การสร้างโปรแกรมประยุกต์เชิงวัตถุด้วย VisualAge

OBJECT – ORIENTED APPLICATION IMPLEMENTATION USING VISUALAGE

โดย

นาย วิโรจน์ ปลื้มวงศ์โรจน์

น.ส. อารยาภา ศิริพานิช

อาจารย์ที่ปรึกษา

ผศ. บรรจง ปิยธำรง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การสร้างโปรแกรมประยุกต์เชิงวัตถุด้วย VisualAge

OBJECT - ORIENTED APPLICATION IMPLEMENTATION USING VISUALAGE

ผู้จัดทำ

1. นายวิโรจน์ ปลื้มวงศ์โรจน์ รหัสประจำตัว 39014486
2. น.ส.อารยาภา ศิริพานิช รหัสประจำตัว 39014676

_____อาจารย์ที่ปรึกษา

(ผศ. บรรจง ปิยธำรง)

การสร้างโปรแกรมประยุกต์เชิงวัตถุด้วย VisualAge

นายวิโรจน์ ปลื้มวงศ์โรจน์ 39014486
น.ส.อารยาภา ศิริพานิช 39014676
ผศ. บรรจง ปิยธำรง อาจารย์ที่ปรึกษา
ปีการศึกษา 2542

บทคัดย่อ

การพัฒนาเชิงวัตถุ เป็นรูปแบบการพัฒนาที่มีข้อดีเหนือกว่าการพัฒนารูปแบบเดิมหลายประการ จึงทำให้เป็นแนวทางใหม่ที่เป็นที่นิยมกันอย่างแพร่หลายในปัจจุบัน มีโมเดลหลายชนิดที่นำเสนอรูปแบบการพัฒนาเชิงวัตถุ แต่ UML เป็นโมเดลมาตรฐานที่รวมเอาข้อดีของโมเดลต่างๆเข้าไว้และนำเสนอในรูปแบบที่เข้าใจง่ายทั้งผู้พัฒนาและผู้ใช้ระบบทำให้ได้ระบบที่ตรงกับความต้องการมากยิ่งขึ้น

การพัฒนาเชิงวัตถุต้องการภาษาที่สนับสนุนการเขียนโปรแกรมเชิงวัตถุ ซึ่งภาษาจาวาก็เป็นหนึ่งในภาษาที่สนับสนุนคุณสมบัตินี้ นอกจากนี้คุณลักษณะเฉพาะตัวที่ไม่ขึ้นกับแพลตฟอร์มของภาษาจาวาทำให้โปรแกรมที่เขียนด้วยภาษาจาวาสามารถใช้งานได้กับส่วนประมวลผลหรือระบบปฏิบัติใดๆ เหมาะอย่างยิ่งกับการทำงานผ่านเครือข่ายที่มีความหลากหลายของคอมพิวเตอร์

วิทยานิพนธ์ฉบับนี้กล่าวถึงทฤษฎีการพัฒนาเชิงวัตถุตามแนวทาง Visual Modeling Technique โดยได้จำลองระบบเงินกู้เพื่อการผ่อนบ้านจัดสรรมาเป็นระบบตัวอย่างในการพัฒนา นำเสนอโมเดลตามสัญลักษณ์ของ UML โดยใช้โปรแกรม Rational Rose ซึ่งเป็น CASE tool มาช่วยในการวิเคราะห์ความต้องการและออกแบบระบบ จากนั้นนำผลที่ได้จากขั้นตอนการวิเคราะห์และออกแบบไปสร้างเป็นโค้ดภาษาจาวาโดยใช้โปรแกรม VisualAge for Java และสร้างฐานข้อมูลรีเลชันแนลโดยใช้ออราเคิลเป็นระบบจัดการฐานข้อมูล

Object – Oriented Application Implementation Using VisualAge

Wiroj Pluemwongroj
Arayapha Siripanich
Asst.Prof. Banjong Piyathamrong Advisor

ABSTRACT

Object – Oriented Development has many advantages over other development methods that make it is a new widely - used style at this time. Many models present Object – Oriented Development. But UML which contains the good things of all models was used as the standard model. UML also presents in easy – understanding way , helping the system developers and users get a system that meets their requirements.

Object – Oriented Development needs the language that supports Object – Oriented Programming. Java Language is the one that has this qualification. Additionally, its " Platform Independent " characteristic lets Java programs run on any processors or operating systems , suitable for working with diverse computers on networks.

This thesis presents the theory of Visual Modeling Technique and shows The Housing Loan System as an example. Representing models with UML notations using Rational Rose in analysis and design phase. Then, take the results from analysis and design phase to construct the Java programs using VisualAge for Java and build the relational database with Oracle.

กิตติกรรมประกาศ

โครงงานและวิทยานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดี เนื่องจากผู้จัดทำได้รับความร่วมมือและความช่วยเหลือจากบุคคลต่างๆ หลายฝ่าย ทั้งทางด้านวิชาการและกำลังใจ ผู้จัดทำขอขอบพระคุณ

อาจารย์ที่ปรึกษา ผศ. บรรจง ปิยธำรง ที่กรุณาชี้แนะให้คำปรึกษา และดูแลให้โครงงานนี้สำเร็จได้ด้วยดี

คณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ทุกๆท่าน สำหรับความรู้และประสบการณ์ที่มีค่ายิ่ง ถึงแม้อาจารย์จะต้องรอส่งงานถึงตอนดึกหรือเข้ามาในวันหยุด อาจารย์ก็มีเวลาให้พวกเราเสมอ

เพื่อนๆ พี่ๆ ห้อง D และ P ผู้ร่วมชะตากรรมทำโครงงานเช่นกัน และได้ให้คำแนะนำปรึกษาต่างๆ อันเป็นประโยชน์ต่อพวกเราเป็นอย่างยิ่ง

คุณไพโรจน์ ร่วมวิบูลย์สุข IBM Software Specialist สำหรับคำแนะนำด้านเทคนิค

และบุคคลสำคัญที่สุดที่ทำให้เรามีวันนี้ คือ คุณพ่อ คุณแม่ และสมาชิกในครอบครัวทุกคน ที่คอยสนับสนุนและเป็นกำลังใจในทุกเรื่อง

นายวิโรจน์ ปลื้มวงศ์โรจน์

น.ส.อารยาภา ศิริพานิช

สารบัญ

สารบัญตาราง

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในอดีตการพัฒนาซอฟต์แวร์หรือระบบสารสนเทศนั้นจะใช้วิธีพัฒนาแบบโครงสร้าง (Structural Development) ซึ่งแสดงฟังก์ชันและขั้นตอนการทำงานของระบบโดยนำเสนอด้วย Data Flow Diagram หรือ Entity Relationship Diagram เป็นต้น แต่ไดอะแกรมเหล่านี้ก็มีข้อเสียตรงที่จะบอกเพียงแค่ว่าระบบมีลักษณะการทำงานอย่างไร ไม่ได้บอกถึงแนวทางในการพัฒนาหรือรูปแบบในการเขียนโปรแกรม ดังนั้นจึงได้มีผู้คิดค้นแนวทางการพัฒนาใหม่เกิดขึ้น

การพัฒนาเชิงวัตถุ (Object – Oriented Development) เป็นแนวทางในการพัฒนาที่ได้รับความนิยมอย่างแพร่หลายในปัจจุบัน วิธีนี้ผู้พัฒนาจะมองระบบเป็นออบเจ็กต์ (Object) โดยใช้มุมมองจากโลกแห่งความเป็นจริงทำให้สามารถเข้าใจในตัวระบบที่ทำการพัฒนาได้ชัดเจนยิ่งขึ้น นอกจากนี้การพัฒนาเชิงวัตถุยังมีข้อดีเหนือแนวทางการพัฒนารูปแบบอื่นดังนี้

- โมเดลมีความสอดคล้องกันในทุกขั้นตอนการพัฒนา ทั้งในขั้นตอนการวิเคราะห์,การออกแบบไปจนถึงการสร้าง โมเดลที่ได้จากขั้นตอนการวิเคราะห์สามารถนำมาเขียนเป็นโค้ดได้โดยตรง จึงช่วยให้ผู้พัฒนาเห็นถึงความสัมพันธ์ระหว่างปัญหากับวิธีการแก้ปัญหาได้ง่าย
- โมเดลเชิงวัตถุมีการแยกแยะเอกลักษณ์ (Abstraction) โดยเป็นการแยกเอาเฉพาะสิ่งที่สนใจออกมา แสดงคุณลักษณ์ที่สำคัญของออบเจ็กต์ และการซ่อนรายละเอียด (Encapsulation) ซึ่งเป็นการซ่อนข้อมูลและความซับซ้อนภายในออบเจ็กต์ เป็นลักษณะที่ดีในการพัฒนาซอฟต์แวร์
- เมื่อมีการเปลี่ยนแปลงความต้องการของระบบ ก็เป็นเพียงการเปลี่ยนแปลงภายในออบเจ็กต์หรือการเพิ่มออบเจ็กต์ โดยไม่ต้องเปลี่ยนแปลงโครงสร้างของระบบทั้งหมด
- สนับสนุนการนำกลับมาใช้ใหม่ (Reuse)
- สนับสนุนการปรับเปลี่ยนขนาดของระบบ เนื่องจากมีการแยกแยะเอกลักษณ์และซ่อนรายละเอียดของแต่ละออบเจ็กต์
- สนับสนุนการออกแบบระบบที่เชื่อถือได้และปลอดภัย เนื่องจากออบเจ็กต์แต่ละตัวสื่อสารกันผ่านส่วนติดต่อที่ผู้พัฒนากำหนดให้ไว้เท่านั้นจึงสามารถควบคุมการติดต่อระหว่างคอมโพเนนท์ได้
- สนับสนุนการทำงานพร้อมๆกัน (Concurrency)

วิธีการพัฒนาเชิงวัตถุประกอบไปด้วยขั้นตอนต่างๆ คือขั้นตอนการวิเคราะห์และออกแบบ โดยจะมองภาพรวมของฟังก์ชันการทำงานของระบบเป็นหลัก จากนั้นก็จะมากำหนดรายละเอียดของออบเจ็กต์,คลาส,ความสัมพันธ์และคุณลักษณะที่จะมีในระบบ และจะเข้าสู่ขั้นตอนการพัฒนาโดยใช้ภาษาที่สนับสนุนการเขียนโปรแกรมเชิงวัตถุ (Object – Oriented Programming Language) ซึ่งเป็นภาษาที่

สนับสนุนการสร้างคลาสและออบเจ็กต์ ทำให้สิ่งที่ได้จากการวิเคราะห์และออกแบบสามารถนำมาใช้ในการเขียนในโปรแกรมได้เกือบจะทันที

ภาษาจาวา (Java) ก็เป็นภาษาหนึ่งที่สนับสนุนการเขียนโปรแกรมเชิงวัตถุ ภาษาจาวาเริ่มเป็นที่สนใจเมื่อบริษัท Sun Microsystems ประกาศให้จาวาเป็นภาษาสำหรับสร้างโปรแกรมที่จะใช้งานบนอินเตอร์เน็ต โดยเน้นข้อได้เปรียบของภาษาจาวาที่โปรแกรมที่สร้างบนคอมพิวเตอร์เครื่องหนึ่งสามารถนำไปทำงานบนคอมพิวเตอร์เครื่องอื่นที่อาจจะมีหน่วยประมวลผลต่างกันโดยไม่ต้องคอมไพล์โปรแกรมใหม่ โดยนำเทคนิคการคอมไพล์โปรแกรมเป็นคำสั่งของหน่วยประมวลผลสมมติตัวหนึ่ง แล้วสร้างตัวแปล (interpreter) ให้กับหน่วยประมวลผลที่จะทำงานโปรแกรมนั้น ด้วยวิธีนี้ทำให้โปรแกรมไม่ผูกติดอยู่กับหน่วยประมวลผลรุ่นใดรุ่นหนึ่ง เรียกคุณสมบัตินี้ว่า Platform Independent

ภาษาจาวาถูกออกแบบโดยมีจุดมุ่งหมายดังนี้

- เป็นภาษาที่ง่าย มีกลไกภาษาที่ไม่ซับซ้อน ทำให้ผู้ใช้สามารถเรียนรู้ได้อย่างรวดเร็ว โดยนำไวยากรณ์ส่วนใหญ่มาจากภาษา C^{++} แต่ตัดกลไกที่ทำให้ภาษายุ่งยากเกินไปออก และออกแบบให้เป็นภาษาเชิงวัตถุที่รอบคอบกว่า C^{++} นำเทคนิคของการออกแบบโปรแกรมเชิงวัตถุมาใช้เพื่อช่วยในการสร้างโปกแกรมที่เกี่ยวกับ Graphic User Interface , Multitasking , Network Programming และ Plug – in program ได้ง่าย
- โปรแกรมที่เขียนด้วยภาษาจาวาจะต้องมีความผิดพลาดที่ไม่คาดคิดมาก่อนน้อยที่สุดและโปรแกรมจะต้องไม่ล้มเหลวง่ายๆด้วยความผิดพลาดเพียงเล็กน้อย คุณสมบัตินี้เรียกว่า ความคงทน (Robust)
- โปรแกรมที่สร้างด้วยภาษาจาวาสามารถถูกส่งผ่านเครือข่ายไปทำงานในเครื่องคอมพิวเตอร์ต่างชนิดกันได้

ในโปรแกรมประยุกต์ที่พัฒนานั้น ออบเจ็กต์ส่วนหนึ่งจะต้องคงอยู่ต่อเพื่อใช้ในการทำงานครั้งต่อๆไปของโปรแกรม ดังนั้นจึงควรมีการจัดการการเก็บออบเจ็กต์เหล่านี้ให้ดี โดยเก็บลงในฐานข้อมูลที่มีระบบจัดการฐานข้อมูลที่มีประสิทธิภาพ

และจากที่ได้กล่าวมาทั้งหมดข้างต้นนั้น โครงงานนี้จึงได้ใช้แนวทางการพัฒนาเชิงวัตถุ โดยกำหนดโดเมนปัญหามาจากระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร ทำการวิเคราะห์ความต้องการของระบบและออกแบบโดยใช้ CASE tool ที่สนับสนุนโมเดลเชิงวัตถุ สร้างโปรแกรมด้วยภาษาจาวาและมีระบบจัดการฐานข้อมูลแบบรีเลชันแนล

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 ศึกษาแนวทางการพัฒนาเชิงวัตถุ โดยใช้ Visual Modeling Technique (VMT)

1.2.2 ศึกษาการนำเสนอโมเดลการพัฒนาเชิงวัตถุด้วย Unified Modeling Language (UML) โดยใช้โปรแกรม Rational Rose ซึ่งเป็น CASE tool ช่วยในการสร้างโมเดล

1.2.3 การสร้างโปรแกรมประยุกต์โดยใช้โปรแกรม IBM VisualAge for Java โดยมีการทำงานแบบไคลเอนท์/เซิฟเวอร์

1.2.4 ศึกษาการสร้างฐานข้อมูลรีเลชันแนลจากออบเจ็กต์ที่ได้จากการพัฒนาเชิงวัตถุ และทำการสร้างฐานข้อมูลแบบรีเลชันแนลด้วย Oracle

1.3 ขอบเขตของโครงงาน

โครงงานนี้จะออกแบบและสร้างโปรแกรมประยุกต์สำหรับระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร โดยมีฟังก์ชันการทำงานและลักษณะของระบบตามที่กำหนดไว้ในรายละเอียดความต้องการของระบบที่ได้จากขั้นตอนการวิเคราะห์ แต่เป็นการจำลองฟังก์ชันการทำงานหลักมาเท่านั้น ยังไม่ครอบคลุมถึงฟังก์ชันการทำงาน,รายละเอียดและข้อกำหนดต่างๆที่เป็นจริงทั้งหมดในทางธุรกิจ ระบบนี้ใช้เพื่อเป็นตัวอย่างในการศึกษาแนวทางการพัฒนาและสร้างระบบเท่านั้น

โดยระบบจะมีฐานข้อมูลกลางซึ่งเป็นออราเคิลเซิฟเวอร์ ไคลเอนท์ทุกเครื่องจะลงโปรแกรมเงินกู้เพื่อการผ่อนบ้านจัดสรรบนระบบปฏิบัติการ Windows NT 4.0 และจะติดต่อกับเซิฟเวอร์ฐานข้อมูลผ่านทางเครือข่าย LAN

1.4 วิธีการดำเนินงาน

เริ่มจากการศึกษาทฤษฎีการพัฒนาเชิงวัตถุ โดยใช้ Visual Modeling Technique กำหนดโดเมนปัญหาของระบบ วิเคราะห์ความต้องการและออกแบบระบบ นำเสนอด้วยโมเดล UML ตามรายละเอียดในบทที่ 2,3 และ 4 จาก Class Diagram ที่ได้นำมาออกแบบฐานข้อมูลแบบรีเลชันแนลสำหรับออบเจ็กต์คงอยู่ ตามทฤษฎีที่กล่าวในบทที่ 5

บทที่ 6 จะเป็นรายละเอียดของระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร แสดงโมเดลต่างๆและทำการออกแบบฐานข้อมูล ตามทฤษฎีที่ได้กล่าวมาแล้ว โดยสร้างโมเดลด้วยโปรแกรม Rational Rose

จากนั้นจะถึงขั้นตอนการสร้างระบบจากโมเดลที่ได้ โดยในบทที่ 7 จะเป็นวิธีการสร้างฐานข้อมูลแบบรีเลชันแนลด้วยออราเคิล (Oracle) และบทที่ 8 จะเป็นวิธีการสร้างโปรแกรมประยุกต์ด้วย VisualAge for Java

บทที่ 9 จะเป็นบทสรุปถึงการทดสอบ การทำงานของโปรแกรมประยุกต์ที่สร้างขึ้น และแนวทางในการพัฒนาโครงงานนี้เพิ่มเติม

บทที่ 2

Visual Modeling Technique

2.1 บทนำ

Visual Modeling Technique (VMT) คือแนวทางในการแก้ปัญหา โดยการออกแบบระบบด้วยโมเดลตามแนวความคิดของโลกแห่งความเป็นจริง โมเดลที่ถูกออกแบบมานั้น จะมีประโยชน์ช่วยในการเข้าใจถึงปัญหาของระบบ , การติดต่อกันของผู้ที่จัดทำโครงงาน (เช่น ลูกค้า , ผู้ดูแลระบบ , นักวิเคราะห์ และนักออกแบบ เป็นต้น) , การเตรียมการทำเอกสาร , การออกแบบโปรแกรม และ database

โมเดลเป็นนามธรรมที่แสดงถึงลักษณะที่สำคัญของปัญหาหรือโครงสร้างของระบบ โดยการกลั่นกรองรายละเอียดที่ไม่สำคัญออก ดังนั้นจึงเป็นการทำให้ปัญหานั้นง่ายต่อการเข้าใจ เราจึงควรสร้างโมเดลสำหรับระบบที่ซับซ้อน เพราะความสามารถของคนที่จะเข้าใจระบบที่ซับซ้อนนั้นมีจำกัด การสร้างโมเดลจะทำให้นักออกแบบสามารถเห็นภาพกว้างของโครงงานได้ โมเดลที่ดีควรจะใช้สัญลักษณ์ที่ถูกต้องและมีการตรวจสอบโมเดลให้ตรงตามความต้องการของระบบ

Visual Modeling Technique จะมีส่วนเกี่ยวข้องกับทุกขั้นตอนของการพัฒนาโปรแกรมเชิงวัตถุ ทั้งการรวบรวมความต้องการ (requirement gathering) , การวิเคราะห์ (analysis) , การออกแบบ (design) , การเขียนโปรแกรม (coding) และการทดสอบ (testing)

2.2 ขั้นตอนการพัฒนาโปรแกรมประยุกต์ (Application Development Phases)

ขั้นตอนของการพัฒนาโปรแกรม สามารถแบ่งออกเป็นส่วนต่างๆ ได้ดังนี้

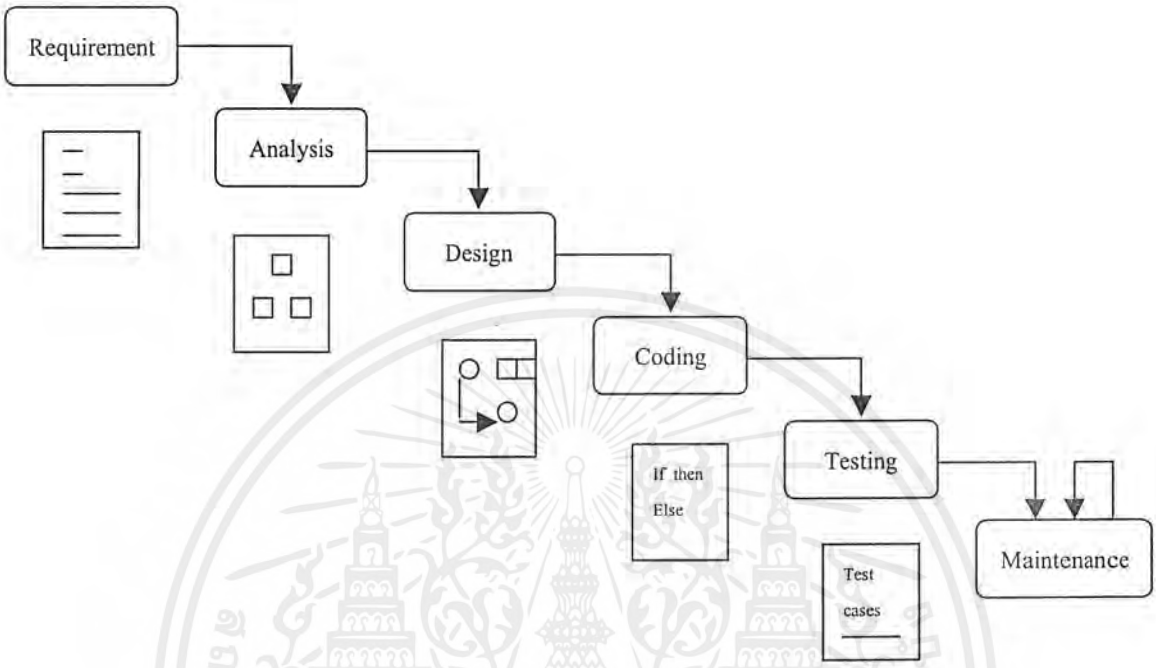
- Requirements specification — เป็นการระบุถึงความต้องการของ user
- Analysis — เป็นการเข้าใจถึงปัญหา มันจะอธิบายถึงสิ่งที่โปรแกรมต้องทำโดยไม่กล่าวถึงว่าต้องทำมันอย่างไร
- Design — เป็นการระบุว่าจะสร้างโปรแกรมขึ้นได้อย่างไร จากรายละเอียดที่กำหนดไว้ในส่วนขั้นตอนการวิเคราะห์
- Implementation — เป็นการแปลงการออกแบบไปเป็นภาษาโปรแกรม
- Testing — เป็นการทดสอบการทำงานของโปรแกรมให้ตรงตามความต้องการ
- Maintenance — เป็นการดูแลในระบบสามารถทำงานได้ หาข้อผิดพลาดและทำการแก้ไข ปรับปรุงระบบตามความต้องการที่เปลี่ยนไป

2.3 รูปแบบการพัฒนาโปรแกรม (Application Development Process Models)

รูปแบบของการพัฒนาโปรแกรม เป็นการกำหนดแนวทางการทำงานในแต่ละขั้นตอนของการพัฒนาโปรแกรม โดยทั่วไปสามารถแบ่งเป็นรูปแบบต่างๆ ได้ดังนี้

2.3.1 การพัฒนาตามลำดับขั้น (Sequential Process Model)

เป็นรูปแบบการพัฒนาโปรแกรมแบบเป็นลำดับขั้นตอน ซึ่งจะมีลักษณะเหมือนกับการไหลของน้ำตก (บางครั้งเรียกว่า Waterfall Model) การทำงานจะทำทีละขั้นตอนจนเสร็จ แล้วจึงจะไปทำในขั้นตอนถัดไป โดยไม่มีการย้อนกลับไปสู่ขั้นตอนการทำงานที่ผ่านมา



รูปที่ 2-1 การพัฒนาตามลำดับขั้น

ลักษณะแบบ waterfall นี้จะสามารถใช้งานได้ดีเมื่อระบบมีความต้องการของผู้ใช้ที่ 1. ชัดเจนและมีความเข้าใจครบถ้วน 2. มีความคงที่ตลอดช่วงของการพัฒนา อย่างไรก็ตามลักษณะการทำงานแบบนี้อาจจะทำให้เกิดปัญหาของระบบขึ้นได้ เนื่องจากผู้ใช้โปรแกรมนั้นจะได้ลองใช้งานโปรแกรมครั้งแรกหลังจากการพัฒนาเสร็จสิ้น แล้วถ้าเกิดผู้ใช้เกิดความไม่พอใจของโปรแกรมจะทำให้ต้องมีการแก้ไขโปรแกรมกันใหม่อีกครั้ง

2.3.2 การวนพัฒนา (Iterative Process Model)

เป็นรูปแบบการพัฒนาโปรแกรมแบบการวนรอบ โดยแต่ละรอบของการพัฒนาจะทำการกำหนดความต้องการของระบบ ทำการวิเคราะห์ ออกแบบ เขียนโปรแกรมและทำการทดสอบ จากนั้นก็จะทำการตรวจสอบโปรแกรมว่าตรงตามความต้องการของระบบหรือไม่ ถ้ายังก็จะทำการพัฒนาในรอบต่อไปจนกระทั่งได้โปรแกรมที่ถูกต้อง

ข้อดีของการทำงานแบบวนพัฒนาคือ ความสามารถในการส่งการเปลี่ยนแปลงจากการตรวจสอบความถูกต้องของโปรแกรมในขั้นตอนหนึ่งๆกลับไปยังขั้นตอนก่อนหน้า โดยทั่วไปแล้วประโยชน์ที่ได้

จากการเขียนโปรแกรมเชิงวัตถุ (object-oriented) นั้นได้มาจากลักษณะการทำงานแบบวนพัฒนา แม้ว่าโปรแกรมเชิงวัตถุอาจจะสามารถถูกพัฒนาแบบตามลำดับขั้นตอนได้ก็ตาม

ข้อเสียของการพัฒนาโปรแกรมรูปแบบนี้ก็คือ ลักษณะของการวนรอบนั้นจะควบคุมได้ยาก ดังนั้นจึงต้องมีการกำหนดจำนวนรอบต่ำสุดและสูงสุดของการพัฒนา โดยทั่วไปจะสนับสนุนการทำการวนพัฒนาต่ำสุดจำนวน 3 รอบ

2.3.3 การวนเพิ่ม (Incremental Process Model)

รูปแบบของการพัฒนาโปรแกรมแบบนี้ คล้ายกับการพัฒนาแบบวนพัฒนาแต่ต่างกันที่ขอบเขตของแต่ละขั้นตอนของการพัฒนาในการกำหนดความต้องการของระบบ จะเริ่มด้วยการกำหนดหน้าที่ขั้นแรกที่ทำได้ ซึ่งไม่ใช่เป็นความต้องการทั้งหมดของระบบจริงๆแล้วจึงค่อยๆเพิ่มความต้องการของระบบขึ้นในการพัฒนารอบต่อๆไป ดังนั้นการพัฒนาโปรแกรมในแต่ละรอบจึงทำให้มีการทำงานที่เร็วขึ้น อย่างไรก็ตามรูปแบบการทำงานแบบนี้ต้องการการวางแผนที่ระมัดระวังและมีการจัดการการควบคุมที่เข้มงวด

2.3.4 การพัฒนาแบบวนรอบ (Spiral Process Model)

รูปแบบของการพัฒนาแบบนี้ เกิดจากการรวมกันของรูปแบบวนพัฒนาและแบบวนเพิ่ม รูปแบบการพัฒนาแบบ spiral จะมีการสร้างต้นแบบ (Prototype) ขึ้นมาหลายระดับและมีการทำการวิเคราะห์ความเสี่ยงในแต่ละระดับของการพัฒนาด้วย รูปแบบนี้ประกอบด้วยขั้นตอนอยู่ 6 ขั้นดังนี้

1. ทำการวางแผนการพัฒนา
2. กำหนดจุดมุ่งหมาย ทางเลือก และข้อบังคับ
3. ทำการวิเคราะห์ความเสี่ยง
4. สร้างแบบจำลองต้นแบบ
5. ตรวจแก้ต้นแบบที่ไม่ตรงกับเป้าหมาย
6. กระทำตามขั้นตอนข้างต้นอีกครั้งจนกว่าได้ผลลัพธ์ที่ถูกต้อง

ลักษณะของการพัฒนาโปรแกรมแบบ Spiral Process Model เป็นรูปแบบการพัฒนาแบบเคลื่อนไหว (dynamic) ซึ่งดีสำหรับโครงงานระดับกลางถึงระดับใหญ่ซึ่ง

- ปัญหาทั้งหมดของระบบ ยังไม่สามารถเข้าใจได้ครบถ้วน
- ในโลกแห่งความเป็นจริง อาจจะเกิดการเปลี่ยนแปลงได้ระหว่างการพัฒนา
- อาจจะมีผลกระทบที่ไม่รู้จักเกิดขึ้นได้ในระบบ

2.4 การสร้างแบบจำลองเชิงวัตถุ (Object Methodologies)

วิธีการสร้างแบบจำลองเชิงวัตถุ เป็นขั้นตอนการออกแบบระบบเพื่อนำไปสู่การเขียนโปรแกรม โดยจะกำหนดลำดับของขั้นตอน และเทคนิคในแต่ละขั้นตอนของการสร้างแบบจำลอง ในปัจจุบันมีวิธีการสร้างแบบจำลองนี้อยู่มากมายหลายแบบ ซึ่งแต่ละแบบก็จะมีวิธีการและสัญลักษณ์ที่ใช้เป็นของตัวเอง

วิธีการที่ได้รับความนิยมมากมีอยู่ 3 แบบ คือ แบบ OMT (Rumbaugh) , แบบ Booch และแบบ OOSE (Jacobson)

2.4.1 The Object Modeling Technique (OMT)

วิธีการแบบ OMT จะครอบคลุมถึงขั้นตอนการวิเคราะห์,การออกแบบและการสร้าง ดังนี้

2.4.1.1 การวิเคราะห์

ในส่วนของการวิเคราะห์จะเพ่งเล็งถึงความเข้าใจของระบบและการจำลองงาน ผลของการวิเคราะห์จะเป็นข้อกำหนดทั่วไปของระบบ โดยสามารถสร้างเป็นโมเดลได้ 3 แบบ คือ

- *The Object Model*

  เป็นแบบจำลองที่แสดงถึงโครงสร้างคงที่ของระบบ โดยแบบจำลองนี้จะแสดงความสัมพันธ์ระหว่างออบเจ็กต์ (Object), แอททริบิวต์ (Attribute) และโอเปอเรชั่น (Operation) ของออบเจ็กต์ของแต่ละคลาส

- *The Dynamic Model*

  เป็นแบบจำลองที่แสดงถึงเวลา,พฤติกรรมและรูปแบบการควบคุมของระบบ รวมถึงลำดับของเหตุการณ์ , สภาวะ และการทำงานที่เกิดขึ้นของออบเจ็กต์ภายในระบบ

- *The Function Model*

  เป็นแบบจำลองที่แสดงรายละเอียดการทำงานของออบเจ็กต์ซึ่งถูกกำหนดเป็นการกระทำใน Dynamic Model ลักษณะของแบบจำลองนี้นำเสนอโดย Data Flow Diagram

2.4.1.2 การออกแบบ

การออกแบบมีอยู่ 2 ส่วน คือ การออกแบบระบบ (System Design) และการออกแบบวัตถุ (Object Design) โดยการออกแบบระบบจะเป็นการออกแบบในส่วนของสถาปัตยกรรมระบบ ซึ่งจะมีการจัดแบ่งเป็นระบบย่อย ส่วนการออกแบบวัตถุจะเป็นการออกแบบรายละเอียดที่กำหนดขึ้นตามแบบจำลองออบเจ็กต์

แบบจำลองทั้ง 3 แบบที่ได้จากการวิเคราะห์จะถูกแก้ไขในขั้นตอนการออกแบบวัตถุเพื่อที่จะแปลงปัญหา(ทางธุรกิจ)ไปสู่ผลลัพธ์(ทางคอมพิวเตอร์)

2.4.1.3 การสร้าง

เป็นการสร้างโปรแกรมจากแบบจำลองเพื่อใช้งานในระบบ โดยในระหว่างขั้นตอนนี้จะไม่มีการสร้างแบบจำลองขึ้นมาใหม่

2.4.2 The Booch Method

เป็นวิธีการออกแบบจากขั้นตอนการกำหนดออบเจ็กต์ของคลาสไปยังขั้นตอนการสร้างคลาส โดยวิธีการนี้ไม่สามารถแบ่งเป็นขั้นตอนการพัฒนาที่ชัดเจนได้ แต่ได้แบ่งเป็น Micro Development Process และ Macro Development Process

2.4.2.1 The Micro Development Process

เป็นขบวนการพัฒนาสถานการณ์ย่อยที่ได้จาก Macro Development Process ซึ่งจะเป็นงานที่ต้องทำรายวันของนักพัฒนากลุ่มย่อย โดยประกอบด้วยขั้นตอนต่อไปนี้

1. กำหนดคลาสและ ออบเจ็กต์ในระดับของนามธรรม
2. กำหนดคำของคลาสและออบเจ็กต์
3. กำหนดความสัมพันธ์ระหว่างคลาสและออบเจ็กต์
4. ระบุส่วนติดต่อ (Interface) และทำการสร้างคลาสและออบเจ็กต์

2.4.2.2 The Macro Development Process

เป็นขบวนการพัฒนาระบบใหญ่ ซึ่งจะควบคุมถึงการทำงานของ Micro Development Process ขบวนการนี้จะเป็นงานของผู้จัดการกลุ่มพัฒนา ดังนั้นจึงเพ่งเล็งไปที่ความเสี่ยงและลักษณะของสถาปัตยกรรมระบบ โดยประกอบด้วยขั้นตอนต่างๆ ดังนี้

1. กำหนดแนวความคิด โดยการสร้างความต้องการของระบบ
2. ทำการวิเคราะห์ระบบ
3. ทำการออกแบบสถาปัตยกรรมของระบบ
4. จัดทำโครงงาน
5. ทำการดูแลรักษาโครงงาน

วิธีการแบบ Booch Method จะไม่มีการกำหนดรายละเอียดของแต่ละขั้นตอนอย่างชัดเจน รายละเอียดต่างๆ จะเกิดขึ้นในขั้นตอนของการออกแบบไปจนถึงขั้นตอนของการสร้าง

2.4.3 Object–Oriented Software Engineering (OOSE)

วิธีการแบบ OOSE จะเป็นการใช้แนวความคิดของ Use Case ในการจัดทำความต้องการของระบบ

Use Case แสดงถึง "การสนทนา" ระหว่างผู้ใช้กับระบบ โดยแนวคิดแบบ Use Case สอดคล้องกับแบบแผนของการวิเคราะห์และการออกแบบเชิงวัตถุ เพราะได้ร่างภาพของระบบงานในโลกแห่งความเป็นจริง

OOSE แบ่งได้เป็น 3 ขั้นตอน ดังนี้

1. การวิเคราะห์ ขั้นตอนนี้เกี่ยวข้องกับการตรวจสอบความต้องการ และการสร้างแบบจำลองความต้องการของระบบ โดยแบบจำลองที่ถูกสร้างขึ้นประกอบด้วย Use Case Model , คำอธิบายส่วนติดต่อ (Interface Description) และโมเดลขอบเขตปัญหา (Problem Domain Model)

2. การสร้าง ขั้นตอนนี้จะรวมขั้นตอนการออกแบบและขั้นตอนการสร้างโครงงานเข้าไว้ด้วยกัน โดยผลลัพธ์ที่ได้จากขั้นตอนนี้ คือ Design Model และ Implementation Model

3. การทดสอบ ขั้นตอนนี้ได้รวมถึงการทดสอบหน่วยย่อย (Unit Testing) , การทดสอบการรวม (Integration Testing), การทดสอบระบบ (System Testing) และทำการสร้าง Test Model

2.4.4 The Unified Modeling Language (UML)

จากวิธีการในข้างต้นทั้ง 3 แบบนั้น ต่างก็มีจุดแข็งและจุดอ่อนที่แตกต่างกัน OMT จะมีจุดแข็งที่การวิเคราะห์ แต่มีจุดอ่อนที่การออกแบบ ส่วนแบบ Booch จะมีจุดแข็งที่การออกแบบและมีจุดอ่อนที่การวิเคราะห์ และแบบ OOSE จะมีจุดแข็งที่วิธีการทำการวิเคราะห์ แต่มีจุดอ่อนในส่วนอื่นๆ

ดังนั้นในเวลาต่อมาจึงได้มีการปรับปรุงข้อดีของวิธีการต่างๆ มารวมกันเป็น Unified Modeling Language (UML) โดยมีนิยามว่า "UML คือภาษามาตรฐานที่ใช้ในการระบุรายละเอียด สื่อที่สามารถมองเห็นได้และการสร้างวัตถุของระบบเชิงวัตถุภายใต้การพัฒนา โดยแสดงถึงการรวมกันของวิธีการแบบ Booch , OMT และเครื่องหมายที่ใช้แทนออบเจ็กต์ ซึ่งเป็นแนวความคิดที่ดีที่สุดจากจำนวนวิธีการต่างๆ ทั้งหลาย "

UML มีการจัดเตรียมสัญลักษณ์และเครื่องหมายเป็นจำนวนมาก จึงทำให้สามารถสร้างโมเดลและไดอะแกรมได้อย่างยืดหยุ่นและครอบคลุมปัญหาของระบบได้ทั้งหมด

ในโครงงานนี้เราจะใช้เครื่องมือช่วยในการออกแบบโปรแกรมเชิงวัตถุตามหลักการ UML ด้วยโปรแกรมที่ชื่อ Rational Rose

บทที่ 3
ขั้นตอนการวิเคราะห์ ตาม Visual Modeling Technique

ขั้นตอนการวิเคราะห์เป็นขั้นตอนหนึ่งของการพัฒนาระบบ ผลลัพธ์ที่ได้จากขั้นตอนนี้คือการอธิบายว่าระบบถูกสร้างขึ้นมาเพื่อทำอะไร ไม่ได้เป็นการระบุว่าระบบควรจะทำงานอย่างไร ในเทคนิคเชิงวัตถุ ผลลัพธ์จากการวิเคราะห์นี้คือโมเดล ซึ่งโดยปกติจะเรียกขั้นตอนนี้ว่า Modeling Phase

ขั้นตอนของการวิเคราะห์มีดังนี้

1. กำหนดประโยคปัญหา (Problem Statement) ของระบบ
2. กำหนดผู้กระทำ (Actor) และ Use Case จากประโยคปัญหา
3. ร่างภาพส่วนติดต่อผู้ใช้ของหน้าที่พื้นฐานที่ผู้ใช้ติดต่อกับระบบ
4. สร้าง Object Model
5. สร้าง Dynamic Model

3.1 ประโยคปัญหา (Problem Statement)

เป็นบทความที่บรรยายถึงลักษณะปัญหาของระบบ หน้าที่การทำงานต่างๆ ที่ระบบต้องมี ความสามารถในการติดต่อกันระหว่างผู้ใช้กับระบบ ขอบเขตการทำงานของระบบและรายละเอียดหน้าที่การทำงานที่มีในระบบ

3.2 Use Case Model

Use case model ประกอบด้วยผู้กระทำ (Actor) และ Use Case

ผู้กระทำจะอยู่นอกระบบ แสดงถึงบุคคลหรือสิ่งของที่มีความสัมพันธ์กับระบบ โดยผู้กระทำอาจจะ

- ให้ข้อมูลแก่ระบบเพียงอย่างเดียว
- รับข้อมูลจากระบบเพียงอย่างเดียว
- ทั้งให้และรับข้อมูลกับระบบ

ผู้กระทำอาจจะพบได้ในประโยคปัญหาและการสนทนาระหว่างลูกค้ากับผู้ดูแลระบบ

Use cases คือ บทความสนทนาระหว่างผู้กระทำกับระบบ ซึ่งแสดงถึงหน้าที่การทำงานที่ถูกจัดเตรียมขึ้นโดยระบบ การกำหนด Use Case. ควรจะแสดงถึงงานหลักๆที่เกิดขึ้นตั้งแต่ต้นจนจบและต้องมีการส่งผลบางอย่างให้แก่ผู้กระทำ

ความสัมพันธ์ระหว่าง Use Case มีอยู่ 2 แบบ คือ *uses* และ *extends* ความสัมพันธ์แบบ *uses* เกิดจาก Use Case A ถูกเรียกใช้จาก Use Case B และ C นั่นคือ Use Case B และ C *uses* Use Case A

ส่วนความสัมพันธ์แบบ *extends* เกิดจาก Use Case B ถูกสร้างเริ่มต้นมาจาก Use Case A และเพิ่มรายละเอียดบางส่วน เราเรียกว่า Use Case B *extends* Use Case A

3.3 ส่วนติดต่อผู้ใช้ (User Interface)

เพื่อความเข้าใจความต้องการของผู้ใช้ในนการใช้งานโปรแกรม จึงควรมีการออกแบบส่วนการติดต่อระหว่างผู้ใช้กับระบบ โดยอาจจะเป็นการสร้างส่วนติดต่อของโปรแกรมในลักษณะคร่าวๆเพื่อให้ผู้ใช้สามารถเห็นภาพของการใช้งานโปรแกรมได้ดีขึ้น ซึ่งการสร้างส่วนติดต่อผู้ใช้นี้จะช่วยให้ผู้พัฒนาสามารถแสดงลักษณะการติดต่อภายใน Use Case ได้และเป็นการช่วยในการออกแบบโปรแกรมในขั้นตอนการออกแบบ

3.4 Object Model

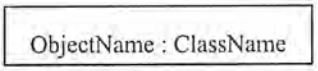
Object Model เป็นแบบจำลองที่สำคัญที่ได้จากขั้นตอนการวิเคราะห์ระบบและเป็นจุดเริ่มต้นในส่วนของการออกแบบและการสร้างระบบ ในโครงงานนี้เราจะสร้าง Object Model ตามรูปแบบของ UML

ขั้นตอนการสร้าง Object Model มีดังนี้

1. กำหนดออบเจ็กต์ที่เป็นไปได้ทั้งหมดของระบบจากประโยคปัญหาและ Use Case
2. กำจัดออบเจ็กต์ที่ไม่ตรงกับปัญหาหรือมีความหมายที่ซ้ำกัน
3. จัดรวมออบเจ็กต์ที่มีลักษณะเหมือนกันเข้าเป็นคลาส
4. กำหนดแอททริบิวต์และโอเปอเรชั่นของแต่ละคลาส
5. กำหนดความสัมพันธ์ระหว่างออบเจ็กต์ของคลาส ซึ่งมีลักษณะความสัมพันธ์ได้แก่ Generalization , Association และ Aggregation

3.4.1 ออบเจ็กต์ (Object)

ออบเจ็กต์เป็นกลุ่มก้อนของเอนติตี้ (Entity) ที่มีคุณสมบัติ,พฤติกรรมและสถานะ ออบเจ็กต์แสดงถึงบางอย่างที่มีตัวตน เช่น รถยนต์,เครื่องคอมพิวเตอร์ หรือแสดงถึงสิ่งที่มีตัวตนในความคิด เช่น บัญชีธนาคาร, เครื่องหมายการค้า, พิธีแต่งงานหรือรายการสินค้า

Object : 

รูปที่ 3-1 สัญลักษณ์แสดงออบเจ็กต์

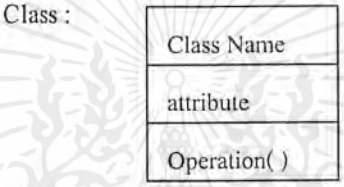
สิ่งที่เราสนใจในตัว object ประกอบไปด้วย

- แอททริบิวต์ (Attributes) เป็นข้อมูลที่แสดงคุณลักษณะของวัตถุ เช่น รถยนต์ จะมีแอททริบิวต์คือ ความกว้าง, ความยาว,สี, น้ำหนัก

- โอเปอเรชั่น (Operations) คือการกระทำที่วัตถุนั้นสามารถกระทำได้ เช่น รถยนต์ จะมี operations คือ ออกตัว, เบรก, เลี้ยวขวา, เลี้ยวซ้าย
- สถานะ (State) สถานะของวัตถุคือหนึ่งในเงื่อนไขที่เป็นไปได้และอาจเกิดขึ้นได้จริง เช่น รถยนต์ จะมี state คือ กำลังวิ่ง, กำลังชะลอ
- เอกลักษณ์ (Identity) เป็นสิ่งที่ระบุว่าวัตถุนั้นมีอยู่เพียงหนึ่งเดียว เช่น รถยนต์มีเอกลักษณ์คือ ทะเบียนรถ
- หน้าที่ (Responsibility) คือ หน้าที่และบทบาทของวัตถุต่อระบบ

3.4.2 คลาส (Class)

คลาส คือการอธิบายถึงกลุ่มของออบเจ็กต์ซึ่งจะมีคุณสมบัติทั่วไป(attributes),พฤติกรรมทั่วไป (operations) และมีความสัมพันธ์ทั่วไปกับออบเจ็กต์อื่นๆในลักษณะเดียวกัน ดังนั้นคลาสจึงเป็นรูปแบบที่ใช้สร้างออบเจ็กต์



รูปที่ 3-2 สัญลักษณ์แสดงคลาส

3.4.3 สเตอริโอไทป์ (Stereoptype)

คือ การสร้างองค์ประกอบชนิดใหม่ของแบบจำลองหรือการสร้างคลาสใหม่ ซึ่งองค์ประกอบชนิดใหม่จะสืบทอดคุณสมบัติขององค์ประกอบเดิมมาทุกอย่าง(รวมทั้งความสัมพันธ์ต่างๆที่ใช้กับองค์ประกอบนั้นๆ ได้) โดยมีจุดประสงค์การใช้งานที่ต่างออกไป

รูปแบบของสเตอริโอไทป์ใน UML จะใช้เครื่องหมาย << และ >> คร่อมชื่อสเตริโอไทป์ เช่น << Entity >> แนวคิดสเตอริโอไทป์ของโครงงานนี้จะแบ่งประเภทของคลาส เป็น เอนติตี้คลาส (Entity Class) , คลาสขอบเขต (Boundary Class) และ คลาสควบคุม (Control Class)

- เอนติตี้คลาส สิ่งที่มีอยู่จริง หรือการกระทำภายในระบบ
- คลาสขอบเขต กล่าวถึงการติดต่อระหว่างสิ่งที่อยู่รอบๆระบบกับภายในระบบ (การติดต่อไปยังผู้กระทำ)
- คลาสควบคุม ส่งผลให้เกิดพฤติกรรมเฉพาะใน Use Case หรือเรียกว่าเป็นการ " running " Use Case

3.4.4 แพ็คเกจ (Packages)

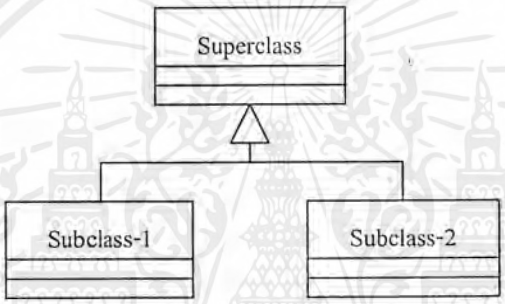
เป็นการรวมคลาสที่มีความสัมพันธ์ใกล้ชิดกันเข้าไว้ในแพ็คเกจ เพื่อประโยชน์สำหรับระบบที่มีคลาสเป็นจำนวนมาก ในระบบที่มีความซับซ้อนเราอาจจะสร้างแพ็คเกจขึ้นมาก่อน แล้วจึงบรรจุคลาสและความสัมพันธ์เข้าไว้ในแพ็คเกจ

3.4.5 ความสัมพันธ์

3.4.5.1 Generalization

เป็นความสัมพันธ์แบบการให้กำเนิดคลาสย่อย (Subclass) จากคลาสแม่ (Superclass) โดยคลาสย่อยจะสืบทอดแอททริบิวต์,โอเปอเรชั่นและความสัมพันธ์ทั้งหมดจากคลาสแม่ ทั้งของตัวเองและจากคลาสแม่ของคลาสแม่ชั้นต่อไปเรื่อยๆ คลาสย่อยอาจจะมีการเพิ่มแอททริบิวต์และโอเปอเรชั่นในคลาสของมันเอง ความสัมพันธ์แบบนี้อาจถูกเรียกได้เป็น is-a หรือ kind-of

Generalization :

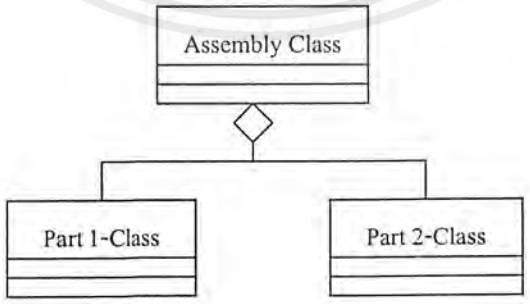


รูปที่ 3-3 ความสัมพันธ์แบบ Generalization

3.4.5.2 Aggregation

เป็นความสัมพันธ์แบบเป็นส่วนประกอบ หรือที่เรียกว่า part-of คำว่าเป็นส่วนประกอบบางครั้งอาจจะมีความหมายที่ต่างกัน เช่น เครื่องยนต์และล้อเป็นส่วนประกอบของรถยนต์ในขณะที่เรากำลังขับรถอยู่ แต่ถ้ารถยนต์คันนั้นอยู่ในระหว่างการซ่อมที่อู่ซ่อมรถ นั่นคือเครื่องยนต์และล้อก็ไม่ได้เป็นส่วนประกอบของรถยนต์แล้ว

Aggregation :

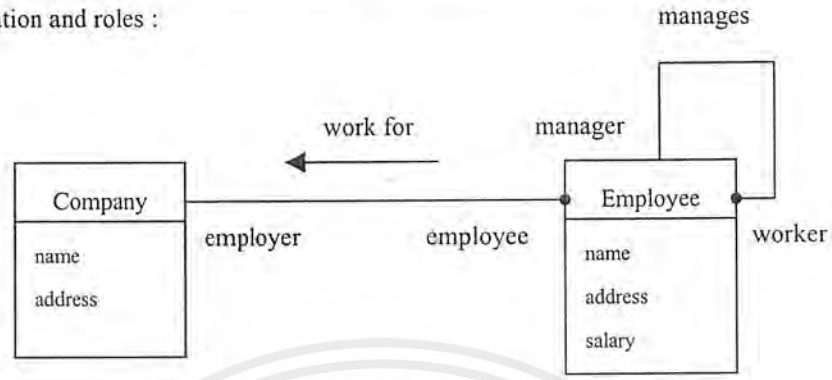


รูปที่ 3-4 ความสัมพันธ์แบบ Aggregation

3.4.5.3 Association

เป็นความสัมพันธ์แบบสองทิศทาง ซึ่งใช้ในการติดต่อกันระหว่างคลาส ความสัมพันธ์แบบ association ระหว่างคลาสหมายถึงมีการเชื่อมต่อระหว่างออบเจ็กต์ในคลาสทั้งสอง ที่ปลายของเส้นความสัมพันธ์จะมีการระบุชื่อบทบาทเพื่อบอกว่าคลาสนั้นถูกมองว่าเป็นอย่างไรจากคลาสอื่น

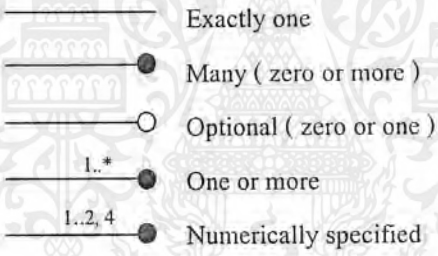
Association and roles :



รูปที่ 3-5 Association และบทบาท

Multiplicity ของ Association เป็นการบอกถึงจำนวนของออบเจ็กต์ที่มีการติดต่อกับคลาสอื่น

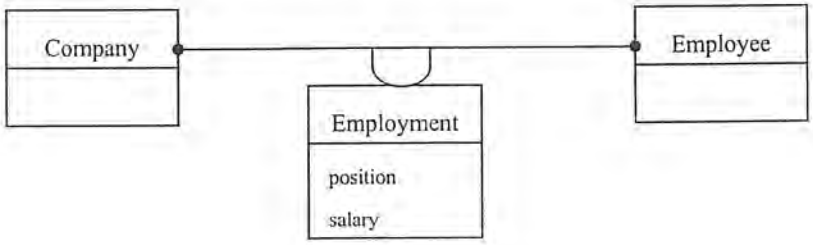
Association :



รูปที่ 3-5 Multiplicity

Association Class ในบางครั้งเราอาจจะออกแบบ Association เป็นเหมือนกับคลาสได้ ถ้าหากมีแอททริบิวต์ที่ไม่ได้เป็นของคลาสใดๆที่มีความสัมพันธ์กัน แอททริบิวต์นั้นจะเป็นของ Association Class

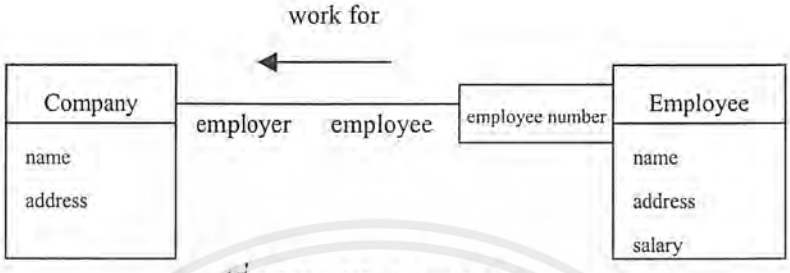
Modeling an association as a class :



รูปที่ 3-6 Association Class

Qualified Association คือ Association ที่มีแอททริบิวต์เรียกว่า Qualifier โดยค่าของแอททริบิวต์นั้นสามารถที่จะแบ่งแยกออบเจ็กต์ต่างๆของคลาสอีกด้านหนึ่งของความสัมพันธ์ได้ สัญลักษณ์ที่ใช้ก็คือรูปสี่เหลี่ยมขนาดเล็กที่ติดอยู่ระหว่างปลายของเส้น Association กับคลาสที่เชื่อมอยู่และจะมีแอททริบิวต์ที่มีลักษณะดังที่กล่าวมาอยู่ภายใน

Qualified Association :

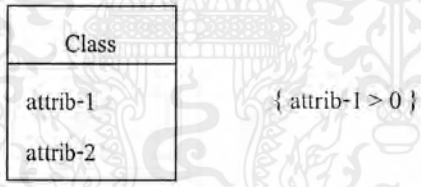


รูปที่ 3-7 Qualified Association

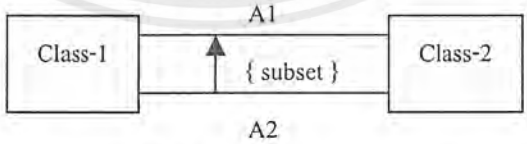
3.4.6 ข้อบังคับ (Constraints)

เป็นการกำหนดข้อบังคับของความสัมพันธ์ระหว่างเอนติตี้ โดยเอนติตี้อาจจะหมายถึงออบเจ็กต์, คลาส, บทบาท,แอททริบิวต์,การเชื่อมต่อ (link) และความสัมพันธ์ ข้อบังคับนี้จะเป็นการจำกัดค่าที่เอนติตี้สามารถเป็นไปได้

รูปแบบทั่วไปของข้อบังคับจะเป็นข้อความที่อยู่ในวงเล็บปีกกา เช่น



รูปที่ 3-8 ข้อบังคับในออบเจ็กต์



รูปที่ 3-9 ข้อบังคับระหว่าง Association

3.5 Dynamic Model

เป็นแบบจำลองที่อธิบายถึงพฤติกรรมของระบบและลำดับของการทำงาน ซึ่งแบ่งเป็น

1. Sequence Diagram
2. Collaboration Diagram

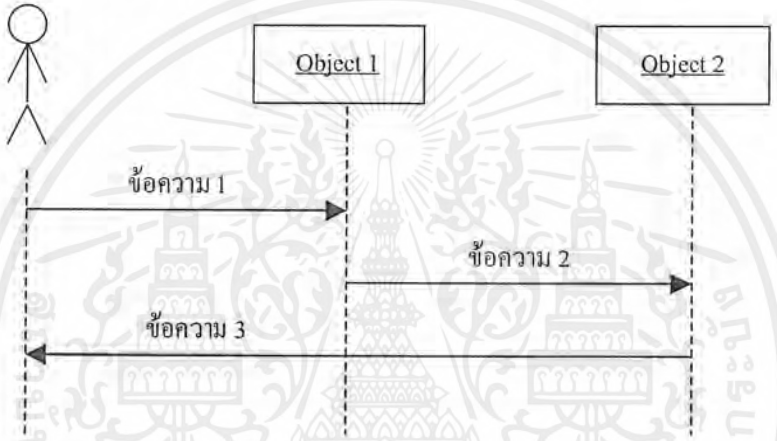
3. State Diagram

3.5.1 Sequence Diagram

Sequence Diagram เป็นการแสดงข้อความและลำดับของการส่งข้อความระหว่างออบเจ็กต์

ข้อความ คือ หน่วยที่ออบเจ็กต์ใช้ในการสื่อสารกัน ตัวอย่างของข้อความที่เกิดขึ้น เช่น เมื่อผู้ใช้ร้องขอบริการ , ออบเจ็กต์ร้องขอบริการจากออบเจ็กต์อื่น หรือออบเจ็กต์ส่งคืนข้อมูลจากบริการที่ถูกร้องขอ

Sequence Diagram จะถูกพัฒนามาจาก Use Case โดยแสดงถึงลำดับของข้อความที่เกิดขึ้นของแต่ละ Use Case ตั้งแต่ต้นจนจบ Sequence Diagram มีจุดประสงค์เพื่อช่วยให้ผู้ใช้และผู้ออกแบบระบบสามารถเข้าใจการทำงานของระบบได้ง่ายขึ้น

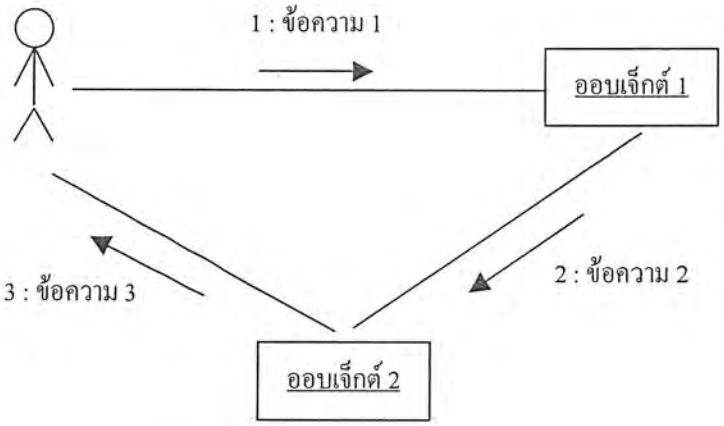


รูปที่ 3-10 Sequence Diagram

จากรูปไดอะแกรมข้างต้น เส้นในแนวตั้งจะหมายถึงออบเจ็กต์โดยมีชื่อของออบเจ็กต์เขียนอยู่ด้านบนหรือเป็นผู้กระทำที่ติดต่อกับระบบ ส่วนเส้นในแนวนอนนั้นจะแสดงถึงข้อความซึ่งมีชื่อของข้อความนั้นอยู่ ข้อความที่เกิดขึ้นจะเป็นลำดับตามแนวตั้งจากบนลงล่าง

3.5.2 Collaboration Diagram

Collaboration Diagram จะแสดงข้อมูลพื้นฐานเช่นเดียวกับ Sequence Diagram แต่ความแตกต่างอยู่ที่ Sequence Diagram จะแสดงลำดับของข้อความเป็นหลัก ส่วน Collaboration Diagram จะแสดงข้อความและโครงสร้างของออบเจ็กต์ที่ทำงานร่วมกันเป็นหลัก



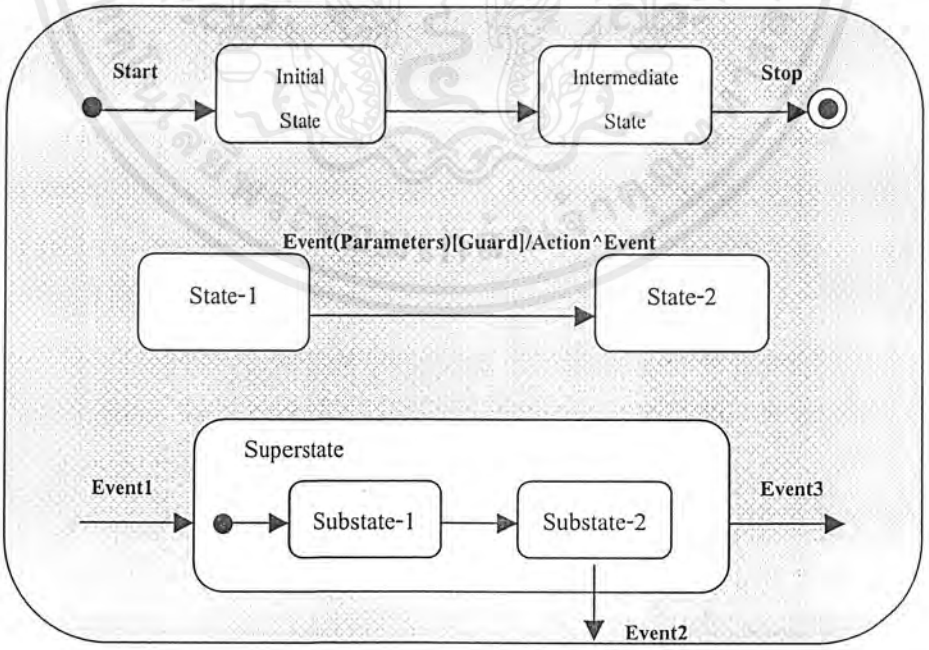
รูปที่ 3-11 Collaboration Diagram

จากรูปไดอะแกรมข้างต้น จะแสดงการไหลของข้อความระหว่างออบเจ็กต์ด้วยกันหรือระหว่างออบเจ็กต์กับผู้กระทำ โดยข้อความที่ใช้ในการติดต่อกันจะประกอบด้วยหมายเลขลำดับ,ชื่อข้อความและทิศทางของข้อความ

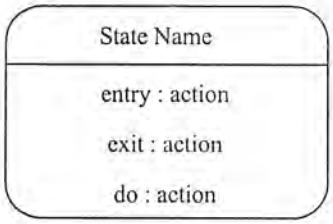
3.5.3 State Diagram

สถานะของออบเจ็กต์ปกติจะถูกกำหนดโดยค่าของแอททริบิวต์หรือค่าตัวแปรภายใน เช่น ออบเจ็กต์หลอดไฟจะมีสถานะคือเปิดและปิด ซึ่งสถานะเปิดและปิดนี้จะเป็นค่าของแอททริบิวต์สถานะของออบเจ็กต์หลอดไฟ

State Diagram แสดงถึงสถานะต่างๆของออบเจ็กต์และการส่งผ่าน (transition) ที่ทำให้เกิดการเปลี่ยนแปลงของสถานะเหล่านั้น และแสดงถึงกิจกรรมที่ออบเจ็กต์กระทำเมื่ออยู่ในสถานะหนึ่งๆ



รูปที่ 3-12 State Diagram



รูปที่ 3-13 รูปแบบของสถานะ

State Diagram สามารถมีสถานะซ้อนภายในสถานะอีกทีได้ ตัวสถานะที่อยู่ภายนอกนั้นเราจะเรียกว่า Superstate ส่วนสถานะที่อยู่ภายในเราจะเรียกว่า Substate ในแต่ละสถานะจะมีการระบุถึงการกระทำที่ออบเจ็กต์จะกระทำเมื่ออยู่ในสถานะนั้น โดยสามารถแบ่งการกระทำได้เป็น

- entry action จะทำงานเมื่อมีการเข้าสู่สถานะนั้น
- exit action จะทำงานเมื่อมีการออกจากสถานะนั้น
- do action จะทำงานเมื่อสถานะนั้นๆ แอกทีฟ

รูปแบบของการส่งผ่านคือ Event-name(Parameters)[Guard] / Action list ^Event list โดยมีรายละเอียดดังนี้

Field	คำอธิบาย
Event-name	เป็นชื่อของ event ที่ทำให้เกิดการเปลี่ยนสถานะ
Parameters	เป็นรายการข้อมูลพารามิเตอร์ ที่ต้องการส่งไปพร้อมกับ event
Guard	เป็นเงื่อนไขในรูปของประโยคบูลีน ซึ่งจะต้องมีค่าเป็นจริงจึงจะทำให้เกิดการเปลี่ยนสถานะได้
Action list	เป็นรายการของโอเปอเรชั่นที่จะกระทำเมื่อเกิดการเปลี่ยนสถานะ โดยโอเปอเรชั่นเหล่านี้อาจจะเป็นของออบเจ็กต์เองหรือของออบเจ็กต์อื่นก็ได้
Event list	เป็นรายการของ events ที่ถูกสร้างขึ้นเมื่อมีการเปลี่ยนสถานะ โดย events เหล่านี้สามารถกระจายไปยัง state machines อื่นๆ เพื่อเป็นการทำงานแบบพร้อมกันได้

ตารางที่ 3-1 รูปแบบการส่งผ่าน

บทที่ 4
ขั้นตอนการออกแบบตามวิธี Visual Modeling Technique

จุดมุ่งหมายของการออกแบบ คือ เป็นสิ่งแรกที่ใช้ในการคิดหาทางแก้ปัญหาและสร้างสถาปัตยกรรมทั้งหมดที่เป็นพื้นฐานสำหรับการสร้างเพื่อนำไปแก้ปัญหาระบบ จากนั้นเราจะทำการสำรวจวิธีการ และสร้างยุทธวิธีในการตัดสินใจเพื่อแปลงการออกแบบไปเป็นองค์ประกอบของการแก้ปัญหา

ในขั้นตอนของการออกแบบ จะแบ่งการออกแบบเป็นส่วนต่างๆ ดังนี้

- การออกแบบระบบ (System Design)

  เป็นการออกแบบระบบในระดับสูง โดยกำหนดสถาปัตยกรรมของซอฟต์แวร์และจัดโครงสร้างของระบบซึ่งจะแบ่งระบบที่มีขนาดใหญ่ออกให้เป็นระบบขนาดย่อย

- การออกแบบออบเจ็กต์ (Object design)

  เป็นการกลั่นกรองโมเดลที่ได้จากการวิเคราะห์ระบบและปรับให้เข้ากับการทำงานในสิ่งแวดล้อมที่มีอยู่ โดยการกำหนดรายละเอียดของของคลาสของออบเจ็กต์เพิ่มขึ้นและสร้างเป็น Design Model ของระบบ

