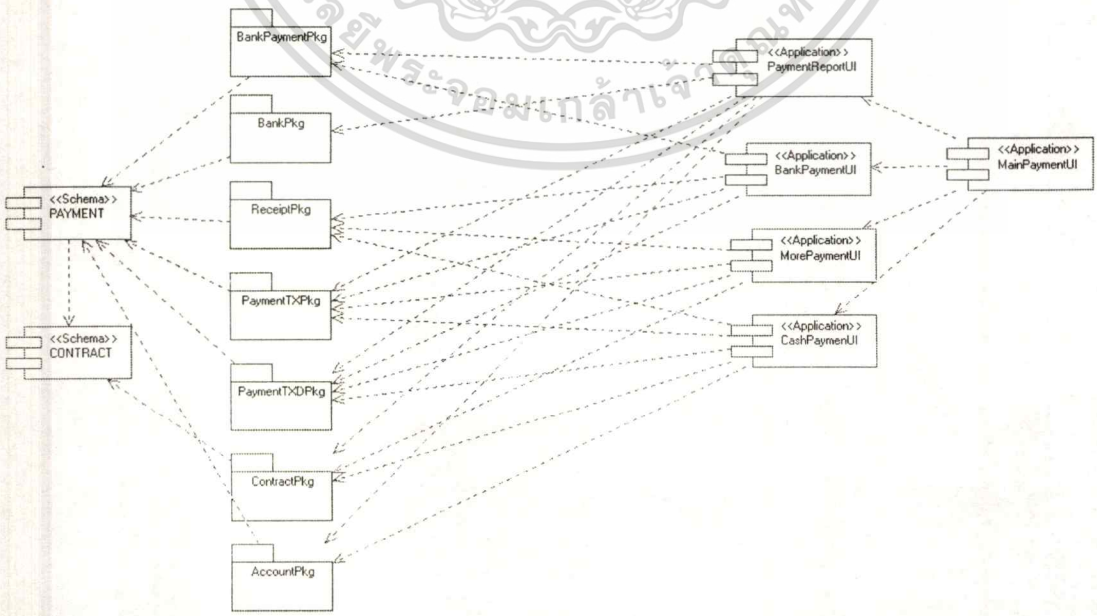
เป็นการแสดงโครงสร้างของฮาร์ดแวร์และซอฟต์แวร์ที่ถูกออกแบบในระบบ ซึ่งจะเป็นการแสดงโครงสร้างทางเทคนิคซึ่งจะเกี่ยวข้องกับการสร้างระบบ เช่นแสดงการทำงานของซอฟต์แวร์ขณะกำลังทำงานอยู่ (Run-Time) เป็นต้น ในขั้นตอนนี้จะเป็นการเปลี่ยนรูปแบบของ Class ที่ได้ในการออกแบบที่ผ่านมา ให้อยู่ในรูปแบบของ Component ต่างๆ ของซอฟต์แวร์ โดยอาศัย Component Diagram และ Deployment Diagram

3.8.1 Component Diagram

เป็นลักษณะการออกแบบสถาปัตยกรรมซอฟต์แวร์ของระบบ โดยการนำสิ่งที่ได้ออกแบบในขั้นต้นของการพัฒนา มาเปลี่ยนรูปแบบให้อยู่ในรูปของส่วนประกอบต่างๆ ของทางซอฟต์แวร์ และต่อไปนี้จะเป็นการเปลี่ยนรูปแบบสิ่งที่ได้ออกแบบของระบบรับชำระโดยจะ

- เปลี่ยนรูปแบบจาก Class ที่ทำหน้าที่เป็น Schema ของระบบ (ในขั้นการออกแบบ Access Layer) ไปอยู่ในรูปแบบที่เป็น Schema จริงในการพัฒนา
- เปลี่ยนรูปแบบจาก Class ที่ทำหน้าที่ในการเข้าถึงข้อมูลของระบบ (ในขั้นการออกแบบ Access Layer) ไปอยู่ในรูปแบบของ Store Procedure (เป็นแบบ Package) ที่อยู่ในฝั่ง Server
- เปลี่ยนรูปแบบจาก Class ที่อยู่ในส่วนของกระบวนการ การทำงานของระบบ (ในขั้นการออกแบบ Business Layer) ให้อยู่ในรูปแบบของ Procedure หรือ Function ของระบบซอฟต์แวร์ (ซึ่งในกรณีจะอยู่ใน Component ที่เป็น UI)
- เปลี่ยนรูปแบบจาก Class ที่ทำหน้าที่เป็นส่วนในการโต้ตอบระหว่างผู้ใช้งานกับระบบ (ในขั้นการออกแบบ View Layer) ให้อยู่ในรูปแบบของ GUI ซึ่งเป็น Application Software ของระบบ

จากการเปลี่ยนรูปแบบดังกล่าวจะสามารถแสดงภาพรวมของระบบให้อยู่ในรูปแบบของ Component Diagram ได้ดังรูปที่ 3.27



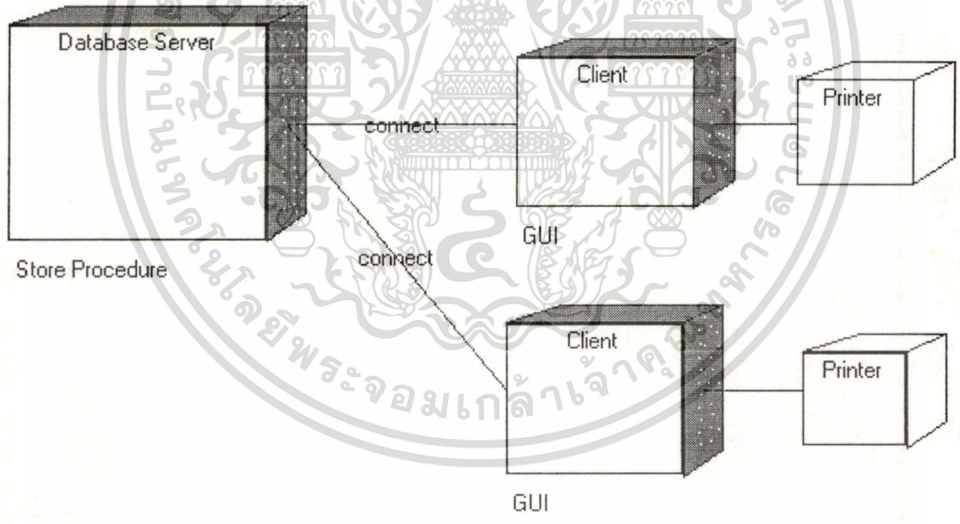
รูปที่ 3.27 แสดง Component Diagram ของระบบรับชำระ

จากรูปที่3.27 จะแสดงให้ Package (Code) ให้อยู่ในรูปสัญลักษณ์ Package (UML) ซึ่งจะหมายความว่าภายใน Package นั้นๆ จะประกอบไปด้วย Package Specification และ Package Body ของตน ซึ่ง Package Specification จะทำหน้าที่เป็น Interface ของ Package นั้นๆ

และจากรูปที่..จะเห็นได้อย่างชัดเจนว่า แต่ละ Package นั้นจะมีความเป็นอิสระระหว่างกัน นั่นคือการที่จะทำอะไรกับ Package ใดก็จะไม่มีผลกระทบกับ Package อื่นๆ (ยกเว้นว่าจะแก้ไข Interface ของ Package นั้น/ซึ่งเป็นปกติ) เช่น การแก้ไขในส่วนเนื้อ Code ของโปรแกรม (ส่วน Package Body) ก็จะไม่ทำให้เกิดปัญหาใดๆ กับ Component ที่เรียกใช้

3.8.2 Deployment Diagram

เป็นการแสดงเวลาขณะที่ซอฟต์แวร์กำลังทำงานอยู่ (Run-Time) กล่าวคือเป็นลักษณะการจำลองการทำงาน ณ เวลาที่เกิดขึ้นจริงของ Component และการทำงานที่จะเกิดขึ้นจริงของระบบรับชำระนี้จะแสดงได้ดังรูปที่ 3.28



รูปที่ 3. 28 แสดง Deployment Diagram ของระบบรับชำระ

จากรูปที่ 3.28 จะเห็นว่า Node จะมีข้อความอยู่ใต้ของ Node ซึ่งข้อความเหล่านั้นจะแทนด้วย Process (Component ที่กำลังทำงาน) ที่จะเกิดขึ้นภายใน Node นั้น และจากรูปนี้สามารถบอกได้ว่า Component ส่วนใดของระบบรับชำระจะมีการทำงานเกิดขึ้น ณ ที่ใด

บทที่ 4

การสร้างระบบ

ในช่วงการสร้างระบบจะเป็นการเขียนโปรแกรมจริงตามที่ได้มีการออกแบบไว้ และการออกแบบที่ได้มานั้นเป็นลักษณะการทำงานแยกกันหลายส่วน หรือเป็นที่รู้จักกันในสถาปัตยกรรม Client/Server ซึ่งโดยรวมแล้วที่ออกแบบไว้จะแบ่งเป็น 2 ส่วนใหญ่ คือส่วนที่เกี่ยวกับ User Interface และส่วนของ Business Layer (Application Logic) จะถูกพัฒนาไว้ในฝั่ง Client และส่วนที่เกี่ยวกับ Access Layer จะถูกพัฒนาไว้บนฝั่ง Server

เครื่องมือที่ใช้ในการพัฒนา

- Rational Rose 2000 (CASE Tool)
- Oracle Developer 6
- Oracle Server 8

4.1 การพัฒนาบนฝั่ง Server

จากการออกแบบ Access Layer จะเป็นกลุ่ม Class ที่ทำงานอยู่ในส่วนของฝั่ง Server ซึ่งจะเป็นงานที่เริ่มจากการสร้างระบบฐานข้อมูลของระบบรับชำระจาก Schema มาในออกแบบ และทำการสร้าง Store Procedure ไว้บนฝั่ง Server โดยจะมีวิธีการดังต่อไปนี้

4.1.1 การติดตั้ง DBMS

DBMS ที่ใช้เป็น Oracle Server เวอร์ชัน 8 ซึ่งก็เปรียบเสมือนเป็นการติดตั้งโปรแกรมแบบทั่วไปเหมือนกับโปรแกรมอื่นๆ (สามารถหาดูได้ตามคู่มือ DBMS ที่มีได้) หลังจากติดตั้ง DBMS เรียบร้อยแล้ว ก็ทำการสร้าง Database ให้กับระบบงาน ซึ่งคล้ายกับเป็นการกำหนดขนาดของ Database เพื่อให้สามารถรองรับส่วนต่างๆ ต่อไป (เช่น Table, Store Procedure เป็นต้น) ต่อจากการสร้าง Database จะเป็นการสร้าง User ของที่ใช้งานกับ Database (พร้อมกับการกำหนดสิทธิที่เหมาะสม) เพื่อเป็น User ของระบบงาน

จากนั้นจะเป็นการกำหนดทางเชื่อมต่อ (หรือที่เรียกว่า Host String) เพื่อให้ผู้พัฒนาเข้าถึง Database ได้ โดยใช้เครื่องมือที่มีมาให้คือ Net Easy Config ช่วยในการกำหนด และขณะนี้ Database ในฝั่ง Server พร้อมรับการบริการ แต่ยังไม่มีโครงสร้างฐานข้อมูลของระบบอยู่

4.1.2 การสร้าง Table ลงใน Database เพื่อเก็บข้อมูลของระบบ

การสร้าง Table ของระบบจะอาศัย Rational Rose 2000 เป็นเครื่องมือช่วยในการสร้างฐานข้อมูล ซึ่งปกติจะต้องมาสร้างฐานข้อมูลขึ้นมาด้วยตัวเอง ซึ่งเครื่องมือนี้จะอาศัยแบบจำลองที่มีการออกแบบไว้เป็นข้อมูลในการสร้าง Table บนฝั่ง Server หรือเรียกว่า Forward Engineering ซึ่งมีวิธีการดังนี้

4.1.2.1 การกำหนด Model สำหรับการสร้าง Table

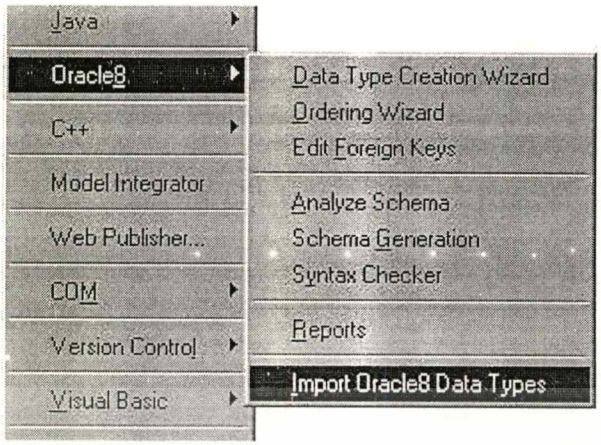
เนื่องจากในการพัฒนาครั้งนี้มีการ Implement บนระบบฐานข้อมูลที่เป็นแบบ Relational จึงต้องมีการ Map จาก Class ที่ได้การออกแบบซึ่งเป็นรูปแบบของ Object ไปเป็น Class ที่สนับสนุนการสร้าง Schema แบบ Relational บน Database จะเป็นลักษณะ Class ที่ไม่มี Method และจะต้องมีการกำหนดลักษณะ Class ที่ต้องการสร้าง Schema ให้เหมาะสมในการสร้าง

การ Map มี 2 วิธีคือ ใช้ Wizard (Data Type Creation Wizard) ช่วย และการสร้าง Class ขึ้นมาใหม่ด้วยตัวเอง และการ Map ในที่นี้จะอาศัย Wizard เป็นเครื่องมือช่วยในการ Map เพื่อความสะดวกและรวดเร็ว รวมถึงการเก็บข้อมูลการกำหนดใน Class Diagram ได้ดีกว่า ส่วนวิธีการ Map ใน Software สามารถทำได้หลายรูปแบบมากมาย ในส่วนที่นำเสนอจะเป็นเพียงรูปแบบเดียวเท่านั้น

การใช้ Wizard ใน Rational Rose ช่วยในการ Map Wizard จะทำการ Map จาก Class ที่มี Stereotype เป็น Object Types หรือ Relational Tables เท่านั้น

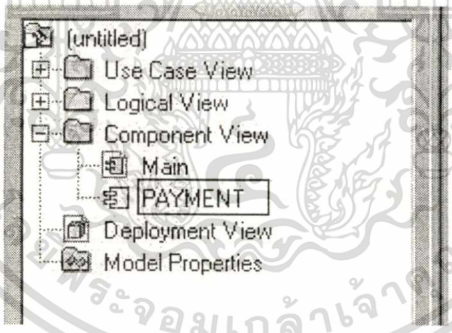
ก่อนที่จะเริ่มต้นนั้นควรจะกำหนด Class Diagram ให้มีความละเอียดเพียงพอเสียก่อน เพราะหากทำการ Map ทาง Wizard จะทำการ Map คุณสมบัติที่กำหนดไว้ใน Class Diagram มาด้วย

1. เริ่มต้นจากการ Map Class โดยนำ Class Diagram ของขั้นการออกแบบมาเป็นจุดเริ่มต้น หากกรณีที่ใช้ Framework ของ Oracle8 อยู่แล้วให้ข้ามขั้นนี้ไป แต่กรณีที่ไม่ได้ใช้ Framework ของ Oracle8 ต้องมีการ Import ชนิดข้อมูลของ Oracle8 ที่ Rational Rose รู้จักก่อน ซึ่งจะแสดงดังรูปที่ 4.1



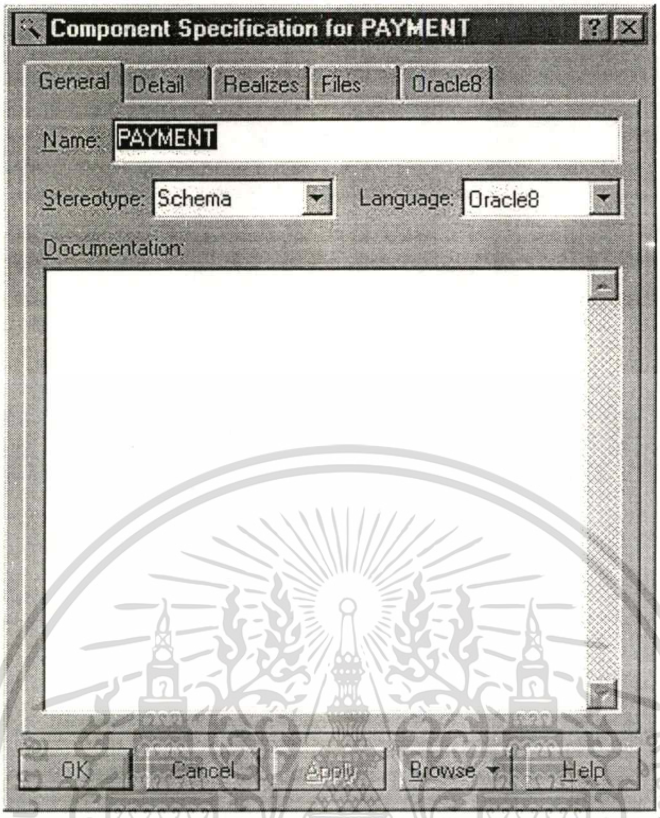
รูปที่ 4.1 แสดงการ Import Oracle8 Data Type

2. ขั้นต่อมาสร้าง Component พร้อมกับตั้งชื่อ แสดงไว้ดังรูปที่ 4.2 และชื่อของ Component จะใช้แทน Owner ของ Table ในการสร้าง Schema เพราะโดยปกติการสร้าง Table ใน Oracle นั้นจะสร้างตามสิทธิของ User และเมื่อสร้าง Table ได้ User ที่เป็นผู้สร้างจะถือเป็น Owner



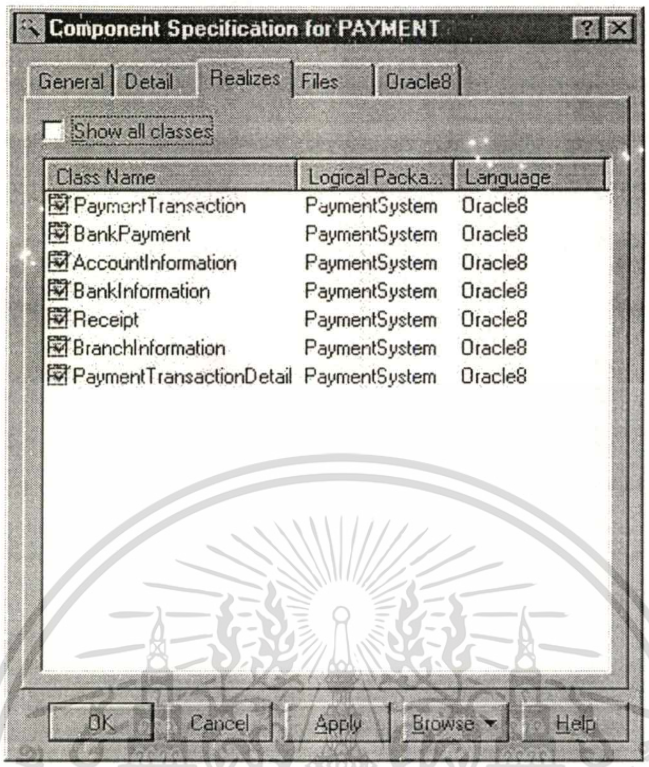
รูปที่ 4.2แสดงการสร้าง Component พร้อมกับการกำหนดชื่อ

3. กำหนดคุณสมบัติของ Component (Component Specification) โดยให้ Language เป็น Oracle8 (แล้ว Apply) หลังจากการ Apply กำหนด Stereotype ให้เป็น Schema ซึ่งจะแสดงดังรูปที่ 4.3



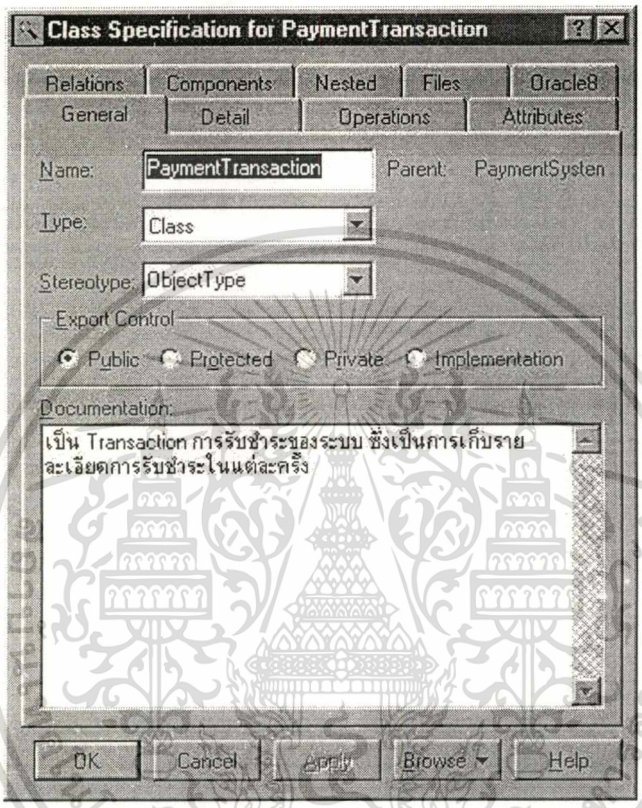
รูปที่ 4.3 แสดงการกำหนดคุณสมบัติของ Component (Component Specification)

4. หลังจากนั้นเลือก Tab Sheet (ยังอยู่ใน Component Specification) Realizes แล้วเลือก Class ที่ต้องการให้ Map โดยการ Assign เมื่อ Assign ได้จะขึ้นเครื่องหมายถูกสีแดงทับ Class ที่ Assign ได้ ซึ่งแสดงดังรูปที่ 4.4 หลังจากนั้นตอบ OK เพื่อยืนยันการกระทำ ซึ่งการทำลักษณะนี้หมายความว่า ให้ Class เหล่านั้นเป็นส่วนหนึ่งของ Component หรือกล่าวอย่างว่า ให้ Class เหล่านั้นเป็น Schema ของ Oracle



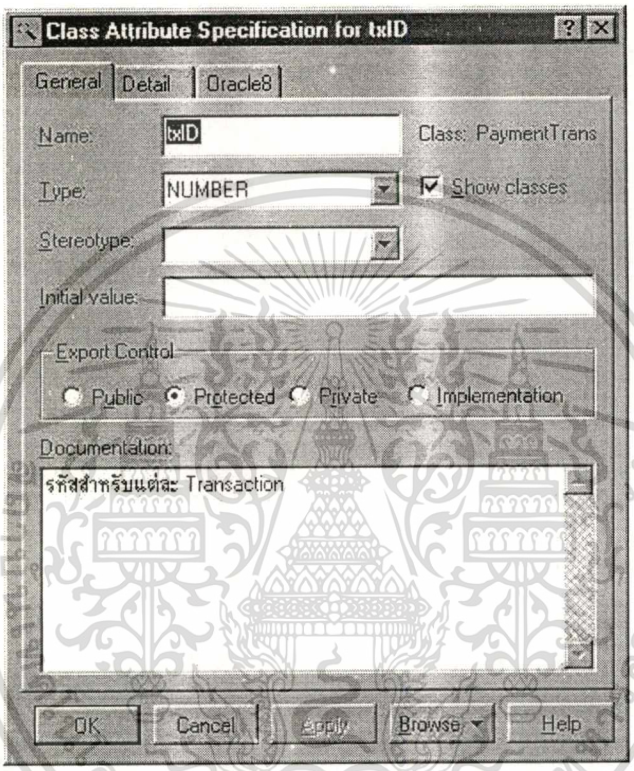
รูปที่ 4.4 แสดงการ Assign Class ที่ต้องการจะ Map

5. จากนั้นมาที่ Class Specification ของ Class ที่ถูก Map สังเกตว่ามี Tab Sheet ชื่อ Oracle8ปรากฏเพิ่มขึ้นมา กำหนด Stereotype เป็น ObjectType ดังรูปที่ 4.5 ซึ่งขั้นตอนนี้หมายความว่าให้ Class นี้มีชนิดเป็น ObjectType



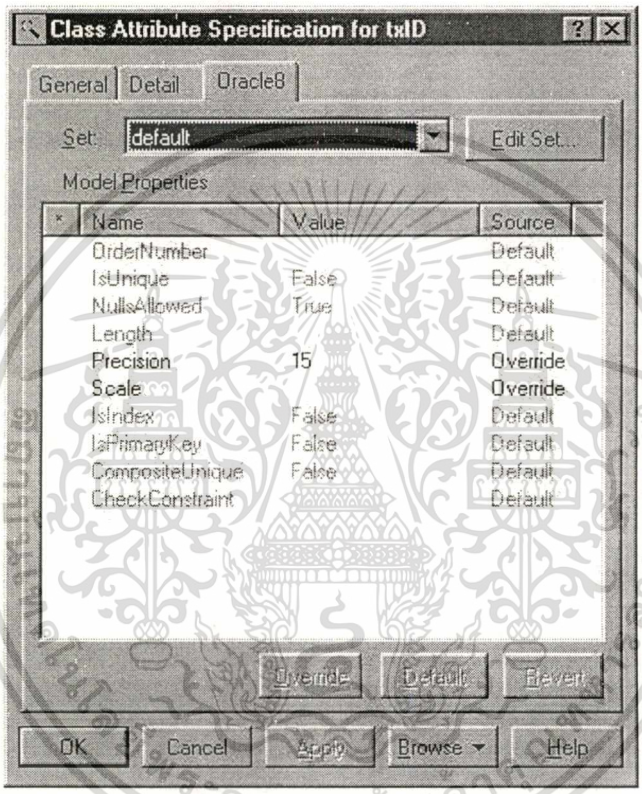
รูปที่ 4.5 แสดงการกำหนด Stereotype ของ Class เป็น ObjectType

6. เปลี่ยนมาที่ Tab Sheet ชื่อ Attributes แล้วเลือก Attribute และกำหนด Attribute Specification เพื่อกำหนดชนิดของข้อมูลของ Attribute ดังแสดงในรูปที่ 4.6 ซึ่งต้องเป็นข้อมูลประเภทที่ Oracle รู้จักด้วย (Scalar Type) และกำหนด Attribute Specification ของ Attribute ทุกตัว



