การพัฒนาโปรแกรมภาษา JAVA ด้วยกลไก RPC

PROGRAM DEVELOPMENT WITH RPC BY JAVA

โดย

นาย รีโอ จับบาง

นาย รณชาติ ผ่องบุรุษ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปีการศึกษา 2542

การพัฒนาโปรแกรมภาษา JAVA ด้วยกลไก RPC

โดย

นาย รีโอ จับบาง

นาย รณชาติ ผ่องบุรุษ

อาจารย์ที่ปรึกษา

ผศ. บรรจง ปิยธำรง

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาโปรแกรมภาษา JAVA ด้วยกลไก RPC

ผู้จัดทำ

1. นาย รีโอ จับบาง 39014433
2. นาย รณชาติ ผ่องบุรุษ 39014421

_____ อาจารย์ที่ปรึกษา

(ผศ. บรรจง ปิยธำรง)

การพัฒนาโปรแกรมภาษาจาวาด้วยกลไกอาร์พีซี

รีโอ จับบาง
รณชาติ ผ่องบุรุษ
ผศ. บรรจง ปียธำรง
ปีการศึกษา 2542

บทคัดย่อ

ปัจจุบันนิยมใช้คอมพิวเตอร์หลายๆ เครื่องเข้ามาเชื่อมต่อกัน และเนื่องจากมีภาษาคอมพิวเตอร์ต่างๆ อยู่มากมายเช่นกัน ในการจัดการด้าน Object Oriented ภาษา Java นับว่าเป็นภาษาที่มีความสำคัญมาก เพื่อเป็นการแบ่งปันทรัพยากรที่มีอยู่ให้สามารถใช้งานร่วมกัน โดยลักษณะการแบ่งงานกันทำนี้จะเรียกการทำงานแบบนี้ว่า Distributed Programming และเป็นการประมวลผลร่วมกัน ซึ่งภาษา Java จะมี RMI ที่ทำงานโดย Call Procedure สำหรับความสามารถที่ Remote Method Invocation ทำให้ความสามารถของ RMI ที่เพิ่มขึ้นคือ สามารถส่ง Argument ได้ในหลายประเภทรวมทั้งที่เป็น Object Type ได้ด้วย และการทำงานของ RMI ยังสามารถรันได้ในระบบในเครื่องที่เป็น Local host และสามารถรันเครื่องอื่นๆ ที่เป็น Remote host ได้ เพื่อเกิดประโยชน์ความคุ้มค่ามากขึ้นของระบบเครือข่ายคอมพิวเตอร์

PROGRAM DEVELOPMENT WITH RPC BY JAVA

Reo Jubbang

Ronnachart Pongburut

Asst.Prof. Bunjong Piyatamrong Advisor

1999

Abstract

Nowaday, Computer applications work many process together over the network be favorite. And many languages support for Object Oriented Programming(OOP). Java is important language. For sharing many resource. By this method is called Distributed Processing. Java will have feature RMI by calling procedure. But increasing ability for Remote Method Invocation, make the RMI can communicate with many types include object type argument. And the RMI can run in local host and the others that is remote hosts. For reaching to higher proficiency of network computer.

สารบัญ

หน้า

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ปัจจุบันนิยมใช้คอมพิวเตอร์หลายๆ เครื่องเข้ามาเชื่อมต่อกัน และเนื่องจากมีภาษาคอมพิวเตอร์ต่างๆ อยู่มากมายเช่นกัน ในการจัดการด้าน Object Oriented ภาษา Java นับว่าเป็นภาษาที่มีความสำคัญมาก เพื่อเป็นการแบ่งปันทรัพยากรที่มีอยู่ให้สามารถใช้งานร่วมกัน โดยลักษณะการแบ่งงานกันทำนี้จะเรียกการทำงานแบบนี้ว่า Distributed Programming และเป็นการประมวลผลร่วมกัน ซึ่งภาษา Java จะมี RMI ที่ทำงานโดย Call Procedure สำหรับความสามารถที่ Remote Object invocation ทำให้ความสามารถของ RMI ที่เพิ่มขึ้นคือ สามารถส่ง Argument ได้ในหลายประเภทรวมทั้งที่เป็น Object Type ได้ด้วย และการทำงานของ RMI ยังสามารถรันได้ในระบบในเครื่องที่เป็น Local host และสามารถรันเครื่องอื่นๆ ที่เป็น Remote host ได้ เพื่อเกิดประโยชน์ความคุ้มค่ามากขึ้นของระบบเครือข่ายคอมพิวเตอร์

1.2 วัตถุประสงค์ของงานวิจัย

- ศึกษาทฤษฎีการทำงานรูปแบบประมวลผลร่วมกันโดย Distributed Programming
- ศึกษาภาษาทางคอมพิวเตอร์ที่มีการทำงานแบบ Object Oriented โดยภาษา Java ได้
- ศึกษาทฤษฎีของภาษาจาวา ซึ่งเป็นภาษาที่สามารถทำงานบนระบบเครือข่าย และเพื่อประกอบการวิจัยในการประมวลผลการทำงานวิธี RMI
- เพื่อเปรียบเทียบการประมวลผลร่วมกันโดยวิธี RPC ซึ่งเป็นแบบเก่า กับวิธี RMI โดย Java ซึ่ง RMI จะมีความสามารถที่เพิ่มขึ้นอย่างไร

1.3 ขอบเขตของงานวิจัย

การเขียนโปรแกรม RMI ที่สามารถใช้ในเครื่องตัวเอง (Local Mode) หรือในเครื่องอื่นๆ (Remote) ได้ โดยในงานวิจัยนี้ใช้ภาษาจาวาเป็นหลักในการพัฒนาทั้งในส่วนที่เป็นเว็บเซิร์ฟเวอร์และส่วนที่เป็นไคล์เอนท์

1.4 วิธีการดำเนินงาน

การดำเนินงานของโครงงานมีขั้นตอนดังนี้

1. ศึกษาค้นคว้าความรู้เกี่ยวกับทฤษฎีและวิธีการในการทำงานของ RMI
2. ศึกษาค้นคว้าภาษาจาวาในส่วนที่ต้องนำมาประยุกต์ใช้งาน
3. ออกแบบและพัฒนาส่วนของโปรแกรม

4. ทดสอบโปรแกรม Demoโดยทำการ Run ในระบบในเครื่อง และการ Run ในระบบข้ามเครื่อง เพื่อนำผลการทดสอบมาวิเคราะห์ถึงการนำทฤษฎีและความรู้ที่ได้จากการศึกษามาปฏิบัติจริง

1.5 ประโยชน์ที่ได้รับ

1. ได้รับความรู้ในทฤษฎีและวิธีการในการทำงานแบบ RMI
2. เข้าใจการเชื่อมต่อและการทำงานของระบบเครือข่ายได้ดียิ่งขึ้น
3. สามารถพัฒนาโปรแกรมร่วมกันได้จากหลายโปรแกรมเมอร์หลายฝ่าย
4. ได้รับความรู้ในการเขียนภาษาจาวามากขึ้น

บทที่ 2
ทฤษฎีและหลักการ

2.1 ความรู้เบื้องต้นของ RPC

RPC หรือ Remote Procedure Call เป็นการยินยอมให้ไคลเอนท์ใช้โปรซีเยอร์ที่อยู่ในคอมพิวเตอร์อื่นหรือ เซอร์ฟเวอร์ในเครือข่ายเดียวกัน RPC มักใช้บนระบบปฏิบัติการแบบ distributed เป็นเครื่องมือช่วยในการเขียนโปรแกรมใช้โมเดลแบบ client/server ทำให้เขียนโปรแกรมได้ง่ายขึ้นและมีประสิทธิภาพมากกว่าแต่ก่อนที่ต้องรู้อินเตอร์เฟสระดับต่ำ การนำมาใช้งานต้องมี protocol compiler เข้ามาช่วยให้ไคลเอนท์มอง remote procedure call เหมือน local procedure call