- การออกแบบออบเจ็กต์คงอยู่ (Designing for object persistence)

  เป็นการออกแบบการเก็บรักษาออบเจ็กต์โดยเกี่ยวข้องกับการสร้างโมเดลและการออกแบบเซิฟเวอร์ฐานข้อมูล รวมถึงการแปลงคลาสของออบเจ็กต์ไปเป็นตารางฐานข้อมูลและกำหนดนโยบายการเข้าถึงข้อมูลเพื่อทำการแก้ไข หรือเปลี่ยนแปลงค่าของข้อมูลที่ใช้งานร่วมกัน

4.1 การออกแบบระบบ (System Design)

การออกแบบระบบเป็นการออกแบบสถาปัตยกรรมในระดับสูง (high-level) ของวิธีการแก้ปัญหาที่วางแผนไว้ โดยสถาปัตยกรรมของระบบจะกำหนดกลุ่มงานหลักๆของระบบและการเชื่อมต่อกันในระดับสูง กลุ่มของระบบนี้จะแสดงถึงฟังก์ชั่นการทำงานของระบบ สถาปัตยกรรมของแอพพลิเคชัน (Application Architecture) กำหนดโครงสร้างของวิธีการแก้ปัญหาโดยแบ่งเป็นระบบย่อย (subsystem) แสดงกลุ่มงาน โดยทั้งสถาปัตยกรรมระบบและสถาปัตยกรรมแอพพลิเคชันแบ่งรายละเอียดเป็น 3 ระดับ คือ แนวคิด (conceptual) , ลอจิคอล (logical) และ กายภาพ (physical)

Use Case Model , Object Model และ Dynamic Model จะใช้แสดงสถาปัตยกรรมทางแนวคิด (Conceptual Architecture)

การออกแบบระบบเป็นการเริ่มต้นพิจารณาในขั้นลอจิคอล ในขั้นนี้จำเป็นต้องมีการตัดสินใจสำหรับการจัดหาข้อมูล, กระบวนการ , แพลตฟอร์ม (platform) ที่ใช้ในการสร้างระบบ โดยพิจารณาจาก

เทคโนโลยีที่มีอยู่และสภาพแวดล้อมปัจจุบัน เช่น ระบบเก่า การตัดสินใจเป็นการเลือกเทคโนโลยีและผลิตภัณฑ์ที่ใช้ในการสร้างกลุ่มงานหลักๆของระบบ

อย่างไรก็ตามผู้ออกแบบจำเป็นที่จะต้องสร้างสถาปัตยกรรมทางกายภาพ (Physical Architecture) ซึ่งแสดงการจัดตั้งของระบบตามลอจิคอลและคอมโพเนนท์ของแอพพลิเคชันตามวิธีแก้ปัญหาไปเป็นแพลตฟอร์มการสร้างเช่นเดียวกับหน่วยย่อยของโปรแกรม (programming module)

การออกแบบสถาปัตยกรรมของระบบเป็นการสร้างแนวความคิดจากโมเดลต่างๆ ที่ได้มาจากขั้นตอนการวิเคราะห์ระบบ เช่น Use Case Model , Object Model และ Dynamic Model เพื่อหาทางแก้ปัญหาของระบบ โดยระบบสามารถแบ่งได้เป็นสถาปัตยกรรมทางกายภาพและสถาปัตยกรรมซอฟต์แวร์ ซึ่งในการออกแบบสถาปัตยกรรมระบบนี้เราจะแปลงซอฟต์แวร์คอมโพเนนท์ขนาดใหญ่ เช่น ระบบย่อย, แพ็คเกจ (packages) และงาน (tasks) ไปยังหลายๆตัวประมวลผลและอุปกรณ์ต่างๆของระบบ สำหรับระบบที่มีขนาดไม่ใหญ่มากนัก เราอาจจะข้ามขั้นตอนของการออกแบบในระดับของสถาปัตยกรรมนี้ได้ แต่สำหรับระบบที่มีขนาดใหญ่ที่มีการทำงานแบบกระจายนั้น ผู้พัฒนาระบบจำเป็นต้องมีการออกแบบระบบในระดับของสถาปัตยกรรมด้วย

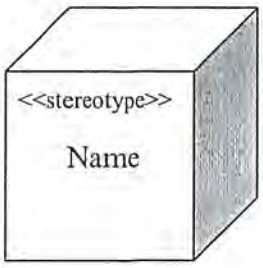
สถาปัตยกรรมทางกายภาพ

ในการตัดสินใจออกแบบอุปกรณ์ฮาร์ดแวร์ของระบบ จะต้องมีความสัมพันธ์โดยเฉพาะกับโครงสร้างทางซอฟต์แวร์ เช่นจำนวน และชนิดของอุปกรณ์ที่ใช้ในระบบ ตัวอย่างของการออกแบบ เช่น ตัวประมวลผลจะมีการเชื่อมต่อเข้าด้วยกันเป็นอย่างไร? ควรจะมีรูปแบบการวางสายการสื่อสารแบบบัสหรือสตาร์ ? ควรใช้อัตราส่งข้อมูลความเร็วเท่าใด ? การตัดสินใจออกแบบอุปกรณ์ต่างๆ เหล่านี้ จะมีผลกระทบอย่างมากกับโครงสร้างทางซอฟต์แวร์

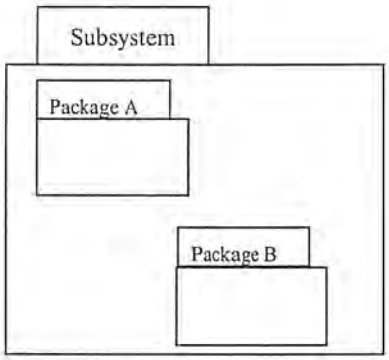
สถาปัตยกรรมซอฟต์แวร์

เป็นการรวมกลุ่มของซอฟต์แวร์คอมโนเนนท์เข้าไปอยู่ในตัวประมวลผลทางกายภาพตัวเดียว โดยคอมโพเนนท์เหล่านี้จะมีชนิดเป็นแพ็คเกจ เราจะทำการสร้างบล็อคของระบบย่อยที่มีแพ็คเกจและ งานบรรจุอยู่ โดยแพ็คเกจอาจจะบรรจุแพ็คเกจย่อย (subpackages) ได้

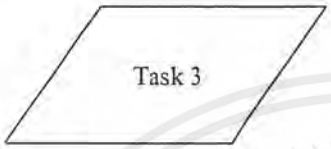
UML จะนำเสนอสถาปัตยกรรมทางกายภาพด้วย Deployment Diagram ซึ่งจะมีรูปสัญลักษณ์ดังนี้



Node



ระบบย่อยและแพ็คเกจ



แอคทีฟออบเจ็กต์

การเชื่อมต่อ Node

รูป 4-1 UML Notation สำหรับสถาปัตยกรรมทางกายภาพ

Nodes แสดงถึงตัวประมวลผล , เซนเซอร์ , ตัวกระทำ (actuators) , เราท์เตอร์ (routers) , ตัวแสดงผล, อุปกรณ์อินพุต, หน่วยความจำหรือออบเจ็กต์ทางกายภาพอื่นๆที่มีความสำคัญต่อโปรแกรม nodes จะมี stereotype เพื่อเป็นตัวแสดงถึงชนิดของ node

การเชื่อมต่อ แสดงถึงการติดต่อภายในระหว่าง nodes

4.1.1 การแบ่ง Object Model เป็นระบบย่อย

ระบบย่อยหมายถึงกลุ่มคลาสที่มีความเกี่ยวข้องกันอย่างเหนียวแน่น เหตุผลที่สนับสนุนการแบ่ง Object Model เป็นระบบย่อยมีดังนี้

- ความพยายามที่จะจัดการความซับซ้อนของระบบ
- แบ่งงานให้แก่ผู้พัฒนาหลายคนหรือหลายทีม
- แยกการตัดสินใจการออกแบบเฉพาะ ที่ผู้ออกแบบเชื่อว่าอาจมีการเปลี่ยนแปลงภายหลัง
- แบ่งระดับต่างๆของ abstraction ที่ให้บริการ
- จัดตั้งหน่วยสำหรับการกระจาย (ใน distributed object application)

เหตุผลแรกจะสำเร็จได้หากการแบ่งเป็นระบบย่อยทำให้สามารถเข้าใจระบบได้ดีขึ้น เหตุผลที่สองจะทำได้ดีหากทีมพัฒนาไม่ต้องขึ้นอยู่กับกันมาก ซึ่งจะเป็นไปได้เมื่อข้อความที่ส่งระหว่างระบบย่อยทั้งหมดมีความซับซ้อนน้อยกว่าข้อความที่ส่งภายในระบบย่อย

แนวทางหนึ่งที่ใช้เป็นหลักในการแบ่ง Object Model เป็นระบบย่อยคือพิจารณาวิธีการส่งข้อความระหว่างระบบย่อยให้มีน้อยที่สุด ตัวอย่างต่อไปนี้สามารถใช้เป็นหลักการเลือกระบบย่อย

MsgSent : กลุ่มของข้อความซึ่งผู้ส่งคือคลาสในระบบย่อย

MsgGoingOut : เป็นสับเซตของ MsgSent ซึ่งสร้างจากระบบย่อยอื่น

MsgStayIn : เป็นสับเซตของ MsgSent ซึ่งสร้างจากภายในระบบย่อยเดียวกัน

การเลือกระบบย่อยจะดีถ้า MsgStayIn >> MsgGoingOut

heuristic ที่เหมาะสมในการเริ่มหากลุ่มระบบย่อยคือ แบ่งตามหน้าที่รับผิดชอบ (Responsibility – Driven Approach) ซึ่งจะแบ่งโดยพิจารณาจากหน้าที่ของแต่ละระบบย่อย

ระบบย่อยแสดงสัญลักษณ์ด้วยสี่เหลี่ยมผืนผ้า ใช้ลูกศรแสดงความขึ้นต่อกันระหว่างระบบย่อย

4.1.1.1 สถาปัตยกรรมแอพพลิเคชันแบบหน้าที่ (Partition) และชั้น (Layer)

ภายในระบบย่อยสามารถแบ่งเป็นระบบย่อยได้อีก โดยเริ่มแรกเราแบ่งแอพพลิเคชันเป็นระบบย่อยตามแนวตั้ง (vertical subsystem) ซึ่งเป็นการแบ่งโดยยึดตามฟังก์ชั่นการทำงานหรืออีกนัยหนึ่ง ระบบย่อยสามารถแบ่งโดยยึดตามบทบาทและหน้าที่

แต่มีอีกหลายระบบที่มีการแบ่งแบบชั้น ซึ่งมีลักษณะชั้นหนึ่งจะสร้างต่อจากชั้นล่างและจะเป็นพื้นฐานในการสร้างชั้นบนจากตัวเองอีก ระบบย่อยจะรู้จักและขึ้นอยู่กับชั้นที่อยู่ข้างล่างและจะให้บริการแก่ชั้นที่อยู่ข้างบน เรียกการแบ่งในลักษณะนี้ได้ว่าการแบ่งระบบย่อยเป็นชั้นตามแนวนอน (layered horizontal subsystem) การแบ่งแอพพลิเคชั่นแบบชั้นช่วยให้การออกแบบไม่ยุ่งยากและสามารถนำไปใช้กับระบบอื่นได้ง่าย (portable) ซึ่งเป็นสิ่งสำคัญในการทำ Visual Programming เช่นใน VisualAge ด้วย

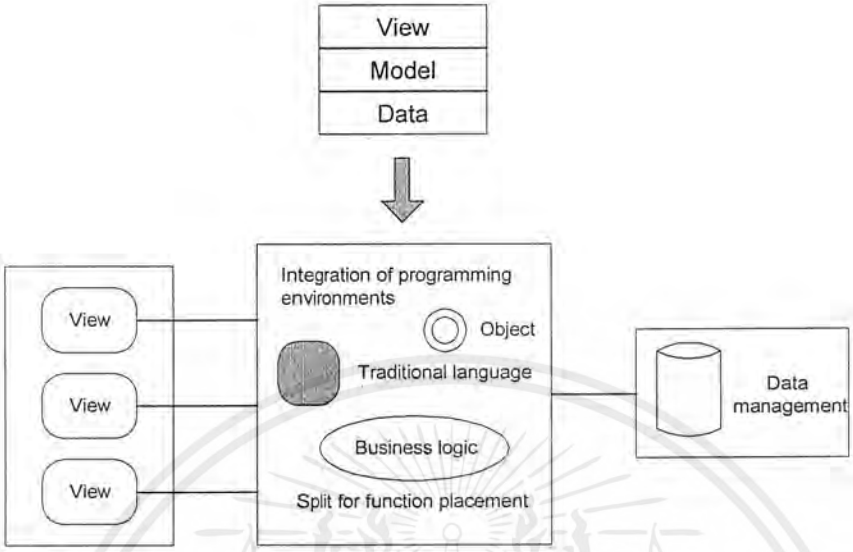
การแบ่งแอพพลิเคชันออกเป็นชั้นมีข้อดีคือ

- เป็นการหลีกเลี่ยงการทำโปรแกรมที่ซับซ้อนยุ่งเหยิง "Visual Spaghetti Programming"
- ช่วยเพิ่มความสามารถในการนำไปใช้กับระบบอื่น (portability)
- ช่วยให้การกระจายออบเจ็กต์ทำได้สะดวกขึ้น (สำหรับ distributed object application)

สถาปัตยกรรมแอพพลิเคชันแบบชั้นสามารถทำได้ทั้งแบบเปิด (open) และปิด (close) ข้อแตกต่างคือในสถาปัตยกรรมแบบเปิด ชั้นต่างๆจะสามารถใช้บริการของชั้นใดๆที่อยู่ต่ำกว่า ในขณะที่สถาปัตยกรรมแบบปิด แต่ละชั้นจะติดต่อกับชั้นที่ติดกันเท่านั้น การเลือกว่าจะใช้สถาปัตยกรรมแบบเปิดหรือปิดมักพิจารณาจากประสิทธิภาพและ Modularity

4.1.1.2 View – Model – Data Segmentation

รูป 4-2 แสดงโครงสร้างชั้นแบบ View – Model – Data ซึ่งใช้ในการ map เป็นลอจิคอลของสถาปัตยกรรม แบบ three – tier สำหรับแอพพลิเคชันแบบไคลเอนท์/เซิฟเวอร์ทั่วไป



รูป 4-2 สถาปัตยกรรมแอพพลิเคชันแบบ View – Model – Data Layer

ในชั้น View กำหนดว่าผู้ใช้จะติดต่อทำงานกับระบบได้อย่างไร โดยนำเสนอหน้าที่ของส่วนติดต่อผู้ใช้ซึ่งใช้ในการแสดงข้อมูลแก่ผู้ใช้และรับอินพุตจากผู้ใช้ในรูปของโมเดล

ชั้น Model ประกอบด้วยออบเจ็กต์หลักๆของแอพพลิเคชันซึ่งเป็นออบเจ็กต์ตามโลกแห่งความเป็นจริง (real – world object) ของแอพพลิเคชัน กำหนดพฤติกรรมของออบเจ็กต์เหล่านี้โดยไม่คำนึงว่า ผู้ใช้จะติดต่อทำงานกับออบเจ็กต์อย่างไร ออบเจ็กต์เหล่านี้จะสอดคล้องกับตรรกทางธุรกิจของแอพพลิเคชัน

ชั้น Data บริการฟังก์ชันการเข้าถึงข้อมูล จากมุมมองของผู้พัฒนาอาจมองได้ว่าเป็นส่วนขยายของ Model ชั้นนี้จะจัดการกับออบเจ็กต์คงอยู่ (persistent object)

ชั้น View จะประกอบด้วยออบเจ็กต์ส่วนติดต่อเป็นหลัก , ชั้น Model ประกอบด้วยออบเจ็กต์ของแอพพลิเคชันและออบเจ็กต์บริการ และชั้น Data ประกอบด้วยออบเจ็กต์บริการ

การแบ่งแอพพลิเคชันเป็นชั้นนี้ช่วยในการแยก Model ออกจาก View ซึ่งทำให้สามารถเชื่อมต่อ View หลายๆอันเข้ากับ Model เดียวกัน เมื่อทำการสร้างโดยใช้ VisualAge เราสามารถสร้าง View จากส่วนอินเตอร์เฟซและส่วนแอพพลิเคชัน และสามารถสร้าง Model ด้วยส่วนแอพพลิเคชันและส่วนบริการ การแยกนี้ช่วยลดความซับซ้อนในการสร้างแอพพลิเคชันด้วย VisualAge และด้วยการแยกชั้นที่สามารถใช้กับระบบอื่นได้กับชั้นที่ขึ้นอยู่กับแพลตฟอร์มยังเป็นการเพิ่ม portability ของแอพพลิเคชันอีกด้วย

ในการออกแบบโครงสร้างของแอพพลิเคชัน เราใช้ทั้งการแบ่งตามหน้าที่และแบ่งเป็นชั้น โดยเราแบ่ง Object Model ของแอพพลิเคชันออกเป็นระบบย่อย ในแต่ละระบบย่อยยังสามารถทำเป็นชั้นและในแต่ละชั้น อาจแบ่งตามหน้าที่ได้อีก เรายังสามารถรวมกลุ่มชั้นที่เหมือนกันในหน้าที่ต่างๆเป็น ชั้นสนับสนุนกลาง (common support layer) ทำงานให้กับหน้าที่ทั้งหมดเหล่านี้ ผลที่ได้ก็เป็นเช่นเดียวกับสถาปัตยกรรมแอพพลิเคชันแบบหน้าที่และชั้นทั่วๆไป

4.1.2 การเลือกแพลตฟอร์มสำหรับทำการสร้าง

ในการสร้างระบบจากวิธีการแก้ปัญหา ขั้นแรกต้องเลือกแพลตฟอร์มของระบบและโครงสร้างภายในซึ่งจำเป็นในการออกแบบแอพพลิเคชัน หรืออีกนัยหนึ่งสถาปัตยกรรมระบบแบบลอจิคอลต้องทำการ map ไปในสภาพแวดล้อมที่ใช้ในการสร้างระบบ สถาปัตยกรรมทางกายภาพของระบบจะแสดงฮาร์ดแวร์และแพลตฟอร์มที่ใช้ในการสร้างซอฟต์แวร์สำหรับวิธีแก้ปัญหาที่กำหนดไว้

4.1.2.1 การเลือกเทคโนโลยีที่สนับสนุนไคลเอนท์/เซิฟเวอร์

การใช้เทคโนโลยีไคลเอนท์/เซิฟเวอร์ในสภาพแวดล้อมที่หลากหลายจำเป็นต้องเลือกเทคโนโลยีที่สนับสนุนไคลเอนท์/เซิฟเวอร์หลายๆแบบขึ้นอยู่กับลักษณะการทำเป็นไคลเอนท์/เซิฟเวอร์ มิดเดิ้ลแวร์ (middleware) ที่ต่างกันสามารถนำมาใช้ในการสร้างแบบวิธีแก้ปัญหา,ขั้นตอนการออกแบบออบเจ็กต์คงอยู่ รวมไปถึงการขัดเกลาการออกแบบออบเจ็กต์ของแอพพลิเคชันให้รวม distributed object ในหลายกระบวนการหรือหลายแพลตฟอร์ม ซึ่งยังรวมถึงการออกแบบส่วนติดต่อระหว่างออบเจ็กต์แอพพลิเคชันทางธุรกิจกับ client/server wrapper เช่นที่มีใน VisualAge

สำหรับในระบบนี้ทำงานบนระบบปฏิบัติการ Windows NT , Oracle และ LAN

4.2 การออกแบบออบเจ็กต์ (Object Design)

การออกแบบออบเจ็กต์เป็นการกลั่นกรองรายละเอียดของออบเจ็กต์ใหม่อีกครั้ง จากขั้นตอนของการออกแบบระบบที่ได้แบ่งโปรแกรมออกเป็นระบบย่อย เราจะกำหนดงานในแต่ละระบบย่อยให้แก่กลุ่มของนักพัฒนา โดยที่แต่ละกลุ่มนั้นจะมีการทำงานที่อิสระต่อกัน จนกว่าจะถึงขั้นตอนของการรวมระบบ

ในระหว่างขั้นตอนการวิเคราะห์ สำหรับแต่ละคลาสที่อยู่ใน Object Model เราจะกำหนดแอททริบิวต์ (ซึ่งได้มาจากข้อมูลในตัวออบเจ็กต์ สำหรับการดูแลรักษาระบบ) และโอเปอเรชั่น สำหรับออบเจ็กต์ที่ถูกเรียกใช้บริการจากคลาสอื่นๆ และ Object Model จะมีการแสดงความสัมพันธ์แบบ associations ระหว่างคลาส

จากข้อมูลเหล่านี้ เราจะทำการออกแบบต่อโดยการกำหนดคลาสและกำหนดรายละเอียดของคลาสในแต่ละระบบย่อย

4.2.1 การออกแบบ Solution - Domain Classes

กลุ่มออบเจ็กต์ที่ใช้ในการรันแอพพลิเคชันจะมากกว่ากลุ่มคลาสที่เราได้ระหว่างการทำการวิเคราะห์เสมอ คลาสที่ได้จากการทำการวิเคราะห์ซึ่งกำหนดใน Object Model ในช่วงแรกๆแสดงแกนหลักของแอพพลิเคชันเท่านั้น ยังต้องทำการออกแบบคลาสอื่นๆเพื่อใช้ในการทำงานเพิ่มด้วย ตัวอย่างคลาสที่ต้องการเพิ่มเติม เช่น คลาสที่ใช้แสดงส่วนติดต่อผู้ใช้และคลาสบริการที่ให้บริการฟังก์ชั่นต่างๆ อาทิ การตรวจสอบอินพุต , การเข้าถึงฐานข้อมูล , โครงร่างการติดต่อสื่อสาร เป็นต้น ในทางปฏิบัติแล้วผู้พัฒนาอาจต้องสร้างคลาสเหล่านี้ขึ้นใหม่หรืออาจนำมาจาก Class Library เช่นเดียวกับขั้นตอนอื่นของการพัฒนาเชิงวัตถุ การออกแบบ Solution - Domain Class ต้องมีการวนพัฒนา

ในการ map คลาสเหล่านี้ใน IBM VisualAge ขอแนะนำให้ทำตามขั้นตอนดังนี้

1. map คลาสหลักๆของแอพพลิเคชันซึ่งแสดงใน Object Model ที่ได้จากการวิเคราะห์เป็น VisualAge nonvisual class
2. เพิ่มส่วนอินเตอร์เฟซและคลาสบริการเพื่อจัดการส่วนติดต่อผู้ใช้และฟังก์ชั่นการให้บริการอื่นๆ

4.2.2 การสร้างโมเดลในขั้นตอนการออกแบบออบเจ็กต์

การสร้างโมเดลในขั้นตอนการออกแบบออบเจ็กต์ใช้เทคนิคเดียวกับในขั้นตอนการวิเคราะห์ เทคนิคนี้รวมไปถึงการทำ CRC card , Event Trace Diagram , State Diagram และ Object Model แต่รายละเอียดจะเน้นไปในวิธีการแก้ปัญหามากกว่าตัวปัญหา

Design CRC แสดงหน้าที่และความขึ้นต่อกันของแต่ละออบเจ็กต์ แสดงแอททริบิวต์,ข้อความและเมธอด ช่วยให้เข้าใจหน้าที่ของแต่ละออบเจ็กต์ได้ถูกต้องมากขึ้น

Design Event Trace Diagram หน้าที่ของออบเจ็กต์แสดงในเทอมของการให้บริการซึ่งกระทำผ่านโอเปอเรชั่นของออบเจ็กต์ รายละเอียดที่แสดงในไดอะแกรมควรมีคำอธิบายหน้าที่ของโอเปอเรชั่นว่าโอเปอเรชั่นนี้ทำอะไร , โอเปอเรชั่นนี้จะเริ่มทำงานได้อย่างไร , ชื่อโอเปอเรชั่น , อินพุต/เอาท์พุตพารามิเตอร์และชนิดของอินพุต/เอาท์พุต ในขั้นตอนการวิเคราะห์เน้นที่เหตุการณ์ภายนอกที่สร้างจากผู้กระทำ แต่ในขั้นตอนการออกแบบจะเน้นที่เหตุการณ์และการทำงานร่วมกันของออบเจ็กต์ภายในระบบ เพื่อทราบรายละเอียดของการส่งข้อความระหว่าง solution - domain object

Design State Diagram ในขั้นตอนนี้จะเน้นที่การเปลี่ยนสถานะภายในออบเจ็กต์มากกว่าสถานะรวมที่เน้นในขั้นตอนการวิเคราะห์

Design Object Model จากขั้นตอนการวิเคราห์ นำ Object Diagram ที่ได้มาทำการขัดเกลาขึ้นโดยการ

- เพิ่ม/ลบคลาส โดยเพิ่มคลาสที่จำเป็นต่อแอพพลิเคชันเข้าไป ลบคลาสที่ไม่เหมาะสมออก อาจสร้างคลาสแม่ (superclass) หรือ Abstract Class เพื่อให้สนับสนุนการนำกลับมาใช้ใหม่หรือการ genelization
- ปรับปรุงลำดับการสืบทอด (inheritance)

- กำหนดชนิด , ช่วงค่าที่ใช้ , ค่าเริ่มต้น ของแอททริบิวต์เพื่อเตรียมพร้อมนำไปสร้าง
- พิจารณาความสัมพันธ์ระหว่างคลาสให้เหมาะสมยิ่งขึ้น
- กำหนด multiplicity สำหรับ association

อย่างไรก็ตามการออกแบบที่ถูกต้องไม่ได้มาจากการวนพัฒนาเพียงครั้งเดียว บางครั้งข้อผิดพลาดอาจจะยังไม่พบจนกว่าจะทำเป็นต้นแบบ (prototype) ออกมา ในทางปฏิบัติการออกแบบและการทำต้นแบบมักมีการวนพัฒนาหลายครั้งจนกระทั่งได้การออกแบบที่พอใจ เพื่อให้การออกแบบดีขึ้นผู้พัฒนาอาจใช้แพทเทิร์น (pattern) หรือเฟรมเวิร์ค (framework)

Design pattern คือการจัดรูปแบบการแก้ปัญหาให้อยู่ในแนวทางเดียวกัน เช่น design pattern แบบ Model - View - Controller , Master - Slave , Microkernal , Proxy , Observer , เป็นต้น

Framework เป็นกลุ่มของคลาสที่ได้ออกแบบมาเพื่อการนำกลับไปใช้ใหม่ในแอพพลิเคชันเฉพาะประเภทเท่านั้น เฟรมเวิร์คจะวางโครงสร้างของระบบโดยแบ่งออกเป็นออบเจ็กต์ , คลาส , หน้าที่รับผิดชอบหลัก , วิธีทำงานร่วมกันของคลาสและออบเจ็กต์ รวมทั้งกระบวนการควบคุมการทำงานทั้งหมดของระบบ เฟรมเวิร์คจะกำหนดรูปแบบในการออกแบบแอพพลิเคชัน ทำให้ผู้พัฒนาสามารถสร้างระบบขึ้นมาจากส่วนประกอบหลักๆที่เฟรมเวิร์คมีให้ ช่วยให้ผู้พัฒนาสามารถพัฒนาแอพพลิเคชันได้เร็วขึ้นและยังสามารถดูแลรักษาซอฟต์แวร์ได้ง่ายขึ้น การเปลี่ยนแปลงโครงสร้างของเฟรมเวิร์คจะมีผลกระทบต่อโครงสร้างของแอพพลิเคชันด้วย ดังนั้นในการออกแบบเฟรมเวิร์คจึงควรมีการนำแพทเทิร์นไปประยุกต์ใช้หลายๆแบบเพื่อให้เฟรมเวิร์คมีความยืดหยุ่นและสามารถขยายฟังก์ชั่นเพิ่มเติมได้ ความแตกต่างระหว่างแพทเทิร์นและ เฟรมเวิร์คมีดังนี้

- แพทเทิร์นเป็นสิ่งที่อยู่ในทางความคิดมากกว่าเฟรมเวิร์ค เนื่องจากเฟรมเวิร์คจะมีโค้ดมาให้ใช้ทำให้มีความพร้อมในการนำไปใช้หรือการนำกลับไปใช้ แต่แพทเทิร์นผู้พัฒนาจะต้องสร้างโค้ดขึ้นมาเองเมื่อต้องการใช้
- แพทเทิร์นมีส่วนประกอบเล็กกว่าเฟรมเวิร์ก โดยทั่วไปเฟรมเวิร์คจะประกอบด้วยหลายๆแพทเทิร์น
- แพทเทิร์นมีความเจาะจงน้อยกว่าเฟรมเวิร์ค โดยทั่วไปเฟรมเวิร์คนั้นได้รับการออกแบบและพัฒนาขึ้นมาสำหรับงานเฉพาะประเภทเท่านั้น แต่แพทเทิร์นสามารถนำไปใช้กับแอพพลิเคชันได้อย่างกว้างขวาง

4.2.2.1 MVC Design Pattern

MVC ประกอบด้วยออบเจ็กต์ 3 ชนิด : Model ประกอบด้วยออบเจ็กต์หลักของแอพพลิเคชัน , View เป็นการนำเสนอข้อมูลผ่านหน้าต่างส่วนติดต่อผู้ใช้และ Controller กำหนดวิธีการทำงานร่วมกันระหว่างส่วนติดต่อผู้ใช้กับอินพุตจากผู้ใช้

แนวคิดหลักของ MVC ก็คือการแยก Model (ออบเจ็กต์หลักของแอพพลิเคชัน) ซึ่งมีความสัมพันธ์ใกล้ชิดกับโดเมนธุรกิจออกจาก View โดยใช้ Controller ดูแลการใช้งานจากผู้ใช้ และเป็นตัวกลางระหว่างออบเจ็กต์ส่วนติดต่อกับออบเจ็กต์ของโมเดล เหตุผลหลักของการแยกคือในขณะที่ออบเจ็กต์ของ

โมเดลมีการเปลี่ยนแปลงน้อย แต่ View อาจมีการเปลี่ยนแปลงได้บ่อยตามความต้องการของผู้ใช้และออบเจ็กต์ของโมเดลหนึ่งสามารถสัมพันธ์กับ View ได้หลาย View ระบบที่สนับสนุน MVC จำเป็นต้องมีเฟรมเวิร์คซึ่งรับบทบาทเป็น Controller คอยอำนวยความสะดวกการสื่อสารระหว่าง View และ Model

4.2.2.2 Notification Framework

เฟรมเวิร์คชนิดนี้อาจมีชื่อเรียกและรายละเอียดการสร้างแตกต่างกันในแต่ละระบบ แต่แนวคิดหลักนั้นเหมือนกัน Notification Framework ประกอบด้วย notifier และ observer โดย notifier ทำหน้าที่รักษารายการออบเจ็กต์ที่ขึ้นอยู่กับเหตุการณ์เฉพาะที่เกิดขึ้น มีกลุ่มของส่วนติดต่อให้ออบเจ็กต์อื่นมาลงทะเบียนการขึ้นต่อเหตุการณ์หนึ่งของออบเจ็กต์เหล่านั้นหรือยกเลิกการขึ้นต่อกัน และออบเจ็กต์จะเพิ่ม observer ลงในรายการของ notifier เพื่อทำการลงทะเบียนกับ notifier

เมื่อมีเหตุการณ์เกิดขึ้น notifier จะค้นหาในรายการและส่งสัญญาณไปยัง observer ทุกตัวที่ขึ้นกับเหตุการณ์นั้น แต่เป็นความรับผิดชอบของ observer ว่าจะตอบรับหรือเพิกเฉยต่อสัญญาณนั้น

4.2.2.3 ออบเจ็กต์ควบคุม (Control Object)

การใช้ออบเจ็กต์ควบคุมนั้นไม่จำเป็นเสมอไป ในกรณีส่วนใหญ่คุณลักษณะที่อยู่ในออบเจ็กต์ส่วนติดต่อ (interface object) และ เอนติตี้ออบเจ็กต์ (entity object) ก็เพียงพอแล้ว แต่ในระบบที่ซับซ้อนบางระบบ เมื่อคุณลักษณะที่มีไม่สามารถวางอยู่ในทั้งออบเจ็กต์ส่วนติดต่อหรือเอนติตี้ออบเจ็กต์ใดๆได้ จึงจะพิจารณาสร้างออบเจ็กต์ควบคุมซึ่งมีฟังก์ชั่นการทำงานเกี่ยวกับทรานแซคชั่น (transaction) เช่น การ update จะต้องมีผลหน่วยการทำงานหรือไม่เลย

ในการวิเคราะห์ระบบ เราควรจะมีความอิสระจากขั้นตอนการสร้าง เช่น ภาษาในการเขียนโปรแกรมและรูปแบบการทำงาน แต่ในความเป็นจริงแล้วออบเจ็กต์ที่ถูกออกแบบมานั้นจะขึ้นอยู่กับโครงสร้างของภาษา ดังนั้นการออกแบบระบบจึงต้องพิจารณาถึงเป้าหมายของสิ่งแวดล้อมในการทำงานของระบบด้วย

บทที่ 5
การออกแบบออบเจ็กต์คงอยู่ (Persistent Object Design)

หากต้องการเก็บออบเจ็กต์ที่ใช้ในการเอ็กซีคิวต์โปรแกรมในอนาคตจะต้องทำการเก็บไว้ในตัวกลางถาวรซึ่งอาจมีวิธีการจัดการที่แตกต่างออกไป ออบเจ็กต์เหล่านี้เรียกว่าออบเจ็กต์คงอยู่ (Persistent Object)

ผู้พัฒนาจะพบทางเลือกในการเก็บและเรียกดูข้อมูลได้หลายวิธี แต่ละทางเลือกก็มีข้อดีข้อเสียที่แตกต่างกัน โดยมีระบบจัดการฐานข้อมูล (Database Management Systems) หลายชนิดที่สามารถนำมาใช้ได้ จะขอกล่าวถึงฐานข้อมูล 2 ชนิดคือฐานข้อมูลเชิงวัตถุ (Object - Oriented Database) และฐานข้อมูลแบบรีเลชันแนล (Relational Database)

5.1 ฐานข้อมูลเชิงวัตถุ (Object-Oriented Database)

ระบบจัดการฐานข้อมูลเชิงวัตถุ (Object-Oriented DBMS) เป็นเครื่องมือที่ออกแบบมาเพื่อใช้เก็บข้อมูลและแบ่งส่วนร่วมของออบเจ็กต์ โดยจะมี data definition language (DDL) และ data manipulation language (DML) ซึ่งภาษาทั้งสองนี้ถูกกำหนดมาเพื่อขยายความสามารถของ application programming language หรือเหมือนเป็น built-in language ที่ฝังอยู่ใน DBMS อย่างไรก็ตาม ถ้า DBMS ต้องการภาษาสำหรับการเข้าถึงฐานข้อมูลที่แตกต่างจากรูปแบบของ application programming languages แล้ว ระบบนั้นจะต้องเสียเวลารวมในการจัดการคัดลอกรูปแบบข้อมูลจาก database language format ไปเป็น programming language format และการแปลงกลับมาอีกครั้ง ซึ่งเรียกว่า impedance mismatch นอกจากนี้ ถ้า DDL และ DML ซึ่งเป็นส่วนขยายของภาษาเฉพาะ เช่น C++ แล้ว ฐานข้อมูลก็จะไม่สามารถมีการเข้าถึงได้โดยตรงจากโปรแกรมที่ถูกเขียนมาจากภาษาอื่น เช่น Smalltalk โดย built-in database language นี้สามารถที่จะกำหนดเส้นทางในการจัดการออบเจ็กต์ได้อย่างอิสระต่อกัน

ออบเจ็กต์ไม่ใช่มีเพียงแต่ข้อมูล แต่จะมีเมธอดที่ใช้ในการเข้าถึงด้วยซึ่งจะมีการจัดการที่ต่างกันในระบบจัดการฐานข้อมูลเชิงวัตถุที่ต่างกัน บางระบบจะมีการเก็บเมธอดไว้ในฐานข้อมูลหรืออาจจะเก็บไว้ในไฟล์ภายนอก (external files) ซึ่งในกรณีหลังนี้ เมื่อคลาสที่อยู่ในฐานข้อมูลถูกเปลี่ยนแปลงไป นั่นจะต้องมีการเปลี่ยนแปลงเมธอดที่อยู่ในไฟล์ภายนอกด้วย

ข้อเสียหลักของการใช้ฐานข้อมูลเชิงวัตถุในการเก็บค่าของออบเจ็กต์ คือ ต้องการระบบฐานข้อมูล 2 ตัวแยกจากกันเมื่อมีการเข้าถึงข้อมูลเป็นจำนวนมาก จากโปรแกรมที่สืบทอดมาอย่างเฉพาะ ฐานข้อมูลเชิงวัตถุจะไม่สนับสนุน Set Operations และฐานข้อมูลแบบรีเลชันแนลที่ต้องการโดยปกติ

5.2 ฐานข้อมูลแบบรีเลชันแนล (Relational Database)

ฐานข้อมูลแบบรีเลชันแนลได้ใช้รูปแบบการเก็บข้อมูลเป็นแบบตาราง ซึ่งประกอบด้วยคอลัมน์และแถว ระบบจัดการฐานข้อมูลแบบรีเลชันแนล (Relational Database Management System : RDBMS)

ช่วยในการจัดการตารางข้อมูลและโครงสร้างที่สัมพันธ์กันเพื่อเพิ่มความสามารถและประสิทธิภาพให้แก่ตาราง การเข้าถึงข้อมูลจะมี Structured Query Language (SQL) เป็นตัวอนุญาติให้ผู้ใช้แก้ไขข้อมูลได้ โดยผู้ใช้ไม่จำเป็นต้องรู้ว่ามีการทำงานอย่างไร

ฐานข้อมูลแบบรีเลชันแนลไม่สามารถสนับสนุนการเก็บออบเจ็กต์แบบต่างๆ ได้ทั้งหมด เนื่องจากว่ามันจะสามารถเก็บได้เฉพาะข้อมูลที่มีโครงสร้างเป็นแบบพื้นฐาน เช่น integer , real , blob และ string โดยจะไม่สามารถเก็บข้อมูลที่มีรูปแบบการเก็บที่ซับซ้อน ค่าของข้อมูล 1 ตัวจะสามารถเก็บได้ในตารางต่อ 1 เซลล์ และเซลล์จะไม่สามารถอ้างผ่านพอยเตอร์ (pointer) ในหน่วยความจำได้

RDBMS ถูกออกแบบมาเพื่อใช้กับทรานแซคชั่นขนาดสั้น ส่วนการจัดการกับข้อมูลที่เป็นแบบทรานแซคชั่นขนาดยาวจะต้องอ้างถึงข้อมูลรูปแบบ temporal data , history และ data versions ซึ่งจะเกินจากขอบเขตของระบบเหล่านี้ อย่างไรก็ตาม RDBMS ก็เป็นการแก้ปัญหาที่ดีเยี่ยมสำหรับการเก็บออบเจ็กต์

ข้อดีของระบบจัดการฐานข้อมูลแบบรีเลชันแนล

- ข้อมูลนำเสนอในรูปแบบตาราง
- มีภาษา SQL เป็นภาษามาตรฐาน
- มีผลิตภัณฑ์ชั้นนำสนับสนุน referential integrity และข้อกำหนดพื้นฐาน

ข้อเสียของ RDBMS ก็มีดังนี้

- ในแอพพลิเคชันที่มีการเคลื่อนย้ายจากออบเจ็กต์หนึ่งไปยังอีกออบเจ็กต์หนึ่งมาก RDBMS จะทำงานได้ช้า
- ขาดส่วนขยายเพิ่มเติมจากฟังก์ชั่นพื้นฐาน เช่น การสืบทอด (inheritance) , ทรานแซคชั่นขนาดยาว (long transaction) , schema evolution เป็นต้น
- ขาดความยืดหยุ่นในการทำ locking protocols
- สนับสนุนชนิดข้อมูลน้อย ไม่สามารถกำหนดชนิดข้อมูลใหม่ได้
- นำเสนอในรูปตารางเท่านั้น ข้อมูลของบางแอพพลิเคชันไม่เหมาะสมนำเสนอในรูปตาราง
- โปรแกรมเมอร์ส่วนใหญ่ไม่คุ้นเคย

แต่จุดแข็งของ RDBMS ก็มีดังนี้

- มีทฤษฎีรองรับ มีมาตรฐานภาษา SQL และมีการใช้อย่างแพร่หลาย
- มี Logical data independence แอพพลิเคชันต้องคำนึงถึงเฉพาะส่วนที่เกี่ยวข้องกับฐานข้อมูลเท่านั้น ทำให้สามารถเพิ่มแอพพลิเคชันใหม่บนฐานข้อมูลโดยไม่กระทบกระเทือนแอพพลิเคชันเดิมที่มีอยู่ และมี Physical data independent แยกแอพพลิเคชันจากกลไกการปรับฐานข้อมูล ทำให้สามารถจัดโครงสร้างฐานข้อมูลใหม่ให้มีประสิทธิภาพมากขึ้นโดยไม่มีผลกระทบต่อ logic ของแอพพลิเคชัน
- หากฐานข้อมูลได้ทำ index ไว้อย่างถูกต้องเหมาะสม RDBMS จะเข้าถึงข้อมูลได้อย่างรวดเร็ว

5.2.1 Normal Form

Normal Form เป็นแนวทางการออกแบบฐานข้อมูลรีเลชันแนล ช่วยให้ข้อมูลมีความสอดคล้องกันมากขึ้น

First Normal Form มีข้อกำหนดว่าข้อมูลที่อยู่ในแต่ละฟิลด์ (field) ของตารางจะต้องมีเพียงค่าเดียว ไม่มี repeating group ของ multiple value

Second Normal Form ต้องผ่าน First Normal Form และแอททริบิวต์ทุกตัวต้องขึ้นอยู่กับ primary key

Third Normal Form ตารางต้องผ่าน Second Normal Form และต้องไม่มีแอททริบิวต์ใดขึ้นอยู่กับแอททริบิวต์ที่ไม่ใช่ primary key

เราสามารถออกแบบตารางให้ได้ถึง Fifth Normal Form แต่โดยทั่วไปแล้วถึงขั้น Third Normal Form ก็เพียงพอแล้ว

5.2.2 การกำหนด Identity

ในการออกแบบเชิงวัตถุนั้น ทุกๆ ตัวแทนของคลาสหรือก็คือออบเจ็กต์จะมี object ID (OID) ถ้าหากออบเจ็กต์ใดไม่มี OID เราก็จะไม่สามารถอ้างถึงออบเจ็กต์นั้นได้ในโปรแกรม

ทุกๆออบเจ็กต์จะมี ID ที่เป็นเอกลักษณ์ จากตัวอย่างชนิดของรถยนต์ เราสามารถกำหนดชื่อเพื่อเป็นตัวแสดงถึงชนิดของรถยนต์ เพราะฉะนั้นชื่อจึงต้องมีความเป็นเอกลักษณ์ หากแต่ว่าอาจจะมีรถยนต์ 2 ชนิดที่ต้องการใช้ชื่อเหมือนกัน นั่นจะทำให้เราไม่สามารถแยกความแตกต่างของออบเจ็กต์เหล่านั้นได้

หากเราได้กำหนดให้ชื่อของชนิดรถยนต์เป็นเอกลักษณ์ ชื่อนั้นจะถูกใช้เป็นตัวอ้างอิงจากออบเจ็กต์อื่นๆ และถ้าเกิดเราจำเป็นต้องเปลี่ยนชื่อชนิดของรถยนต์แล้ว การอ้างอิงทั้งหมดที่ใช้ชื่อชนิดรถยนต์เก่าจะต้องทำการแก้ไข ไม่เช่นนั้นอาจจะทำให้เกิดการอ้างอิงไปยังออบเจ็กต์ที่ไม่มีอยู่จริงได้

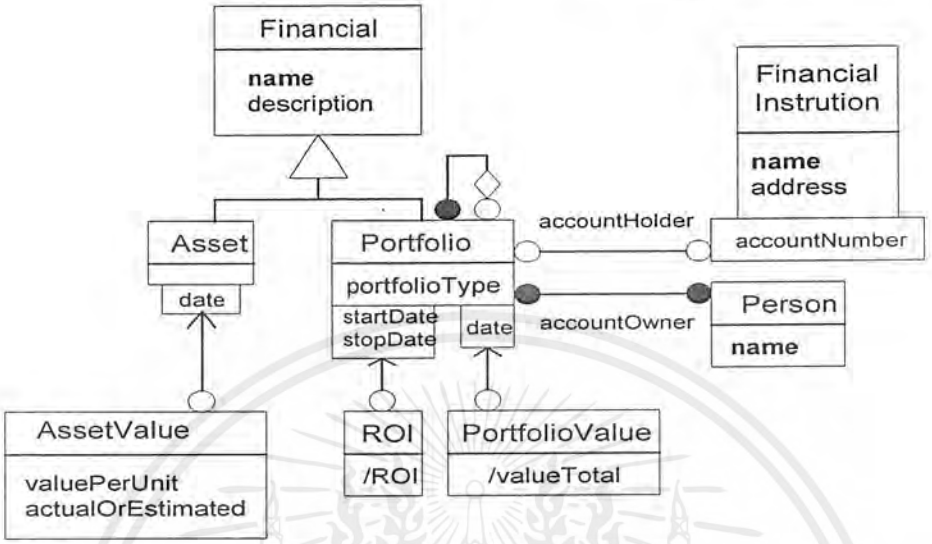
จากปัญหาข้างต้น เราอาจจะใช้หมายเลข ID ที่ไม่ซ้ำกันในการแก้ปัญหาได้ เช่นในตัวอย่าง เราจะกำหนดหมายเลข ID ให้กับรถยนต์แต่ละชนิด การอ้างอิงทั้งหมดจากออบเจ็กต์อื่นจะอ้างอิงไปยังหมายเลข ID แทนการอ้างอิงไปที่ชื่อ ดังนั้น ชื่อของชนิดรถยนต์สามารถที่จะเปลี่ยนได้โดยไม่ต้องมีการแก้ไขการอ้างอิงใหม่ทั้งหมด

5.2.2.1 Existence - Based Identity

เป็นแนวทางการสร้าง identity ที่ทำได้ง่าย โดยการเพิ่มแอททริบิวต์บ่งบอกออบเจ็กต์ (object identifier attribute) ลงในแต่ละคลาสและใช้เป็น primary key สำหรับตารางของความสัมพันธ์ primary key จะมาจากคลาสที่สัมพันธ์กัน 1 คลาสหรือมากกว่า ในทางอุดมคติควรจะใช้ identifier ค่าเดียวกันในลำดับการสืบทอดด้วย identifier อาจจะไม่แสดงใน Object Model แต่ต้องรวมอยู่ในตารางด้วย

5.2.2.2 Value - Based Identity

ถ้าหากเลือกทำแบบ Value - Based Identity ควรจะเพิ่มสัญลักษณ์การมอบ identity ลงใน Object Model เพื่อกำหนด primary key ให้แต่ละคลาสซึ่งสัญลักษณ์นี้จะใช้เฉพาะในแอพพลิเคชันที่ใช้วิธี Value - Based Identity แสดงในทุกๆคลาส ไม่ว่าจะเป็น identity ที่อยู่ภายในคลาสหรือได้มาจากคลาสอื่น



รูป 5-1 การมอบ identity แบบ Value - Based

สัญลักษณ์ที่ใช้ในการมอบ identity มีดังนี้

- คลาสอิสระ : ออบเจ็กต์ที่เป็นคลาสอิสระแสดงด้วยแอททริบิวต์ที่อยู่ภายในคลาสโดยแอททริบิวต์ที่เป็น primary key จะแสดงด้วยตัวหนา
- คลาสที่ขึ้นต่อกันด้วยความสัมพันธ์ : ออบเจ็กต์ของคลาสที่ขึ้นต่อกันได้รับ identity จากออบเจ็กต์อื่น หัวลูกศรแสดงความสัมพันธ์ที่เป็นที่มาของ identity โดยชี้ที่แหล่งกำเนิดซึ่ง multiplicity เป็น 1 เท่านั้น ปลายลูกศรอาจเป็น multiplicity แบบ many หากคลาสได้รับ identity จากหลายแหล่งกำเนิด
- คลาสที่ขึ้นต่อกันด้วย Generalization : โดยปกติ primary key ของคลาสแม่ (superclass) จะสืบทอดไปสู่คลาสย่อย (subclass) และแอททริบิวต์ของคลาสแม่ที่เป็น primary key จะแสดงด้วยตัวหนา

5.2.3 การกำหนดโดเมน

ถึงแม้ว่าโดเมนจะไม่มีใน RDBMS บางตัวแต่ก็ควรจะกำหนดในการออกแบบฐานข้อมูล โดเมนเป็นส่วนสำคัญของทฤษฎี RDBMS และ OMT Methodology ช่วยสนับสนุนให้การออกแบบมีความสอดคล้องมากกว่าการใช้ชนิดข้อมูลโดยตรง เราสามารถกำหนดข้อบังคับบนโดเมนได้ เช่น บังคับการนับ , บังคับ multiplicity ของแอททริบิวต์ สำหรับแอททริบิวต์ที่ใช้โดเมนจะต้องเพิ่มคำสั่ง Check ของ SQL ใน

ข้อบังคับโดเมนทุกข้อ โดเมนง่ายๆ เช่น name , longString , money สามารถสร้างได้โดยตรงโดยการแทนชนิดข้อมูลที่สอดคล้องและขนาดเท่านั้น แต่โดเมนที่ซับซ้อน เช่น identifier,enumeration,structured, multivalued ต้องใช้วิธีการสร้างพิเศษ

5.2.3.1 Identifier Domains

RDBMS ส่วนใหญ่สนับสนุน Existence - Based Identity ด้วยโดเมน identifier ใน RDBMS ของออราเคิล (Oracle) สามารถกำหนดลำดับของชื่อโดยใช้ syntax : CREATE SEQUENCE sequenceName เมื่อแทรกเรคคอร์ด (record) ลงไปลำดับของชื่อจะเป็นเสมือนค่าข้อมูลและออราเคิลจะสร้าง integer ลำดับต่อไป เนื่องจากออราเคิลเก็บลำดับได้ถึง integer หลักมากๆ ในทางปฏิบัติหมายเลขที่ใช้ได้จึงเสมือนมีใช้ได้เรื่อยๆ ไม่หมด

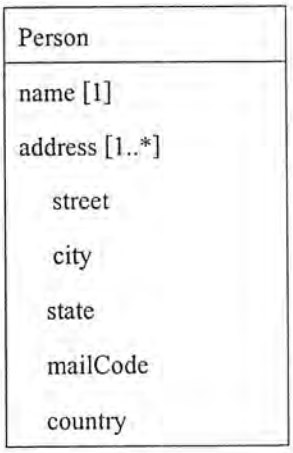
5.2.3.2 Enumeration Domains

โดเมน Enumeration มีขอบเขตอยู่ในเซ็ตของค่าที่กำหนดไว้ มี 4 วิธีในการสร้างโดเมนชนิดนี้

- Enumeration string เก็บอยู่ในรูป string สามารถกำหนดขอบเขตให้อยู่ในช่วงที่อนุญาตได้ หาก RDBMS สนับสนุนหรือสามารถบังคับค่า enumeration ได้ด้วยโค้ดของแอพพลิเคชัน
- กำหนดเป็น Boolean Attribute สำหรับ enumeration แต่ละค่า เช่นกำหนด enumeration ของ color ให้มีค่า red , yellow , green , blue เป็นต้น แต่ควรใช้ชื่อที่เข้าในได้ง่าย ปัญหาที่พบคืออาจเข้าใจไม่ชัดเจนจนกว่าจะเห็นค่า
- Enumeration Table เหมาะสำหรับ enumeration ที่มีค่ามาก อาจใช้ 1 ตารางต่อ 1 enumeration หรือ 1 ตารางใช้กับ enumeration ทั้งหมด วิธีนี้สอดคล้องกับกลยุทธ์การเก็บข้อกำหนดทางธุรกิจไว้ในฐานข้อมูลดีกว่าการฝังข้อกำหนดไว้กับโค้ดของแอพพลิเคชัน ทำให้ผู้ดูแลสามารถสร้าง enumeration ค่าใหม่โดยไม่ต้องเปลี่ยนแปลงโค้ดของแอพพลิเคชัน แต่วิธีนี้ต้องการซอฟต์แวร์พิเศษสำหรับอ่าน Enumeration table และบังคับค่า
- Enumeration Encoding ทำการเข้ารหัสค่าของ enumeration ให้เป็นตัวเลขธรรมดา แอพพลิเคชันจะทำการเข้ารหัสและถอดรหัสโดยอัตโนมัติเมื่อทำการเขียนและอ่าน สามารถใช้ Enumeration Table ในการเก็บค่าและรหัส integer ใช้เนื้อที่น้อยกว่า string หน้าต่างส่วนติดต่อผู้ใช้สามารถแสดงค่า enumeration โดยทำการเข้ารหัสและถอดรหัสภายในแอพพลิเคชันซอฟต์แวร์ วิธีนี้มีประโยชน์มากกับซอฟต์แวร์ที่ใช้หลายภาษาแต่การเข้ารหัสก็มีข้อเสียในการดูแลรักษาและการดีบัก (debug)

5.2.3.3 Structured Domains

ตัวอย่างของ Structured Domain แสดงในรูป 5-2 สามารถสร้าง Structured Domain ได้ 3 วิธีคือ



รูป 5-2 Structured Domain

1. Concatenated เก็บโดเมนเป็นชนิด string โดยไม่สนใจโครงสร้างภายใน

personID	name	Address : string

2. Multiple Columns เก็บแยกแต่ละคอลัมน์

personID	name	street	city	State	mailCode	country

3. Additional Table แยก Structured Domain ออกเป็นอีกตาราง โดยใช้ primary key และ foreign key

personID	name	AddressID (Foreign key)

addressID	street	city	state	mailCode	country

5.2.3.4 Multivalued Domains

จำนวนค่าของแอททริบิวต์หนึ่งสามารถกำหนดให้เป็น 1 ค่า [1] , ศูนย์หรือหนึ่ง [0..1] , ไม่กำหนดขอบเขตแต่มีค่าต่ำสุด [LowerLimit..*] หรืออยู่ภายในขอบเขต [LowerLimit..UpperLimit] สามารถแทนแอททริบิวต์ชนิดเป็นศูนย์หรือหนึ่งด้วย null value แต่อย่างไรก็ตาม RDBMS ไม่สามารถสร้างโดเมนชนิดนี้ได้โดยตรง ต้องใช้เทคนิคการสร้าง Structured Domain ในการสร้าง Multivalued Domain

5.2.4 การแปลงคลาสเป็นตาราง

หลักการแปลงคลาสไปเป็นตารางคือเราจะแปลงแต่ละคลาสไปเป็นหนึ่ง(หรืออาจมากกว่าหนึ่ง) ตารางโดยแต่ละแอททริบิวต์ของออบเจ็กต์จะแปลงไปเป็นคอลัมน์ของตาราง ซึ่งชนิดของแอททริบิวต์นั้นจะต้องสามารถเก็บไว้อยู่ในฐานข้อมูลแบบรีเลชันแนลได้

ในแต่ละตารางจะมี primary key ซึ่งเหมือนกับเป็นตัว unique identifier ของออบเจ็กต์ที่เก็บอยู่ในตาราง โดย primary key อาจจะไม่ใช่เป็นเลข ID แต่จะเป็นชื่อหรือวันที่หรือข้อมูลชนิดอื่นๆ ในบางโปรแกรมจะมีการใช้ artificial ID เป็นตัวแทนของ attribute ID ข้อดีของการใช้ artificial ID อยู่ที่ค่าของมันจะไม่มีการเปลี่ยนแปลงจึงมีความอิสระในการเปลี่ยนแปลงค่าข้อมูลของออบเจ็กต์และทำให้เกิดความเสถียรของ ID สำหรับความสัมพันธ์ที่ถูกอ้างจากออบเจ็กต์อื่น เช่น หากเราอ้างอิงออบเจ็กต์โดยใช้ชื่อ เราจำเป็นต้องมีการแก้ไขความสัมพันธ์ทั้งหลายที่อ้างถึงเมื่อออบเจ็กต์มีการเปลี่ยนแปลงค่าของชื่อนั้น

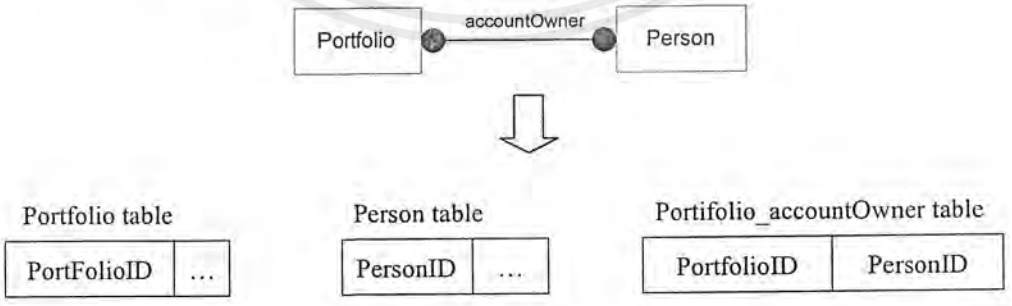
ข้อเสียของการใช้ artificial ID คือมีความลำบากในการสร้าง ID สำหรับ RDBMS ที่ไม่มีการสนับสนุน เช่น การหา ID ก่อนหน้า และการนำ ID ที่ถูกลบไปกลับมาใช้ใหม่

5.2.5 การแปลง Simple Associations

มีหลายวิธีในการแปลง association เราแบ่งวิธีเหล่านี้เป็น 3 ระดับ คือ วิธีที่แนะนำ , วิธีที่เป็นทางเลือก และวิธีที่ควรหลีกเลี่ยง ในการแปลง Object Model ควรเลือกวิธีที่แนะนำเป็นอันดับแรก อย่างไรก็ตามในบางครั้งเนื่องจากเหตุผลทางประสิทธิภาพ , สไตล์ อาจทำให้ต้องใช้วิธีที่เป็นทางเลือก และไม่ควรใช้วิธีที่ควรหลีกเลี่ยง

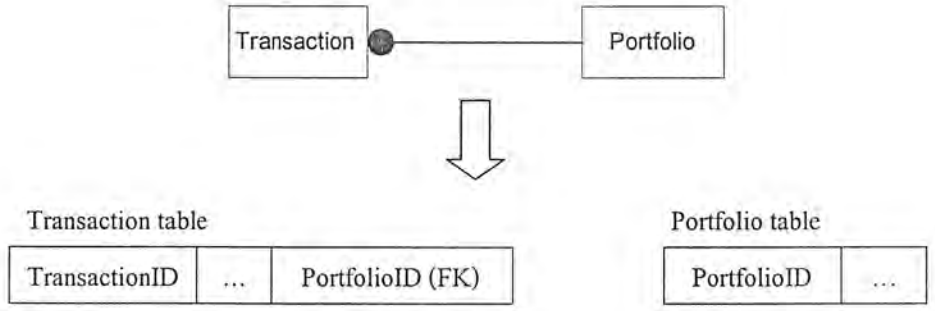
5.2.5.1 วิธีที่แนะนำในการแปลง Simple Associations

- *แยกตารางสำหรับ many - to - many association* ควรแปลงแต่ละ many - to - many association เป็น ตารางต่างหาก primary key ของความสัมพันธ์นี้ได้จากการรวม primary key จากแต่ละตาราง โดยลำดับของคลาสไม่มีผลกับ primary key และอาจมี non key ที่เป็นแอททริบิวต์ของความสัมพันธ์มาประกอบด้วย



รูป 5-3 วิธีที่แนะนำในการแปลง many – to – many association

⊙ *รวม one - to - many association* นำ primary key ของ class ที่มี multiplicity เป็น 1 ไปเป็น foreign key ในตารางของคลาสที่มี multiplicity เป็น many



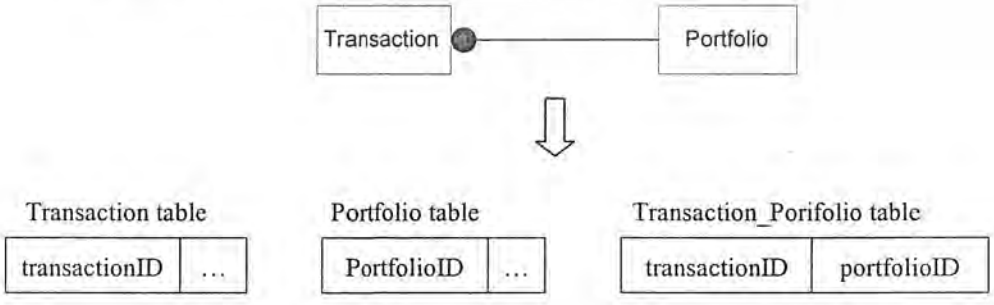
รูป 5-4 วิธีที่แนะนำในการแปลง one – to – many association

⊙ *รวม zero - or - one - exactly one association* นำ primary key ของ 1 ไปเป็น foreign key รวมกับตารางของคลาสที่เป็น 0 หรือ 1 โดยกำหนดให้ foreign key นั้นห้ามเป็น null

⊙ *รวม one - to - one association* นำ primary key ของคลาสใดคลาสหนึ่งไปเป็น foreign key ในตารางของอีกคลาสหนึ่ง foreign key จะเป็น null หรือไม่นั้นขึ้นอยู่กับ minimum multiplicity ตามที่กำหนดไว้

5.2.5.2 วิธีที่เป็นทางเลือกในการแปลง Simple Associations

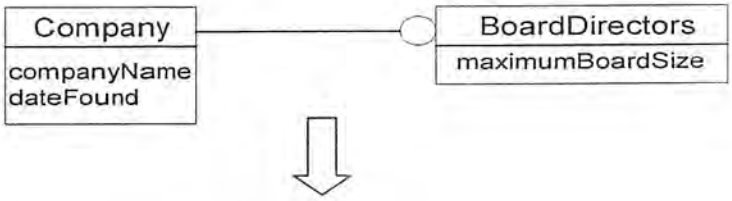
⊙ *แยกตาราง* วิธีที่แนะนำคือให้รวมตารางสำหรับ one – to – one และ one – to – many association แต่ก็สามารถสร้างเป็นตารางต่างหากได้ โดยใน one – to – many association เลือก foreign key ของ many มาเป็น primary key ของตารางความสัมพันธ์ ใน one – to – one association สามารถเลือกจากคลาสใดก็ได้ ข้อดีของการแยกตารางคือช่วยให้การออกแบบเป็นไปในทางเดียวกันและสามารถขยายเพิ่มเติมได้หากมีการเปลี่ยน multiplicity ของความสัมพันธ์ สิ่งที่ต้องทำคือกำหนดข้อบังคับใหม่เท่านั้น ไม่จำเป็นต้องเปลี่ยนโครงสร้างของตาราง แต่ข้อเสียคือทำให้ฐานข้อมูลกระจาย



รูป 5-5 วิธีที่เป็นทางเลือกในการแปลง one – to – many association

5.2.5.3 วิธีที่ควรหลีกเลี่ยงในการแปลง Simple Associations

⊙ *การรวมหลายๆคลาสและ association ลงในตารางเดียวกัน* ได้ 1 ตาราง วิธีนี้ขัดแย้งกับวิธีที่แนะนำซึ่งได้ 2 ตารางอย่างเห็นได้ชัด ข้อเสียที่สำคัญของวิธีนี้คืออาจไม่ผ่าน Third Normal Form นอกจากนี้การรวมหลายๆคลาสเข้าด้วยกันก็ยังขัดแย้งกับหลักการ Object – Oriented ที่ว่าแต่ละคลาสควรจะเป็นเอกภาพ



Company_BoardOfDirectors table

companyName	dateFounded	maximumBoardSize

รูป 5-6 วิธีที่ควรหลีกเลี่ยง : รวม class เข้ากับความสัมพันธ์

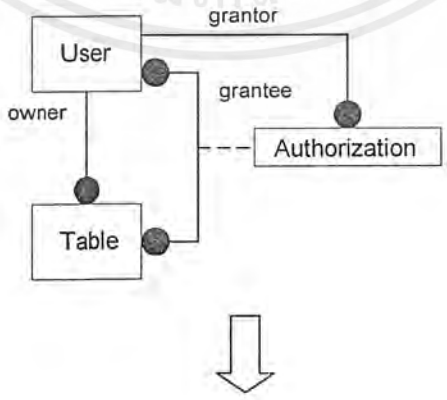
⊙ *การรวม foreign key เข้ากับทั้ง 2 ตารางใน one – to – one association* ทำให้เกิดความซ้ำซ้อน แต่ก็มีข้อดีเนื่องจาก one – to – one association มักมีในโมเดลน้อย และการ query ส่วนใหญ่ก็เป็นการรวมทั้ง 2 ตารางซึ่งต้องใช้การ join

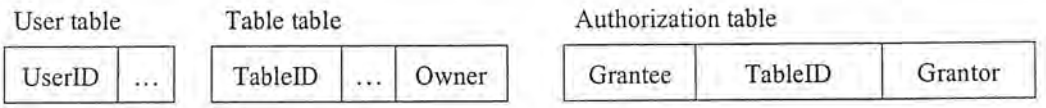
5.2.6 การแปลง Advanced Associations

5.2.6.1 วิธีที่แนะนำในการแปลง Advanced Associations

⊙ *Link Attributes* แยกตารางต่างหากสำหรับ association พร้อมด้วย Link Attributes

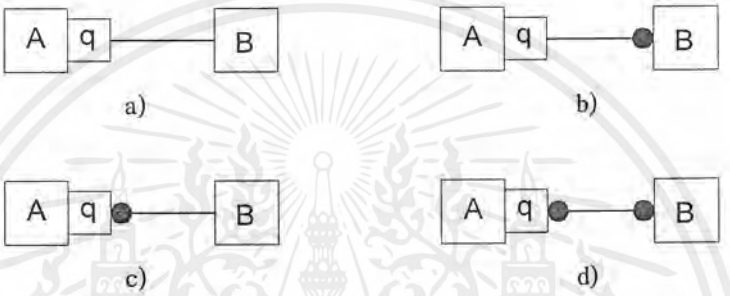
⊙ *Association Classes* สร้างตารางสำหรับ Association class แยกต่างหากอินสแตนซ์ของ association class ได้จากคลาสที่สัมพันธ์กัน





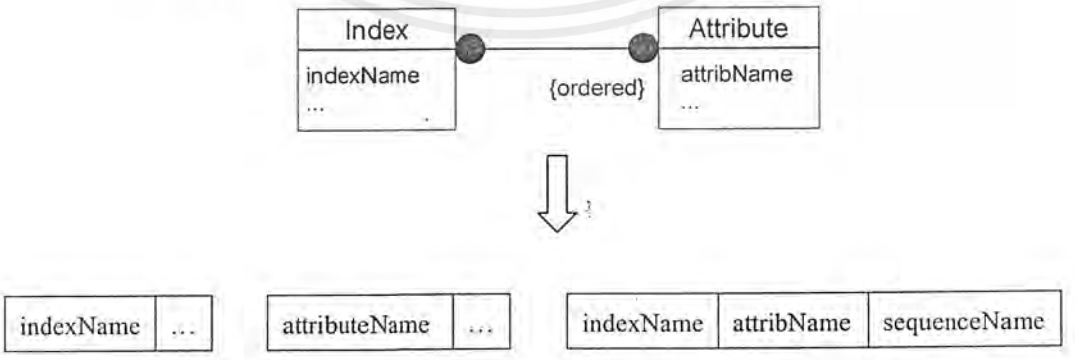
รูป 5-7 วิธีที่แนะนำในการแปลง Association Class

- ⊙ *Ternary Associations* สร้างตารางแยกแต่ละ Ternary Association ต่างหาก
- ⊙ *Qualified Associations* ทำตามหลักการแปลง Simple Association ขอแนะนำให้สร้างตารางของ Qualified Association ในรูป 5-8 a) และ 5-8 b) ให้รวมไว้ใน B class แต่ในตัวอย่าง c) และ d) ควรแยกตารางเนื่องจากมี many – to – many อยู่ primary key ของ d) ได้จากการรวม primary key ของ A,B และ q



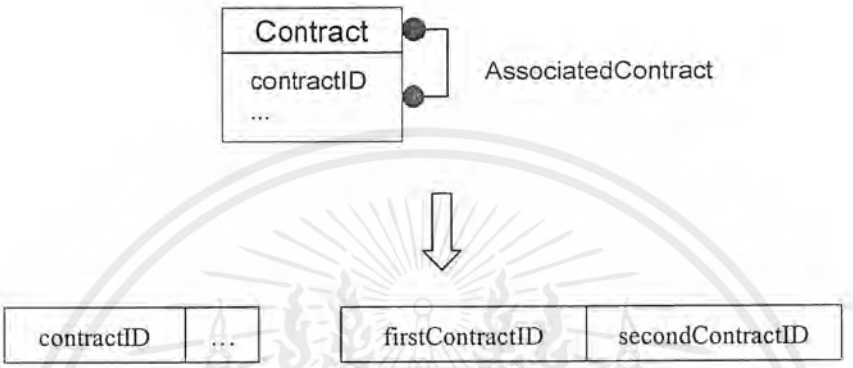
รูป 5-8 multiplicity ที่เป็นไปได้ของ Qualified Association

- ⊙ *Ordered Associations* ควรสร้างตารางของ Ordered Association โดยการมี attribute แสดงหมายเลขลำดับ ดังตัวอย่างในรูป 5-9 แอททริบิวต์จะเรียงลำดับตามความสำคัญในการสร้าง index เช่น index ของสมุดบันทึกหมายเลขโทรศัพท์จะให้ความสำคัญกับ last name มากกว่า first name ในตาราง IndexAttribute มี candidate key 2 ตัวคือ indexName + attribName และ indexName + sequenceNumber โดยเลือก indexName + attribName เป็นprimary key



รูป 5-9 วิธีที่แนะนำในการแปลง Ordered Association

⊙ *Symmetric Associations* ความสัมพันธ์ชนิดนี้ไม่ค่อยมีบ่อยนัก แต่ก็ให้เกิดปัญหาในการสร้าง เช่นในตาราง AssociatedContract หากไม่แตกความสัมพันธ์ออกจะต้องเก็บข้อมูลหรือค้นหาในตารางซ้ำ 2 ครั้ง หาก contractA สัมพันธ์กับ contractB สามารถเก็บเรคคอร์ด (A,B) และ (B,A) ทำให้เกิดความซ้ำซ้อน หรือหากต้องการหา contract ทั้งหมดที่เกี่ยวข้องกับ A สามารถทำได้โดยหา record ที่มี firstContractID เท่ากับ A และต่อมาก็หาเรคคอร์ดที่มี secondContractID เท่ากับ A แต่หากเป็นไปได้ควรจะเปลี่ยนโมเดลเพื่อหลีกเลี่ยงความสัมพันธ์แบบนี้



รูป 5-10 วิธีที่แนะนำในการแปลง Symmetric Association

⊙ *Aggregations* ก็เป็น association อย่างหนึ่ง จึงสามารถแปลงได้โดยใช้กฎเดียวกับ association

5.2.6.2 วิธีที่เป็นทางเลือกในการแปลง Advanced Associations

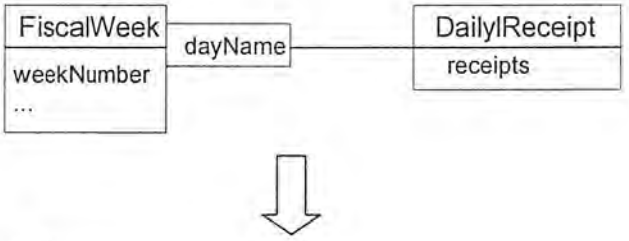
⊙ *Link Attributes* สามารถแปลงความสัมพันธ์แบบ one – to – one และ one – to – many ให้มี link attribute อยู่ด้วยโดยใช้การรวม foreign key และ link attribute เข้าไป แต่อย่างไรก็ตามการรวม link attribute ทำให้ไม่ผ่าน Second Normal Form ใน one – to – many และไม่ผ่าน Third Normal Form ใน one – to - one ด้วยเหตุนี้จึงแนะนำให้สร้างตารางสำหรับ association และ link attribute แยกต่างหาก

⊙ *Qualified Associations* สามารถสร้างตารางสำหรับ Qualified Association ที่เป็น one – to – many ต่างหากซึ่งสามารถกำจัด qualifier ออกได้ ช่วยในการบังคับ uniqueness

5.2.6.3 วิธีที่ควรหลีกเลี่ยงในการแปลง Advanced Associations

⊙ *Qualified Associations* ไม่ควรแปลงให้มี parallel attribute ซึ่งมักจะเกิดกับ Qualified Association ที่มี enumerated qualifier ดังตัวอย่างในรูป 5-11 ข้อดีเพียงอย่างเดียวของการทำ parallel attribute คือได้ตารางน้อย แต่อย่างไรก็ตามก็จะยากต่อการค้นหา ทำให้โปรแกรมมี

ความซับซ้อน และไม่ยืดหยุ่น หากต้องการเพิ่มค่าของแอททริบิวต์เข้าไปก็ต้องเพิ่มคอลัมน์ในตารางด้วย



การแปลงที่ควรหลีกเลี่ยง

FiscalWeek table

WeekNumber	...	MonReceipt	TueReceipt	WedReceipt	ThuReceipt	FriReceipt

การแปลงที่ดีกว่า

FiscalWeek table

weekNumber	...