รูปที่ 4.6 แสดงการกำหนดชนิดข้อมูลของ Attribute

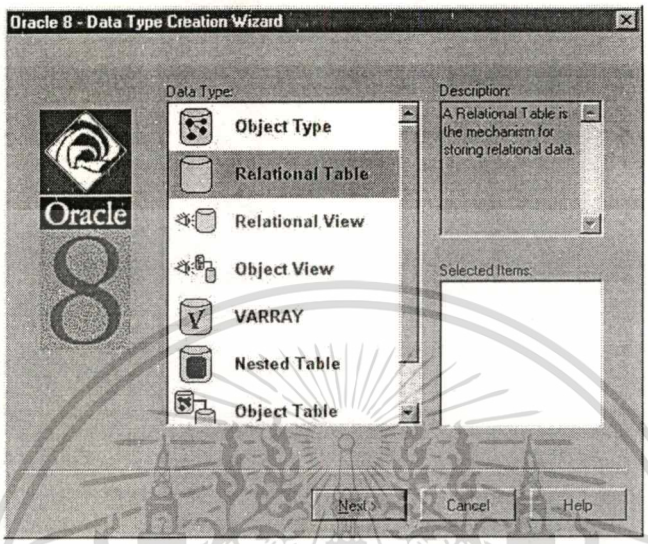
7. เนื่องจากข้อมูลบ้างชนิดของ Oracle นั้นต้องมีการกำหนดขนาดของข้อมูลด้วย เช่น CHAR(30) หรือ NUMBER(11,2) จึงต้องมีการกำหนดเพิ่มเติมที่ Tab Sheet Oracle8 (ของ Attibute Specification) ในช่อง Length สำหรับ Attribute ประเภท CHAR (ตระกูล CHAR) และช่อง Precision และช่อง Scale สำหรับ Attribute ประเภท NUMBER (ตระกูล NUMBER) ซึ่งแสดง Tab Sheet นี้ไว้ดังรูปที่ 4.7



รูปที่ 4. 7 แสดง Tab Sheet Oracle8 ของ Attribute Specification

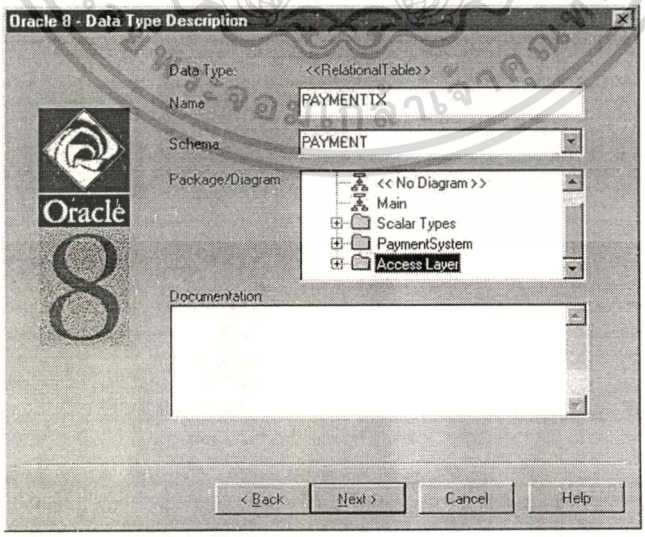
8. จากนั้นทำซ้ำขั้นตอน 5 จนถึงขั้นตอนที่ 7 เพื่อเป็นการกำหนด Class Specification ที่สมบูรณ์ และขณะนี้เป็นเพียงการกำหนด Class Diagram ที่จะนำมา Map เท่านั้น

9. หลังจากนั้นใช้ Wizard ของ Rational Rose 2000 ช่วยในการสร้าง Class เหล่านั้นขึ้นมาใหม่โดยจะแสดงดังรูปที่ 4.8



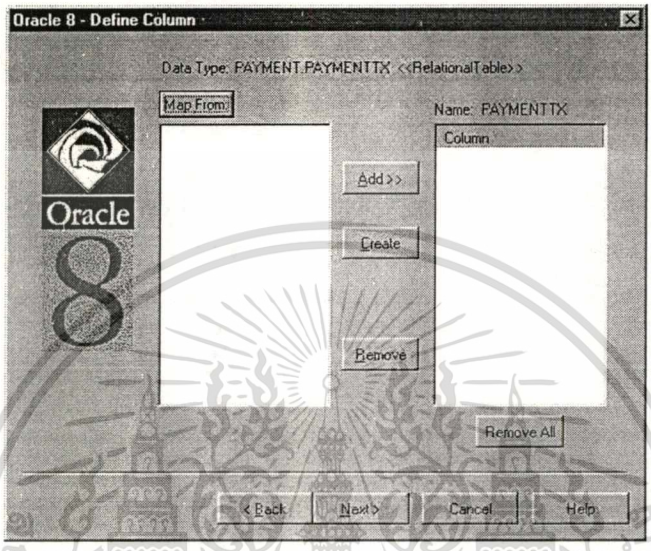
รูปที่ 4. 8 แสดง Wizard ที่ 1 ในการ Map (Data Type Creation Wizard)

10. เลือก Data Type เป็น Relational Table แล้ว Next ซึ่งจะแสดงดังรูปที่ 4.9 แล้วใส่ชื่อของ Class ที่จะสร้างขึ้นมาใหม่ พร้อมกับเลือก Schema ที่มีให้ในช่อง Schema และระบุตำแหน่งที่ให้สร้าง Class ใหม่นี้ในที่ใดในช่อง Package/Diagram

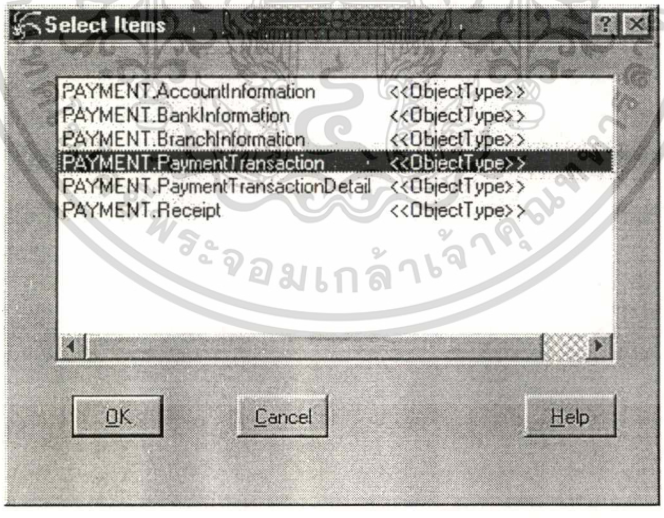


รูปที่ 4. 9 แสดง Wizard ที่ 2 ในการ Map (Data Type Description)

11. และหน้าจอต่อไปจะเป็นการกำหนด Attribute ให้แก่ Class ที่จะสร้างใหม่ ซึ่งสามารถทำได้ทั้งสร้างใหม่เองหรือทำการ Map มาจากสิ่งที่มีอยู่แล้ว จะแสดงดังรูปที่ 4.10 และในที่นี้จะเป็นการสร้างจากสิ่งที่มีอยู่แล้วคือ Map Form และจะแสดงดังรูปที่ 4.11

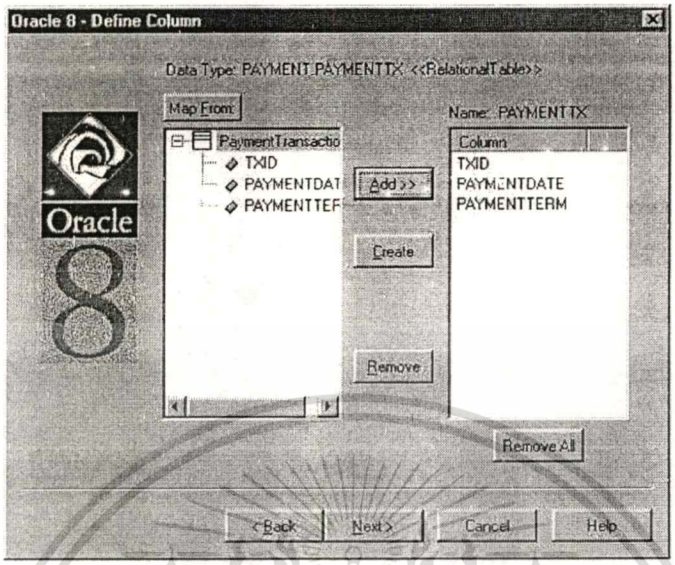


รูปที่ 4.10 แสดง Wizard ที่ 3 ในการ Map (การกำหนด Column)



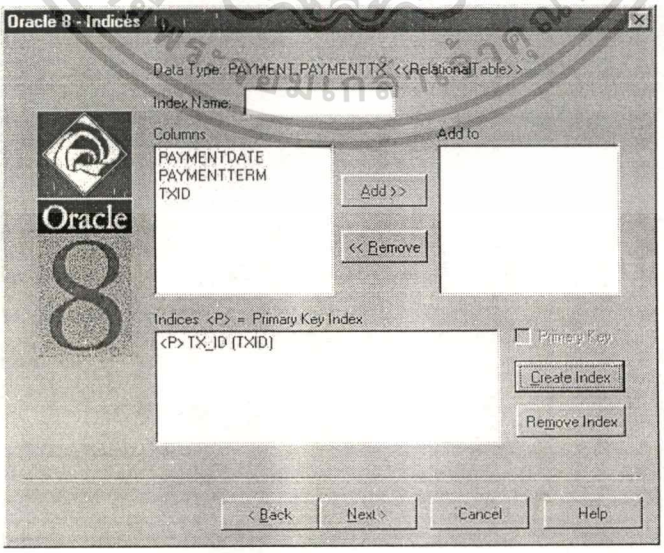
รูปที่ 4.11 แสดงการเลือก Class เพื่อทำการนำ Attribute ของ Class ที่เลือกมาสร้างเป็น Class ใหม่

ควรจะเลือก Class ที่ในอนาคตไม่มีการกำหนด Foreign Key เพราะ Class เหล่านั้นจะมี Primary Key ที่ทำหน้าที่เป็น Foreign Key ของ Class อื่นๆ เมื่อเลือกเสร็จแล้วจะมี Class ปรากฏขึ้นในช่อง Map Form และทำการ Add Attribute ที่ต้องการ ซึ่งจะปรากฏดังรูปที่ 4.12



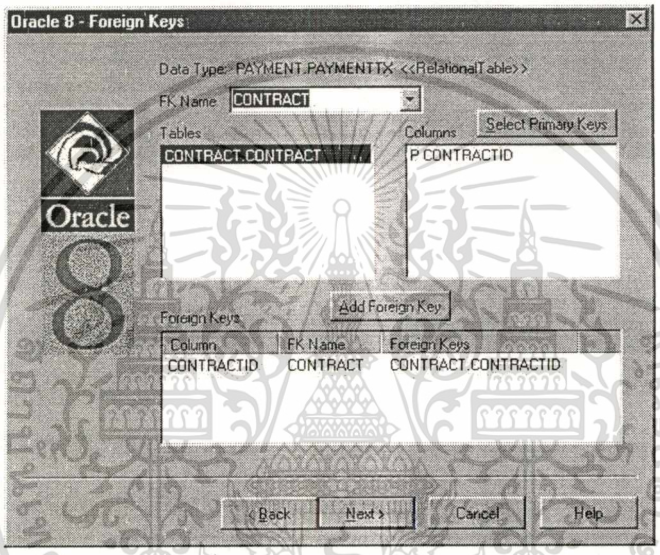
รูปที่ 4.12 แสดงการเลือก Attribute ที่ต้องการจะสร้าง

12. ขั้นต่อไปจะเป็นการกำหนด Index ที่ต้องการซึ่งสามารถสร้างได้หลายตัว และอาจกำหนดให้ Index ทำหน้าที่เป็น Primary Key จะแสดงไว้ดังรูปที่ 4.13 ซึ่งชื่อของ Index ที่สร้างขึ้นมา ในการสร้างบน Oracle DBMS จะทำหน้าเป็น Constraint ของการกำหนด Column ดังนั้นการตั้งชื่อของ Index ในขั้นตอนนี้จะต้องไม่ซ้ำกันเพราะ Constraint จะต้อง Unique



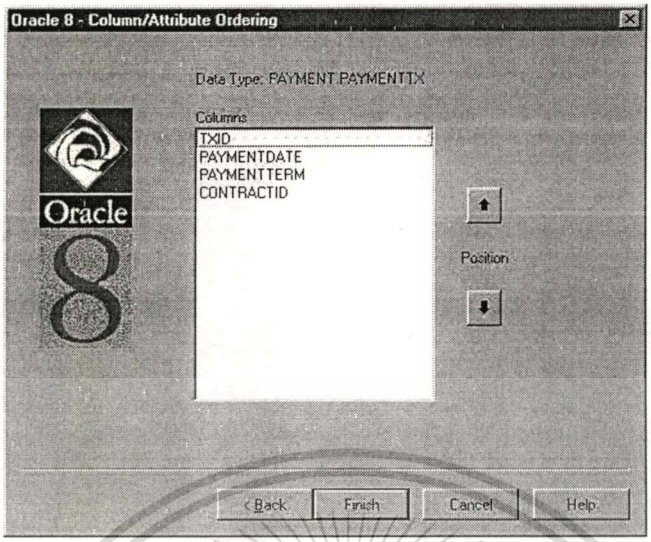
รูปที่ 4.13 แสดง Wizard ที่ 4 ของการ Map (การกำหนด Indices / Primary Key)

13. ขั้นต่อไปเป็นการกำหนด Foreign Key ให้กับ Class ที่จะสร้าง (อาจจะมีการสร้าง Foreign Key ในภายหลังจาก Wizard ได้) โดยการกำหนดชื่อ Foreign Key ในชื่อ FK Name และเลือก Class ที่ต้องการอ้างถึง เมื่อเลือกแล้วจะมีชื่อของ Column ปรากฏอยู่ในช่อง Column ซึ่งอาจมีตัวอักษร P และ U นำหน้าชื่อ Column ซึ่ง P แทนด้วย Primary Key ของ Class ที่เลือกและ U แทนด้วย Column ที่ Unique ซึ่งจะแสดงดังรูปที่ 4.14



รูปที่ 4.14 แสดง Wizard ที่ 5 ของการ Map (การกำหนด Foreign Key)

14. ขั้นต่อไปเป็นการเรียงลำดับของ Attribute ของ Class ที่สร้างใหม่ และเป็นขั้นตอนสุดท้ายในการ Map ซึ่งแสดงไว้ดังรูปที่ 4.15 จะสังเกตเห็นว่ามี Attribute ที่ทำหน้าที่เป็น Foreign Key ปรากฏขึ้นมาด้วย

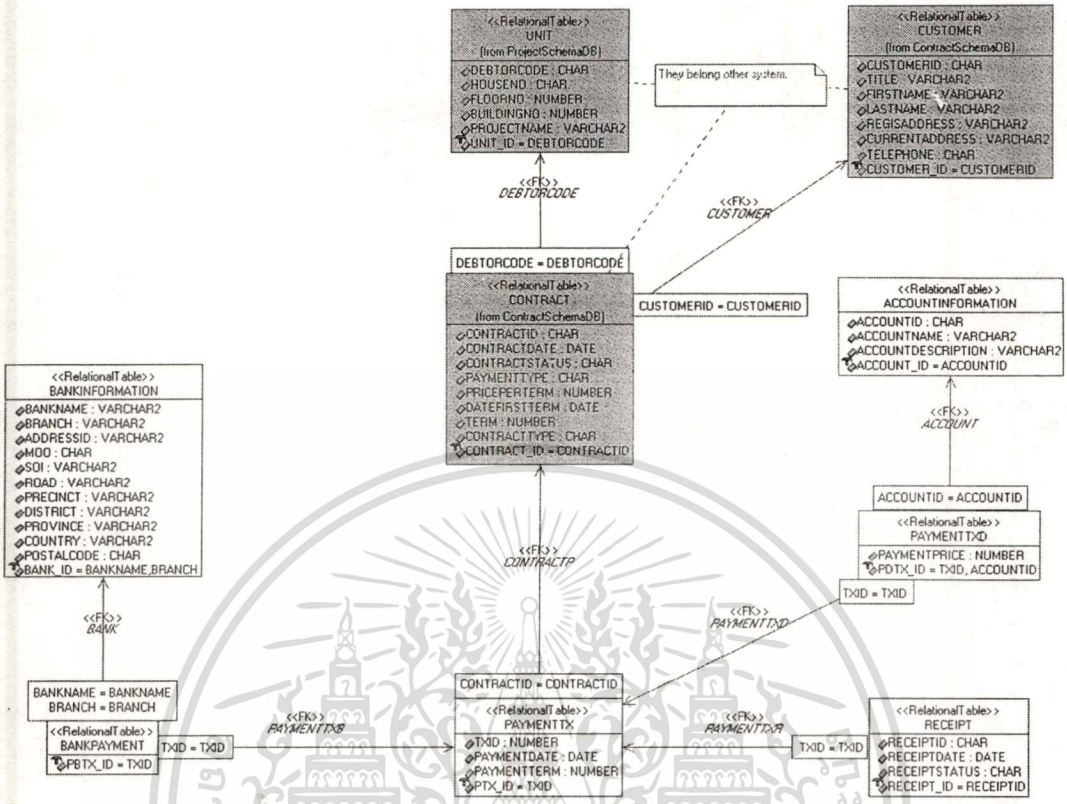


รูปที่ 4.15 แสดง Wizard ที่ 6 ของการ Map (การเรียงลำดับ Attribute)

15. ทำซ้ำจนกว่าจะได้โครงสร้างของ Class ที่ต้องการ

เนื่องจากมีการใช้ระบบฐานข้อมูลเป็นแบบ Relational จึงมีความจำเป็นในการปฏิบัติตามหลักเกณฑ์และทำให้ได้ประสิทธิภาพสูงสุดในสภาพแวดล้อมที่ทำงานอยู่ และสิ่งหนึ่งที่ควรปฏิบัติคือการ Normalization กับโครงสร้างของ Table ในระบบเพื่อมิให้เกิดปัญหาความซ้ำซ้อนของข้อมูล

หลังจากผลที่ได้นั้นมีความจำเป็นอย่างมากที่จะตรวจสอบว่าโครงสร้าง Class ที่ได้นั้น เกิดปัญหาเรื่องการซ้ำซ้อนของข้อมูลหรือไม่ หากยังก็จัดการทำ Normalize ซึ่งอาจเป็นการเพิ่มหรือลด Class ที่ Map ได้และ/หรือ Attribute ที่ Map ได้เช่นกัน สำหรับโครงสร้าง Class ที่ Map ได้ในโครงการนี้นั้น ไม่มีปัญหาเรื่องความซ้ำซ้อนของข้อมูลเกิดขึ้น ซึ่งแสดงไว้ดังรูปที่ 4.16



รูปที่ 4.16 แสดงโครงสร้าง Class ใช้เป็น Schema แบบ Relational ของระบบรับชำระ

จากรูปที่ 4.16 นี้สัญลักษณ์ Foreign Key ที่ใช้ใน Rational Rose จะใช้สัญลักษณ์ที่เรียกว่า Qualifier ของ Class Diagram ซึ่งแสดงเป็น Attribute ที่มีความสัมพันธ์กัน ชื่อความสัมพันธ์จะใช้ชื่อของ Foreign Key ที่กำหนด (เป็น Constraint นั้นเอง) พร้อมกับ Stereotype เป็น FK

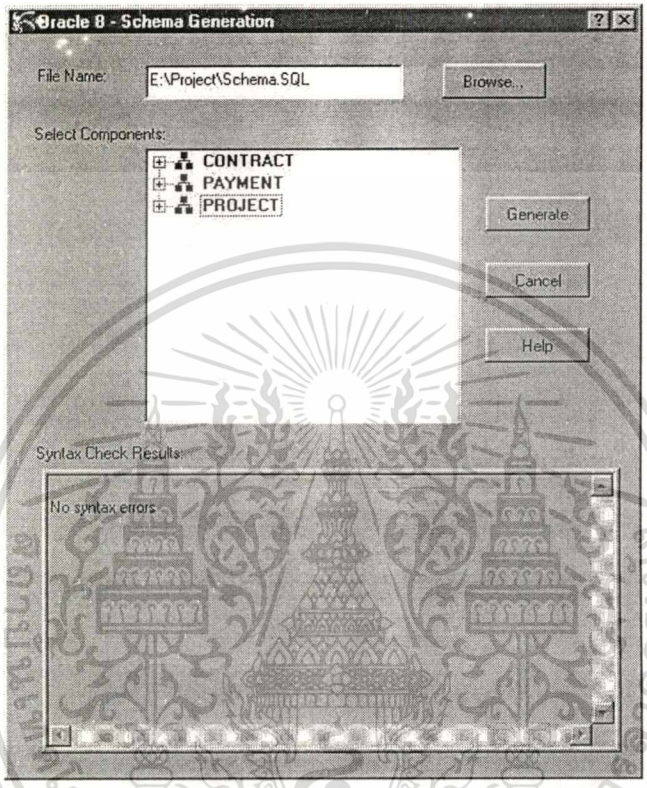
ในกรณีหากมีการสร้าง Schema ที่เป็นแบบ Object-Orient ในขั้นตอนนี้จะเป็นกำหนด Model ให้เหมาะสมกับการสร้างเท่านั้น

สรุปเงื่อนไขที่สำคัญในการสร้าง Table ของ Rational Rose 2000

1. Rational Rose จะอ่านเฉพาะ Class ที่ถูก Assign ให้เป็นส่วนหนึ่งของ Component ที่ใช้ภาษา Oracle8 และ Stereotype ของ Component เป็น Schema เท่านั้น
2. ในการ Map ด้วย Wizard ต้องกำหนด Stereotype ของ Class ให้เป็นชนิด Object Types หรือ Relational Tables เท่านั้น
3. ควรจะสร้าง Class ที่ไม่มี Foreign Key ก่อนเพราะ Class เหล่านั้นมี Primary Key ที่ทำหน้าที่เป็น Foreign Key ของ Class อื่น

4.1.2.2 การสร้าง Table จาก Model ที่กำหนด

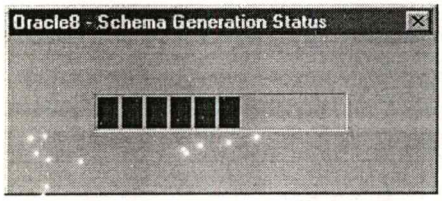
หลังจากการกำหนด Model เสร็จเรียบร้อยแล้ว จึงให้ Rational Rose เริ่มต้นสร้าง Table ซึ่งจะแสดงดังรูปที่ 4.17



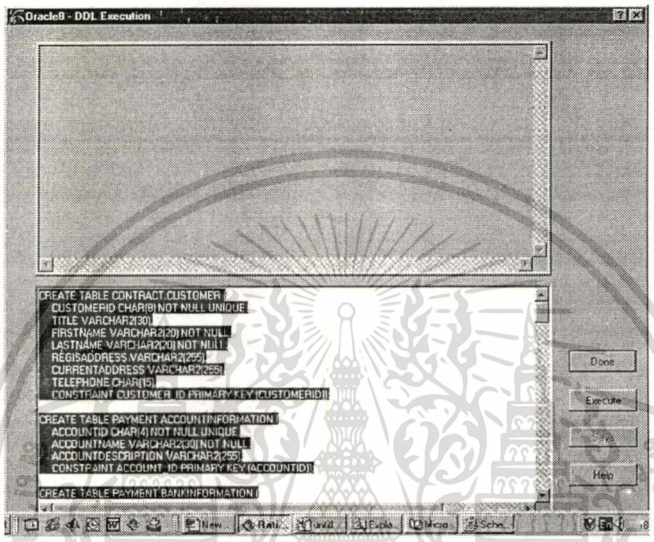
รูปที่ 4.17แสดงหน้าจอเริ่มต้นของการทำ Forward Engineering ของ Rational Rose

จากรูปที่ 4.17 ช่อง “File Name” จะเป็นช่องสำหรับใส่ที่อยู่และชื่อของ File ที่เก็บ Script DDL (Data Definition Language) ส่วนช่อง “Select Component” เป็นช่องที่ให้เลือก Component (ซึ่งหมายถึง Class ที่เป็นประเภท Object หรือ Relation ก็ตาม) ที่ต้องการจะสร้าง Script ในไฟล์ที่ระบุไว้ และส่วน “Syntax Check Results” เป็นส่วนที่แสดงผลของ Component ใดที่กำหนดผิดพลาด ที่ไม่สามารถจะสร้าง Script ได้ ซึ่งรวมถึงข้อความการเตือนต่างๆ สำหรับ Component

หลังจากการกำหนดสิ่งต่างที่ต้องการในการสร้าง Script แล้ว ก็เริ่มสร้าง Script DDL ได้ที่ปุ่ม Generate ผลที่ได้จะแสดงดังรูปที่ 4.18 ซึ่งจะแสดงระดับความก้าวหน้าของการสร้าง Script DDL และรูปที่ 4.19 จะแสดงผลลัพธ์ของการสร้าง Script DDL

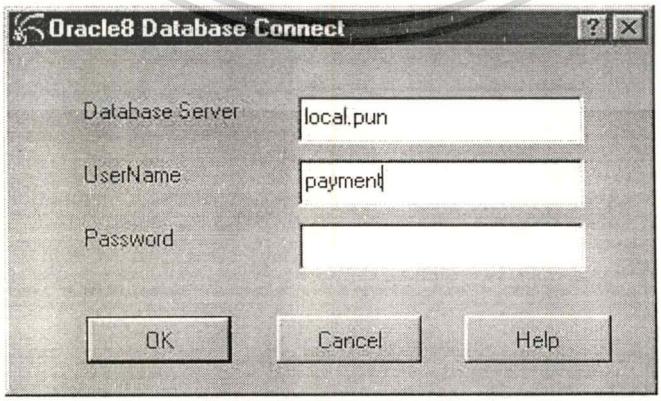


รูปที่ 4.18 แสดงระดับความก้าวหน้าของการสร้าง Script DDL



รูปที่ 4.19 แสดงผลลัพธ์ของการสร้าง Script DDL

เมื่อได้ผลลัพธ์ของการสร้าง Script นี้ แล้วต้องการสร้าง Table ลงบน DBMS ในฝั่ง Server ก็เริ่มสร้างได้ที่ปุ่ม Execute ซึ่งจะแสดงดังรูปที่ 4.20 หรือหากยังไม่สร้าง Table ในทันที ก็อาจจะนำ Script File ที่ได้นี้ไป Run บน DBMS ภายหลังได้