2.1.1 โมเดลแบบ client/server

เป็นโมเดลสำหรับระบบที่ทำงานแบบ distributed แอพพลิเคชั่นในโมเดลนี้แบ่งเป็น2 ส่วนคือ ไคลเอนท์และเซอร์ฟเวอร์ ไคลเอนท์มักเป็นผู้ส่งสัญญาณ request ไปให้เซอร์ฟเวอร์เสมอ และเซอร์ฟเวอร์เป็นผู้ส่งข้อมูลไคลเอนท์ตามที่ request ไว้ เซอร์ฟเวอร์แยกได้หลายชนิดเช่น เซอร์ฟเวอร์สำหรับค้นหาไฟล์ เซอร์ฟเวอร์สำหรับงานพิมพ์ เซอร์ฟเวอร์สำหรับการสื่อสาร เซอร์ฟเวอร์สำหรับแชร์ข้อมูลหรืออุปกรณ์ต่างๆ ตัวอย่างระบบปฏิบัติการโมเดลนี้เช่น windows NT,UNIX

ใน RPC ผู้ที่คอลล์โปรซีเยอร์ที่อยู่ไกลหรือโปรซีเยอร์บนคอมพิวเตอร์เครื่องอื่นเรียกว่า ไคลบเอนท์และผู้ที่ให้บริการ remote procedure call เรียกว่าเซอร์ฟเวอร์ ในเทอร์มของ remote ของ RPC ไม่จำเป็นที่ไคลเอนท์และเซอร์ฟเวอร์ต้องสื่อสารข้ามเครือข่าย หมายความว่าไคลเอนท์และเซอรฟเวอร์อยู่บนเครื่องเดียวกันได้

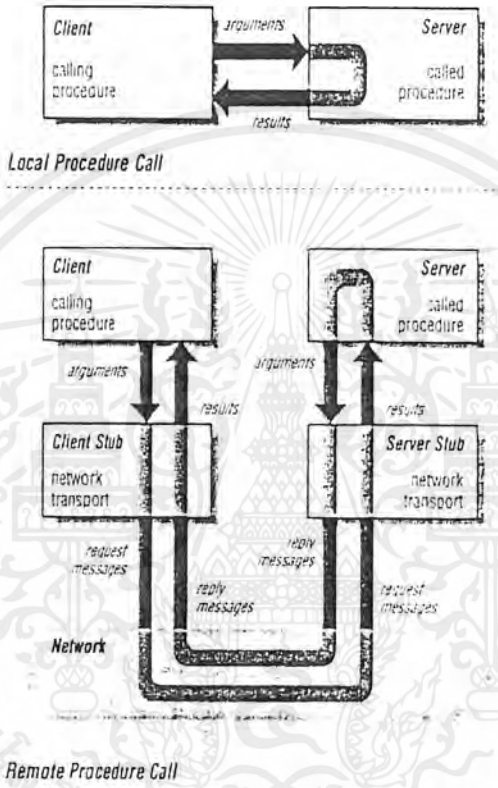
แอพพลิเคชั่น RPC ง่ายๆประกอบด้วยหนึ่งไคลเอนท์และหนึ่งเซอร์ฟเวอร์ ไคลเอนท์เป็นผู้คอลล์โปรซีเยอร์บนเซอร์ฟเวอร์และรอจนกระทั่งโปรซีเยอร์นั้นทำงานบนเซอร์ฟเวอร์เสร็จแล้วส่งผลลัพธ์กลับมาที่ไคลเอนท์จากนั้นการควบคุมของแอพพลิเคชั่นบนไคลเอนท์จะกลับคืนมา

2.1.2 RPC และ LPC

LPC หรือ Local Procedure Call แสดงไว้ในรูป 1 ด้านบน โปรเซสคอลล์(ไคลเอนท์)นำโปรซีเยอร์จากเซอร์ฟเวรอร์มาทำงานบนหน่วยความจำของมันเอง และภาพด้านล่างแสดงการทำงานของ RPC

RPC ใช้สัญญาณ request และ reply ทางด้านไคลเอนท์ส่งเมสเสส request ไปให้เซอร์ฟเวอร์และเซอร์ฟเวอร์ส่งเมสเสส reply กลับมาให้กับไคลเอนท์ ไคลเอนท์และเซอร์ฟเวอร์สื่อสารกันโดยใช้ 2 stub คือ client stub และ server stub Stub เป็นอินเตอร์เฟสที่ใช้ในการสื่อสารของโปรโตคอล RPC ซึ่ง จะระบุวิธีการ

สร้างและแลกเปลี่ยนเมสเสส stub ถูกสร้างโดย protocol compiler (ONC RPCGEN บน UNIX และ MIDL บน Windows NT) และทำหน้าที่เชื่อมต่อระหว่างโปรแกรมบนไคลเอนท์และเซอร์ฟเวอร์ stub บรรจุไปด้วยฟังก์ชั่นทำหน้าที่แบบ LPC ให้เป็นชุดฟังก์ชั่นของ RPC ดังนั้นจึงทำให้การคอลล์ RPC แบบ LPCได้ซึ่งจะทำให้สะดวกในการเขียนโปรแกรมมาก หลังจากคอลล์โปรซีเยอร์แบบ remote แล้วเมื่อไคลเอนท์พบโปรซีเยอร์ที่คอลล์อยู่ใน stub ของมัน มันใช้ RPC library ค้นหา remote process และสร้างเมสเสส request ขึ้นมา และเมื่อ server stub ได้รับ request จะเรียกโปรแกรมย่อยของบริการซึ่งเป็น LPC ขึ้นมาทำงาน



รูปที่ 2-1 เปรียบเทียบโลคอล และ รีโมทคอลล์

ไคลเอนท์และเซอร์ฟเวอร์จะสื่อสารซึ่งกันและกันโดยข้อมูลไม่ขึ้นอยู่กับรูปแบบของเครื่องคอมพิวเตอร์ RPC ใช้ฟอร์แมตข้อมูลที่ได้ตกลงกันไว้สื่อสารระหว่างไคลเอนท์และเซอร์ฟเวอร์ (ONC RPC บน UNIX ใช้ External Data Representation หรือ XDR ส่วนบน Windows NT ใช้ Network Data Representation หรือ NDR) client stub รวมพารามิเตอร์ต่างๆเป็นแพกเกจที่มีโครงสร้างที่ซับซ้อน (XDR หรือ NDR)แล้วส่งข้ามเครือข่ายไปสู่เซอร์ฟเวอร์และถูกแยกแพกเกจที่ server stubต่อจากนั้นถูกส่งไปที่ remote procedure ที่ไคลเอนท์เรียกใช้งาน ผลลัพธ์ถูกแปลงเป็นเมสเสสแล้วส่งกลับมาที่ไคลเอนท์เป็นค่าหนึ่ง

จะเห็นได้ว่างานจำนวนมากถูกซ่อนไว้จึงทำให้คอลล์ RPC เหมือนกับ LPC โมเดล RPC ช่วยให้เขียนโปรแกรมได้ง่ายขึ้น โปรแกรมเมอร์ไม่รู้เลยว่าโปรซีเยอร์ใดทำงานแบบ remote หรือ local