DailyReceipt table

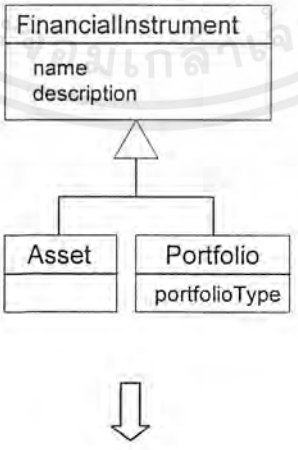
weekNumber	dayName	receipt

รูป 5-11 วิธีที่ควรหลีกเลี่ยงในการแปลง Qualified Association

5.2.7 การแปลง Single Inheritance

5.2.7.1 วิธีที่แนะนำในการแปลง Single Inheritance

โดยปรกติจะทำการแปลง Generalization ให้เป็นตารางคลาสแม่และคลาสย่อย ในทางอุดมคติแล้วออบเจ็กต์ควรจะได้ primary key ผ่านทางลำดับการสืบทอด อย่างไรก็ตาม RDBMS ไม่อาจรับประกันได้ว่าแต่ละเรคคอร์ดของคลาสแม่จะสอดคล้องกับบางเรคคอร์ดของคลาสย่อยและอาจบังคับในส่วนของการสืบทอดไม่ได้ ดังนั้นจึงต้องเพิ่มหน้าที่ส่วนนี้เข้าไปในโค้ดของแอพพลิเคชัน



FinancialInstrument table

FinancialInstrumentID	name	description	financialInstrumentType

Asset table

assetID

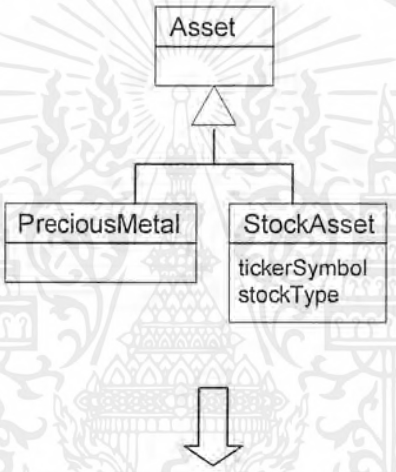
Portfolio table

portfolioID	portfolioType

รูป 5-12 วิธีที่แนะนำในการแปลง Single Inheritance

5.2.7.2 วิธีที่เป็นทางเลือกในการแปลง Single Inheritance

⊙ *กำจัดตารางของคลาสย่อยที่ไม่มีแอททริบิวต์* ข้อดีคือจะมีตารางน้อยลง 1 ตาราง แต่ข้อเสียคือการแปลงจะไม่เป็นมาตรฐานรูปแบบเดียวกัน



Asset table

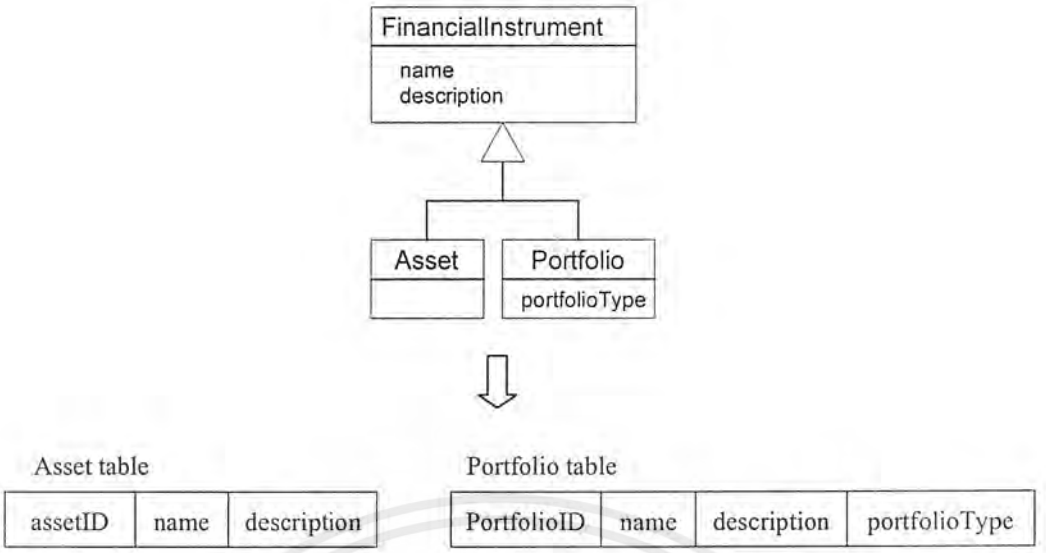
assetID	assetType

StockAsset table

stockAssetID	tickerSymbol	stockType

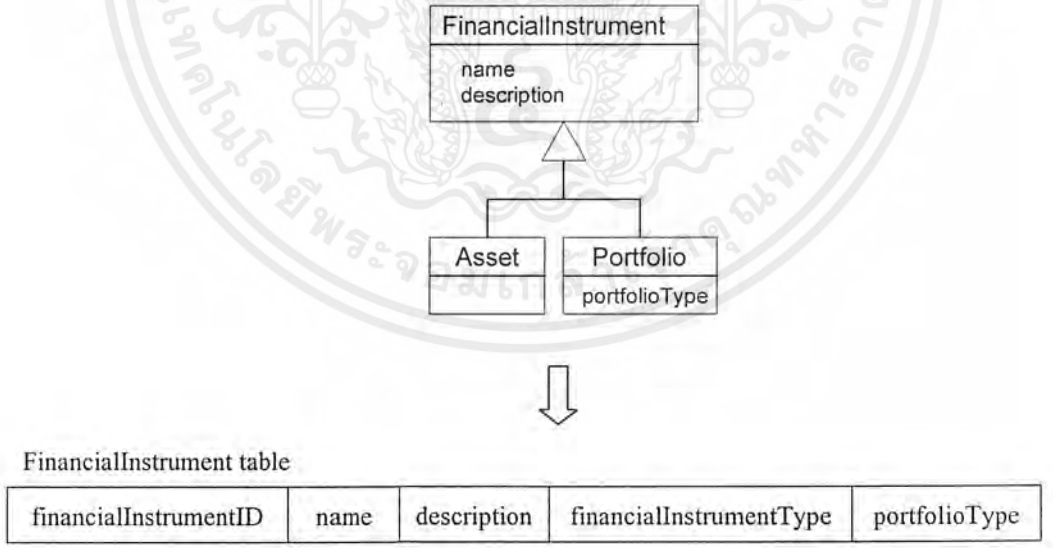
รูป 5-13 กำจัดตารางคลาสย่อยที่ไม่มีแอททริบิวต์

⊙ *ยุบแอททริบิวต์ของคลาสแม่รวมลงไปในแต่ละตารางของคลาสย่อย* ข้อดีคือแต่ละออบเจ็กต์จะแสดงด้วยตารางเพียงตารางเดียว จึงไม่จำเป็นต้อง join ตารางเพื่อมาประกอบเป็นออบเจ็กต์ ถึงแม้ว่าการแปลงแบบนี้จะผ่าน Normal Form แต่ก็ทำให้เกิดความซ้ำซ้อน และยิ่งมีลำดับการสืบทอดหลายขั้นแล้ว การแปลงแบบนี้ทำให้สูญเสียโครงสร้างในการอธิบายการสืบทอดไป



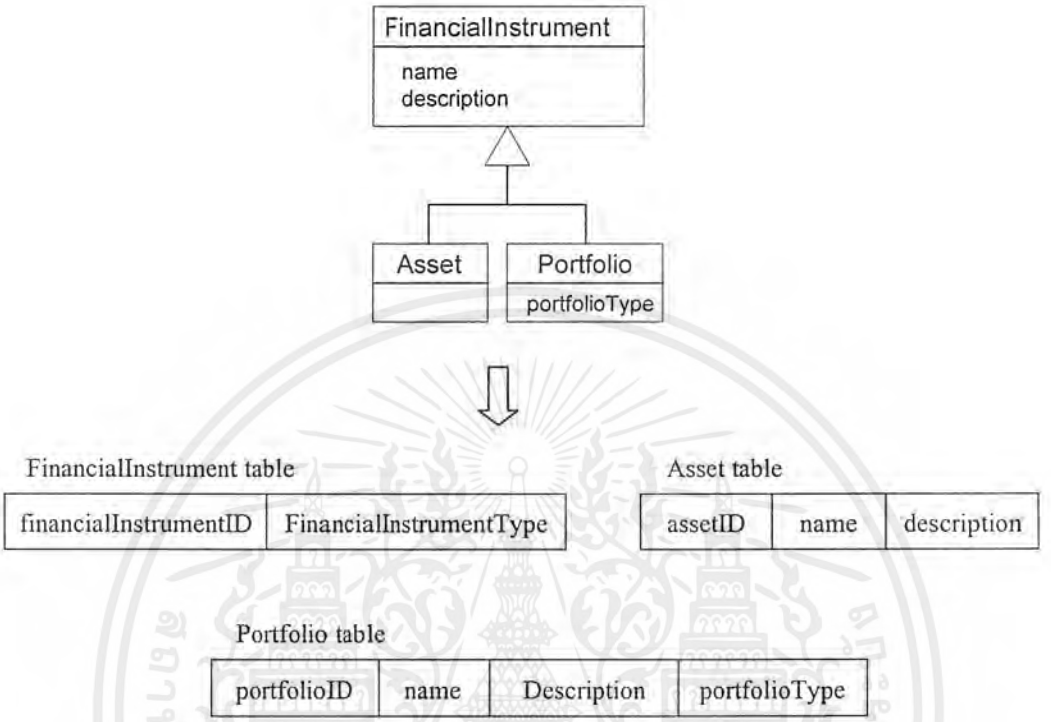
รูป 5-14 ยุบแอททริบิวต์ของคลาสแม่รวมลงไปในแต่ละตารางของคลาสย่อย

- *ดึงแอททริบิวต์ของคลาสย่อยมารวมอยู่ในตารางคลาสแม่* ทำให้ไม่ต้องมีตารางของคลาสย่อยและสามารถแสดงออบเจ็กต์ได้โดยใช้เพียง 1 ตาราง แต่วิธีนี้ไม่ผ่าน Second Normal Form เนื่องจากแอททริบิวต์บางตัวไม่ขึ้นอยู่กับ primary key และเช่นเดียวกันหากมีลำดับการสืบทอดหลายขั้นแล้ว การแปลงวิธีนี้จะทำให้สูญเสียโครงสร้างในการอธิบายการสืบทอด



รูป 5-15 ดึงแอททริบิวต์ของคลาสย่อยไปรวมกับคลาสแม่

⊙ *วิธีผสม* โดยการยุบแอททริบิวต์ของคลาสแม่ลงไปและเก็บตารางของคลาสแม่ไว้ด้วยเพื่อใช้ในการชี้นำคลาสย่อย ข้อดีคือสามารถแสดงออบเจ็กต์ได้โดยใช้เพียง 1 ตารางและตารางคลาสแม่ยังแสดงตารางที่เก็บออบเจ็กต์ไว้ด้วย แต่ข้อเสียก็คือความซ้ำซ้อนและสูญเสียการอธิบายโครงสร้างการสืบทอด แต่ในบางครั้งวิธีนี้ก็มีประโยชน์



รูป 5-16 วิธีผสม

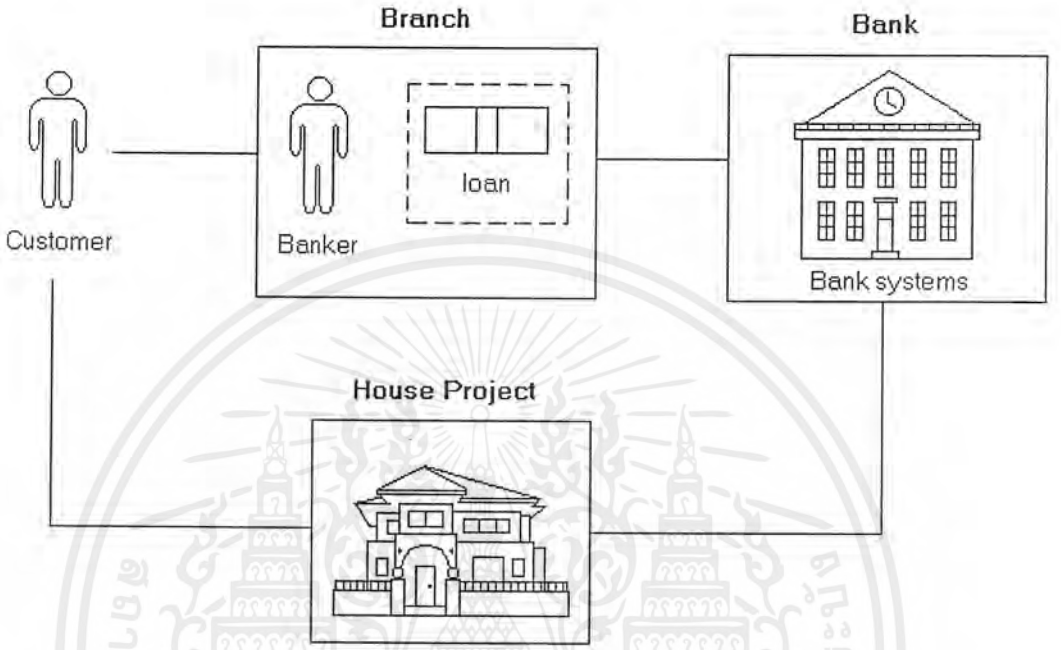
⊙ ตารางการสืบทอด วิธีนี้จะมีตารางคลาสแม่ , ตารางของแต่ละคลาสย่อยและตารางการสืบทอด โดยตารางการสืบทอดจะเชื่อมโยง primary key ของคลาสแม่เข้ากับ primary key ของแต่ละคลาสย่อย แต่ตารางการสืบทอดทำให้ schema และฐานข้อมูลมีการใช้คำมากเกินไป แต่อย่างไรก็ตามเทคนิคนี้ก็มีประโยชน์ในการรวมฐานข้อมูล

5.2.8 การแปลง Multiple Inheritance

ควรแปลงโดยการสร้างตารางคลาสแม่และตารางสำหรับแต่ละคลาสย่อย

บทที่ 6
ระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

6.1 ประโยคปัญหา (Problem Statement)



รูปที่ 6-1 ระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

ระบบเงินกู้เพื่อการผ่อนชำระบ้านจัดสรรเป็นโปรแกรมช่วยการทำงานในฝ่ายสินเชื่อของธนาคาร โดยมีฟังก์ชันการทำงานที่เกี่ยวข้องกับการขอกู้เงินและการผ่อนชำระ ระบบสามารถคำนวณวงเงินกู้สูงสุดและเงินผ่อนชำระต่องวดแสดงเป็นตัวอย่างเพื่อช่วยในการตัดสินใจของลูกค้า,สามารถเก็บประวัติลูกค้าและแก้ไขเมื่อมีการแจ้งเปลี่ยนแปลงจากลูกค้า,เก็บรายละเอียดโครงการบ้าน,รับชำระเงินผ่อนแต่ละงวด,ตรวจสอบประวัติการชำระเงินของลูกค้าและติดตามรายชื่อลูกค้าที่ชำระเงินล่าช้า

ถ้าคำร้องขอกู้เงินของลูกค้าผ่าน ทางธนาคารก็จะเก็บประวัติของลูกค้าและรายละเอียดของโครงการบ้านที่ทำสัญญาลงในฐานข้อมูล พร้อมทั้งเปิดบัญชีเงินกู้ในนามของลูกค้าด้วย

เมื่อลูกค้ามาติดต่อกับธนาคารก็จะกรอกรายละเอียดลงในแบบฟอร์ม พนักงานธนาคารก็จะป้อนข้อมูลตามในแบบฟอร์มนั้นลงในระบบ และเมื่อพนักงานธนาคารต้องการเรียกดูข้อมูลต่างๆก็จะแสดงผ่านฟอร์มทางหน้าจอ

การประมาณวงเงินกู้จะ พิจารณาจากรายได้ครอบครัวต่อเดือน โดยอ้างอิงกับข้อกำหนดของทางธนาคารและอยู่ในช่วง 100,000 – 10,000,000 บาท ลูกค้าสามารถขอกู้เงินได้ไม่เกินวงเงินกู้สูงสุด โดยสามารถเลือกระยะเวลาการผ่อนชำระได้แบบ 5,10,15 หรือ 20 ปี

การประมาณเงินชำระต่องวดจะพิจารณาจากจำนวนเงินที่ลูกค้ากู้ อัตราดอกเบี้ยเงินกู้ในขณะที่ลูกค้าขอกู้และระยะเวลาการผ่อนชำระ จำนวนเงินที่ลูกค้าจะต้องชำระต่องวดประกอบด้วยส่วนของเงินต้นและส่วนดอกเบี้ย โดยส่วนดอกเบี้ยจะคิดจากอัตราดอกเบี้ยเงินกู้ปัจจุบันอ้างอิงกับข้อกำหนดของทางธนาคารซึ่งใช้อัตราตามที่ธนาคารแห่งประเทศไทยกำหนด ธนาคารทำงานวันจันทร์ – ศุกร์

เงินผ่อนแต่ละงวดจะต้องชำระก่อนวันสิ้นเดือนและต้องชำระเป็นเงินสด ในกรณีปรกติจำนวนเงินขั้นต่ำที่ลูกค้าต้องชำระแต่ละงวดจะคงที่ตามที่ได้ประมาณไว้ ยกเว้นในกรณีที่ลูกค้าชำระเงินล่าช้าก็จะคิดดอกเบี้ยอัตราปรับเพิ่มขึ้นด้วยตามจำนวนวันที่ล่าช้า ลูกค้าสามารถชำระเงินเกินกว่าที่ประมาณไว้ได้ ส่วนที่เกินมาจะนำไปหักออกจากส่วนเงินต้น

ทางธนาคารจะเก็บจำนวนเงินที่ลูกค้าชำระในแต่ละงวดไว้ในฐานข้อมูล เพื่อใช้เป็นประวัติการชำระเงินของลูกค้า

หากลูกค้าชำระเงินล่าช้าเกินกว่า 90 วันนับจากวันชำระเงินครั้งสุดท้าย จะถูกยึดบ้านเป็นของธนาคาร

6.2 รายละเอียดข้อมูล , การคำนวณในระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

6.2.1 ตารางเทียบอัตราส่วนที่ใช้ในการคำนวณเงินชำระขั้นต่ำต่องวด

โดยพิจารณาจากอัตราดอกเบี้ยเงินกู้และระยะเวลาการผ่อนชำระ

	7%	7.5%	8%	8.5%	9%
5 ปี	0.0198012	0.0200380	0.0202764	0.0205166	0.0207584
10 ปี	0.0116109	0.0118702	0.0121328	0.0123986	0.0126676
15 ปี	0.0077530	0.0092702	0.0095566	0.0098474	0.0101427
20 ปี	0.0059383	0.0080560	0.0083645	0.0086783	0.0089973

ตารางที่ 6-1 ตาราง เทียบอัตราส่วนที่ใช้ในการคำนวณเงินชำระขั้นต่ำต่องวด

6.2.2 การประมาณเงินขั้นต่ำที่ต้องชำระต่องวด

สมมติว่าลูกค้าต้องการกู้เงิน 800,000 บาท ระยะเวลาการผ่อน 15 ปี

อัตราดอกเบี้ยเงินกู้ในขณะขอกู้ 8.5% นำเทียบหาอัตราส่วนจากตารางในข้อ 6.2.1

เงินชำระต่อเดือน 800,000 x 0.0098474 = 7,877.92 บาท

เงินเผื่อเสี่ยงเนื่องจากอัตราดอกเบี้ยเปลี่ยนแปลงประมาณ 10% ของเงินที่ชำระต่อเดือน

รวมเงินที่ชำระต่อเดือน 7877.92 + 787.792 = 8,665.712 บาท → 8,666 บาท/เดือน

6.2.3 จำนวนเงินที่ต้องชำระในแต่ละงวด

สมมติว่าลูกค้าขอกู้เงินตั้งแต่วันที่ 1 ม.ค.42 และต้องชำระเงินภายในเดือนมกราคม แต่วันที่ 30 และ 31 ม.ค. เป็นวันหยุดเสาร์-อาทิตย์ จึงต้องมาชำระเงินภายในวันที่ 29 ม.ค.

เงินชำระงวดที่ 1 8,666 บาท

เป็นส่วนดอกเบี้ย 28 วัน (1/1/42 – 28/1/42) ในวันที่ชำระเงิน 29 ม.ค. ไม่นับมาคิดในส่วนดอกเบี้ย (800,000 x 8.5% x 28) / 365 = 5,216.438 บาท

เป็นส่วนเงินต้น 3,449.561 บาท

เงินต้นคงเหลือ 796,550.439 บาท

เงินชำระงวดที่ 2 8,666 บาท มาชำระในวันที่ 26 ก.พ. 42 (วันที่ 27-28 ก.พ. เป็นวันเสาร์-อาทิตย์)

เป็นส่วนดอกเบี้ย 28 วัน (29/1/42 – 25/2/42) ในวันชำระเงิน 26 ก.พ. ไม่นับมาคิดในส่วนดอกเบี้ย (796,550.439 x 8.5% x 28) / 365 = 5,193.945 บาท

เป็นส่วนเงินต้น 3,472.055 บาท

เงินต้นคงเหลือ 793,078.384 บาท

6.2.4 กรณีชำระเงินล่าช้า

ในงวดชำระเงินที่ 3 ลูกค้าชำระเงินล่าช้า ไปชำระในวันที่ 16 เม.ย. เมื่อลูกค้าชำระเงินช้า ธนาคารจะคิดอัตราดอกเบี้ยปรับ 15%

เงินชำระงวดที่ 3

เป็นส่วนดอกเบี้ยของงวดที่ 3 34 วัน (26/2/42 - 31/3/42)

(793,078.384 x 8.5% x 34) / 365 = 6,279.442 บาท

ดอกเบี้ยปรับ 15% คิดถึงวันที่ 15 เม.ย.

(793,078.384 x 15% x 15) / 365 = 4,888.839 บาท

ดังนั้นลูกค้าจึงต้องชำระเงินในส่วนดอกเบี้ย 11,168.281 บาท

และลูกค้าชำระเฉพาะส่วนดอกเบี้ยเท่านั้น เงินต้นคงเหลือจึงเท่าเดิม

เงินชำระงวดที่ 4 8,666 บาท ชำระในวันที่ 30 เม.ย.

เป็นส่วนดอกเบี้ย 14 วัน (16/4/42 – 29/4/42)

(793,078.384 x 8.5% x 14) / 365 = 2,585.652 บาท

เป็นส่วนเงินต้น 6,080.348 บาท

เงินต้นคงเหลือ 786,998.036 บาท

6.3 Use Case Model

6.3.1 กำหนดผู้กระทำ (Actors)

ผู้กระทำของระบบงานนี้คือเจ้าหน้าที่ธนาคาร เพราะเป็นผู้ที่ใช้งานระบบโดยตรง ส่วนลูกค้าจะติดต่อระบบผ่านเจ้าหน้าที่ธนาคารเท่านั้น จึงไม่ถือเป็นผู้กระทำ

6.3.2 กำหนด Use Cases

Use cases ของระบบงานนี้ ได้แก่

ประเมินวงเงินกู้และเงินชำระต่องวด แสดงวงเงินกู้และประมาณเงินชำระต่องวด ตามอัตราดอกเบี้ยเงินกู้,จำนวนเงินที่ลูกค้าขอกู้และระยะเวลาการผ่อนชำระ เพื่อเป็นตัวอย่างให้ลูกค้าเข้าใจและช่วยในการตัดสินใจของลูกค้า

เปิดการขอกู้ใหม่ เมื่อลูกค้าขอกู้เงินและทางธนาคารอนุมัติ ระบบก็จะ

- *เก็บประวัติของลูกค้า* รายละเอียดประวัติลูกค้าบันทึกไว้ในฐานข้อมูล
- *เปิดบัญชีเงินกู้ของลูกค้า* จำนวนเงินที่ขอกู้,อัตราดอกเบี้ย ฯลฯ เก็บไว้ในฐานข้อมูล
- *บันทึกรายละเอียดโครงการบ้าน* เก็บรายละเอียดโครงการบ้านที่ลูกค้าทำสัญญาไว้ลงในฐานข้อมูล

แก้ไขประวัติลูกค้า หากลูกค้ามีการเปลี่ยนแปลงรายละเอียด เช่น ที่อยู่,ชื่อ,นามสกุล ก็ต้องมาแจ้งการเปลี่ยนแปลงกับทางธนาคาร เพื่อทำการแก้ไขประวัติให้สอดคล้องกัน

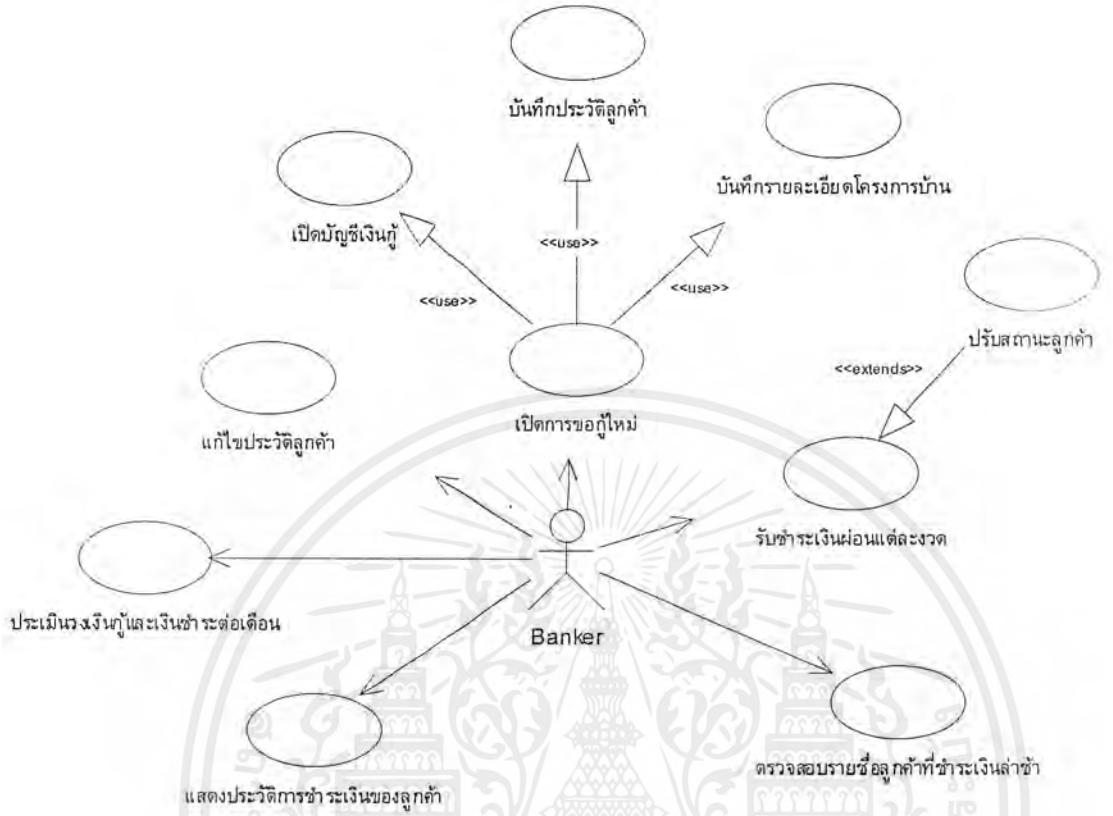
รับชำระเงินผ่อนแต่ละงวด คำนวณส่วนเงินต้น ส่วนดอกเบี้ย เงินต้นคงเหลือ และจะเก็บรายละเอียดของเงินที่ลูกค้าชำระแต่ละงวดไว้ในฐานข้อมูล ในกรณีปรกติจำนวนเงินขั้นต่ำที่ลูกค้าต้องชำระจะคงที่ แต่ในกรณีล่าช้าลูกค้าจะต้องจ่ายในส่วนดอกเบี้ยอัตราปรับด้วย

- *ปรับสถานะลูกค้า* ปรับสถานะของลูกค้าว่าเป็นปรกติ,ล่าช้าหรือถูกยึด

แสดงประวัติการชำระเงินของลูกค้า พนักงานธนาคารสามารถเรียกดูประวัติการชำระเงินของลูกค้าแต่ละคนได้

ตรวจสอบรายชื่อลูกค้าที่ชำระเงินล่าช้า ลูกค้าที่ชำระเงินล่าช้ากว่ากำหนดจะมีชื่อปรากฏอยู่ในบัญชีรายชื่อลูกค้าที่ชำระเงินล่าช้า พนักงานธนาคารสามารถตรวจสอบจากบัญชีรายชื่อนี้เพื่อออกจดหมายเตือนได้

6.3.3 Use Case Diagram ของระบบเงินกู้เพื่อการผ่อนชำระบ้านจัดสรร

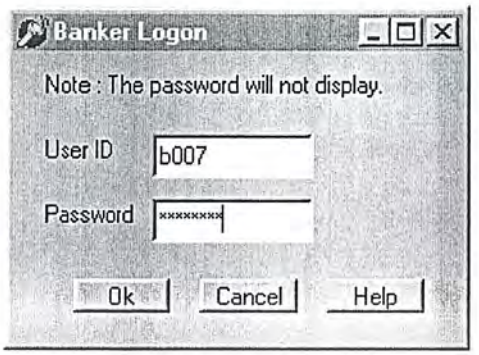


รูปที่ 6-2 Use Case Model ของระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

6.4 ต้นแบบส่วนติดต่อผู้ใช้

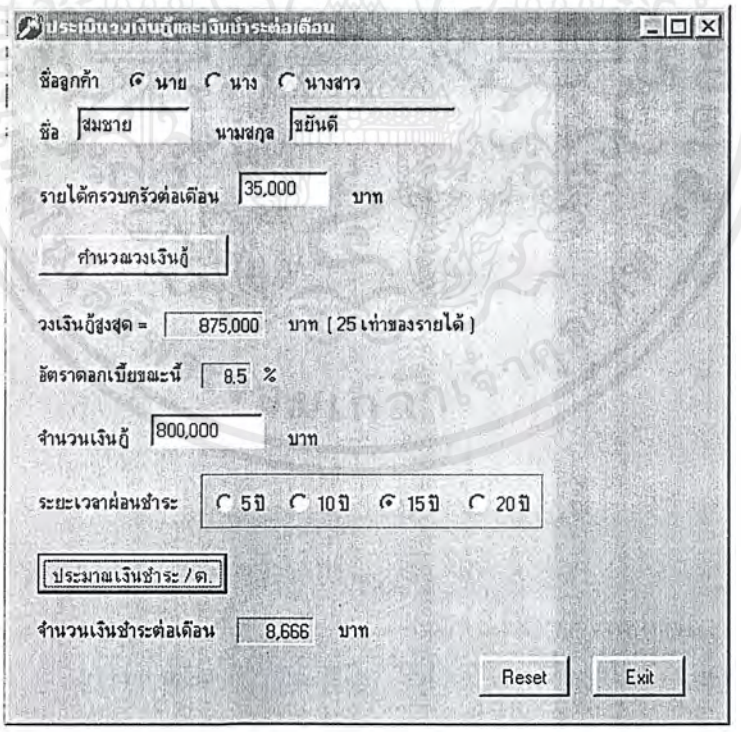
สร้างต้นแบบของระบบโดยให้ผู้ใช้ทดลองใช้งานจากต้นแบบที่สร้างขึ้นมานี้ เพื่อใช้เป็นเครื่องมือช่วยในการหาความต้องการของระบบได้อย่างถูกต้องมากยิ่งขึ้น หาส่วนที่เข้าใจผิดพลาดไปหรือส่วนที่ยังหาไม่พบในตอนแรก

6.4.1 ผู้ใช้ login เข้าสู่ระบบ จะปรากฏฟอร์มนี้เมื่อเริ่มใช้งานโปรแกรม หากรหัสประจำตัวผู้ใช้หรือรหัสผ่านไม่ถูกต้อง จะไม่สามารถเข้าสู่โปรแกรมได้



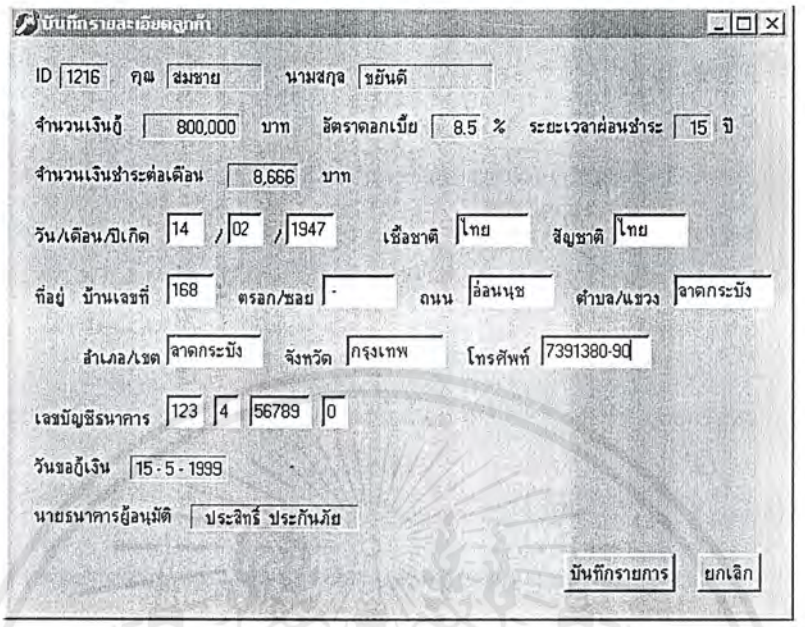
รูปที่ 6-3 ฟอร์มรับรหัสประจำตัวผู้ใช้และรหัสผ่าน

6.4.2 ประเมินวงเงินกู้และประมาณเงินชำระต่องวด เพื่อเป็นตัวอย่างช่วยในการตัดสินใจของลูกค้า



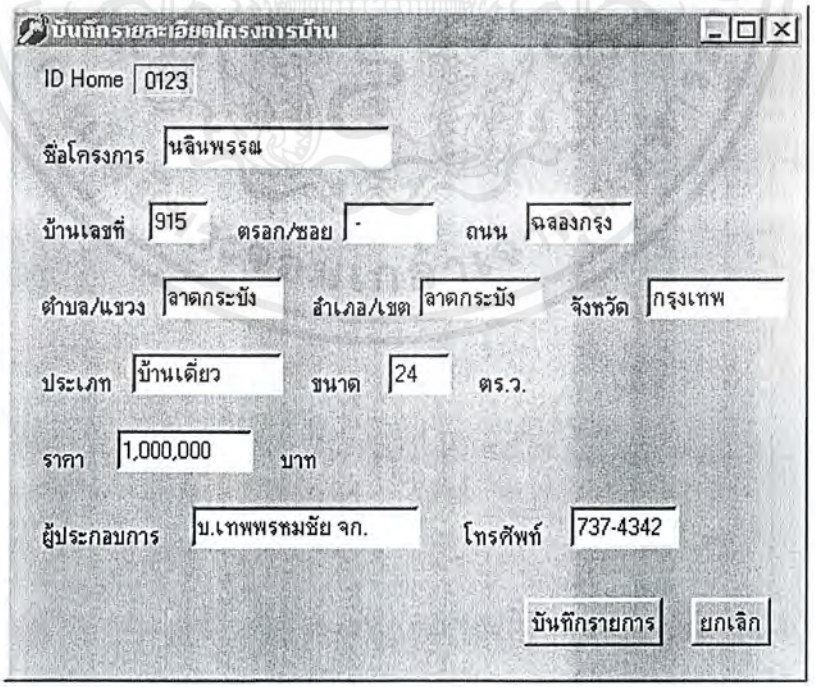
รูปที่ 6-4 ฟอร์มประมาณวงเงินกู้และเงินชำระต่องวด

6.4.3 บันทึกประวัติของลูกค้า เมื่อคำร้องขอกู้เงินของลูกค้าผ่านการอนุมัติ ทางธนาคารจะเก็บประวัติของลูกค้าไว้ในฐานข้อมูล



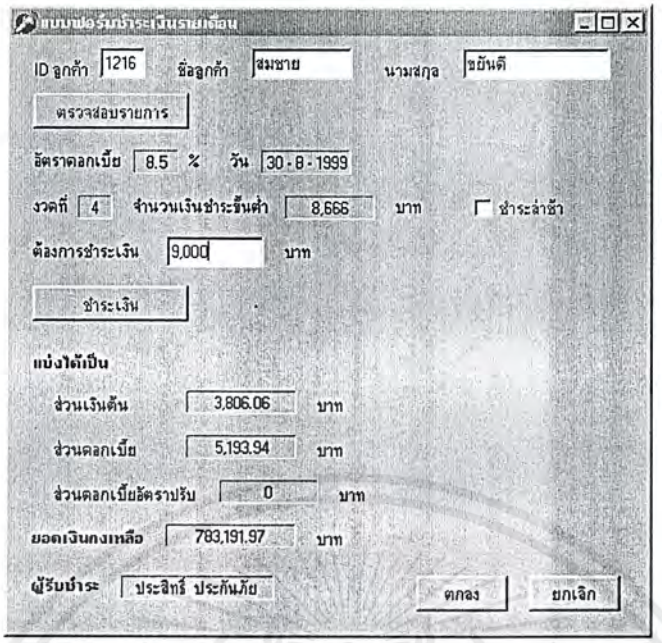
รูปที่ 6-5 ฟอร์มรับประวัติลูกค้า

6.4.4 บันทึกรายละเอียดโครงการบ้านจัดสรร



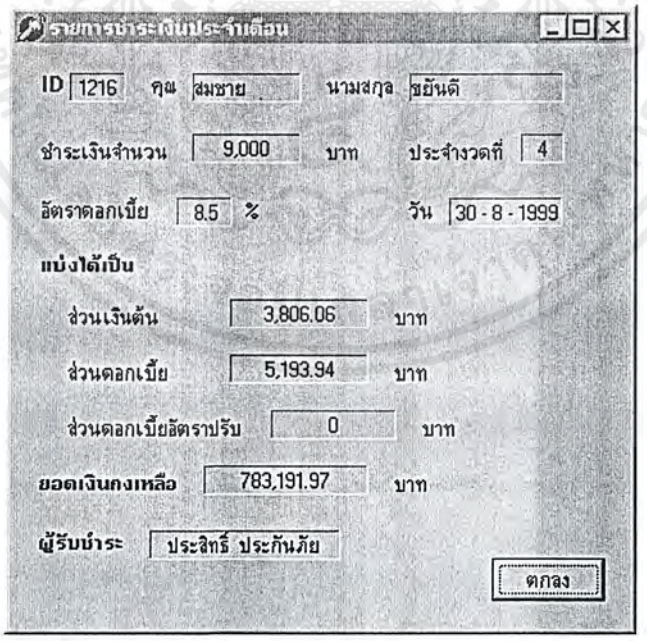
รูปที่ 6-6 ฟอร์มรับรายละเอียดโครงการบ้านจัดสรร

6.4.5 แบบฟอร์มรับชำระเงินผ่อนแต่ละงวด



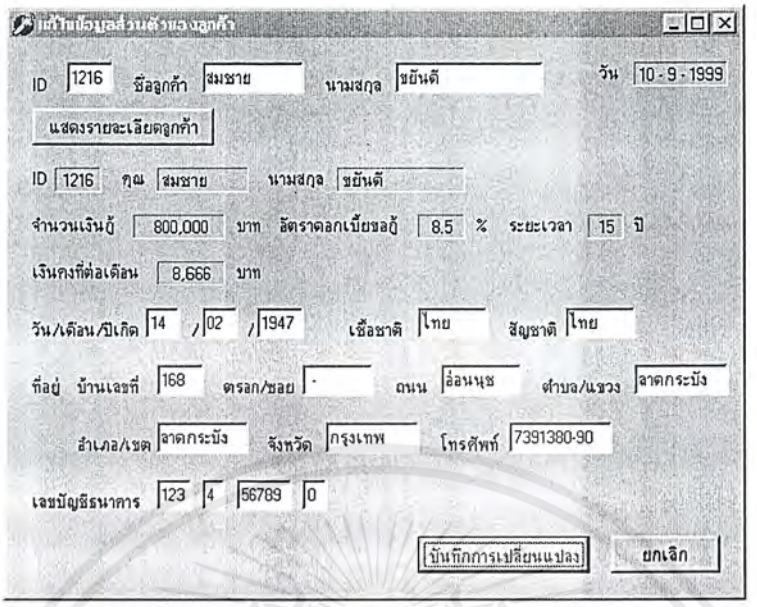
รูปที่ 6-7 ฟอร์มรับชำระเงินผ่อนแต่ละงวด

6.4.6 แบบฟอร์มสรุปจำนวนเงินที่ลูกค้าชำระในงวดนั้น



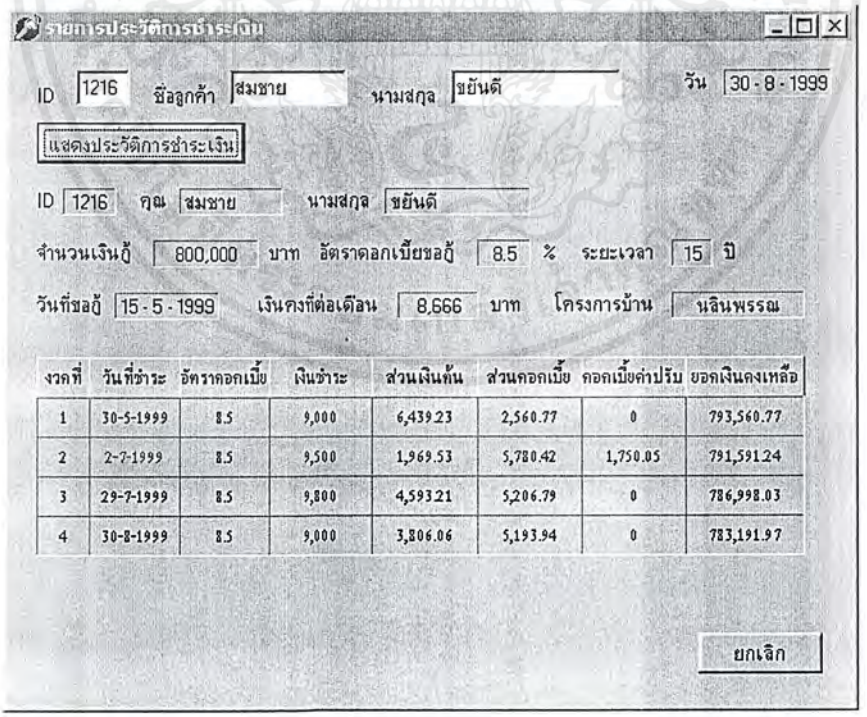
รูปที่ 6-8 ฟอร์มสรุปจำนวนเงินที่ลูกค้าชำระ

6.4.7 แก้ไขประวัติของลูกค้า



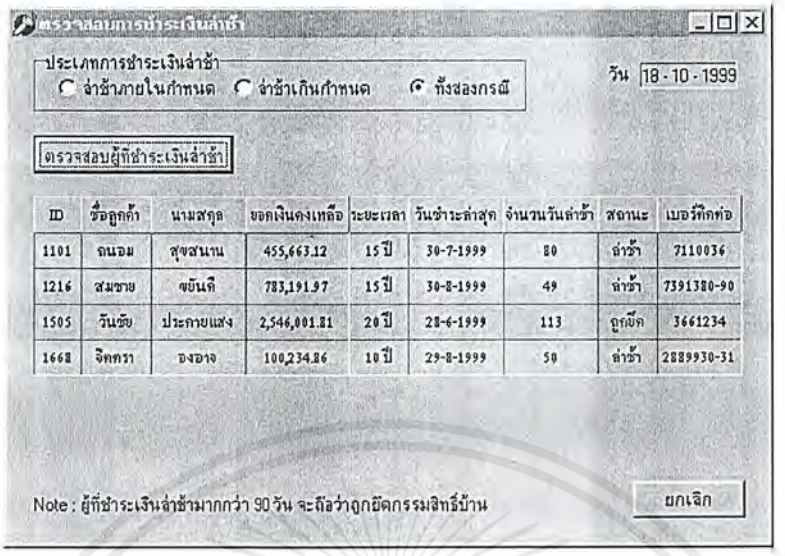
รูปที่ 6-9 ฟอร์มรับข้อมูลเพื่อแก้ไขประวัติลูกค้า

6.4.8 แสดงประวัติการชำระเงินของลูกค้า



รูปที่ 6-10 ฟอร์มแสดงประวัติการชำระเงินของลูกค้า

6.4.9 ตรวจสอบรายชื่อลูกค้าที่ชำระเงินล่าช้า



รูปที่ 6-11 ฟอร์มแสดงรายชื่อลูกค้าที่ชำระเงินล่าช้า

6.5 Object Model

6.5.1 กำหนดออบเจ็กต์

ออบเจ็กต์ที่ได้จากประโยคปัญหาและจาก Use Cases มีดังนี้

ลูกค้า , ธนาคาร , รายได้ครอบครัวต่อเดือน , วงเงินกู้ , ธนาคารแห่งประเทศไทย , อัตราดอกเบี้ยเงินกู้ , ประวัติส่วนตัวของลูกค้า , ประวัติการชำระเงิน , โครงการบ้าน , ระยะเวลาการผ่อนชำระ , เงินผ่อนชำระต่องวด , ส่วนเงินต้น , ส่วนดอกเบี้ย , จำนวนเงินที่ลูกค้าชำระ , เงินต้นคงเหลือ , ดอกเบี้ยอัตราปรับ , พนักงานธนาคาร , บัญชีเงินกู้ , จำนวนเงินกู้ , เงินชำระขั้นต่ำต่อเดือน , ฝ่ายสินเชื่อ , แบบฟอร์ม , ข้อกำหนดของทางธนาคาร , เงิน

6.5.2 กำจัดออบเจ็กต์ที่ไม่จำเป็นและไม่ถูกต้อง

จากออบเจ็กต์ที่ได้ข้างต้น เราจะกำจัดคลาสที่ไม่จำเป็นและไม่ถูกต้องออก ดังนี้

6.5.2.1 ออบเจ็กต์ที่ซ้ำซ้อนกัน (Redundant Objects) กำจัดออบเจ็กต์ที่เกินความจำเป็น ซึ่งได้แก่

ลูกค้า , ประวัติส่วนตัวลูกค้า	เป็น	ลูกค้า
บ้าน , โครงการบ้าน	เป็น	โครงการบ้าน
ฝ่ายสินเชื่อ , ธนาคาร	เป็น	ธนาคาร

6.5.2.2 ออบเจ็กต์ที่ไม่เกี่ยวข้องกับระบบ (Irrelevant Objects) กำจัดออบเจ็กต์ที่อยู่นอกเหนือระบบ ซึ่งได้แก่ ธนาคารแห่งประเทศไทย

6.5.2.3 ออบเจ็กต์ที่คลุมเครือ (Vague Objects) กำจัดออบเจ็กต์ที่มีความหมายไม่ชัดเจน ซึ่งได้แก่ เงิน , ประวัติการชำระเงิน

6.5.2.4 ออบเจ็กต์ที่เป็นแอททริบิวต์ของคลาส กำจัดออบเจ็กต์ที่เป็นคุณลักษณะของคลาส ซึ่งได้แก่ รายได้ครอบครัวต่อเดือน , วงเงินกู้ , อัตราดอกเบี้ยเงินกู้ , ระยะเวลาการผ่อนชำระ , เงินต้นคงเหลือ , ดอกเบี้ยอัตราปรับ , ส่วนเงินต้น , ส่วนดอกเบี้ย , จำนวนเงินกู้ , เงินชำระขั้นต่ำต่อเดือน

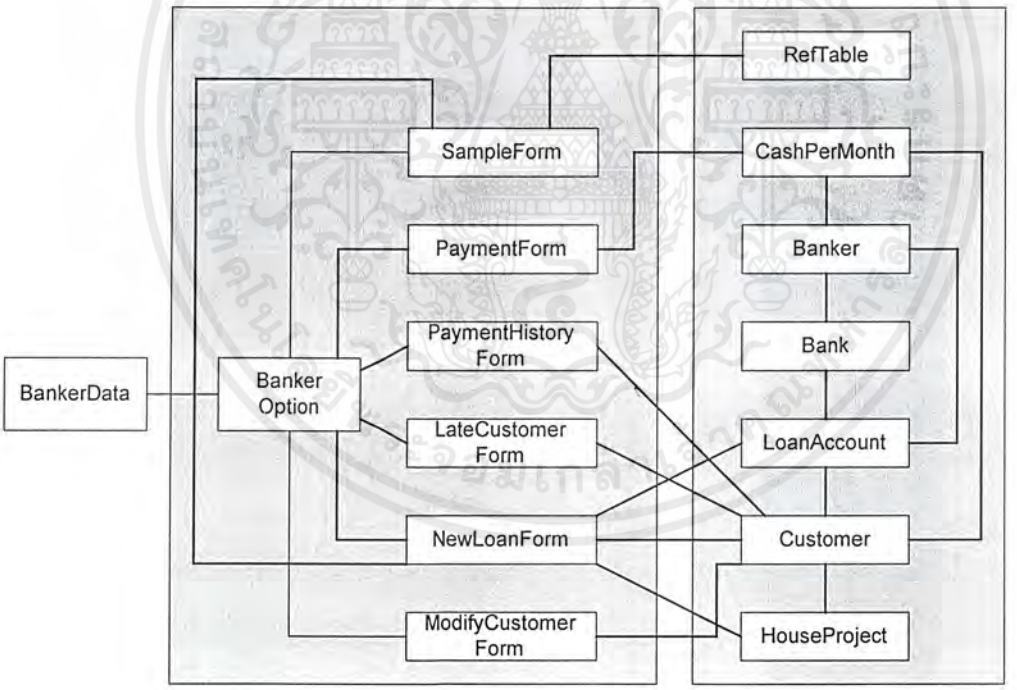
6.5.2.5 ออบเจ็กต์ที่เป็นโอเปอเรชั่นของคลาส กำจัดออบเจ็กต์ที่เป็นพฤติกรรมของคลาส

6.5.2.6 ออบเจ็กต์ที่เป็นหน้าที่ของคลาส กำจัดออบเจ็กต์ที่แสดงหน้าที่บางอย่างของคลาส

6.5.2.7 ออบเจ็กต์ที่สร้างจากระบบ (Implementation Construct) กำจัดออบเจ็กต์ที่เป็นสิ่งที่ถูกทำขึ้นจากระบบ

6.5.3 Object Model ของระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

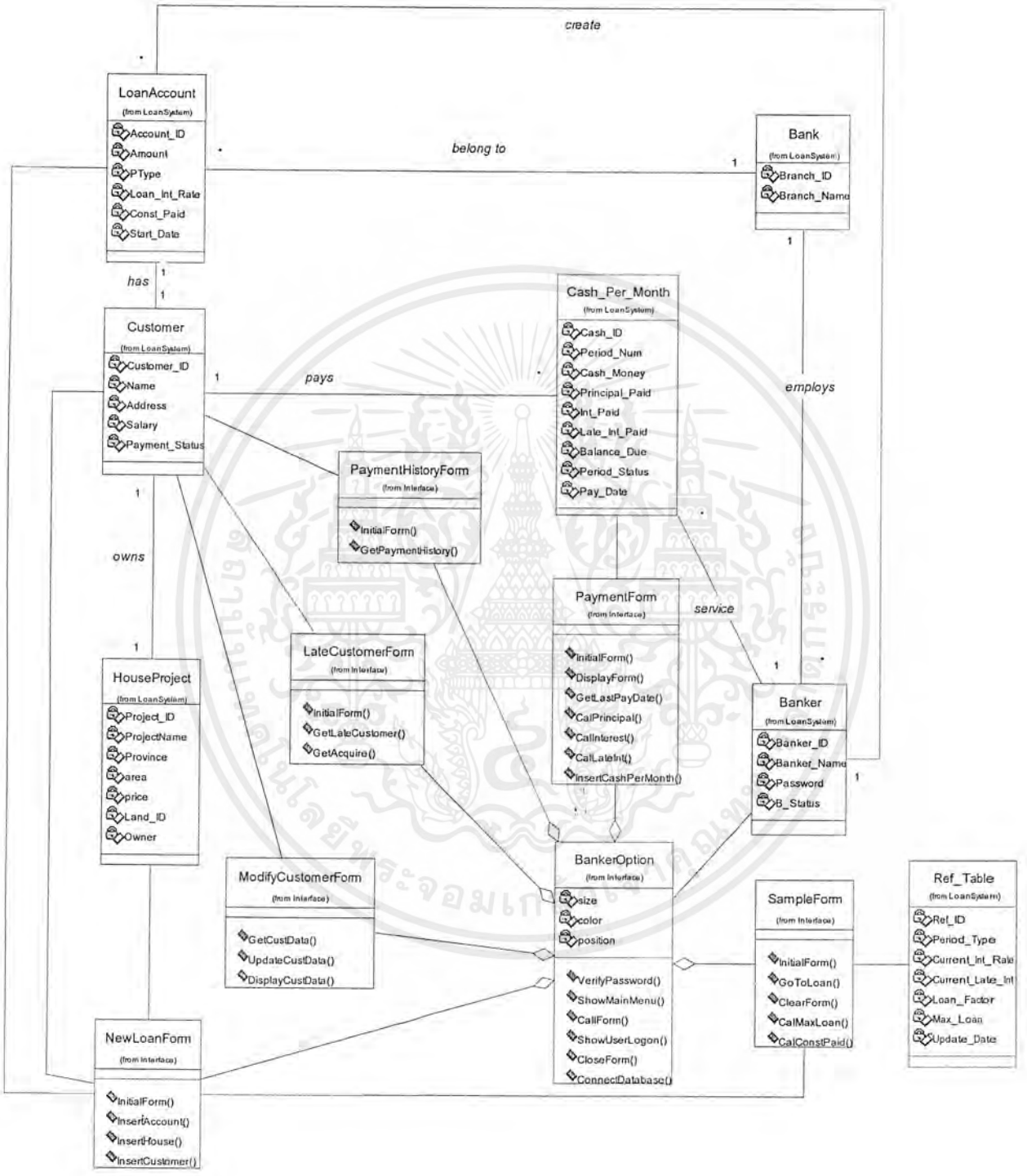
จากออบเจ็กต์ที่ได้นำมาเขียน Object Model ของระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร โดยได้มีการพัฒนาแก้ไขโมเดลหลายครั้งเพื่อให้ได้โมเดลที่ตรงตามความต้องการของระบบมากที่สุด



รูปที่ 6-12 Object Model ของระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

6.6 Class Diagram

เมื่อทำการวิเคราะห์เพิ่มก็สามารถกำหนดคลาสของออบเจ็กต์ที่ได้ กำหนดแอททริบิวต์,เมธอด และความสัมพันธ์ระหว่างคลาสต่างๆ เขียนเป็น Class Diagram ได้ดังนี้

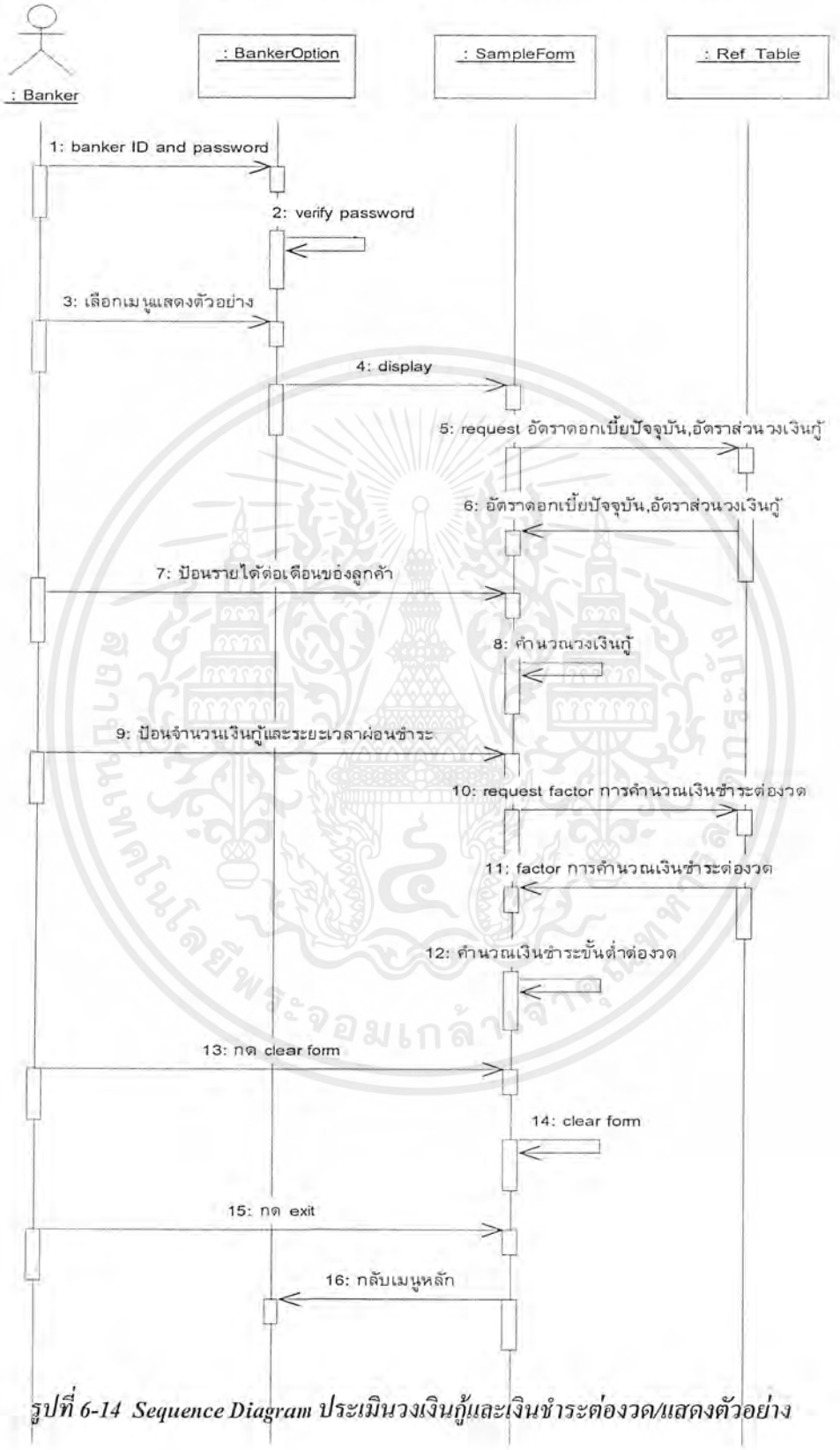


รูปที่ 6-13 Class Diagram ของระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