รูปที่ 4.20 แสดงการ Login เพื่อต่อเข้ากับ DBMS

จากรูปที่ 4.20 ก็ใส่ Host String, User Name และ Password ลงไป ซึ่งเป็นการ Login เข้าสู่ DBMS เพื่อนำ Script File ไปทำงานบน DBMS (เพื่อสร้าง Table บน DBMS)

สุดท้ายผลลัพธ์ที่ได้ในขั้นนี้คือมี Table เปล่าๆ ที่ต้องการอยู่ใน Database และพร้อมจะเก็บข้อมูลของระบบได้

4.1.3 การสร้าง Store Procedure ใน Database

เป็นการสร้าง Store Procedure จาก Class ในกลุ่ม Access Layer โดยจะสร้างเป็น Package 1 Package ต่อ Class 1 Class ใน Access Layer หรือสามารถจะนำ Component Diagram ในขั้นตอนการออกแบบมาเป็น Diagram สำหรับการอ้างอิงได้เช่นกัน

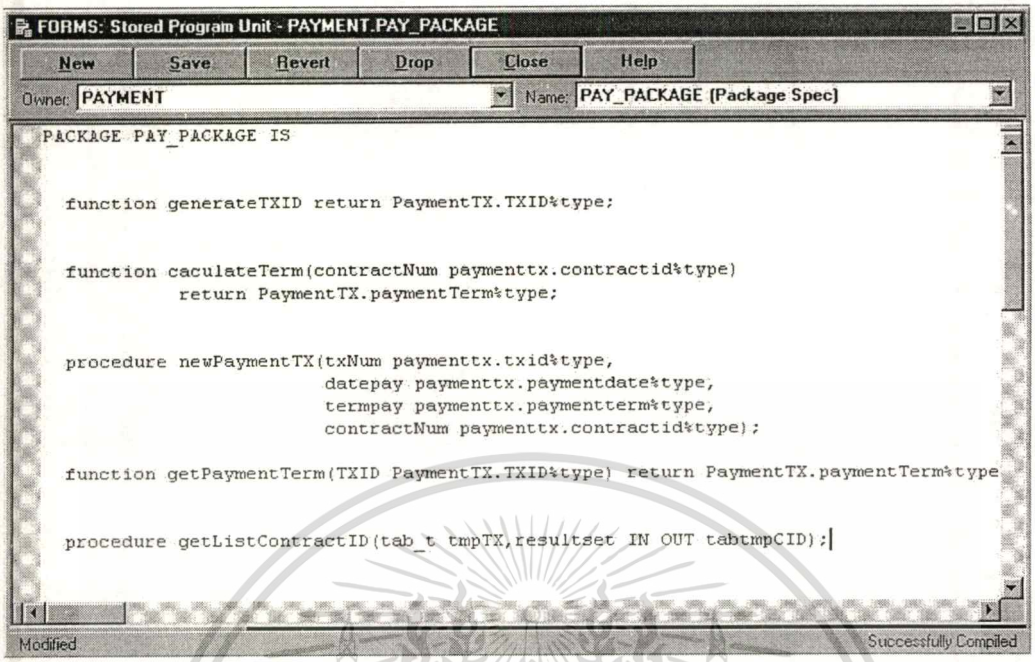
การสร้าง Store Procedure ของระบบรับชำระจะอาศัย Oracle Developer 6 ช่วยในการสร้าง โดยจะให้ Developer 6 ทำการติดต่อไปยัง Server และสร้าง Store Procedure บนฝั่ง Server ผ่านทาง Developer 6 และใช้ภาษา PL/SQL สร้าง Store Procedure ของระบบ

Package ของ Oracle นั้นจะถูกแบ่งเป็น 2 ส่วนคือส่วนที่เป็น Program Specification และ Program Body ซึ่ง Program Specification จะเปรียบเสมือนเป็น Interface สำหรับใช้งานและภายในจะมีเพียง ชื่อ Function และ/หรือ Procedure และ Parameter ต่างๆ พร้อมกับ ค่าที่ต้องการจะส่งกลับ และส่วน Program Body จะเป็นรายละเอียด Code โปรแกรม ที่มี Function และ/หรือ Procedure ตรงกับที่ประกาศไว้ในส่วนของ Program Specification และอาจมี Function / Procedure เพิ่มภายใน Program Body ได้ แต่จะ Interface ผ่านตรงๆ ไม่ได้

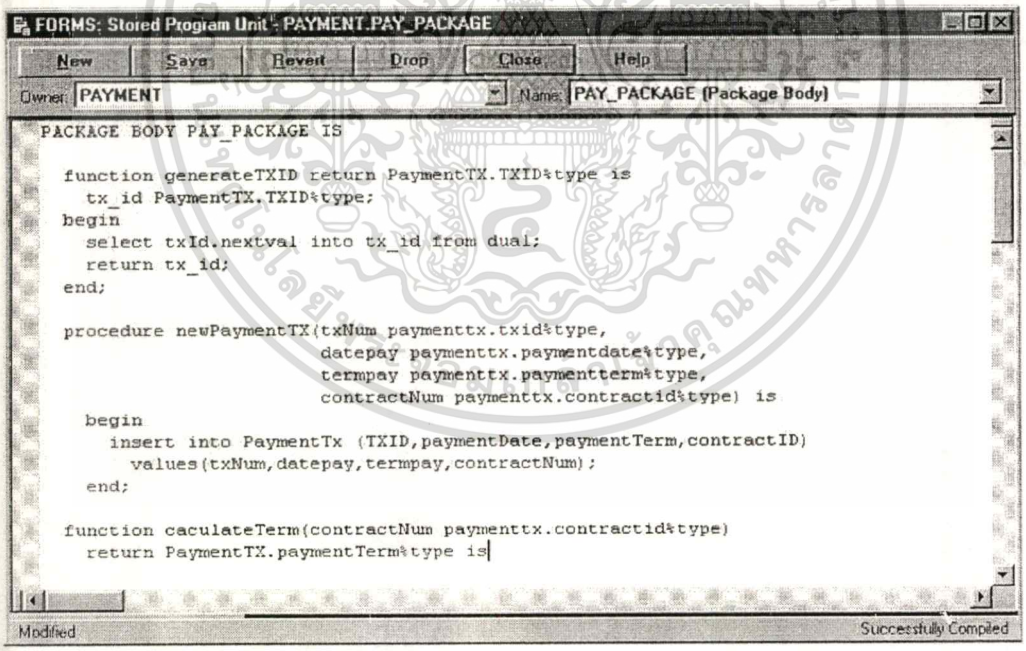
Program Specification ใน Package นั้นเปรียบเป็นการประกาศ Visibility เป็น Public ของ Object-Oriented และส่วน Program Body ใน Package ก็เปรียบเป็นการประกาศ Visibility เป็น Private ใน Object-Oriented

Program Specification ใน Package มีรายละเอียดเพิ่มขึ้นจาก Public ของ Object-Oriented คืออาจจะกำหนดการเข้าถึง (การใช้งาน Package) นี้เป็นราย User บน Oracle ก็ได้ หรือจะประกาศเป็น Public ที่เหมือนกับ Oracle เลยก็ย่อมได้

ตัวอย่างของ Package ซึ่งจะแสดง Program Specification ไว้ในรูปที่ 4.22 และแสดง Program Body ไว้ในรูปที่ 4.23 โดยจะพัฒนาผ่านทาง Developer 6 แต่จะถูก Compile โดย DBMS



รูปที่ 4. 21 แสดงตัวอย่าง Program Specification ของระบบรับชำระ



รูปที่ 4. 22 แสดงตัวอย่าง Program Body ของระบบรับชำระ

4.2 การพัฒนาบนฝั่ง Client

การพัฒนาบนฝั่ง Client นั้นเป็นการสร้าง UI (User Interface) โดยนำ UI ที่ได้จากขั้นการออกแบบมาพัฒนาต่อจนเสร็จสมบูรณ์ ซึ่งเป็นการสร้าง UI บนระบบปฏิบัติการ MS Windows ทำให้สามารถแสดง UI เป็นแบบรูปภาพได้ (GUI : Graphic User Interface) และจะใช้ Oracle Developer 6 ช่วยในการพัฒนา GUI นี้ขึ้นมา

ภายใน Oracle Developer 6 มีเครื่องมือที่ช่วยอำนวยความสะดวกในการสร้าง UI หลายตัว โดยหลักๆ ก็เป็นเครื่องมือในการสร้าง Form, Report, Query, Graphic, Procedure, Project, Schema และ Translation ซึ่งที่จำเป็นอย่างยิ่งในการใช้พัฒนาครั้งนี้คือเครื่องมือในการสร้าง Form และเครื่องมือในการสร้าง Report

4.2.1 การสร้างส่วนรับข้อมูลและแสดงผล (Form)

ในเครื่องการสร้าง Form จะมีองค์ประกอบหลักๆ ในการสร้าง Form ดังนี้คือ

- Window เป็นหน้าต่างในการแสดงผล
- Canvas เป็นส่วนที่เปรียบเสมือนพื้นที่สำหรับวางและจัดเรียง Component ต่างๆ
- Data Block ซึ่งเป็นส่วนในการเก็บ Component ต่างๆ ในการรับและแสดงผล เช่น Text Box, Command Button หรือ Check Box เป็นต้น รวมถึงอาจใช้ในการรวมกลุ่มของ Component ที่มีหน้าที่คล้ายๆ กัน
- Program Unit เป็นส่วนในการเก็บ Code โปรแกรมที่ใช้เฉพาะใน Form นั้นๆ อาจเป็น Package, Procedure หรืออาจเป็น Function เป็นต้น
- Trigger เป็นเหมือนเหตุการณ์ต่างๆ ที่จะเกิดขึ้น เช่น When-Button-Pressed จะเกิดขึ้นเมื่อปุ่มถูกกด เป็นต้น หรือที่เรียกว่า Event ในเครื่องมืออื่นๆ

ในส่วนของการใช้งานจะต้องมีการใช้ Canvas ร่วมกับ Window จึงนับว่าเป็น Form หนึ่ง เพราะ Window เป็นเหมือนกับหน้าต่างเปล่าๆ แล้วนำ Canvas ที่เปรียบเหมือนผ้าใบมาขึงบนหน้าต่าง ดังนั้นจะมีลักษณะคล้ายกับ Form หนึ่ง Form

การพัฒนาจะเริ่มต้นจากการนำ UI แต่ละอันที่ได้จากขั้นการออกแบบมาพัฒนาต่อจนกระทั่งมีหน้าที่ครบสมบูรณ์ตรงตามความต้องการของผู้ใช้ โดยอาศัย Algorithm ของแต่ละ Method ที่ออกแบบมา

4.2.2 การสร้างส่วนรายงาน (Report)

ในเครื่องการสร้าง Report จะมีองค์ประกอบหลักๆ ในการสร้าง Report ดังนี้คือ

- Data Model เป็นส่วนที่ใช้ในการกำหนดรูปแบบชุดของข้อมูลสำหรับสร้างรายงาน หรือเปรียบเป็นที่ในการสืบค้นข้อมูลมาสำหรับการทำรายงาน

- Layer Model เป็นส่วนที่ใช้ในการจัดรูปแบบของรายงานที่ต้องการ โดยจะแสดงข้อมูลที่ Data Model กำหนด

จะสร้างรายงานโดยใช้รูปแบบของรายงานที่แสดงอยู่ในภาคผนวกของเอกสาร

4.2.3 การรวมส่วนต่างๆ ที่พัฒนาเข้าด้วยกัน (Integration)

การรวมส่วนต่างๆ ที่พัฒนาเข้าด้วยกัน (Integration) จะเป็นการเชื่อมโยงสิ่งที่พัฒนาเข้าด้วยกัน คือทำให้ทุกสิ่งที่พัฒนามีทางเข้าถึงได้เพียงทางเดียวและทำให้เป็น Application ที่สามารถใช้งานได้ตรงตามความต้องการของผู้ใช้ และในที่นี้คือการเชื่อมประสานกันระหว่าง Form และรายงานที่พัฒนาขึ้นมาให้มีการใช้รายงานโดยผ่าน Form ที่ออกแบบสำหรับเลือกพิมพ์รายงาน

หลังจากนั้น เมื่อมีการเข้าถึงใน Form หรือรายงานต่างๆ จากทางเข้าเพียงทางเดียวแล้ว ต่อไปจะเป็นการคล้ายกับการ Compile เพื่อทำให้เป็น Application อย่างสมบูรณ์ ซึ่งอาจจะใช้เครื่องมือในการสร้าง Project เข้ามาช่วยทำในส่วนนี้ก็ได้

บทที่ 5

สรุป

5.1 ปัญหาต่างๆ ที่เกิดในการพัฒนาระบบ

เนื่องจากการพัฒนาระบบเชิงวัตถุนี้เป็นการพัฒนาแบบ Iterative คือเป็นลักษณะของการทำซ้ำ รวมถึงการแก้ไขปรับปรุงให้ดีขึ้น ทำให้ต้องมีการแก้ไขรายละเอียดต่างๆ อยู่มากและตลอดเวลา ทำให้มีความต้องการสำหรับนำการเครื่องมือเข้ามาช่วยในการพัฒนา

ในส่วนของการสร้างระบบครั้งนี้นั้นปัญหาใหญ่ๆ ที่พบอย่างชัดเจนคือเรื่องของการพัฒนาบนระบบฐานข้อมูลแบบ Relational หากยังคงรักษาทฤษฎีของ Object-Orient แล้วจะใช้ประสิทธิภาพของภาษา SQL ได้ไม่เต็มประสิทธิภาพ เช่นในเรื่องของการ Join จะไม่ได้ใช้เลยเพราะการ Join เป็นวิธีการเข้าถึงข้อมูลโดยตรงของอีก Table หนึ่งทันที หรือแม้แต่ Sub-Query ก็ไม่ได้ใช้เช่นกัน และยังรวมถึงไม่ได้ใช้ประสิทธิภาพ DBMS ของ Oracle ในส่วนของ Query Optimizer เลย

5.2 เปรียบเทียบกับการพัฒนาระบบในแนวทางเดิม

การพัฒนาระบบเชิงวัตถุมีการแสดงระบบที่พัฒนาในระดับที่สูงกว่าการพัฒนาระบบในแนวทางเดิม เนื่องจากจะแสดงอยู่ในระดับ Object เพราะภายใน Object จะมีทั้งข้อมูลและหน้าที่ของตัวมันเองที่ทำงานร่วมกัน

ในแต่ละขั้นตอนของการพัฒนาระบบเชิงวัตถุ จะมีความกลมกลืนกันมาก กล่าวคือการเปลี่ยนจากขั้นหนึ่งไปยังอีกขั้นหนึ่งนั้น เป็นไปอย่างเรียบง่าย เพราะมีการใช้ภาษาที่เหมือนกันตลอดการพัฒนา และภาษาในที่นี้ก็คือแบบจำลองนั่นเอง ซึ่งการใช้แบบจำลองเดียวกันตลอดในการพัฒนาจะมีผลทำให้เวลาในการพัฒนานั้นมีความรวดเร็วยิ่งขึ้น ขณะที่การพัฒนาในแนวทางเดิมนั้นในแต่ละขั้นของการพัฒนาจะมีรูปแบบและวิธีที่ต่างกันไป

ในการพัฒนาเชิงวัตถุนี้จะมีการสนับสนุนการนำของที่มีอยู่กลับมาใช้ใหม่ (Reusability) เพราะจะนำ Object แทนด้วยโลกของความเป็นจริง และแต่ละ Object นั้นสามารถอยู่ได้ด้วยตัวเอง เมื่อนำ Object แทนด้วยโลกของความจริง ก็แสดงว่าเป็นการไม่ผูกมัดว่าจะต้องอยู่ในเพียงระบบนี้เท่านั้น กล่าวคือ การออกแบบนั้นจะพิจารณาถึงการให้ใช้งานได้ทั่วไป

5.3 แนวทางในการพัฒนาระบบ

การพัฒนาระบบเชิงวัตถุนี้เป็นแนวทางล่าสุดในการพัฒนาระบบ ซึ่งยังไม่มีข้อพิสูจน์ที่แน่ชัดว่าพัฒนาระบบเชิงวัตถุจะเป็นแนวทางสำหรับการพัฒนาระบบที่ดีที่สุด แต่ก็เป็นแนวทางหนึ่งของการพัฒนาระบบที่สามารถรองรับความซับซ้อนที่เกิดขึ้นได้ และมีการใช้แบบจำลองที่เหมือนกันตลอดทั้งการพัฒนานั่นคือ UML (Unified Modeling Language) และยังมีการสนับสนุนการ Reuse อย่างเต็มที่ ซึ่งทำให้มีผลกับระยะเวลาของการพัฒนานั้นเร็วขึ้น

สำหรับ UML เองนั้นมีส่วนช่วยในการพัฒนาเป็นอย่างมาก เนื่องจากมีแบบจำลองที่สามารถสนับสนุนสภาวะแวดล้อมในการพัฒนาระบบเชิงวัตถุทุกๆ ด้าน ไม่ว่าจะเป็นด้านการรวบรวมความต้องการ ด้านการวิเคราะห์ ด้านการออกแบบ หรือแม้แต่ด้านการสร้างระบบ และยิ่งในปัจจุบันมีเครื่องมือที่สนับสนุนการพัฒนาระบบเชิงวัตถุเพิ่มขึ้นเรื่อยๆ และมีผลทำให้การพัฒนาระบบเชิงวัตถุมีแนวโน้มเป็นการพัฒนาระบบในอนาคตต่อไปได้

บทที่ 6

คู่มือระบบ

6.1 ขอบเขต (Scope) ของระบบ

6.1.1 ผู้อ่าน (Audience)

เป็นผู้ที่มีหน้าที่ดูแลรักษาระบบที่ถูกพัฒนาขึ้น และจะต้องมีความรู้ด้านการพัฒนาระบบเล็กน้อย และโดยเฉพาะอย่างยิ่งต้องอ่านสัญลักษณ์ต่างๆ ของ UML (Unified Modeling Language) ได้

6.1.2 องค์กร (Organization)

หน่วยงานรับชำระเป็นหน่วยงานหนึ่งของการเคหะแห่งชาติ และระบบรับชำระเป็นเพียงส่วนหนึ่งของหน่วยงานรับชำระ

6.2 ความต้องการของระบบ (System Requirement)

ลูกหนี้สามารถชำระหนี้ได้โดยการชำระด้วยเงินสด หรือชำระผ่านธนาคาร ซึ่งการชำระผ่านธนาคารนั้นจะเป็นการให้ทางธนาคารเป็นผู้หักเงินจากบัญชีของลูกหนี้

รายละเอียดการรับชำระของลูกหนี้แต่ละรายจะต้องถูกบันทึกเก็บไว้ได้ทั้งชำระด้วยเงินสดและชำระผ่านธนาคาร ลูกหนี้ที่ชำระหนี้ในแต่ละงวดจะได้รับใบเสร็จรับเงินเพื่อเป็นการยืนยันการชำระ

ลูกหนี้จะชำระจำนวนเงินเท่าที่สัญญาระบุไว้ต่องวด (ต้องไม่น้อยกว่า) และลูกหนี้สามารถชำระหนี้ได้มากกว่าจำนวนเงินที่กำหนดไว้ต่องวด ซึ่งอาจเป็นการชำระหนี้ที่เหลือทั้งหมดหรือเป็นการชำระเพียงบางส่วน ดังนั้นต้องมีการคำนวณยอดหนี้คงเหลือได้ และหากลูกหนี้มีการชำระมากกว่าจำนวนเงินที่กำหนดไว้ต่องวด (เป็นกรณีที่ชำระเพียงบางส่วน) สัญญาเดิมของลูกหนี้ที่ทำไว้และต้องมีการแก้ไขข้อมูลภายในสัญญาใหม่ เพื่อเป็นการกำหนดจำนวนหนี้ทั้งหมดและจำนวนเงินที่จ่ายต่องวดใหม่ ดังนั้นจึงต้องมีการแก้ไขสถานะของสัญญาเพื่อเป็นการบอกหน่วยงานที่รับผิดชอบทราบและดำเนินการต่อไป

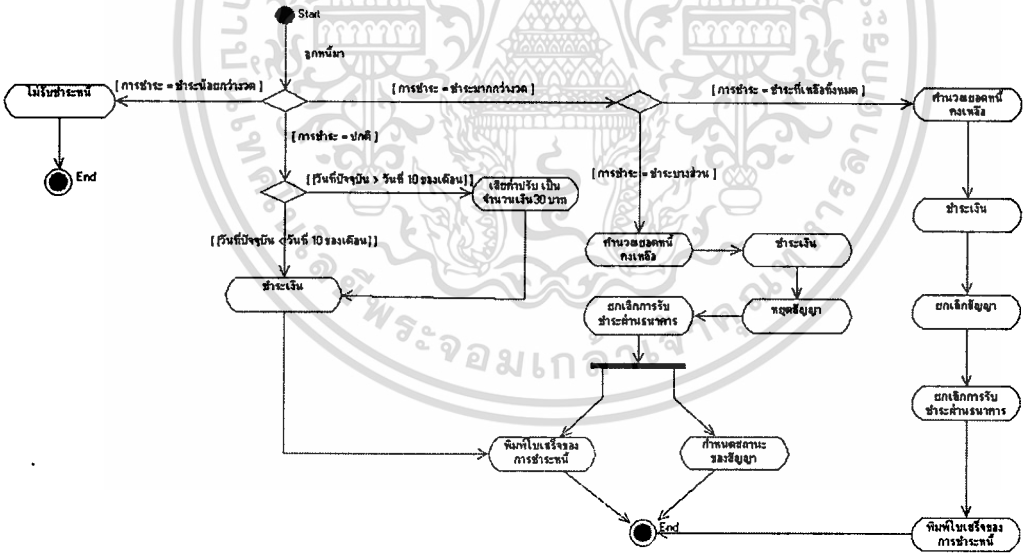
ระบบสามารถออกรายงานประมาณการรายรับและรายงานการรับจริงได้ สรุปการรับชำระประจำวัน สรุปการรับชำระประจำเดือน สรุปการหักบัญชีได้-ไม่ได้ของแต่ละธนาคาร สรุปลูกหนี้คงค้าง และรายงานลูกหนี้คงเหลือ

ข้อกำหนด

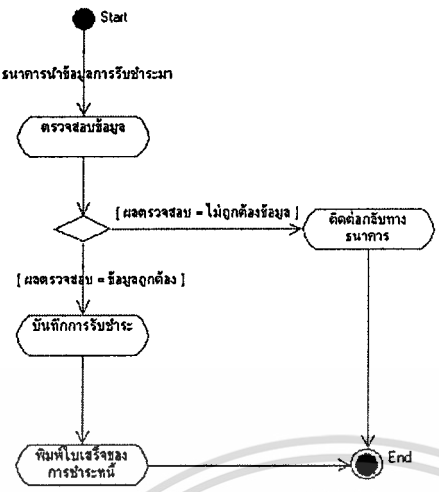
- ลูกหนี้จะไม่สามารถเปลี่ยนรูปแบบการชำระหนี้ได้ตามใจ จะต้องมีการแจ้งให้ทางการเคหะรับทราบก่อนทุกครั้งที่ต้องการเปลี่ยนรูปแบบการชำระ
- ลูกหนี้ไม่สามารถชำระหนี้น้อยกว่าจำนวนเงินที่กำหนดไว้ต่องวด
- กรณีที่ลูกหนี้มีการชำระหนี้ผ่านทางธนาคาร และลูกหนี้ต้องการจะชำระหนี้มากกว่าจำนวนเงินที่กำหนดไว้ เมื่อชำระแล้วจะถือว่าความประสงค์ที่แจ้งไว้เพื่อการขอชำระผ่านธนาคารที่ผ่านมานั้นจะถูกยกเลิกโดยทันที
- หากลูกหนี้ชำระเงินหลังจากวันที่ 10 ของทุกเดือน ลูกหนี้จะต้องเสียค่าปรับเป็นจำนวนเงิน 30 บาท

6.3 แบบจำลองของกระบวนการทางธุรกิจ

แบบจำลองของกระบวนการทางธุรกิจของระบบรับชำระจะแสดงดังรูปที่ 6.1 และ รูปที่ 6.2 ซึ่งจะแสดงกระบวนการของการรับชำระเงินสดและกระบวนการของการรับชำระผ่านธนาคารตามลำดับ



รูปที่ 6.1แสดงกระบวนการการรับชำระเงินสด



รูปที่ 6.2 แสดงกระบวนการการรับชำระผ่านธนาคาร

6.4 แบบจำลอง Use Case

6.4.1 Actor

Bank (ธนาคาร) ส่งข้อมูลที่ลูกหนี้ชำระผ่านธนาคาร

Debtor (ลูกหนี้) ชำระหนี้ด้วยเงินสดและทำการขอชำระผ่านธนาคาร

Official (พนักงานรับชำระ) ทำงานกับระบบรับชำระ

6.4.2 Use Case

AllTermCashPayment (รับชำระเงินสดที่เหลือทั้งหมด)

ผู้เริ่มต้น : ลูกหนี้

หน้าที่ : บันทึกการรับชำระด้วยเงินสดที่เหลือทั้งหมดของลูกหนี้

BankPayment (รับชำระผ่านธนาคาร)

ผู้เริ่มต้น : ธนาคาร

หน้าที่ : บันทึกการรับชำระผ่านธนาคารของลูกหนี้

CalculateRemainDebt (คำนวณยอดหนี้คงเหลือ)

ผู้เริ่มต้น : พนักงานรับชำระ

หน้าที่ : คำนวณยอดหนี้คงเหลือของลูกหนี้

CashPayment (รับชำระเงินสด)

ผู้เริ่มต้น : ลูกหนี้

หน้าที่ : บันทึกการรับชำระด้วยเงินสดของลูกหนี้

MoreTermCashPayment (รับชำระเงินสดมากกว่าจำนวนต่องวด)