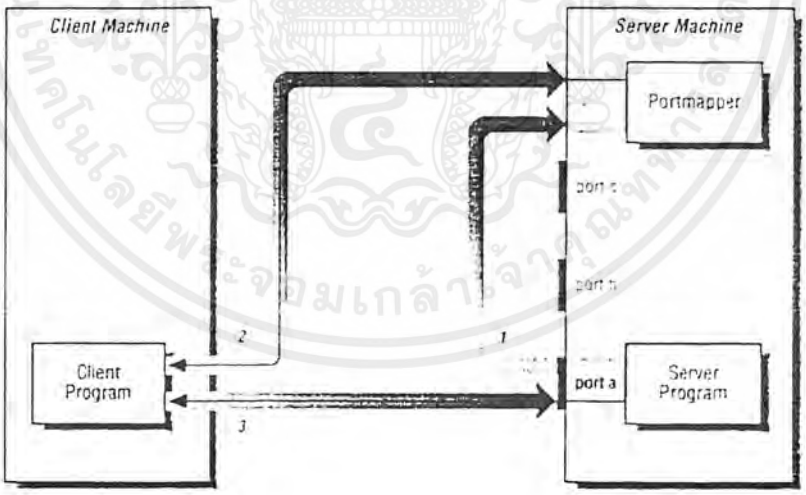
2.1.3 การทำงานของระบบ RPC

การบริการ(services)ทำให้รู้ว่าในเครือข่ายใครเป็นไคลเอนท์และใครเป็นเซอร์ฟเวอร์ service ต่างๆ ใช้พื้นที่หน่วยความจำของไคลเอนท์สร้างช่องทางการสื่อสารกับเซอร์ฟเวอร์ ทางด้านเซอร์ฟเวอร์สามารถที่จะรับหรือไม่รับ request ของไคลเอนท์และส่ง reply เท่าที่จำเป็น หลังจากการสื่อสารเสร็จสิ้นช่องทางการสื่อสารนี้จะถูกปิดลง

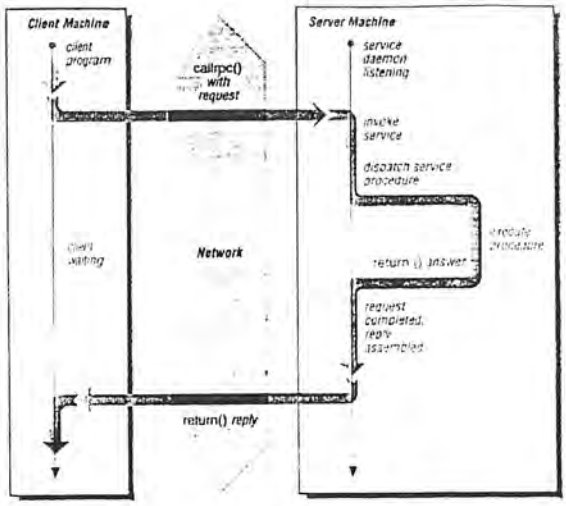
service เป็นโปรแกรม daemon ซึ่งจะคอยดักฟัง request ขอใช้บริการจากแอพพลิเคชั่นต่างๆหรือจากเครื่องอื่นๆ service ถูกกำหนดโดย port ซึ่ง port เป็นช่องทางการสื่อสารระหว่างเครือข่าย portmapper เป็นบริการชนิดหนึ่งสำหรับแมบ service เข้ากับ port ปกติทุกโฮส portmapper ถูกกำหนดให้มีหมายเลข port เป็น 111 ซึ่งยอมให้แอกเซสโดยตรงโดยไคลเอนท์และเซอร์ฟเวอร์ในการแมบ port ต่างๆผ่านโปรแกรม portmap

รูป 2 แสดงขั้นตอนต่างๆขณะที่ไคลเอนท์คอลล์โปรซีเยอร์จากเซอร์เวอร์

1. เมื่อ RPC server (หรือ dacmon) เริ่มทำงาน มันจะจองแอดเดรสส่วนหนึ่งหรับรอ request โดยกำหนดหมายเลข port (หรือแอดเดรส) โดย portmapper และกำหนดหมายเลขโปรแกรม RPC และหมายเลข version ซึ่งเซอร์ฟเวอร์จัดเตรียมสำหรับบริการ
2. ก่อนที่ไคลเอนท์จะสร้าง remote procedure call (ใช้หมายเลขโปรแกรมเซอร์ฟเวอร์) portmapper ของเซอร์ฟเวอร์จะทำการกำหนดหมายเลข port สำหรับรับเมสเสส request
3. ไคลเอนท์และเซอร์ฟเวอร์สามารถเปิดเส้นทางการสื่อสารเพื่อทำการรัน remote procedure สร้าง request และเซอร์เวอร์ส่ง reply



รูปที่ 2-2 RPC client/server setup

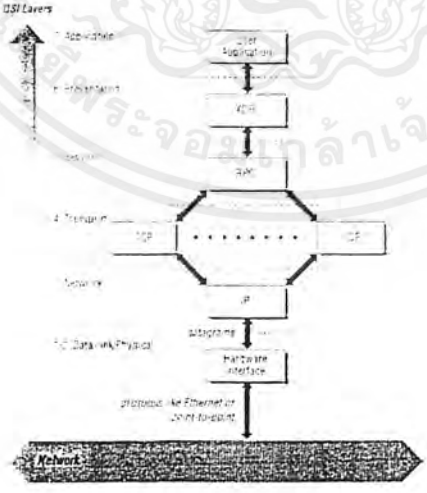


รูปที่ 2-3 การสื่อสารของ RPC

รูปที่ 3 แสดงลำดับเหตุการณ์ขณะทำ remote procedure call (ในขั้นตอนที่ 3) โปรเซสสามารถเริ่มทำงานได้ต่อเมื่อ service ได้ถูกกำหนดหมายเลข port โดย portmapper และไคลเอนท์ได้รับแอดเดรสสำหรับการสื่อสารได้ถูกต้อง เมื่อไคลเอนท์ส่งเมสเสส request สำหรับคอลล์โปรซีเยอร์ให้กับเซอร์ฟเวอร์และจะคอยจนกว่าเซอร์ฟเวอร์จะตอบกลับมา เซอร์ฟเวอร์ทำงานบริการที่ request เข้ามาและส่งเมสเสส reply พร้อมทั้งผลลัพธ์กลับไป

2.2 ระบบ RPC และ OSI Reference Model

Open Systems Interconnection(OSI) reference model ได้กำหนดให้การสื่อสารบนเครือข่ายคอมพิวเตอร์มี 7 เลเยอร์ในรูป 4 แสดงระบบของ remote procedure call แอพพลิเคชั่นอยู่เลเยอร์บนสุด และสื่อสารโดยใช้รูปแบบข้อมูลที่ใช้กับ RPC library ซึ่ง RPC library สนับสนุน TCP และ UDP

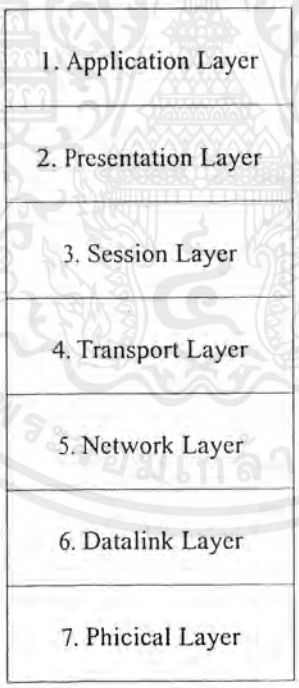


รูปที่ 2-4 ระบบ RPC กับ OSI reference model

2.3 โมดูล OSI

OSI เป็นคำย่อที่มาจากคำว่า Open Systems Interconnection โดยที่เป็นมาตรฐานที่ถูกเสนอขึ้นโดย International Standards Organization ซึ่งเป็นองค์กรที่จัดตั้งขึ้นมาเพื่อดูแลและส่งเสริม ตลอดจนกำหนดมาตรฐานของการติดต่อสื่อสารของระบบเครือข่ายคอมพิวเตอร์ โดยโมเดล OSI นี้มีลักษณะเป็นสถาปัตยกรรมแบบระบบเปิดเพราะมุ่งที่จะให้ระบบคอมพิวเตอร์ในหลาย ๆ รูปแบบที่แตกต่างกันสามารถเชื่อมต่อกันได้ OSI โมเดลได้แบ่งโปรโตคอลในการสื่อสารออกเป็น 7 เลเยอร์ ซึ่งโปรโตคอลคือชุดของกฎหรือข้อตกลงในการติดต่อ ข้อสังเกตโมเดล OSI เป็นเพียงข้อเสนอแนะมิใช่ข้อกำหนดและควรรู้ว่ายังไม่มีระบบการเชื่อมต่อใดที่สร้างเหมือนกันโมเดล OSI จริงๆ