6.7 Dynamic Model

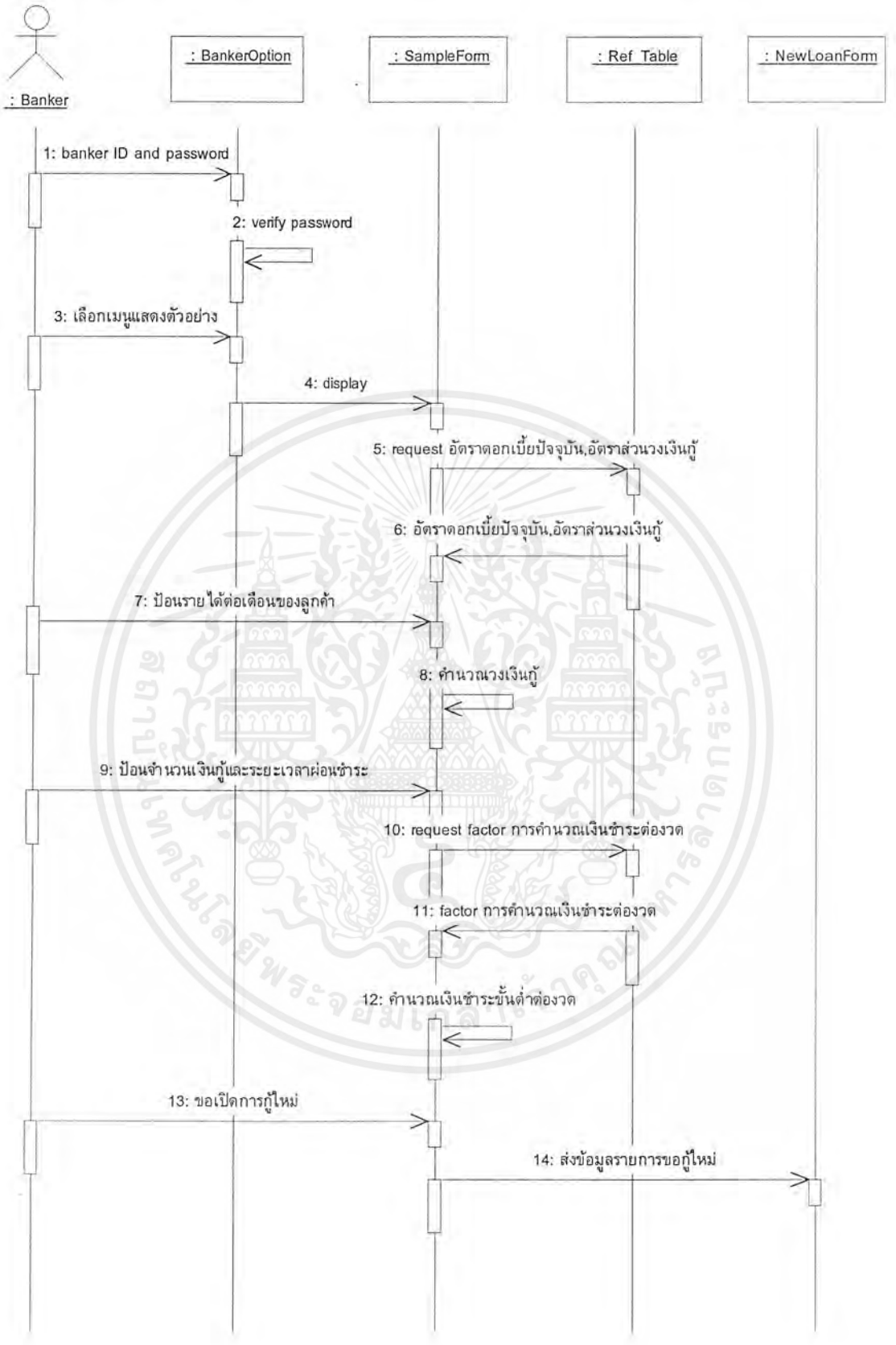
6.7.1 Sequence Diagram

6.7.1.1 Sequence Diagram : ประเมินวงเงินกู้และเงินชำระต่องวด/แสดงตัวอย่าง



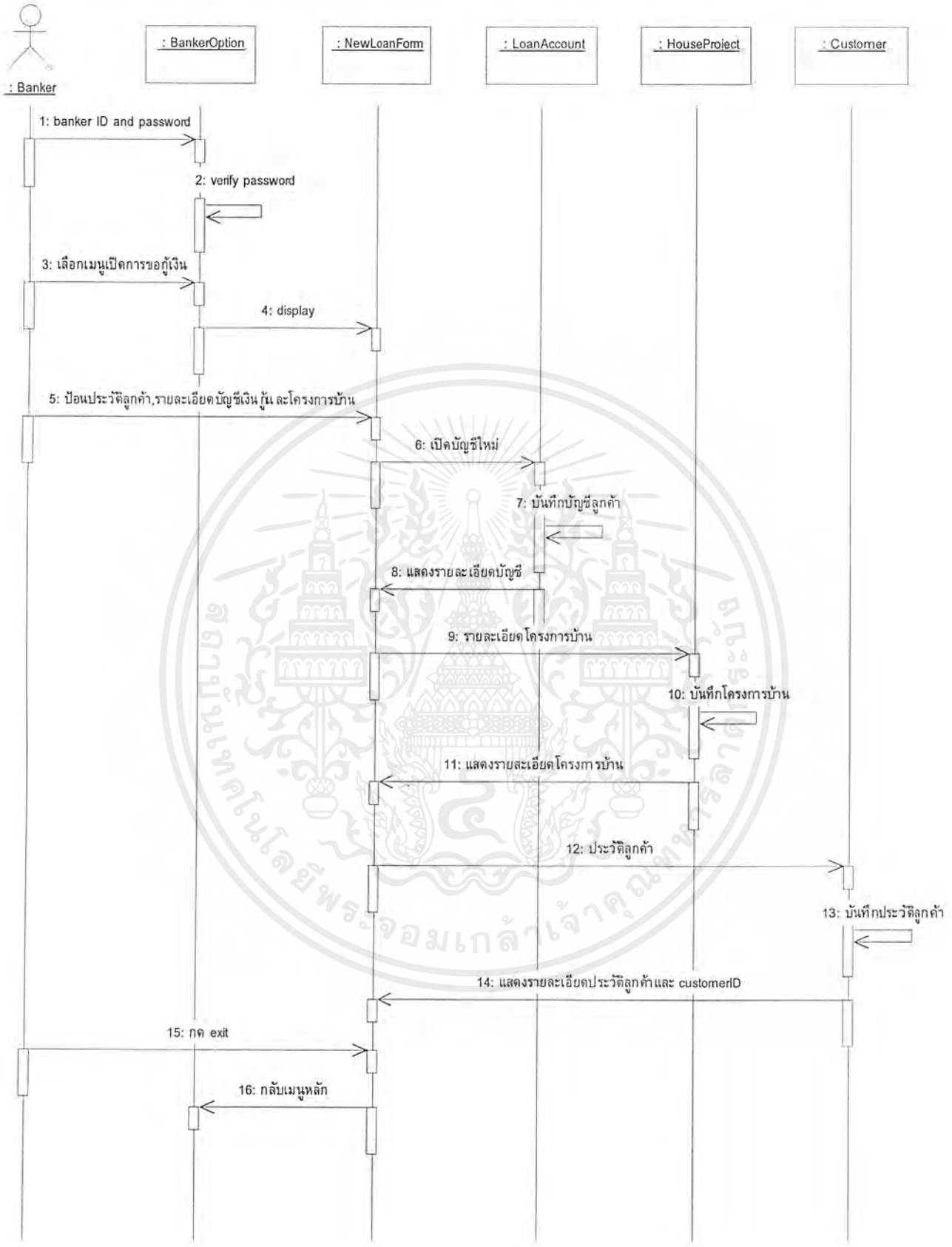
รูปที่ 6-14 Sequence Diagram ประเมินวงเงินกู้และเงินชำระต่องวด/แสดงตัวอย่าง

6.7.1.2 Sequence Diagram : ประเมินวงเงินกู้และเงินชำระต่องวด/เปิดการขอกู้ใหม่



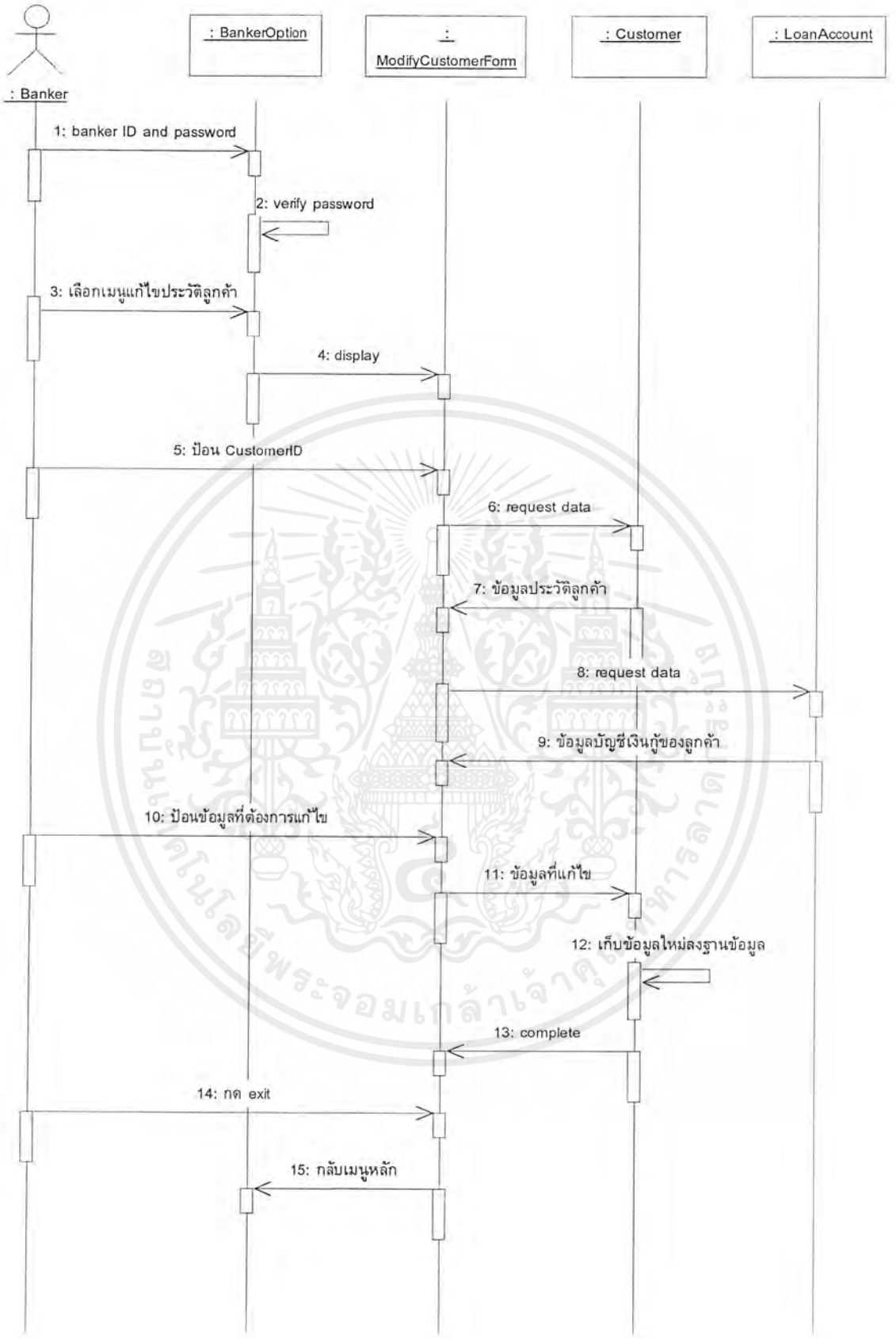
รูปที่ 6-15 Sequence Diagram ประเมินวงเงินกู้และเงินชำระต่องวด/เปิดการขอกู้ใหม่

6.7.1.3 Sequence Diagram : เปิดการขอกู้ใหม่



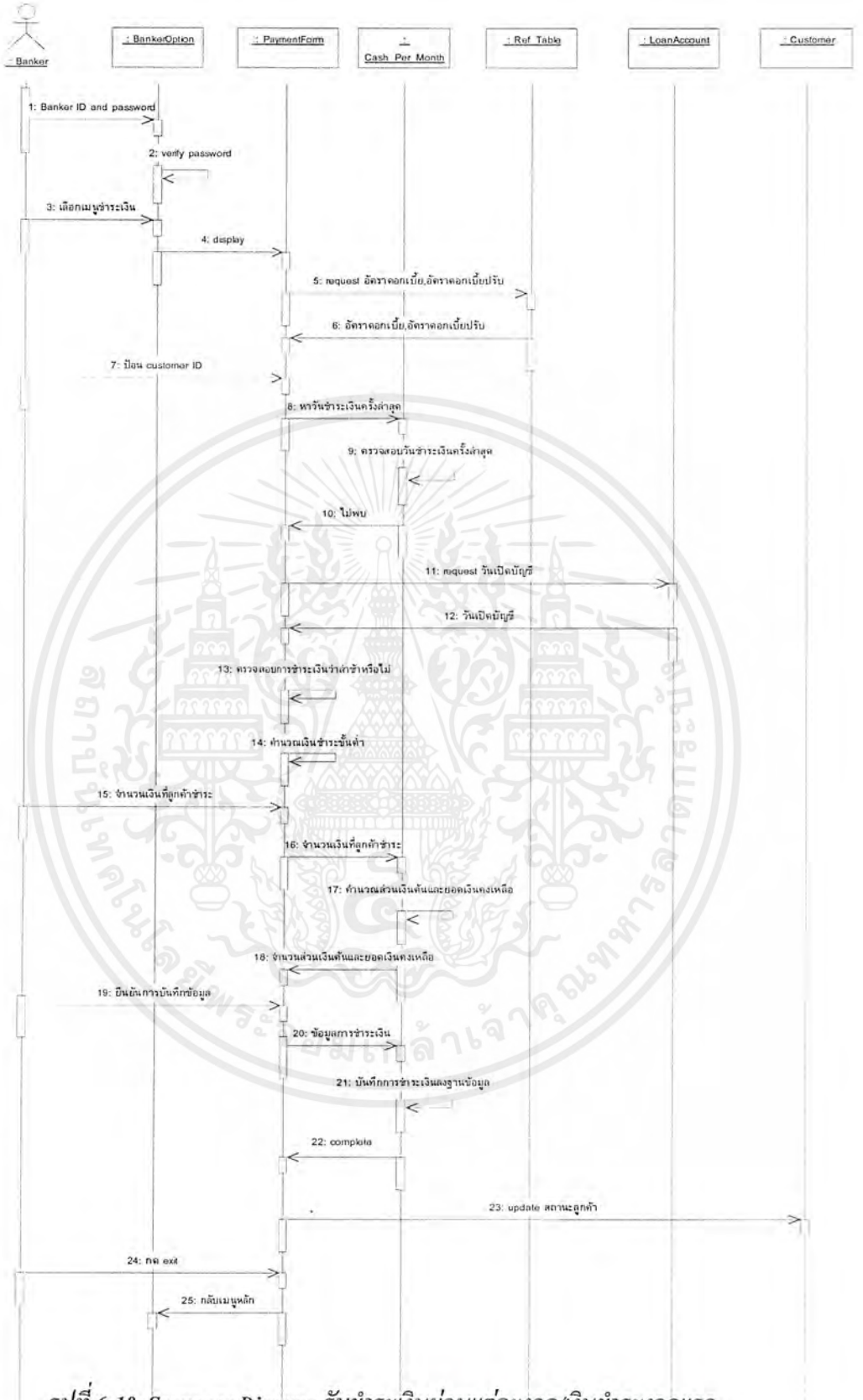
รูปที่ 6-16 Sequence Diagram เปิดการขอกู้ใหม่

6.7.1.4 Sequence Diagram : แก้ไขประวัติลูกค้า



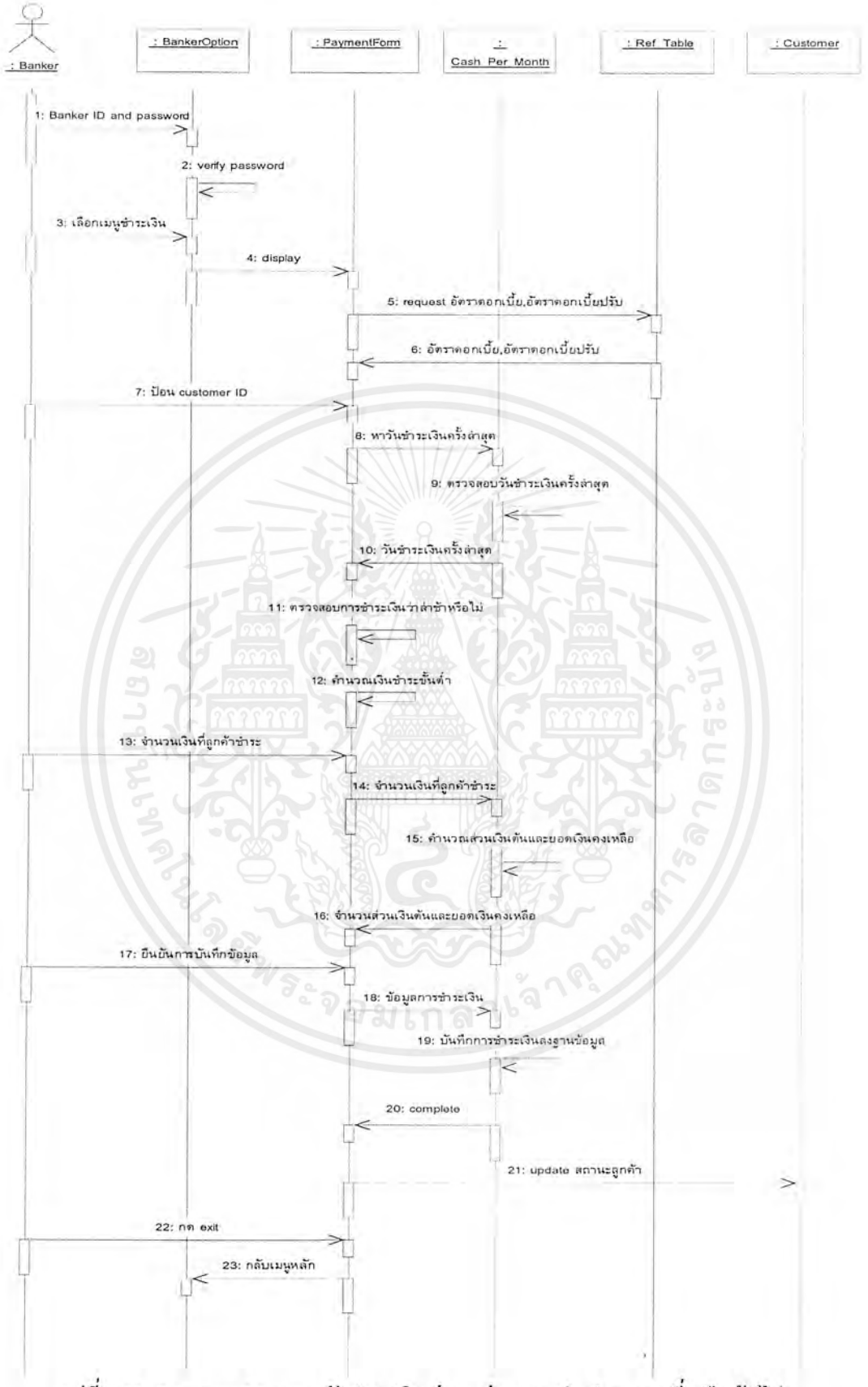
รูปที่ 6-17 Sequence Diagram แก้ไขประวัติลูกค้า

6.7.1.5 Sequence Diagram : รับชำระเงินผ่อนแต่ละงวด/เงินชำระงวดแรก



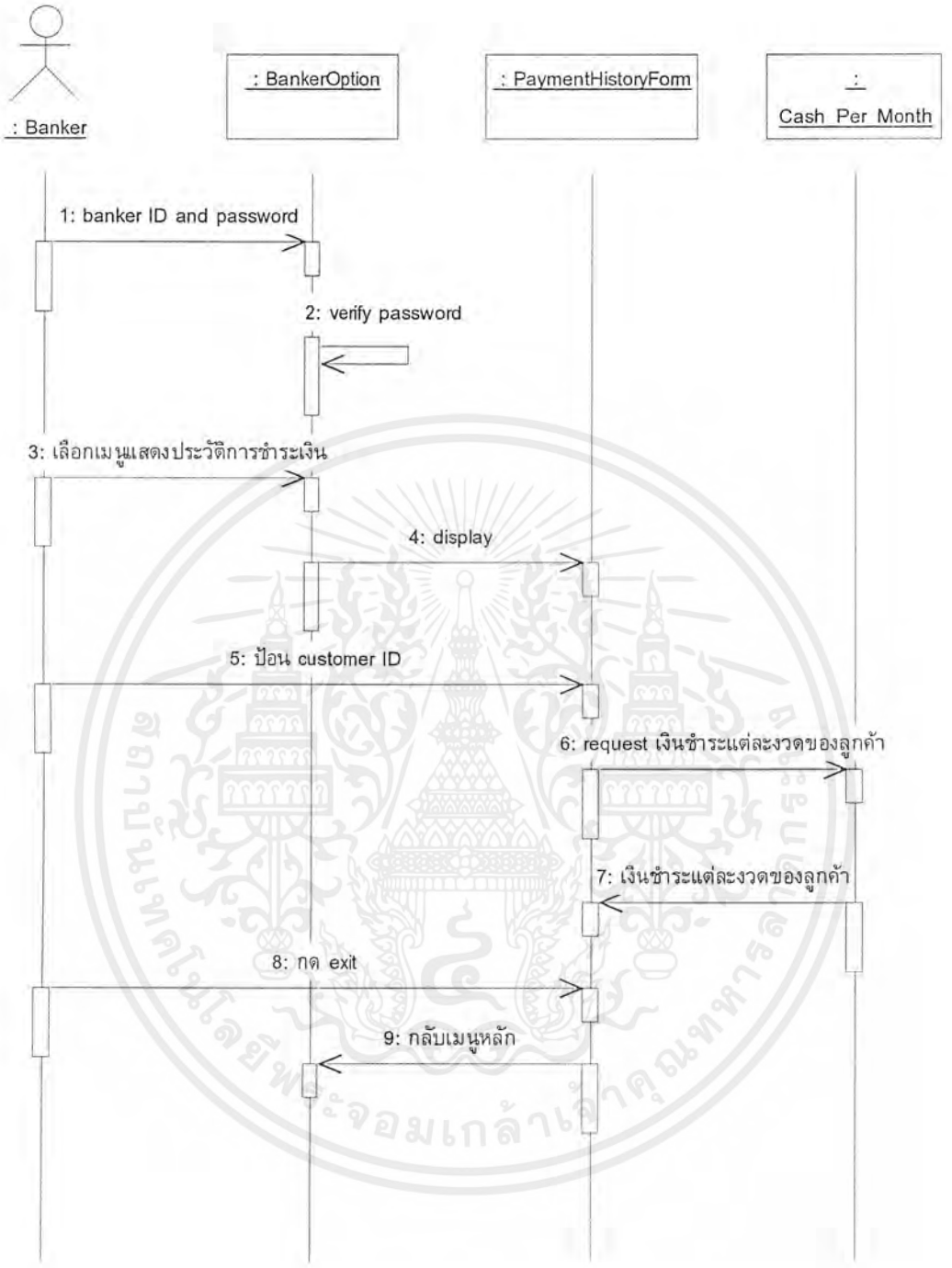
รูปที่ 6-18 Sequence Diagram รับชำระเงินผ่อนแต่ละงวด/เงินชำระงวดแรก

6.7.1.6 Sequence Diagram : รับชำระเงินผ่อนแต่ละงวด/เงินชำระงวดที่สองเป็นต้นไป



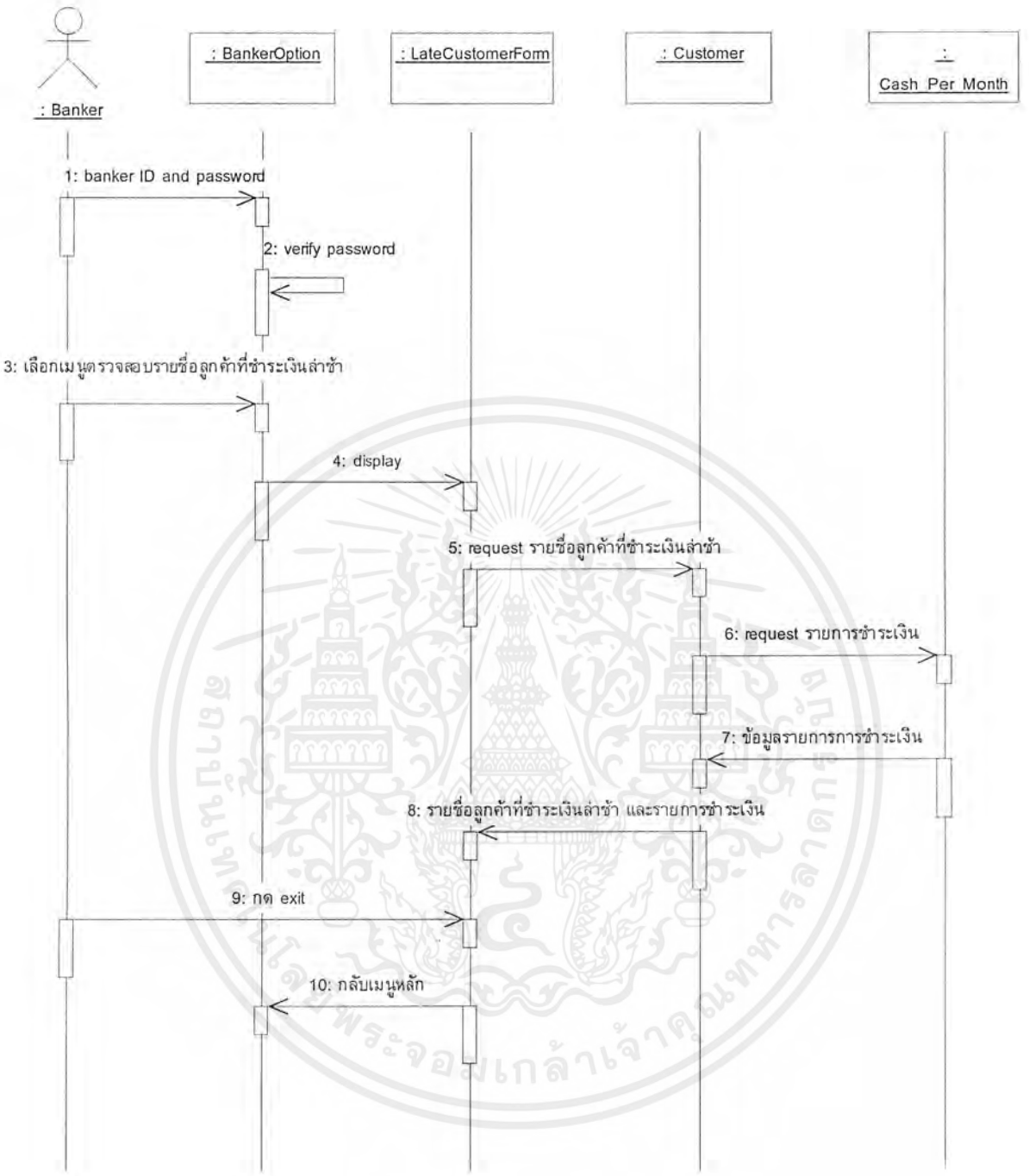
รูปที่ 6-19 Sequence Diagram รับชำระเงินผ่อนแต่ละงวด/เงินชำระงวดที่2เป็นต้นไป

6.7.1.7 Sequence Diagram : แสดงประวัติการชำระเงินของลูกค้า



รูปที่ 6-20 Sequence Diagram แสดงประวัติการชำระเงินของลูกค้า

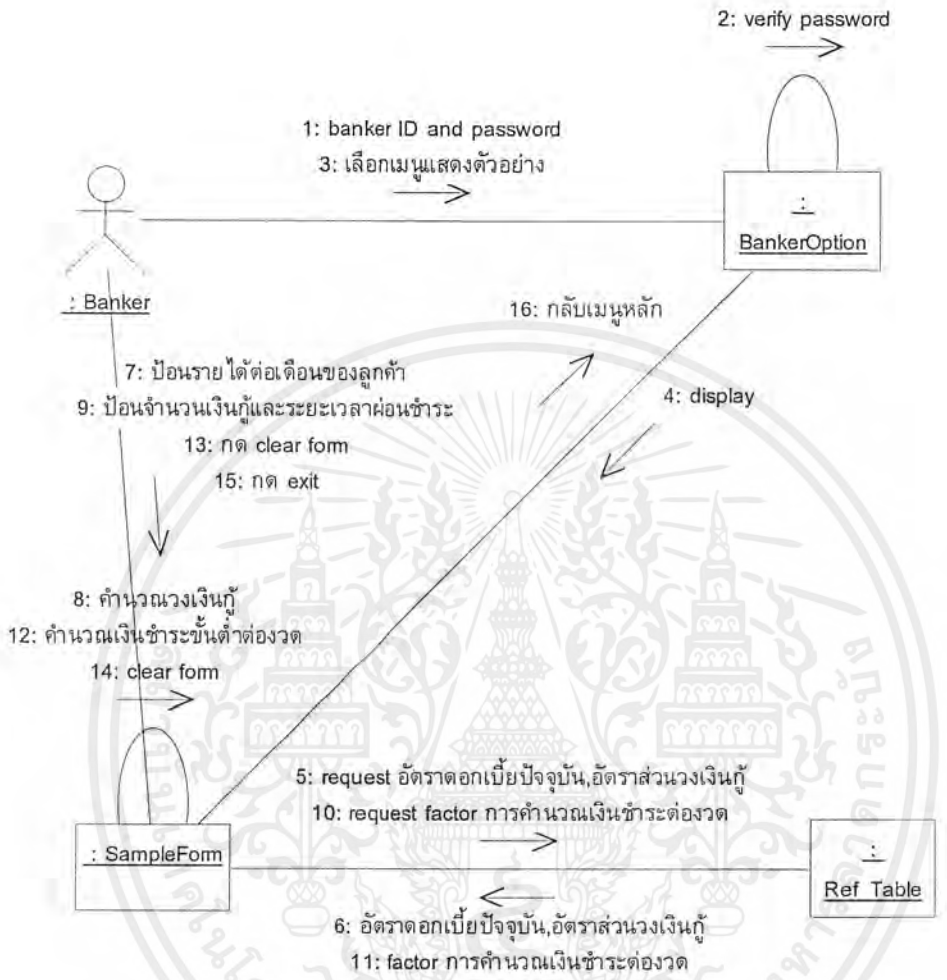
6.7.1.8 Sequence Diagram : ตรวจสอบรายชื่อลูกค้าที่ชำระเงินล่าช้า



รูปที่ 6-21 Sequence Diagram ตรวจสอบรายชื่อลูกค้าที่ชำระเงินล่าช้า

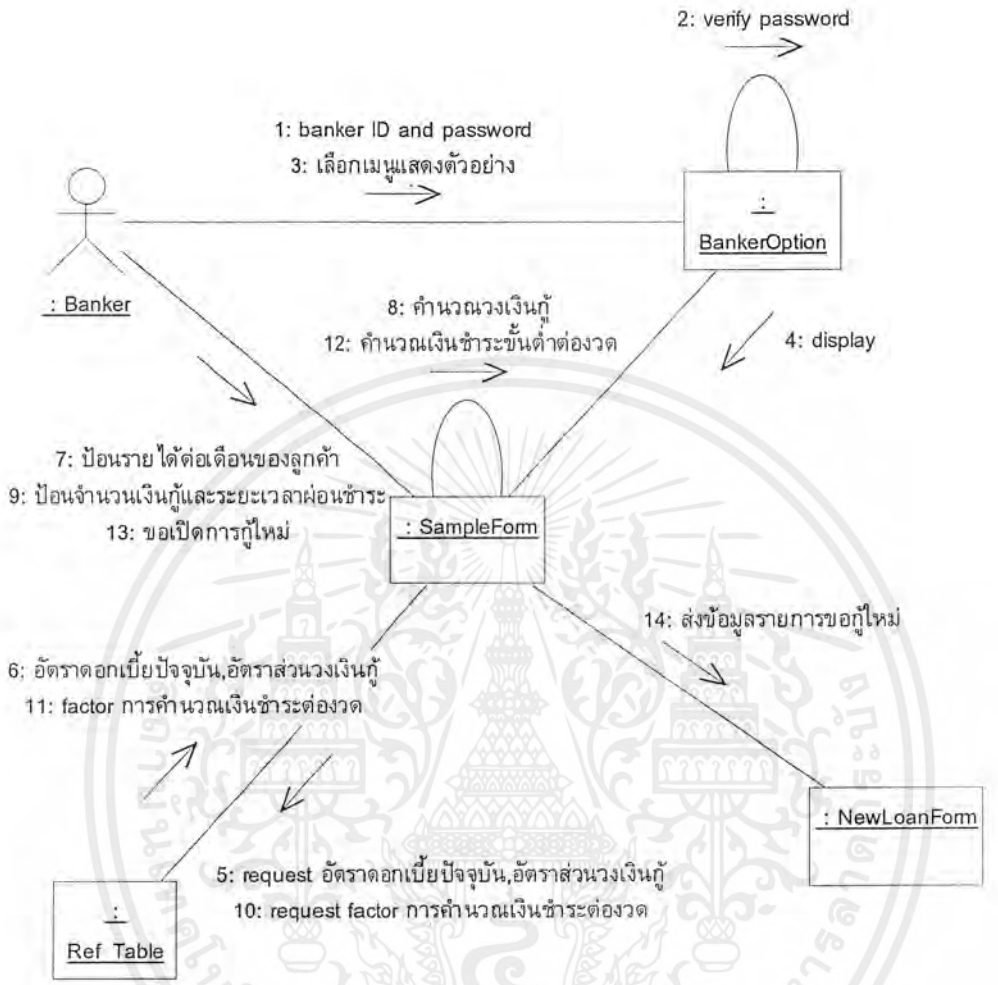
6.7.2 Collaboration Diagram

6.7.2.1 Collaboration Diagram : ประเมินวงเงินกู้และเงินชำระต่องวด / แสดงตัวอย่าง



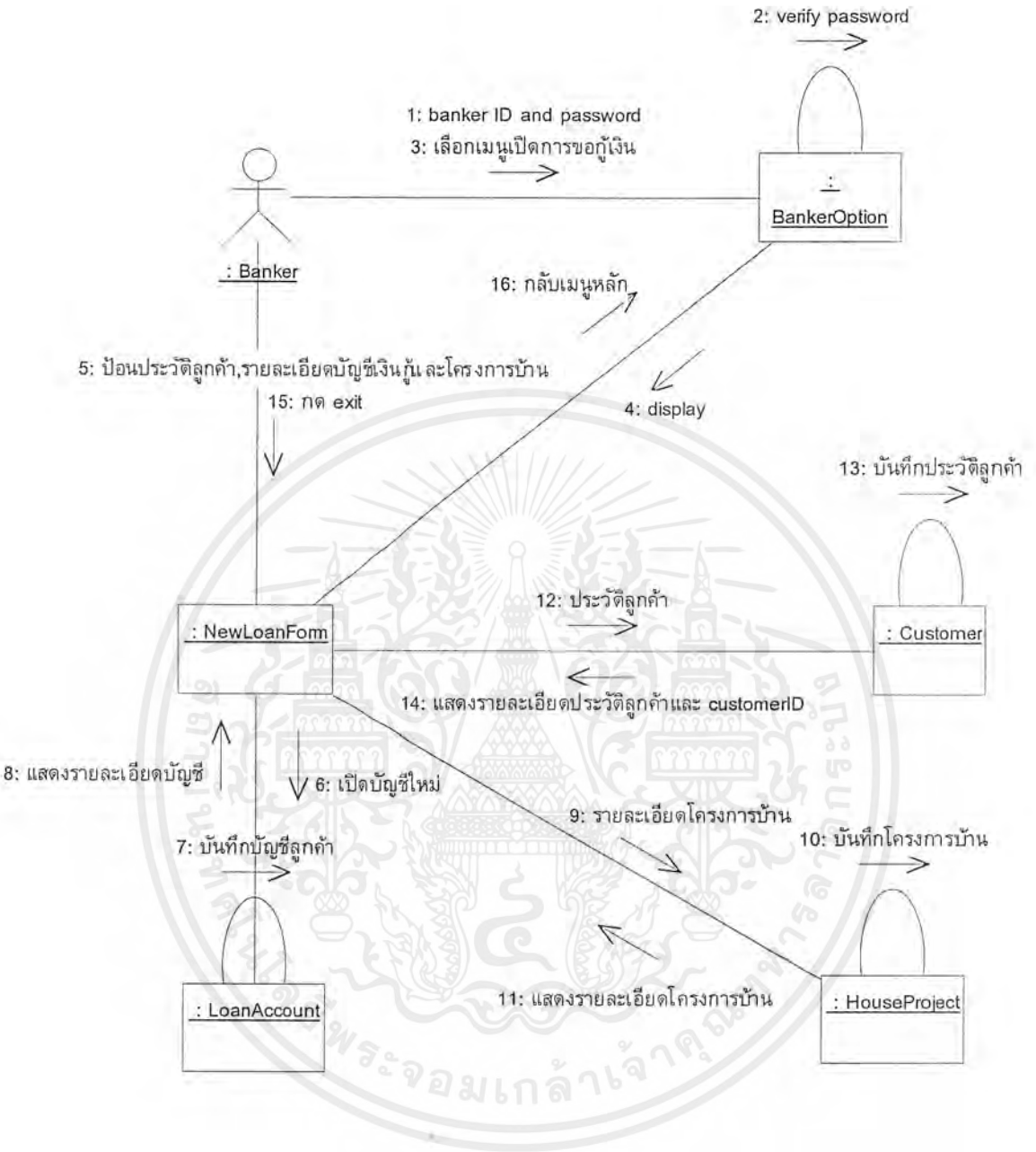
รูปที่ 6-22 Collaboration Diagram ประเมินวงเงินกู้และเงินชำระต่องวด/แสดงตัวอย่าง

6.7.2.2 Collaboration Diagram : ประเมินวงเงินกู้และเงินชำระต่องวด /เปิดการกู้ใหม่



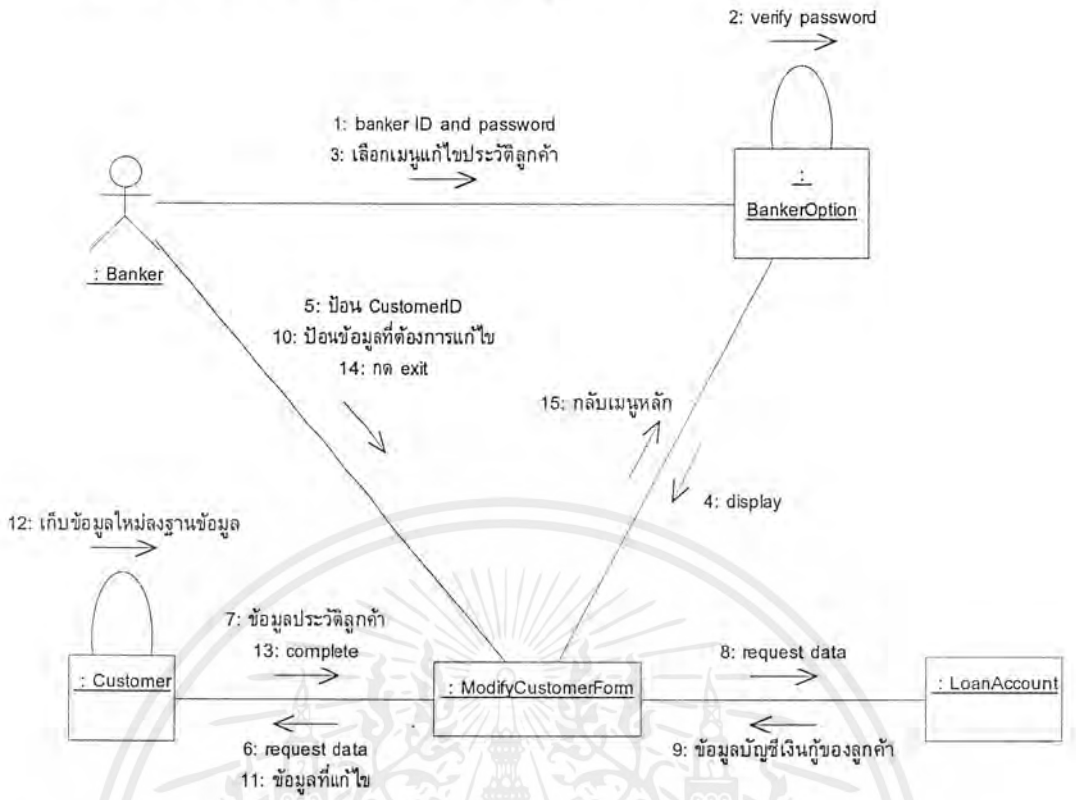
รูปที่ 6-23 Collaboration Diagram ประเมินวงเงินกู้และเงินชำระต่องวด/เปิดการขอกู้ใหม่

6.7.2.3 Collaboration Diagram : เปิดการขอกู้ใหม่



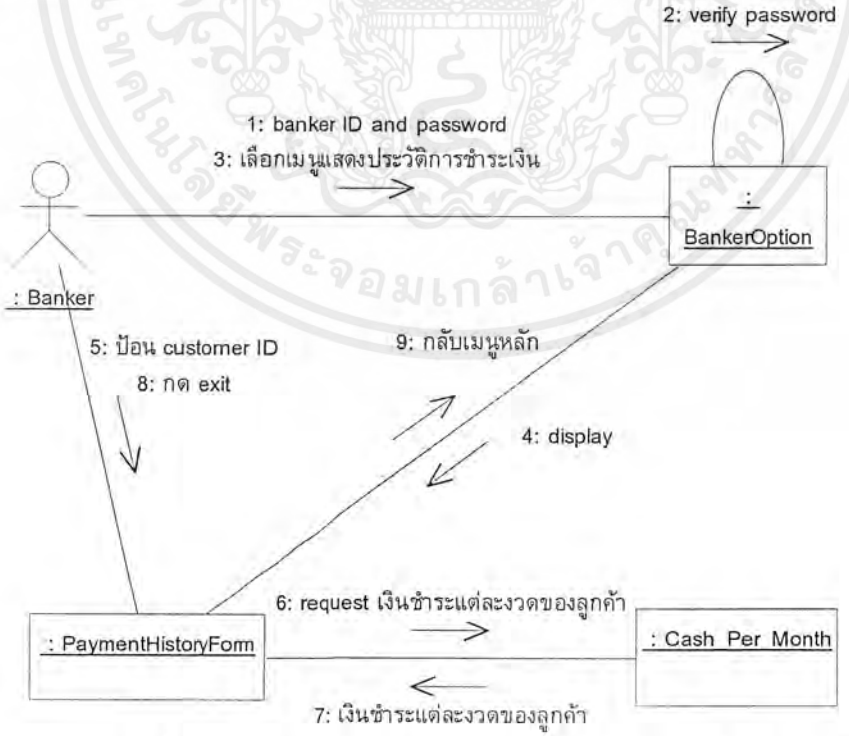
รูปที่ 6-24 Collaboration Diagram เปิดการขอกู้ใหม่

6.7.2.4 Collaboration Diagram : แก้ไขประวัติลูกค้า



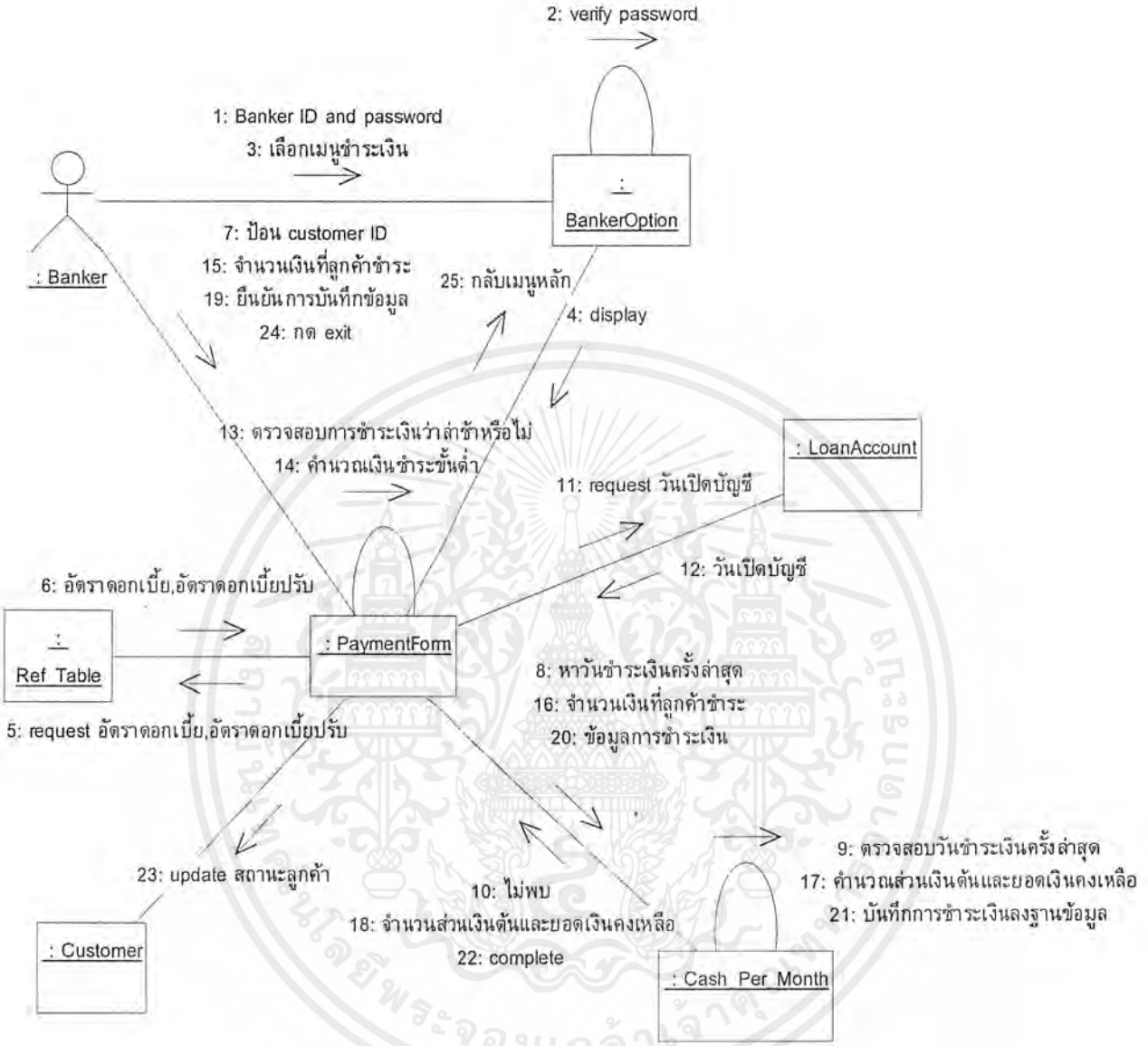
รูปที่ 6-25 Collaboration Diagram แก้ไขประวัติลูกค้า

6.7.2.5 Collaboration Diagram : แสดงประวัติการชำระเงินของลูกค้า



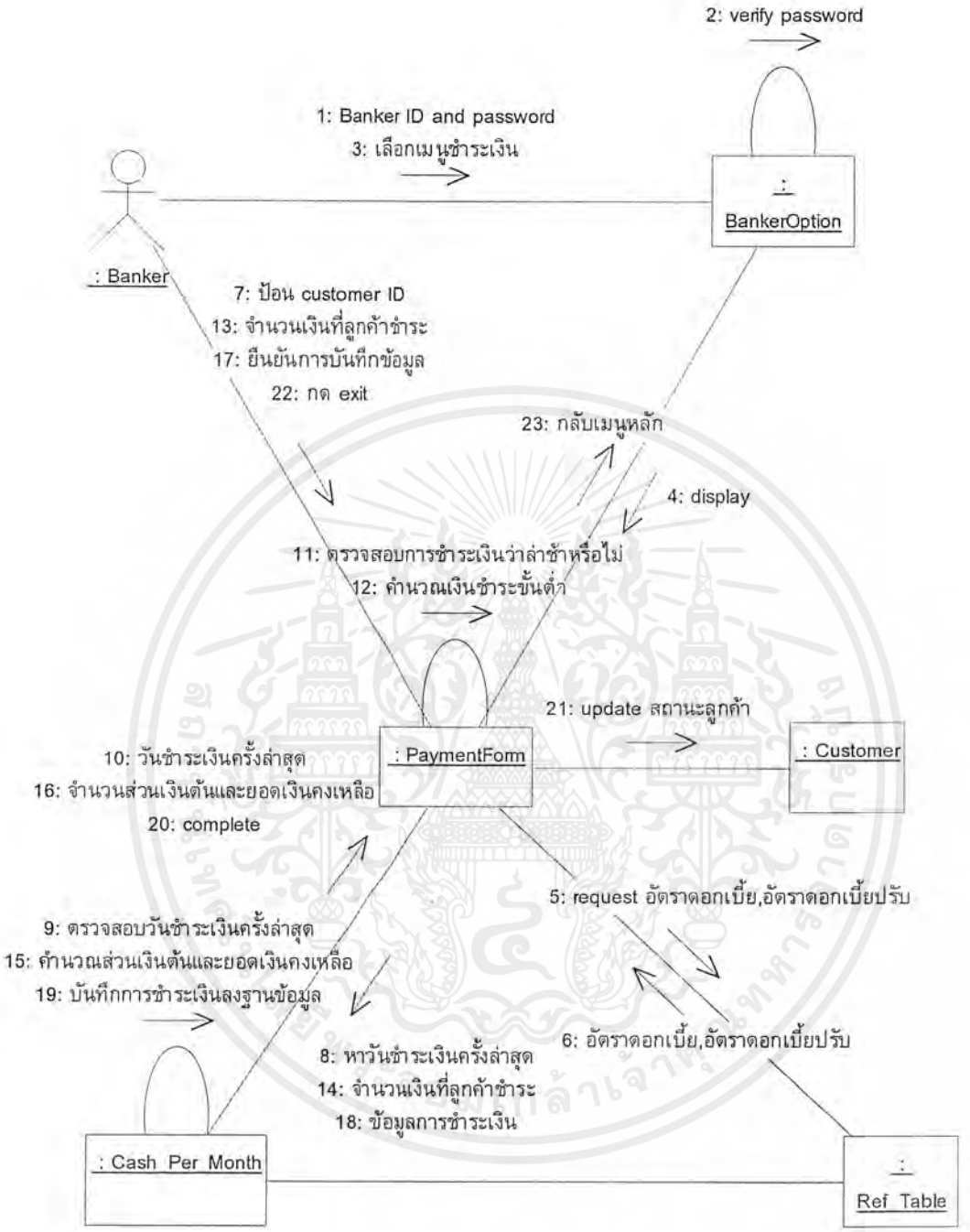
รูปที่ 6-26 Collaboration Diagram แสดงประวัติการชำระเงินของลูกค้า

6.7.2.6 Collaboration Diagram : รับชำระเงินผ่อนแต่ละงวด/เงินผ่อนชำระงวดแรก



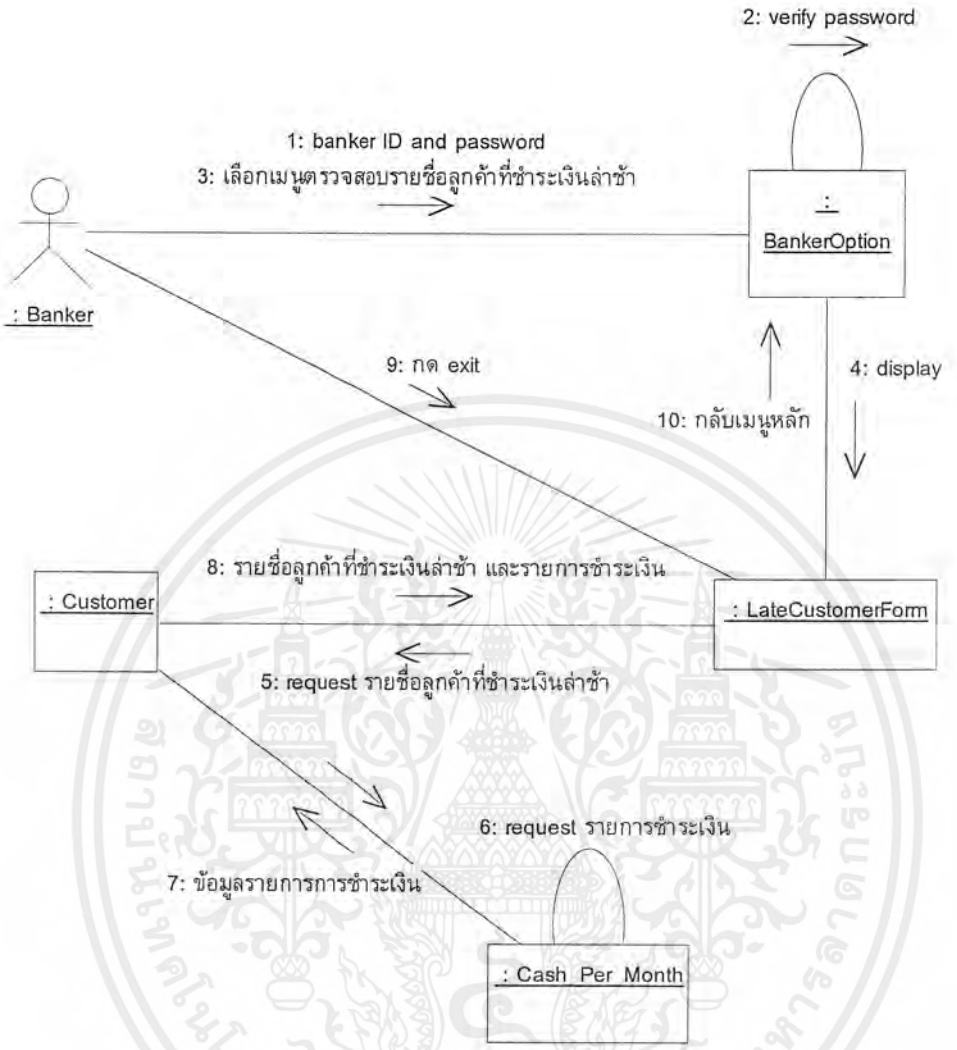
รูปที่ 6-27 Collaboration Diagram รับชำระเงินผ่อนแต่ละงวด/เงินผ่อนชำระงวดแรก

6.7.2.7 Collaboration Diagram : รับชำระเงินผ่อนแต่ละงวด/เงินผ่อนชำระงวดที่2เป็นต้นไป



รูปที่ 6-28 Collaboration Diagram รับชำระเงินผ่อนแต่ละงวด/เงินผ่อนชำระงวดที่2เป็นต้นไป

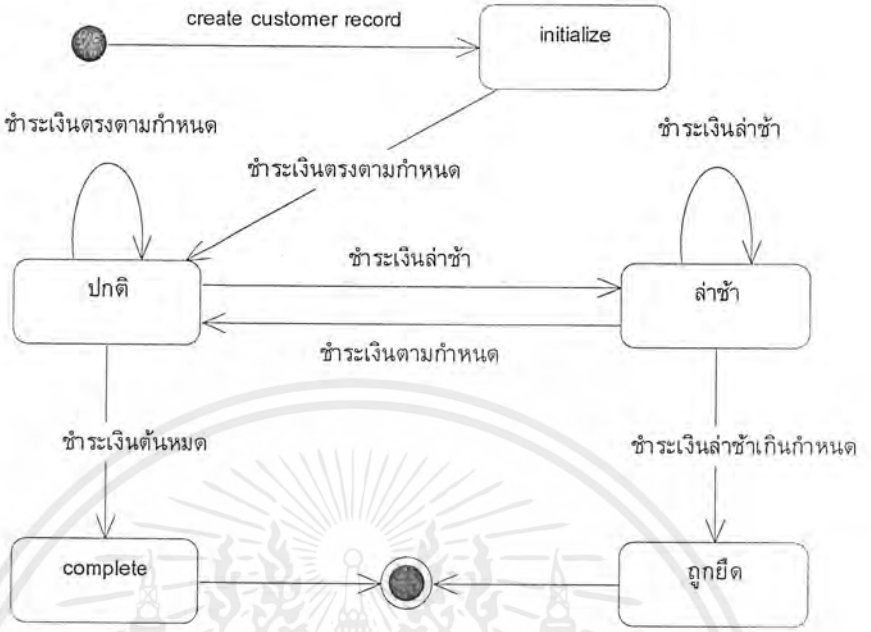
6.7.2.8 Collaboration Diagram : แสดงรายชื่อลูกค้าที่ชำระเงินล่าช้า



รูปที่ 6-29 Collaboration Diagram แสดงรายชื่อลูกค้าที่ชำระเงินล่าช้า

6.7.3 State Diagram

State Diagram ของคลาส Customer แสดงดังต่อไปนี้



รูปที่ 6-30 State Diagram ของคลาส Customer

6.8 การสร้างตารางฐานข้อมูลจาก Class Diagram

จาก Class Diagram ที่ได้ นำออบเจ็กต์คงอยู่และความสัมพันธ์ระหว่างออบเจ็กต์มาสร้างเป็นตารางตามวิธีที่ได้กล่าวไว้ในบทที่ 5

ตาราง CUSTOMER

คอลัมน์	ชนิดข้อมูล	ข้อบังคับ
Customer_ID	Number(7)	Primary Key
Name	Varchar2(30)	Not Null
Address	Varchar2(60)	Not Null
Salary	Number(10)	Not Null
Payment_Status	Varchar2(8)	Check [acquire , normal , late]
Account_ID	Number(11)	Foreign Key อ้างอิง LOANACCOUNT
Project_ID	Number(7)	Foreign Key อ้างอิง HOUSEPROJECT

ตารางที่ 6-2 ตารางฐานข้อมูลของคลาส Customer

ตาราง LOANACCOUNT

คอลัมน์	ชนิดข้อมูล	ข้อบังคับ
Account_ID	Number(11)	Primary Key
Amount	Number(8)	Check [100,000 – 10,000,000]
PType	Number(2)	Check [5,10,15,20]
Loan_Int_Rate	Number(2,1)	Check [7,7.5,8,8.5,9]
Const_Paid	Number(6)	Not Null
Start_Date	Date	Not Null
Branch_ID	Number(5)	Foreign Key อ้างอิง BANK
Banker_ID	Number(7)	Foreign Key อ้างอิง BANKER

ตารางที่ 6-3 ตารางฐานข้อมูลคลาส LoanAccount

ตาราง HOUSEPROJECT

คอลัมน์	ชนิดข้อมูล	ข้อบังคับ
Project_ID	Number(7)	Primary Key
ProjectName	Varchar2(20)	Not Null
Province	Varchar2(20)	Not Null
Area	Number(4)	Not Null , Check [<= 5,000]
Price	Number(8)	Not Null , Check [<= 30,000,000]
Land_ID	Number(10)	Unique , Not Null
Owner	Varchar2(30)	Not Null

ตารางที่ 6-4 ตารางฐานข้อมูลคลาส HouseProject

ตาราง BANK

คอลัมน์	ชนิดข้อมูล	ข้อบังคับ
Branch_ID	Number(5)	Primary Key
Branch_Name	Varchar2(20)	Unique , Not Null

ตารางที่ 6-5 ตารางฐานข้อมูลคลาส Bank

ตาราง BANKER

คอลัมน์	ชนิดข้อมูล	ข้อบังคับ
Banker_ID	Number(7)	Primary Key
Banker_Name	Varchar2(30)	Not Null
Password	Varchar2(8)	Not Null
B_Status	Varchar2(5)	Check [user , admin]
Branch_ID	Number(5)	Foreign Key อ้างอิง BANK

ตารางที่ 6-6 ตารางฐานข้อมูลคลาส Banker

ตาราง CASH_PER_MONTH

คอลัมน์	ชนิดข้อมูล	ข้อบังคับ
Cash_ID	Number(12)	Primary Key
Period_Num	Number(3)	Check [1-240]
Cash_Money	Number(8)	Check [<= 10,000,000]
Principal_Paid	Number(8)	Check [>= 0]
Int_Paid	Number(8)	Check [>= 0]
Late_Int_Paid	Number(8)	Check [>= 0]
Balance_Due	Number(8)	Check [>= 0]
Period_Status	Varchar2(6)	Check [late , normal]
Pay_Date	Date	Not Null
Customer_ID	Number(7)	Foreign Key อ้างอิง CUSTOMER
Banker_ID	Number(7)	Foreign Key อ้างอิง BANKER

ตารางที่ 6-7 ตารางฐานข้อมูลคลาส CashPerMonth

สำหรับคลาส RefTable ต้องแยกเป็น 2 ตารางเพื่อให้ได้ 3 Normal Form

ตาราง REF_TABLE

คอลัมน์	ชนิดข้อมูล	ข้อบังคับ
Ref_ID	Number(3)	Primary Key
Current_Int_Rate	Number(2,1)	Check [7,7.5,8,8.5,9]
Current_Late_Int	Number(3,1)	Check [<= 20]
Max_Loan	Number(2)	Check [20 –30]
Update_Date	Date	Not Null

ตาราง FACTOR

คอลัมน์	ชนิดข้อมูล	ข้อบังคับ
Int_Rate	Number(2,1)	Primary Key (concatenated key)
Period_Type	Number(2)	Primary Key (concatenated key)
Loan_Factor	Number(8,7)	Not Null

ตารางที่ 6-8 ตารางฐานข้อมูลคลาส RefTable

บทที่ 7
การสร้างฐานข้อมูลด้วยออราเคิล (Oracle)

7.1 บทแนะนำ

ตารางข้อมูลนำเสนออยู่ในรูปของแถวและคอลัมน์ แต่จริงๆแล้วตารางเก็บอยู่ในไฟล์ตามระบบปฏิบัติการซึ่งแตกต่างจากรูปแบบคอลัมน์และแถวมาก

SQL (Structured Query Language) เป็นภาษาที่ใช้ในการเข้าถึงข้อมูลสำหรับฐานข้อมูลแบบรีเลชันแนล ได้รับการรับรองจากสถาบันมาตรฐานแห่งสหรัฐอเมริกา (The American National Standards Institute) และกำหนดเป็นมาตรฐาน ANSI X3.135-1989 ออราเคิลสร้าง SQL ซึ่งเข้ากันได้ 100% กับมาตรฐาน ANSI และได้เพิ่มเติมความสามารถเฉพาะเข้าไปทำให้ภาษา SQL ของออราเคิลสามารถฝังตัวอยู่ในโปรแกรมภาษายุคที่สาม (Third – generation language) , สามารถเอ็กซีคิวได้จากฐานข้อมูลระยะไกล , ฝังอยู่ในเซิฟเวอร์เก็บโพรซีเยอร์ (procedure) และทริกเกอร์ (trigger) และทำงานแบบปฏิสัมพันธ์โดยใช้ SQL*PLUS

SQL*PLUS เป็นส่วนรับคำสั่งอินเตอร์เฟซกับฐานข้อมูล สนับสนุนการทำงานแบบปฏิสัมพันธ์กับคำสั่ง insert , delete และ update ,การเรียกดูข้อมูล,การเอ็กซีคิวโพรซีเยอร์ เป็นต้น ใช้ในการกำหนดข้อมูล,จัดการข้อมูลและแก้ไขสถานะและแอททริบิวต์บนตารางข้อมูลและออบเจ็กต์อื่นๆ

PL/SQL เป็นภาษายุคที่สี่ (Fourth – generation language) ที่ออราเคิลพัฒนาขึ้นเพิ่มเติมจากภาษา SQL มาตรฐาน มี procedural control construct ซึ่งสามารถรวมกับ SQL ตามมาตรฐาน ANSI ได้ รวมไปถึงการสร้างโพรซีเยอร์,แพ็คเกจและทริกเกอร์

7.1.1 ชนิดข้อมูล (Data Type)

- CHAR ใช้สำหรับอักษรที่ความยาวคงที่ มีค่าพารามิเตอร์กำหนดความยาวอักษรไว้ หากไม่ได้กำหนดค่าพารามิเตอร์ไว้ จะใช้ค่าเบื้องต้นคือ 1 ตัวอักษร อย่างไรก็ตามความยาวอักษรสูงสุดในคอลัมน์ชนิด Char นี้คือ 255 ไบต์ หากอักษรมีความยาวมากกว่านี้ต้องใช้ข้อมูลชนิด Long ซึ่งจะอธิบายต่อไป
- VARCHAR2 ใช้สำหรับอักษรที่มีความยาวไม่คงที่ มีค่าพารามิเตอร์กำหนดขอบเขตความยาวอักษรสูงสุด อย่างไรก็ตามความกว้างสูงสุดของคอลัมน์ฐานข้อมูลคือ 2000 ไบต์ หากต้องการ insert อักษรที่มีความยาวมากกว่านี้ต้องใช้ข้อมูลชนิด Long
- LONG สำหรับข้อมูลอักษร มีความยาวได้สูงสุดถึง 2 กิกะไบต์ คอลัมน์ที่มีชนิดข้อมูล Long สามารถเก็บอักษร , อาเรย์ของตัวอักษร หรือแม้แต่เอกสารสั้นๆได้
- NUMBER สำหรับเลขที่กำหนดขนาดหรือเลขทศนิยม มีค่าพารามิเตอร์ (a,b) โดย a คือจำนวนหลักทั้งหมด b คือจำนวนหลักตามหลังจุดทศนิยม เรียกว่า พริซีชัน (precision) ดังนั้นจำนวนหลักที่อยู่หน้าจุดทศนิยมคือ a - b ตัวอย่างเช่น หากต้องการเก็บข้อมูลที่มี

ลักษณะ เป็นเลขจำนวนเต็ม 2 หลักและมีทศนิยม 1 หลัก เช่น 15.5 ต้องกำหนด a = 3 และ b = 1

- RAW เก็บข้อมูลเลขฐานสองได้ถึง 2000 ไบต์
- LONG RAW เก็บข้อมูลเลขฐานสองได้ยาวสูงสุดถึง 2 กิกะไบต์
- DATE เก็บวัน (วัน – เดือน – ปี) และเวลา (ชั่วโมง – นาที – วินาที) โดยเริ่มนับจากเที่ยงคืน รูปแบบเบื้องต้นของข้อมูลชนิด Date ที่แสดงคือ DD – MON – YY
- BINARY_INTEGER สำหรับเลขจำนวนเต็มมีเครื่องหมาย โดยมีช่วงค่าอยู่ระหว่าง -2^{31} –1 ถึง 2^{31} –1 (-2147483647 ถึง 2147483647)
- ROWID ภายในแล้ว ทุกๆตารางจะมีคอลัมน์ที่เป็น ROWID ซึ่งเก็บค่าเลขฐานสองขนาด 6 ไบต์เรียกว่า rowid โดยจะมีค่าไม่ซ้ำกัน ใช้ระบุแถวและช่วยให้การเข้าถึงแถวทำได้รวดเร็ว ชนิดข้อมูล ROWID ใช้เก็บ rowid ในรูปแบบที่สามารถอ่านได้
- BOOLEAN เก็บค่า True , False และ Null (ในกรณีที่ค่านั้นสูญหาย , ไม่ทราบแน่ชัด หรือไม่สามารถนำไปใช้ได้)
- MLSLABEL เก็บลาเบล (Label) เลขฐานสอง,ความยาวไม่คงที่ของระบบปฏิบัติการ โดย Trusted Oracle ใช้ลาเบลเพื่อควบคุมการเข้าถึงข้อมูล
- BLOB
- CLOB
- NCLOB
- BFILE

7.1.2 ข้อบังคับ (Constraints)

- Primary Key ตารางส่วนใหญ่จะมีข้อบังคับ primary key 1 คอลัมน์ และอาจจะสร้างมาจากมากกว่า 1 คอลัมน์ เป็น concatenated key คอลัมน์ที่เป็น primary key จะบังคับใช้กฎดังนี้
  - ค่าของคอลัมน์ต้องเป็นเอกลักษณ์ คือไม่ซ้ำกัน
  - ค่าของคอลัมน์เป็น Null ไม่ได้