ผู้เริ่มต้น : ลูกหนี้

หน้าที่ : บันทึกการรับชำระด้วยเงินสดที่มากกว่าจำนวนเงินต่องวดของลูกหนี้ (ยังคงเหลือหนี้อยู่)

Receipt (พิมพ์ใบเสร็จ)

ผู้เริ่มต้น : พนักงานรับชำระ

หน้าที่ : พิมพ์ใบเสร็จสำหรับผู้ที่ชำระหนี้

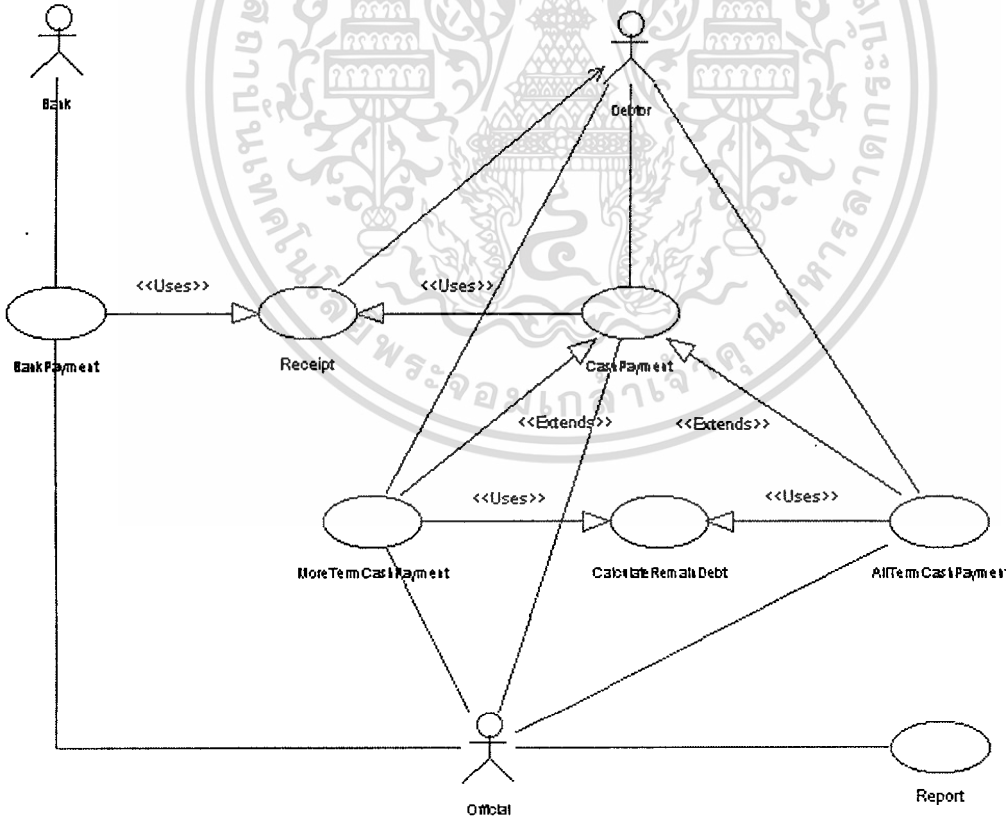
Report (พิมพ์รายงาน)

ผู้เริ่มต้น : พนักงานรับชำระ

หน้าที่ : พิมพ์รายงานต่างๆ ของการรับชำระ

6.4.3 Use Case Diagram

จะแสดง Use Case Diagram ดังรูปที่ 6.3

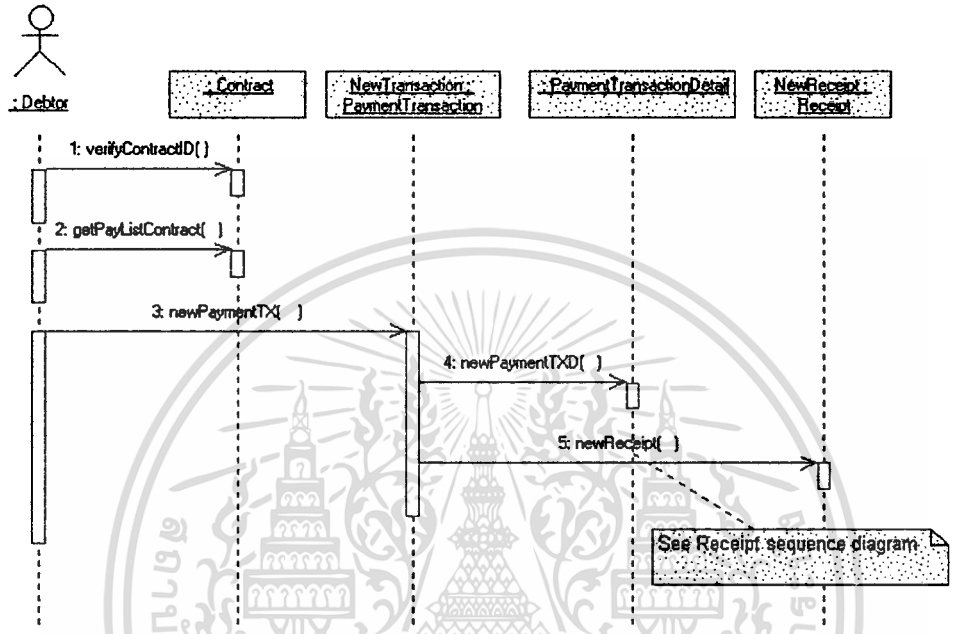


รูปที่ 6.3 แสดง Use Case Diagram ของระบบรับชำระ

6.5 Interaction Diagram

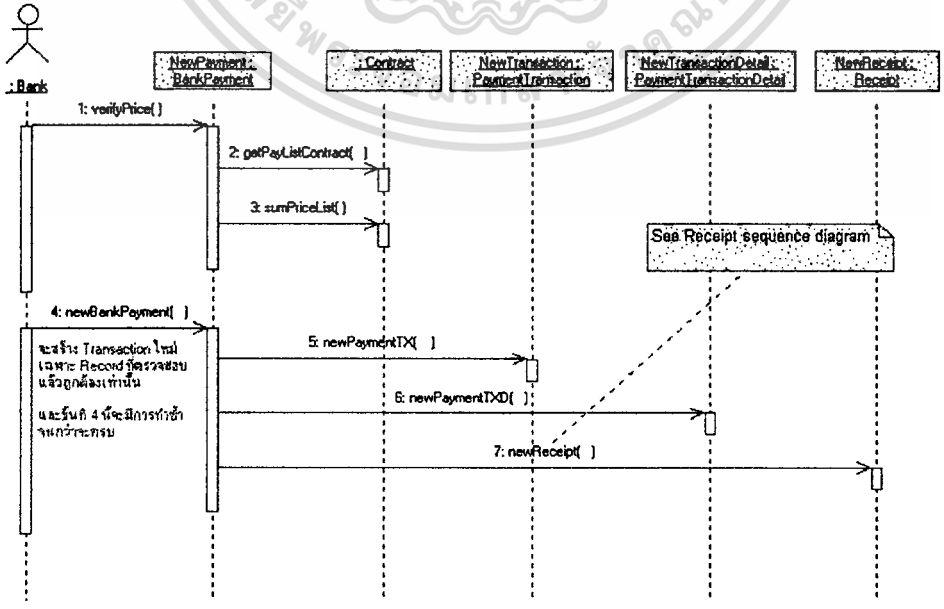
แสดง Sequence Diagram ของแต่ละ Use Case ดังนี้

6.5.1 Sequence Diagram ของ CashPayment Use Case



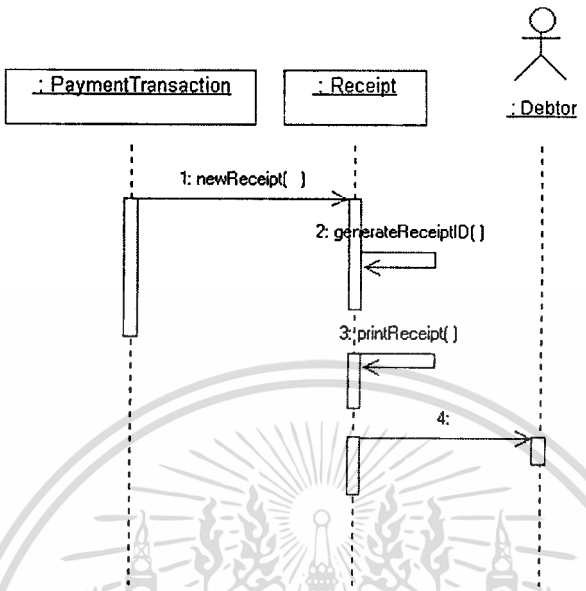
รูปที่ 6.4 แสดง Sequence Diagram ของ CashPayment Use Case

6.5.2 Sequence Diagram ของ BankPayment Use Case



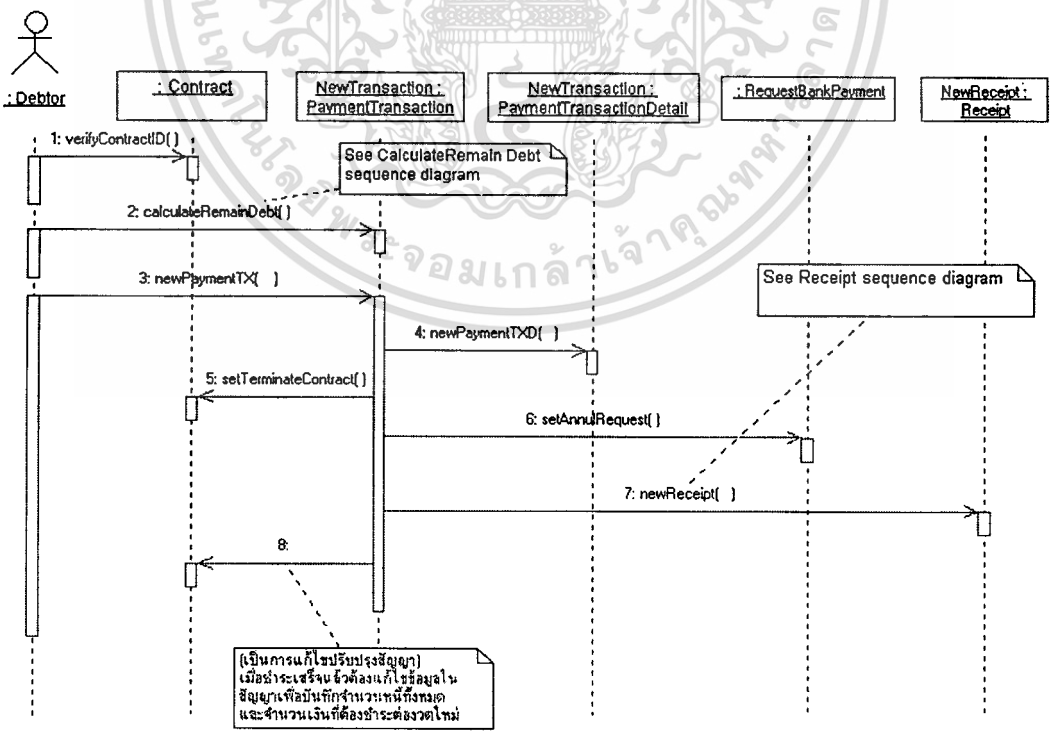
รูปที่ 6.5 แสดง Sequence Diagram ของ BankPayment Use Case

6.5.3 Sequence Diagram ของ Receipt Use Case



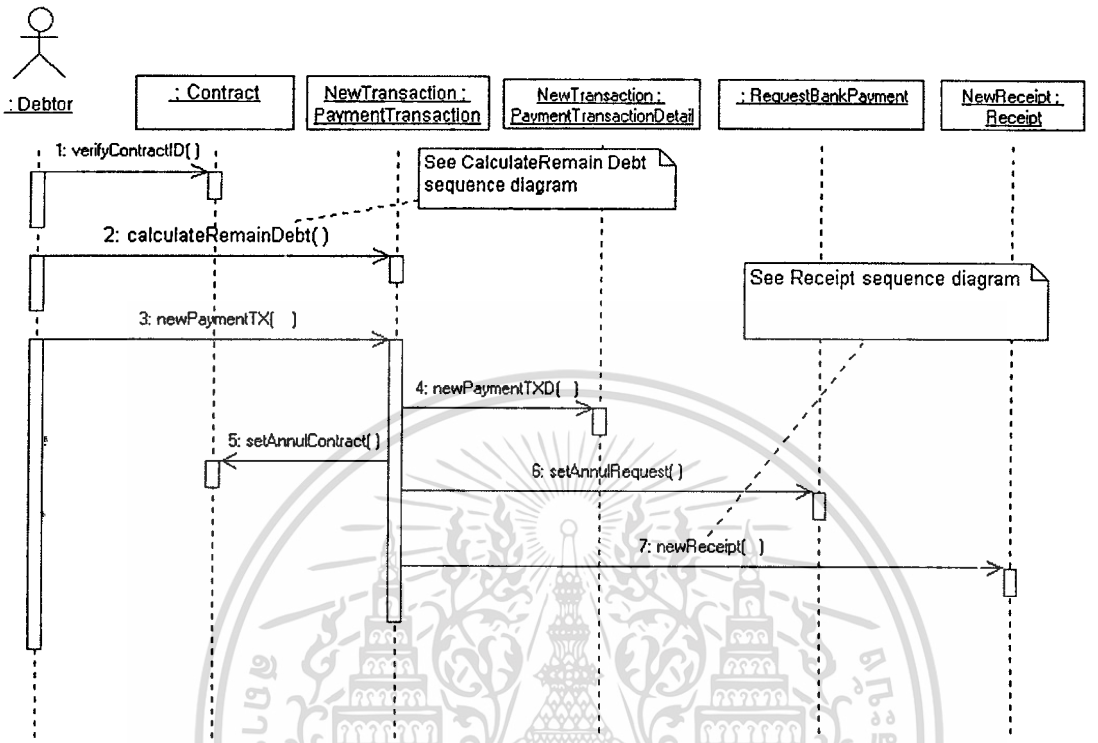
รูปที่ 6.6 แสดง Sequence Diagram ของ Receipt Use Case

6.5.4 Sequence Diagram ของ MoreTermCashPayment Use Case



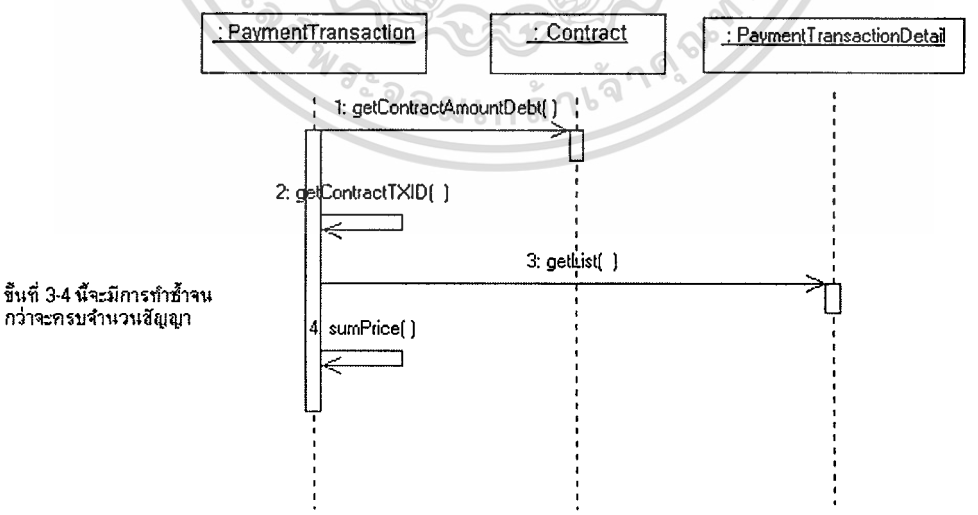
รูปที่ 6.7 แสดง Sequence Diagram ของ MoreTermCashPayment Use Case

6.5.5 Sequence Diagram ของ AllTermCashPayment Use Case



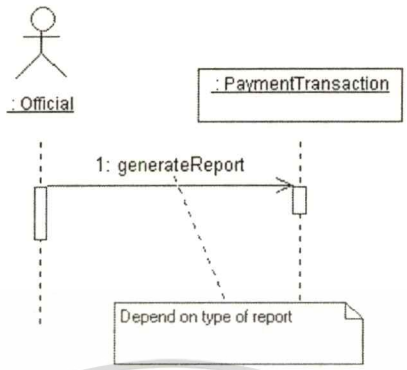
รูปที่ 6.8 แสดง Sequence Diagram ของ AllTermCashPayment Use Case

6.5.6 Sequence Diagram ของ CaculationRemainDebt



รูปที่ 6.9 แสดง Sequence Diagram ของ CaculationRemainDebt

6.5.7 Sequence Diagram ของ Report Use Case



รูปที่ 6.10 แสดง Sequence Diagram ของ Report Use Case

6.6 Class Diagram

6.6.1 Class

ตารางที่ 6.1 แสดงคลาสของระบบรับชำระในขั้นการวิเคราะห์

ชื่อ	วัตถุประสงค์
AccountInformation	เป็นรายการบัญชีที่ใช้ระบบรับชำระ ซึ่งเป็นข้อมูลที่ต้องการในการบันทึกการรับชำระแต่ละรายการ
BankInformation	เป็นธนาคารที่ลูกหนี้ชำระหนี้ผ่านให้กับการเคหะ
BankPayment	เป็นการเก็บรายละเอียดของการรับชำระผ่านธนาคารของลูกหนี้
BranchInformation	เป็นสาขาใดของหน่วยงานห้างร้าน ซึ่งในกรณีนี้จะเป็นสาขาของธนาคารที่ลูกหนี้ชำระหนี้ผ่านให้กับการเคหะ
Contract	เป็นสัญญาของลูกหนี้ ที่ได้ทำไว้กับทางการเคหะ ซึ่งเป็นข้อมูลที่ต้องการในการรับชำระ
PaymentTransaction	เป็น Transaction การรับชำระของระบบ ซึ่งเป็นการเก็บรายละเอียดการรับชำระในแต่ละครั้ง
PaymentTransactionDetail	เป็นรายละเอียดของแต่ละ Transaction ของการรับชำระ
Receipt	เป็นใบเสร็จสำหรับลูกหนี้ที่ชำระหนี้ในแต่ละครั้ง

6.6.2 ความสัมพันธ์ของ Class

6.6.2.1 Contract กับ PaymentTransaction

สัญญา (Contract) ของลูกหนี้ทุกสัญญาที่มีการทำไว้กับทางการเคหะ ลูกหนี้ที่ถือสัญญานั้นๆ จะต้องมีการจ่ายเงิน (PaymentTransaction) เกิดขึ้น และสัญญาหนึ่งๆ จะต้องมีการจ่ายเงินหนึ่งครั้งหรือมากกว่าหนึ่งครั้งขึ้นไป ดังนั้นจึงเกิดความสัมพันธ์ระหว่าง Contract กับ PaymentTransaction และเป็นความสัมพันธ์แบบ Association

6.6.2.2 PaymentTransaction กับ Receipt

การที่ลูกหนี้มาจ่ายเงินหนึ่งครั้งนั้นลูกหนี้จะได้รับใบเสร็จ (Receipt) หนึ่งใบ ภายในใบเสร็จหนึ่งใบนี้จะประกอบไปด้วยการจ่ายเงินครั้งนั้นๆ ทำให้เกิดความสัมพันธ์แบบ Aggregation ระหว่าง Receipt กับ PaymentTransaction ซึ่ง Receipt จะเป็นฝั่งทั้งหมด (Whole) และ Payment-Transaction จะเป็นฝั่งส่วนประกอบ (Part) ซึ่งเป็นความสัมพันธ์ชนิด Container เพราะปกติตามหลักความเป็นจริงใบเสร็จใบหนึ่งๆ อาจไม่ได้ประกอบไปด้วยการชำระเสมอไป

6.6.2.3 PaymentTransactionDetail กับ PaymentTransaction

ในหนึ่งครั้งของการชำระเงินของลูกหนี้จะมีรายละเอียดการชำระเงิน (Payment-TransactionDetail) เกิดขึ้นเพราะการชำระเงินนั้นอาจมีการชำระได้ตั้งแต่หนึ่งรายการขึ้นไปภายในหนึ่งครั้ง จึงทำให้เกิดความสัมพันธ์ระหว่าง PaymentTransactionDetail กับ PaymentTransaction และเป็นความสัมพันธ์แบบ Aggregation ชนิด Composition เพราะการชำระเงินหนึ่งรายการหรือหลายๆ รายการนั้นจะเกิดขึ้นภายในครั้งหนึ่งๆ เท่านั้นและหากไม่เกิดขึ้น (ในหนึ่งครั้งคือไม่เกิดเลย) ก็จะไม่มีรายละเอียดรายการที่ชำระเลย ดังนั้น PaymentTransactionDetail จึงเป็นด้านทั้งหมด (Whole) และ PaymentTransaction เป็นด้านส่วนประกอบ (Part)

6.6.2.4 PaymentTransactionDetail กับ AccountInformation

เนื่องจากระบบรับชำระนี้เป็นส่วนหนึ่งของระบบบัญชีลูกหนี้ ดังนั้นในการชำระแต่ละรายการที่ชำระก็จะมีการบ่งบอกถึงชนิดบัญชี (AccountInformation) ที่ลูกหนี้ชำระมาซึ่งรายการหนึ่งก็จะบอกถึงชนิดบัญชีหนึ่งเท่านั้น จึงทำให้เกิดความสัมพันธ์แบบ Association ระหว่าง PaymentTransactionDetail กับ AccountInformation

6.6.2.5 BankPayment กับ PaymentTransaction

ในกรณีที่ลูกหนี้ชำระผ่านธนาคารนั้น การชำระผ่านธนาคาร (BankPayment) หนึ่งครั้งก็จะประกอบไปด้วยการจ่ายเงินหนึ่งครั้ง แต่หากลูกหนี้ไม่จ่ายเงินก็จะถือว่าไม่ได้ทำการชำระผ่านธนาคารเช่นกัน ดังนั้นจึงเกิดความสัมพันธ์แบบ Aggregation ชนิด Composition ระหว่าง

BankPayment กับ PaymentTransaction ซึ่ง BankPayment จะเป็นส่วนทั้งหมดและ Payment-Transaction จะส่วนประกอบ

6.6.2.6 BranchInformation กับ BankInformation

โดยทั่วไปแล้วคำว่าสาขา (BranchInformation) นั้นจะแทนด้วยพื้นที่ที่อยู่ตำแหน่งหนึ่งๆ แล้วแต่ละสาขาจะประกอบหน่วยงานหรือห้างร้าน (BankInformation) ดังนั้นจึงเกิดความสัมพันธ์แบบ Aggregation แบบ Container ระหว่าง BranchInformation กับ BankInformation ซึ่ง Branch-Information จะเป็นส่วนทั้งหมดและ BankInformation จะเป็นส่วนประกอบ

6.6.2.7 BankPayment กับ BranchInformation

สำหรับการชำระเงินผ่านธนาคารนั้น ลูกหนี้จะถูกหักบัญชีที่สาขาของธนาคารใดธนาคารหนึ่ง ซึ่งหมายความว่าเป็นการชำระเงินผ่านสาขาของธนาคาร ดังนั้นการหักบัญชี (การชำระเงินผ่านธนาคาร) จะประกอบไปด้วยสาขาของธนาคารใดธนาคารหนึ่ง และหากสาขานั้นๆ ไม่หักเงินจากบัญชีดังนั้นก็ไม่มีการหักอะไรเกิดขึ้นทั้งนั้น ซึ่งหมายความว่าจะไม่มีการชำระเงินเกิดขึ้น จึงทำให้เกิดความสัมพันธ์แบบ Aggregation แบบ Composition ระหว่าง BankPayment กับ Branch-Information ส่วน BankPayment จะเป็นส่วนทั้งหมดและ BranchInformation จะเป็นส่วนประกอบ

6.6.3 Attribute

โดยจะแสดงชื่อ Attribute ประเภทข้อมูล พร้อมคำอธิบายของแต่ละ Attribute ดังนี้

AccountInformation

Protected Attributes:

AccountID : CHAR(4)	รหัสบัญชี เช่น 0520
AccountName : VARCHAR2(30)	ชื่อบัญชี เช่นค่าเช่า, เช่าซื้อ (Install), ค่าบริการ (Service), ค่าประกัน (Insurance), ภาษี (Tax), ค่าที่ดินส่วนเกิน (Ex-land), สินเชื่อ (Loan), ค่าปรับ (Re-Schedule), อื่นๆ (Other)
AccountDescription : VARCHAR2(255)	คำอธิบายเพิ่มเติมแต่ละชื่อบัญชี

BankInformation

Protected Attributes:

bankCode : CHAR(4)	รหัสที่ใช้ธนาคาร เช่น SCB
bankName : VARCHAR2(30)	ชื่อธนาคาร
bankDetail : VARCHAR2(255)	เป็นคำอธิบายสำหรับแต่ละธนาคาร

BankPayment

Protected Attributes:

txID : NUMBER(15)	รหัสสำหรับแต่ละ Transaction
branch : VARCHAR2(30)	ชื่อสาขา
bankCode : CHAR(4)	รหัสที่ใช้ธนาคาร เช่น SCB

BranchInformation

Protected Attributes:

branch : VARCHAR2(30)	ชื่อสาขา
AddressID : VARCHAR2(20)	บ้านเลขที่
Moo : CHAR(2)	หมู่
Soi : VARCHAR2(30)	ซอย
Road : VARCHAR2(30)	ถนน
Precinct : VARCHAR2(30)	แขวง/ตำบล
District : VARCHAR2(30)	เขต/อำเภอ
Province : VARCHAR2(30)	จังหวัด
Country : VARCHAR2(30) = 'Thailand'	ประเทศ
PostalCode : CHAR(5)	รหัสไปรษณีย์