ในหนึ่งชั้นของเลเยอร์ไม่ได้กำหนดว่าจะต้องมีเพียงหนึ่งโปรโตคอลเท่านั้นที่อยู่ในระดับเลเยอร์เดียวกันและในทางตรงข้ามชุดของโปรโตคอลใด ๆ อาจจะมีมากกว่าหนึ่งเลเยอร์ประกอบกันเป็นข้อกำหนดของระบบเครือข่ายเรียกว่าชุดโปรโตคอล เช่น ชุดโปรโตคอล TCP/IP (Transmission Control Protocol/Intesnet Protocol) เป็นต้น ประโยชน์ในการแบ่งเป็นเลเยอร์ คือกำหนดการติดต่อระหว่างเลเยอร์ทำได้โดยไม่ต้องคำนึงถึงการเปลี่ยนเลเยอร์ใด ๆ ที่ติดกัน



รูปที่ 2-5 OSI โมเดล ทั้ง 7 เลเยอร์

2.4 ภาษาจาวา

ทุกวันนี้ internet กำลังเติบโตเป็นอย่างมากซึ่งนำไปใช้งานในด้านต่าง ๆ ทั้งธุรกิจการค้า การเรียนการสอน การติดต่อสื่อสาร และอื่น ๆ อีกมาก ซึ่งในการพัฒนาแอพพลิเคชั่นบนอินเตอร์เน็ตนั้นมีโปรแกรมต่าง ๆ มากมายที่สนับสนุน แต่โปรแกรมที่มีความสามารถมากที่สุดตัวหนึ่งบนอินเตอร์เน็ตคือโปรแกรมที่สร้างขึ้นมาจากภาษาจาวา ซึ่งถูกพัฒนาโดยบริษัทซันไมโครซิสเต็มส์ซึ่งจุดประสงค์เริ่มแรกคือพัฒนาขึ้นมาเพื่อทำการควบคุมอุปกรณ์อิเล็คทรอนิคส์ได้โดยไม่ขึ้นอยู่กับแพลทฟอร์ม (platform) ที่ใช้คือการพัฒนาซอฟต์แวร์ให้สามารถทำงานได้โดยไม่ต้องสนใจว่าฮาร์ดแวร์ที่ใช้มีลักษณะอย่างไร นอกจากนี้จาวายังสนับสนุนความสามารถในการเชื่อมต่อเครือข่าย การติดต่อสื่อสารผ่านทางระบบเครือข่ายคอมพิวเตอร์ด้วย

คุณสมบัติทั่วไปของภาษาจาวามีดังนี้

1. เขียนง่าย เนื่องจากจาวามีลักษณะเป็นภาษาคอมพิวเตอร์ระดับสูงแต่ได้ตัดบางคุณลักษณะที่ไม่จำเป็นในภาษาระดับสูงออกไป เช่นไม่สนับสนุนพอยเตอร์ (point math), การโหลดเกินของตัวดำเนินการ (operator overloading), การสืบทอดจากคลาสหลายคลาส (multiple inheritance)
2. มีชนิดเป็นสแตติก คือ ออบเจ็กต์ทั้งหมดที่ถูกใช้ในโปรแกรมต้องถูกกำหนดก่อนที่มันจะถูกใช้งาน ทำให้ตัวคอมไพเลอร์ของจาวาสามารถรายงานชนิดที่ขัดแย้งกันได้
3. เมื่อคอมไพล์โปรแกรมแล้วผลลัพธ์จะเป็นไฟล์ข้อมูลแบบไบท์โค้ด (ซึ่งเหมือนกับ machine code) ซึ่งมันสามารถถูกเอ็กซีคิวต์ภายใต้ระบบปฏิบัติการด้วย Java Interpreter ซึ่งตัว Interpreter จะอ่านไฟล์ที่เป็นไบท์โค้ด แล้วจะแปลงคำสั่งที่เป็นไบท์โค้ดไปเป็นคำสั่งที่เป็นภาษาเครื่อง (machine-language commands) ซึ่งจะถูกเอ็กซีคิวต์ได้โดยตรงโดยเครื่องที่กำลังรันโปรแกรมจาวาอยู่ เพราะฉะนั้นเราจะกล่าวได้ว่าจาวาเป็นทั้งตัว compile และ interpret
4. สนับสนุนการทำงานได้หลาย ๆ งานในเวลาเดียวกัน
5. โปรแกรมจาวาจะจัดการกับขยะที่เกิดจากโปรแกรมด้วยตัวมันเอง ซึ่งหมายความว่าโปรแกรมจะไม่ต้องการที่จะลบออบเจ็กต์ซึ่งอยู่ในหน่วยความจำ
6. ความมั่นคง (Robust) เพราะว่า Java Interpreter จะตรวจสอบระบบทั้งหมดที่จะถูกเข้าถึงโดยโปรแกรม ดังนั้นจึงทำให้มั่นใจได้ว่าโปรแกรมจาวาจะไม่ทำให้ระบบพัง และถ้าเกิดความผิดพลาดขึ้นจะมี throw exception ออกมาซึ่งตัว exception นี้จะถูกจับและจัดการโดยโปรแกรมเพื่อทำให้ไม่เกิดความเสี่ยงกับระบบ
7. ความปลอดภัย ระบบจาวาไม่ใช่แค่รับรองการทำงานในการเข้าถึงหน่วยความจำเท่านั้น แต่ยังทำให้แน่ใจได้ว่าจะไม่มีไวรัสเกิดขึ้นในการรันโปรแกรม เพราะว่าพอยเตอร์ไม่ถูกใช้ในภาษาจาวา
8. มีความยืดหยุ่น โปรแกรมจาวาสนับสนุน native method (ซึ่งเป็นฟังก์ชันที่ถูกเขียนขึ้นโดยภาษาอื่น ๆ เช่น C++) โดยจะทำให้โปรแกรมเมอร์นั้นเขียนฟังก์ชันซึ่งอาจจะถูกเอ็กซีคิวต์ได้เร็วกว่าฟังก์ชันเดียวกันในการเขียนด้วยภาษาจาวา โดย native method เชื่อมต่อแบบไดนามิกกับ

(dynamically link) กับโปรแกรมจาวาแต่เมื่อใดที่ภาษาจาวาแก้ไขในเรื่องความเร็วได้ก็ไม่มีความจำเป็นต้องใช้ native method อีกต่อไป

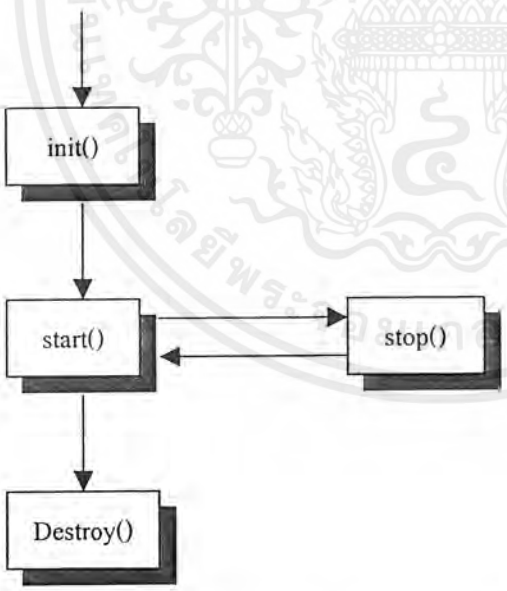
9. เข้าใจง่าย

2.4.1 จาวาแอพเพล็ต

ความสำเร็จของจาวาส่วนหนึ่งมาจากความสามารถของแอพเพล็ต การเขียนแอพเพล็ตก็มีการเขียนเช่นเดียวกับการเขียนจาวา ซึ่งอาศัยหลักการพื้นฐานจากจาวา แอพเพล็ตสามารถรันได้จากเว็บเบราเซอร์ที่สนับสนุนการทำงานกับจาวา (Java - Enabled) ทั้งสามารถโต้ตอบและแสดงอนิเมชันบน HTML ซึ่งกล่าวได้ว่าแอพเพล็ตเป็นตัวทำให้จาวามีเสน่ห์ ดึงดูดและเป็นที่นิยมใช้อย่างแพร่หลาย