  กฎนี้จะถูกละเมิดเมื่อมีการ insert , update ค่าที่ซ้ำกันหรือ Null ลงในคอลัมน์ สามารถตั้งชื่อข้อบังคับได้เพื่อความสะดวกเมื่อเกิดข้อผิดพลาดหรือข้อบังคับถูกละเมิด โดยมีความยาวได้ไม่เกิน 30 ตัวอักษร หากไม่ตั้งชื่อ ออราเคิลจะสร้างให้เอง ขึ้นต้นด้วย SYS_C ตามด้วยเลข 6 หลักไม่ซ้ำกัน
- Unique บังคับใช้กฎเช่นเดียวกับ primary key ยกเว้นสามารถมีค่าคอลัมน์เป็น null ได้
- Not Null เป็นข้อบังคับป้องกันไม่ให้ insert หรือ update คอลัมน์ด้วย Null

- Check กำหนดช่วงค่าที่อนุญาติในคอลัมน์ โดยสามารถกำหนดในข้อบังคับหรือกำหนดเป็นสมการทางคณิตศาสตร์ แต่มีข้อควรระวังในเรื่องตัวอักษรใหญ่และเล็กนั้นแตกต่างกัน เช่น ในข้อบังคับกำหนดค่าคอลัมน์เป็น 'LONDON' หรือ 'paris' หากทำการ insert ('London', 'Paris') จะละเมิดข้อบังคับทันที Check สามารถใช้ร่วมกับ Not Null , Unique และ Primary Key และยังสามารถสร้างจากหลายคอลัมน์ในตารางได้
- Foreign Key สร้างการอ้างอิงค่าของคอลัมน์ไปยังค่าปัจจุบันที่ปรากฏอยู่ในคอลัมน์อื่น ซึ่งอาจเป็นคอลัมน์ในตารางเดียวกันหรือต่างตารางกันก็ได้ โดยคอลัมน์ที่อ้างอิงถึงนั้นจะต้องอยู่ภายใต้ข้อบังคับ Primary Key หรือ Unique และหากคอลัมน์ที่อ้างถึงนั้นกำหนด Delete Cascade ไว้ เมื่อลบแถวออก แถวของตารางที่อ้างอิงแถวที่ลบไปนั้นก็จะถูกลบไปด้วย

7.1.3 สโตร์โพรซีเยอร์ (Stored Procedures)

สโตร์โพรซีเยอร์(Stored Procedure) หมายถึงหน่วยโปรแกรมที่ถูกเรียกใช้ได้ซึ่งประกาศเป็นออบเจ็กต์อยู่ในฐานข้อมูล ออราเคิลสามารถสร้างสตอร์โพรซีเยอร์ได้ 3 ชนิดคือโพรซีเยอร์ (procedure) , ฟังก์ชัน (function) และแพ็คเกจ (package) โพรซีเยอร์และฟังก์ชันเรียกได้อีกอย่างหนึ่งว่าโปรแกรมย่อย (subprogram) ไม่ว่าจะอยู่เดี่ยวๆหรือรวมอยู่ในแพ็คเกจ ส่วนแพ็คเกจนั้นคือหน่วยโปรแกรมซึ่งหุ้มข้อมูลที่คงอยู่และโปรแกรมย่อยไว้ สโตร์โพรซีเยอร์สามารถเอ็กซีคิวต์ได้จาก SQL*PLUS , ฝังอยู่ในโปรแกรมภาษายุคที่สาม , จาก GUI และจากฐานข้อมูลออราเคิลระยะไกล

7.1.3.1 โพรซีเยอร์

โพรซีเยอร์มีโครงสร้างดังนี้

```
CREATE OR REPLACE PROCEDURE procedure_name (ตัวแปรอาร์กิวเมนต์) IS
        {ส่วนประกาศตัวแปรโลคอล , ฟังก์ชันหรือโพรซีเยอร์ที่ใช้}
BEGIN
        {ส่วนโปรแกรมย่อย}
EXCEPTION
        {ส่วนโปรแกรมควบคุมเมื่อมีความผิดพลาดเกิดขึ้นขณะทำงาน}
END procedure_name;
```

7.1.3.2 ฟังก์ชัน

ฟังก์ชันมีลักษณะคล้ายกับโพรซีเยอร์ยกเว้นแต่ว่าฟังก์ชันจะต้องคืนค่ากลับ โดยใช้คำสั่ง RETURN หากไม่มีคำสั่งนี้จะเกิดข้อผิดพลาดในขณะคอมไพล์ อัลกอริธึมของฟังก์ชันจะต้องสิ้นสุดที่คำสั่ง RETURN เพื่อคืนค่ากลับออกมา หากในส่วนโปรแกรมย่อยมีการโยงไปยังส่วน EXCEPTION ส่วนโปรแกรมควบคุมจะต้องมีคำสั่ง RETURN อยู่ด้วย ไม่เช่นนั้นจะเกิดข้อผิดพลาดในขณะรันโปรแกรม

ฟังก์ชันมีโครงสร้างดังนี้

```
CREATE OR REPLACE FUNCTION function_name (ตัวแปรอาร์กิวเมนต์)
```

```
RETURN ชนิดข้อมูลที่จะคืนกลับ IS
        {ส่วนประกาศตัวแปรโลคอล , ฟังก์ชันหรือโพรซีเยอร์ที่ใช้}
BEGIN
        {ส่วนโปรแกรมย่อย}
EXCEPTION
        {ส่วนโปรแกรมควบคุมเมื่อมีความผิดพลาดเกิดขึ้นขณะทำงาน}
END function_name;
```

7.1.3.3 แพ็คเกจ

ประกอบด้วยส่วนกำหนดรายละเอียด (Specification) และส่วนโปรแกรม (Body) ในส่วนกำหนดรายละเอียดนั้นจะเป็นการประกาศชื่อฟังก์ชันและโพรซีเยอร์ต่างๆที่อยู่ในแพ็คเกจ ส่วนโปรแกรมจะเป็นคำสั่งภาษา PL/SQL สำหรับโพรซีเยอร์และฟังก์ชันที่ได้ประกาศไว้ในส่วนรายละเอียด และยังมีส่วนประกาศตัวแปรโลคอล,ค่าคงที่และโพรซีเยอร์หรือฟังก์ชันที่เรียกใช้ภายในแพ็คเกจด้วย

แพ็คเกจมีโครงสร้างดังนี้

```
CREATE OR REPLACE  PACKAGE package_name IS
        {ส่วนประกาศตัวแปร,ค่าคงที่,Type}
        PROCEDURE  P1 ;
        FUNCTION F1 RETRUN NUMBER;
END package_name;
CREATE OR REPLACE PACKAGE BODY  package_name IS
        {ส่วนประกาศตัวแปร,ค่าคงที่,ฟังก์ชันและโพรซีเยอร์ที่ใช้}
        PROCEDURE  P1  IS
        BEGIN
                {ส่วนโปรแกรมย่อย}
        END P1;
        FUNCTION  F1  RETURN NUMBER  IS
        BEGIN
                {ส่วนโปรแกรมย่อย}
        END F1;
END package_name;
```

7.2 การสร้างตารางด้วย Schema Manager

จาก Class Diagram ที่ได้จากการออกแบบ สามารถ map เป็นตารางฐานข้อมูลรีเลชันแนลโดยใช้หลักการดังที่กล่าวในบทที่ 5 นำตารางที่ได้มาสร้างโดยใช้ Schema Manager ซึ่งเป็นเครื่องมือตัวหนึ่งที่มี

อยู่ในออราเคิล ภายใน Schema Manger แบ่งออกเป็นโฟลเดอร์ตามออบเจ็กต์ต่างๆ เช่น Tables , Sequences , Functions , Procedures , Indexes เป็นต้น

7.2.1 การสร้างตาราง,กำหนดคอลัมน์และชนิดข้อมูล

คลิกเมาส์ขวาที่โฟลเดอร์ Tables เลือก Create... จะปรากฏฟอร์มดังแสดงในรูป 7-1

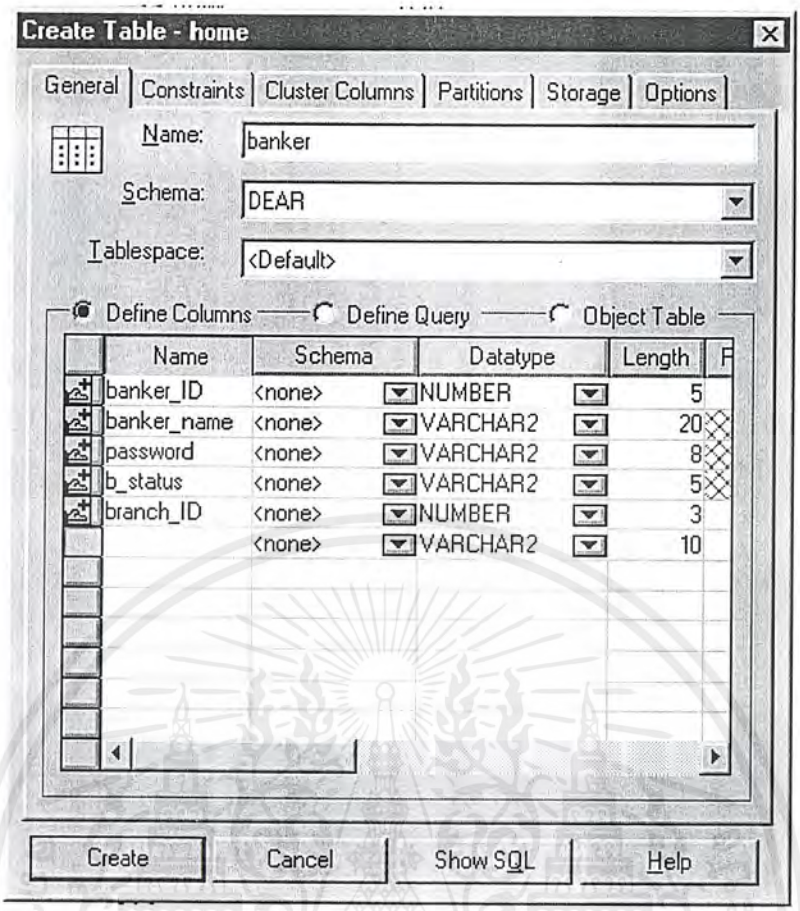
รูปที่ 7-1 ฟอร์มเลือกวิธีสร้างตารางใหม่

การสร้างตารางสามารถทำได้ 2 วิธี คือ ใช้ Table Wizard หรือสร้างตารางด้วยตัวเอง ข้อแตกต่างประการหนึ่งก็คือหากสร้างตารางด้วยตัวเองจะสามารถตั้งชื่อข้อบังคับเพื่อความสะดวกในการค้นหาว่าข้อบังคับใดของตารางใดถูกละเมิด ในที่นี้เลือกการสร้างตารางด้วยตนเอง จะปรากฏฟอร์มดังในรูป 7-2

ในส่วน General จะเป็นการตั้งชื่อตาราง การกำหนดคอลัมน์,ชนิดข้อมูลของคอลัมน์,ความยาวข้อมูล (ยกเว้นกรณีข้อมูลชนิด Date) ,ข้อบังคับ Not Null ,ค่าเริ่มต้น (Default Value) ของคอลัมน์ เป็นต้น

การกำหนดคอลัมน์ของตาราง (Define Column)

Name	:	ชื่อคอลัมน์ เช่น Banker_ID , Banker_Name เป็นต้น
Datatype	:	ชนิดข้อมูลของคอลัมน์
Length	:	ความยาวของข้อมูล
Precision	:	สำหรับข้อมูลชนิด Number กำหนดจำนวนหลักหลังจุดทศนิยม
Nulls?	:	เป็นการกำหนดข้อบังคับว่าจะอนุญาติให้คอลัมน์นั้นมีค่า Null หรือไม่
Default Value	:	กำหนดค่าเบื้องต้นให้แก่คอลัมน์

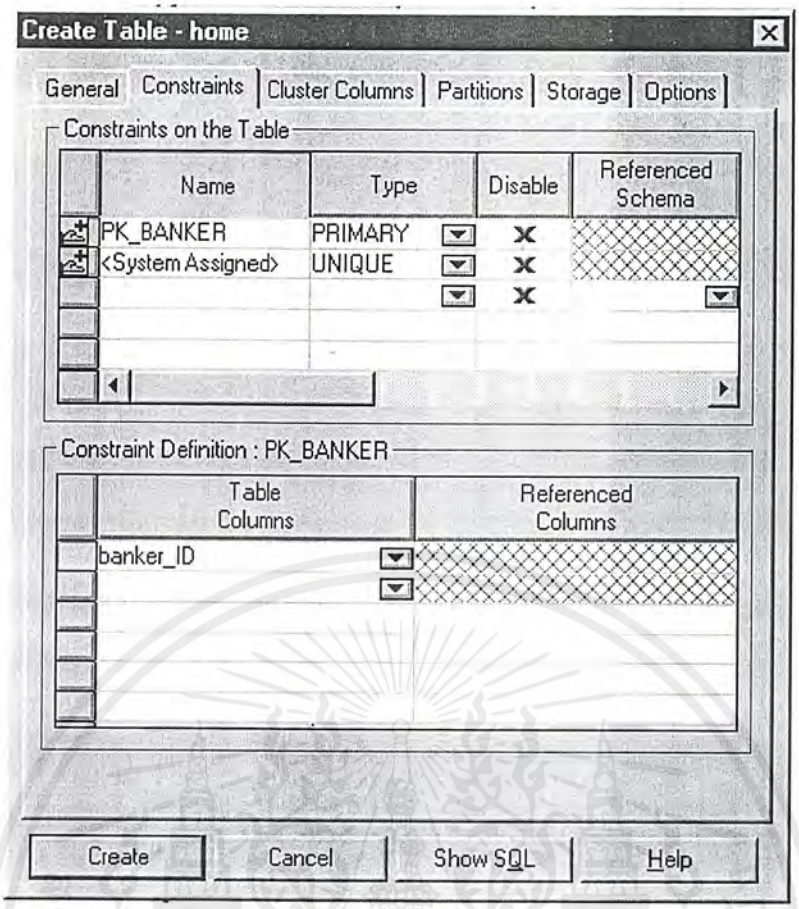


รูปที่ 7-2 ฟอร์มสร้างตารางใหม่

7.2.2 การกำหนดข้อบังคับ

ส่วน Constraints จะเป็นการกำหนดข้อบังคับต่างๆสำหรับตาราง

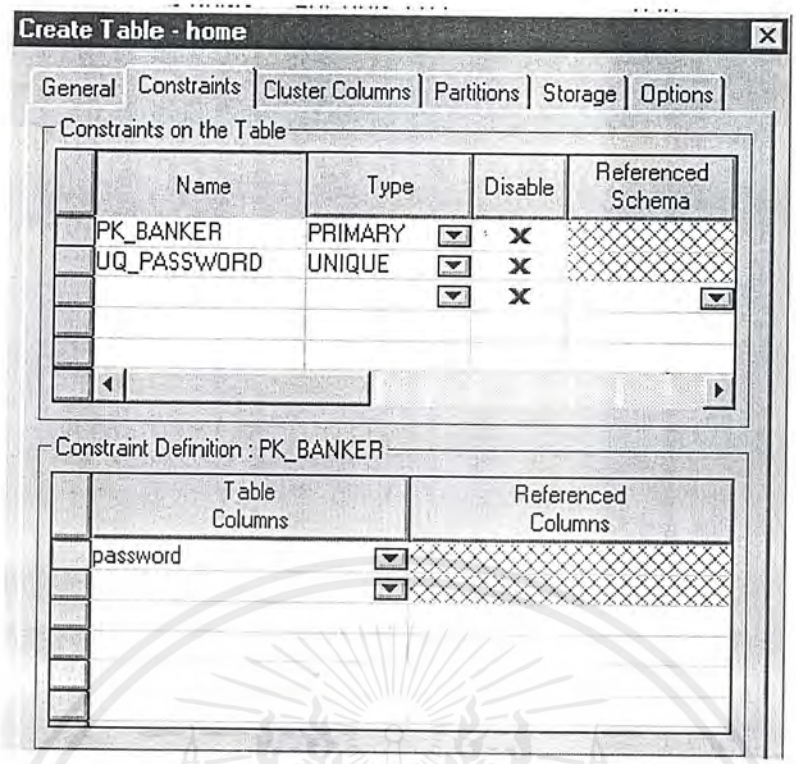
- Primary Key ในตัวอย่างนี้กำหนดคอลัมน์ Banker_ID เป็น primary key ของตาราง Banker สามารถตั้งชื่อข้อบังคับเพื่อความสะดวกเมื่อเกิดการละเมิด โดยใช้ชื่อ PK_BANKER หรือให้ระบบตั้งให้เองก็ได้ เลือกชนิดข้อบังคับ (Type) เป็น PRIMARY และข้อบังคับนี้มีผลต่อคอลัมน์ Banker_ID



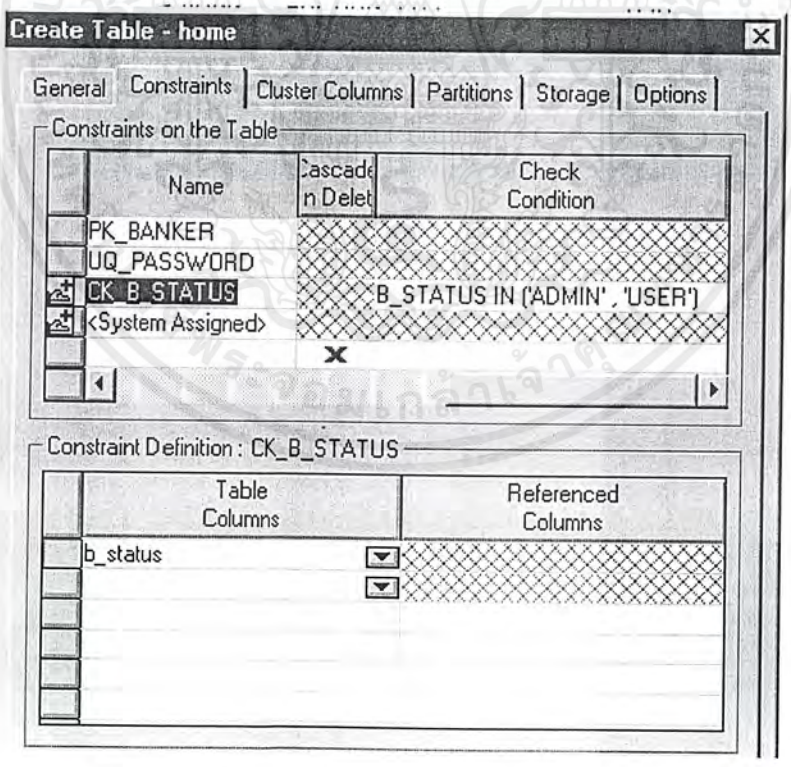
รูปที่ 7-3 การกำหนดข้อบังคับ Primary Key

- Unique กำหนดคอลัมน์ Password ให้เก็บค่าไม่ซ้ำกัน ทำการกำหนดได้ในลักษณะเดียวกัน Primary Key โดยตั้งชื่อข้อบังคับนี้ว่า UQ_PASSWORD เลือกชนิดข้อบังคับเป็น UNIQUE และคอลัมน์ตาราง Password ดังแสดงอยู่ในรูป 7-4

- Check ในตัวอย่างกำหนดให้สถานะของพนักงานเป็น Admin หรือ User เท่านั้น ดังนั้นจึงต้องทำการกำหนดขอบเขตของค่าในคอลัมน์ B_Status โดยทำลักษณะเช่นเดียวกัน ตั้งชื่อข้อบังคับนี้ว่า CK_B_STATUS เลือกชนิดข้อบังคับเป็น CHECK และคอลัมน์ตาราง B_Status แต่ข้อบังคับชนิด Check ต้องกำหนดเงื่อนไข (Check Condition) โดยกำหนดให้ B_Status IN ('Admin' , 'User') ดังแสดงอยู่ในรูป 7-5

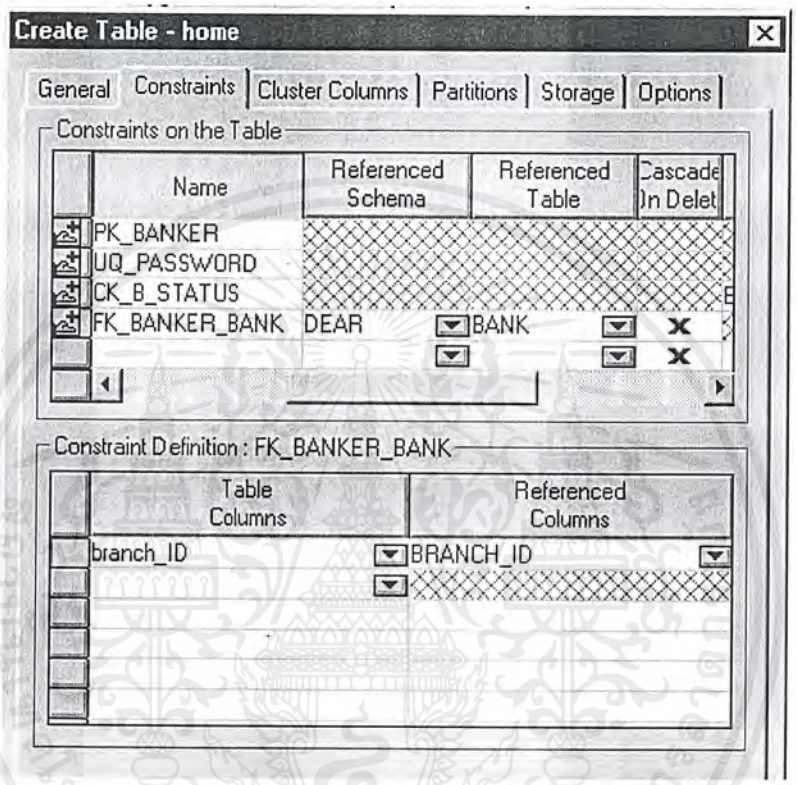


รูปที่ 7-4 การกำหนดข้อบังคับ Unique



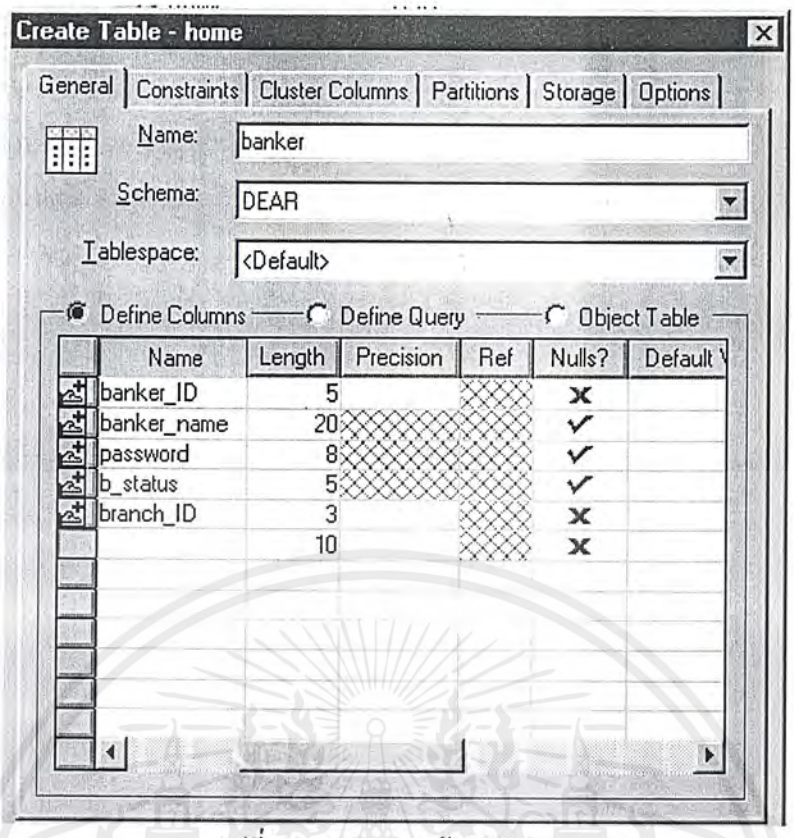
รูปที่ 7-5 การกำหนดข้อบังคับ Check

- Foreign Key ตาราง Banker จะอ้างอิงกับตาราง Bank โดยมี primary key ของตาราง Bank มาเป็น foreign key ในตาราง Banker การกำหนดข้อบังคับ Foreign Key ก็ทำเช่นเดียวกับข้อบังคับอื่นๆ ในตัวอย่างตั้งชื่อข้อบังคับว่า FK_BANKER_BANK เลือกชนิดข้อบังคับเป็น FOREIGN แต่ต้องกำหนดตารางที่อ้างอิงถึง (Referenced Table) คือ Bank และคอลัมน์ Branch_ID ของตาราง Banker จะอ้างอิงกับคอลัมน์ Branch_ID ของตาราง Bank ดังแสดงในรูป 7-6



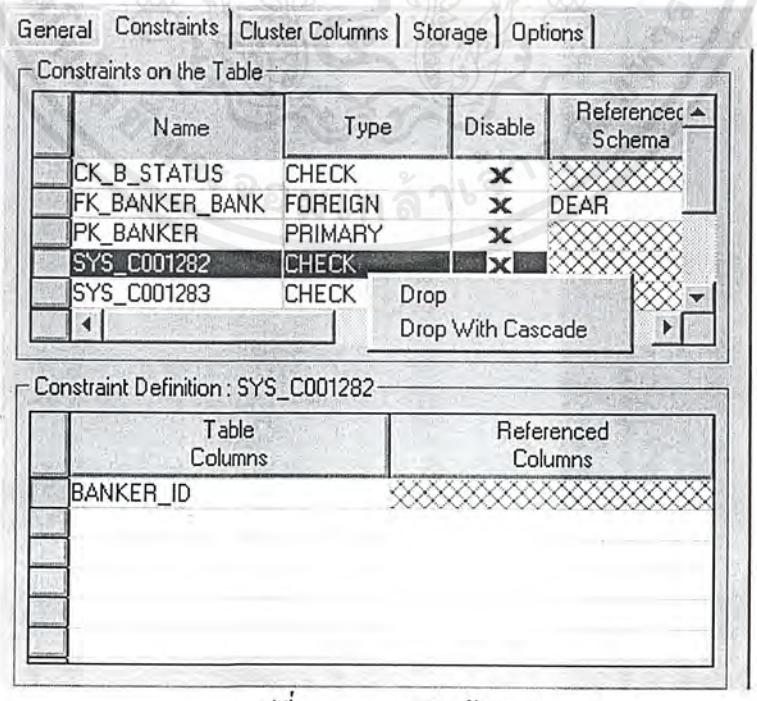
รูปที่ 7-6 การกำหนดข้อบังคับ Foreign Key

- Not Null การกำหนดว่าจะอนุญาติให้มีค่า Null ให้คอลัมน์ได้หรือไม่ สามารถทำได้จากส่วน General ในคอลัมน์ Nulls? หากมีเครื่องหมาย ✔ จะหมายถึงอนุญาติให้คอลัมน์นั้นมีค่า Null ได้ หากคลิกเปลี่ยนเป็นเครื่องหมาย ✘ ก็จะหมายถึงไม่อนุญาติให้มีค่า Null ดังแสดงในรูป 7-7



รูปที่ 7-7 การกำหนดข้อบังคับ Not Null

- การ Drop ข้อบังคับ เป็นการยกเลิกข้อบังคับ ทำได้โดยคลิกขวาที่ข้อบังคับนั้น แล้วเลือก Drop หากเป็นข้อบังคับ Primary Key หรือ Unique ซึ่งถูกอ้างอิงโดย foreign key ถูกยกเลิกจะต้องใช้การ Cascade ซึ่งจะ drop ข้อบังคับForeign Key ไปด้วย ดังแสดงในรูป 7-8



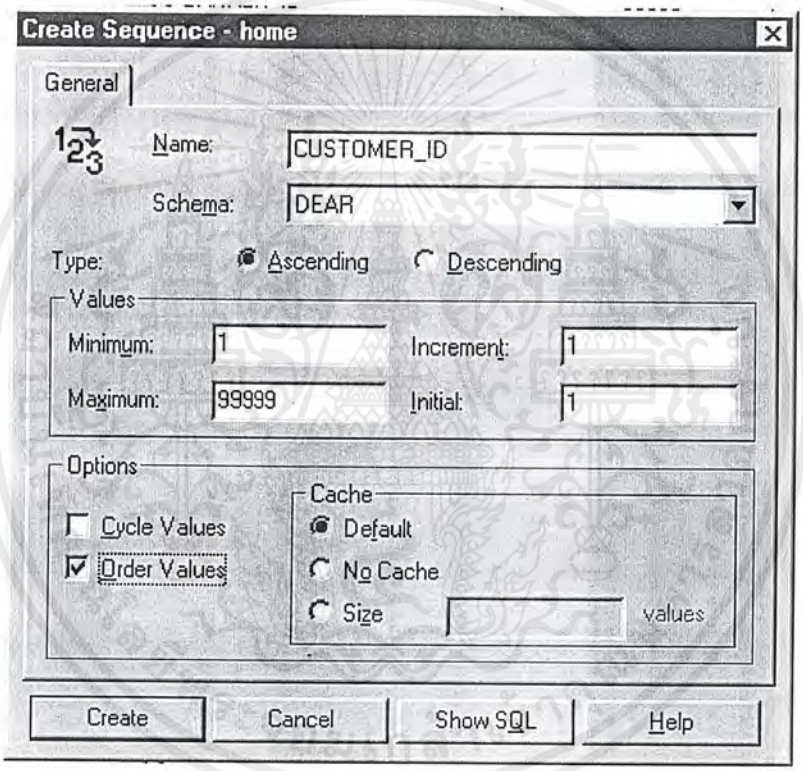
รูปที่ 7-8 การยกเลิกข้อบังคับ

เมื่อทำการกำหนดคอลัมน์ ชนิดข้อมูลและข้อบังคับต่างๆเสร็จแล้ว คลิก Create เพื่อสร้างตาราง ตารางที่สร้างเสร็จแล้วจะปรากฏอยู่ในโฟลเดอร์ Tables และสามารถกลับมาทำการแก้ไขอีกได้

7.3 การสร้างลำดับ (Sequence)

คอลัมน์ที่เป็น Primary Key เช่น CustomerID ต้องการเลขประจำตัวที่สร้างได้โดยอัตโนมัติและไม่ซ้ำกัน แต่ก่อนนี้การสร้างเลขลำดับต้องกำหนดเป็นออบเจ็กต์คงอยู่ (Persistent Object) ในโค้ดของแอพพลิเคชัน แต่ในปัจจุบันสามารถจัดการได้โดยออราเคิล

การสร้างเลขลำดับสามารถสร้างได้โดย คลิกขวาที่โฟลเดอร์ Sequences เลือก Create จะปรากฏฟอร์มสำหรับสร้างเลขลำดับขึ้นมา



รูปที่ 7-9 การสร้างเลขลำดับ

Name : ชื่อเลขลำดับ

Type : Ascending ค่าจะเพิ่มขึ้น Descending ค่าจะลดลง

Value : กำหนดช่วงค่าของเลขลำดับ

- Minimum ค่าต่ำสุด
- Increment เพิ่มขึ้นครั้งละ (ถ้าเลือก Descending จะเป็น Decrement)

- Maximum ค่าเลขลำดับสูงสุด จะต้องสอดคล้องกับความยาวที่กำหนดไว้ในคอลัมน์ด้วย
- Initial ค่าเริ่มต้นของเลขลำดับ

Option : Cycle Value ค่าจะเพิ่ม (หรือลด)ไปเรื่อยๆแล้วจะวนกลับ

Order Value ค่าจะเพิ่ม(หรือลด) ตามลำดับไปเรื่อยๆ

ฟังก์ชันที่ใช้กับเลขลำดับ

ชื่อฟังก์ชัน	การใช้	คำอธิบาย
NEXTVAL	Sequence_name.nextval	สร้างเลข ID ลำดับถัดไป
CURRVAL	Sequence_name.currval	ให้ค่าเลขลำดับปัจจุบัน

ตารางที่ 7-1 ฟังก์ชันที่ใช้กับเลขลำดับ

ตัวอย่างการใช้งานฟังก์ชัน

```
insert into Customer (Customer_ID , Customer_name , City)
values (customer_ID. Nextval, 'John' , 'New York');
```

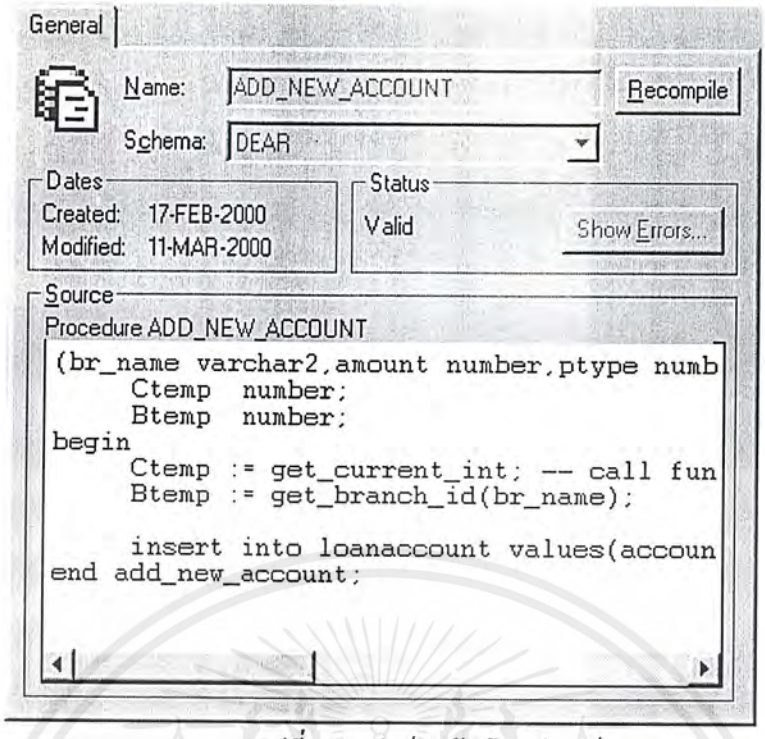
ข้อสังเกต เลขลำดับเมื่อสร้างขึ้นมาแล้ว แต่การ insert เกิดข้อผิดพลาดขึ้น เช่นละเมิดข้อบังคับ ทำให้ insert ไม่ได้ เลขลำดับที่สร้างขึ้นมานี้จะถูกข้ามไปเลย

เลขลำดับสามารถลบทิ้งได้โดยการคลิกขวาที่ชื่อลำดับนั้น แล้วเลือก Remove

7.4 การสร้างโพรซีเยอร์ด้วย Schema Manager

ทำได้โดยคลิกขวาที่โฟลเดอร์ Procedures เลือก Create.. จะปรากฏฟอร์มดังแสดงในรูปที่ 7-10

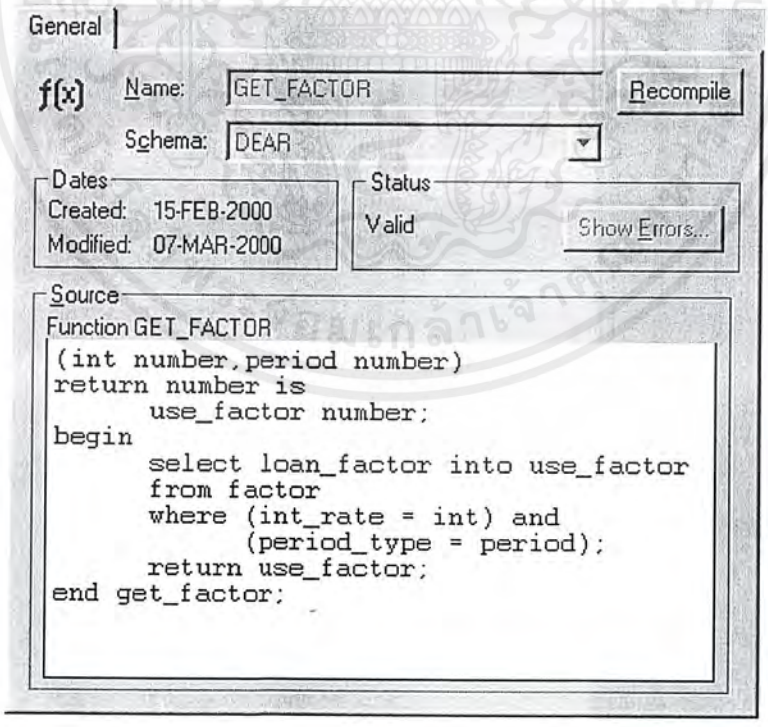
ในส่วน Source จะเป็นภาษา PL/SQL โดยโครงสร้างการเขียนโพรซีเยอร์จะเป็นไปตามที่แสดงไว้ในหัวข้อ 7.1.3.1 โดยต่อจากชื่อโพรซีเยอร์จะเป็นตัวแปรอากิวเมนต์ (ถ้ามี) ภายในโพรซีเยอร์สามารถเรียกใช้ฟังก์ชันหรือโพรซีเยอร์อื่นๆได้ เมื่อเขียนเสร็จกดปุ่ม Create จะทำการคอมไพล์โพรซีเยอร์ที่เขียนขึ้นมา หากคอมไพล์ไม่ผ่านสถานะจะเป็น Invalid ดูข้อผิดพลาดได้จาก Show Errors ... หลังจากทำการแก้ไขกดปุ่ม Apply จะทำการบันทึกและคอมไพล์ใหม่อีกครั้ง



รูปที่ 7-10 ฟอร์มสร้างโพรซีเยอร์

7.5 การสร้างฟังก์ชันด้วย Schema Manager

ทำได้โดยคลิกขวาที่โฟลเดอร์ Functions เลือก Create จะปรากฏฟอร์มดังแสดงในรูปที่ 7-11

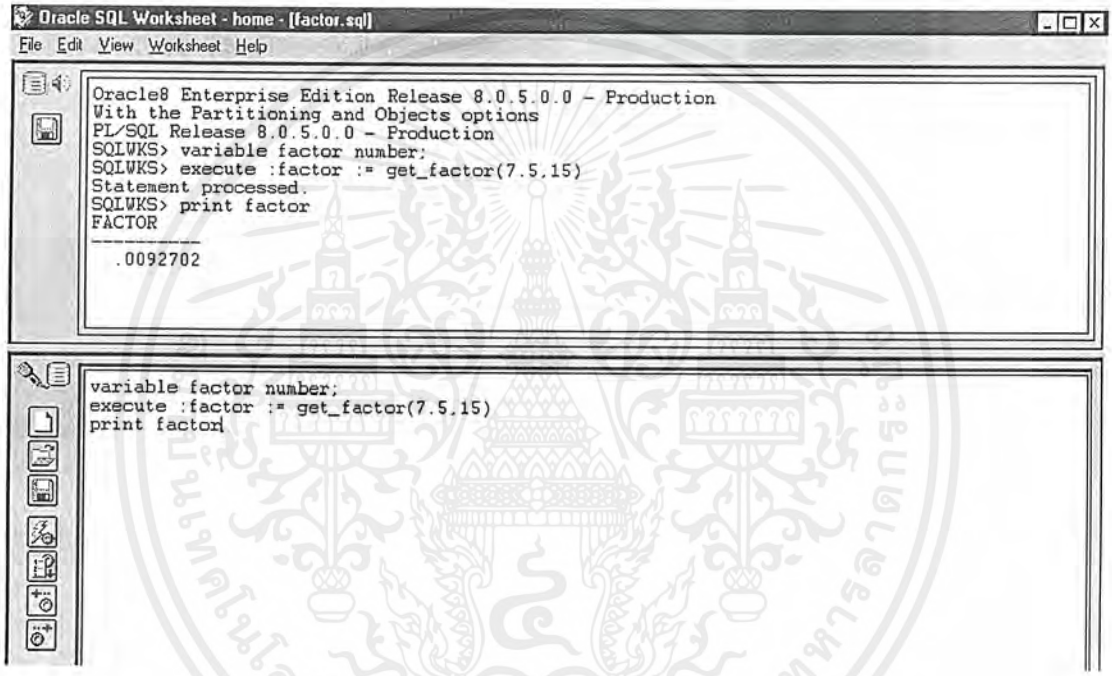


รูปที่ 7-11 ฟอร์มสร้างฟังก์ชัน

การสร้างฟังก์ชันมีลักษณะเดียวกับการสร้างโพรซีเยอร์ โครงสร้างการเขียนฟังก์ชันเป็นไปตามที่แสดงไว้ในหัวข้อ 7.1.3.2

7.6 Oracle SQL Worksheet

SQL Worksheet เป็นเครื่องมือตัวหนึ่งในออราเคิล ช่วยให้สามารถทำงานแบบปฏิสัมพันธ์กับฐานข้อมูลได้เช่นเดียวกับ SQL*PLUS ทั้งการสร้างตาราง,โพรซีเยอร์,ฟังก์ชัน และ การ insert , update ข้อมูลในตารางเป็นต้น แต่สะดวกมากกว่า โดยสามารถทำได้ทีละหลายคำสั่ง , มีรายการคำสั่งที่ได้ทำไปแล้วสามารถย้อนกลับไปยังคำสั่งนั้นได้อีก และบันทึกเก็บเป็นไฟล์ .sql ไว้ใช้งานในครั้งต่อๆไป สามารถใช้เป็นเครื่องมือช่วยในการทดสอบการทำงานของโพรซีเยอร์และฟังก์ชันต่างๆที่ได้เขียนไว้



รูปที่ 7-12 Oracle SQL Worksheet

บทที่ 8
การสร้างโปรแกรมประยุกต์ด้วยโปรแกรม VisualAge for Java

8.1 พื้นฐานการเขียนโปรแกรมด้วยภาษาจาวา (Java)

8.1.1 ลักษณะของโปรแกรมภาษาจาวา

ภาษาจาวา เป็นโปรแกรมภาษาที่มีลักษณะคล้ายกันกับโปรแกรมภาษา C และ C++ แต่แตกต่างกันที่ภาษาจาวาได้ถูกออกแบบมาเพื่อให้ใช้งานบนระบบปฏิบัติการใดๆ ก็ได้ เช่น บน DOS, Windows, UNIX หรือแม้กระทั่งบนเครื่องเมนเฟรม โดยการเขียนโปรแกรมเพียงครั้งเดียวให้สามารถใช้งานบนระบบปฏิบัติการหลายๆ ระบบได้

การเขียนโปรแกรมด้วยภาษาจาวาจะต้องเขียนโปรแกรมในลักษณะเชิงวัตถุ (Object – Oriented Programming : OOP) ซึ่งการเขียนโปรแกรมในลักษณะนี้มีข้อดีที่สามารถนำโปรแกรมที่มีการเขียนไว้แล้วนำกลับมาใช้ใหม่ได้

โปรแกรมภาษาจาวาจะเป็นไฟล์ที่มีนามสกุลเป็น .java เมื่อเขียนโปรแกรมเสร็จจะต้องทำการคอมไพล์ให้ได้เป็นไบต์โคดจาวาซึ่งมีนามสกุลเป็น .class ไฟล์ที่ได้นี้สามารถนำไปเรียกใช้งานในระบบปฏิบัติการใดๆ ก็ได้โดยใช้ตัวแปลงไบต์โค้ดภาษาจาวาของระบบปฏิบัติการนั้นๆ

8.1.2 ลักษณะของตัวแปรในภาษาจาวา

ชนิดของตัวแปรในภาษาจาวาสามารถแบ่งออกได้เป็น 3 ชนิดใหญ่ๆ คือ ตัวแปรพื้นฐาน , ตัวแปรคลาส , ตัวแปรอาร์เรย์ การตั้งชื่อตัวแปรนั้นไม่สามารถตั้งชื่อให้ซ้ำกับชื่อคำสงวน (reserved word) ที่มีอยู่ของภาษาจาวาได้ และชื่อที่มีตัวอักษรเป็นตัวพิมพ์ใหญ่และตัวพิมพ์เล็กนั้นมีผลต่างกัน เช่น JAVA และ java ถือว่าเป็นคนละชื่อกัน

ตัวแปรพื้นฐาน คือ ชนิดของตัวแปรเบื้องต้นที่ใช้ในการเขียนโปรแกรม ในภาษาจาวานี้จะถือว่าชื่อของตัวแปรพื้นฐานเหล่านี้เป็นคำสงวนและจะเขียนด้วยตัวอักษรตัวเล็กทั้งหมด

ชนิดของตัวแปร	ค่าเริ่มต้น
Byte	0
Short	0
Int	0
Long	0L
Float	0.0f
Double	0.0d
Char	'0'
Boolean	False

ตารางที่ 8-1 ลักษณะตัวแปรพื้นฐานของโปรแกรมภาษาจาวา

การประกาศตัวแปรพื้นฐาน มีลักษณะดังนี้

ชนิดของตัวแปร ชื่อตัวแปร ;

ตัวอย่างเช่น

```
int         min , max;
String      name  =  "JAVA";
boolean     Flag , flag  =  true;
```

ตัวแปรคลาส คือ ตัวแปรที่มีชนิดเป็นคลาส หรือที่เรียกว่าอินสแตนซ์ (instance) หรือออบเจ็กต์ (object) ในภาษาจาวาจะมีคลาสอยู่ส่วนหนึ่งที่เก็บไว้แยกเป็นหมวดหมู่เรียกว่าแพ็กเกจ (package) คลาสแต่ละคลาสในแพ็กเกจนั้นจะมีชื่อที่ขึ้นต้นด้วยตัวอักษรตัวพิมพ์ใหญ่เสมอ เช่น คลาส String เป็นคลาสที่อยู่ในแพ็กเกจ java.lang ในการสร้างอินสแตนซ์ของคลาสหรือออบเจ็กต์นั้น จะใช้โอเปอเรเตอร์ new ในการสร้าง โดยมีรูปแบบดังนี้

ชื่อของคลาส ชื่อของอินสแตนซ์ = new ชื่อคลาส( )

ตัวอย่างเช่น

```
Dog         puddle  =  new Dog();
String      name  =  new String("JAVA");
```

ในคลาสอาจจะมีการประกาศตัวแปรชนิดต่างๆ เช่น ตัวแปรพื้นฐาน , ตัวแปรของคลาส, ตัวแปรประเภทค่าคงที่ โดยตัวแปรพื้นฐานมีลักษณะเป็นตัวแปรของอินสแตนซ์ที่ การเปลี่ยนแปลงของค่าตัวแปรที่อยู่ในอินสแตนซ์หนึ่งๆจะไม่มีผลกับค่าของตัวแปรที่อยู่ในอินสแตนซ์อื่นๆ ส่วนตัวแปรของคลาสนั้น แต่ละอินสแตนซ์ของคลาสหนึ่งๆจะมีค่าของตัวแปรของคลาสเป็นค่าเดียวเหมือนกันทั้งหมด การเปลี่ยนแปลงของตัวแปรคลาสจะส่งผลกับอินสแตนซ์ทุกๆตัวที่เป็นอินสแตนซ์ของคลาสนั้นๆ

การประกาศตัวแปรของคลาสจะใช้คำว่า static เติมนำหน้าไว้ ดังนี้

static ชนิดของตัวแปร ชื่อตัวแปรที่ 1 , ชื่อตัวแปรที่ 2 , ...;

ตัวแปรประเภทค่าคงที่ เป็นการประกาศตัวแปรให้มีค่าคงที่ โดยใช้คำว่า final เติมนำหน้า ดังนี้

final ชนิดของตัวแปร ชื่อตัวแปร = ค่าตัวแปรนั้นๆ

ตัวแปรอาร์เรย์ ประกอบด้วยตัวแปรชนิดเดียวกันจำนวนหนึ่ง ซึ่งตัวแปรเหล่านั้นอาจมีชนิดเป็นตัวแปรพื้นฐานหรือคลาสก็ได้ การอ้างอิงถึงสมาชิกในตัวแปรอาร์เรย์จะใช้ index เป็นตัวอ้าง โดยตัวแปรที่มีสมาชิก n ตัว จะใช้ index ที่มีค่าตั้งแต่ 0 ถึง n – 1

การประกาศตัวแปรอาร์เรย์ มีลักษณะดังนี้

ชนิดของตัวแปรอาร์เรย์ [ ] ชื่อของตัวแปรอาร์เรย์ ;
หรือ
ชนิดของตัวแปรอาร์เรย์ ชื่อของตัวแปรอาร์เรย์ [ ] ;

ตัวอย่างเช่น

```
int [ ]          array_int;
int              array_int[ ] = {2,4,6,8,10};
```

8.2 การใช้งาน JDBC Drivers สำหรับโปรแกรมประยุกต์ภาษาจาวาในการติดต่อกับฐานข้อมูล

8.2.1 ชนิดของ JDBC Drivers

เนื่องจาก Application นี้ได้ใช้ Oracle เป็น DBMS สำหรับการจัดเก็บฐานข้อมูล ดังนั้นในการติดต่อกับฐานข้อมูลจึงได้ใช้ JDBC Drivers ที่มีให้อยู่ใน Oracle 8.05 ซึ่งเป็น JDBC version 1.22 แบ่งเป็น 2 ชนิด คือ

- JDBC Thin สำหรับ Java applets และ Java applications
- JDBC OCI สำหรับ Java applications

8.2.1.1 JDBC Thin

เป็นไดรเวอร์ Type 4 ที่ใช้สำหรับการติดต่อโดยตรงกับ Oracle โดยใช้การติดต่อผ่าน TCP/IP สำหรับเวอร์ชั่นของ Oracle ที่เป็น Net8 เนื่องจากว่าไดรเวอร์นี้เขียนขึ้นด้วยภาษาจาวาจึงทำให้มีความอิสระในการใช้งานไม่ขึ้นอยู่กับรูปแบบของระบบปฏิบัติการ สำหรับการใช้ Thin driver นี้ไม่จำเป็นต้องลงโปรแกรม Oracle ในทางฝั่งไคลเอนท์โดยการทำงานจะใช้การติดต่อผ่าน TCP/IP ไปยังฝั่งเซิฟเวอร์ที่มีการลงโปรแกรม Oracle Database version 8.0.4 ขึ้นไป

8.2.1.2 JDBC OCI

เป็น JDBC driver Type 2 การทำงานของไดรเวอร์ชนิดนี้จะใช้ OCI (Oracle Call Interface) ในการติดต่อกับฐานข้อมูลออราเคิล โดย driver นี้สามารถติดต่อได้กับออราเคิลเซิฟเวอร์ที่มี version 8.0.4 ขึ้นไป

เนื่องจาก JDBC OCI นี้เป็นรูปแบบเฉพาะของออราเคิล จึงใช้ได้กับระบบปฏิบัติการดังนี้

- Solaris : version 2.5 ขึ้นไป

- Window : 95 และ NT 3.51 ขึ้นไป

การใช้ไดรเวอร์ชนิดนี้จะต้องมีการลงโปรแกรม Oracle 8.0.4 ทางฝั่งไคลเอนท์ซึ่งรวมถึง Net8 และไฟล์อื่นๆที่เกี่ยวข้องด้วย เนื่องจากไดรเวอร์ชนิดนี้ใช้ C library ซึ่งเป็นรูปแบบที่เฉพาะ จึงไม่เหมาะสำหรับการใช้งานของ Java Applets

8.2.2 ขั้นตอนการติดตั้ง JDBC Drivers

1. เลือก JDBC Driver ให้ตรงกับระบบปฏิบัติ โดยออราเคิลได้จัดเตรียมไฟล์ไดรเวอร์ในรูปแบบต่างๆ ดังนี้

- Oracle 8.0.4 CD
- Windows zip file (ใช้กับ Windows 95 หรือ Windows NT 3.51 ขึ้นไป)
- Solaris tar file (ใช้กับ Solaris 2.5 ขึ้นไป)
- อื่นๆ เช่น tar file หรือ zip file

2. ติดตั้งไดรเวอร์เข้ากับระบบปฏิบัติการ

การติดตั้งจากแผ่น CD

- รันโปรแกรม Oracle Installer
- เลือก JDBC Driver จากรายการและทำการติดตั้ง

การติดตั้งสำหรับ Windows95 หรือ Windows NT โดยมีโปรแกรม Oracle Installer

- Unzip file ไปใน c:\temp
- ชี้ตัวติดตั้งไปยัง c:\temp
- ใช้ Oracle Installer ในการติดตั้ง โดยเลือกติดตั้งไดรเวอร์ในไดเรกทอรี่บนสุดของออราเคิล เช่น [ORACLE HOME]\JDBC
- กำหนด CLASSPATH เพิ่มเติมดังนี้

```
[ORACLE HOME]\JDBC\LIB\CLASSES111.zip  สำหรับผู้ที่ใช้ JDK 1.1.1 ขึ้นไป
```

```
[ORACLE HOME]\JDBC\LIB\CLASSES102.zip  สำหรับผู้ที่ใช้ JDK 1.0.2
```

การติดตั้งสำหรับ Windows95 หรือ Windows NT โดยไม่มีโปรแกรม Oracle Installer

- Unzip file ไปใน c:\JDBC
- กำหนด CLASSPATH เพิ่มเติมดังนี้

```
C:\JDBC\LIB\CLASSES111.zip  สำหรับผู้ที่ใช้ JDK 1.1.1 ขึ้นไป
```

C:\JDBC\LIB\CLASSES102.zip สำหรับผู้ที่ใช้ JDK 1.0.2

- กำหนด PATH = C:\JDBC\LIB เพิ่มเติม

การติดตั้งสำหรับ Solaris

- สร้างไดเรกทอรี่ /local/jdbc
- Un-tar ไปใน /local/jdbc
- กำหนด CLASSPATH เพิ่มเติมดังนี้

/local/jdbc/lib/classes111.zip สำหรับผู้ที่ใช้ JDK 1.1.1 ขึ้นไป

/local/jdbc/lib/classes102.zip สำหรับผู้ที่ใช้ JDK 1.0.2

- เพิ่ม /local/jdbc/lib ไปใน LD_LIBRARY_PATH

3. กำหนดสิ่งแวดล้อมต่างๆให้ถูกต้อง

สำหรับ JDBC drivers ของออราเคิลทุกๆ ตัวจะต้องมีการกำหนด CLASSPATH ไปยัง zip file ที่บรรจุคลาสของจาวาในการติดตั้งไดรเวอร์

สำหรับ JDBC OCI Drivers ของออราเคิล จะต้องมีการกำหนด PATH (Windows) หรือ LD_LIBRARY_PATH (Solaris) ไปยังไดเรกทอรี่ที่บรรจุ DLL หรือ so library file

4. ทดสอบผลการติดตั้ง

สามารถทดสอบการทำงานได้จากโปรแกรมตัวอย่างซึ่งมีอยู่ในออราเคิล โดยได้แบ่งไดเรกทอรี่ที่เก็บไฟล์ตัวอย่างตามชนิดของไดรเวอร์ที่ใช้ในการติดต่อ ซึ่งได้แก่ OCI7 , OCI8 , Thin , Thin-1.0.2 ไฟล์ตัวอย่างนี้คือ JdbcCheckup.java และ Employee.java โดยไฟล์ตัวแรกใช้ทดสอบการติดตั้งและไฟล์ตัวที่สองใช้สำหรับทดสอบการทำงานกับฐานข้อมูล

8.2.3 การใช้งาน JDBC Drivers ของออราเคิล

ขั้นตอนการใช้ JDBC OCI และ JDBC Thin Driver

- Import JDBC classes

กำหนด import statements เริ่มต้นที่หัวโปรแกรม เพื่อให้โปรแกรมสามารถเรียกใช้ JDBC classes ได้ โดยมีการกำหนดดังนี้

```
import java.sql.* ;
```

```
import java.math.* ;
```

- ลงทะเบียน JDBC driver (OCI และ Thin Driver)

ในโปรแกรมจะต้องทำการลงทะเบียนออราเคิลไดรเวอร์ก่อนจึงจะสามารถใช้งานมันได้ โดยทำการลงทะเบียนเพียงครั้งเดียวสำหรับโปรแกรมประยุกต์จาวา มีรูปแบบดังนี้

```
DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ( ));
```

- เปิดการเชื่อมต่อกับฐานข้อมูล

เปิดการติดต่อกับฐานข้อมูลโดยใช้ method getConnection ซึ่งจะต้องระบุชนิดของการติดต่อเป็น String เพื่อแสดงถึงการใช้ไดรเวอร์ในการติดต่อและต้องระบุ user logon และรหัสผ่านด้วย ในตัวอย่างเป็นการติดต่อกับฐานข้อมูลจากผู้ใช้ "scott" และมีรหัสผ่านคือ "tiger"

ตัวอย่างการติดต่อโดยใช้ JDBC OCI Driver

```
Connection conn =
  DriverManager.getConnection ("jdbc:oracle:oci8:@mydatabase",
                "scott", "tiger");
```

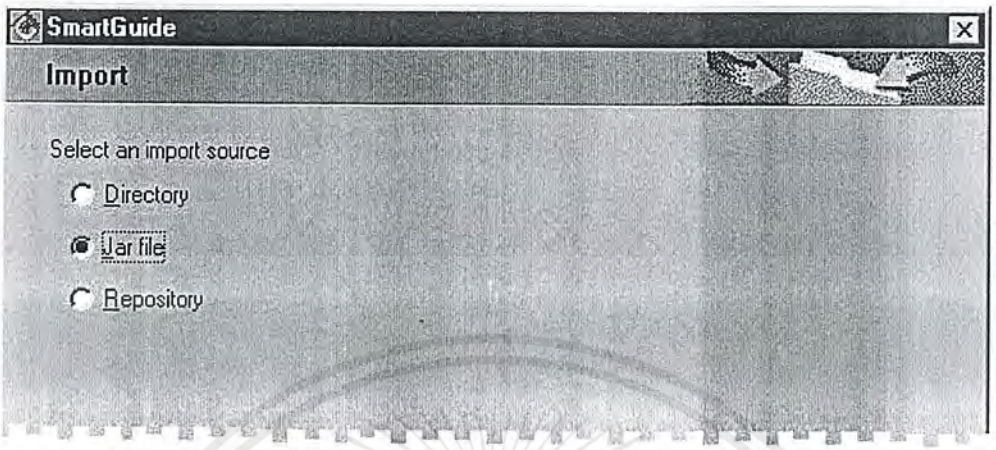
ตัวอย่างการติดต่อโดยใช้ JDBC Thin Driver

```
Connection conn =
  DriverManager.getConnection ("jdbc:oracle:thin:@myhost:1521:orcl",
                "scott", "tiger");
```

8.2.4 การใช้งาน JDBC Driver สำหรับโปรแกรม VisualAge for Java

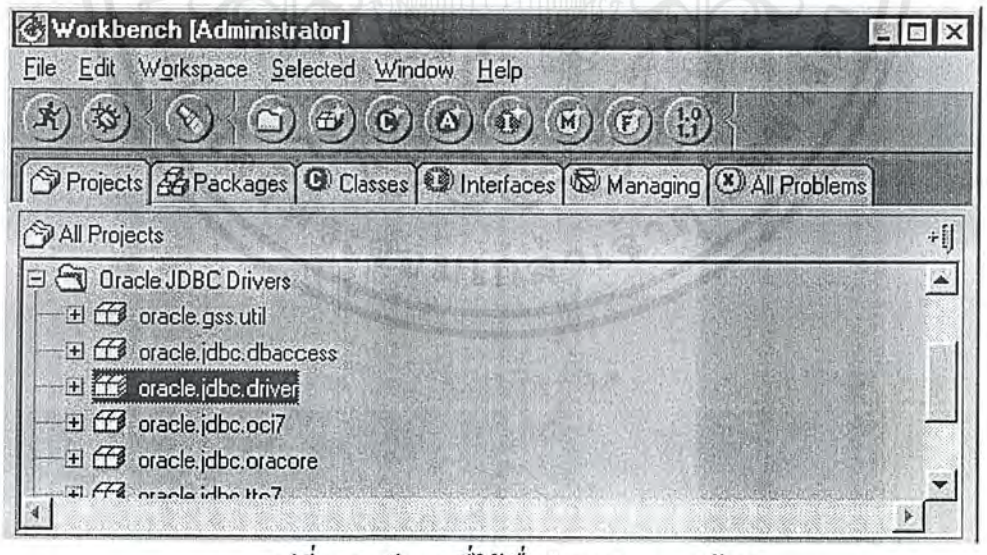
1. จัดหาไฟล์ JDBC Driver ที่ต้องการใช้ โดยในโครงงานนี้ ได้เลือกใช้ JDBC Thin Driver เนื่องจากไม่ต้องการลงโปรแกรมออราเคิลที่ฝั่งไคลเอนท์ทุกๆเครื่อง และเนื่องจากได้เลือกใช้โปรแกรม VisualAge for Java version 2.0 ซึ่งสนับสนุน JDK 1.1.6 จึงต้องเลือกใช้ไฟล์ classes111.zip ที่มีให้อยู่ในออราเคิล

2. ทำการ import file classes111.zip โดยเปิด Workbench ของโปรแกรม VisualAge แล้วเลือก import แบบ Jar file เลือกชนิดของไฟล์ที่ต้องการ import เป็นแบบ .class และพิมพ์ชื่อ Project เป็น Oracle JDBC Drivers



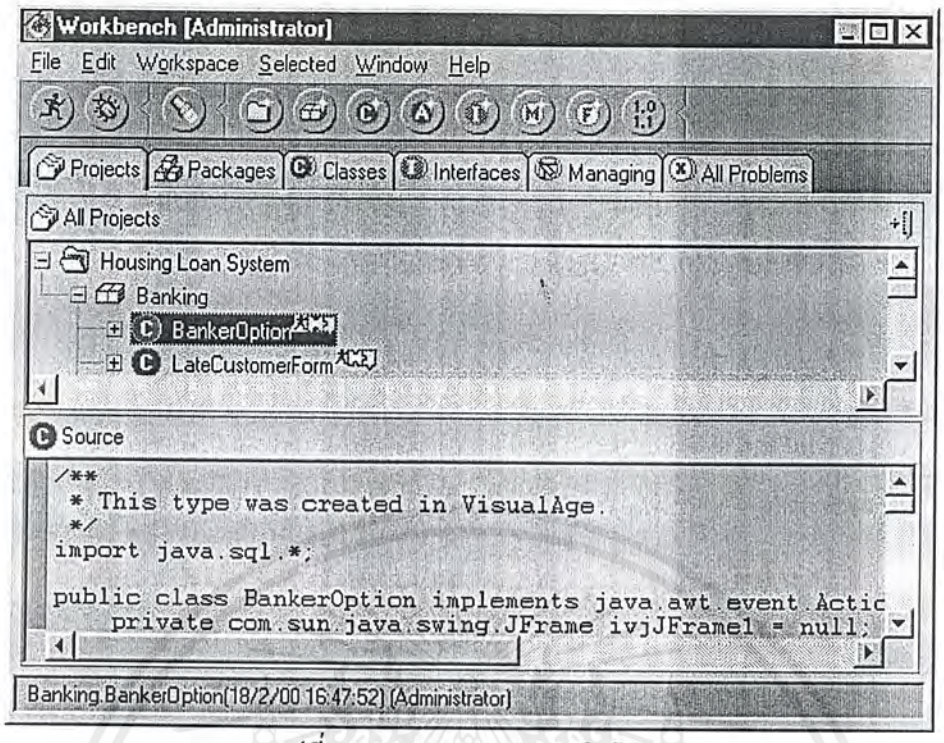
รูปที่ 8-1 เลือกการ Import แบบ Jar file

3. โปรแกรม VisualAge จะทำการ import file .class ที่อยู่ในไฟล์ classes111.zip มาเก็บไว้ใน Project ที่ชื่อ Oracle JDBC Drivers ที่เราได้ระบุไว้ในตอนต้น โดยแยกเป็นแพ็กเกจต่างๆ ซึ่งในแต่ละแพ็กเกจก็จะประกอบด้วยคลาสหลายๆ คลาส ในการลงทะเบียน JDBC Driver เราได้อ้างถึงคลาส OracleDriver( ) ซึ่งอยู่ในแพ็กเกจที่มีชื่อแพ็กเกจว่า oracle.jdbc.driver



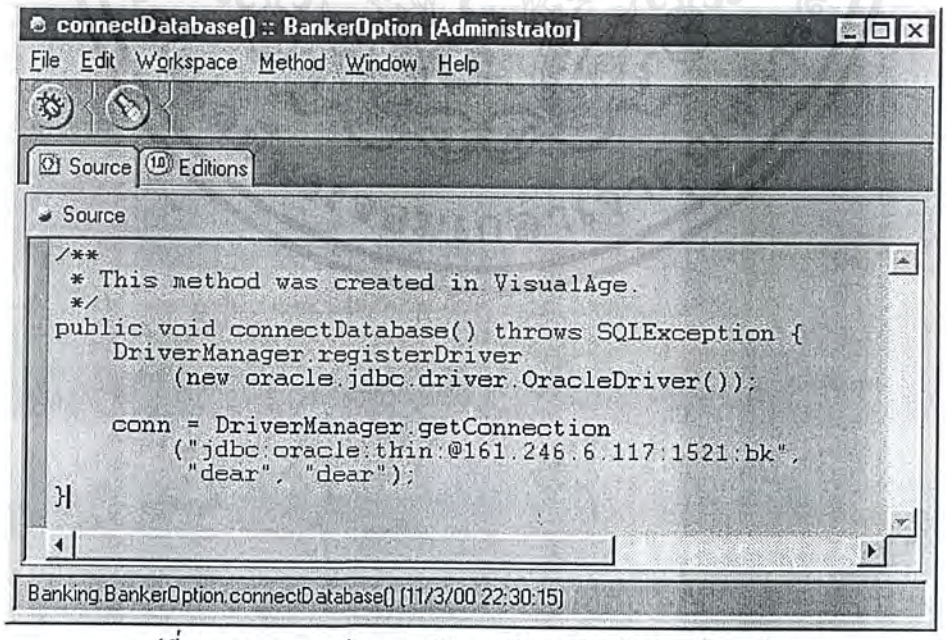
รูปที่ 8-2 แพ็คเกจที่ได้เมื่อทำการ Import เข้ามา

4. กำหนด import statement ที่ส่วนหัวของโปรแกรม เพื่อให้โปรแกรมนั้นสามารถเรียกใช้คลาสที่ใช้สำหรับการติดต่อกับฐานข้อมูล โดยคลาสต่างๆเหล่านี้ถูกบรรจุอยู่ในแพ็กเกจที่มีชื่อว่า java.sql การระบุทำได้ดังรูป 8-3



รูปที่ 8-3 การกำหนด Import ในโปรแกรม

5. เขียนโค้ดในการติดต่อกับฐานข้อมูล โดยจะต้องทำการลงทะเบียน JDBC Thin Driver เสียก่อน แล้วจึงเปิดการเชื่อมต่อ โดยระบุ TCP/IP ที่ใช้ติดต่อกับเซิฟเวอร์ฐานข้อมูล,ชื่อผู้ใช้และรหัสผ่าน บนฝั่งของเซิฟเวอร์