Contract

Protected Attributes:

contractID : CHAR(8)	เลขที่สัญญา
contractDate : DATE	วันที่ทำสัญญา
contractStatus : CHAR(1)	สถานะสัญญาใช้ U แทนสัญญาที่ใช้งาน, T แทนสัญญาที่รอการปรับปรุงเปลี่ยนแปลง และ A แทนสัญญาที่มีการชำระครบแล้ว
paymentType : CHAR(1)	ประเภทการชำระเงิน ใช้ C แทนชำระเงินสด และ B แทนชำระผ่านธนาคาร
pricePerTerm : NUMBER(11,2)	เงินที่ต้องจ่ายต่องวด
dateFirstTerm : DATE	วันที่ทำการจ่ายเงินงวดแรก
term : NUMBER(5)	จำนวนงวดที่ต้องจ่าย
contractType : CHAR(1)	ประเภทของสัญญาใช้ B แทนสัญญาเช่าซื้อ, R แทนสัญญาเช่า, L แทนสัญญากู้เงิน, CB แทนเช่าซื้อ-ประนอมหนี้ และ CL แทนเงินกู้-ประนอมหนี้
debtorCode : CHAR(14)	รายละเอียดลูกหนี้
customerID : CHAR(8)	รหัสลูกหนี้

PaymentTransaction

Protected Attributes:

txID : NUMBER(15)	รหัสสำหรับแต่ละ Transaction
paymentDate : DATE	วันที่ของ Transaction ที่รับชำระ
paymentTerm : NUMBER(5)	งวดที่ชำระ
contractID : CHAR(8)	เลขที่สัญญา

PaymentTransactionDetail

Protected Attributes:

txID : NUMBER(15)	รหัสสำหรับแต่ละ Transaction
AccountID : CHAR(4)	รหัสบัญชี เช่น 0520
paymentPrice : NUMBER(11,2)	จำนวนเงินที่ชำระต่อรายการบัญชี

Receipt

Protected Attributes:

receiptID : CHAR(8)	เลขที่ใบเสร็จ
receiptDate : DATE	วันที่ออกใบเสร็จครั้งแรก
receiptStatus : CHAR(1) = U	สถานะของใบเสร็จ ซึ่งใช้ U แทน Use และใช้ C แทน Cancel
txID : NUMBER(15)	รหัสสำหรับแต่ละ Transaction

6.6.4 Method

โดยจะแสดงชื่อ Method, Parameter พร้อมคำอธิบายของแต่ละ Method และ Algorithm สำหรับ Method ที่มีความซับซ้อนดังนี้

AccountInformation

Public Operations:

- getAccountName (account_ID : CHAR) : VARCHAR2

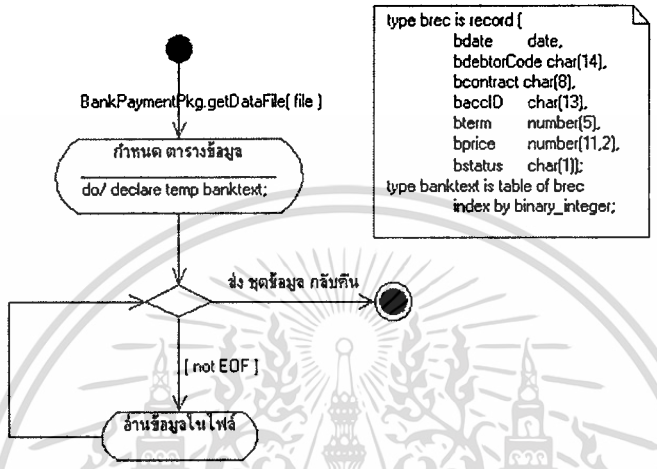
  ค้นหาชื่อบัญชี จากเลขที่บัญชี

BankPayment

Public Operations:

getDataFile (file : VARCHAR2)

นำข้อมูลจากไฟล์ที่ทางธนาคารส่งมาให้เข้าสู่โปรแกรม



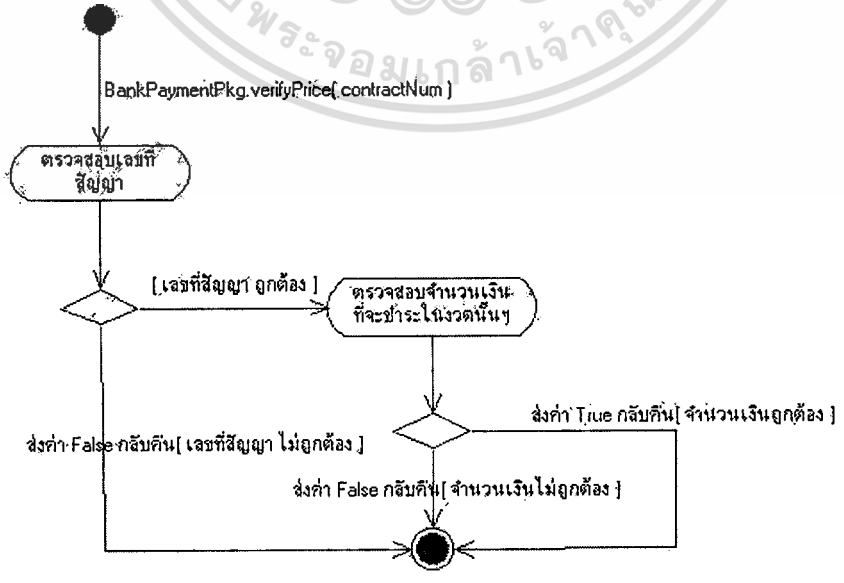
รูปที่ 6.11 แสดง Algorithm ของ getDataFile Method

newBankPayment (txID : NUMBER, bank : CHAR, branch : VARCHAR2)

เพิ่ม Object การรับชำระผ่านธนาคาร

verifyPrice (contractID : CHAR) : BOOLEAN

ตรวจสอบข้อมูลที่ธนาคารส่งมาให้



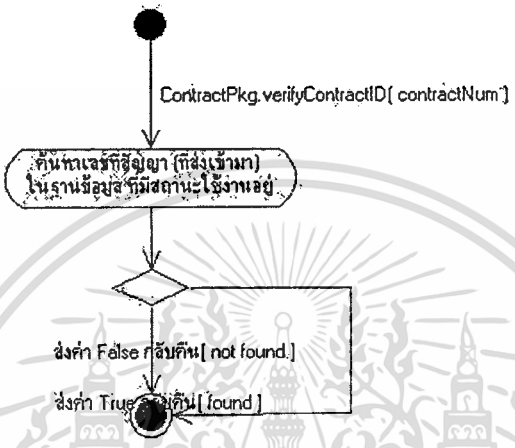
รูปที่ 6.12 แสดง Algorithm ของ verifyPrice Method

Contract

Public Operations:

verifyContractID (contractID : CHAR) : BOOLEAN

ตรวจสอบรหัสสัญญา ว่ามีหรือจริง ใช่ได้หรือไม่



รูปที่ 6.13 แสดง Algorithm ของ verifyContractID Method

getContractType (contractID : CHAR) : CHAR

ค้นหาประเภทของสัญญา จากเลขที่สัญญา

getContractStatus (contractID : CHAR) : CHAR

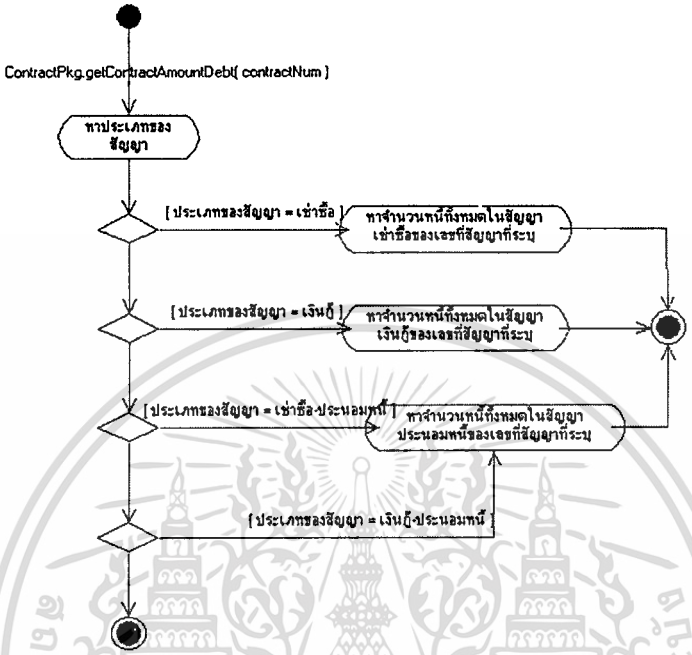
ค้นหาสถานะของสัญญา จากเลขที่สัญญา

getContractAmountTerm (contractID : CHAR) : NUMBER

ค้นหาจำนวนงวดที่ต้องชำระทั้งหมด จากเลขที่สัญญา

getContractAmountDebt (contractID : CHAR) : NUMBER

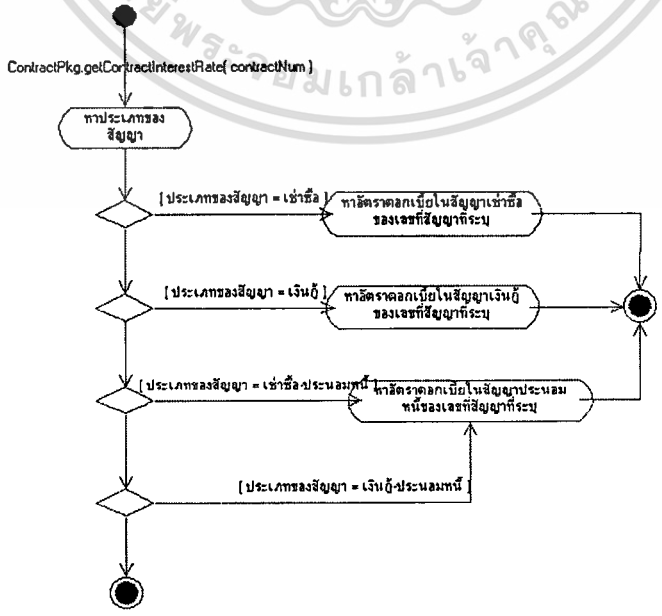
ค้นหาจำนวนหนี้ทั้งหมดของสัญญา จากเลขที่สัญญา



รูปที่ 6. 14 แสดง Algorithm ของ getContractAmountDebt Method

getContractInterestRate (contractID : CHAR) : NUMBER

ค้นหาอัตราดอกเบี้ยของสัญญา จากเลขที่สัญญา



รูปที่ 6. 15 แสดง Algorithm ของ getContractInterestRate Method

getContractPaymentType (aContactID : CHAR) : CHAR

ค้นหาประเภทของการชำระเงิน จากเลขที่สัญญา

getContractPricePerTerm (contractNum : CHAR) : NUMBER

ค้นหาจำนวนเงินที่ต้องชำระต่องวด จากเลขที่สัญญา

getContractDateFirstTerm (contractNum : CHAR) : DATE

ค้นหาวันที่ชำระเงินวันแรก จากเลขทีสัญญา

getCustomerContractID (customerID : VARCHAR2, tableOfContractID : TAB_TMP) :

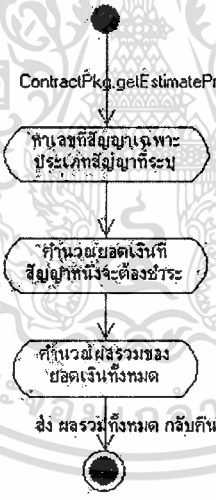
หา List (ชุดข้อมูล) ของเลขที่สัญญา จากข้อมูลลูกค้า

getDebtorContract (debtorCode : CHAR, tableOfContractID : TAB_TMP) :

หา List (ชุดข้อมูล) ของเลขที่สัญญา จากรหัสลูกหนี้

getEstimatePrice (con_type : CHAR) : NUMBER

คำนวณจำนวนเงินทั้งหมดต่องวดที่สัญญาควรจะชำระ (ประมาณรายรับ)



รูปที่ 6. 16 แสดง Algorithm ของ getEstimatePrice Method

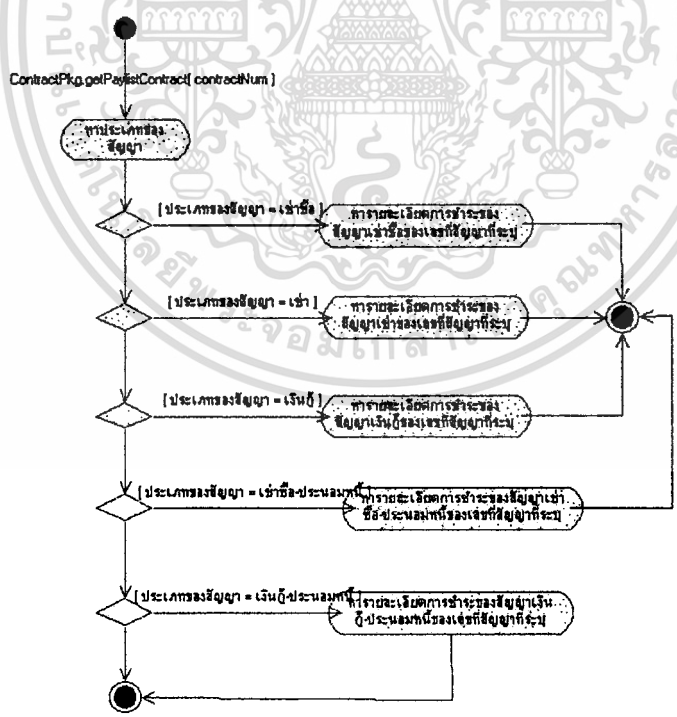
getEstimateCount (con_type : CHAR) : NUMBER

คำนวณจำนวนสัญญาทั้งหมดที่สัญญาควรจะชำระ (ประมาณจำนวนราย)

รูปที่ 6. 17 แสดง Algorithm ของ getEstimateCount Method

getPayListContract (contractID : CHAR, term : NUMBER, tableOfList : TAB_TMP) :

หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญาและงวดที่ชำระ



รูปที่ 6. 18 แสดง Algorithm ของ getPayListContract Method

sumPayListContract (tableOfList : TAB_TMP) : NUMBER

คำนวณผลรวมของ List (ชุดข้อมูล) รายละเอียดการชำระ จาก List ที่ส่งเข้ามา

setAnnulContract (contractStatus : CHAR = A) :

กำหนดสถานะของสัญญา จากเลขที่สัญญา := ยกเลิกสัญญา (กรณีชำระหนี้หมด)

setTerminateContract (contractStatus : CHAR = T) :

กำหนดสถานะของสัญญา จากเลขที่สัญญา := สิ้นสุดสัญญา (เพื่อให้หน่วยงานที่เกี่ยวข้องทำการเปลี่ยนแปลงแก้ไขข้อมูลในสัญญาสำหรับ กรณีมาชำระบางส่วนที่มากกว่าจำนวนงวด)

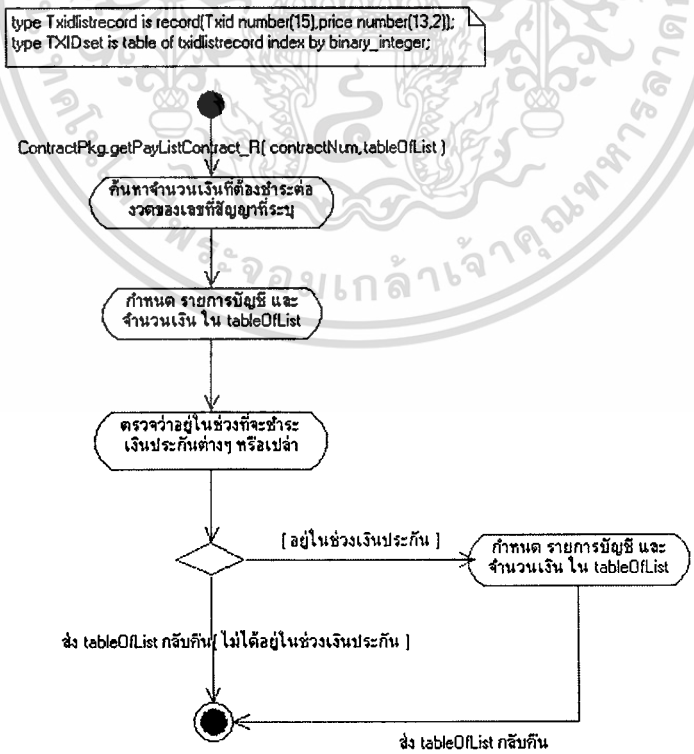
setPaymentType (contractID : CHAR) :

กำหนดสถานะของการชำระเงินให้เป็นการชำระแบบเงินสด โดยจะกำหนดจากเลขที่สัญญา

Protected Operations:

getPaylistContract_R (contractNum : CHAR, term : NUMBER, tableOfList : TAB_TMP) :

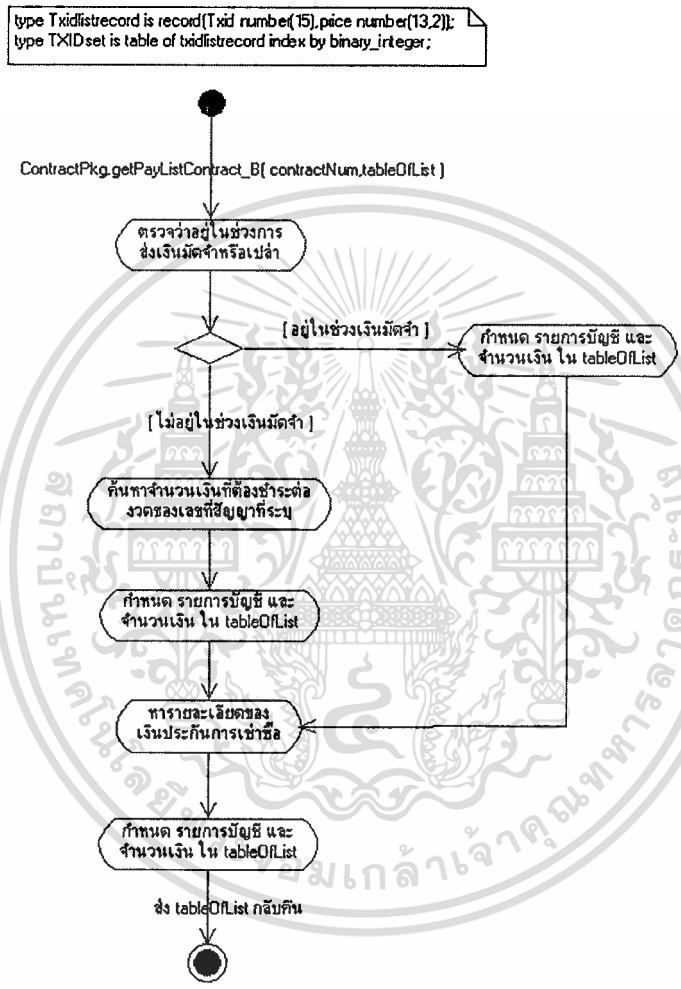
หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเช่าใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญาและงวดที่ชำระ



รูปที่ 6. 19 แสดง Algorithm ของ getPaylistContract_R Method

getPaylistContract_B (contractNum : CHAR, term : NUMBER, tableOfList : TAB_TMP) :

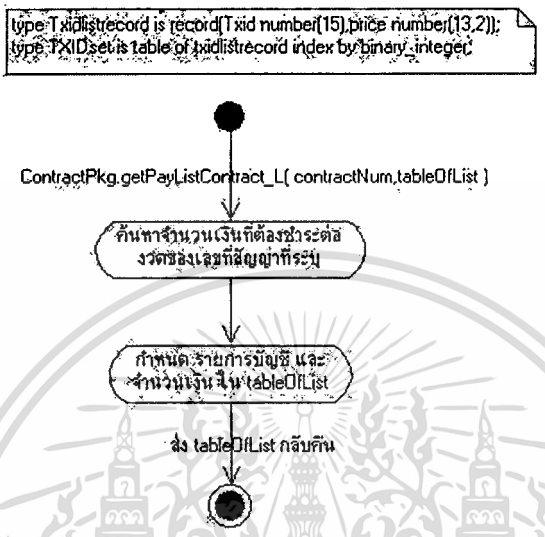
หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเช่าซื้อใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญาและงวดที่ชำระ



รูปที่ 6. 20 แสดง Algorithm ของ getPaylistContract_B Method

getPaylistContract_L (contractNum : CHAR, tableOfList : TAB_TMP) :

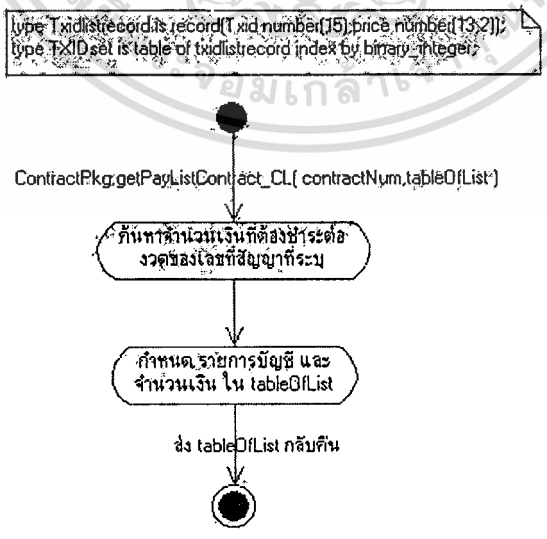
หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเงินกู้ใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญา



รูปที่ 6. 21 แสดง Algorithm ของ getPaylistContract_L Method

getPaylistContract_CL (contractNum : CHAR, tableOfList : TAB_TMP) :

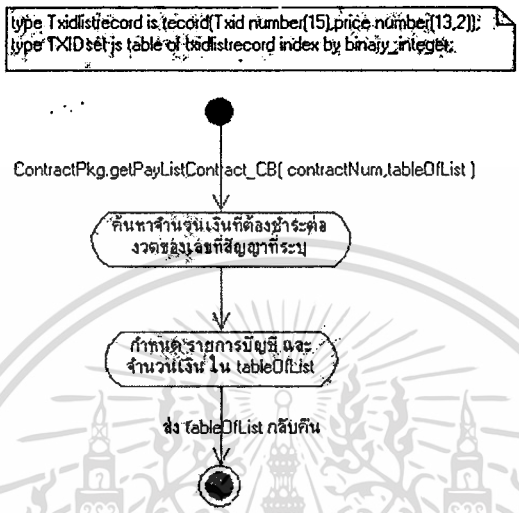
หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเงินกู้-ประนอมหนี้ใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญา



รูปที่ 6. 22 แสดง Algorithm ของ getPaylistContract_CL Method

getPaylistContract_CB (contractNum : CHAR, tableOfList : TAB_TMP) :

หา List (ชุดของข้อมูล) ของรายละเอียดที่สัญญาเช่าซื้อ-ประนอมหนี้ใบหนึ่งควรจะชำระในงวดนั้นๆ จาก เลขที่สัญญา



รูปที่ 6. 23 แสดง Algorithm ของ getPaylistContract_CB Method

getCompromiseInterestRate (contractNum : CHAR) : NUMBER

ค้นหาอัตราดอกเบี้ยของสัญญาประนอมหนี้ จากเลขที่สัญญา

getBuyInterestRate (contractNum : CHAR) : NUMBER

ค้นหาอัตราดอกเบี้ยของสัญญาเช่าซื้อ จากเลขที่สัญญา

getLoanInterestRate (contractNum : CHAR) : NUMBER

ค้นหาอัตราดอกเบี้ยของสัญญาเงินกู้ จากเลขที่สัญญา

getCompromiseAmountDebt (contractNum : CHAR) : NUMBER

ค้นหาจำนวนหนี้ทั้งหมดของสัญญาประนอมหนี้ จากเลขที่สัญญา

getBuyAmountDebt (contractNum : CHAR) : NUMBER

ค้นหาจำนวนหนี้ทั้งหมดของสัญญาเช่าซื้อ จากเลขที่สัญญา

getLoanAmountDebt (contractNum : CHAR) : NUMBER

ค้นหาจำนวนหนี้ทั้งหมดของสัญญาเงินกู้ จากเลขที่สัญญา

PaymentTransaction

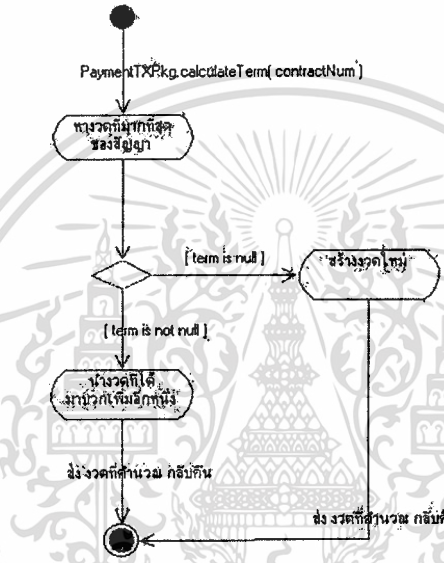
Public Operations:

newPaymentTX (txID : NUMBER, date : DATE, term : NUMBER, contractID : CHAR)

เพิ่ม Object การรับชำระ

calculateTerm (contractID : CHAR) : NUMBER

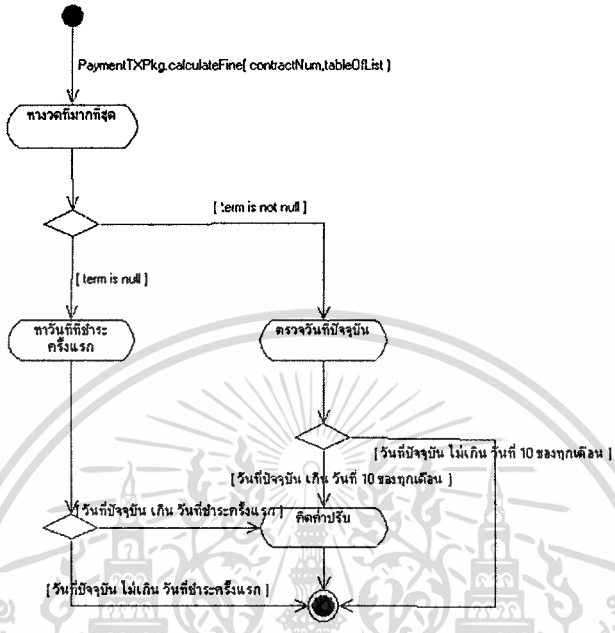
คำนวณงวดที่จะชำระ



รูปที่ 6. 24 แสดง Algorithm ของ calculateTerm Method

calculateFine (contractNum : CHAR, TableOfList : TAB_TMP)