2.4.1.1 คลาส Applet

จาวาแอพเพล็ตต้องทำการ extends java.applet.Applet โดยที่แอพเพล็ตจะจัดหาโครงสร้างการทำงานที่สำคัญให้กับเราในการรันบนเบราเซอร์จาวาแอพพลิเคชันจะมีเมธอด main() ที่จะทำการรันเมื่อแอพพลิเคชันเริ่มทำงาน ต่างจากแอพเพล็ตซึ่งไม่จำเป็นต้องมีเมธอด main() โดยที่เมธอดในแอพเพล็ตจะขึ้นอยู่กับการทำงานของเบราเซอร์ ซึ่งมีโครงสร้างการทำงานดังรูป



รูปที่ 2-6 การทำงานของเว็บเบราเซอร์ที่ควบคุมการทำงานของเมธอดต่างๆ ในแอพเพล็ต

เบราเซอร์จะเป็นตัวควบคุมการเรียกเมธอด init(), start(), stop() และ destroy() โดยปกติแล้วเมธอดเหล่านี้จะไม่เกิดการทำงานใดๆ เว้นแต่ต้องการทำงานเฉพาะบางอย่าง ซึ่งทำได้โดยการเขียนโค้ดลงให้เหมาะสมกับเมธอดที่ต้องการจะทำงานด้วย

2.4.1.2 วงจรชีวิตของแอพเพล็ต (Applet Life Cycle)

วงจรชีวิตในที่นี้จะหมายถึง วงจรที่แอพเพล็ตต้องทำงานกับเมธอดต่างๆ ที่ตัวแอพเพล็ตอ้างอิงด้วย ซึ่งปกติจะมีการทำงานวนเวียนดังนี้

1. ตั้งค่าเริ่มต้นให้ตัวเอง (initialize)
2. เริ่มรันแอพเพล็ต (start)
3. หยุดรันแอพเพล็ต (stop)
4. ทำลายตัวเองเพื่อเตรียมจะหยุดการทำงานจนกว่าจะมีการเรียกใช้ใหม่ (destroy)

ซึ่งอธิบายได้ว่าแอพเพล็ตจะเริ่มต้นจากการตั้งค่าเริ่มต้นให้ตัวเอง แล้วจบลงด้วยการทำลายตัวเอง แต่ในขณะที่ทำงานบางครั้งเราทิ้งหน้าจอไว้ชักพัก แล้วกลับมาดูอีกครั้ง เราจะพบว่าแอพเพล็ตไม่ทำงานตัวเอง เพียงแต่จะหยุดการทำงานเท่านั้น และเมื่อเรากลับมาดูใหม่ ก็ไม่จำเป็นต้องทำการตั้งค่าเริ่มต้นใหม่อีก ซึ่งจะทำให้การทำงานเร็วขึ้น แต่อย่างไรก็ตาม ก็จะมีการใช้ทรัพยากรสิ้นเปลืองเช่นกัน

วงจรชีวิตที่กล่าวมาข้างต้นอาศัยเมธอดพื้นฐานดังนี้

1 init()

เมธอด init() จะถูกเรียกเมื่อแอพเพล็ตเริ่มทำงานครั้งแรกเท่านั้นส่วนมากแล้วเราจะใช้เมธอดนี้เพื่อจัดการเตรียมค่าองค์ประกอบเริ่มต้นต่างๆ ให้พร้อมต่อการใช้งาน เช่น การกำหนดค่าเริ่มต้นให้กับตัวแปร การโหลดไฟล์ภาพหรือไฟล์เสียง ซึ่งกิจกรรมเหล่านี้บางครั้งถ้าไปทำขณะที่กำลังทำงานอยู่อาจจะทำให้การทำงานช้าลง ดังนั้นกิจกรรมใดที่เราคิดว่าต้องทำเพียงครั้งเดียวหรือคสรจัดการก่อนเข้าสู่การทำงานจริงก็ควรจะจัดการในเมธอดนี้ ซึ่งมีรูปแบบ

```
public void init()
```

2 start()

เมธอด start() จะถูกเรียกหลังจากเมธอด init() และจะทำงานใหม่ทุกครั้งเมื่อแอพเพล็ตถูกเรียกกลับมาให้ทำงาน เช่น เมื่อเราสั่งให้ Internet Explorer ย้อนกลับไปโหลดโฮมเพจก่อนหน้านี้ (Back) แล้วเราสั่งให้กลับมาที่หน้าเดิม (Forward) ที่มีแอพเพล็ตของเราทำงานอยู่ แอพเพล็ตจะได้รับเหตุการณ์ start อีกครั้งหนึ่ง มีรูปแบบ

```
public void start()
```

3 stop()

เมธอด stop() จะทำงานตรงข้ามกับเมธอด start() กล่าวคือจะถูกเรียกเมื่อออกจากโฮมเพจหน้าที่มีแอพเพล็ตนั้นๆ ทำงานอยู่เพื่อไปทำงานที่โฮมเพจหน้าอื่น และเมื่อเรากลับมาทำงานที่แอพเพล็ตหน้าเดิม เมธอด start() ก็จะถูกเรียกใช้งานอีกครั้งหนึ่ง และสุดท้ายเมธอด stop() ก็จะถูกเรียกใช้งานก่อนเมธอด destroy() เสมอ ซึ่งมีรูปแบบ

public void stop()

4 destroy()

เมธอด destroy() จะถูกเรียกใช้งานหลังจากเมธอด stop() ทุกครั้งเพื่อทำการสะสางทรัพยากรต่างๆ ที่ได้จองไว้ใช้งานก่อนหน้านี้ เช่น ยกเลิกการติดต่อกับระบบเครือข่ายใดๆ ที่เรากำลังติดต่ออยู่ หากเรามีกิจกรรมใดๆ ที่จะต้องทำก่อนจบการทำงานของแอพเพล็ตก็ควรจะจัดการในเมธอดนี้ มีรูปแบบ

public void destroy()

2.4.2 เมธอดที่ใช้ในการวาดภาพและเมธอดที่ตอบสนองต่อเหตุการณ์

สำหรับเมธอดที่ใช้ในการแสดงผลหน้าจอประกอบด้วย

paint(Graphics g)

เมธอด paint(Graphics g) ใช้ในการวาดภาพภายในพื้นที่แสดงผลของแอพเพล็ต ซึ่งเมธอดนี้เป็นเมธอดเดียวที่เราสามารถใช้เมธอดที่เกี่ยวกับกราฟฟิกในการวาดภาพหรือวาดตัวอักษรบนพื้นที่แสดงผลของแอพเพล็ต มีรูปแบบ

public void paint(Graphics g)

update()

เมธอด update() ทำหน้าที่คล้ายเมธอด paint(Graphics g) คือใช้จัดการกับการแสดงผลในแอพเพล็ต แต่ใช้ในลักษณะปรับปรุงการแสดงผลของภาพ กรณีการทำ Double Buffered และขณะที่มีการวาดภาพใหม่เมื่อเรียกเมธอด update() จะไม่ทำการลบพื้นที่แสดงผลภาพใหม่ โดยเมื่อเมธอด update() ถูกเรียกใช้งาน ก็จะไปเรียกให้เมธอด paint(Graphics g) ให้ทำงานอีกครั้งต่อหนึ่ง ดังนั้นเมื่อเราต้องการที่จะปรับปรุงส่วนใดๆ บนพื้นที่การแสดงผล เราก็จะมาจัดการกับเมธอดนี้ มีรูปแบบ

public void update()