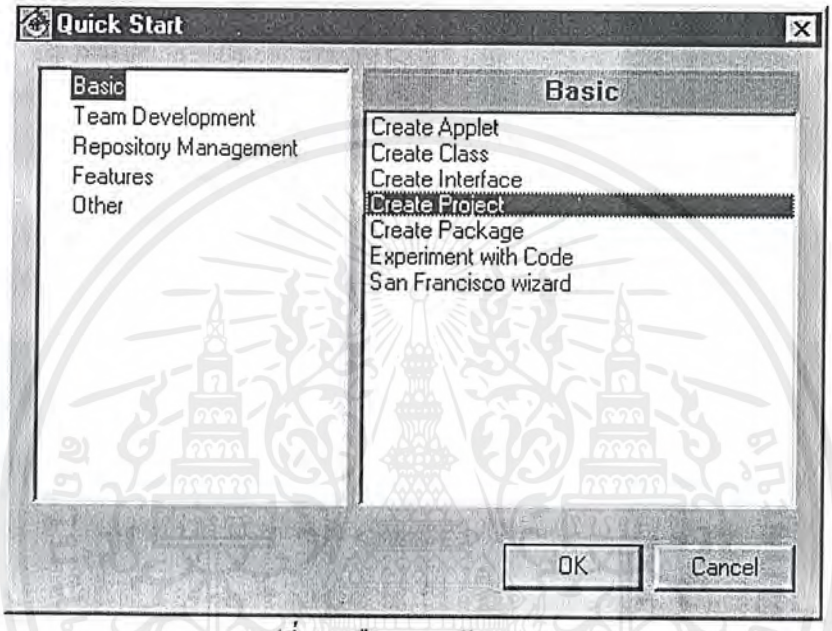
รูปที่ 8-4 การลงทะเบียนและเปิดการติดต่อกับเซิฟเวอร์ฐานข้อมูล

8.3 การสร้างโปรแกรมประยุกต์ภาษาจาวาโดยโปรแกรม VisualAge for Java

8.3.1 เริ่มต้นด้วยการสร้าง Project

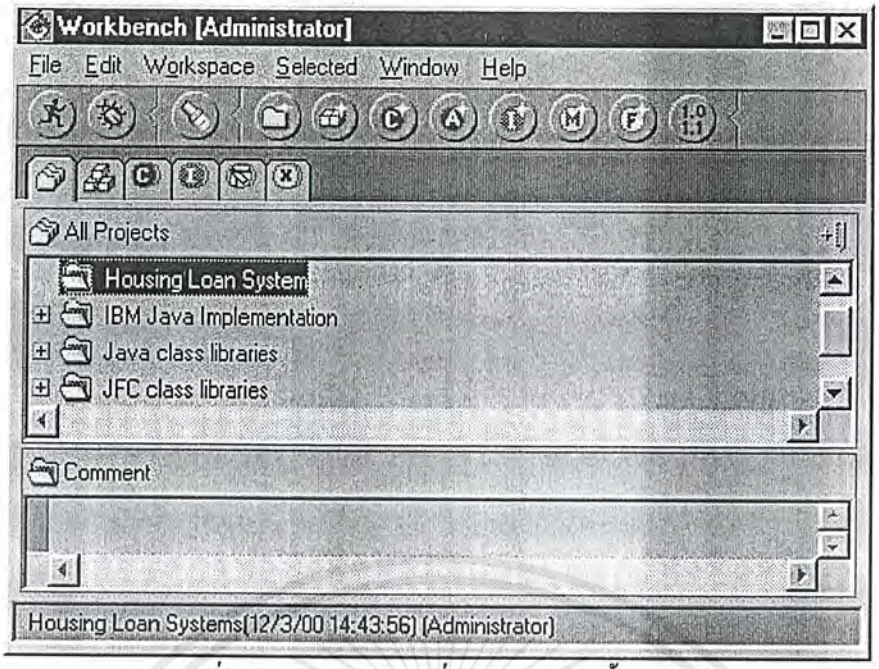
ในโปรแกรม VisualAge จะมีการแบ่งแยกงานต่างๆ เป็น Project ดังนั้นเมื่อเราต้องการสร้างโปรแกรมใดๆ ขึ้นมา ก็ควรเริ่มด้วยการสร้าง Project ของงานนั้นๆ เสียก่อน ซึ่งมีขั้นตอนดังนี้

1. ใช้ Quick Start สำหรับสร้าง Project ขึ้นมาใหม่ โดยการเปิดโปรแกรม VisualAge เมื่อเข้าสู่หน้าจอ Workbench เลือก Quick Start ในเมนู File แล้วเลือก Create Project ดังรูป 8-5



รูปที่ 8-5 เลือกการสร้าง Project

2. ทำการป้อนชื่อ Project ที่ต้องการสร้าง โดยในโครงงานนี้ได้ตั้งชื่อ Project เป็น Housing Loan System แล้วจึงคลิกปุ่ม Finish เพื่อเริ่มต้นการสร้าง ขณะนี้ใน Project Housing Loan System ยังไม่มีอะไรบรรจุอยู่ ซึ่งจะต้องทำการสร้างคลาสและแพ็กเกจในการใช้งานต่อไป

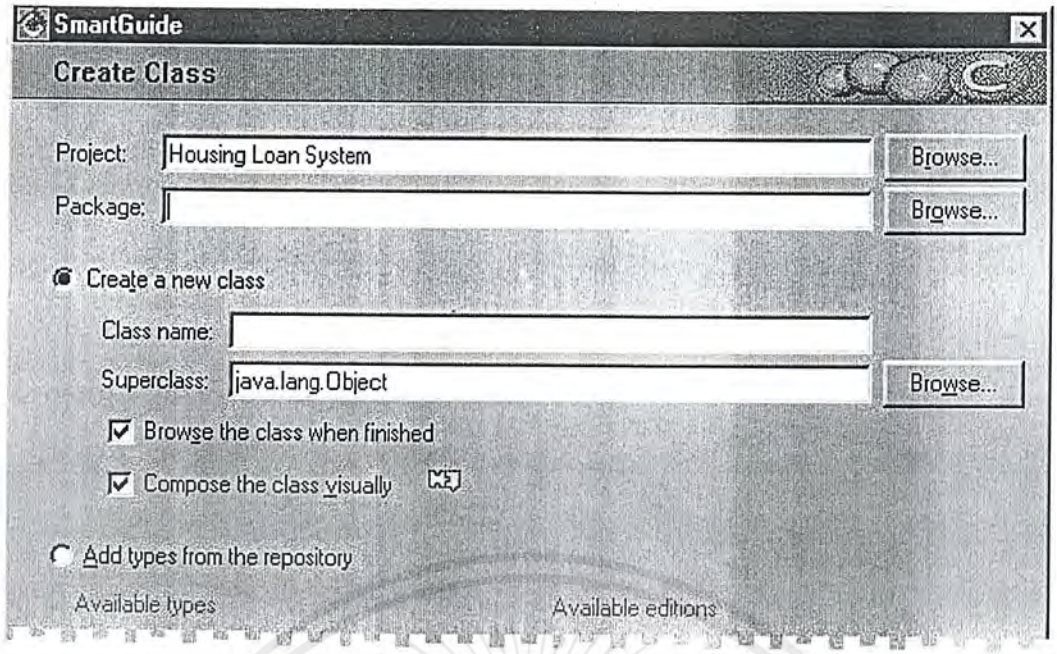


รูปที่ 8-6 แสดง Project ที่ได้ทำการสร้างขึ้นมาใหม่

3. ทำการสร้างคลาสตามที่ได้ออกแบบเอาไว้ โดยในโปรแกรม VisualAge จะมีการแบ่งคลาสตามแพ็กเกจ ซึ่งในแต่ละแพ็กเกจอาจจะมีหลายๆ คลาสก็ได้ที่เกี่ยวข้องกัน จากคลาสไดอะแกรมที่ได้ทำการออกแบบไว้สามารถแบ่งได้เป็น 2 ส่วน คือ คลาสที่ทำการแปลงไปเป็นตารางในฐานข้อมูลออราเคิลและคลาสในส่วนที่เป็นส่วนติดต่อของโปรแกรมสำหรับผู้ใช้ติดต่อกับระบบ ดังนั้นเราจึงทำการสร้างคลาสเฉพาะส่วนที่เป็นติดต่อซึ่งได้แก่

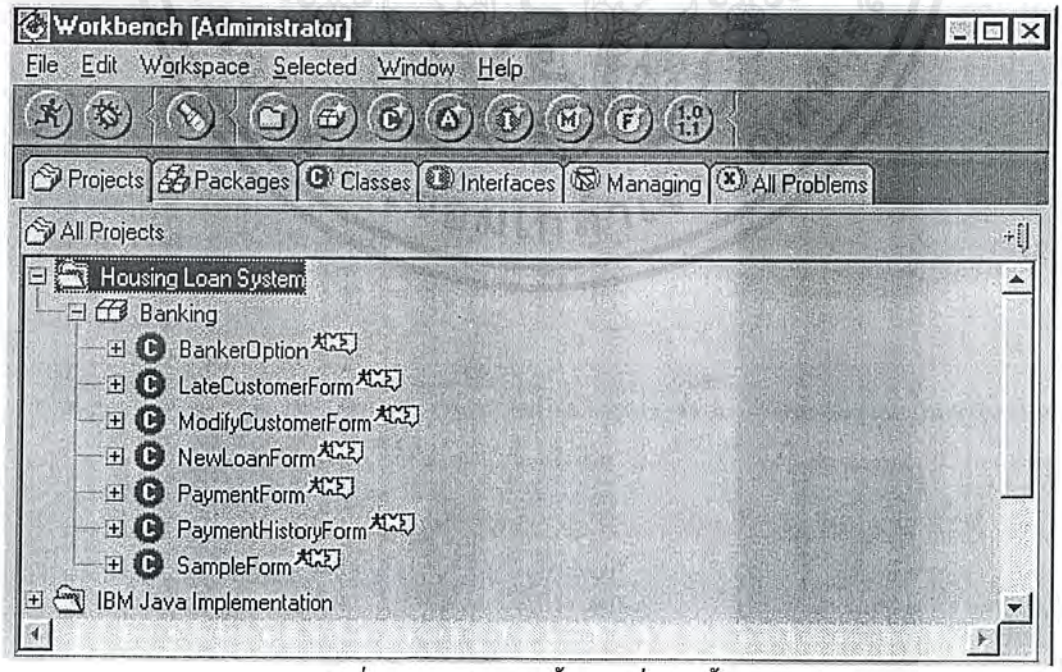
   a. Class BankerOption
   b. Class LateCustomerForm
   c. Class ModifyCustomerForm
   d. Class NewLoanForm
   e. Class PaymentForm
   f. Class PaymentHistoryForm
   g. Class SampleForm

เนื่องจากเรามีจำนวนคลาสไม่มากนัก จึงสามารถรวมคลาสต่างๆเหล่านี้ไว้ในแพ็คเกจเดียวกันได้ โดยให้ชื่อแพ็คเกจเป็น Banking ในการสร้างคลาสต่างๆ เหล่านี้ ทำได้โดยการคลิกเมาส์ปุ่มขวาบนชื่อของ Project Housing Loan System แล้วเลือก Add -> Class... จากนั้นจะขึ้นฟอร์มสำหรับการสร้างคลาสดังนี้



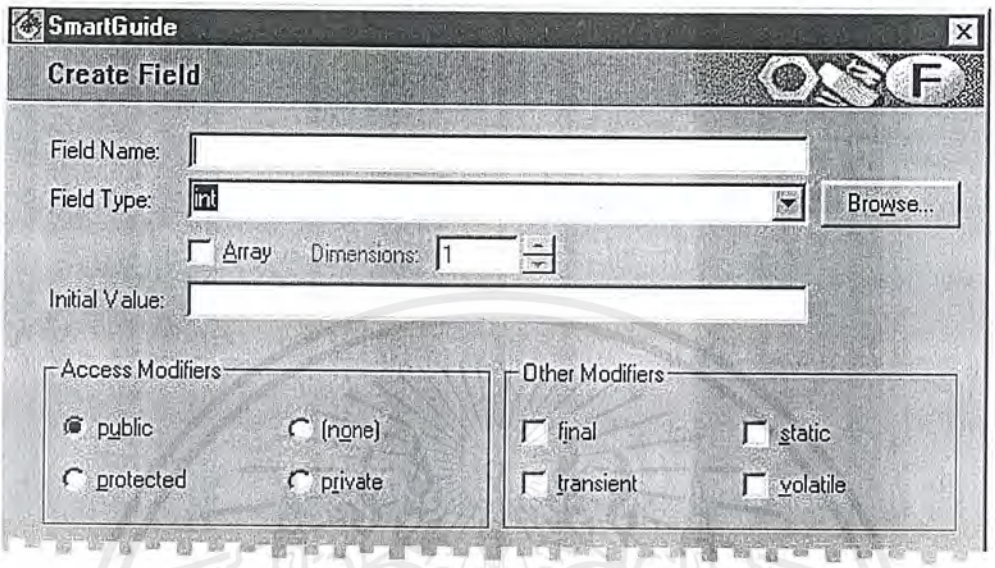
รูปที่ 8-7 แสดงฟอร์มสำหรับการสร้างคลาสขึ้นมาใหม่

ป้อนชื่อแพ็คเกจเป็น Banking แล้วป้อนชื่อ Class name ตามชื่อคลาสที่ต้องการจะสร้าง หากคลาสใดที่จะสร้างต้องการให้มี Visual Component ก็ให้คลิกบอกซ์เครื่องหมายถูกที่ช่อง Compose the class visually จากนั้นจึงคลิกที่ปุ่ม Finish แล้วโปรแกรม VisualAge จะทำการสร้างคลาสและโค้ดในส่วนที่จำเป็นเบื้องต้นให้ เมื่อทำการสร้างคลาสทั้งหมดที่ต้องการแล้วจะเป็นดังรูป 8-8



รูปที่ 8-8 แสดงคลาสทั้งหมดที่สร้างขึ้น

4. ในคลาสสามารถที่จะเพิ่มแอททริบิวต์ซึ่งก็คือตัวแปรที่แสดงคุณลักษณะของคลาส โดยในภาษาจาวาจะเรียกแอททริบิวต์ว่าฟิลด์ (Field) การเพิ่มฟิลด์สามารถทำได้โดยการคลิกเมาส์ปุ่มขวาที่ชื่อคลาสที่ต้องการ แล้วเลือก Add -> Field จากนั้นก็กำหนดชื่อตัวแปรและชนิดของตัวแปร

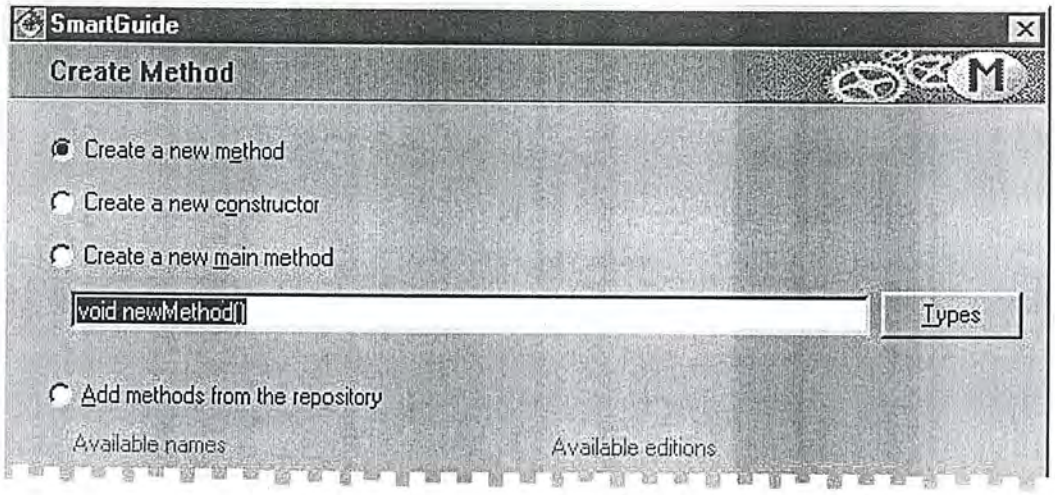


รูปที่ 8-9 แสดงการเพิ่มแอททริบิวต์ขึ้นในคลาส

การสร้างฟิลด์สามารถสร้างได้เป็นแบบอาร์เรย์ โดยคลิกที่ปุ่ม Array และสามารถกำหนดค่าเริ่มต้นให้สำหรับตัวแปรนั้นๆ ได้ในช่อง Initial Value ส่วนการเลือกประเภทการเข้าถึงตัวแปรสามารถเลือกได้ดังนี้

- None ไม่ทำการระบุการเข้าถึง
- Public เป็นการระบุให้ตัวแปรนั้นถูกอ้างได้จากนอกคลาส
- Private เป็นการระบุให้ตัวแปรนั้นไม่สามารถถูกอ้างถึงได้จากนอกคลาส
- Protected เป็นการระบุให้ตัวแปรนั้นถูกอ้างได้จากนอกคลาสภายในแพ็กเกจเดียวกัน หรือคลาสที่ขยายออกมาจากคลาสนั้นที่อยู่ในแพ็กเกจอื่น

5. ในคลาสสามารถที่จะเพิ่ม Operation() ซึ่งก็คือการกระทำของคลาสนั้นๆ โดยในภาษาจาวาจะเรียกโอเปอเรชั่นว่า เมธอด (Method) การเพิ่มเมธอดสามารถทำได้โดยการคลิกเมาส์ปุ่มขวาที่ชื่อคลาสที่ต้องการ แล้วเลือก Add -> Method จากนั้นก็พิมพ์ชื่อของเมธอดตามต้องการ

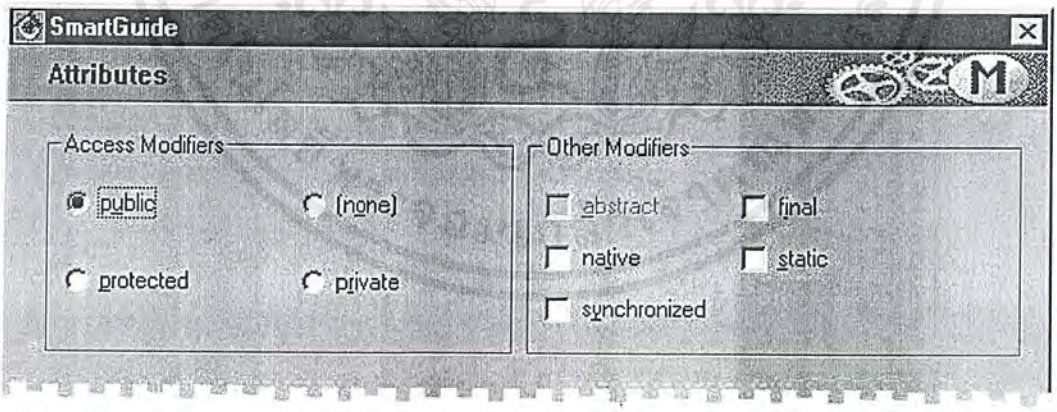


รูปที่ 8-10 แสดงการเพิ่มเมธอดขึ้นใหม่ในคลาส

การสร้างเมธอดขึ้นมาใหม่นั้น สามารถสร้างเมธอดได้ 3 รูปแบบ ดังนี้

- method เป็นเมธอดทั่วๆ ไปของคลาส
- constructor เป็นส่วน constructor ซึ่งจะถูกเรียกใช้ขณะสร้างคลาส
- main method เป็น main ของคลาสซึ่งจะเป็นเมธอดหลักสำหรับการทำงานของโปรแกรม

หรืออาจจะสร้างเมธอดจากคลังข้อมูล (Repository) ซึ่งเป็นการสร้างเมธอดจากเมธอดที่เราได้ทำการลบทิ้งไป หรือพูดอีกนัยหนึ่งก็คือเป็นการกู้เมธอดกลับมาใหม่นั่นเอง จากนั้นก็เลือกคลิกที่ปุ่ม Next

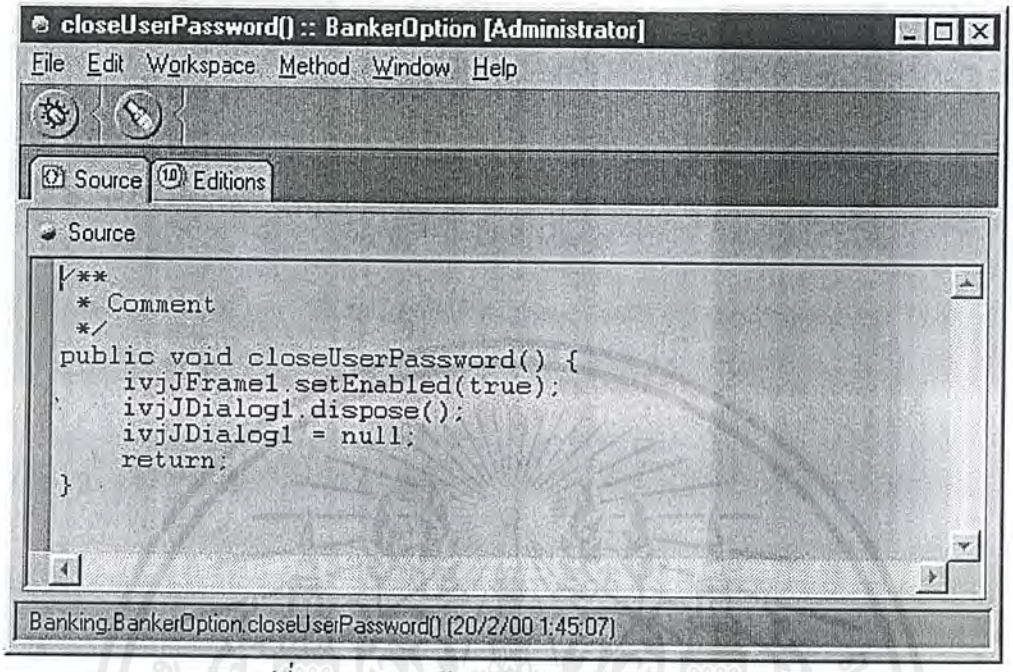


รูปที่ 8-11 เลือกชนิดของเมธอดที่จะทำการสร้าง

เราสามารถกำหนดการเข้าถึงเมธอดได้เหมือนกับการเข้าถึงฟิลด์ ซึ่งมีความหมายเหมือนกัน เมื่อเลือกชนิดของเมธอดที่จะทำการสร้างแล้วก็ให้คลิกที่ปุ่ม Finish

8.3.2 การเขียนโปรแกรม

เราสามารถเปิดหน้าจอใหม่สำหรับการเขียนโค้ด โดยการดับเบิ้ลคลิกที่ชื่อของคลาสที่ต้องการ จากนั้นก็จะขึ้นหน้าต่างใหม่สำหรับแก้ไขส่วนของโปรแกรม



รูปที่ 8-12 แสดงหน้าต่างใหม่สำหรับการเขียนโค้ด

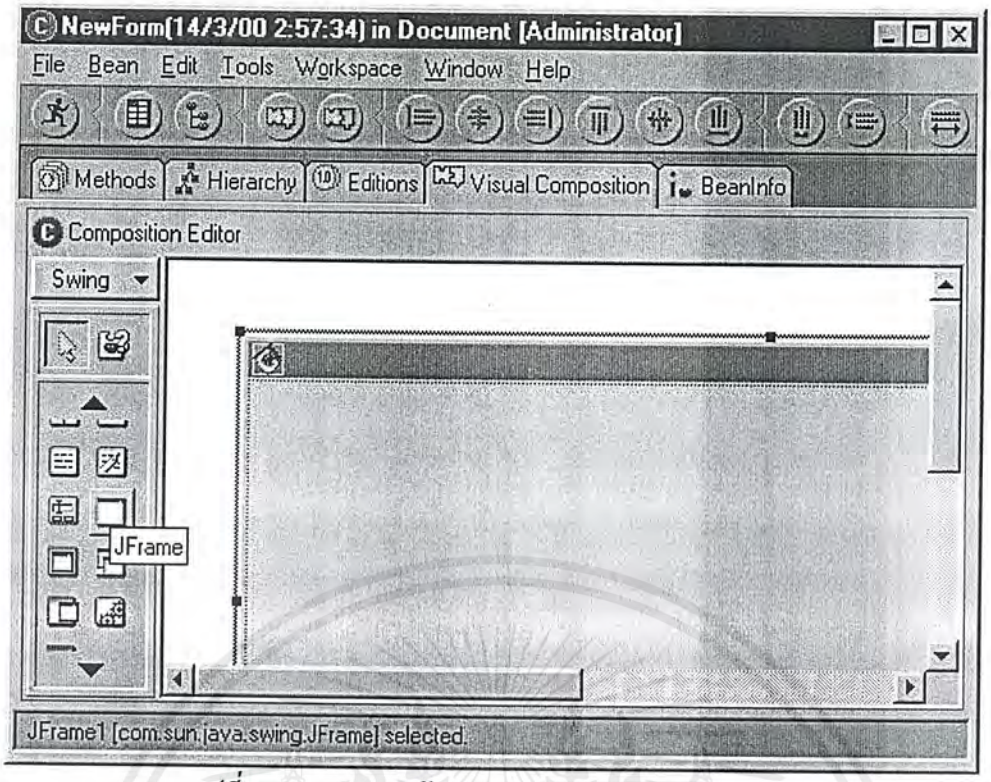
เมื่อได้ทำการเพิ่มหรือแก้ไขโค้ดแล้วให้ทำการบันทึกโดยการเลือกที่เมนู Edit -> Save โดยก่อนที่จะทำการบันทึกนั้น โปรแกรม VisualAge จะทำการตรวจสอบความถูกต้องของโค้ดที่เราเขียนขึ้นก่อน หากเกิดข้อผิดพลาดก็จะมีการแจ้งเตือน และถามยืนยันความต้องการว่ายังต้องการที่จะบันทึกหรือไม่

เราสามารถเลือกดูการแก้ไขโค้ดที่เคยทำมาก่อนได้โดยคลิกที่ Sheet Editions หาก method นี้เคยมีการแก้ไขมาแล้วก็จะขึ้นวันเวลาที่เคยแก้ไข เราสามารถเลือกใช้โค้ดใน Editions ใดก็ได้

เมื่อได้ทำการเขียนโปรแกรมเสร็จแล้ว ก็สามารถสั่งให้โปรแกรม VisualAge ทำการคอมไพล์และรันโปรแกรมของเราได้ โดยคลิกเลือกที่คลาสที่ต้องการ แล้วคลิกที่ปุ่มรัน 

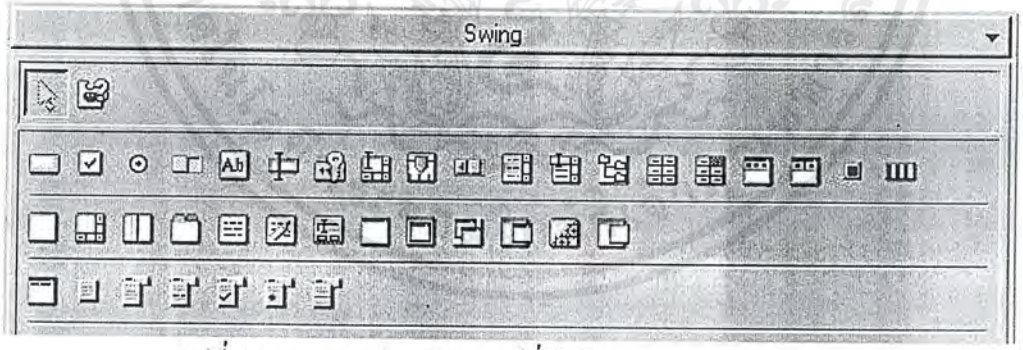
8.3.3 การสร้าง Visual Composition ในโปรแกรม VisualAge

เราสามารถสร้างรูปแบบหน้าตาของแอพพลิเคชันได้โดยใช้ Java Bean ที่มีให้อยู่ใน Visual Composition โดยแบ่งได้เป็น AWT และ Swing สำหรับในโครงงานนี้ได้เลือกใช้ Component Swing เนื่องจากมีหน้าตาที่สวยกว่า Component AWT การสร้างเฟรมนั้น ใน Swing จะมีชื่อเรียกว่า JFrame ซึ่งสามารถสร้างได้จากการลาก Component JFrame ที่อยู่บนแถบเครื่องมือมาวางบนพื้นที่สีขาวดังรูป 8-13



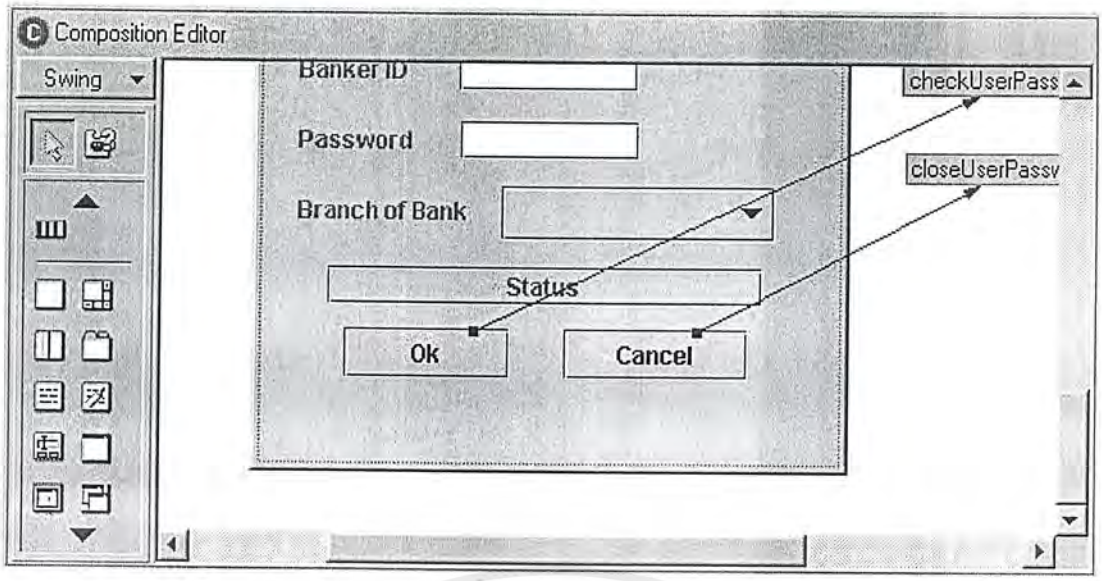
รูปที่ 8-13 แสดงการสร้าง JFrame ใน Visual Composition

สำหรับการสร้างคอมโพเนนท์อื่นๆ ก็มีลักษณะเช่นเดียวกัน แต่เราไม่สามารถนำ Component AWT มาใช้บน Component Swing ได้ ดังนั้นจึงต้องเลือกใช้เฉพาะแต่คอมโพเนนท์ที่มีให้ในกลุ่มของ Component Swing เท่านั้น ซึ่งมีดังรูป 8-14



รูปที่ 8-14 แสดงรูปคอมโพเนนท์ที่มีให้ในชุด Component Swing

เมื่อเราได้สร้างคอมโพเนนท์ที่จำเป็นสำหรับโปรแกรมแล้ว ต่อไปก็เป็นการสร้างการเชื่อมต่อสำหรับแต่ละคอมโพเนนท์ว่าให้คอมโพเนนท์นั้น มีการติดต่อกับเมธอดใดๆในคลาสหรืออาจจะติดต่อกับเมธอดใดๆของคอมโพเนนท์อื่นๆก็ได้ การสร้างการเชื่อมต่อสามารถทำได้โดยการคลิกเมาส์ปุ่มขวาที่คอมโพเนนท์ที่ต้องการ แล้วเลือกไปที่ Event to Code สำหรับการติดต่อไปยัง method ใดๆ ที่เราทำการสร้างไว้ หรือเลือกไปที่ Connect สำหรับการติดต่อระหว่างเมธอดกับเมธอดของแต่ละคอมโพเนนท์



รูปที่ 8-15 แสดงการสร้างการเชื่อมต่อจากคอมโพเนนท์ไปยังโค้ด

เมื่อเราได้ทำการสร้างส่วนของ Visual Composition เสร็จแล้ว ก็ให้ทำการบันทึก โดยเลือกที่ Bean -> Save Bean จากนั้นโปรแกรม VisualAge ก็จะทำการสร้างโค้ดและตรวจสอบโค้ดที่ได้แล้วจึงบันทึก

8.4 การเขียนโปรแกรมติดต่อกับฐานข้อมูล โดยใช้ Oracle 8.0.5 เป็น DBMS

ในโครงงานนี้ได้ติดตั้ง Oracle 8.0.5 ไว้บนเครื่องที่เป็นเซิฟเวอร์ฐานข้อมูลและได้ลงโปรแกรม VisualAge บนเครื่องที่เป็นไคลเอนท์ สำหรับสร้างโปรแกรมประยุกต์ภาษาจาวาให้ติดต่อกับฐานข้อมูลออราเคิลจำเป็นต้องใช้ Oracle JDBC Driver ซึ่งได้เลือก JDBC Thin Driver สำหรับการติดต่อกับฐานข้อมูล ขั้นตอนการติดตั้ง JDBC Driver ได้อธิบายไปแล้วในหัวข้อ 8.2 ในส่วนนี้เราจะอธิบายถึงการเรียกใช้ฟังก์ชันหรือการใช้คำสั่ง SQL จากโปรแกรมประยุกต์ภาษาจาวา

8.4.1 การเรียกใช้คำสั่ง SQL

เราสามารถเรียกใช้คำสั่ง SQL ได้โดยการสร้าง Statement ซึ่งจะมีลักษณะการเขียนโปรแกรมดังนี้

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
conn = DriverManager.getConnection ("jdbc:oracle:thin:@161.246.6.117:1521:bk",
                                    "dear", "dear");
Statement stmt = conn.createStatement ();
ResultSet rset = stmt.executeQuery ("select Banker_Name from Banker where Banker_id = " +
                                    bankerID);
if (rset.next ()) { bankerName = rset.getString(1); }
```

จากโปรแกรมตัวอย่างข้างต้นเป็นการใช้คำสั่ง SQL ในการ Select Banker_Name จากตาราง Banker โดยเลือกเอาเฉพาะแถวที่มี Banker_id ตรงกับค่าที่อยู่ในตัวแปร bankerID แล้วผลของการ Select ที่ได้จะถูกเก็บอยู่ในตัวแปร rset เราสามารถดึงผลลัพธ์ได้โดยใช้ method rset.getString(n) โดย n คือ ผลลัพธ์คอลัมน์ n

8.4.2 การเรียกใช้ฟังก์ชันหรือโพรซีเยอร์ที่เก็บไว้ในออราเคิลเซิฟเวอร์

การเรียกใช้สโตร์โพรซีเยอร์จะต้องทำการสร้าง CallableStatement ซึ่งจะเป็นการระบุถึงการเรียกใช้ฟังก์ชันหรือโพรซีเยอร์ใดๆในฐานข้อมูล โดยมีลักษณะการเขียนโปรแกรมดังนี้

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
conn = DriverManager.getConnection ("jdbc:oracle:thin:@161.246.6.117:1521:bk",
                                    "dear", "dear");
CallableStatement cstmt = conn.prepareCall ("{? = call CHECK_USER (?,?,?,?)}");
cstmt.registerOutParameter (1, Types.CHAR);
cstmt.setInt (2, bankerID);
cstmt.setString (3, password);                    // not upper font //
cstmt.setString (4, bankerStatus);
cstmt.setString (5, branchName);
cstmt.execute ();
passwordOk = cstmt.getString(1).trim();
```

จากโปรแกรมตัวอย่างข้างต้น เป็นการเรียกใช้ฟังก์ชัน CHECK_USER โดยจะมีการคืนค่ากลับมาเป็น CHAR และมีการส่งค่าตัวแปรไปยังฟังก์ชัน ได้แก่ bankerID, password, bankerStatus, branchName เสร็จแล้วจึงทำการเอ็กซีคิวต์ เมื่อฟังก์ชันทำงานเสร็จ เราก็จะสามารถอ่านค่าตัวแปรที่คืนกลับมาได้ด้วยเมธอด cstmt.getString(1)

ในการเรียกใช้โพรซีเยอร์ก็จะมีลักษณะเช่นเดียวกับการเรียกใช้ฟังก์ชัน แต่ว่าการเรียกใช้โพรซีเยอร์จะไม่มีการคืนค่ากลับมาเท่านั้นเอง

8.5 การนำเข้า (Import) และส่งออก (Export) ไฟล์ในโปรแกรม VisualAge

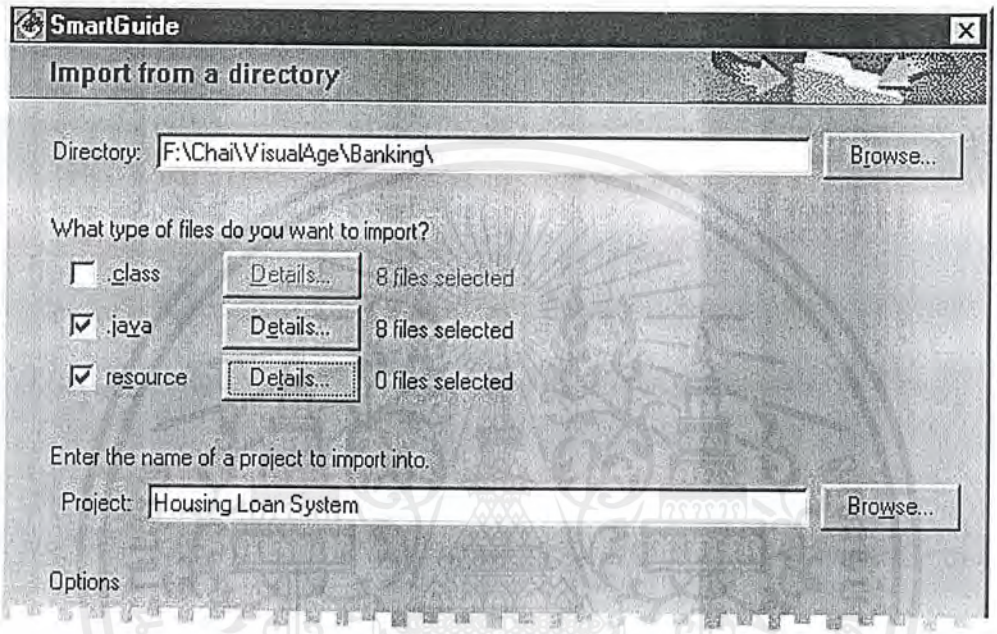
เราสามารถที่จะนำเข้าและส่งออกไฟล์ที่มีนามสกุล .class หรือ .java หรืออาจจะเป็นไฟล์ที่มีการบีบอัดเช่น .jar และ .zip โดยใน VisualAge จะแบ่งประเภทการนำเข้าและส่งออกเป็น 3 ประเภทคือ

- Directory เป็นการเลือกการนำเข้าและส่งออกแบบเป็นไดเรอทอรีโดยไฟล์ดังกล่าวจะเป็นไฟล์ .class หรือ .java

- Jar file — เป็นการเลือกการนำเข้าและส่งออกไฟล์ที่เป็นไฟล์ประเภทถูกบีบอัด ซึ่งได้แก่ไฟล์ที่มีนามสกุล .jar หรือ .zip
- Repository — เป็นการเลือกการนำเข้าและส่งออกไฟล์ที่เก็บไว้เป็นคลังข้อมูลของโปรแกรม

ขั้นตอนการนำเข้าไฟล์เข้าไปในโปรแกรม VisualAge แบบไดเรกทอรี

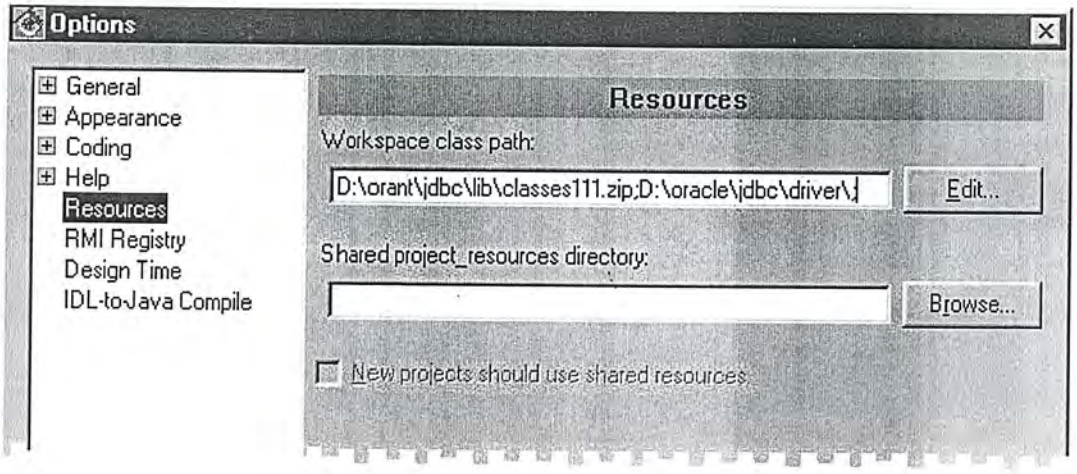
1. เลือก Import -> Directory จาก Workbench ของ VisualAge



รูปที่ 8-16 แสดงการนำเข้าไฟล์ แบบ Directory

พิมพ์ชื่อไดเรกทอรีที่เก็บไฟล์ที่ต้องการจะนำเข้า โดยสามารถเลือกชนิดของไฟล์ที่จะนำเข้าได้เป็น .class หรือ .java จากนั้นก็ให้พิมพ์ชื่อของ Project ที่จะต้องการจะเก็บไฟล์ที่เราได้นำเข้ามา

2 จากนั้นก็ให้คลิกที่ปุ่ม Finish แล้วโปรแกรม VisualAge ก็จะทำการนำเข้าไฟล์ที่เราได้เลือกไว้มาเก็บไว้ใน Project ตามชื่อที่เราได้ระบุไว้ บางครั้งหากไฟล์ที่เราได้นำเข้ามามีการเรียกใช้คลาสอื่นๆ เราจำเป็นต้องระบุ CLASSPATH ให้แก่โปรแกรม โดยระบุที่ Workbench ของ VisualAge ตรงเมนู Window -> Options... แล้วเลือกที่ Resources ดังรูป 8-17

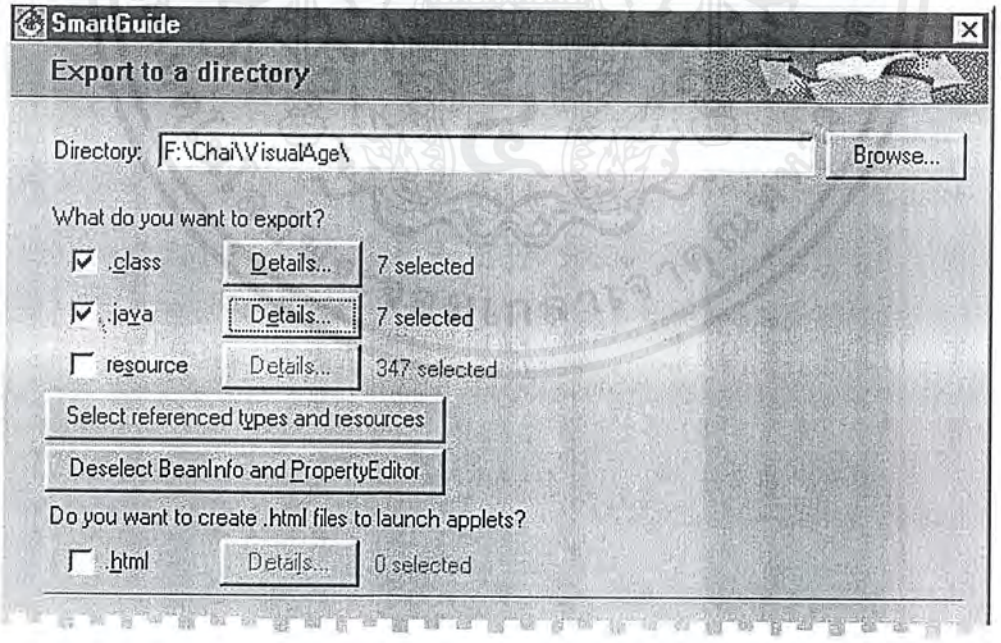


รูปที่ 8-17 แสดงการกำหนด Class Path ให้กับโปรแกรม

ทำการแก้ไข Class Path โดยสามารถอ้าง Class Path ได้แบบไดเรกทอรีหรืออ้างจากไฟล์ที่มีการบีบอัด เช่น ไฟล์ .jar หรือ .zip จากนั้นจึงคลิกที่ปุ่ม OK เพื่อเป็นการกำหนด Class Path ใหม่

ขั้นตอนการส่งออกไฟล์จากโปรแกรม VisualAge แบบไดเรกทอรี

สำหรับการส่งออกไฟล์สามารถทำได้ในลักษณะเดียวกับการนำเข้าไฟล์ โดยการเลือก Export -> Directory จาก Workbench ของ VisualAge



รูปที่ 8-18 แสดงการส่งออกไฟล์ แบบ Directory

เลือกไดเรกทอรีที่ต้องการส่งออกไฟล์และเลือกชนิดของไฟล์ที่ต้องการจะส่งออกไป โดยสามารถกำหนดให้เป็นไฟล์ที่มีนามสกุล .class ซึ่งก็คือไบนารี่โค้ต หรือให้เป็นไฟล์นามสกุล .java เราสามารถเลือกคลาสที่ต้องการจะส่งออกได้โดยการคลิกที่ปุ่ม Details และคลิกเลือกที่ชื่อคลาสที่ต้องการ เราสามารถเลือกส่งออกคลาสที่ถูกอ้างถึงได้โดยการคลิกที่ปุ่ม Select Referenced types and resources แล้วโปรแกรม VisualAge จะทำการค้นหาและเลือกคลาสที่ถูกอ้างอิงมาทำการส่งออกไปด้วย จากนั้นจึงคลิกที่ปุ่ม Finish เพื่อเริ่มทำการส่งออกไฟล์

ไฟล์ที่ได้จากการส่งออกจะถูกแยกไปเป็นไดเรกทอรีตามชื่อของแพ็คเกจที่ทำการเก็บคลาสนั้นๆ ไว้ โดยไฟล์ต่างๆนั้นจะมีชื่อตามชื่อคลาสที่เราได้สร้างไว้ เราสามารถนำไฟล์ .class และ .java ที่ได้สร้างไว้นี้ไปทำงานต่อ หรืออาจจะเก็บไว้เป็นไฟล์สำรองก็ได้

บทที่ 9
บทสรุปโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

9.1 ลักษณะของระบบ

จากประโยคปัญหาของระบบ (Problem Statement) ที่ได้กำหนดให้ระบบนี้เป็นระบบสำหรับการขอกู้เงินเพื่อการผ่อนชำระหมู่บ้านจัดสรร โดยระบบนี้จัดทำขึ้นเพื่อการใช้งานสำหรับธนาคาร ซึ่งธนาคารจะประกอบด้วยสำนักงานใหญ่และสาขา มีการติดต่อระหว่างสาขาด้วยเครือข่ายที่เป็นระบบ WAN

ในการเก็บข้อมูลลูกค้าที่มาทำการขอกู้เงินกับทางธนาคาร จะจัดเก็บเป็นฐานข้อมูลส่วนกลางอยู่ที่สำนักงานใหญ่ ซึ่งจะใช้เครื่องเซิฟเวอร์ตั้งเป็นเครื่องให้บริการฐานข้อมูล (Database Server) โดยใช้ระบบจัดการฐานข้อมูลแบบรีเลชันแนลของออราเคิล (Oracle) เป็นตัวบริหารฐานข้อมูล ให้บริการแก่สาขาต่างๆ ในการสืบค้นและการบันทึกข้อมูลลูกค้าที่มาติดต่อกับสาขานั้นๆ

สำหรับในแต่ละสาขาจะมีการลงโปรแกรมระบบเงินกู้ เพื่อใช้ในการทำงานและติดต่อกับฐานข้อมูลส่วนกลางที่สำนักงานใหญ่ โดยเครื่องที่ได้ทำการติดตั้งโปรแกรมนี้จะเรียกว่าเครื่องสำหรับการใช้งานลูกข่าย (Client) ก่อนการเข้าใช้โปรแกรมและทำการติดต่อระหว่างโปรแกรมที่สาขากับระบบจัดการฐานข้อมูลส่วนกลางที่สำนักงานใหญ่ จะมีการตรวจสอบรหัสเจ้าหน้าที่ธนาคารผู้ใช้งานซึ่งแบ่งเป็นเจ้าหน้าที่ธนาคารผู้ใช้งานทั่วไป (User) กับเจ้าหน้าที่ธนาคารผู้มีสิทธิ์ในการแก้ไขข้อมูล (Admin) โดยข้อมูลเจ้าหน้าที่ธนาคารและรหัสผ่านจะเก็บอยู่ในฐานข้อมูลส่วนกลางที่สำนักงานใหญ่เท่านั้น ผู้ใช้งานจึงจำเป็นต้องติดต่อผ่านสำนักงานใหญ่เสมอ ลักษณะของระบบจึงเป็นลักษณะการทำงานแบบรวมศูนย์กลาง (Centralized)

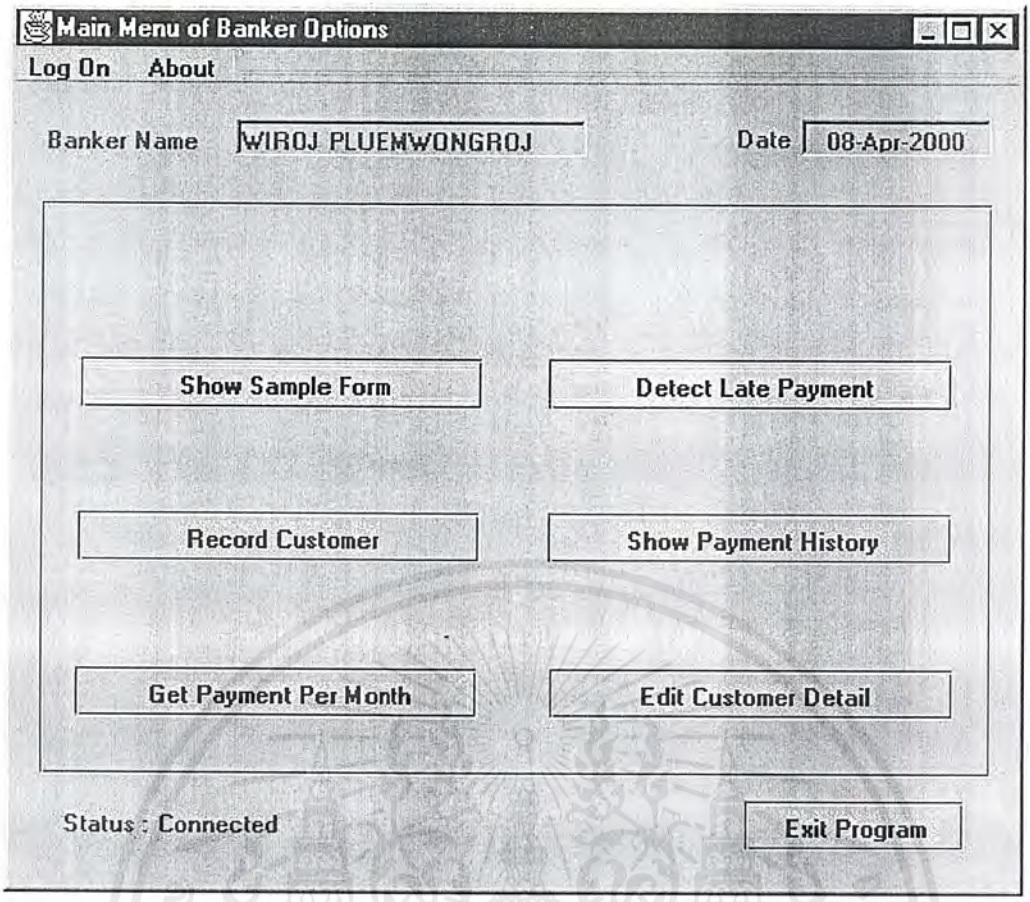
9.2 ลักษณะของโปรแกรมและการใช้งาน

โปรแกรมระบบเงินกู้ เป็นโปรแกรมที่ใช้สำหรับการขอกู้เงินและรับการชำระเงินรายเดือนจากลูกค้าของธนาคาร โดยมีฟังก์ชันการทำงานต่างๆตามที่ประโยคปัญหาได้กำหนดไว้ ลักษณะของโปรแกรมเป็นโปรแกรมประยุกต์ที่มีการร้องขอข้อมูลและการทำงานบางอย่างจากคอมพิวเตอร์ส่วนกลาง โดยมีการใช้ JDBC Driver สำหรับการติดต่อกับระบบจัดการฐานข้อมูลออราเคิลดังที่ได้กล่าวไว้ในบทที่ 8

ลักษณะหน้าตาของโปรแกรมจะมีรูปแบบใกล้เคียงกับต้นแบบส่วนติดต่อผู้ใช้ที่ได้ออกแบบไว้ โดยผลลัพธ์ของโปรแกรมที่ได้สร้างขึ้น จะมีรูปแบบของโปรแกรมและการใช้งานต่างๆ ดังนี้

9.2.1 เริ่มต้นการใช้งานโปรแกรม

ผู้ใช้งานโปรแกรมสามารถเปิดโปรแกรมระบบเงินกู้นี้ได้ โดยโปรแกรมนี้สามารถใช้งานได้บนเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีการติดต่อผ่านเครือข่าย LAN และ WAN ไปยังเครื่องที่เป็นเครื่องให้บริการฐานข้อมูลส่วนกลาง ลักษณะของโปรแกรมที่ได้เป็นดังรูปที่ 9-1

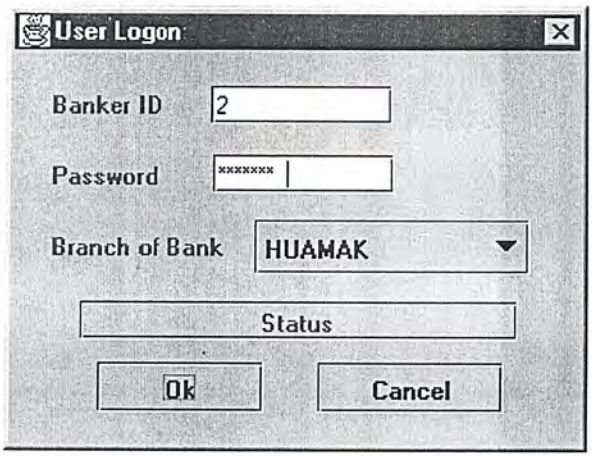


รูปที่ 9-1 แสดงลักษณะเมนูหลักของโปรแกรม

เมื่อเริ่มเปิดโปรแกรมระบบเงินกู้นี้ โปรแกรมจะแสดงเมนูหลักและวันที่ปัจจุบันที่เรียกใช้งาน โดยในเมนูหลักนี้จะแสดง ถึงฟังก์ชันก ารทำงานของเจ้าหน้าที่ธนาคารผู้ใช้งานทั่วไปสำหรับระบบการกู้เงินผ่อนบ้าน ผู้ใช้จะยังไม่สามารถเรียกใช้ฟังก์ชันต่างๆได้หากไม่ได้ทำการป้อนรหัสผ่านเพื่อเข้าสู่ระบบ (logon) และผู้ใช้สามารถออกจากโปรแกรมนี้ได้โดยการคลิกที่ปุ่ม Exit Program

9.2.2 การป้อนรหัสผ่านเพื่อเข้าสู่ระบบของผู้ใช้งานโปรแกรม

ผู้ใช้งานโปรแกรมนี้ แบ่งได้เป็น 2 ประเภท คือ เจ้าหน้าที่ธนาคารผู้ใช้งานระบบ (User) และเจ้าหน้าที่ธนาคารผู้ดูแลข้อมูล (Admin) โดยผู้ใช้งานโปรแกรมทั้งสองประเภทจะต้องทำการป้อนรหัสผ่านเพื่อเข้าสู่ระบบ แล้วโปรแกรมจะทำการตรวจสอบรหัสประจำตัวผู้ใช้ (User ID) กับรหัสผ่าน (Password) ว่าตรงกับข้อมูลที่เก็บไว้ในฐานข้อมูลหรือไม่ ลักษณะโปรแกรมในการป้อนรหัสผ่านเป็นดังรูปที่ 9-2

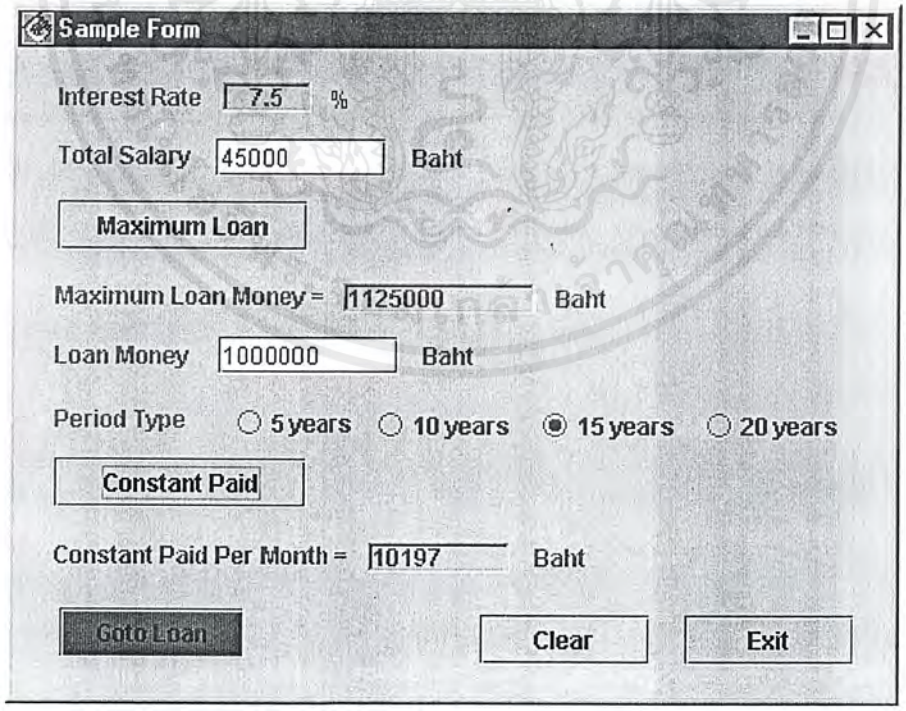


รูปที่ 9-2 แสดงลักษณะของโปรแกรมในการป้อนรหัสผ่านเพื่อเข้าสู่ระบบ

ผู้ใช้งานโปรแกรมจะต้องป้อนรหัสประจำตัวผู้ใช้,รหัสผ่านและเลือกสาขาที่ผู้ใช้ดูแลอยู่ จากนั้นจึงคลิกที่ปุ่ม Ok เพื่อเข้าสู่ระบบ หากผู้ใช้ป้อนรหัสผิดพลาด โปรแกรมจะขึ้นข้อความแจ้งความผิดพลาดว่า "Invalid Username/Password" แล้วให้ผู้ใช้ป้อนรหัสใหม่อีกครั้ง

เมื่อผู้ใช้ป้อนรหัสผ่านเพื่อเข้าสู่ระบบถูกต้องเรียบร้อยแล้ว โปรแกรมจะกลับไปยังเมนูหลักและแสดงชื่อผู้ใช้ เพื่อรอรับการใช้งานโปรแกรมต่อไป

9.2.3 การแสดงตัวอย่างการคำนวณวงเงินกู้ และเงินชำระต่องวด



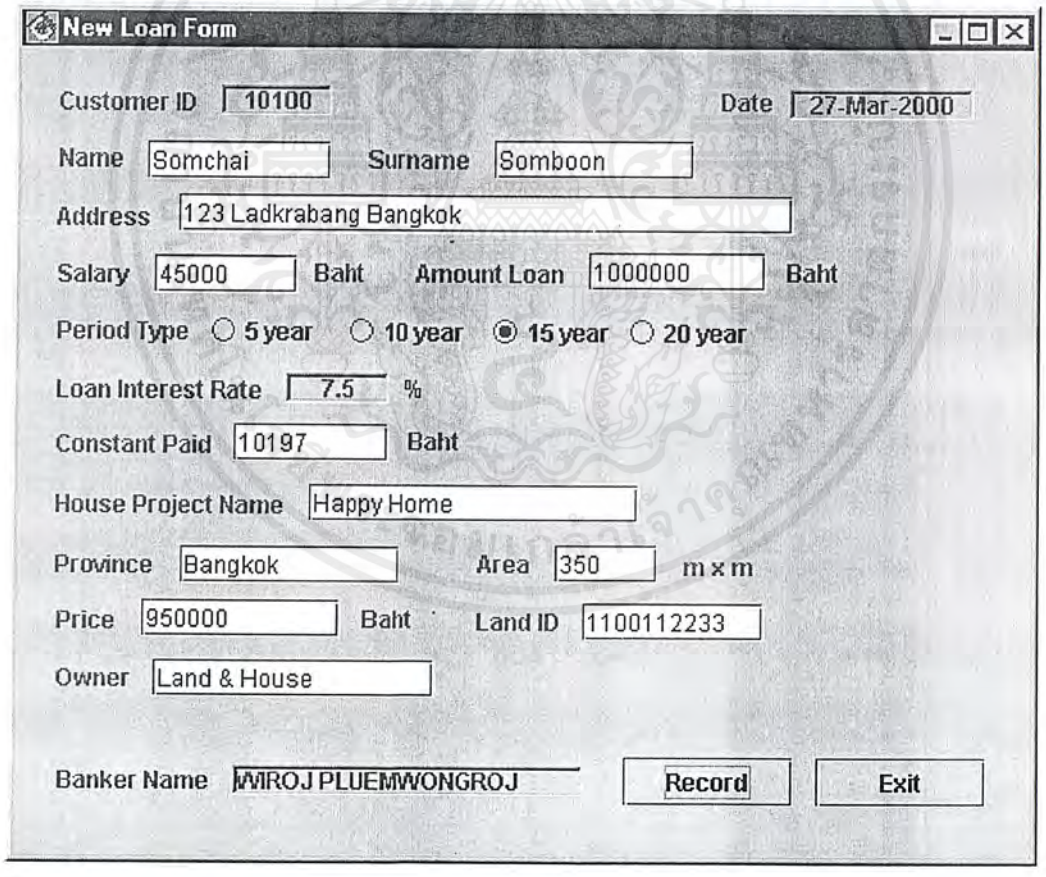
รูปที่ 9-3 แสดงแบบฟอร์มตัวอย่างการคำนวณวงเงินกู้ และเงินชำระต่องวด

จากรูปเป็นการแสดงตัวอย่างการคำนวณเงินสำหรับลูกค้าที่มาขอกู้เงินกับทางธนาคาร โดยจะแสดงการคำนวณวงเงินกู้สูงสุดและเงินชำระคงที่ต่องวด เพื่อให้ลูกค้าได้ใช้ในการตัดสินใจเลือกประเภทระยะเวลาการผ่อนชำระและจำนวนเงินกู้

หากลูกค้าตัดสินใจทำการขอกู้เงิน สามารถเลือกไปทำการขอกู้เงินได้โดยการคลิกที่ปุ่ม Goto Loan แล้วโปรแกรมจะส่งค่าตัวแปรต่างๆ ไปยังแบบฟอร์มการของกู้เงิน แต่หากลูกค้ายังไม่สามารถตัดสินใจได้ก็สามารถลบค่าต่างๆในแบบฟอร์มโดยการคลิกที่ปุ่ม Clear หรือออกจากฟอร์มนี้โดยการคลิกที่ปุ่ม Exit

9.2.4 การบันทึกรายละเอียดลูกค้าที่มาขอกู้เงินผ่อนบ้าน

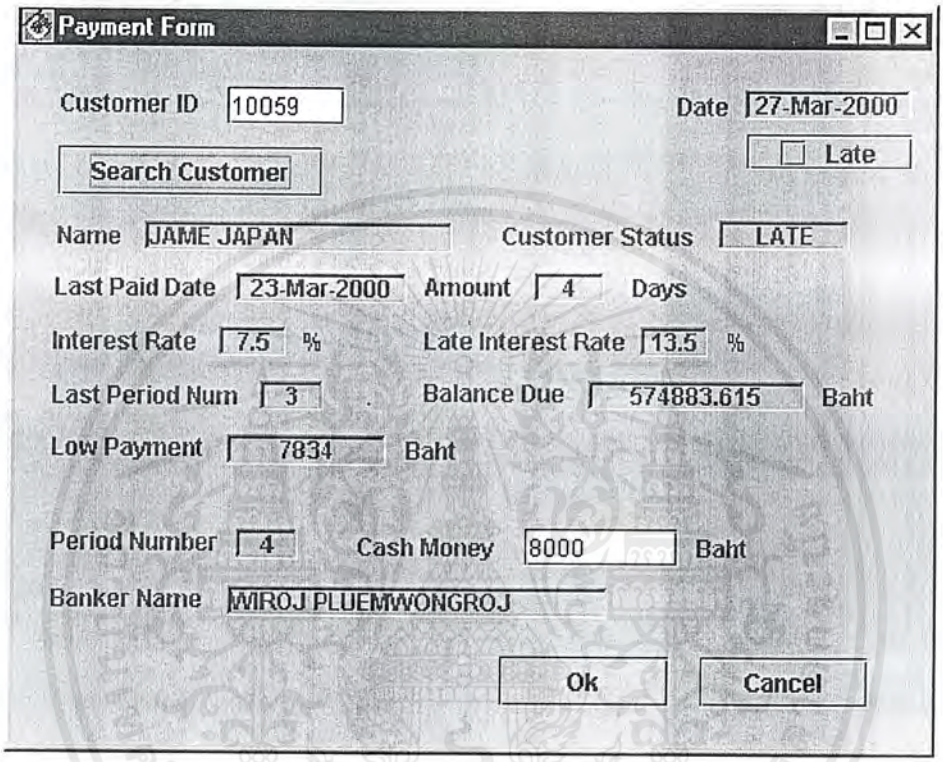
เมื่อลูกค้าต้องการขอกู้เงินกับทางธนาคาร โปรแกรมจะทำการบันทึกรายละเอียดต่างๆ ซึ่งได้แก่ ประวัติส่วนตัวลูกค้า, รายการบัญชีเงินกู้และรายละเอียดโครงการบ้าน โดยจะเก็บข้อมูลต่างๆ เหล่านี้ลงในฐานข้อมูลส่วนกลางและจะบันทึกเจ้าหน้าที่ธนาคารผู้ทำการอนุมัติรายการขอกู้นี้ด้วย เมื่อโปรแกรมทำการบันทึกข้อมูลต่างๆ เสร็จแล้ว จะแสดงรหัสลูกค้า (Customer ID) สำหรับลูกค้า เพื่อใช้ในการติดต่อกับทางธนาคารครั้งต่อไป