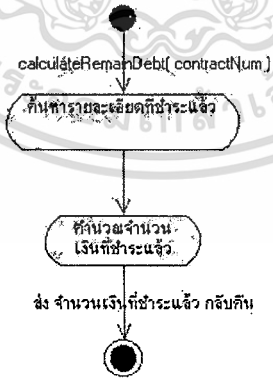
คำนวณค่าปรับ



รูปที่ 6. 25 แสดง Algorithm ของ calculateFine Method

calculateRemainDebt (contractID : CHAR)

คำนวณจำนวนเงินที่ลูกหนี้ชำระไปแล้ว



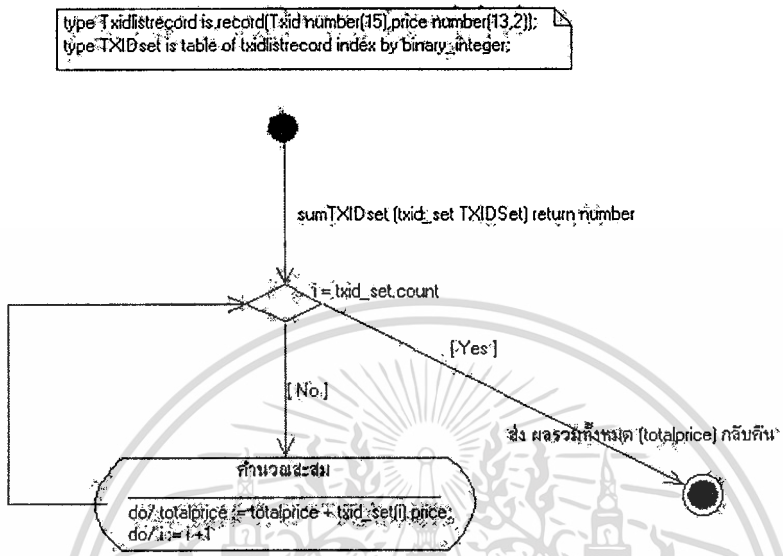
รูปที่ 6. 26 แสดง Algorithm ของ calculateRemainDebt Method

getContractTXID (contractID : CHAR, tableOfTXID : TAB_TMP) :

หา List (ชุดข้อมูล) ของ TXID จากเลขที่สัญญา

sumTXIDSet (txID_Set : TAB_TMP) : NUMBER

คำนวณผลรวมใน List (ชุดข้อมูล) ของ TXID



รูปที่ 6. 27 แสดง Algorithm ของ sumTXIDSet Method

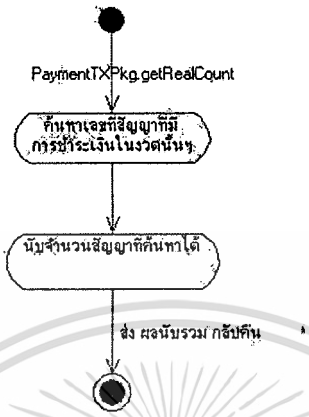
getRealPrice (con_type : CHAR, sdate : DATE, edate : DATE) : NUMBER

คำนวณจำนวนเงินทั้งหมดที่ลูกหนี้มาชำระ (คำนวณเงินที่รับจริง)

รูปที่ 6. 28 แสดง Algorithm ของ getRealPrice Method

getRealCount (con_type : CHAR, sdate : DATE, edate : DATE) : NUMBER

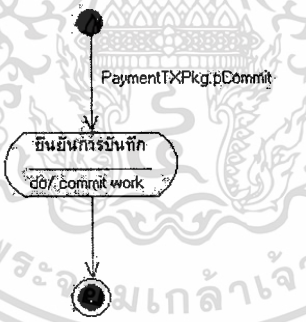
คำนวณจำนวนรายทั้งหมดที่สัญญาทำการชำระ (คำนวณจำนวนที่รับจริง)



รูปที่ 6. 29 แสดง Algorithm ของ getRealCount Method

pcommit ()

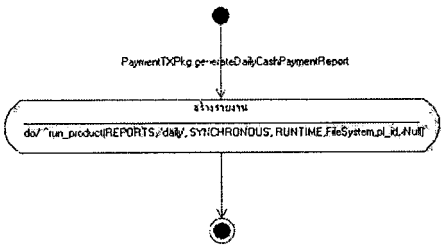
บันทึกลงฐานข้อมูลจริง



รูปที่ 6. 30 แสดง Algorithm ของ pcommit Method

generateDailyCashPaymentReport (date : DATE)

สร้างรายงานรับชำระประจำวัน



รูปที่ 6. 31 แสดง Algorithm ของ generateDailyCashPaymentReport Method

generateMonthlyCashPaymentReport (startDate : DATE, endDate : DATE)

สร้างรายงานรับชำระประจำเดือน

generateIncomeEstimateReport (startDate : DATE, endDate : DATE)

สร้างรายงานประมาณการรายรับ-รับจริง

generateRemainDebtorReport (customerID : CHAR)

สร้างรายงานลูกหนี้คงเหลือ (เป็นรายคน ว่าเหลือจำนวนหนี้ทั้งหมดเป็นจำนวนเท่าใด)

generateRemainDebtReport (startDate : DATE, endDate : DATE)

สร้างรายงานลูกหนี้คงค้าง

generateYClearing (startDate : DATE, endDate : DATE)

สร้างรายงานการตัดบัญชีได้

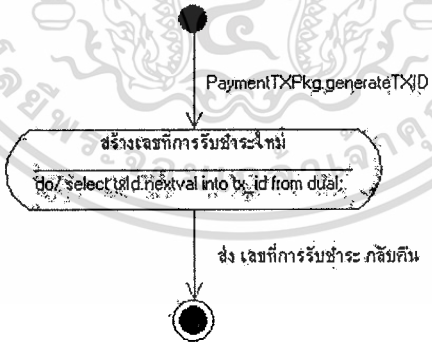
generateNClearing (startDate : DATE, endDate : DATE)

สร้างรายงานการตัดบัญชีไม่ได้

Protected Operations:

generateTXID () : NUMBER

สร้างเลขที่ของ Object การรับชำระ



รูปที่ 6. 32 แสดง Algorithm ของ generateTXID Method

PaymentTransactionDetail

Public Operations:

newPaymentTXD (txID : NUMBER, accountID : CHAR, price : NUMBER)

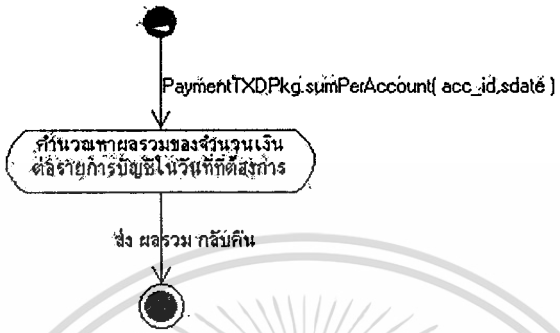
เพิ่ม Object รายละเอียดการรับชำระ

getPaymentDetail (txID : NUMBER, tableOfList : TAB_TMP)

หา List ของรายละเอียด จาก TXID

sumPerAccount (acc_id : CHAR, sdate : DATE) : NUMBER

หาผลรวมของจำนวนเงินทั้งหมด จากเลขที่บัญชีและวันที่ที่กำหนด



รูปที่ 6. 33 แสดง Algorithm ของ sumPerAccount Method

sumPerAccount (acc_id : CHAR, sdate : DATE, edate : DATE) : NUMBER

หาผลรวมของจำนวนเงินทั้งหมด จากเลขที่บัญชีและช่วงของวันที่ที่กำหนด

countPerAccount (acc_id : CHAR, sdate : DATE) : NUMBER

หาผลรวมของจำนวนรายทั้งหมด จากเลขที่บัญชีและวันที่ที่กำหนด

รูปที่ 6. 34 แสดง Algorithm ของ countPerAccount Method

countPerAccount (acc_id : CHAR, sdate : DATE, edate : DATE) : NUMBER

หาผลรวมของจำนวนรายทั้งหมด จากเลขที่บัญชีและช่วงของวันที่ที่กำหนด

Receipt

Public Operations:

newReceipt (receiptID : CHAR, receiptDate : DATE, txID : NUMBER)

เพิ่ม Object ใบเสร็จใหม่

setStatus (receiptID : CHAR)

กำหนดสถานะของ Object ใบเสร็จนั้นๆ ให้ยกเลิก

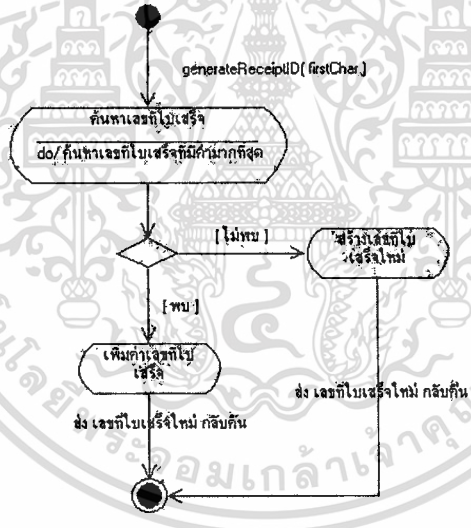
printReceipt (txID : NUMBER)

พิมพ์ใบเสร็จ

Protected Operations:

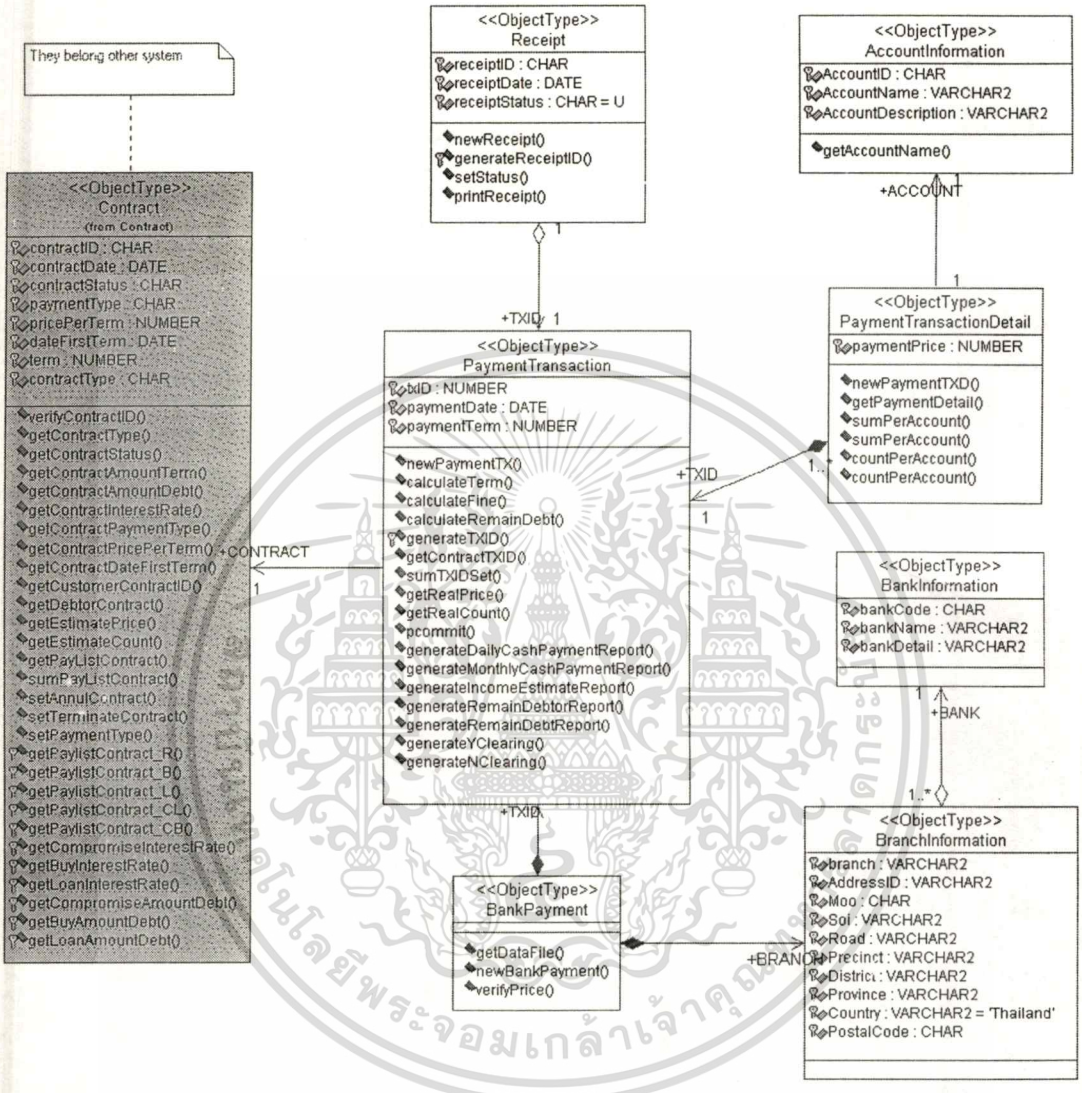
generateReceiptID (firstCharacter : CHAR = K) : CHAR

สร้างเลขที่ใบเสร็จใหม่



รูปที่ 6. 35 แสดง Algorithm ของ generateReceiptID Method

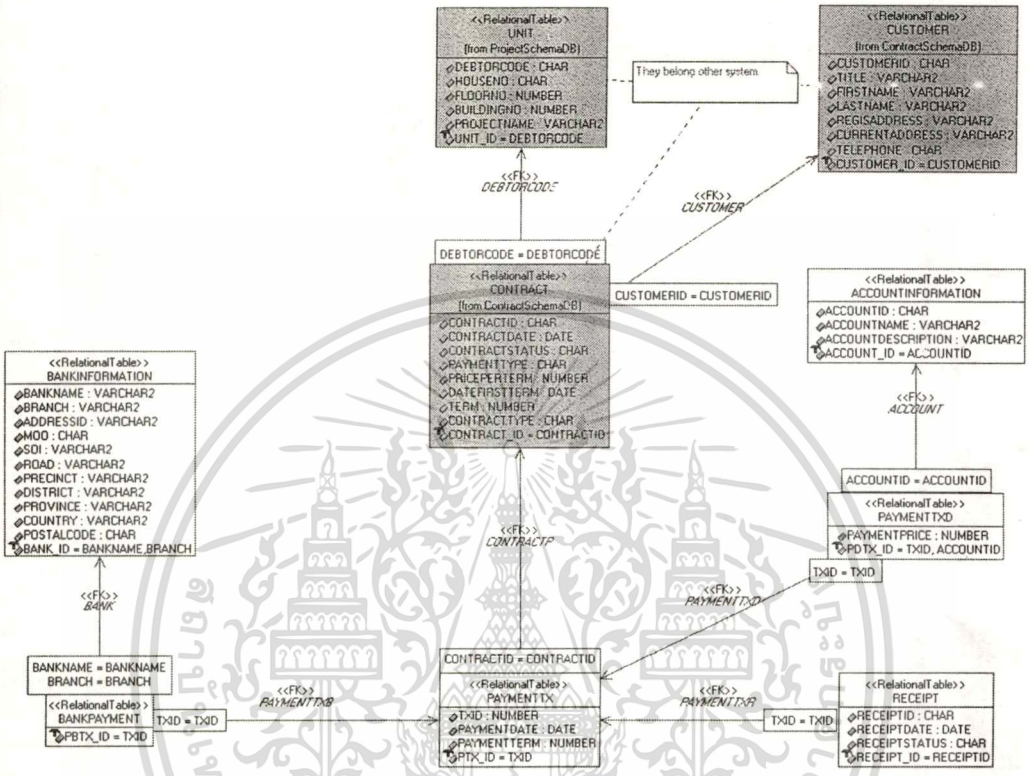
6.6.5 Class Diagram in Application Logic



รูปที่ 6. 36 แสดง Class Diagram ในส่วนของ ApplicationLogic

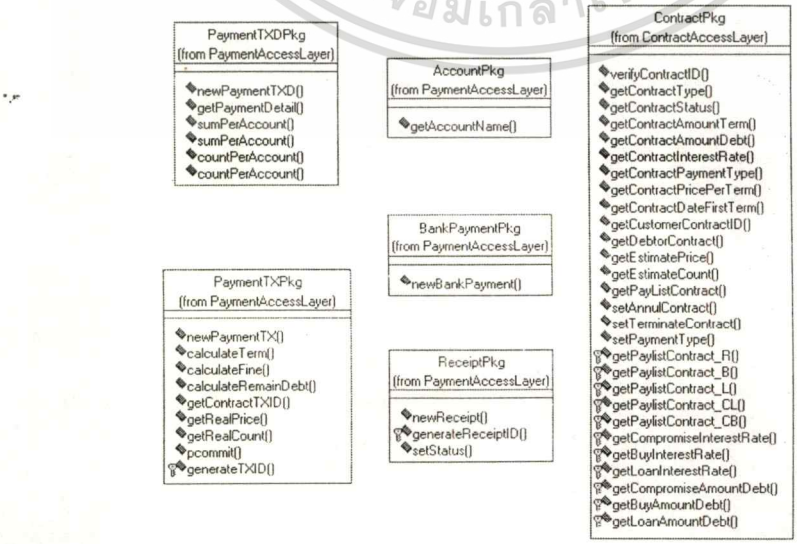
6.6.6 Class Diagram in Access Layer

6.6.6.1 Schema



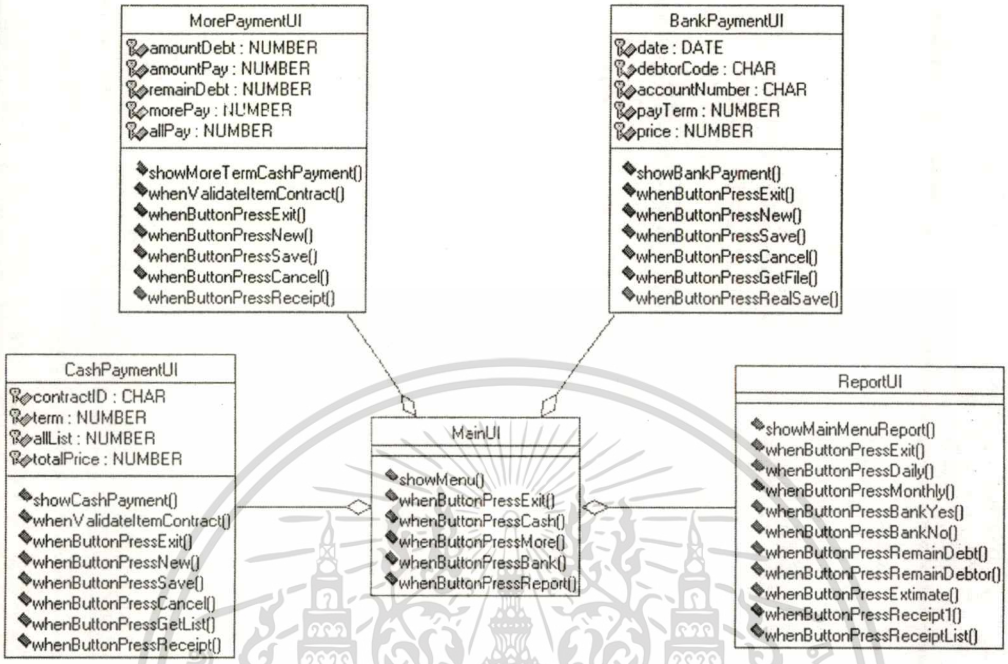
รูปที่ 6. 37 แสดง Schema ของระบบฐานข้อมูลโดยใช้ Class Diagram

6.6.6.2 Class Diagram in Access Layer



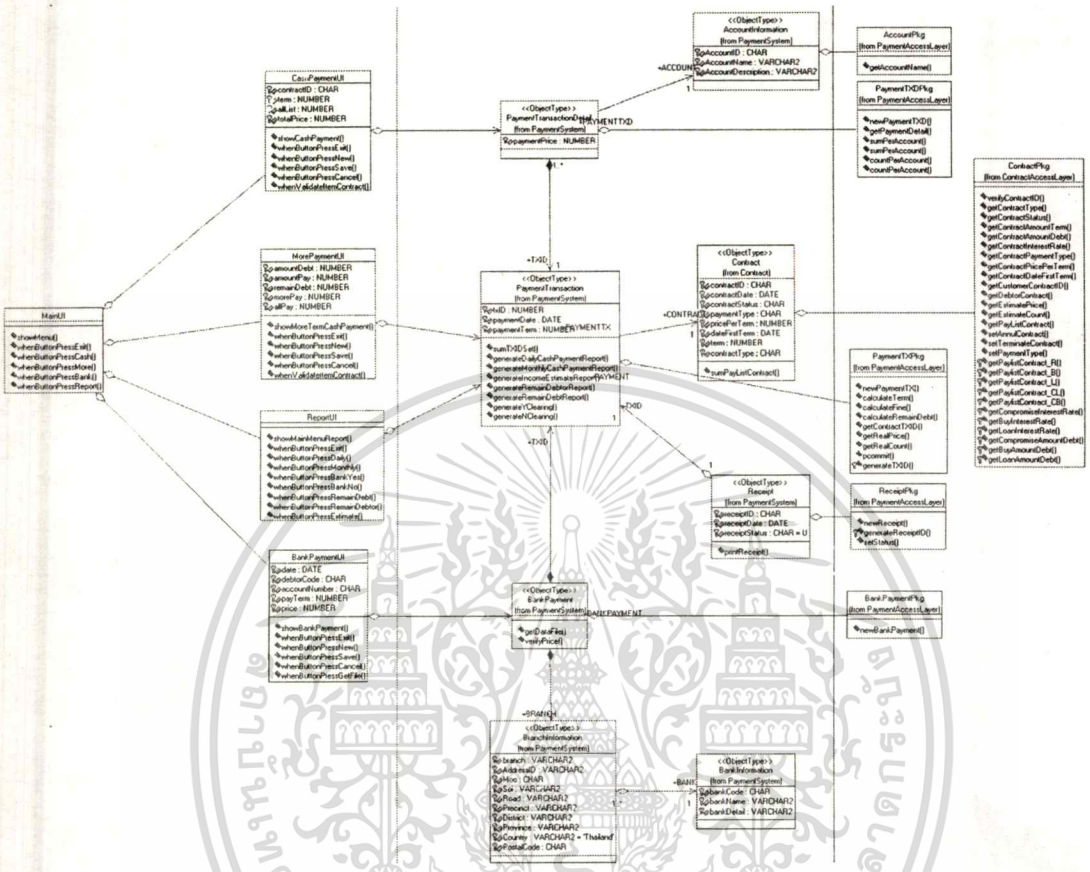
รูปที่ 6. 38 แสดง Class Diagram ในส่วนของ Access Layer

6.6.7 Class Diagram in View Layer



รูปที่ 6. 39 แสดง Class Diagram ในส่วนของ View Layer

6.6.8 Class Diagram in Three-Tiered Architecture

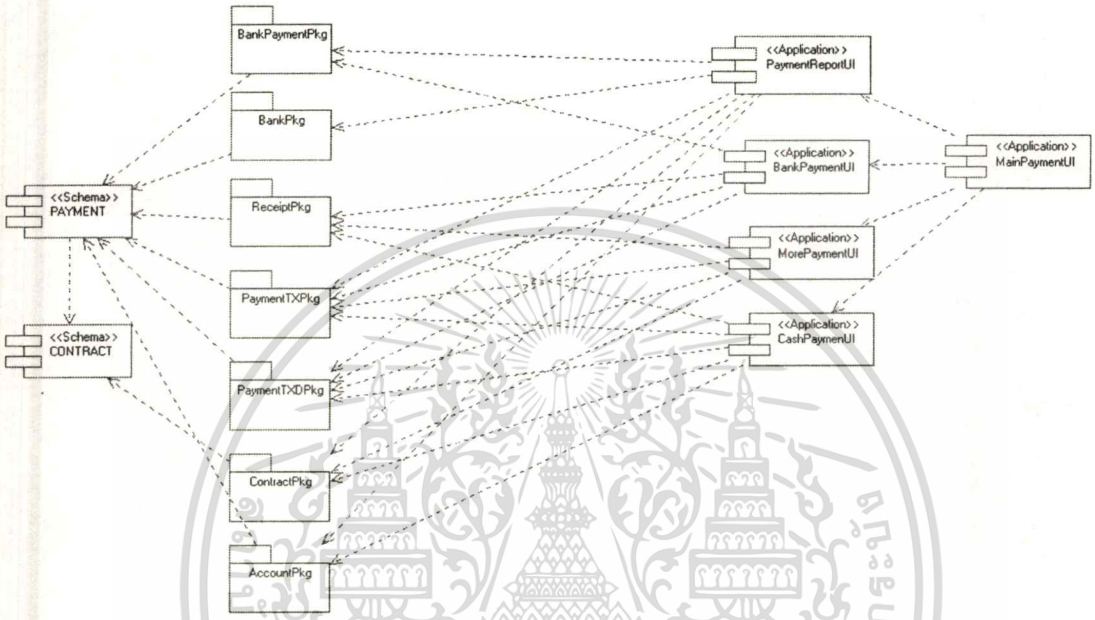


รูปที่ 6. 40 แสดง Class Diagram ของระบบรับชำระในรูปแบบ Three-Tiered

6.7 Physical Architecture

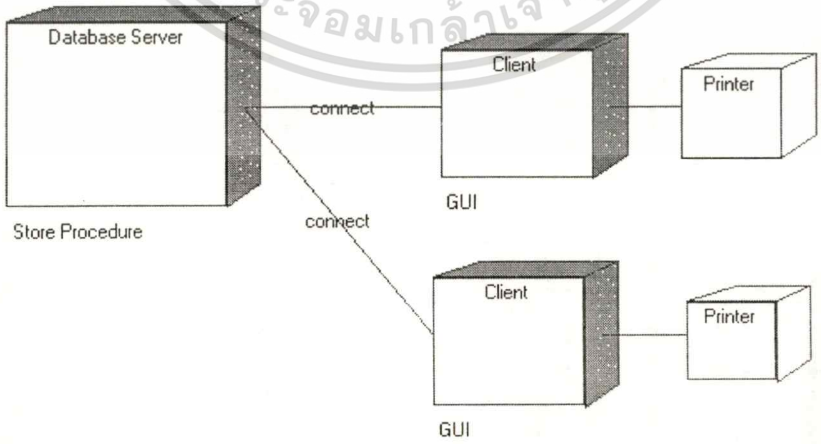
เป็นการแสดงสถาปัตยกรรมของซอฟต์แวร์ของระบบรับชำระ