2.4.3 การใส่แอพเพล็ตลงในเอกสาร HTML

สามารถทำได้โดยใส่แทก <APPLET> ลงในโค้ดของ HTML ซึ่งมีรูปแบบของการเขียนโค้ดดังนี้

<APPLET CODE = Applet.class WIDTH = widthInt HEIGHT = heightInt> </APPLET>

ทุกรูปแบบข้างต้นแสดงถึงองค์ประกอบพื้นฐานต่อไปนี้

1. เริ่มต้นด้วยการประกาศเป็น APPLET-tag คือ <APPLET>
2. บอกถึงไฟล์ที่จะใช้ในการรัน ซึ่งต้องเป็นไฟล์ที่คอมไพล์มาแล้ว นั่นคือเป็นไฟล์ *.class ตามด้วยการบอกถึงขนาดความกว้างและความยาวของพื้นที่ที่ต้องการแสดงแอพเพล็ตนั้น โดยใส่กับ WIDTH และ HEIGHT ตามลำดับ
3. ปิดท้ายด้วย </APPLET>

นอกจากองค์ประกอบข้างต้นแล้ว ยังมีฟังก์ชันที่บอกถึงที่อยู่ของไฟล์ไบท์โค้ดนั้นด้วย ซึ่งมีรูปแบบ

CODEBASE = URL

ในส่วน URL ของ CODEBASE นี้ นอกจากจะใช้เป็น URL แล้วยังสามารถแทนที่ด้วยที่อยู่ไดเร็กทอรีได้ด้วย และกรณีที่มีการรับข้อมูลเข้าสู่แอพเพล็ตก็สามารถทำได้โดยผ่านแทก <PARAM> ตามรูปแบบ

```
<APPLET CODE = Applet.class WIDTH = 600 HEIGHT = 400>
<PARAM NAME = param1 VALUE = value1>
<PARAM NAME = param2 VALUE = value2>
</APPLET>
```

แต่สำหรับการเขียนโค้ด HTML นั้นไม่มีความจำเป็นต้องเป็นตัวใหญ่ทั้งหมด และกรณีที่เบราเซอร์นั้นไม่สามารถรันแอพเพล็ตได้ เราจะใส่แทก <blockquote> เพื่อให้การทำงานทำข้ามบล็อกนี้ไป

2.5 Java Application

จาวาแอพพลิเคชั่นสามารถรันได้โดยตัวมันเองซึ่งต่างจากจาวาแอพเพล็ตตรงที่แอพเพล็ตต้องการบราวเซอร์ที่สนับสนุน และจาวาแอพพลิเคชั่นจะรันโดยเรียกคำสั่ง java ตามด้วยชื่อโปรแกรมที่มีชนิดเป็นคลาส

จาวาแอพพลิเคชั่นจะเริ่มต้นรันที่เมธอด main() ซึ่งแสดงดังนี้

```
public static void main(String args[])
{
....
}
```

public หมายถึง เมธอดนี้เป็นประโยชน์ต่อคลาสและออบเจ็กต์ต่าง ๆ ดังนั้นเมธอด main() จึงสามารถประกาศเป็น public

static หมายถึง main() เป็นเมธอดของคลาส

void หมายถึง เมธอด main() ไม่ส่งค่าใด ๆ กลับ

main() ใช้พารามิเตอร์เดียวเป็นอะเรย์ของสตริง พารามิเตอร์นี้จะรับอากิวเมนต์ที่ส่งผ่านมาจากบรรทัดคำสั่งของ DOS (DOS command line)

ส่วนตัวลำดับของเมธอด main() สามารถบรรจุโค้ดใด ๆ ก็ได้ตามที่ต้องการเช่น

- การกำหนดค่าเริ่มแรกให้ตัวแปร
- การสร้าง instance ต่าง ๆ ของคลาสใด ๆที่ประกาศ

เมื่อจาวาประมวลผลเมธอด main() คลาสจะยังไม่ถูกสร้าง instance โดยอัตโนมัติ ดังนั้นเมื่อดำเนินการคลาสในลักษณะออบเจ็กต์แล้วจะต้องสร้าง instance ในเมธอดของ main() ด้วยตัวเราเอง

เพราะว่าแอพพลิเคชั่นของจาวาเป็นโปรแกรมเดี่ยว มันใช้ประโยชน์ในการผ่านอาร์กิวเมนต์หรือตัวเลือกให้โปรแกรม เพื่อพิจารณาวิธีการที่โปรแกรมจะรันหรือช่วยโปรแกรมดำเนินการอินพุทที่แตกต่างกันมากมาย อาร์กิวเมนต์ของ command-line ซึ่งเราสามารถใช้ตามวัตถุประสงค์ที่เราต้องการได้ เช่น ใช้อินพุทของ debug เพื่อระบุชื่อไฟล์ที่จะอ่านหรือบันทึก หรือสารสนเทศอื่น ๆ ที่เราต้องการให้โปรแกรมของจาวารับทราบ

2.5.1 การผ่านอาร์กิวเมนต์ต่าง ๆให้โปรแกรมของจาวา

การผ่านอาร์กิวเมนต์ให้โปรแกรมของจาวาเราจะต้องใส่ค่าของอาร์กิวเมนต์ที่ command-line ของ MS-DOS ในโปรแกรมของจาวาดังนี้

java myProgram arg1 arg2 arg3

บนบรรทัดคำสั่ง (command-line) นี้ได้ใช้อาร์กิวเมนต์ 3 ค่าคือ arg1, arg2 และ arg3 แต่ละอาร์กิวเมนต์จะแยกจากกันโดยช่องว่าง เช่น

java myProgram java is best

ซึ่งในตัวอย่างจะส่งไป 3 อาร์กิวเมนต์ แต่ถ้ากลุ่มของอาร์กิวเมนต์ล้อมรอบด้วยเครื่องมาย " " แล้วจะถือว่าเป็น 1 อาร์กิวเมนต์ เช่น

java myProgram "java is best"

2.5.2 การเชื่อมโยงอาร์กิวเมนต์ต่าง ๆ ในโปรแกรม

จาวาสามารถเชื่อมโยงอาร์กิวเมนต์โดยบรรจุไว้ในอะเรย์ของสตริง ซึ่งจะถูกส่งผ่านไปให้เมธอด main() ซึ่งก็คือ args ภายในเมธอด main() เราสามารถเชื่อมโยงอาร์กิวเมนต์ args[] เป็นตัวรับอาร์กิวเมนต์จาก command-line ได้

2.6 RMI (Remote Method Invocation)

โปรแกรมแบบ RMI แบ่งโปรแกรมออกได้เป็น 2 ส่วน คือส่วน Server และส่วน Client โดยทั่วไปแล้วแอปพลิเคชั่นที่เป็นส่วน Server จะสร้าง object ขึ้นมา เพื่อให้อ้างถึง Remote 0bject เหล่านั้นได้ และรอฝั่ง Client เรียกใช้ method บน remote object เหล่านั้น ปกติแอปพลิเคชั่นฝั่ง Client จะเป็นผู้เรียก object ที่ฝั่งรีโมท RMI ของจาวาจัดเตรียมวิธีการที่ server และ client จะสื่อสารและแลกเปลี่ยนข้อมูลกันทั้งไปและกลับ บางครั้งอาจมองได้ว่าเป็น Distributed object ซึ่งจำเป็นที่จะต้อง