รูปที่ 9-4 แสดงแบบฟอร์มการบันทึกรายการขอกู้เงินจากลูกค้า

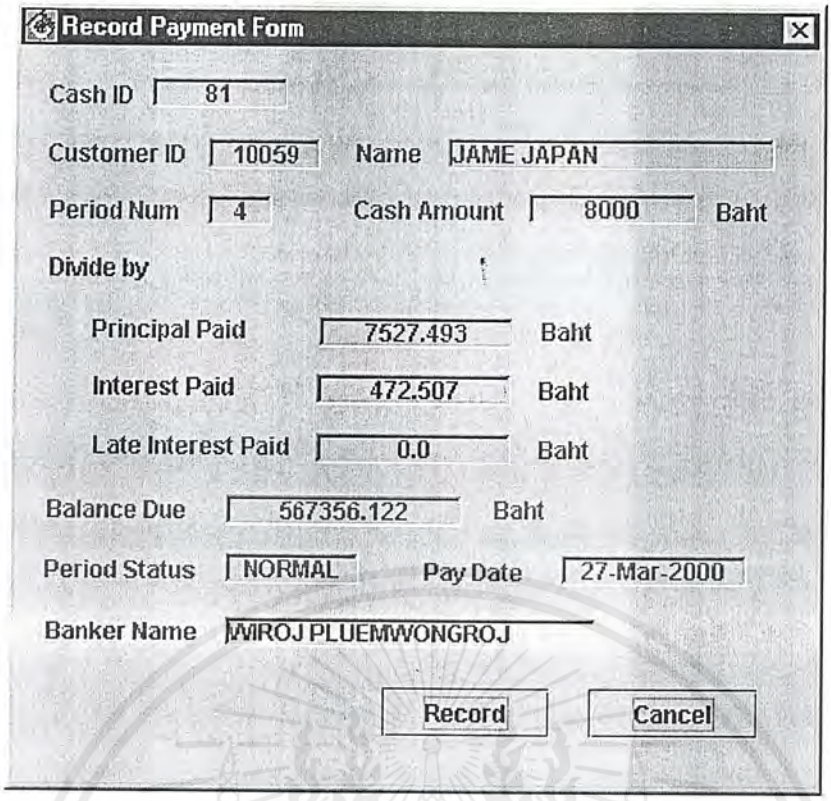
9.2.5 รับการชำระเงินแต่ละงวดจากลูกค้า

ลูกค้าที่มาขอกู้เงินกับทางธนาคารจะต้องมาชำระเงินแต่ละงวดก่อนสิ้นเดือนด้วยเงินสด โดยเจ้าหน้าที่ธนาคารจะทำการป้อนรหัสลูกค้า (Customer ID) เพื่อสืบค้นรายละเอียดลูกค้าและจำนวนเงินขั้นต่ำที่ต้องชำระ โดยในแบบฟอร์มรับการชำระเงินนี้จะแสดงวันที่ปัจจุบัน อัตราดอกเบี้ยปกติ อัตราดอกเบี้ยค่าปรับ และชื่อของนายธนาคารผู้ใช้งานโปรแกรม



รูปที่ 9-5 แสดงแบบฟอร์มรับการชำระเงินรายเดือน

จำนวนเงินชำระที่ลูกค้านำมาชำระนั้น จะต้องมากกว่าหรือเท่ากับจำนวนเงินขั้นต่ำที่คำนวณได้ โดยจะแบ่งจำนวนเงินชำระนี้ออกเป็นส่วนของเงินต้น,ส่วนของเงินดอกเบี้ยและส่วนของเงินดอกเบี้ยค่าปรับ จากนั้นเมื่อลูกค้าตกลงชำระเงิน โปรแกรมจะแสดงรายการจำนวนเงินชำระต่างๆให้ลูกค้าดูเพื่อยืนยันก่อนการบันทึกลงในฐานข้อมูล

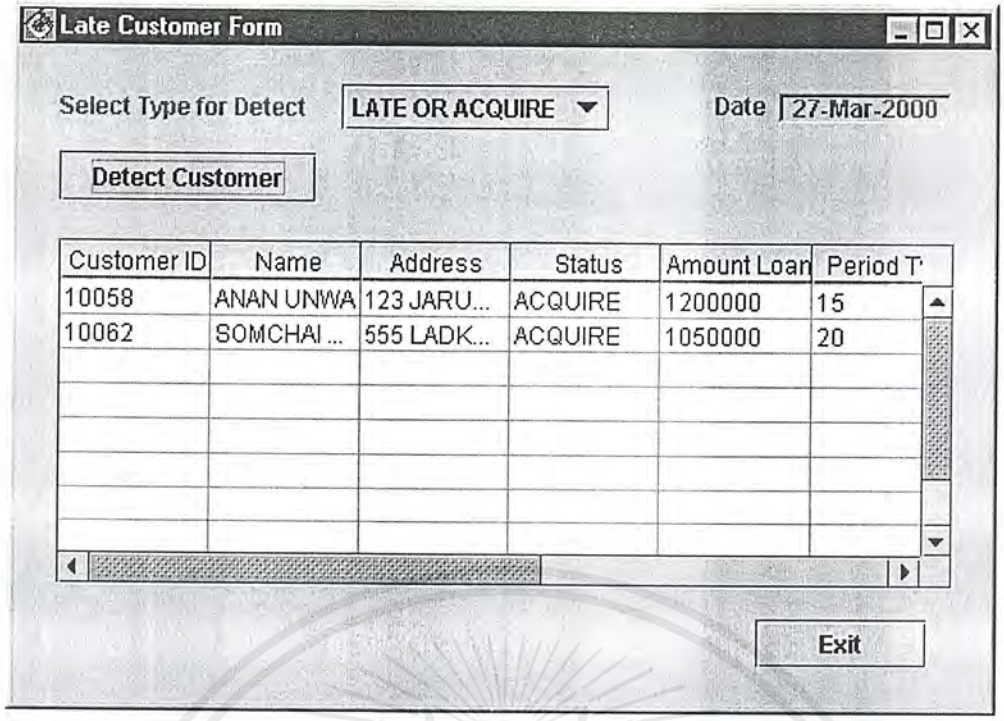


รูปที่ 9-6 แสดงรายการชำระเงินสำหรับลูกค้า เพื่อยืนยันการชำระ

เจ้าหน้าที่ธนาคารสามารถยกเลิกการชำระได้โดยการคลิกที่ปุ่ม Cancel เพื่อกลับไปยังแบบฟอร์มรับการชำระเงินใหม่หรือทำการบันทึกรายการชำระเงินประจำงวดนี้โดยการคลิกที่ปุ่ม Record จากนั้นโปรแกรมจะทำการบันทึกรายการชำระเงินและแสดงรหัสการชำระ (Cash ID) ประจำงวดนี้ให้แก่ลูกค้า

9.2.6 ตรวจสอบรายชื่อลูกค้าที่ชำระเงินล่าช้า

เจ้าหน้าที่ธนาคารสามารถตรวจสอบลูกค้าที่มีสถานะการชำระเงินเป็น "ล่าช้า (LATE)" หรือ "ถูกยึด (ACQUIRE)" ได้ โดยสถานะเริ่มต้นของลูกค้าจะเป็น "ปกติ (NORMAL)" แต่หากลูกค้ามาทำการชำระเงินประจำงวดล่าช้า สถานะของลูกค้าก็จะเปลี่ยนไปเป็น "ล่าช้า" และถ้าลูกค้ามาชำระเงินล่าช้าเกิน 90 วัน ก็จะมีสถานะเป็น "ถูกยึด" ซึ่งแสดงว่าลูกค้าคนนั้นจะไม่สามารถมาติดต่อชำระเงินกับทางธนาคารได้อีกและบ้านก็จะตกเป็นของธนาคาร

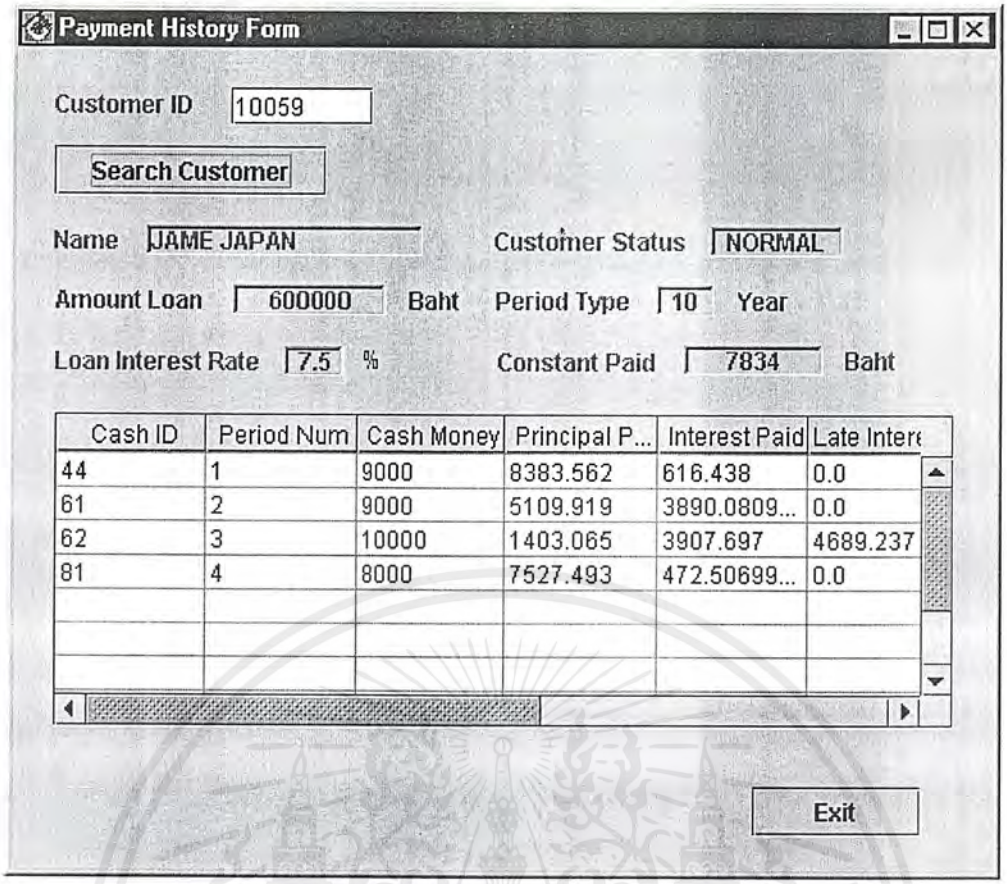


รูปที่ 9-7 แสดงลักษณะโปรแกรมการตรวจสอบลูกค้าที่สถานะการชำระเงินล่าช้า

เจ้าหน้าที่ธนาคารสามารถเลือกรูปแบบการตรวจสอบได้ 3 แบบ คือ ให้แสดงรายชื่อลูกค้าที่มีสถานะล่าช้าอย่างเดียว,แสดงรายชื่อลูกค้าที่มีสถานะถูกยึดหรือแสดงรายชื่อลูกค้าที่มีสถานะล่าช้าหรือถูกยึด การตรวจสอบรายชื่อลูกค้าที่มีสถานะล่าช้านี้ จะใช้ในการตรวจสอบและติดตามลูกค้าของธนาคาร เช่น อาจจะมีการส่งจดหมายเตือนไปยังลูกค้าหรือการติดต่อทางโทรศัพท์เพื่อแจ้งให้ลูกค้าทราบ

9.2.7 แสดงประวัติการชำระเงินของลูกค้า

เจ้าหน้าที่ธนาคารสามารถดูประวัติการชำระเงินของลูกค้าได้ โดยโปรแกรมจะแสดงรายการชำระเงินประจำงวดต่างๆที่ผ่านมาของลูกค้า และแสดงรายการบัญชีเงินกู้ของลูกค้าคนนั้นด้วย ผู้ใช้จะต้องป้อนรหัสลูกค้า (Customer ID) เพื่อสืบหาข้อมูลในฐานข้อมูล จากนั้นจึงคลิกที่ปุ่ม Search Customer เพื่อเริ่มการสืบค้นข้อมูลและผู้ใช้สามารถออกจากการแสดงประวัติการชำระเงินโดยการคลิกที่ปุ่ม Exit

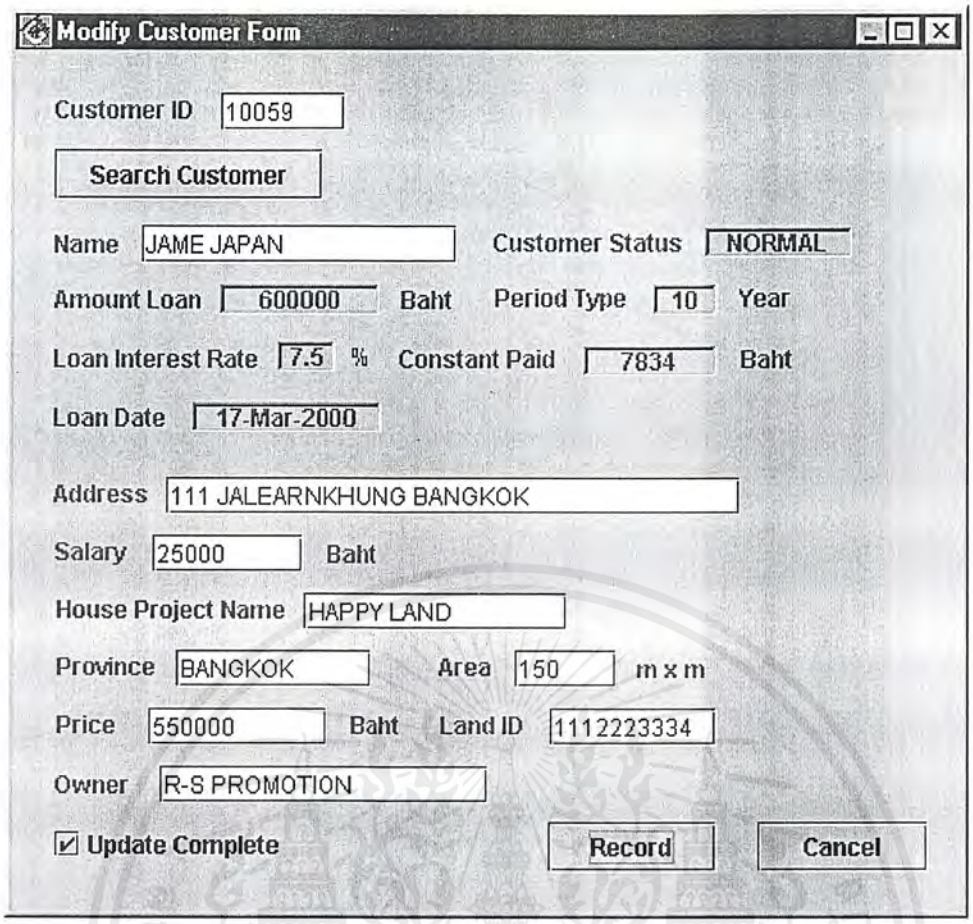


รูปที่ 9-8 แสดงลักษณะของโปรแกรมสำหรับแสดงประวัติการชำระเงินของลูกค้า

9.2.8 แก้ไขประวัติส่วนตัวของลูกค้า

ลูกค้าสามารถมาแจ้งการแก้ไขข้อมูลประวัติส่วนตัวกับเจ้าหน้าที่ธนาคารได้ โดยเจ้าหน้าที่ธนาคารจะทำการตรวจสอบรหัสลูกค้าและแสดงข้อมูลประวัติส่วนตัวที่สามารถแก้ไขได้แล้วทำการบันทึกข้อมูลที่ต้องการแก้ไขใหม่ ข้อมูลที่สามารถทำการแก้ไขได้ ได้แก่ข้อมูลส่วนตัวและข้อมูลรายละเอียดโครงการบ้าน

ลูกค้าที่ต้องการจะแก้ไขข้อมูลประวัติส่วนตัวจะต้องมาแจ้งรหัสประจำตัวลูกค้าแก่เจ้าหน้าที่ธนาคาร จากนั้นเจ้าหน้าที่ธนาคารจะทำการตรวจสอบรายการของลูกค้าโดยการคลิกที่ปุ่ม Search Customer แล้วโปรแกรมจะแสดงรายการข้อมูลต่างๆของลูกค้า โดยจะมีทั้งส่วนที่สามารถแก้ไขได้และส่วนที่ไม่สามารถแก้ไขได้ดังรูปที่ 9-9 เมื่อทำการแก้ไขข้อมูลเสร็จแล้ว สามารถบันทึกข้อมูลที่มีการเปลี่ยนแปลงได้โดยการคลิกที่ปุ่ม Record หรืออาจจะยกเลิกการแก้ไขได้โดยการคลิกที่ปุ่ม Cancel แล้วโปรแกรมจะเข้าสู่เมนูหลัก



รูปที่ 9-9 แสดงแบบฟอร์มแก้ไขข้อมูลประวัติลูกค้า

ลักษณะของโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรรที่ได้นี้ เป็นโปรแกรมที่ใช้งานสำหรับเจ้าหน้าที่ธนาคารผู้ใช้ระบบ โดยลูกค้าจะติดต่อกับระบบการขอเงินกู้นี้ผ่านเจ้าหน้าที่ธนาคารเสมอ ซึ่งการทำงานต่างๆ ของโปรแกรมจะถูกกระทำโดยเจ้าหน้าที่ธนาคารผู้มีรหัสผ่านเท่านั้น

9.3 การทดสอบการทำงานของโปรแกรม

การทดสอบการทำงานของโปรแกรม จะแบ่งแนวทางการทดสอบเป็น 2 ประเภท คือการทดสอบการทำงานทั่วๆไปเพื่อให้ระบบสามารถทำงานได้ถูกต้องตามที่คาดหวังไว้ และการทดสอบเพื่อหาข้อผิดพลาดของโปรแกรมซึ่งทำได้โดยการป้อนข้อมูลที่คาดว่าจะทำให้เกิดข้อผิดพลาดขึ้น เช่น ข้อมูลที่เกินช่วงขอบเขตที่ระบุไว้ในโปรแกรม

9.3.1 การทดสอบการทำงานทั่วๆ ไปของโปรแกรม

เราจะทำการทดสอบการทำงานต่างๆ ของโปรแกรมให้เป็นไปตามโดเมนของปัญหาที่ได้กล่าวไว้ในบทที่ 6 ซึ่งสามารถจำแนกหัวข้อการทดสอบต่างๆ ได้ดังนี้

- การคำนวณวงเงินกู้และเงินผ่อนชำระต่องวดเพื่อแสดงเป็นตัวอย่าง
- การเก็บบันทึกประวัติลูกค้า รายละเอียดโครงการบ้าน และบัญชีการขอกู้ของลูกค้า
- การรับชำระเงินผ่อนแต่ละงวดจากลูกค้า
- การแสดงประวัติการชำระเงินของลูกค้า
- การตรวจสอบรายชื่อลูกค้าที่ชำระเงินล่าช้า
- การแก้ไขข้อมูลส่วนตัวของลูกค้าเมื่อมีการแจ้งการเปลี่ยนแปลง

9.3.2 การทดสอบเพื่อหาข้อผิดพลาดของโปรแกรม

ในการทดสอบเพื่อหาข้อผิดพลาดของโปรแกรม เราจะทดสอบระบบเพื่อหาข้อผิดพลาดที่เกิดจากเหตุการณ์ต่างๆ เพื่อหาทางแก้ไขข้อผิดพลาดเหล่านั้นให้เหลือน้อยที่สุด โดยจะมีหัวข้อในการทดสอบการทำงานต่างๆ ดังนี้

- ทำการป้อนข้อมูลที่เกินช่วงที่กำหนดไว้ของระบบ เช่น จำนวนเงินกู้เกินช่วงที่ธนาคารอนุมัติ
- ป้อนข้อมูลผิดประเภท เช่น ป้อนข้อมูลที่เป็นสตริงในช่องจำนวนเงิน
- ป้อนข้อมูลลูกค้าซ้ำ เพื่อทดสอบการบันทึกข้อมูลลูกค้าที่ซ้ำซ้อน
- ทดสอบการทำงานของเครือข่าย เช่น การทำงานเมื่อการติดต่อผ่านเครือข่ายมีปัญหา
- ทดสอบการทำงานเมื่อมีผู้ใช้ระบบหลายรายพร้อมกัน

หัวข้อการทดสอบต่างๆข้างต้น เป็นหัวข้อการทดสอบเบื้องต้นสำหรับจำลองการใช้งาน เนื่องจากโปรแกรมที่ได้สร้างขึ้นมานั้นเป็นเพียงการจำลองขึ้นเพื่อศึกษาถึงวิธีการพัฒนาโปรแกรมประยุกต์เชิงวัตถุ ไม่ใช่โปรแกรมที่ใช้สำหรับการทำงานจริง

9.4 บทสรุปการพัฒนาโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

การพัฒนาโปรแกรมประยุกต์สำหรับระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร ได้ใช้แนวทางการพัฒนาเชิงวัตถุทำให้โมเดลที่ได้มีความสอดคล้องกันในทุกขั้นตอนของการพัฒนา โดยได้ใช้เครื่องมือช่วยต่างๆ ในการจัดทำโครงงาน ซึ่งได้แก่

- โปรแกรม Rational Rose เป็น CASE tool ที่มีการนำเสนอโมเดลในสัญลักษณ์ UML ทำให้การวิเคราะห์และออกแบบระบบทำได้ง่าย, รวดเร็ว และตรงตามความต้องการมากยิ่งขึ้น
- โปรแกรมออราเคิล เป็นโปรแกรมระบบจัดการฐานข้อมูล ช่วยในการสร้างตารางฐานข้อมูลแบบรีเลชันแนลที่ได้จากการแปลงคลาสที่ได้จากการออกแบบไปเป็นตารางฐานข้อมูล การสร้างฐานข้อมูลดังกล่าวสามารถสร้างได้โดยง่ายจากเครื่องมือช่วยที่มีมาให้และยังบริหารจัดการฐานข้อมูลกลางของระบบ เป็น

	โปรแกรมที่ช่วยอำนวยความสะดวกและมีประสิทธิภาพในการจัดการฐานข้อมูลสูง
- โปรแกรม VisualAge	เป็นเครื่องมือช่วยในการเขียนโปรแกรมภาษาจาวา ซึ่งเป็นภาษาที่สนับสนุนการเขียนโปรแกรมเชิงวัตถุ จากคลาสที่ได้สามารถนำมาสร้างเป็นโปรแกรมได้เกือบจะทันที โปรแกรม VisualAge ได้รวมเครื่องมือช่วยต่างๆ เช่น Java Debugger ,Java Complier และสามารถใช้งาน Visual Component ได้ง่าย มีคอมโพเนนท์ที่สามารถนำมาใช้งานได้ทันที สามารถตรวจสอบการทำงานและแก้ไขข้อผิดพลาดในโปรแกรมได้โดยสะดวก

9.5 แนวทางในการพัฒนาเพิ่มเติม

9.5.1 ออกใบเสร็จรับเงินเพื่อเป็นหลักฐานให้แก่ลูกค้า เพิ่มเติมฟังก์ชันให้ระบบสามารถพิมพ์ใบเสร็จรับเงินให้แก่ลูกค้าได้ โดยใบเสร็จรับเงินจะมีรายละเอียดแสดงจำนวนเงินในส่วนต่างๆตามที่ลูกค้าชำระ ในขณะนี้ระบบสามารถแสดงฟอร์มสรุปรายละเอียดการชำระเงินของลูกค้าเท่านั้น

9.5.2 เพิ่มแนวทางให้ผู้ใช้ที่มีสถานะเป็น Admin สามารถทำการแก้ไขข้อมูลบางอย่างในฐานข้อมูล เช่น อัตราดอกเบี้ยปัจจุบัน ผ่านทางโปรแกรมระบบบนเครื่องไคลเอนท์ได้ ไม่จำกัดอยู่ที่เครื่องเซิฟเวอร์ฐานข้อมูลเท่านั้น ในขณะนี้โปรแกรมมีการเก็บสถานะของผู้ใช้เป็น Admin และ User แต่มีเฉพาะฟังก์ชันการทำงานสำหรับผู้ใช้ที่มีสถานะ User เท่านั้น ซึ่งเป็นฟังก์ชันการทำงานหลักที่ระบบต้องการ

9.5.3 เพิ่มเติมฟังก์ชันและข้อกำหนดรายละเอียดการทำงานให้เหมือนจริง ขอบเขตฟังก์ชันการทำงานของระบบอยู่ในโดเมนที่จำลองมาและกำหนดความต้องการของระบบขึ้นมาเอง ยังไม่ครอบคลุมฟังก์ชันการทำงานและรายละเอียดจริงทางธุรกิจทั้งหมด หากต้องการนำระบบไปใช้งานทางธุรกิจจริงจะต้องแก้ไขหรือเพิ่มเติมให้เป็นไปตามโลกแห่งความเป็นจริงทางธุรกิจด้วย

9.5.4 เปิดบริการให้ลูกค้าปัจจุบันสามารถตรวจสอบยอดเงินคงเหลือ,ประวัติการชำระเงิน, จำนวนเงินที่ต้องชำระและให้ผู้ที่สนใจกำลังจะเป็นลูกค้าสามารถทดลองประเมินวงเงินกู้และเงินชำระต่อเดือน เพื่อช่วยในการตัดสินใจ ผ่านทางเครือข่ายอินเตอร์เน็ตได้ คอมโพเนนท์ของระบบนี้สร้างตามแนวทางการพัฒนาเชิงวัตถุและเขียนด้วยภาษาจาวาจึงสามารถนำกลับมาใช้ใหม่และสร้างเป็น Applet ใช้งานบนเว็บเพจได้โดยสะดวก

9.5.5 ปรับปรุงส่วนติดต่อผู้ใช้ให้มีรูปแบบสวยงามเป็นระเบียบ น่าใช้ เข้าใจง่าย เป็นที่พอใจและตรงกับความต้องการของผู้ใช้งานมากยิ่งขึ้น

ภาคผนวก ก

IBM VisualAge for Java version 2.0 Features

IBM VisualAge for Java version 2.0

Professional	Enterprise	New Features Summary
✔	✔	Support for JDK 1.1.6 and Swing 1.0.2
✔	✔	New IDE features
✔	✔	New Visual Composition Editor features
✔	✔	JavaBeans for easy access to data
✔	✔	Integration with VisualAge TeanConnection , ClearCase and PVCS
✔	✔	Open Tool Integrator APIs
	✔	Java Team Programming support
	✔	Enterprise Toolkits , including a high performance compiler and a remote debugger
	✔	Servlet Debugger
	✔	New Enterprise Access Builder
	✔	Automated object to Relational Mapping
	✔	Support for SanFrancisco , Tivoli , Lotus and Component Broker
	✔	AIX Development Environment

Support for JDK 1.1.6

VisualAge for Java 2.0 supports the JDK 1.1.6. This support includes Swing 1.0.2, inner classes and anonymous classes, and the Java Native Interface (JNI).

New IDE Features

- Advanced coding tools such as automatic formatting, automatic code completion and fix-on-save
- Context Sensitive Help
- Advanced Debugging tools such as conditional breakpoints and both multiple and incremental program debug

- Support for JavaDoc output
- Enhanced searching capabilities

New Visual Composition Editor Features

- Visual programming support for Swing beans
- Wizards for string externalization to assist in building multi-language applications
- Complete support for Object Serialization
- Ability to import GUIs built in other Java IDEs

JavaBeans for Easy Access to Data

New Data Access Beans give your Java application the power to access relational data from any JDBC-enabled database and make it available on the Web.

Java Team Programming Support

The ultimate quality of your Java applications depends on how well you manage your development process. VisualAge for Java includes a built-in source code and version control system that provides you with a complete audit trail of your project and helps in recovery from undesired code changes. In addition to source code and version control, Enterprise Edition users also get a fully integrated team development environment that improves productivity and reuse levels for any size team. Each developer gets a personalized workspace which is seamlessly integrated with a collaborative repository providing fine-grained versioning of individual components, change identification, and impact analysis across multiple projects. This tight integration avoids time-consuming switching between the repository and the development environment and gives every developer instant "live access" to a library of reusable components.

Integration with VisualAge TeamConnection, ClearCase and PVCS

If you are developing on the Windows platform, you can also check in/out your VisualAge for Java code to either VisualAge TeamConnection, ClearCase, or PVCS.

Enterprise Toolkits, including a High Performance Compiler and a Remote Debugger

The increasing popularity of Java as a server language has placed new requirements for application scalability on Java development shops. Enterprise Edition 2.0 is ready to meet those requirements with a new High Performance Compiler for Java that maximizes the execution speed of your server code. We have also filled our toolkit with cross-platform debugging, testing and performance analysis tools that are all accessed from your development workstation and that target applications built to run on OS/2, Windows NT, AIX, OS/400, and OS/390. Plus, our VisualAge for Java Remote Debugger tests and debugs interpreted Java, compiled Java and C++ on multiple platforms, giving you a true multi-tier development environment. AS/400 developers will also find updates to our AS/400 toolkit such as better integration with the IDE and a new wizard for subfile support. New for S/390 developers is JPort, which pre-screens Java programs to ensure OS/390 portability, and profiles OS/390 Java applications to detect performance bottlenecks.

Enterprise Access Builders

Extending existing enterprise application servers to the Web is a critical success factor for leading-edge IT shops. VisualAge for Java, Enterprise Edition provides a collection of Enterprise Access Builders that give you access to enterprise systems such as relational data, CICS transactions or SAP R/3 applications from your Java programs. Our unique approach lets you access multiple systems from a single Java application and uses a consistent programming interface across diverse enterprise systems to reduce your learning curve, maximize your productivity and increase the run time performance of your applications. Our Enterprise Access Builders include:

- **Access Builder for CICS** including CICS ECI, CICS EPI, CICS EXCI
- **Access Builder for Encina** using the DCE Encina Lightweight Client (DE-Light)
- **Access Builder for SAP R/3** using SAP R/3 BAPI Business Objects
- **Access Builder for Data** for JDBC access to enterprise data

- **Access Builder for J2C++** for access to C++ programs
- **Access Builder for RMI** for creating distributed Java applications
- **Access Builder for Persistence** for mapping Java objects to relational databases

Automated Object to Relational Mapping

A new Enterprise Access Builder for Persistence provides a set of tools that automate the task of mapping the persistent state of Java objects to relational databases. These tools generate a layer of code that implements all of the JDBC access calls necessary to insert, update or retrieve the data for an object from an SQL database. The programming model used to create the persistent Java objects is based on the industry standard Enterprise JavaBeans architecture .

The programming model used to create the persistent Java objects is based on the industry standard Enterprise JavaBeans Architecture. The Enterprise Access Builder for Persistence and its generated Java code will be upward compatible to support full EJBs in a future release of VisualAge for Java, targeted for delivery in the fall of 1998. This functionality will coincide with the delivery of IBM's EJS environments, such as IBM's WebSphere web application server and IBM's Component Broker.

Servlet Builder

Enterprise Edition users can now use visual programming techniques to create and test servlets. Using Servlet Builder, a wide variety of custom and off-the-shelf business objects can be Web-enabled and used within the VisualAge for Java reusable parts library. When used along with the IBM WebSphere Application Server, site builders can test and debug a combination of pages built using pure HTML compiled JavaServer Pages (JSP), and Servlet Builder visual servlets.

Open Tool Integrator APIs

Advanced users and commercial software developers who need to extend VisualAge for Java can use our API to:

- Add third-party tools that are launched from within the IDE
- Store and retrieve components from the integrated repository

- Add JavaBeans to the Visual Composition Editor's parts palette

Build Total Solutions with Java: Support for SanFrancisco, Tivoli, Lotus, and Component Broker

IBM has a rich portfolio of Java-based solutions, and VisualAge for Java is the tool of choice for developing many of these systems. Enterprise Edition includes SanFrancisco wizards for building applications from the SanFrancisco Business Application Components. VisualAge for Java can also be used to build Java-based business productivity applications using the Lotus eSuite components and to develop, debug and test Lotus Notes Agents. Version 2.0 includes new Tivoli Beans used to make Java applications "ready to manage" with Tivoli's enterprise management software. Tivoli lets you easily track activities such as version upgrade, daily use monitoring, and operation and distribution to target systems. Our complete IDL environment can be used to create and manage applicatins that can communicate with CORBA business objects, such as those deployed on IBM's Component Broker application server.

AIX Development Environment

Now, with Version 2.0, you can use AIX 4.2 and 4.3 workstations as your development platform.

ภาคผนวก ข

Frequently Asked Question

About VisualAge for Java

VisualAge for Java Frequently Asked Questions

What differentiates VisualAge for Java, Enterprise Edition from other enterprise application development tools?

VisualAge for Java is a superior development environment for building scalable applications that allow businesses to leverage what runs their business today and extend these applications to e-business. And it is a superior Enterprise Computing Environment, with remote debug that allows users to test applications, client to services, end-to-end. Adherence to a client to services computing model allows businesses to optimize their logic on a controllable server, while providing universal access through thin clients. As a team development environment, VisualAge for Java automatically manages source and version control for teams of Java developers working on multiple projects. And it provides excellent connectivity to CICS, IMS, MQSeries, Java, CORBA IDL, D++, Notes, Host OnDemand and SAP R/3. The VisualAge toolset builds on top of what Studio offers, providing specialized Java development capabilities for building javabeans, EJB's, and eConnectors as mentioned above (for IMS, CICS, MQSeries, and CORBA development, for example). VisualAge offers testing and debugging capabilities unique in the marketplace. Developers can work in isolation, with no impact on production or test systems, and still be able to test and debug complete web applications involving HTML, servlets, javabeans, EJB's running in EJB Server containers (WebSphere), JavaServer Pages, and even compiled Java components running on remote servers without ever leaving their IDE. The integration between IBM's tooling and WebSphere is outstanding and second to none in the industry.

Does VisualAge for Java support the complete application development life cycle?

VA Java is poised to offer an end-to-end solution that supports the application development life cycle from start through re-engineering. Currently VisualAge for Java supports construction/coding, testing, prototyping, and re-engineering. And as a result of a partnership with Rational Software, it will support planning, analysis, design, BPR, and maintenance. The XML toolkit, which provides a bridge between VisualAge for Java and Rational Rose, is shipped

in Version 3.0 of the Enterprise EditionWhat is VisualAge for Java's database functionality features and what is its relative intuitiveness of establishing database connectivity?

VisualAge for Java has several database functionality features including seamless integration with database objects, support for visual editing and formatting of data controls, control palette access to data controls, and the ability to change data in tables. When establishing connectivity, Developers use the Data aware JavaBean components on the Visual Composition Editor Palette. By selecting which database, table, columns, and joins, the selector bean is generated for the developer. The beans can then be attached to the visual controls to form a complete application. A visual Stored Procedure builder, new in Version 3.0, allows developers to create, test and deploy Java stored procedures

What are VisualAge for Java's capabilities for team development?

- Library management
- Release control
- Code management
- Version control
- Check in/check out facilities
- Customizable preferences for each developer
- Repository reports
- Global updates
- Parallel development

What are some additional ease of use features of VisualAge for Java?

- A powerful browser for easy application object search and transversal, including a hierarchical explosion of application objects
- Facilitation of prototyping and iterative development
- The ability to paint SQL graphically
- Automated version control
- An expression builder
- "On-the-fly" syntax checking

- The ability to view the code in an application at any level of detail (e.g., view the inherited code for a specific object).

What facilities does VisualAge for Java provide in version control?

- The ability to maintain multiple versions of data model and applications
- Automatic version numbering
- Integrate external version control systems

VisualAge for Java also provides SCCI Integration with file base source control systems, which gives it an advantage over its competitors.

For which system management components does VisualAge for Java provide automated support?

VisualAge for Java provides automated support for the development, roll-out and maintenance for the following system management components:

- Configuration management
- Distributed impact sensitivity reporting
- Intelligent regeneration
- Library management
- Package versioning
- State control
- Turnover management

What application venders make third party components for use with VisualAge for Java?

Oracle and SAP

What are some of the component based development features of VisualAge for Java?

VisualAge for Java provides an incremental team-based development environment, which sets it apart from its competitors. Other development features of the product are:

- The capability to identify components by granularity
- The capability to search for components by name

- The capability to search for components by type
- The capability to search names by synonym
- The capability to search types by synonym
- The capability to search for components by user
- The capability to search for components by type
- The capability to search for national languages implemented in components
- The capability to notify users of changes to the components
- A central environment for data administration teams to manipulate components
- The facilities to electronically publish set standards for components
- An electronically published set of standards for components available on-line during software development
- The facilities to test components
- The capacity to establish multiple libraries for component review and quality assurance
- The facility to track components use by individual
- The facilities to test components
- The facilities to track the number of times a component is used
- The capability for component versioning
- The capability for component authorization
- The capability for check in facilities
- The capability for component authorization
- The ability for support for reuse of inheritance
- GUI capabilities to connect components
- Wizards for component creation
- Wizards for component modification

In what technical environments does VisualAge for Java provide the capability for building full range applications?

VisualAge for Java provides the capabilities to build a full range of applications in the following technical environments:

- OS/2
- Windows 95
- Windows 98
- Windows NT
- AIX
- Linux
- OS/390
- OS/400

What capabilities in archival storage does VisualAge for Java provide?

The VisualAge for Java team server is a robust scalable team environment that allows multiple developers to work in multiple releases of multiple projects simultaneously. Powerful merge browsers allow developers to migrate code changes into a common supported base.

What access controls and roles does VisualAge for Java support?

VisualAge for Java components are "owned" by a component owner, who is responsible for the quality, reliability, publishing into the code base, and the communication of the components interface to the development team, ENVY based. This feature gives VisualAge for Java an edge over its competitors. Other access control features are:

- An administrator role, who can define and authorize users and groups
- The ability to Read, update and delete authority against components and configurations
- A built-in security features that allow the repository administrator to extend read and/or update authorities to repository users
- The ability to define security by individual user
- The ability to define certain repository access as public

What capabilities in change management and version control facility does VisualAge for Java provide?

- Facility for shared real-time updates

- Facilities for developers to accept changes once they have been notified of changes
- Facility for creating a central repository of objects
- Facility to copy configurations to create new configurations and new versions of configurations
- Ability to record changes to configurations and objects and by whom
- Ability to freeze a configuration to prevent undesirable updates
- Ability to create and maintain relationships between versions of configurations
- Facility to synchronize changes between versions of configurations by moving collections of objects across configuration versions

What capabilities in backup and recovery does VisualAge for Java provide?

VisualAge for Java has the ability to backup and recover individual configurations as well as backup and restore the entire repository. Its archiving feature has a configurable scope of the archiving process. The products meta management has data compression capabilities.

What capabilities in application partitioning does VisualAge for Java provide?

VisualAge for Java's profiler supports OS/390 executables. Other application partitioning capabilities include:

- The repository possesses a facility which exists natively within the product to separate the application into three distinct components: GUI, application logic, and database and associated logic
- The tool is capable of deploying these components on three distinct platforms
- The tool provides a drag-and-drop user interface to move an application component from one node to another

What standards does VisualAge for Java enforce?

The standards that VisualAge for Java enforces include the following:

- Java language Version 1.1
- Multivendor relational query--ODBC

- Multivendor relational query--JDBC
- OMG CORBA--complaint object request broker dynamic link support
- OMB CORBA--complain object request broker static linkage support
- Support for CICS API
- Support for IMS
- Support for Lotus Notes
- Support for MQSeries

What features of object-oriented enablement does VisualAge for Java possess?

VisualAge for Java possesses object-based and object-oriented enablement

What features of object-oriented programming does VisualAge for Java possess?

- Facility for class and object browsing
- Facility for creation of reusable objects
- Support for object-oriented features like inheritance
- Availability of prefabricated object repository
- Sub-classing and super-classing of controls

What object-oriented concepts does VisualAge for Java support?

- Objects containing objects (i.e., objects must be able to form a complete hierarchy)
- Single inheritance
- Reference semantics (i.e., objects must have unique identities and not be value based like relational systems)
- Classes for encapsulation (i.e., classes must provide the ability to hide internal implement restricting object access to method invocation)
- Private functions
- Classes as type system (i.e., there are not two separate data structuring procedures, but rather class system should be type system of language)
- Shared variables (i.e., each instance of a class can have the same classwide value for a shared variable)

- Metaclasses (i.e., they allow development of classes with shared, but modified class behavior)
- Dynamic typing (i.e., runtime binding of class to a given instance should be supported)
- Shadowing (e.g. overriding inherited behavior) and specialization (modifying inherited behavior)
- Overloading (e.g. the capability of operators having different behavior depending on the class of the object being processed)
- Message polymorphism

What object-oriented analysis and design methodologies does VisualAge for Java support?

VisualAge for Java supports the Booch model, UML (which is based on the Booch model), the Jacobsen model/OOSE, the Rumbaugh model/OMT methodologies, Rational Performance STudio and Mercury Interactive LoadRunner.

What debugger features does VisualAge for Java have?

VisualAge offers testing and debugging capabilities unique in the marketplace. Developers can work in isolation, with no impact on production or test systems, and still be able to test and debug complete web applications involving HTML, servlets, javabeans, EJB's running in EJB Server containers (WebSphere), JavaServer Pages, and even compiled Java components running on remote servers without ever leaving their IDE. Other debugging features include:

- A facility for point and click as well as drag and drop features
- A code viewing facility during runtime
- Support to examine or set variables
- Support to step through interpreted code during the test process
- Capacity to edit and continue during break mode
- Capacity for multiple tier debugging
- Capacity for SQL debugging
- Capabilities for cross-language debugging
- Capability to debug multiple tasks in parallel

- Point and click capability for breakpoint settings
- Capability to profile an application as it runs
- Capability to profile an application in an event log to monitor actions
- A filter that can be activated with an event log to capture specific elements of an application.

What data debugging features does VisualAge for Java have?

Unlike its competitors, VisualAge for Java allows for remote debugging from multi-platform client to services. Other debugging features of the product include:

- Facility for editing variables at runtime
- Ability to display contents of a named register/variable
- Ability to display contents of a field
- Ability to set watch variables
- Ability to set content of a register/variable

What capabilities in syntax checking does VisualAge for Java have?

VisualAge for Java allows for automatic syntax checking on save, which lets developers know if an item will compile, before it is saved in the repository. Other features include:

- Object checking
- Break points on async. events and messages
- Thread debugging

With what third party products with VisualAge for Java interface?

VisualAge for Java will interface with the following:

Object Modeling Tools:

- Rational Rose Version 98 through a bi-directional type of interface. Update occurs are passive and data bridges are through the interface.
- XMI Interface through a bi-directional type of interface. Update occurs are passive and data bridges are through the interface.

Software Configuration Management

- Merant PVCS through a bi-directional type of interface.
- Rational ClearCase through a bi-directional type of interface. Update occurs are passive and data bridges are through the interface.
- IBM TeamConnection through a bi-directional type of interface. Update occurs are passive and data bridges are through the interface.

Systems Management Tools

- BMC Patrol
- Computer Associates Unicenter TNG
- Hewlett Packard Operations Center
- Tivoli (Tivoli managed) TME through a bi-directional type of interface. Update occurs are passive and data bridges are through the interface.

What capabilities does VisualAge for Java's programming language have?

VisualAge for Java's Data Persistence Framework maps existing data schema to Enterprise JavaBean components, as well as provides support for Entity Container Managed Beans Persistence. The programming language for the product is portable and future releases will provide full support for embedded and dynamic SQL.

What capabilities with interface features does VisualAge for Java have?

- Support for multiple database connections from an application
- Support for two-phase commit across multiple databases
- Support for standard and extended SQL and DDL statements of target database
- Support for Static SQL
- Support for Dynamic SQL
- Support for transaction management
- Support for asynchronous database operations
- Binary Large Object (BLOB) support
- Character Large Object (CLOB) support
- Ability to call stored procedures
- Ability to create cursors

- Ease of establishing connectivity to a database

What capabilities in data administration does VisualAge for Java have?

- Stored procedures
- Tables
- Views
- Primary keys
- ER diagrammer
- MQSeries
- Triggers
- SQL Editor to execute ad hoc queries
- Row level locking
- Calculated field display

What capabilities in database sources in grid control concepts does VisualAge for Java have?

- Database query result set
- External Procedure Call
- TP Monitor Interfaces
- RPC Call Interfaces

What execution platfoms does VisualAge for Java support?

VisualAge for Java creates Java bytecodes, which could be run anywhere. Platforms that have tested applications, applets, etc. include:

- Non-programmable terminal
- DOS
- Macintosh O/S
- OS/2
- Unix Motif
- Windows 3.X
- Windows 95

- Windows 98
- Windows NT
- AIX
- Linux
- OS/390
- OS/400
- Solaris

On what presentation services can VisualAge for Java build fully functional applications?

VisualAge for Java provides support for Java, AWT, Swing and Servlets.

In what database management systems can VisualAge for Java build fully functional applications?

VisualAge for Java can build fully-functional applications that support read-only, full-write and distributed full function for the following database management systems:

- DB2/Unix
- DB2/400
- DB2/MVS
- DB2 Universal Database
- Oracle
- Sybase
- Lotus Notes

What TP monitors does VisualAge for Java support?

VisualAge for Java supports CICS, Encina and Asynchronous Transaction Process TP monitors

For what native database drivers does VisualAge for Java provide support?

VisualAge for Java provides support for Oracle, Sybase (through JDBC only, not EJB persistence), and DB2.

What capabilities with database connectivity does VisualAge for Java have?

- Supports IBM DB2 Universal Database via JDBC access beans, SQLJ, Stored Procedures and Persistent EJBs
- Supports Oracle via JDBC and persistence framework
- Supports Sybase via JDBC

What Java standards does VisualAge for Java support?

- JDBC 1.1
- JDBC 2.0
- Enterprise Java Beans
- JNDI (naming)
- JMS (messaging)
- JTS (transactions)
- JMAPI
- Java Server Pages

What are VisualAge for Java's capabilities in developing application types?

- Batch
- OLTP
- Real-time
- Web applications

What are VisualAge for Java's capabilities in developing application complexity?

- Read-only
- Distributed updates
- Single-dedicated database(s)

- Many heterogeneous databases
- Complex logic
- Simple retrievals/calculations
- Single-character platform/GUI
- Many portable platforms(GUI)

VisualAge for Java also supports the following complex data manipulations:

- Create
- Read
- Update
- Delete

What application topology capabilities does VisualAge for Java have?

VisualAge for Java supports n-tier client to services computing as outlined in the IBM Application Framework for e-business. Other application topology capabilities include:

- Client/server two-tier data passing
- Client/server two-tier message passing
- Client/server three-tier message passing monolithic application server
- Client/server three-tier message passing granular application server

What are VisualAge for Java's features for generating end user reports?

VisualAge for Java provides support for drag-and-drop capabilities to place and move controls, as well as support to resize controls.

ภาคผนวก ค

Applying the Model-View-Controller Design Paradigm in VisualAge for Java

Applying the Model-View-Controller Design Paradigm in VisualAge for Java

by Scott Stanchfield

The Model-View-Controller (MVC) design paradigm is an incredibly useful tool in writing maintainable programs. MVC lets you separate your business logic from your Graphical User Interface (GUI), making it easier to modify either one without affecting the other. Initially, MVC requires a little extra planning and coding, but the long-term benefits are well worth it.

When MVC was first publicized, visual composition tools were limited in capability and not nearly as popular as they are today. The visual composition tools of today, such as the VisualAge for Java Visual Composition Editor (VCE), support true visual programming, enabling you to visually create much of an application's GUI and business logic.

Unfortunately, this kind of simple visual programming has eroded solid design techniques. Many developers simply sit down at their desk, open the VCE, and start clicking the mouse. Rather than thinking through GUI and application designs, they experiment. Eventually they get things to "look right," then draw lines to "make the application work."

This article discusses the MVC design paradigm and demonstrates its application in visual programming. Keep in mind that MVC is but a single technique in a developer's design arsenal, and should be used in combination with other techniques, including object-oriented analysis and design.

What is MVC?

The MVC paradigm was introduced by Smalltalk developers at Xerox PARC (Palo Alto Research Center) in the late 1970s. The basic idea is to split your application into three distinct parts, each of which can be replaced without affecting the others:

- **Model** – the data of your application, along with the business logic that defines how to change and access that data. The model can be shared among any number of view and controller objects.

- **View** – The means of presenting the model's data to the outside world. This could take the form of a GUI, generated speech, audible tones, printouts, or even non-user oriented output, such as turning on an air conditioner.
- **Controller** – The means of gathering user or other environmental input and providing feedback to the model, normally changing some of the data in that model.

Interaction Between Components

Let's examine the interaction between these three pieces. Initially, the view, and possibly the controller, ask the model for its current state. The view may present data to the user, and the controller may check the data to help decide how to handle user interaction.

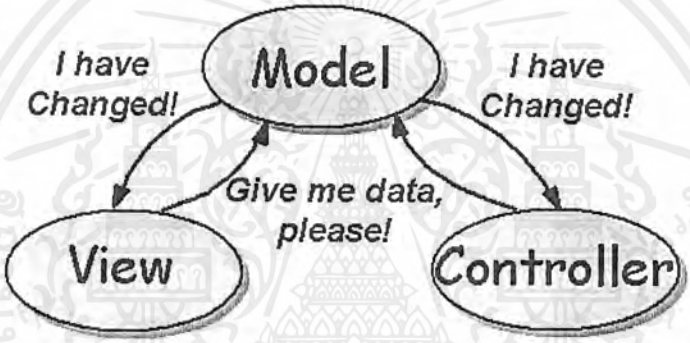


Figure 1 Model and user interface communication

As seen in Figure 1, the view and controller will typically "listen" for changes to the model. The Java language implements this notification using events. Whenever the model says "I have changed," the view and controller can ask for the new state of the model's data, then update their presentation to the user.

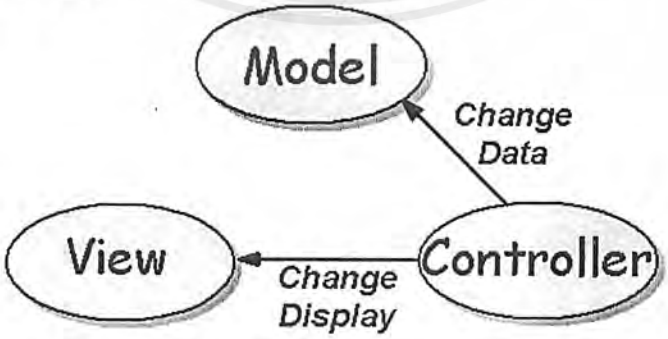


Figure 2. Controller to model communication

Figure 2 shows user interaction in the application. If the user decides to interact, the controller takes charge. It watches for user input, such as clicking or moving the mouse or pressing keyboard keys. It decides what the interaction means, and asks the model to update its data and/or the view to change the way it displays the data.

For example, you could have a view that displays a set of data in a Swing JList component. You add a scrollbar as a controller, which directs the view to change which items are displayed. Further, you could add *another* controller, perhaps a Swing JTextField, to take user input and ask the model to add the new value to the set. Of course this would cause the model to shout "I have changed!", to which the view responds by asking for the set of data.

The above scenario has two controllers. One watches for mouse interaction with a scrollbar, while the other accepts user input to add to the model. You can have any number of controllers, and any number of views in your application.

Suppose you have an application that presents information from a database in a table and pie chart. The table can be scrolled using horizontal and vertical scroll bars, and new data can be entered via a pair of text fields. The MVC pattern might be applied as shown in Figure 3.

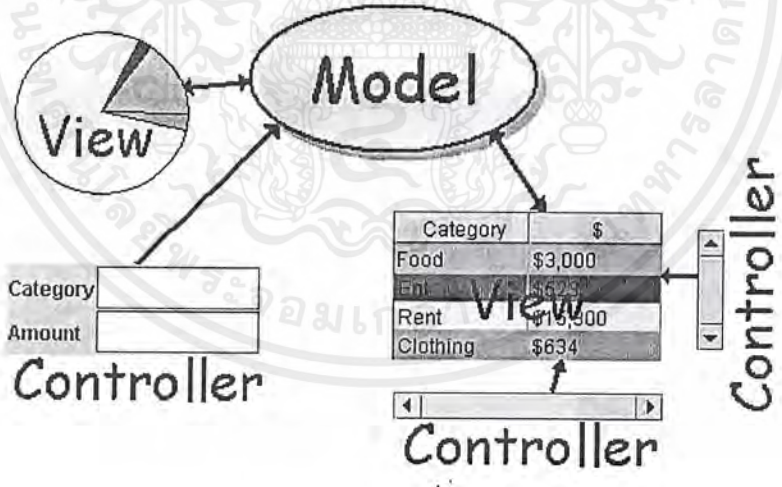


Figure 3. Multiple views and controllers

This example includes a model, two views and three controllers. The scrollbar controllers update only the table view, while the text field controller updates the model.

Delegates -- Combining View and Controller

You may be concerned at this point about separation between the scrollbars and the table they scroll. In theory this separation is good, but in practice it can make life much more difficult:

- You need to separate the GUI among multiple VCE sessions, and provide promoted variables and events to communicate between the controller GUI and the view GUI
- Where does an AWT TextField go? It acts like a view to display existing data, and like a controller to change the data.
- Many components have built-in support for user interaction such as scrolling.
- Interaction between multiple controllers and views can be quite heavy and complex.

Because of such issues, the MVC paradigm is often simplified by *combining* views and controllers. There are several names for this approach -- this article uses the name *delegate* to refer to a combined view/controller. (This is the term used by Sun in describing the Swing GUI components.). This combination into a delegate is shown in Figure 4.

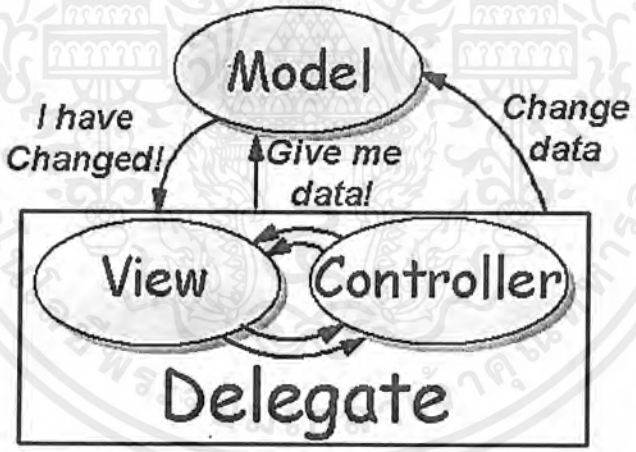


Figure 4. Combining view and controller into a delegate

In a delegate, the view and controller communicate as necessary to perform their duty. It is a good idea to keep this communication separate when possible, but often it's impossible to break a component into a view or a controller. And, in practice, the separation doesn't provide nearly as significant a benefit as separation of the Model and Delegate.

The delegate *as a whole* communicates with the model in the same way described earlier for the view and controller. The separation between the model and the user interaction of the

delegate is the key to the success of this model. Although this version of MVC is somewhat simplified by the combination of the view and controller into the delegate, the design is still MVC-based. You can use it to create some interesting and very flexible applications in VisualAge for Java.

Why is MVC So Important?

By this point you may be thinking, "Sounds like neat theory, but it also sounds like a lot of work!" You're right on the first point, but as you'll see later, implementation in VisualAge for Java is actually very easy! Another question might be "Why is MVC so important that Scott would take the time to write this article?" Development typically occupies 10% or less of a program's life cycle. Therefore, program maintenance should be a developer's *number one* concern. A clean, understandable design is a good start, but to be really effective, the design should separate business logic from user interface.

Think about some of the possible ways a program changes over time:

- The current state of GUI design evolves, and a company decides to update their GUIs. (Often this is referred to as "making the GUI look more professional.")
- A company changes network architectures, and wants to port their applications.
- A company decides to create a limited-feature demo of their application.
- A company wants to add a new way of examining existing data.

Changes like these often involve *either* the GUI *or* the business logic of an application. If the GUI code and business logic are tightly coupled, making these changes can be quite difficult. Using an MVC-based design makes the changes less extensive, and more importantly, more isolated, reducing the chance of introducing bugs in unrelated code.

Using an MVC-based design, the above changes are fairly simple:

- Updating the GUI requires *only* changing GUI code. The stable business logic is not touched.
- Updating network architectures, perhaps changing from a two-tier to a three-tier database architecture requires modifying only part of the model. The stable GUI is not touched.
- Creating a limited feature demo might merely be a matter of subclassing the model to block access to some features. Again, no change to the GUI.

- Adding a new way to examine data is simply a matter of adding a new view. Often *no* change to the model is necessary, nor is it necessary to change other views!

A final thought on the importance of MVC: by separating the business logic from the GUI, you can also separate the coding tasks. Separate developers can work on each part -- a GUI specialist coding the GUI and a domain expert coding the business logic.

Implementing MVC in VisualAge for Java

Note: This article assumes VisualAge for Java, Version 3.0. The techniques explained will also work with VisualAge for Java 2.x, but you cannot use the Quick Form support to create your GUI.

Now that we've laid out the concepts and rationale, let's look at implementation details by walking through a simple address book application. Being good MVC designers, we decide to split the application into models and delegates. We design the application with a two-level model:

- **Address Book** -- A model that stores a collection of addresses. The interface that defines this model requires methods to add and look up individual address records.
- **Address Data** -- A model that represents a single address record. Many of these will be stored in an Address Book.

We choose this two-level model because it helps show that MVC can be applied at many levels in an application. You can use MVC when creating GUI components (such as Sun's Swing components), as a good division of labor for your entire application, or at any level in between. Models can contain references to other models and delegates can contain references to other delegates.

To understand the creation of this application, you can break the design into four phases:

1. **Define a model interface** -- Define how the model and delegate communicate. Defining a protocol (or contract) between them is best represented in Java by writing an interface class.
2. **Define concrete model(s)** -- Implement the interface to create actual model classes for use in an application.

3. **Create delegate(s)** -- Create a delegate (GUI) class that has a variable of the model interface type. The GUI can communicate with the model, and any model class that implements that interface can be plugged in at run time.
4. **Connect as an application** -- Finally, create an application by simply connecting one or more delegate classes with a concrete model class.

We will create a concrete model that stores the data in a hash table, but the actual data location does not matter. In part 2 of this series, we implement filtering and sorting models and discuss how you can use data from other sources, stored in data bases or on other machines.

Defining a Model Interface

The first step in defining an MVC application is to define how your model and delegate will communicate:

- How will the model inform the delegates that it has changed?
- What data can the delegates obtain from the model, and how do they access it?
- What data can the delegates request be changed, and how do they request it?

Model-to-Delegate Change Notification

Several issues are involved with change notification. The first and most important is the granularity of the notification. You can notify at several levels. For example:

- You can fire an event that says "something's changed," without giving details. The delegate then needs to examine *all* the data to determine what has changed. This is effective if the delegates will normally need to reread all the data after any change. For example, Swing provides a generic ChangeEvent for just this purpose.
- You can simply use JavaBean bound properties as the available data in your bean. PropertyChangeEvents are fired any time one of the properties changes, which effectively limits the scope of the change notification. This approach works well when you have a one-to-one correspondence between properties and the GUI components that represent them to the user.
- You can create a custom event class and listener interface that captures specific information about the nature of a change. For example, Swing's TreeModel fires a

TreeModelEvent to its TreeModelListeners. The TreeModel can be more specific about the type of change, telling its listeners that nodes were added or removed from other nodes, structure has changed for some subtree, and so forth. This allows the delegate to update its presentation more efficiently; if the updated area doesn't affect the current display, nothing changes.

Bound Properties and GUI Components

We can take advantage of the JavaBean specification's definition of bound properties to assist in creation of our models. Bound properties are properties that fire a PropertyChangeEvent any time their value changes. We can use this as the "I've changed" notification from our model.

We can also take advantage of bound properties in our GUI components. VisualAge for Java provides excellent support for bound properties in the form of property-to-property connections. Two bound properties connected via a property-to-property connection will update each other's values when they change.

Assuming we have bound properties in the model and in the delegate, much of the communication between a model and a delegate can be accomplished through a property-to-property connection!

The problem is that Swing and AWT *do not* bind properties that should be bound. For example, TextField does not bind its *text* property. Fortunately, we'll be using the Quick Form dialog in the VCE, and that makes dealing with changes in a TextField easy. For Swing components, however, we would need to create a BoundJTextField. For a discussion on how to do this, see Binding a Non-Bound Property at the VisualAge Tips and Tricks site. Note that this article will be using a TextField.

Making Things Easier in VisualAge

VisualAge for Java provides excellent tools for developing *classes*, but not much support for developing *interfaces*. The above four-step process assumes we define the interface first, then implement that interface with concrete classes. Because we're using bound properties in our model, it would be much easier to *start* with a concrete model. The BeanInfo editor in VisualAge for Java provides a very easy way to define a bean with bound properties. We will use the

BeanInfo editor to flesh out a concrete model, then create a model interface based on that concrete class.

Once we have the interface, we can implement any other models we like and plug them into our application.

To create this interface, you can use the AutoGut tool, a plug-in for VisualAge for Java. AutoGut examines a class and creates an interface by "gutting" the methods from that class. We'll walk through the use of AutoGut in a moment, but first you need to download and install the tool. You can get AutoGut from http://www.javadude.com/tools/autogut. Follow the instructions on that page to install the tool.