6.7.1 Component Diagram



รูปที่ 6.41 แสดง Component Diagram ของระบบรับชำระ

6.7.2 Deployment Diagram



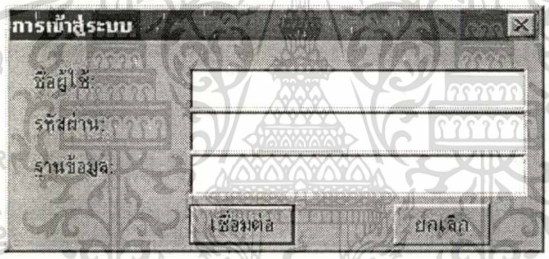
รูปที่ 6.42 แสดง Deployment Diagram ของระบบรับชำระ

บทที่ 7

คู่มือผู้ใช้งาน

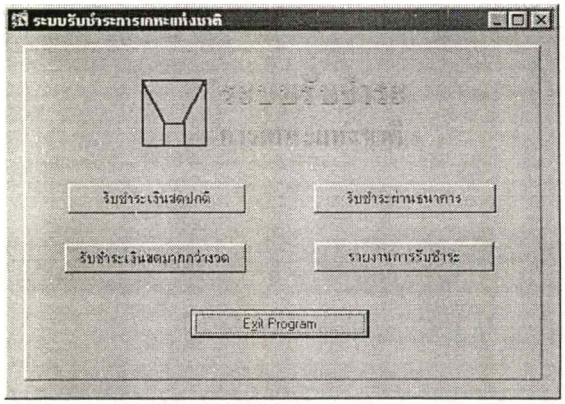
7.1 วิธีการเข้าสู่โปรแกรม

ก่อนที่ผู้ใช้งานจะสามารถทำงานกับโปรแกรมนี้ได้ ผู้ใช้งานจะต้องติดตั้งชุดโปรแกรมที่เรียกว่า Oracle Client ลงบนเครื่องที่ต้องการทำงานเสียก่อน หลังจากมีการติดตั้งชุดโปรแกรม Oracle Client ก็จะสามารถทำงานกับโปรแกรมนี้ได้ โดยขั้นตอนแรกเมื่อเริ่มเข้าสู่โปรแกรม จะปรากฏหน้าจอดังรูปที่ 7.1 ซึ่งเป็นรูปภาพที่แสดงการ Login เข้าสู่ระบบ



รูปที่ 7.1 แสดงการ Login เข้าสู่ระบบ

เมื่อหน้าจอปรากฏดังรูปที่ 7.1 นี้ให้ผู้ใช้งานใส่ชื่อผู้ใช้งาน (User Name) รหัสผ่าน (Password) และฐานข้อมูล (Host String) ลงไปเมื่อทั้งหมดถูกต้องก็จะสามารถเข้าสู่โปรแกรมได้ และจะแสดงภาพขึ้นมาดังรูปที่ 7.2 ซึ่งเป็นภาพที่แสดงหน้าจอหลักของโปรแกรม

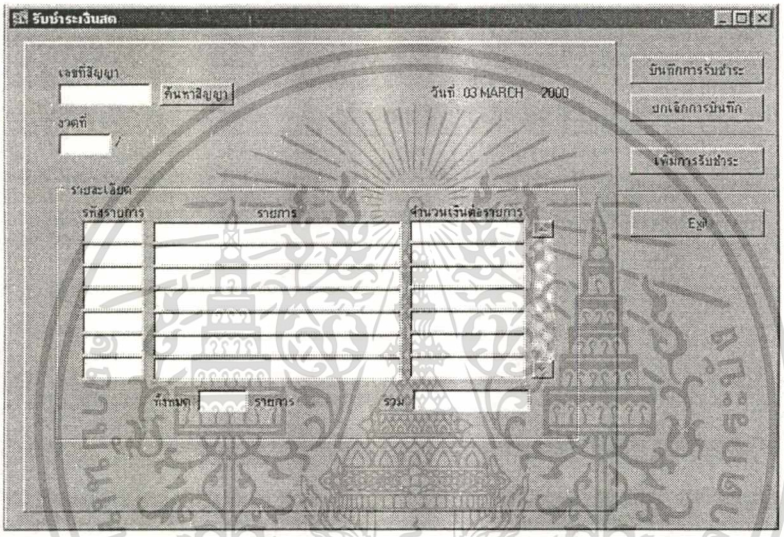


รูปที่ 7.2 แสดงหน้าต่างหลักของโปรแกรม

จากนั้นผู้ใช้งานสามารถจะเลือกทำงานที่ต้องการได้โดยการเลือกปุ่มต่างๆ (ซึ่งจะอธิบายวิธีการทำงานในแต่ละหัวข้อในช่วงต่อไป) หรือจะออกจากโปรแกรมโดยการเลือกที่ปุ่ม "Exit Program"

7.2 วิธีการรับชำระเงินสดปกติ

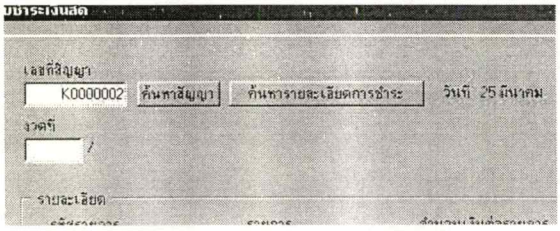
จากหน้าจอหลัก (รูปที่ 7.2) เลือกที่ปุ่ม "รับชำระเงินสดปกติ" ซึ่งโปรแกรมจะแสดงดังรูปที่ 7.3 ซึ่งเป็นรูปที่แสดงหน้าจอเกี่ยวกับการรับชำระเงินสดแบบปกติ



รูปที่ 7.3 แสดงหน้าต่างของการรับชำระเงินสดแบบปกติ

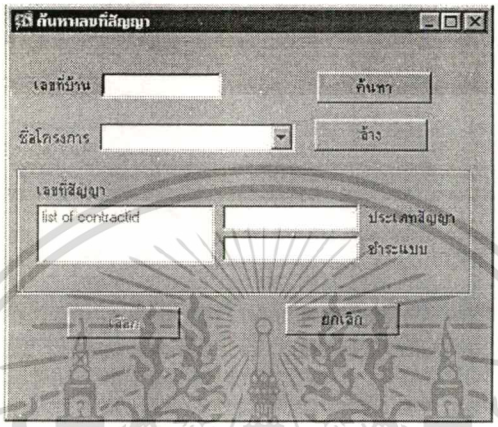
การรับชำระเงินสดแบบปกติจะวิธีการดังนี้

1. ใส่เลขที่สัญญาของลูกหนี้ลงในช่องเลขที่สัญญา แล้วกด Enter หากเลขที่สัญญานี้ถูกต้อง แล้วโปรแกรมจะมีการแสดงปุ่ม "ค้นหารายละเอียดการชำระ" ขึ้นมาดังรูปที่ 7.4



รูปที่ 7.4 แสดงปุ่ม "ค้นหารายละเอียดการชำระ" ที่ปรากฏขึ้นมาใหม่

ในกรณีที่ไม่ทราบเลขที่สัญญา ผู้ใช้งานอาจทำการค้นหาเลขที่สัญญาโดยเลือกที่ปุ่ม “ค้นหาสัญญา” ซึ่งหลังจากการเลือกที่ปุ่ม “ค้นหาสัญญา” แล้วจะปรากฏหน้าต่างดังรูปที่ 7.5 ซึ่งเป็นหน้าต่างที่ช่วยอำนวยความสะดวกในการค้นหาเลขที่สัญญาจาก เลขที่บ้านและชื่อโครงการ



รูปที่ 7.5 แสดงหน้าต่างการค้นหาเลขที่สัญญา

ให้ผู้ใช้ใส่เลขที่บ้านและเลือกชื่อโครงการที่ต้องการค้นหา และเลือกที่ปุ่ม “ค้นหา” หากเลขที่บ้านและชื่อโครงการที่เลือกนี้ถูกต้องทางโปรแกรมจะแสดงเลขที่สัญญาในช่องเลขที่สัญญา และให้ผู้ใช้เลือกเลขที่สัญญาในช่องเลขที่สัญญา และให้ผู้ใช้เลือกที่ปุ่ม “เลือก” จากนั้นโปรแกรมจะทำการแสดงเลขที่สัญญาในช่องเลขที่สัญญาในหน้าจอของการรับชำระเงินสดแบบปกติโดยอัตโนมัติ

2. ทำการค้นหารายละเอียดของการชำระของสัญญาเลขที่ดังกล่าวโดยการเลือกที่ปุ่ม “ค้นหารายละเอียดการชำระ” และโปรแกรมจะทำการแสดงรายละเอียดของการชำระขึ้นมา
3. หลังจากลูกหนี้ทำการชำระเงิน ผู้ใช้งานจะทำการบันทึกการรับชำระโดยเลือกที่ปุ่ม “บันทึกการรับชำระ” และหลังจากทำการเลือกที่ปุ่ม “บันทึกการรับชำระ” แล้วโปรแกรมจะแสดงปุ่ม “พิมพ์ใบเสร็จ” ขึ้นมาดังรูปที่ 7.6 เพื่อให้ผู้ใช้งานทำการพิมพ์ใบเสร็จรับเงินให้แก่ลูกหนี้

รูปที่ 7.6 แสดงปุ่ม “พิมพ์ใบเสร็จ” ที่ปรากฏขึ้นมาใหม่

หากเกิดเหตุผิดพลาดในการชำระเงินของลูกหนี้หรือกรณีใดๆ ระหว่างนี้ผู้ใช้สามารถทำการยกเลิกการบันทึกได้ โดยเลือกที่ปุ่ม “ยกเลิกการบันทึก”

** ในโปรแกรมนี้จะยังไม่ทำการบันทึกลงในระบบฐานข้อมูลจริงจนกว่าผู้ใช้งานจะมีการพิมพ์ใบเสร็จรับเงินให้ลูกหนี้

4. ผู้ใช้งานจะพิมพ์ใบเสร็จโดยเลือกที่ปุ่ม “พิมพ์ใบเสร็จ” โปรแกรมจะแสดงผลการประมวลผลการพิมพ์ใบเสร็จและแสดงใบเสร็จที่พร้อมจะพิมพ์ ดังรูปที่ 7.7 และรูปที่ 7.8 ตามลำดับ

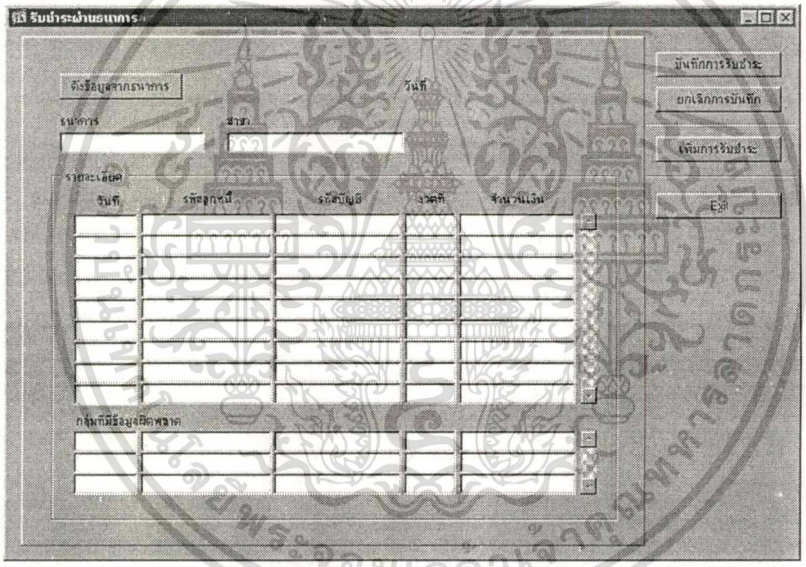
รูปที่ 7.7 แสดงความก้าวหน้าของการประมวลผลการพิมพ์ใบเสร็จ

รูปที่ 7.8 แสดงใบเสร็จที่พร้อมจะพิมพ์

5. หลังจากพิมพ์ใบเสร็จให้ลูกหนี้แล้วจะทำการปิดหน้าจอของการพิมพ์ใบเสร็จลงไป หากผู้ใช้งานต้องการจะทำการรับชำระกับลูกหนี้รายต่อไป ก็จะทำได้โดยการเลือกที่ปุ่ม "เพิ่มการรับชำระ" และทำการปฏิบัติการรับชำระในลักษณะเดิม หรือหากผู้ใช้งานต้องการออกจากการรับชำระเงินสดแบบปกติ ก็ทำได้โดยการเลือกที่ปุ่ม "Exit" และจะกลับมาที่หน้าจอหลักของโปรแกรม

7.3 วิธีการรับชำระผ่านธนาคาร

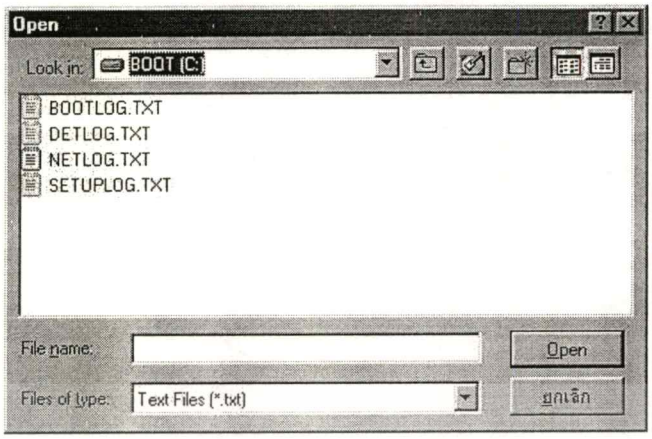
จากหน้าจอหลัก (รูปที่ 7.2) เลือกที่ปุ่ม "รับชำระผ่านธนาคาร" ซึ่งโปรแกรมจะแสดงดังรูปที่ 7.9 ซึ่งเป็นรูปที่แสดงหน้าจอเกี่ยวกับการรับชำระผ่านธนาคาร



รูปที่ 7.9 แสดงหน้าต่างของการรับชำระผ่านธนาคาร

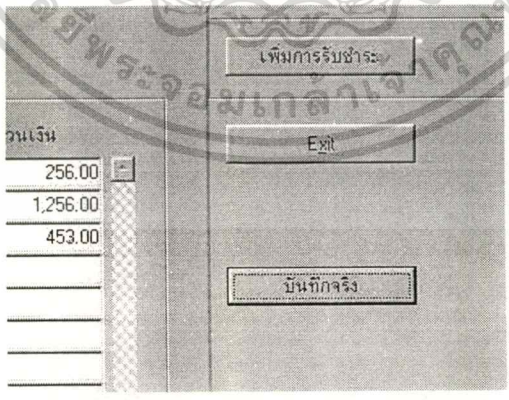
การรับชำระผ่านธนาคารจะมีวิธีการดังนี้

1. ให้ผู้ใช้งานนำแผ่นข้อมูล (Diskette) ที่ทางธนาคารส่งมาให้ใส่ในเครื่อง
2. ทำการดึงข้อมูลจากแผ่นข้อมูลของธนาคาร โดยการเลือกที่ปุ่ม "ดึงข้อมูลจากธนาคาร" หลังจากเลือกที่ปุ่มนี้แล้วทางโปรแกรมจะทำการแสดงหน้าต่างดังรูปที่ 7.10 ซึ่งแสดงหน้าต่างที่ให้ระบุตำแหน่งที่อยู่ของไฟล์ข้อมูล



รูปที่ 7.10 แสดงหน้าต่างที่ให้ระบุตำแหน่งที่อยู่ของไฟล์ข้อมูล

3. ให้ผู้ใช้งานระบุตำแหน่งที่อยู่ของไฟล์ โดยการเลือกตำแหน่งไดร์ฟ และไฟล์ข้อมูล และจากนั้นเลือกที่ปุ่ม "Open" จากนั้นโปรแกรมจะทำการตรวจสอบข้อมูลที่ดึงขึ้นมา หากข้อมูลใดเกิดผิดพลาด (ในกรณีของเลขที่สัญญาผิด หรือจำนวนเงินผิด) จะถูกแสดงอยู่ในส่วนของ "กลุ่มที่มีข้อมูลผิดพลาด" พร้อมกับบอกสาเหตุที่ผิดในช่องซ้ายสุด สำหรับข้อมูลที่ถูกต้องจะถูกแสดงในส่วนปกติ
4. ทำการบันทึกโดย ให้ผู้ใช้เลือกที่ปุ่ม "บันทึกการรับชำระ" และเมื่อเลือกที่ปุ่มนี้แล้วทางโปรแกรมจะแสดงปุ่ม "บันทึกจริง" ขึ้นมาดังรูปที่ 7.11 เพื่อให้ผู้ใช้งานยืนยันการบันทำจริง



รูปที่ 7.11 แสดงปุ่ม "บันทึกจริง" ที่ปรากฏขึ้นมาใหม่

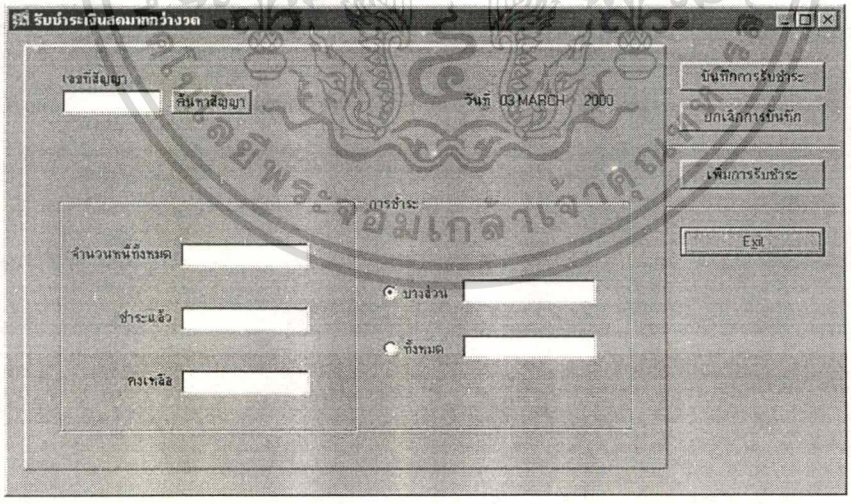
หากเกิดเหตุผิดพลาดในกรณีใดๆ ระหว่างนี้ผู้ใช้สามารถทำการยกเลิกการบันทึกได้ โดยเลือกที่ปุ่ม "ยกเลิกการบันทึก"

** ในโปรแกรมนี้จะยังไม่ทำการบันทึกลงในระบบฐานข้อมูลจริงจนกว่าผู้ใช้งานจะทำการบันทึกจริงเสียก่อน

5. ผู้ใช้งานสามารถทำการบันทึกจริงได้โดย การเลือกที่ปุ่ม "บันทึกจริง" ซึ่ง ณ ขณะนี้ทางโปรแกรมจะทำการบันทึกการรับชำระผ่านธนาคารลงในระบบฐานข้อมูลจริง จะไม่สามารถยกเลิกการบันทึกได้
6. หากผู้ใช้งานต้องการจะทำการรับชำระผ่านธนาคารในสาขาอื่น หรือธนาคารอื่น ก็สามารถทำได้โดยการเลือกปุ่ม "เพิ่มการรับชำระ" และทำการปฏิบัติในขั้นตอนเดิม หรือกรณีที่ต้องการออกจากการรับชำระผ่านธนาคารก็ทำได้โดยการเลือกที่ปุ่ม "Exit" และทางโปรแกรมจะกลับมาที่หน้าต่างหลักของโปรแกรม
7. กรณีที่ต้องการพิมพ์ใบเสร็จรับเงินของลูกหนี้ที่ชำระผ่านธนาคาร จะอธิบายในเรื่องของการพิมพ์รายงานการรับชำระในส่วนการพิมพ์ใบเสร็จ (ผ่านธนาคาร)

7.4 วิธีการรับชำระเงินสดมากกว่างวด

จากหน้าจอหลัก (รูปที่ 7.2) เลือกที่ปุ่ม "รับชำระเงินสดมากกว่างวด" ซึ่งโปรแกรมจะแสดงดังรูปที่ 7.12 ซึ่งเป็นรูปที่แสดงหน้าจอเกี่ยวกับการรับชำระเงินสดมากกว่างวด



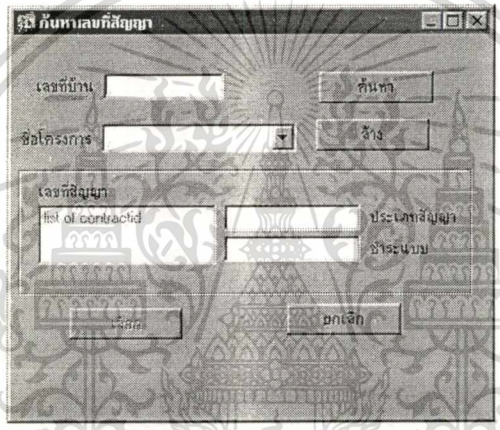
รูปที่ 7.12 แสดงหน้าต่างการรับชำระเงินสดมากกว่างวด

กรณีที่จะเกิดการชำระเงินสดมากกว่างวดของลูกหนี้นี้จะต้องมีกระบวนการการยืนยันของลูกหนี้กับทางการเคหะมาก่อนแล้วจึงจะทำการรับชำระเงินสดกรณีนี้ได้ ซึ่งกระบวนการดังกล่าวจะเกิดขึ้นในส่วนของระบบสัญญาของทางการเคหะ

การรับชำระเงินสดมากกว่างวดจะวิธีการดังนี้

1. ใส่เลขที่สัญญาของลูกหนี้ลงในช่องเลขที่สัญญา แล้วกด Enter หากเลขที่สัญญานี้ถูกต้อง แล้วโปรแกรมจะทำการคำนวณหาจำนวนหนี้คงเหลือ และแสดงจำนวนหนี้ทั้งหมด จำนวนเงินที่ชำระแล้ว และจำนวนเงินคงเหลือขึ้นมาให้โดยอัตโนมัติ

   ในกรณีที่ไม่ทราบเลขที่สัญญา ผู้ใช้งานอาจทำการค้นหาเลขที่สัญญาโดยเลือกที่ปุ่ม "ค้นหาสัญญา" ซึ่งหลังจากการเลือกที่ปุ่ม "ค้นหาสัญญา" แล้วจะปรากฏหน้าต่างดังรูปที่ 7.13 ซึ่งเป็นหน้าต่างที่ช่วยอำนวยความสะดวกในการค้นหาเลขที่สัญญาจาก เลขที่บ้าน และชื่อโครงการ

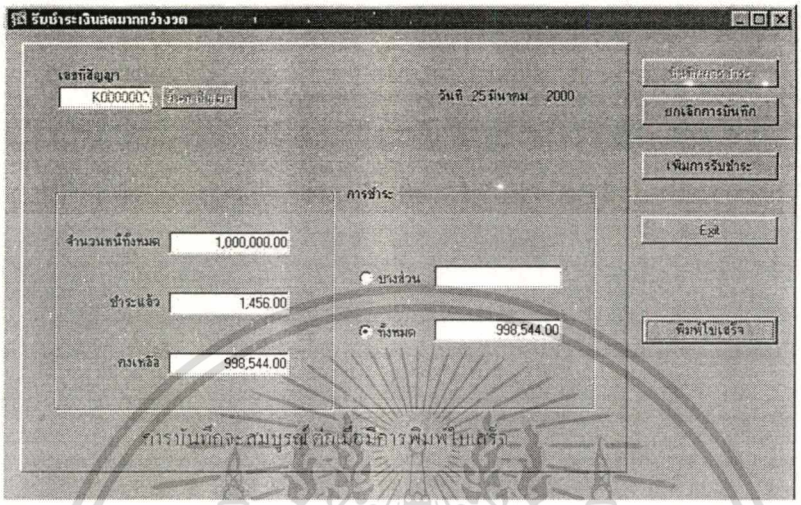


รูปที่ 7.13 แสดงหน้าต่างการค้นหาเลขที่สัญญา

ให้ผู้ใช้ใส่เลขที่บ้านและเลือกชื่อโครงการที่ต้องการค้นหา และเลือกที่ปุ่ม "ค้นหา" หากเลขที่บ้านและชื่อโครงการที่เลือกนี้ถูกต้องทางโปรแกรมจะแสดงเลขที่สัญญาในช่องเลขที่สัญญา และให้ผู้ใช้เลือกเลขที่สัญญาในช่องเลขที่สัญญา และให้ผู้ใช้เลือกที่ปุ่ม "เลือก" จากนั้นโปรแกรมจะทำการแสดงเลขที่สัญญาในช่องเลขที่สัญญาในหน้าจอของการรับชำระเงินสดแบบปกติโดยอัตโนมัติ

2. ให้ผู้ใช้ทำการเลือกการชำระ "บางส่วน" หรือ "ทั้งหมด" (ขึ้นอยู่กับข้อตกลงของลูกหนี้)
   - หากเป็นกรณีที่ทำการชำระบางส่วนก็ให้ผู้ใช้งานเลือกที่ "บางส่วน" และระบุจำนวนเงินที่ลูกหนี้จะชำระลงไปในช่องข้างๆ
   - หากเป็นกรณีที่ทำการชำระทั้งหมดก็ให้ผู้ใช้งานเลือกที่ "ทั้งหมด" และทางโปรแกรมจะนำจำนวนเงินคงเหลือมาใส่ไว้ที่ช่องข้างๆ โดยอัตโนมัติ
3. หลังจากลูกหนี้ทำการชำระเงิน ผู้ใช้งานจะทำการบันทึกการรับชำระโดยเลือกที่ปุ่ม "บันทึกการรับชำระ" และหลังจากทำการเลือกที่ปุ่ม "บันทึกการรับชำระ" แล้วโปรแกรม