- สามารถอ้างตำแหน่งของ remote object ได้

เพื่ออ้างถึง remote object. แอปพลิเคชั่นจะประกาศตัว(register) remote object ด้วยการเรียกเครื่องช่วยของ RMI ด้วยวิธีการง่าย ๆ คือ เรียก rmiregistry จากนั้นแอปพลิเคชั่นสามารถส่งผ่านและคืนค่าที่อ้างถึงเสมือนเป็นส่วนหนึ่งของกระบวนการธรรมดา

- การสื่อสารกับ remote object

รายละเอียดสำหรับการสื่อสารระหว่าง object แบบรีโมทจะถูกจัดการโดย RMI เพื่อว่าโปรแกรมเมอร์ จะมองว่าการสื่อสารแบบ remote จะเป็นเพียงการเรียกใช้ method ตามมาตรฐานของจาวาเท่านั้น

- โหลดไบท์โค้ดคลาสเป็นออบเจ็กต์ ซึ่งผ่านมาเป็น parameter หรือ เป็น return value

เนื่องจาก RMI ยอมให้ผู้เรียกให้ส่งผ่านจาวาออบเจ็กต์ไปยังรีโมทออบเจ็กต์ RMI ได้จัดกลวิธีที่จำเป็นสำหรับส่งผ่านโค้ดของออบเจ็กต์ให้เป็นเช่นเดียวกันกับการส่งผ่านข้อมูล(data)

แอปพลิเคชั่น RMI แบบกระจาย จะใช้เพื่ออ้างถึงรีโมออบเจ็กต์ ส่วนทางฝั่งเซริฟเวอร์ได้เรียกรีจิสตรีเพื่อที่จะผนวกชื่อเข้าไว้กับรีโมทออบเจ็กต์ ฝั่งไคลเอนต์ ทำการมองรีโมทออบเจ็กต์ ด้วยชื่อเดียวกันกับที่ทางฝั่ง server ได้ประกาศไว้ (ทำการ รีจิสตรี) จึงเรียกใช้เมธอดนั้น จากภาพเราจะเห็นด้วยว่าระบบ RMI ใช้สำหรับเว็บเซริฟเวอร์ เพื่อจะโหลดออบเจ็กต์ ที่เป็นไบต์โค้ดของจาวา ระหว่างฝั่งเซริฟเวอร์และฝั่งไคลเอนต์เมื่อต้องการได้ RMI สามารถโหลดไบต์โค้ดได้โดยใช้ URL protocol (เช่น HTTP, FTP,..) ซึ่งรองรับการใช้งานบนระบบจาวา

2.6.1 Definition of Terms

ในแบบจำลองเชิงวัตถุของจาวานั้นรีโมทออบเจ็กต์ถือว่าเป็นออบเจ็กต์หนึ่ง ซึ่งสามารถเรียกเมธอดจาก java virtual machine ตัวอื่น ๆ ที่อยู่ต่างโฮสได้อย่างมีศักยภาพ ซึ่งชนิดของออบเจ็กต์นี้สามารถอ้างถึงด้วยรีโมทอินเตอร์เฟสหนึ่งหรือมากกว่านั้น ซึ่งอินเตอร์เฟสจะเป็นการประกาศเมธอดของรีโมทออบเจ็กต์เหล่านั้น

Remote Method Invocation (RMI) เป็นการเรียกเมธอดของอินเตอร์เฟสแบบรีโมทบน รีโมทออบเจ็กต์ สิ่งที่สำคัญคือ การเรียกเมธอดบนรีโมทออบเจ็กต์ มีไวยากรณ์เดียวกันกับการเรียกบนโลคอลออบเจ็กต์.

2.6.1.1 The Distributed and Nondistributed Models Contrasted

แบบจำลองกระจายของจาวานั้น เหมือนกันกับแบบจำลองของจาวาดังนี้

- การอ้างถึงรีโมทออบเจ็กต์ สามารถส่งผ่านเหมือนดังอาร์กิวเมนต์หรือค่าที่รีเทิร์นในการเข้าถึงเมธอด (ทั้งโลคอลและรีโมท)
- รีโมทออบเจ็กต์สามารถจะถูกคาสในกลุ่มของรีโมทอินเตอร์เฟส ซึ่งรองรับกับการ implement โดยใช้การคาสที่เป็นของจาวา
- การทำ instanceof ของจาวาที่มีให้มานั้น สามารถถูกใช้เพื่อทดสอบรีโมทอินเตอร์เฟส ซึ่งรองรับกับรีโมทออบเจ็กต์ แบบจำลองเชิงวัตถุของจาวาแบบกระจาย ต่างกันกับแบบจำลองเชิงวัตถุของจาวา คือ
- ไคลเอนต์ของรีโมทออบเจ็กต์ จะสื่อสารกับรีโมทอินเตอร์เฟส จะไม่ได้สื่อสารกับส่วนที่ implement ของอินเตอร์เฟส
- อาร์กิวเมนต์ ที่เป็นการคอลล์แบบรีโมท จะถูกส่งไปด้วยการคัดลอกมากกว่าการอ้างอิงถึง ที่เป็นเพราะว่าการอ้างอิงออบเจ็กต์เป็นประโยชน์กับ Virtual Machine เพียงตัวเดียวเท่านั้น
- รีโมทออบเจ็กต์ตัวหนึ่งจะถูกอ้างถึงโดยการอ้าง ไม่ใช่การคัดลอกส่วนที่ implement จริง ๆ
- ความหมายของบางเมธอด ถูกกำหนดโดยคลาส java.lang.Object ซึ่งเป็นกรณีเฉพาะสำหรับ remote object
- เนื่องด้วยความล้มเหลวในการเข้าถึงรีโมทออบเจ็กต์ ปกติจะซับซ้อนมากกว่าความล้มเหลวของการเข้าถึงโลคอลออบเจ็กต์ ฝั่งไคลเอนต์ จะต้องจัดการทำ exception เพิ่มเข้ามาด้วย ซึ่งอาจจะเกิดขึ้นได้ ระหว่างการทำ Remote Method Invocation.

2.6.2 Overview of RMI Interfaces and Classes

ทั้งอินเตอร์เฟสและคลาสที่จำเป็นสำหรับการกำหนดลักษณะและพฤติกรรมของระบบ RMI ถูกกำหนดไว้แล้วในแพ็กเกจ java.rmi ซึ่งเป็นลักษณะแบบ hierarchy

2.6.2.1 The java.rmi.Remote Interface

ใน RMI รีโมทอินเตอร์เฟสเป็นอินเตอร์เฟสซึ่งประกาศกลุ่มของเมธอด ซึ่งอาจจะถูกเรียกจาก Java Virtual Machine ที่เป็นรีโมท โดยรีโมทอินเตอร์เฟส จะต้องมีดังนี้

- อย่างน้อยรีโมทอินเตอร์เฟสต้องทำการขยายอินเตอร์เฟส java.rmi.Remoteไม่ว่าทางตรงหรือทางอ้อมก็ตาม
- การประกาศเมธอดแต่ละครั้งในรีโมทอินเตอร์เฟส ต้องประกาศรีโมทเมธอด ดังนี้
- การประกาศรีโมทเมธอด ต้องรวม exception java.rmi.RemoteException (หรือ superclass เช่น java.io.IOException หรือ jav.lang.Exception) ยิ่งไปกว่านั้น คือเพื่อที่จะมี exception ที่เป็นเฉพาะของแอปพลิเคชั่นนั้น ๆ (สังเกตว่า แอปพลิเคชั่นที่พิเศษนี้จะไม่ extend java.rmi.RemoteException)
- ในการประกาศ Remote Method Invocation , รีโมทออบเจ็กต์จะประกาศเป็น ดังเช่น พารามิเตอร์หรือ ค่ารีเทิร์น (อาจจะเป็นการประกาศโดยตรงในส่วนของพารามิเตอร์ หรือ ออบเจ็กต์ ที่ไม่ใช่รีโมท ฝังอยู่ในกลุ่มของพารามิเตอร์ก็ได้) จะต้องถูกประกาศแบบรีโมทอินเตอร์เฟส ไม่ใช่ ส่วนที่ implement คลาสของอินเตอร์เฟสนั้น

interface java.rmi.Remote เป็นตัวสร้างอินเตอร์เฟสซึ่งไม่ได้กำหนดเมธอด

```
public interface Remote { }
```