นำมาจากส่วนหนึ่งใน website

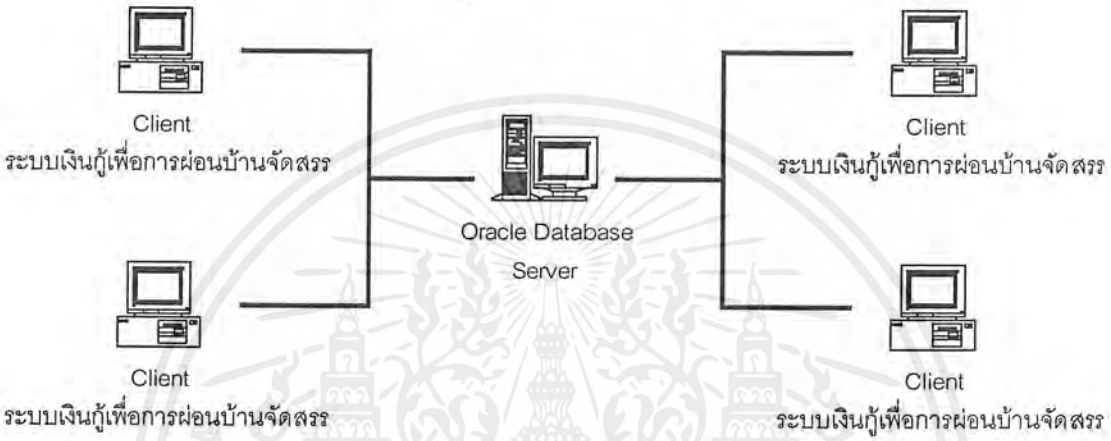
http://www7.software.ibm.com/vad.nsf/Data/Document2672

ภาคผนวก ง
คู่มือการติดตั้งและใช้งาน
โปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

คู่มือการใช้งานโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

1. ความต้องการของระบบ

ระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร มีลักษณะการทำงานแบบ Client / Server ผ่านเครือข่าย LAN โดยมี Oracle Server เป็นฐานข้อมูลกลางของระบบ ในเครื่อง Client จะทำการลงโปรแกรมระบบเงินกู้เพื่อการผ่านบ้านจัดสรรซึ่งเขียนขึ้นมาด้วยภาษา Java โดยใช้โปรแกรม VisualAge for Java 2.0 ซึ่งสนับสนุน JDK 1.1.6



รูปที่ 1 ระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

1.1 Oracle Database Server เป็นฐานข้อมูลกลางของระบบ เก็บตารางฐานข้อมูลรีเลชันแนล, ฟังก์ชันและโพรซีเยอร์ที่เขียนขึ้นมาเพื่อสนับสนุนการใช้งานโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร เครื่อง Server นี้ใช้ Oracle 8.05 บนระบบปฏิบัติการ Window NT 4.0

1.2 Client เป็นเครื่องเทอร์มินอลสำหรับเจ้าหน้าที่ธนาคารใช้ในการทำงาน ในเครื่อง Client นี้ลงโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร เนื่องจากโปรแกรมนี้เขียนขึ้นมาด้วยภาษา Java จึงทำให้ไม่ขึ้นอยู่กับแพลตฟอร์มใดแพลตฟอร์มหนึ่ง สามารถทำงานบนระบบปฏิบัติการได้ทั้ง Windows 95/98/NT

1.3 เครือข่าย LAN เชื่อมต่อระหว่างเครื่อง Client กับ Database Server

2. การติดตั้งระบบ

2.1 ไฟล์ที่ใช้ในโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

ไฟล์ทั้งหมดที่ใช้ในโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรรนั้น รวมอยู่ในโฟลเดอร์ HousingLoanSystems ประกอบด้วย

- โฟลเดอร์ Banking เก็บไบนารี่ไฟล์ของโปรแกรมที่เขียนขึ้น ซึ่งได้มาจากการคอมไพล์ด้วยโปรแกรม VisualAge for Java

- โฟลเดอร์ Bin เก็บไฟล์ที่ใช้ในการรันโปรแกรม ซึ่งได้นำมาจาก JDK 1.1.8
- โฟลเดอร์ Lib เก็บไลบรารี่ไฟล์ที่ถูกอ้างถึงจากตัวโปรแกรม
- ไฟล์ HousingLoan.bat เป็น BAT file ที่ใช้เรียกโปรแกรมขึ้นมา ภายในไฟล์นี้มีการกำหนด Class Path เพื่อให้โปรแกรมคอมไพล์สามารถอ้างอิงไฟล์ที่เป็นไลบรารี่ได้

2.2 การติดตั้งฐานข้อมูลของระบบในเครื่อง Database Server

ทำการติดตั้งโปรแกรม Oracle Server 8.05 ลงในเครื่องที่เป็น Database Server จากนั้นจึงทำการสร้าง User ใหม่ โดยมีรายละเอียดดังนี้

SID : bk

User : dear

Password : dear

IP Address : 161.246.6.117

หากไม่สามารถตั้งค่ารายละเอียดต่างๆ ได้ดังข้างต้น จำเป็นจะต้องทำการแก้ไขโปรแกรมในไฟล์ BankerOption.java ในเมทธอด connectDatabase() ให้ได้เป็นค่าที่ถูกต้อง

ทำการ import ไฟล์ HousingLoanDB.dmp ซึ่งเป็นไฟล์ที่ประกอบด้วย Tables, Sequences, Store Procedures และ Store Functions โดย import ไว้ใน User DEAR

2.3 การติดตั้งโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรรลงในเครื่อง Client

คัดลอกไฟล์ต่างๆ ที่อยู่ในโฟลเดอร์ Execute ได้แก่

- โฟลเดอร์ Banking
- โฟลเดอร์ bin
- โฟลเดอร์ lib
- ไฟล์ HousingLoan.bat

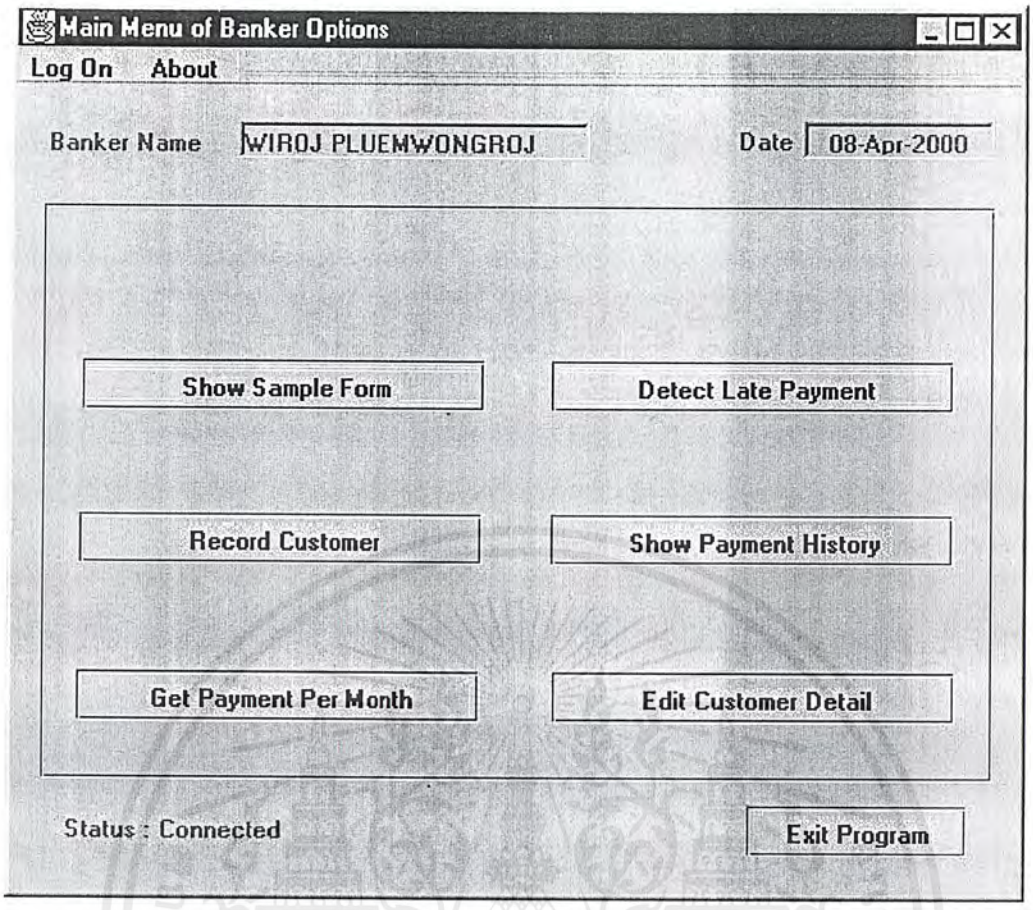
คัดลอกเก็บไว้ใน path C:\HousingLoanSystems\ จากนั้นจึงสร้าง Shotcut บน Desktop ให้ไปเรียกไฟล์ HousingLoan.bat อีกทีหนึ่ง

3. การใช้งานโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

3.1 เมนูหลัก

เมื่อเรียกใช้งานระบบจาก Shortcut บน Desktop จะเป็นการเปิด BAT file เพื่อไปเรียกใช้งานโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร โปรแกรมจะแสดงหน้าจอหลักขึ้นมา ดังแสดงในรูปที่ 2 ผู้ใช้จะไม่สามารถเข้าสู่ฟังก์ชันงานใดๆ ได้หากยังไม่ได้ทำการ Logon เข้าสู่ระบบ

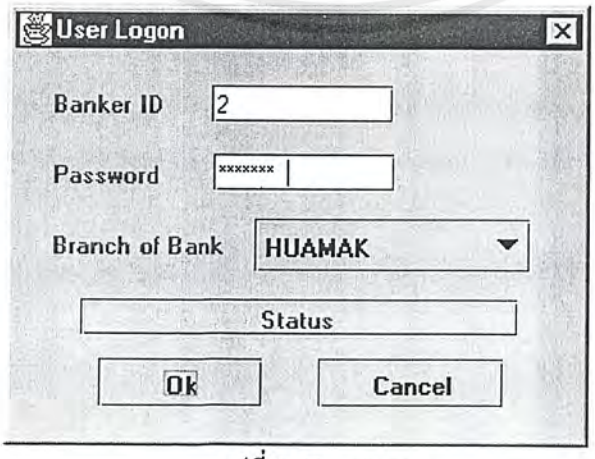
หากคลิกที่ปุ่ม Exit Program จะเป็นการปิดโปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร



รูปที่ 2 หน้าจอหลัก

3.2 User Logon

ผู้ใช้ระบบนี้แบ่งเป็น 2 ประเภทคือ Admin และ User ในขณะนี้ระบบมีฟังก์ชันงานเฉพาะสำหรับ User ใช้งานทั่วไปเท่านั้น ผู้ใช้สามารถทำการเข้าสู่ระบบโดยคลิก Logon บนแถบเมนูและเลือก User Logon จะขึ้นฟอร์มให้ป้อนรหัสประจำตัวผู้ใช้ , รหัสผ่าน และสาขาที่สังกัดอยู่ให้ถูกต้อง ดังแสดงในรูปที่ 3



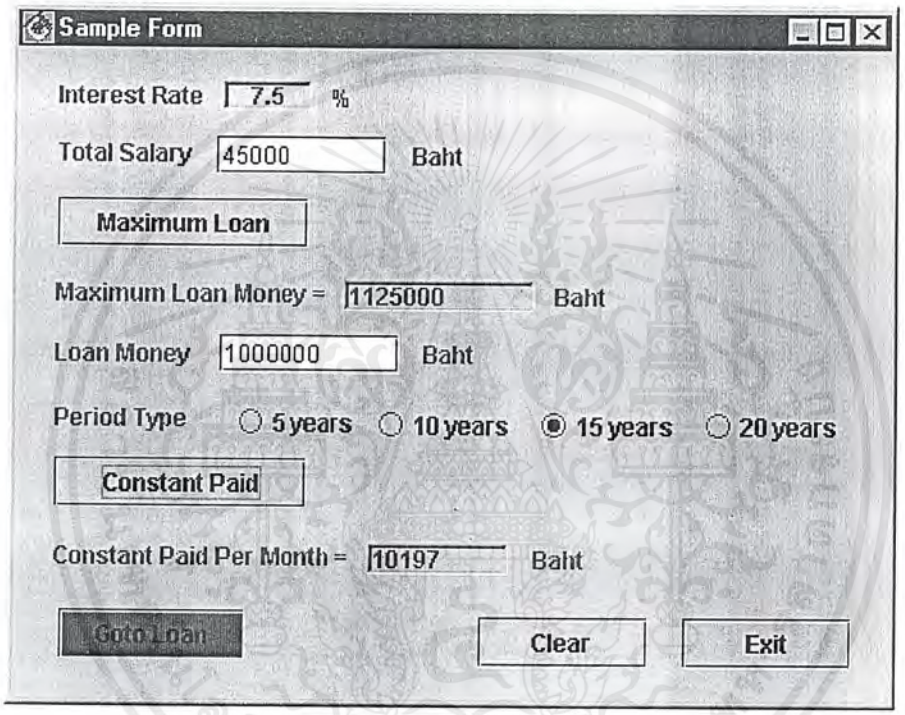
รูปที่ 3 User Logon

ถ้าหากผู้ใช้ป้อนรหัสผ่าน, รหัสประจำตัวผู้ใช้ หรือสาขาที่ผู้ใช้สังกัดอยู่ผิด จะมีข้อความว่า invalid username/password ในช่อง Status และยังไม่สามารถเข้าสู่ระบบได้

เมื่อผู้ใช้เข้าสู่ระบบได้แล้ว จะปรากฏชื่อของผู้ใช้ในช่อง Banker Name และจะสามารถใช้งานฟังก์ชันต่างๆ ที่แสดงบนเมนูหลักได้

3.3 Show Sample Form

เมื่อคลิกที่ปุ่ม Show Sample Form บนเมนูหลัก จะปรากฏฟอร์มแสดงตัวอย่างวงเงินกู้และเงินชำระต่องวด ดังแสดงในรูปที่ 4



รูปที่ 4 Sample Form

ฟังก์ชันงานคำนวณวงเงินกู้สูงสุดที่ลูกค้าสามารถขอกู้ได้จะพิจารณาจากรายได้ครอบครัวต่อเดือนของลูกค้า และสามารถประมาณเงินผ่อนชำระต่องวดตามจำนวนเงินกู้และระยะเวลาการผ่อนชำระ เพื่อช่วยในการตัดสินใจของลูกค้า ซึ่งลูกค้าสามารถเลือกได้ว่าจะผ่อนชำระแบบ 5 ปี, 10 ปี, 15 ปี หรือ 20 ปี โดยคิดดอกเบี้ยตามอัตราที่ทางธนาคารกำหนดในปัจจุบัน

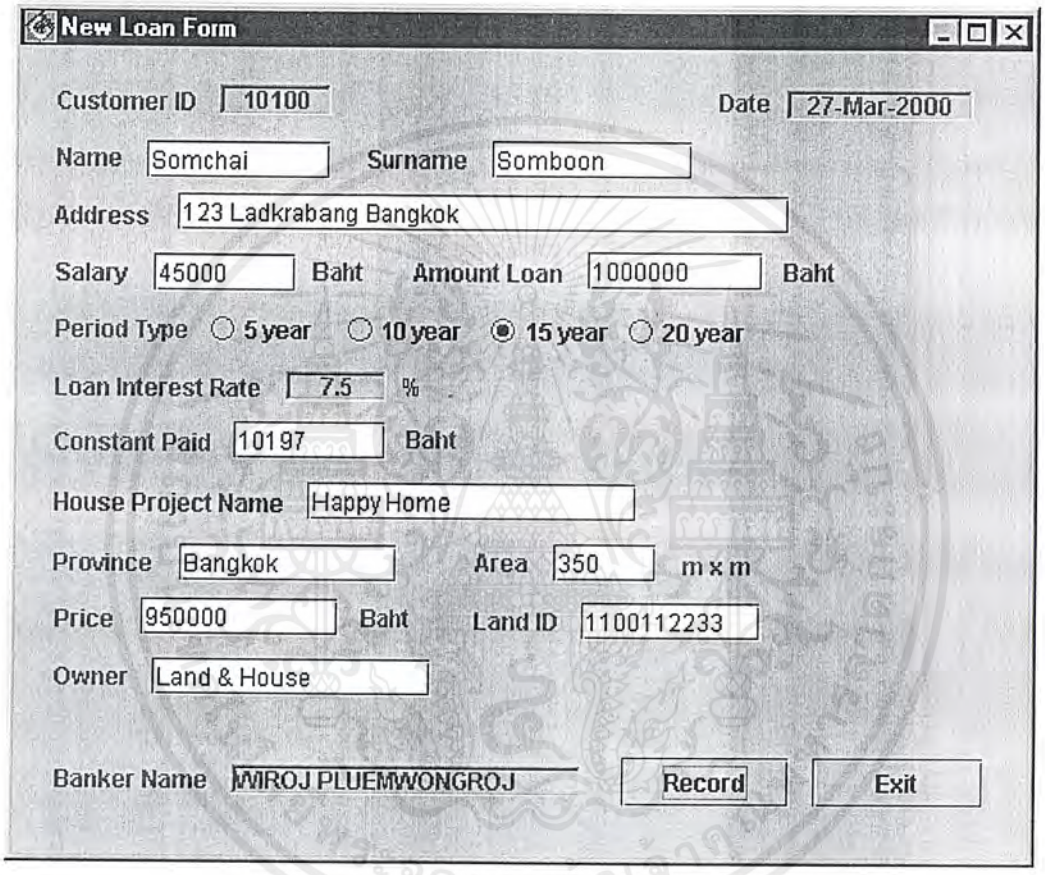
อัตราดอกเบี้ยเงินกู้ปัจจุบันจะแสดงอยู่ในช่อง Interest Rate ทำการป้อนจำนวนรายได้ครอบครัวต่อเดือนของลูกค้าในช่อง Total Salary แล้วคลิกที่ Maximum Loan จะปรากฏจำนวนวงเงินกู้สูงสุดที่ลูกค้าสามารถขอกู้ได้ในช่อง Maximum Loan Money

ป้อนจำนวนเงินที่ลูกค้าต้องการขอกู้ในช่อง Loan Money โดยจำนวนเงินที่ป้อนต้องไม่เกินจำนวนวงเงินกู้สูงสุด เลือกระยะเวลาการผ่อนชำระจากช่อง Period Type แล้วคลิกที่ Constant Paid จะ

แสดงจำนวนเงินชำระคงที่ขั้นต่ำต่องวดที่ลูกค้าต้องชำระ ตามระยะเวลาการผ่อนชำระที่ลูกค้าเลือกในช่อง Constant Paid Per Month

สามารถทำการลบค่าทั้งหมดที่แสดงบนฟอร์มได้โดยคลิกที่ปุ่ม Clear และถ้าต้องการกลับไปที่เมนูหลักคลิกที่ปุ่ม Exit

หากลูกค้าตกลงขอกู้เงินและคำร้องของลูกค้าได้รับการอนุมัติ สามารถคลิกที่ปุ่ม Goto Loan เพื่อทำการบันทึกประวัติลูกค้า, เปิดบัญชีเงินกู้ใหม่ และบันทึกรายละเอียดโครงการบ้าน ไว้ในฐานข้อมูลกลางของระบบ ฟอร์มเปิดการขอกู้ใหม่แสดงในรูปที่ 5



รูปที่ 5 New Loan Form

3.4 Record Customer

เมื่อคลิกที่ปุ่ม Record Customer บนเมนูหลักจะปรากฏฟอร์ม New Loan Form เพื่อทำการเปิดการขอกู้ใหม่ ฟอร์มนี้มีลักษณะเช่นเดียวกับฟอร์มที่ปรากฏจากการคลิกปุ่ม Goto Loan ในฟอร์ม Sample Form แตกต่างกันตรงที่หากเลือกเปิดฟอร์มนี้จาก Sample Form จะมีรายละเอียดของบัญชีเงินกู้ตามที่ลูกค้าได้เลือกไว้แล้วปรากฏให้ทันที ไม่ต้องทำการป้อนเข้าไปใหม่ แต่หากเลือกจาก Record Customer จะเป็นฟอร์มเปล่าให้กรอกข้อมูลเอง

เริ่มแรกในช่อง Customer ID จะว่าง เนื่องจากจะได้รหัสประจำตัวลูกค้าหลังจากทำการบันทึกรายละเอียดต่างๆลงในฐานข้อมูลแล้ว ในช่องอื่นๆ ของฟอร์มประกอบด้วย

Name และ Surname	: ชื่อ – นามสกุลของลูกค้า
Address	: ที่อยู่ปัจจุบันที่สามารถติดต่อได้ของลูกค้า
Salary	: รายได้ครอบครัวต่อเดือนของลูกค้า
Amount Loan	: จำนวนเงินที่ลูกค้าขอกู้
Period Type	: ระยะเวลาการผ่อนชำระ
Loan Interest Rate	: อัตราดอกเบี้ยเงินกู้ปัจจุบัน
Constant Paid	: เงินชำระต่องวดขั้นต่ำ
House Project Name	: ชื่อโครงการบ้านจัดสรร
Province	: จังหวัดที่ตั้งของโครงการบ้าน
Area	: เนื้อที่ (ตารางเมตร)
Price	: ราคาบ้านจัดสรร
Land ID	: หมายเลขโฉนดที่ดินของบ้านจัดสรร
Owner	: เจ้าของโครงการบ้านจัดสรร
Banker Name	: เจ้าหน้าที่ธนาคารผู้ทำรายการ

เมื่อป้อนข้อมูลเสร็จแล้วทำการกดปุ่ม Record โปรแกรมจะบันทึกรายละเอียดต่างๆ ลงฐานข้อมูล และจะแสดงเลขรหัสประจำตัวลูกค้า ซึ่งลูกค้าจะต้องใช้รหัสนี้ในการติดต่อครั้งต่อๆ ไปกับทางธนาคาร หากกดปุ่ม Exit ที่ฟอร์ม New Loan นี้จะกลับสู่เมนูหลักและยังไม่มีการบันทึกข้อมูลใดๆ ของลูกค้าลงในฐานข้อมูล

3.5 Get Payment Per Month

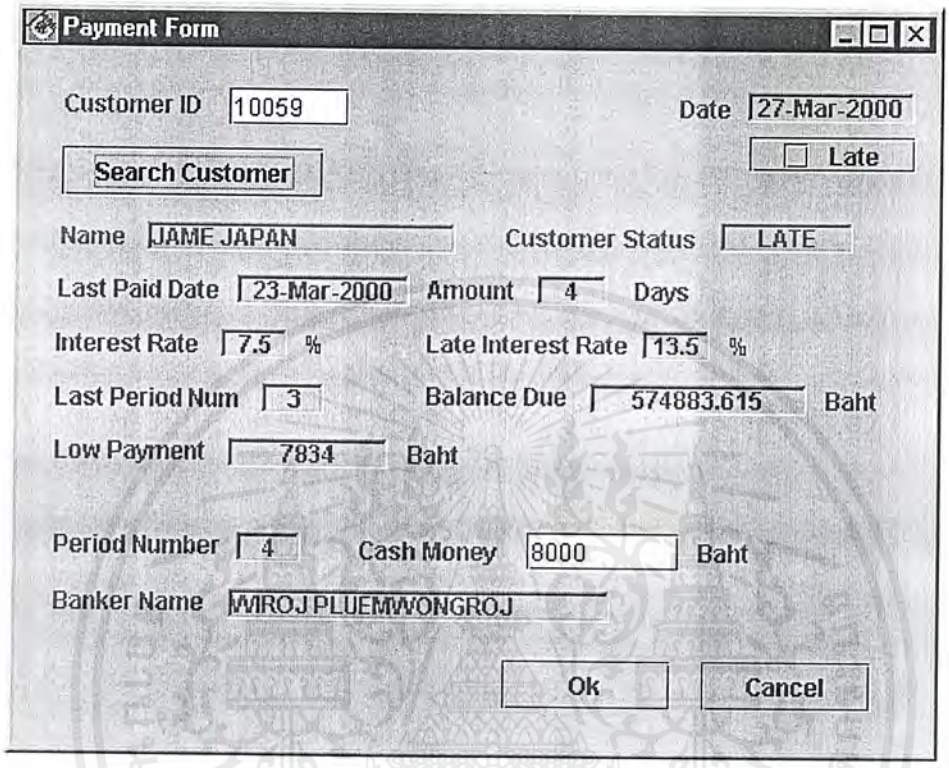
เมื่อลูกค้ามาขอชำระเงินผ่อนแต่ละงวด เลือก Get Payment Per Month จากเมนูหลัก จะปรากฏฟอร์ม Payment Form ดังรูปที่ 6 ให้ป้อนรหัสประจำตัวของลูกค้าลงในช่อง Customer ID แล้วคลิก Search Customer จะแสดงชื่อ, สถานะการชำระเงิน และรายละเอียดของเงินที่ต้องชำระในงวดนี้ของลูกค้า

Last Paid Date	: วันที่ชำระเงินครั้งล่าสุด
Amount	: จำนวนวันนับจากวันที่ชำระครั้งล่าสุด
Interest Rate	: อัตราดอกเบี้ยเงินกู้ปัจจุบัน
Late Interest Rate	: อัตราดอกเบี้ยปรับในกรณีที่ชำระเงินล่าช้า
Last Period Number	: งวดชำระเงินครั้งล่าสุด
Balance Due	: จำนวนเงินต้นคงเหลือ
Low Payment	: จำนวนเงินที่ลูกค้าต้องชำระขั้นต่ำในงวดนี้ หากล่าช้าก็จะรวมเงินส่วนดอกเบี้ยอัตราปรับด้วย

Period Number : งวดชำระเงินครั้งนี้

Cash Money : จำนวนเงินสดที่ลูกค้าชำระ อาจมากกว่าจำนวนเงินชำระขั้นต่ำได้

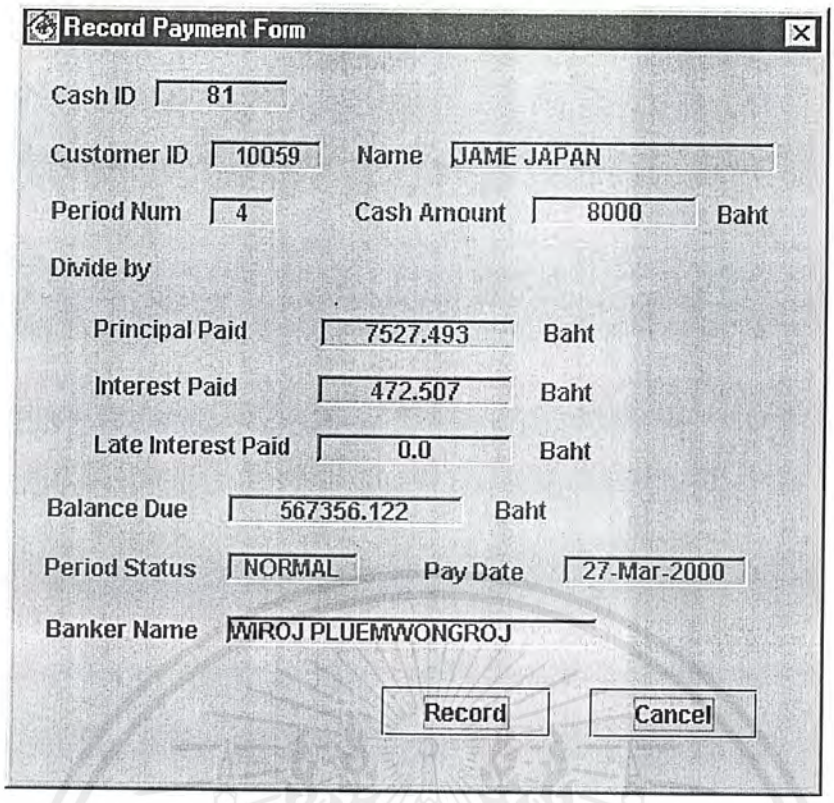
Banker Name : เจ้าหน้าที่ธนาคารผู้รับชำระเงิน



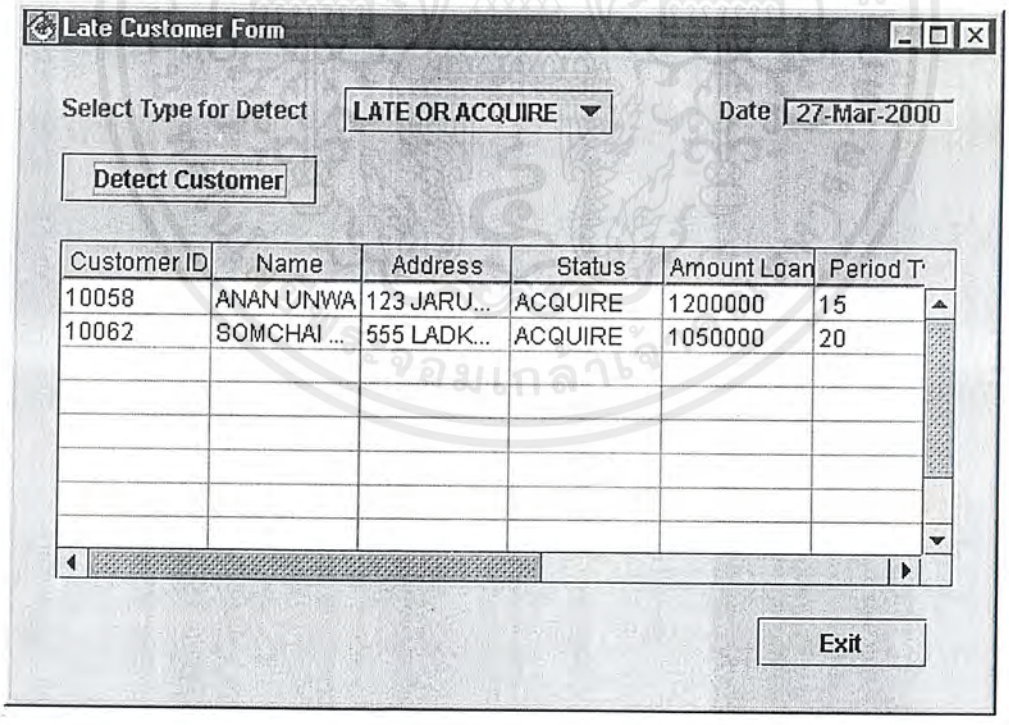
รูปที่ 6 Payment Form

เมื่อกรอกข้อมูลที่ต้องการครบแล้ว กดปุ่ม OK จะปรากฏฟอร์มแสดงรายการชำระเงินดังรูปที่ 7 ซึ่งจะแจกแจงรายละเอียดต่างๆ ของงวดการชำระเงินนี้ แบ่งเป็น Period Number (งวดการชำระเงินที่), Cash Amount (จำนวนเงินที่ลูกค้าชำระ), Principal Paid (ส่วนเงินต้น), Interest Paid (ส่วนดอกเบี้ยเงินกู้), Late Interest Paid (ส่วนดอกเบี้ยอัตราปรับ), Balance Due (เงินต้นคงเหลือ), Period Status (สถานะการชำระเงินในงวดนี้), Pay Date (วันที่ชำระเงิน)

กดปุ่ม Record เพื่อทำการบันทึกรายละเอียดการชำระเงินนี้เก็บไว้ในฐานข้อมูล แต่หากกดปุ่ม Cancel จะยกเลิกรายการและกลับสู่เมนูหลัก



รูปที่ 7 Record Payment Form



รูปที่ 8 Late Customer Form

3.6 Detect Late Payment

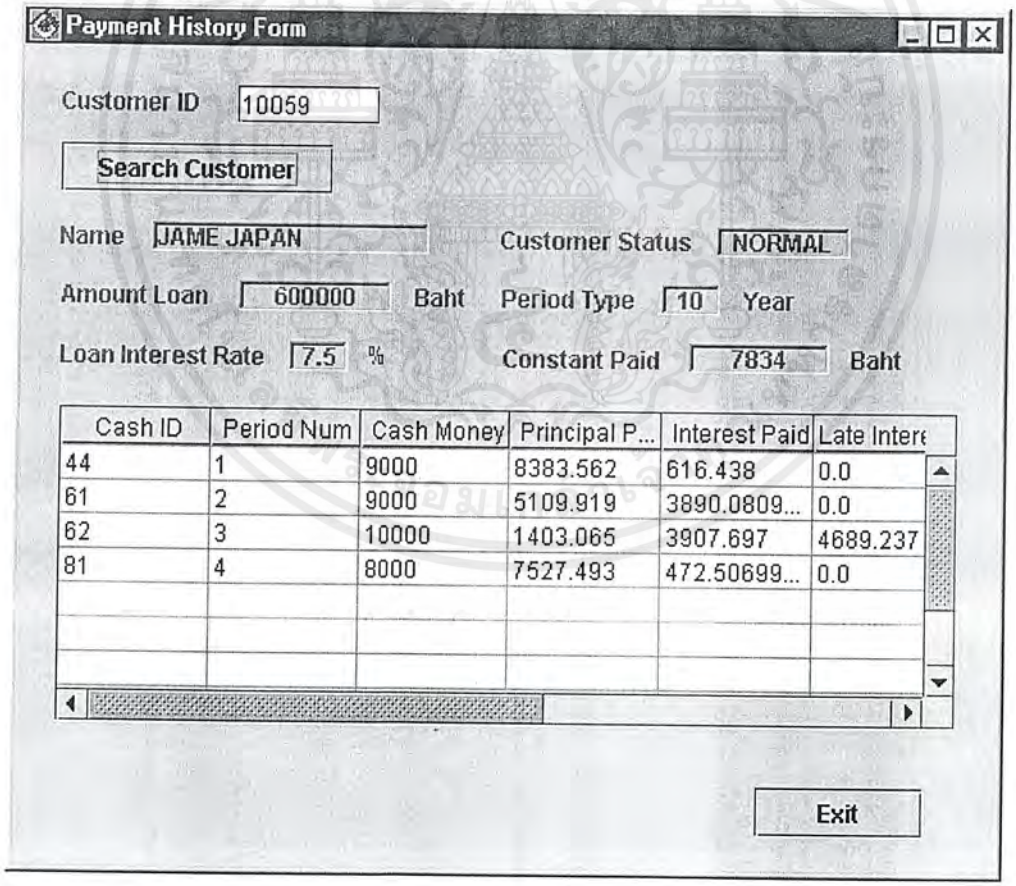
หากต้องการตรวจสอบรายชื่อลูกค้าที่ชำระเงินล่าช้าทำได้โดยเลือก Detect Late Payment จากเมนูหลัก จะปรากฏฟอร์ม Late Customer Form ดังรูปที่ 8 เลือกประเภทการล่าช้าได้จาก Select Type of Detect ว่าจะหารายชื่อลูกค้าที่ชำระเงินล่าช้าอย่างเดียว (Late Only) , ล่าช้าหรือถูกยึด (Late or Acquire) หรือถูกยึด (Acquire Only) จากนั้นกดปุ่ม Detect Late Customer จะปรากฏรายชื่อลูกค้าที่มีสถานะตรงกับเงื่อนไขที่เลือกไว้ พร้อมทั้งแสดงประวัติและรายละเอียดบัญชีเงินกู้ของลูกค้าด้วย

กด Exit เพื่อกลับสู่เมนูหลัก

3.7 Show Payment History

เจ้าหน้าที่ธนาคารสามารถตรวจสอบประวัติการชำระเงินของลูกค้าโดยเลือก Show Payment History จากเมนูหลัก จะปรากฏฟอร์ม Payment History Form ดังรูปที่ 9 ป้อนรหัสประจำตัวลูกค้าที่ต้องการตรวจสอบลงในช่อง Customer ID แล้วกด Search Customer จะแสดงชื่อ, สถานะ, รายละเอียดของบัญชีเงินกู้และประวัติการชำระเงินในงวดต่างๆ ที่ผ่านมาของลูกค้า

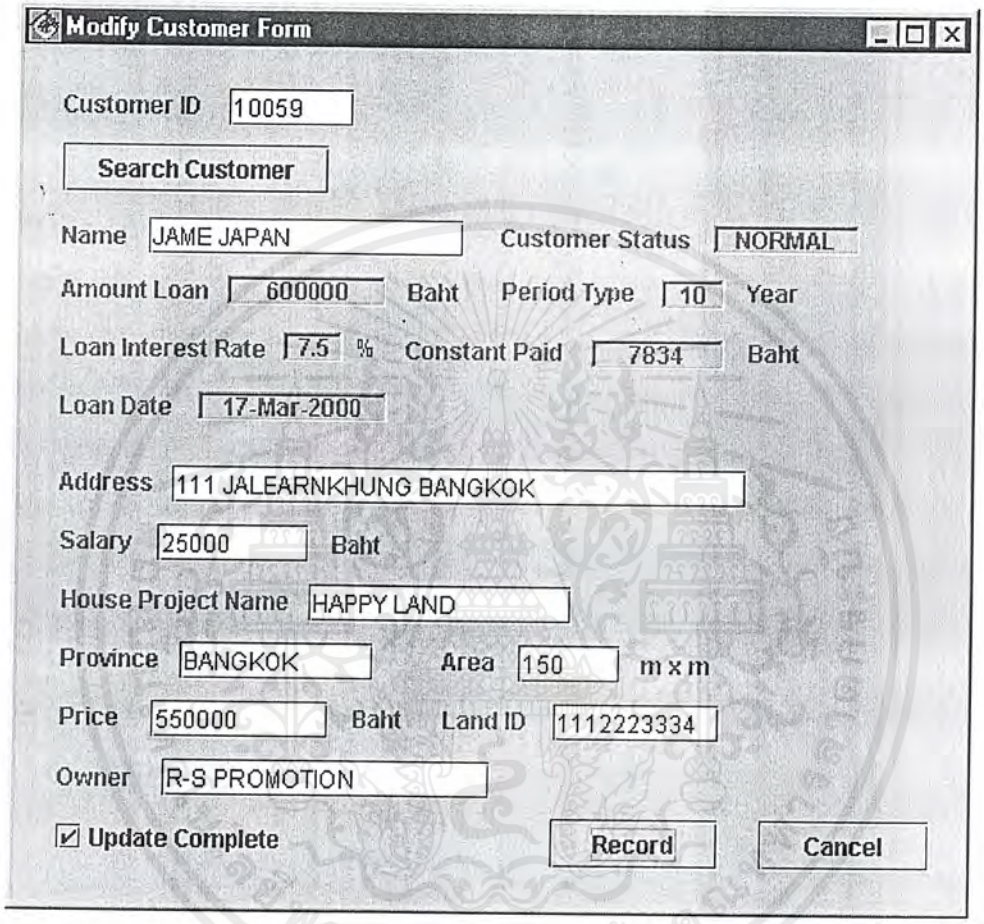
กด Exit เพื่อกลับสู่เมนูหลัก



รูปที่ 9 Payment History Form

3.8 Edit Customer Detail

หากลูกค้ามีการเปลี่ยนแปลงข้อมูลประวัติ เช่น ชื่อ, นามสกุล, ที่อยู่ เป็นต้น เจ้าหน้าที่ธนาคารสามารถทำการแก้ไขประวัติลูกค้าได้โดยเลือก Edit Customer Detail จากเมนูหลัก จะปรากฏฟอร์ม Modify Customer Form ดังรูปที่ 10 ป้อนรหัสประจำตัวลูกค้าแล้วกดปุ่ม Search Customer แล้วฟอร์มจะแสดงประวัติ, รายละเอียดบัญชีเงินกู้ และรายละเอียดโครงการบ้านจัดสรรของลูกค้า ส่วนที่สามารถแก้ไขได้จะอยู่ในช่องสีขาว



รูปที่ 10 Modify Customer Form

เมื่อทำการแก้ไขเสร็จแล้วกดปุ่ม Record จะทำการบันทึกข้อมูลใหม่นี้ลงในฐานข้อมูล เมื่อบันทึกเสร็จเรียบร้อยแล้วจะมีสัญลักษณ์แสดงที่ Update Complete หากต้องการยกเลิกการแก้ไขนี้ให้กดปุ่ม Cancel จะกลับสู่เมนูหลัก

ภาคผนวก จ
คู่มือโปรแกรมเมอร์
โปรแกรมระบบเงินกู้เพื่อการผ่อนบ้านจัดสรร

คู่มือสำหรับโปรแกรมเมอร์

1. โครงสร้างของโปรแกรม

1.1 โปรแกรมที่สร้างจาก Oracle

เราได้ใช้ Oracle สร้าง Store Procedure และ Store Function เพื่อสนับสนุนการทำงานของโปรแกรมหลัก โดยจะประกอบด้วย

Procedure

- ADD_NEW_ACCOUNT
- PAST_UPDATE

Function

- CHECK LATE
- CHECK USER
- GET_BRANCH_ID
- GET_CURRENT_INT
- GET_FACTOR
- GET_INT_PAID
- GET_LATE_INT
- GET_LATE_INT_PAID
- GET_LOW_PAYMENT
- GET_MAX_LOAN
- GET_NUM_LATE
- GET_SYSDATE
- LAST_PAY_DATE
- STATUS_PAID

Store Procedure และ Store Function ต่างๆ เหล่านี้ เราได้ทำการ export มาจาก Oracle เก็บเป็นไฟล์ชื่อ HousingLoanDB.dmp โดยไฟล์นี้จะรวมทั้ง Tables และ Sequence ร่วมอยู่ด้วย

1.2 โปรแกรมที่สร้างจาก VisualAge

เราได้สร้างคลาสจาก VisualAge จำนวน 7 คลาส โดยเก็บเป็นไฟล์จุดจาวา ดังนี้

- BankerOption.java
- LateCustomerForm.java
- ModifyCustomerForm.java
- NewLoanForm.java

- PaymentForm.java
- PaymentHistoryForm.java
- SampleForm.java

ในแต่ละไฟล์จะถูกแปลงไปเป็นไฟล์จุดคลาสในขณะคอมไพล์ แต่ละคลาสประกอบด้วย Fields และ Methods ที่สำคัญต่างๆ ดังนี้

1.2.1 Class BankerOption

Fields :

- bankerID	Int	เลขรหัสนายธนาคาร
- bankerName	String	ชื่อนายธนาคาร
- password	String	รหัสผ่าน
- bankerStatus	String	สถานะของนายธนาคาร
- branchName	String	ชื่อสาขาของธนาคาร
- today	String	วันที่ปัจจุบัน

Methods :

- public void adminLogon(java.awt.event.MouseEvent mouseEvent)
- public void bankerOption_Initialize()
- public void checkUserPassword()
- public void closeAbout()
- public void closeUserPassword()
- public void connectDatabase()
- public void ExitApplication()
- public void initializeChoices(com.sun.java.swing.JComboBox myChoices)
- public void openAbount()
- public void showCenter(java.awt.Component component)
- public void showLateCustomerForm()
- public void showLogonForm()
- public void showMainMenu()
- public void showModifyCustomerForm()
- public void showNewLoanForm()
- public void showPaymentForm()
- public void showPaymentHistoryForm()
- public void showSampleForm()
- public void userLogon(java.awt.event.MouseEvent mouseEvent)

1.2.2 Class LateCustomerForm

Fields :

- lateTableValue String[10][10] ตารางแสดงลูกค้าที่มีสถานะล่าช้า

Methods :

- public void addComboBoxItem()
- public void createColumnModel()
- public void detectCustomerLate()
- public void exitForm()
- public void initialLateCustomerForm()
- public void showForm()
- public void showLateTable()

1.2.3 Class ModifyCustomerForm

Fields :

- customerID Int เลขรหัสลูกค้า
- houseID Int เลขรหัสโครงการบ้าน
- customerName String ชื่อลูกค้า
- address String ที่อยู่ลูกค้า
- salary Int จำนวนรายได้ครอบครัวต่อเดือน
- houseName String ชื่อโครงการบ้าน
- province String จังหวัด
- area Int จำนวนพื้นที่ของบ้าน
- price Int จำนวนราคาบ้าน
- landID Int เลขโฉนดที่ดิน
- owner String ชื่อบริษัทเจ้าของโครงการบ้าน

Methods :

- public void exitForm()
- public void initialModifyCustomerForm()
- public void recordModifyCustomer()
- public void searchCustomerToModify()
- public void showForm()

1.2.4 Class NewLoanForm

Fields :

-	customerID	Int	เลขรหัสลูกค้า
-	customerName	String	ชื่อลูกค้า
-	address	String	ที่อยู่ลูกค้า
-	salary	Int	จำนวนรายได้ครอบครัวต่อเดือน
-	amountLoan	Int	จำนวนเงินกู้
-	periodType	Int	ระยะเวลาการผ่อนชำระ
-	interestRate	Double	อัตราดอกเบี้ยปัจจุบัน
-	constantPaid	Int	จำนวนเงินชำระขั้นต่ำต่อเดือน
-	houseName	String	ชื่อโครงการบ้าน
-	province	String	จังหวัด
-	area	Int	จำนวนพื้นที่ของบ้าน
-	price	Int	จำนวนราคาบ้าน
-	landID	Int	เลขโฉนดที่ดิน
-	owner	String	ชื่อบริษัทเจ้าของโครงการบ้าน

Methods :

- public void exitForm()
- public void initialNewLoanForm()
- public void parameterOfNewLoan(int totalSalary,int loanMoney,int period,int constant)
- public void recordNewLoan()
- public void setType5()
- public void setType10()
- public void setType15()
- public void setType20()
- public void showForm()

1.2.5 Class PaymentForm

Fields :

-	customerID	Int	เลขรหัสลูกค้า
-	late	Boolean	สถานะการชำระเงินล่าช้า
-	customerName	String	ชื่อลูกค้า
-	customerStatue	String	สถานะการชำระเงินของลูกค้า

- lastPaidDate String วันชำระเงินครั้งหลังสุด
- amountDay Int จำนวนวันที่ขาดการชำระ
- interestRate Double อัตราดอกเบี้ยปกติ
- lateIntRate Double อัตราดอกเบี้ยค่าปรับ
- lastPeriodNum Int งวดการชำระครั้งหลังสุด
- balanceDue Double ยอดเงินคงเหลือ
- lowPayment Int จำนวนเงินชำระขั้นต่ำ
- periodNum Int งวดการชำระปัจจุบัน
- cashMoney Int จำนวนเงินชำระประจำงวด
- principalPaid Double เงินชำระส่วนเงินต้น
- intPaid Double เงินชำระส่วนดอกเบี้ยปกติ
- lateIntPaid Double เงินชำระส่วนดอกเบี้ยค่าปรับ
- periodStatus String สถานะการชำระเงินประจำงวด
- constPaid Int จำนวนเงินชำระคงที่ต่อเดือน

Methods :

- public void exitForm()
- public void exitRecordPaymentForm()
- public void initialPaymentForm()
- public void payCash()
- public void recordCashPerMonth()
- public void searchCustomer()
- public void showForm()
- public void showRecordPaymentForm()
- public void updateCustomerStatus()

1.2.6 Class PaymentHistoryForm

Fields :

- historyTableValue String[10][10] ตารางแสดงประวัติการชำระเงิน

Methods :

- public void createHistoryColumnModel()
- public void exitForm()
- public void initialPaymentHistoryForm()
- public void searchCustomerHistomer()
- public void showForm()

- public void showHistoryTable()

1.2.7 Class SampleForm

Fields :

-	interestRate	Double	อัตราดอกเบี้ยปัจจุบัน
-	totalSalary	Int	รายได้ครอบครัวต่อเดือน
-	maxLoan	Int	จำนวนวงเงินกู้สูงสุด
-	loanMoney	Int	จำนวนเงินกู้
-	periodType	Int	ระยะเวลาการผ่อนชำระ
-	constantPaid	Int	จำนวนเงินชำระคงที่ต่อเดือน
-	multipleLoan	Int	ตัวคูณจำนวนวงเงินกู้สูงสุด

Methods :

- public void calConstPaid()
- public void calMaxLoan()
- public void clearSampleForm()
- public void exitForm()
- public void gotoLoan()
- public void setType5()
- public void setType10()
- public void setType15()
- public void setType20()
- public void showForm()
- public void showInterestRate()

2. การทำงานของโปรแกรมย่อย

2.1 การทำงานของโปรแกรมย่อยที่สร้างจาก Oracle

Procedure ADD_NEW_ACCOUNT

Description	เป็น Store Procedure สำหรับการ insert รายการบัญชีเงินกู้ของลูกค้าใหม่ที่มาขอกู้เงินกับทางธนาคาร ลงในตาราง Loanaccount	
Parameter Input	br_name	ชื่อสาขาของธนาคาร
	amount	จำนวนเงินกู้
	ptype	ประเภทระยะเวลาการผ่อนชำระ
	const_paid	จำนวนเงินชำระคงที่ต่อเดือน
	bk_id	เลขรหัสนายธนาคารผู้อนุมัติ

Parameter Output	-	

Procedure PAST_UPDATE

Description	เป็น Store Procedure สำหรับการ insert รายการอัตราดอกเบี้ยปัจจุบัน ซึ่งได้แก่ อัตราดอกเบี้ยปกติ อัตราดอกเบี้ยค่าปรับ ตัวคูณจำนวนวงเงินกู้สูงสุด และวันที่บันทึก ลงไปในตาราง Ref_Table	
Parameter Input	cur_int	อัตราดอกเบี้ยปกติขณะปัจจุบัน
	cur_late	อัตราดอกเบี้ยค่าปรับขณะปัจจุบัน
	maxloan	ตัวคูณจำนวนวงเงินกู้สูงสุด
	mydate	วันที่บันทึก
Parameter Output	-	

Function CHECK_LATE

Description	เป็น Store Function สำหรับตรวจสอบสถานะการชำระเงินของลูกค้าว่าล่าช้าหรือไม่ หากลูกค้ามาชำระเงินก่อนสิ้นเดือนของงวดปัจจุบันก็จะมีสถานะการชำระล่าช้าเป็นเท็จ แต่หากลูกค้ามาชำระเงินหลังจากสิ้นเดือนประจำงวดนั้นๆ ก็จะมีสถานะการชำระล่าช้าเป็นจริง	
Parameter Input	c_id	เลขรหัสลูกค้า
	acc_id	เลขรหัสบัญชีเงินกู้
Parameter Output	'true' / 'false'	สถานะการชำระเงินล่าช้า

Function CHECK_USER

Description	เป็น Store Function สำหรับตรวจสอบรหัสผ่านของผู้ใช้โปรแกรม โดยจะตรวจสอบจากรหัสนายธนาคาร รหัสผ่าน และสาขาที่นายธนาคารประจำอยู่ หากรหัสที่ผู้ใช้ป้อนเข้ามาตรงกับข้อมูลในตาราง Banker ผู้ใช้คนนั้นก็จะสามารถใช้งานโปรแกรมได้	
Parameter Input	b_id	เลขรหัสนายธนาคาร
	pword	รหัสผ่าน
	b_status	สถานะของนายธนาคาร
	branch_n	ชื่อสาขาที่นายธนาคารประจำอยู่
Parameter Output	'true' / 'false'	แสดงถึงรหัสผ่านถูกต้องหรือไม่

Function GET_BRANCH_ID

Description	สำหรับแปลงชื่อสาขาธนาคาร ไปเป็นรหัสสาขา

Parameter Input	br_name	ชื่อสาขาของธนาคาร
Parameter Output	branch_id	รหัสสาขาของธนาคาร

Function GET_CURRENT_INT

Description	select อัตราดอกเบี้ยปกติขณะปัจจุบันจากตาราง Ref_Table	
Parameter Input	-	
Parameter Output	current_int_rate	อัตราดอกเบี้ยปกติขณะปัจจุบัน

Function GET_FACTOR

Description	select factor สำหรับการคำนวณจำนวนเงินชำระคงที่ต่อเดือน โดยค่า factor ที่ได้จะขึ้นอยู่กับอัตราดอกเบี้ยปัจจุบัน และระยะเวลาการผ่อนชำระ ซึ่งจะเก็บอยู่ในตาราง Factor	
Parameter Input	int	อัตราดอกเบี้ยเงินกู้ปัจจุบัน
	period	ระยะเวลาการผ่อนชำระ
Parameter Output	loan_factor	factor สำหรับการคำนวณเงินกู้

Function GET_INT_PAID

Description	สำหรับคำนวณจำนวนเงินชำระส่วนดอกเบี้ยปกติ	
Parameter Input	balance_due	ยอดเงินคงเหลือ
	int_rate	อัตราดอกเบี้ยปกติ
	late_int_rate	อัตราดอกเบี้ยค่าปรับ
	last_pay	วันที่ชำระครั้งหลังสุด
	late	สถานะการชำระงวดที่แล้ว
Parameter Output	int_mon	จำนวนเงินชำระส่วนดอกเบี้ยปกติ

Function GET_LATE_INT

Description	select อัตราดอกเบี้ยค่าปรับขณะปัจจุบันจากตาราง Ref_Table	
Parameter Input	-	
Parameter Output	current_late_int	อัตราดอกเบี้ยค่าปรับขณะปัจจุบัน

Function GET_LATE_INT_PAID

Description	สำหรับคำนวณจำนวนเงินชำระส่วนดอกเบี้ยค่าปรับ	
Parameter Input	balance_due	ยอดเงินคงเหลือ
	int_rate	อัตราดอกเบี้ยปกติ

	late_int_rate	อัตราดอกเบี้ยค่าปรับ
	last_pay	วันที่ชำระครั้งหลังสุด
	late	สถานะการชำระงวดที่แล้ว
Parameter Output	late_mon	จำนวนเงินชำระส่วนดอกเบี้ยค่าปรับ

Function GET_LOW_PAYMENT

Description	สำหรับคำนวณจำนวนเงินชำระขั้นต่ำประจำงวด หากลูกค้ามาชำระตรงตามเวลา จำนวนเงินชำระขั้นต่ำนี้จะเท่ากับจำนวนเงินชำระคงที่ต่อเดือน แต่หากลูกค้ามาชำระล่าช้า จำนวนเงินชำระจะต้องทำการคำนวณตามระยะเวลาที่ล่าช้า	
Parameter Input	c_id	เลขรหัสลูกค้า
	acc_id	เลขรหัสบัญชีเงินกู้
	balance_due	ยอดเงินคงเหลือ
	int_rate	อัตราดอกเบี้ยปกติ
	late_int_rate	อัตราดอกเบี้ยค่าปรับ
	last_pay	วันที่ชำระครั้งหลังสุด
	const_pay	จำนวนเงินชำระคงที่ต่อเดือน
Parameter Output	low_money	จำนวนเงินชำระขั้นต่ำประจำงวด

Function GET_MAX_LOAN

Description	select ตัวคูณจำนวนวงเงินกู้สูงสุด จากตาราง Ref_Table	
Parameter Input	-	
Parameter Output	max_loan	ตัวคูณจำนวนวงเงินกู้สูงสุด

Function GET_NUM_LATE

Description	สำหรับคำนวณหาจำนวนวันทั้งหมด ที่ลูกค้าค้างชำระ โดยพิจารณาจากวันที่ชำระปัจจุบัน กับวันที่ชำระครั้งหลังสุด	
Parameter Input	c_id	เลขรหัสลูกค้า
	acc_id	เลขรหัสบัญชีเงินกู้
Parameter Output	num_date	จำนวนวันทั้งหมด ที่ลูกค้าค้างชำระ

Function GET_SYSDATE

Description	อ่านค่าวันที่ปัจจุบันจากเครื่อง Oracle Server
Parameter Input	-

Parameter Output	today	วันที่ปัจจุบัน

Function LAST_PAY_DATE

Description	select วันที่ชำระครั้งหลังสุดของลูกค้าจากตาราง Cash_Per_Month แต่หากลูกค้ายังไม่เคยมาชำระเงินเลย ก็จะ select วันที่ขอกู้เงินจากตาราง Loanaccount	
Parameter Input	c_id	เลขรหัสลูกค้า
	acc_id	เลขรหัสบัญชีเงินกู้
Parameter Output	last_day	วันที่ชำระครั้งหลังสุด

Function STATUS_PAID

Description	select สถานะการชำระเงินครั้งหลังสุดจากตาราง Cash_Per_Month แต่หากลูกค้ายังไม่เคยมาชำระ ก็ select สถานะการชำระเงินของลูกค้าจากตาราง Customer	
Parameter Input	c_id	เลขรหัสลูกค้า
Parameter Output	status	สถานะการชำระเงิน

2.2 การทำงานของโปรแกรมย่อยที่สร้างจาก VisualAge

Class BankerOption

public void adminLogon(java.awt.event.MouseEvent mouseEvent)

รอรับเหตุการณ์จาก mouse แล้วจึงไปเรียก method showLogonForm() เพื่อแสดงฟอร์มป้อนรหัส

public void bankerOption_Initialize()

กำหนดค่าเริ่มต้นให้แก่ฟอร์ม BankerOption ทำการติดต่อกับฐานข้อมูล แสดงวันที่ปัจจุบัน และแสดงสถานะการติดต่อกับฐานข้อมูลออราเคิล

public void checkUserPassword()

ตรวจสอบรหัสผ่านของผู้ใช้ที่ป้อนเข้าสู่ฟอร์ม โดยจะไปเรียก Store Function CHECK_USER เพื่อตรวจสอบข้อมูลกับฐานข้อมูลว่าตรงกันหรือไม่ หากรหัสผ่านถูกต้องก็จะ select ชื่อนายธนาคารและแสดงขึ้นที่ฟอร์ม แต่ถ้ารหัสไม่ถูกต้องก็จะไม่สามารถใช้โปรแกรมได้

public void closeAbout()

ปิดฟอร์ม About และเซตการทำงานให้แก่ฟอร์ม BankerOption

public void closeUserPassword()

ปิดฟอร์ม UserPassword และเซตการทำงานให้แก่ฟอร์ม BankerOption

public void connectDatabase()

เปิดการติดต่อกับฐานข้อมูลออราเคิล โดยจะต้องป้อนเลข IP , ชื่อผู้ติดต่อ และรหัสผ่าน โดยในโปรแกรมของโครงงานนี้ได้กำหนดค่าต่างๆ ดังนี้

IP	:	161.246.6.117
User name	:	dear
Password	:	dear

public void ExitApplication()

ออกจากโปรแกรมระบบ

public void initializeChoices(com.sun.java.swing.JComboBox myChoices)

ติดต่อกับฐานข้อมูล แล้ว select ชื่อสาขาต่างๆ จากตาราง Bank นำมาใส่ใน ComboBox เพื่อแสดงถึงสาขาต่างๆ ของธนาคารให้ผู้ใช้เลือก

public void openAbout()

ปิดการทำงานของฟอร์ม BankerOption แล้วเปิดฟอร์ม About

public void showCenter(java.awt.Component component)

คำนวณ และกำหนดตำแหน่งของฟอร์มใหม่ เพื่อให้สามารถแสดงฟอร์มที่ตรงกลางหน้าจอได้

public void showLateCustomerForm()

ซ้อนฟอร์ม BankerOption แล้วเรียกแสดงฟอร์ม LateCustomer

public void showLogonForm()

ปิดการทำงานของฟอร์ม BankerOption แล้วเปิดฟอร์ม Logon สำหรับป้อนรหัสผ่านของผู้ใช้

public void showMainMenu()

เซตให้แสดงฟอร์ม BankerOption (MainMenu)

public void showModifyCustomerForm()

ซ้อนฟอร์ม BankerOption แล้วเรียกแสดงฟอร์ม ModifyCustomer

public void showNewLoanForm()

ซ้อนฟอร์ม BankerOption แล้วเรียกแสดงฟอร์ม NewLoan

public void showPaymentForm()

ซ้อนฟอร์ม BankerOption แล้วเรียกแสดงฟอร์ม Payment

public void showPaymentHistoryForm()

ซ้อนฟอร์ม BankerOption แล้วเรียกแสดงฟอร์ม PaymentHistory

public void showSampleForm()

ซ้อนฟอร์ม BankerOption แล้วเรียกแสดงฟอร์ม Sample

public void userLogon(java.awt.event.MouseEvent mouseEvent)

รอรับเหตุการณ์จาก mouse แล้วจึงไปเรียก method showLogonForm() เพื่อแสดงฟอร์มป้อนรหัส

Class LateCustomerForm

public void addComboBoxItem()

กำหนดค่าเริ่มต้นให้แก่ ComboBox เป็น 'LATE ONLY' , 'ACQUIRE ONLY' , 'LATE OR ACQUIRE'

public void createColumnModel()

กำหนดค่าเริ่มต้นให้แก่ตารางแสดงรายการลูกค้าที่ชำระเงินล่าช้า

public void detectCustomerLate()

select รายการลูกค้าที่ชำระเงินล่าช้า ตามตัวเลือกที่ได้กำหนดไว้จาก ComboBox แล้วนำมาแสดงตามรายการต่างๆ ของตารางแสดงรายการลูกค้าที่ชำระเงินล่าช้า

public void exitForm()

ซ้อนการแสดงฟอร์ม LateCustomer แล้วเซตการทำงานไปยังฟอร์ม BankerOption

public void initialLateCustomerForm()

กำหนดค่าเริ่มต้นต่างๆ ให้แก่ฟอร์ม LateCustomer

public void showForm()

แสดงฟอร์ม LateCustomer และเรียกเมทธอด initialLateCustomerForm() สำหรับกำหนดค่าเริ่มต้นให้แก่ฟอร์ม

public void showLateTable()

แสดงข้อมูลในตารางแสดงรายการลูกค้าที่ชำระเงินล่าช้าขึ้นบนฟอร์ม

Class ModifyCustomerForm

public void exitForm()

ซ้อนการแสดงฟอร์ม ModifyCustomer แล้วเซตการทำงานไปยังฟอร์ม BankerOption

public void initialModifyCustomerForm()

กำหนดค่าเริ่มต้นให้แก่ฟอร์ม ModifyCustomer

public void recordModifyCustomer()

รับข้อมูลที่ป้อนให้แก่ฟอร์ม ModifyCustomer แล้วทำการ update ข้อมูลที่ได้แก้ไขไปยังตาราง Customer และตาราง HouseProject หากทำการ update สำเร็จก็จะแสดงเครื่องหมายถูกที่ CheckBox

public void searchCustomerToModify()

ค้นหาข้อมูลส่วนตัวของลูกค้าจาก customer_id ที่ป้อนไปยังฟอร์ม หากพบข้อมูลในตาราง Customer ก็จะทำการแสดงรายการข้อมูลลูกค้าต่างๆ ไปยังฟอร์ม

public void showForm()

แสดงฟอร์ม ModifyCustomer และเรียกเมทธอด initialModifyCustomerForm() สำหรับกำหนดค่าเริ่มต้นให้แก่ฟอร์ม

Class NewLoanForm

public void exitForm()

ซ้อนการแสดงฟอร์ม NewLoan แล้วเซตการทำงานไปยังฟอร์ม BankerOption

public void initialNewLoanForm()

กำหนดค่าเริ่มต้นให้แก่ฟอร์ม NewLoan

public void parameterOfNewLoan(int totalSalary,int loanMoney,int period,int constant)

ส่งค่าพารามิเตอร์ต่างๆ จากฟอร์ม Sample มายังฟอร์ม NewLoan ซึ่งได้แก่ รายได้ครอบครัวต่อเดือน , จำนวนเงินกู้ , ระยะเวลาการผ่อนชำระ , จำนวนเงินชำระคงที่ต่อเดือน และแสดงค่าข้อมูลต่างๆ นี้ขึ้นที่ฟอร์ม NewLoan

public void recordNewLoan()

รับค่าข้อมูลต่างๆ ที่ป้อนให้แก่ฟอร์ม แล้วบันทึกข้อมูลเหล่านั้นลงในฐานข้อมูล โดยจะเรียก Store Procedure ADD_NEW_ACCOUNT สำหรับบันทึกบัญชีเงินกู้ลูกค้าใหม่ หากบันทึกได้ก็จะบันทึกข้อมูลโครงการบ้านลงในตาราง HouseProject และบันทึกข้อมูลลูกค้าลงในตาราง Customer จากนั้นจึงแสดงเลขรหัสลูกค้าขึ้นที่ฟอร์ม เพื่อแสดงถึงการบันทึกข้อมูลสำเร็จ แต่หากทำการบันทึกไม่สำเร็จก็จะทำ rollback ไปยังตำแหน่ง commit ครั้งหลังสุด

public void setType5()

เซตระยะเวลาการผ่อนชำระเป็นแบบ 5 ปี

public void setType10()

เซตระยะเวลาการผ่อนชำระเป็นแบบ 10 ปี

public void setType15()

เซตระยะเวลาการผ่อนชำระเป็นแบบ 15 ปี

public void setType20()

เซตระยะเวลาการผ่อนชำระเป็นแบบ 20 ปี

public void showForm()

แสดงฟอร์ม NewLoan และเรียกเมทธอด initialNewLoanForm() สำหรับกำหนดค่าเริ่มต้นให้แก่ฟอร์ม

Class PaymentForm

public void exitForm()

ซ้อนการแสดงฟอร์ม Payment แล้วเซตการทำงานไปยังฟอร์ม BankerOption

public void exitRecordPaymentForm()

เรียกแสดงฟอร์ม Payment และปิดฟอร์ม RecordPayment

public void initialPaymentForm()

กำหนดค่าเริ่มต้นให้แก่ฟอร์ม Payment

public void payCash()

รับการชำระเงินจากลูกค้า โดยรับจำนวนเงินชำระที่ป้อนให้แก่ฟอร์ม หากจำนวนเงินชำระสูงกว่าจำนวนเงินชำระขั้นต่ำของลูกค้าก็สามารถชำระได้ จากนั้นจะทำการคำนวณจำนวนเงินชำระออกเป็น ส่วนเงินต้น ส่วนเงินดอกเบี้ยปกติ ส่วนเงินดอกเบี้ยค่าปรับ และคำนวณหายอดเงินคงเหลือของลูกค้า

public void recordCashPerMonth()

ทำการบันทึกรายการชำระเงินประจำงวดของลูกค้า ไปเก็บในฐานข้อมูลตาราง Cash_Per_Month แล้วทำการ update สถานะการชำระของลูกค้าตามสถานะการชำระเงินประจำงวด หากสามารถทำการบันทึกข้อมูลได้ ก็จะแสดงเลขรหัสรายการชำระ (cash_id) ขึ้นที่ฟอร์ม

public void searchCustomer()

รับเลขรหัสลูกค้า (CustomerID) จากฟอร์ม แล้วทำการ select ข้อมูลลูกค้าจากตาราง Customer แสดงขึ้นที่ฟอร์ม

เรียก Store Function LAST_PAY_DATE เพื่อหาวันที่ชำระของลูกค้าครั้งหลังสุด

เรียก Store Function CHECK_LATE เพื่อตรวจสอบงวดการชำระว่าล่าช้าหรือไม่

เรียก Store Function GET_NUM_LATE เพื่อตรวจสอบหาจำนวนวันค้างชำระ

select รายการชำระเงินงวดที่แล้วจากตาราง Cash_Per_Month ได้แก่ งวดการชำระครั้งที่แล้ว และยอดเงินคงเหลือ

select ข้อมูลบัญชีเงินกู้ของลูกค้าจากตาราง Loanaccount

เรียก Store Function GET_LOW_PAYMENT เพื่อคำนวณจำนวนเงินชำระขั้นต่ำประจำงวด

เรียก Store Function GET_INT_PAID เพื่อคำนวณหาจำนวนเงินชำระส่วนดอกเบี้ยปกติ

เรียก Store Function GET_LATE_INT_PAID เพื่อคำนวณหาจำนวนเงินชำระส่วนดอกเบี้ยค่าปรับ

ปรับสถานะลูกค้าเป็น 'ACQUIRE' หากลูกค้ามาชำระเงินล่าช้ากว่า 90 วัน

public void showForm()

แสดงฟอร์ม Payment และเรียกเมทธอด initialPaymentForm() สำหรับกำหนดค่าเริ่มต้นให้แก่ฟอร์ม

public void showRecordPaymentForm()

แสดงฟอร์ม RacordPayment และกำหนดค่าเริ่มต้นของฟอร์ม

public void updateCustomerStatus()

update สถานะการชำระเงินของลูกค้าในตาราง Customer เป็น 'ACQUIRE' และแสดงสถานะขึ้นที่ฟอร์ม

Class PaymentHistoryForm

public void createHistoryColumnModel()

กำหนดค่าเริ่มต้นให้แก่ตารางแสดงประวัติการชำระเงิน ของลูกค้า

public void exitForm()

ซ้อนการแสดงฟอร์ม PaymentHistory แล้วเซตการทำงานไปยังฟอร์ม BankerOption

public void initialPaymentHistoryForm()

กำหนดค่าเริ่มต้นให้แก่ฟอร์ม PaymentHistory

public void searchCustomerHistomer()

select ข้อมูลลูกค้าจากตาราง Customer ตามเลขรหัสลูกค้าที่ป้อนให้แก่ฟอร์ม และ select รายการบัญชีเงินกู้ของลูกค้าจากตาราง Loanaccount และ select ข้อมูลรายการชำระเงินจากตาราง Cash_Per_Month เพื่อแสดงข้อมูลต่างๆ ขึ้นบนฟอร์ม

public void showForm()

แสดงฟอร์ม PaymentHistory และเรียกเมทธอด initialPaymentHistoryForm() สำหรับกำหนดค่าเริ่มต้นให้แก่ฟอร์ม

public void showHistoryTable()

แสดงข้อมูลในตารางแสดงประวัติการชำระเงินของลูกค้าขึ้นบนฟอร์ม

Class SampleForm

public void calConstPaid()

คำนวณจำนวนเงินชำระคงที่ต่อเดือน จากจำนวนเงินกู้ , ระยะเวลาการผ่อนชำระ และอัตราดอกเบี้ยปัจจุบัน โดยจะไปเรียก Store Function GET_FACTOR เพื่ออ่านค่า factor สำหรับการคำนวณ

public void calMaxLoan()

เรียก Store Function GET_MAX_LOAN สำหรับอ่านค่าตัวคูณวงเงินกู้สูงสุด และทำการคำนวณจำนวนวงเงินกู้สูงสุดสำหรับลูกค้า ซึ่งเท่ากับรายได้ครอบครัวต่อเดือนคูณตัวคูณวงเงินกู้สูงสุด

public void clearSampleForm()

ลบค่าข้อมูลต่างๆ บนฟอร์ม Sample

public void exitForm()

ซ้อนการแสดงฟอร์ม Sample แล้วเซตการทำงานไปยังฟอร์ม BankerOption

public void gotoLoan()

ซ้อนการแสดงฟอร์ม Sample แล้วเรียกแสดงฟอร์ม NewLoan จากนั้นจึงส่งค่าตัวแปรไปยังฟอร์ม NewLoan ซึ่งได้แก่ รายได้ครอบครัวต่อเดือน , จำนวนเงินกู้ , ระยะเวลาการผ่อนชำระ และจำนวนเงินชำระคงที่ต่อเดือน

public void setType5()

เซตระยะเวลาการผ่อนชำระเป็นแบบ 5 ปี

public void setType10()

เซตระยะเวลาการผ่อนชำระเป็นแบบ 10 ปี

public void setType15()

เซตระยะเวลาการผ่อนชำระเป็นแบบ 15 ปี

public void setType20()

เซตระยะเวลาการผ่อนชำระเป็นแบบ 20 ปี

public void showForm()

แสดงฟอร์ม Sample และเรียกเมทธอด clearSampleForm() สำหรับกำหนดค่าเริ่มต้นให้แก่ฟอร์ม

public void showInterestRate()

เรียก Store Function GET_CURRENT_INT เพื่ออ่านค่าอัตราดอกเบี้ยปัจจุบัน แล้วแสดงขึ้นบนฟอร์ม Sample

บรรณานุกรม

[1] Owens Kevin T. (1996) : "Building Intelligent Databases with Oracle PL/SQL , Triggers , and Stored Procedures" , Prentice Hall , New Jersey.

[2] Portfolio Tom (1992) : "PL/SQL User's.Guide and Reference Version 2.0" , Oracle Corporation.

[3] Blaha Michael (1998) : "Object – Oriented Modeling and Design for Database Applications" , Premerlani William , Prentice – Hall , New Jersey.

[4] Tkach Daniel (1996) : "Visual Modeling Techinque" , Fang Walter , So Andrew , Addison Wesley California.

[5] Quatrani Terry (1998) : "Visual Modeling with Rational Rose and UML" , Addison Wesley Longman Inc.

[6] Douglass Bruce Powel (1998) : "Real – Time UML Developing Efficient Objects for Embedded Systems" , Addison – Wesley Longman Inc.

[7] Booch Grady (1994) : "Object – Oriented Analysis and Design with Applications" , Addison – Wesley , California.

[8] Gamma Erich (1994) : "Design Patterns : Element of Reuseable Object – Oriented Software" , Addison – Wesley.

[9] ดร.วีระศักดิ์ ซึงถาวร (1998) : "Fundamental of Java Programming volume I" , SUM Publishing Department , SUM System Company Limited. , กรุงเทพฯ

[10] Cooper James W. : "Using JDBC to Create Database Objects : Principles of Object – Oriented Programming in Java 1.1" , IBM T.J. Watson Research Center , www.ibm.com/java/

[11] Akerley John (1999) : "Programming with VisualAge for Java 2" , Li Nina , Parlavecchia Antonello , Prentice Hall.