จะแสดงปุ่ม “พิมพ์ใบเสร็จ” ขึ้นมาดังรูปที่ 7.14 เพื่อให้ผู้ใช้งานทำการพิมพ์ใบเสร็จรับเงินให้แก่ลูกหนี้

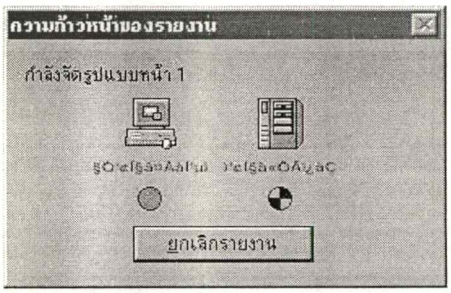


รูปที่ 7.14 แสดงปุ่ม “พิมพ์ใบเสร็จ” ที่ปรากฏขึ้นมาใหม่

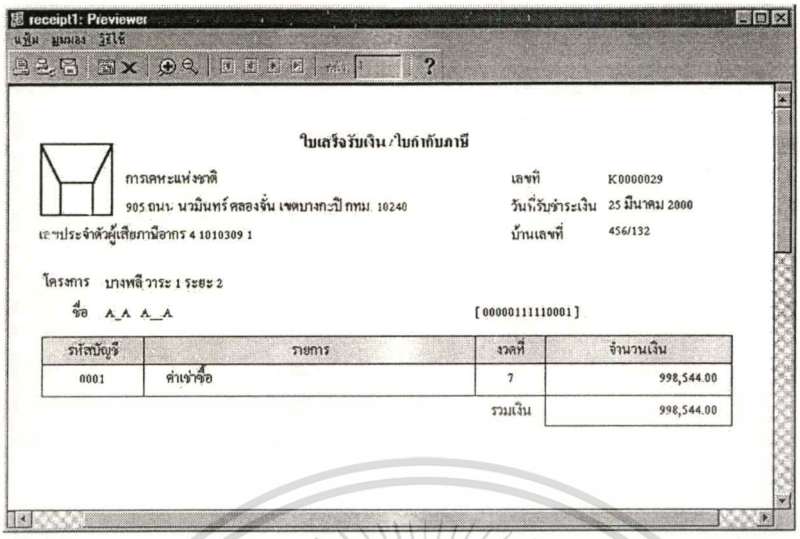
หากเกิดเหตุผิดพลาดในการชำระเงินของลูกหนี้หรือกรณีใดๆ ระหว่างนี้ผู้ใช้สามารถทำการยกเลิกการบันทึกได้ โดยเลือกที่ปุ่ม “ยกเลิกการบันทึก”

** ในโปรแกรมนี้จะยังไม่ทำการบันทึกลงในระบบฐานข้อมูลจริงจนกว่าผู้ใช้งานจะมีการพิมพ์ใบเสร็จรับเงินให้ลูกหนี้

4. ผู้ใช้งานจะพิมพ์ใบเสร็จโดยเลือกที่ปุ่ม “พิมพ์ใบเสร็จ” โปรแกรมจะแสดงผลการประมวลผลการพิมพ์ใบเสร็จและแสดงใบเสร็จที่พร้อมจะพิมพ์ ดังรูปที่ 7.15 และรูปที่ 7.16 ตามลำดับ



รูปที่ 7.15 แสดงความก้าวหน้าของการประมวลผลการพิมพ์ใบเสร็จ

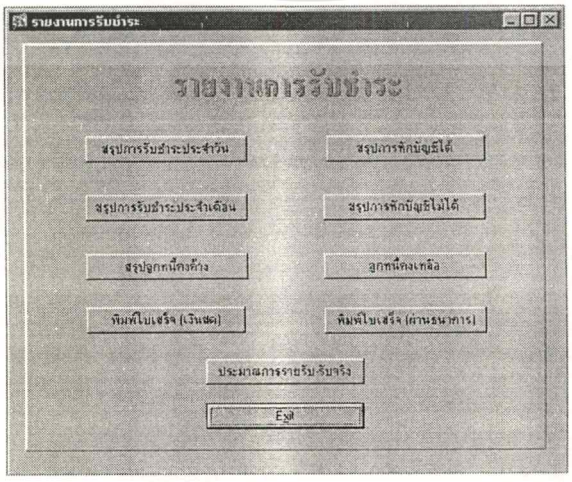


รูปที่ 7.16 แสดงใบเสร็จที่พร้อมจะพิมพ์

5. หลังจากพิมพ์ใบเสร็จให้ลูกหนี้แล้วจะทำการปิดหน้าจอของการพิมพ์ใบเสร็จลงไป หากผู้ใช้งานต้องการจะทำการรับชำระกับลูกหนี้รายต่อไป ก็จะทำได้โดยการเลือกที่ปุ่ม "เพิ่มการรับชำระ" และทำการปฏิบัติการรับชำระในลักษณะเดิม หรือหากผู้ใช้งานต้องการออกจากการรับชำระเงินสดมากกว่างวด ก็ทำได้โดยการเลือกที่ปุ่ม "Exit" และจะกลับมาที่หน้าจอหลักของโปรแกรม

7.5 วิธีการพิมพ์รายงานการรับชำระ

จากหน้าจอหลัก (รูปที่ 7.2) เลือกที่ปุ่ม "รายงานการรับชำระ" ซึ่งโปรแกรมจะแสดงดังรูปที่ 7.17 ซึ่งเป็นรูปที่แสดงหน้าจอเกี่ยวกับการพิมพ์รายงานการรับชำระ

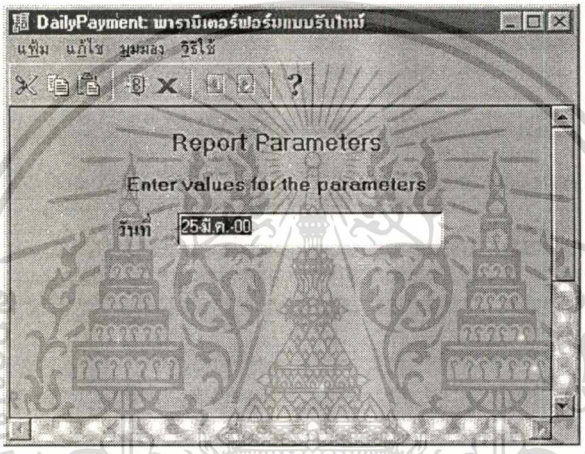


รูปที่ 7.17 แสดงหน้าต่างการพิมพ์รายงานการรับชำระ

** รูปแบบวันที่ที่ใส่ลงในโปรแกรมเป็นรูปแบบของ วัน/เดือน/ปี (DD/MM/YY) ในลักษณะตัวเลข

7.5.1 การพิมพ์รายงานการรับชำระ-สรุปการรับชำระประจำวัน

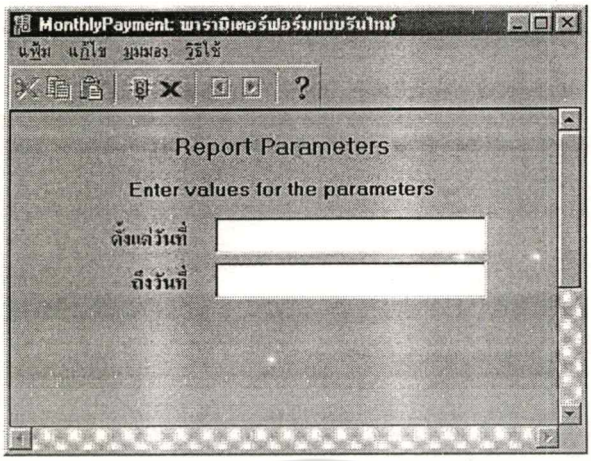
จากรูปที่ 7.17 เลือกปุ่ม "สรุปการรับชำระประจำวัน" หลังจากนั้นจะปรากฏหน้าต่างดังรูปที่ 7.18 สำหรับรับค่าวันที่ที่ต้องการพิมพ์รายงานรับชำระ ทางโปรแกรมจะแสดงวันที่ปัจจุบันเป็นค่าเริ่มต้น (หากต้องการเปลี่ยนก็ใส่วันที่ที่ต้องการลงไป) หลังจากนั้นกด Enter ก็ได้รายงานสรุปการรับชำระประจำวัน สำหรับวันที่ที่ผู้ใช้ระบุลงไป



รูปที่ 7.18 แสดงหน้าต่างสำหรับรับค่าวันที่ของรายงานสรุปการรับชำระประจำวัน

7.5.2 การพิมพ์รายงานการรับชำระ-สรุปการรับชำระประจำเดือน

จากรูปที่ 7.17 เลือกปุ่ม "สรุปการรับชำระประจำเดือน" หลังจากนั้นจะปรากฏหน้าต่างดังรูปที่ 7.19 สำหรับรับช่วงของวันที่ที่ต้องการพิมพ์รายงานรับชำระ เช่นหากต้องการพิมพ์รายเงินเดือน มีนาคม ก็อาจจะใส่ 11/03/00 และ 10/04/00 หลังจากนั้นกด Enter ก็ได้รายงานสรุปการรับชำระประจำเดือน สำหรับช่วงวันที่ที่ผู้ใช้ระบุลงไป



รูปที่ 7.19 แสดงหน้าต่างสำหรับรับช่วงของวันที่ของรายงานสรุปการรับชำระประจำเดือน

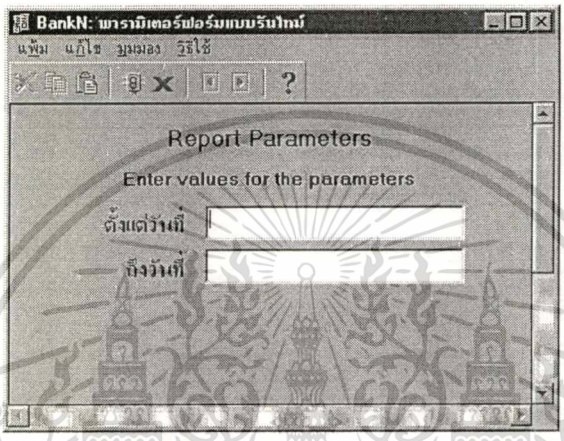
7.5.3 การพิมพ์รายงานการรับชำระ-สรุปการหักบัญชีได้

จากรูปที่ 7.17 เลือกปุ่ม "สรุปการหักบัญชีได้" หลังจากนั้นจะปรากฏหน้าต่างดังรูปที่ 7.20 สำหรับรับช่วงของวันที่ที่ต้องการพิมพ์รายงาน เช่นหากต้องการพิมพ์รายเงินเดือน มีนาคม ก็อาจจะใส่ 11/03/00 และ 10/04/00 หลังจากนั้นกด Enter ก็ได้รายงานสรุปการหักบัญชีได้ สำหรับช่วงวันที่ที่ผู้ใช้ระบุลงไป

รูปที่ 7.20 แสดงหน้าต่างสำหรับรับช่วงของวันที่ของรายงานสรุปการหักบัญชีได้

7.5.4 การพิมพ์รายงานการรับชำระ-สรุปการหักบัญชีไม่ได้

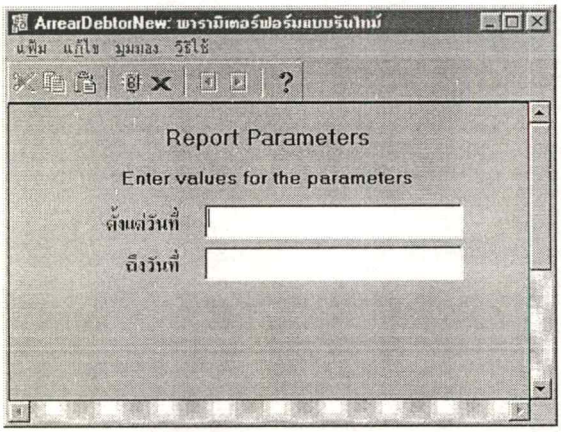
จากรูปที่ 7.17 เลือกปุ่ม "สรุปการหักบัญชีไม่ได้" หลังจากนั้นจะปรากฏหน้าต่างดังรูปที่ 7.21 สำหรับรับช่วงของวันที่ที่ต้องการพิมพ์รายงาน เช่นหากต้องการพิมพ์รายเงินเดือน มีนาคม ก็อาจจะใส่ 11/03/00 และ 10/04/00 หลังจากนั้นกด Enter ก็ได้รายงานสรุปการหักบัญชีไม่ได้สำหรับช่วงวันที่ที่ผู้ใช้ระบุลงไป



รูปที่ 7.21 แสดงหน้าต่างสำหรับรับช่วงของวันที่ของรายงานสรุปการหักบัญชีไม่ได้

7.5.5 การพิมพ์รายงานการรับชำระ-สรุปลูกหนี้คงค้าง

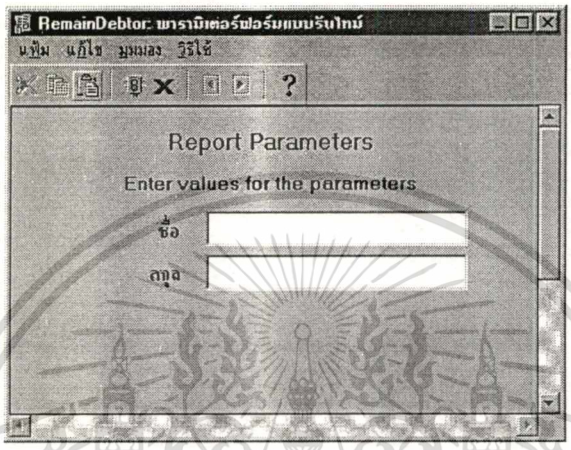
จากรูปที่ 7.17 เลือกปุ่ม "สรุปลูกหนี้คงค้าง" หลังจากนั้นจะปรากฏหน้าต่างดังรูปที่ 7.22 สำหรับรับช่วงของวันที่ที่ต้องการพิมพ์รายงาน เช่นหากต้องการพิมพ์รายเงินเดือน มีนาคม ก็อาจจะใส่ 11/03/00 และ 10/04/00 หลังจากนั้นกด Enter ก็ได้รายงานสรุปลูกหนี้คงค้าง สำหรับช่วงวันที่ที่ผู้ใช้ระบุลงไป



รูปที่ 7.22 แสดงหน้าต่างสำหรับรับช่วงของวันที่ของรายงานสรุปลูกหนี้คงค้าง

7.5.6 การพิมพ์รายงานการรับชำระ-ลูกหนี้คงเหลือ

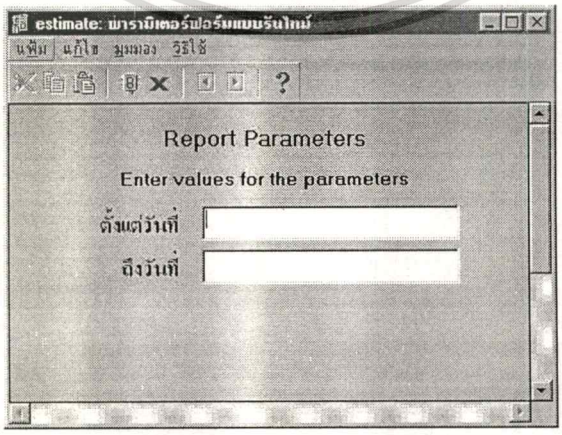
จากรูปที่ 7.17 เลือกปุ่ม "ลูกหนี้คงเหลือ" หลังจากนั้นจะปรากฏหน้าต่างดังรูปที่ 7.23 สำหรับรับชื่อและสกุลของลูกหนี้ที่ต้องการออกรายงาน หลังจากนั้นกด Enter ก็ได้รายงานลูกหนี้คงเหลือ สำหรับชื่อและสกุลที่ผู้ใช้ระบุลงไป



รูปที่ 7.23 แสดงหน้าต่างสำหรับรับชื่อและสกุลของรายงานลูกหนี้คงเหลือ

7.5.7 การพิมพ์รายงานการรับชำระ-ประมาณการรายรับ-รับจริง

จากรูปที่ 7.17 เลือกปุ่ม "ประมาณการรายรับ-รับจริง" หลังจากนั้นจะปรากฏหน้าต่างดังรูปที่ 7.24 สำหรับรับช่วงของวันที่ที่ต้องการพิมพ์รายงาน เช่นหากต้องการพิมพ์รายเงินเดือน มีนาคม ก็อาจจะใส่ 11/03/00 และ 10/04/00 หลังจากนั้นกด Enter ก็ได้รายงานประมาณการรายรับ-รับจริง สำหรับช่วงวันที่ที่ผู้ใช้ระบุลงไป



รูปที่ 7.24 แสดงหน้าต่างสำหรับรับช่วงของวันที่ของรายงานประมาณการรายรับ-รับจริง

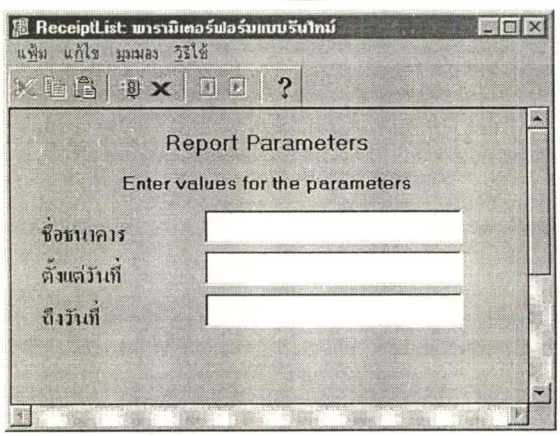
7.5.8 การพิมพ์รายงานการรับชำระ-พิมพ์ใบเสร็จ (เงินสด)

จากรูปที่ 7.17 เลือกปุ่ม "พิมพ์ใบเสร็จ (เงินสด)" หลังจากนั้นจะปรากฏหน้าต่างดังรูปที่ 7.25 สำหรับรับเลขที่ใบเสร็จที่ต้องการพิมพ์ หลังจากนั้นกด Enter ก็ได้ใบเสร็จเลขที่ที่ผู้ใช้ระบุลงไป

รูปที่ 7.25 แสดงหน้าต่างสำหรับรับเลขที่ใบเสร็จของการพิมพ์ใบเสร็จ (เงินสด)

7.5.9 การพิมพ์รายงานการรับชำระ-พิมพ์ใบเสร็จ (ผ่านธนาคาร)

จากรูปที่ 7.17 เลือกปุ่ม "พิมพ์ใบเสร็จ (ผ่านธนาคาร)" หลังจากนั้นจะปรากฏหน้าต่างดังรูปที่ 7.26 สำหรับรับชื่อของธนาคารและช่วงของวันที่ที่ต้องการพิมพ์ใบเสร็จ เช่นหากต้องการพิมพ์ใบเสร็จของธนาคาร "ไทยพานิชย์" เงินเดือน มีนาคม ก็อาจจะใส่ชื่อธนาคาร ไทยพานิชย์ 11/03/00 และ 10/04/00 ตามลำดับ หลังจากนั้นกด Enter ก็ได้ชุดของใบเสร็จ สำหรับธนาคารและช่วงวันที่ที่ผู้ใช้ระบุลงไป



รูปที่ 7.26 แสดงหน้าต่างสำหรับรับชื่อธนาคารและช่วงของวันที่ ที่จะพิมพ์ใบเสร็จ (ผ่านธนาคาร)

บรรณานุกรม

Bahrami, Ali. 1998. Object Oriented Systems Development using Unified Modeling Language. Singapore: McGraw-Hill.

Coad, P. and Yourdon, E. 1991. Object-Oriented Design. New Jersey: Yourdon Press.

Dittman, Kevin C. 1997. Object Modeling Systems Analysis and Design. Irwin McGraw-Hill.

Eliksson, Hans-Erik and Magnus Penker. 1998. UML Toolkit. John Wiley & Sons.

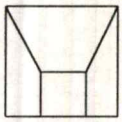
Montgomery, Stephen L. 1998. Building Object-Oriented Software. McGraw-Hill.

Quatrani, Terry. 1998. Visual Modeling with Rational Rose and UML. Wesley Longman.

Wirfs-Brock, Rebecca., Brian Wilerson And Lauren Wiener. 1992. Designing Object-Oriented Software. New Jersey: Prentice-Hall.

ภาคผนวก

รายงานของระบบรับชำระซึ่งจะประกอบด้วย ใบเสร็จรับเงิน สรุปการรับชำระเงินประจำวัน สรุปการรับชำระเงินประจำเดือน รายงานลูกหนี้คงเหลือ รายงานการประมาณรายรับ-รับจริง สรุปลูกหนี้คงค้าง สรุปการหักบัญชีได้ สรุปการหักบัญชีไม่ได้ ซึ่งจะสามารถแสดงได้ดังนี้



การเคหะแห่งชาติ **ใบเสร็จรับเงิน/ใบกำกับภาษี**

905 ถนนนวมินทร์ คลองจั่น เขตบางกะปิ กทม. 10240

เลขประจำตัวผู้เสียภาษีอากร 4 1010309 1

เลขที่ K1001010

วันที่รับชำระเงิน 10/11/2542

บ้านเลขที่ 0001/0001

โครงการ เช่า ร่มเกล้า ระยะ 1

ชื่อ สัญญา ดีจัง (100505 016100 0001/0001)

รหัสบัญชี	รายการ	งวด	จำนวนเงิน
0520	ค่าเช่า	4	1,500.00
0800	ค่าประกันการใช้น้ำ	4	100.00
		รวมเงิน	1,600.00

.....

พนักงานรับเงิน

การเคหะแห่งชาติ เป็นผู้ประกอบการในระบบภาษีมูลค่าเพิ่ม เฉพาะกิจการขายน้ำ

ใบเสร็จรับเงินฉบับนี้จะสมบูรณ์ต่อเมื่อ มีลายมือชื่อพนักงานรับเงิน

รายงานสรุปการรับชำระเงิน

วันที่ 10 มีนาคม 40

รหัสบัญชี	รายการ	ราย	จำนวนเงิน
0520	ค่าเช่า	259	416,000.00
0650	ค่าเช่าซื้อ	464	758,643.00
0800	ค่าประกันการใช้น้ำ	152	50,653.00
0850	ค่าประกันการใช้ไฟ	120	17,917.00
9061	ค่าปรับ	190	5,700.00
	ยอดรวมทั้งสิ้น	1185	1,248,913.00

รายงานสรุปการรับชำระเงิน

เดือน มีนาคม 2540

รหัสบัญชี	รายการ	ราย	จำนวนเงิน
0520	ค่าเช่า	3520	1,256,452.00
0650	ค่าเช่าซื้อ	2596	2,223,145.00
0800	ค่าประกันการใช้น้ำ	1200	265,235.00
0850	ค่าประกันการใช้ไฟ	865	117,917.00
9061	ค่าปรับ	320	9,600.00
	ยอดรวมทั้งสิ้น	8501	3,872,349.00

รายงานสรุปลูกหนี้คงค้าง

เดือน พฤษภาคม 2539

รหัสลูกหนี้	เลขที่สัญญา	ประเภทสัญญา	ชื่อลูกหนี้	จำนวนเงิน
105107 011180 0080/0158	C1002456	เช่าซื้อ	นายทวีศักดิ์ เจริญด้วยศีล	2,560.00
108104 015180 0095/1256	C1003568	เช่าซื้อ	นายนพพร ชำนิ	2,480.00
200104 015153 1025/1253	C4200125	เช่า	นางเจตนา ปันผล	2,958.00
101102 152360 4582/4586	C4200785	เช่า	นายระริน แซ่แอ๊ว	2,756.00
		รวม	4	10,754.00

รายงานลูกหนี้คงเหลือ

รหัสลูกหนี้ 100505 016100 0010/0001

วันที่ 10 / 3 /42

ชื่อ นายสัญญา คุณากร

เลขที่สัญญา	ประเภทสัญญา	อัตราดอกเบี้ย	จำนวนหนี้ทั้งหมด	ชำระแล้ว	เงินต้นคงเหลือ	ดอกเบี้ยคงเหลือ	จำนวนหนี้คงเหลือ
C5125001	เช่าซื้อ	8 %	1,250,000.00	850,000.00	368,000.00	32,000.00	400,000.00

รายงานการประมาณรายรับ-รับจริง

เดือน มีนาคม 2540

ประเภทสัญญา	จำนวนราย ทั้งหมด	จำนวนราย ที่ส่ง	ประมาณการ รายรับ	รายรับจริง	คงค้าง จำนวน ราย	คงค้าง จำนวนเงิน
เช่า	536	501	485,000.0	470,100.0	35	14,900.00
เช่าซื้อ	458	408	515,980.0	482,020.0	50	33,960.00
กู้ยืม	259	210	409,750.0	384,120.0	49	25,630.00
รวม	1,253	1,119	1,410,730.0	1,336,240.0	134	74,490.00

รายงานสรุปการหักบัญชีได้

เดือน ธันวาคม 2542

ธนาคาร กรุงเทพ

รหัสลูกหนี้	รหัสบัญชี	จำนวนเงิน
100505 016100 0010/0011	101-0-01463-5	2,935.00
100505 016100 0010/0125	101-0-02259-6	1,717.00
100505 016100 0010/1452	101-0-08935-9	854.00
101605 017100 0865/5632	101-0-12021-8	12,005.00
รวม	4 ราย	17,511.00

ธนาคาร ทหารไทย

รหัสลูกหนี้	รหัสบัญชี	จำนวนเงิน
100507 018108 0020/0084	054-0-43072-4	2,563.00
100507 018108 0020/0089	054-0-45918-6	5,610.00
100507 018108 0020/0098	054-0-47600-8	2,500.00
100507 018108 0020/0119	054-0-47815-2	1,950.00
รวม	4 ราย	12,623.00

รวมทั้งหมด 2 ธนาคาร	8 ราย	30,134.00

ประวัติผู้เขียน

ชื่อ นายวรพันธ์ วิสุทธิ

เกิด 20 พฤศจิกายน 2520

ประวัติการศึกษา

มัธยม สารวิทยา

อุดมศึกษา สถาบันราชภัฏจันทรเกษม

ประวัติการทำงาน

ผู้ช่วยวิทยากร ฝ่ายฝึกอบรม NECTEC

Technician ร้านค้า ComStreet พันธุ์ทิพย์พลาซ่า

ฝ่าย MIS (EDP) Tops Supermarket