รีโมทอินเตอร์เฟสนี้อย่างน้อยต้องทำการขยายอินเตอร์เฟส java.rmi.Remote (หรือ remote interface ใด ๆ ซึ่งขยาย java.rmi.Remote) แต่อย่างไรก็ตามรีโมทอินเตอร์เฟสอาจขยายอินเตอร์เฟส ที่ไม่ใช่รีโมทภายใต้ข้อแม้เหล่านี้ รีโมทอินเตอร์เฟสอาจจะทำการขยายอินเตอร์เฟส ที่ไม่เป็นรีโมทแต่โดยรวมแล้วครบตามที่ต้องประกาศของรีโมทเมธอด

2.6.2.2 The RemoteException Class

คลาส java.rmi.RemoteException เป็น ซูเปอร์คลาสของ exception ที่ใช้ โดยระหว่างที่มีการรัน Remote Method Invocation เพื่อให้แน่ใจว่าแอปพลิเคชั่นที่กำลังทำ RMI นั้นมีความคงทน การประกาศรีโมทเมธอดในรีโมทอินเตอร์เฟส ต้องระบุ java.rmi..RemoteException (หรือ ที่เป็นซูเปอร์คลาสใด ๆ เช่น java.io.IOException หรือ java.lang.Exception) ในส่วนของการทำ throw. Exception java.rmi.RemoteException ถูก throw เมื่อ Remote Method Invocation ล้มเหลวเนื่องด้วยเหตุผลใดก็ตาม บางเหตุผลสำหรับ remote method invocation ก็คือ

- ความล้มเหลวเนื่องมาจากการสื่อสาร (remote server ไม่สามารถทำการเชื่อมต่อ,หรือการเชื่อมต่อถูกปิดโดย server)
- ความล้มเหลวระหว่างการทำ marshal หรือ unmarshal ของ parameter และ return value
- ความผิดพลาดอันเนื่องมาจากโปรโตคอล

class RemoteException จะถูกเช็ค exception ไม่ใช่ RuntimeException

2.6.2.3 The RemoteObject Class and its Subclasses

ฟังก์ชั่น RMI server ถูกจัดไว้ใน java.rmi.server.RemoteObject และ subclass ของมัน java.rmi.server.RemoteServer และ java.rmi.server.UnicastRemoteObject และ java.rmi.activation.Activatable.

- คลาส java.rmi.server.RemoteObject ประกอบไปด้วย implement สำหรับ java.lang.Object method. HashCode, equals และ toString ซึ่งเหมาะสมสำหรับรีโมทออบเจ็กต์

- เมธอดต้องสร้างรีโมทออบเจ็กต์และต้องเอ็กซ์ปอร์ต (สำหรับ remote clients) ซึ่งจัดโดยคลาส UnicastRemoteObject และสับคลาส Activatble สำหรับตัวอย่างเซิร์ฟเวอร์ ใด ๆ เซิร์ฟเวอร์เป็น รีโมทออบเจ็กต์ธรรมดาหรือเป็น รีโมทออบเจ็กต์กระตุ้นได้ (ซึ่งสามารถ execute ได้เมื่อเรียกใช้)

- java.rmi.server.UnicastRemoteObject class กำหนด singleton (unicast) remote object ซึ่งอ้างถึง server process ที่ยังคงมีชีวิตอยู่และใช้งานได้

- class java.rmi.activation.Activatable เป็น abstract class ซึ่งได้กำหนด activatable remote object ซึ่งเริ่มทำงานเมื่อรีโมทเมธอดถูกเรียก และปิดตัวเองลงเมื่องานสำเร็จ

เมื่อเขียนแอปเพล็ตหรือแอปพลิเคชั่นที่ใช้รีโมทออบเจ็กต์ โปรแกรมเมอร์จะต้องตระหนักถึงอินเตอร์เฟสที่ไคลเอนต์ของระบบสามารถมองเห็นได้ ซึ่งมีอยู่ในแพ็กเกจ java.rmi

2.6.3 The Remote Interface

Java.rmi.Remote ถูกใช้เพื่อเก็บรวมรวมรีโมทอินเตอร์เฟสทั้งหมด ซึ่ง remote object ทั้งหมดจะต้อง implement อินเตอร์เฟสเหล่านี้ไม่ทางตรงก็ทางอ้อม

Implementation คลาสสามารถ Implement Remote Interface จำนวนหนึ่งซึ่งสามารถขยาย Implementation คลาสอื่น ๆ RMI ได้จัดเตรียมคลาส ซึ่งการ Implementation Remote Object สามารถทำการขยายและสร้างรีโมทออบเจ็กต์ขึ้นมาอีกคลาสเหล่านี้ได้แก่

java.rmi.server.UnicastRemoteObject และ java.rmi.activation.Activatable

2.6.3.1 The RemoteException Class

คลาส java.rmi.RemoteException เป็นซูเปอร์คลาสธรรมดาของ exception ที่เกี่ยวข้องกับการสื่อสาร ซึ่งอาจจะเกิดขึ้นได้ระหว่างการทำการคอลล์แบบ Remote Method แต่ละ Method ของ Remote Interface จะต้องรวม RemoteException (หรือ ซูเปอร์คลาสใดคลาสหนึ่ง เช่น java.io.IOException หรือ java.lang.Exception ในรูปแบบของ throws exception

RemoteException สามารถสร้างขึ้นด้วยข้อความที่มีเนื้อหา s และ nested exception ex (Throwable) โดยปกติแล้ว nested exception ex แสดงเป็นพารามิเตอร์ในแบบคอนสตรัคเตอร์

method getMessage จะคืนค่าของ exception เป็นการรวม message จาก nested exception

2.6.3.2 The Naming Class

คลาส java.rmi.Naming ได้จัดเตรียมเมธอด สำหรับเก็บรวบรวมและทำการอ้างอิงถึง รีโมทออบเจ็กต์ ในรีจิสตรีของรีโมทออบเจ็กต์ ส่วนเมธอดของ Naming คลาสประกอบไปด้วยอาร์กิวเมนต์ และ ชื่อในแบบ URL รูปแบบของ java.lang.String คือ //host:port/name

ซึ่ง host หมายถึง host (remote หรือ local) ที่รีจิสตรีติดตั้งอยู่ port เป็นหมายเลข port ซึ่งรับรีจิสตรี และ name เป็น string ธรรมดาที่ยังไม่ได้แปลงโดยรีจิสตรี. ทั้ง host และ port เป็นเพียงตัวเลือก ถ้า host ถูกละไป ค่าที่ตั้งไว้คือ localhost ถ้า port ละไปจะหมายถึง 1099 ซึ่งเป็นค่าที่ rmiregistry รู้จักเป็นอย่างดี

การ binding ชื่อของรีโมทออบเจ็กต์เป็นการทำงานร่วมกัน และเป็นการสร้างชื่อไว้สำหรับรีโมทออบเจ็กต์ ซึ่งถูกใช้ทีหลังเพื่อมองหารีโมทออบเจ็กต์ รีโมทออบเจ็กต์อันหนึ่งสามารถใช้รวมกับชื่อโดยการใช้ Naming คลาสเป็นเมธอดชื่อ bind หรือ rebind.

ครั้งแรกที่รีโมทออบเจ็กต์ได้รับการรีจิสเตอร์ด้วย RMI registry บน localhost ผู้เรียกหรือ รีโมท (หรือโลคอล) โฮส สามารถมองหารีโมทออบเจ็กต์ด้วยชื่อที่อ้างไว้นั้น และจากนั้นก็สามารถเรียกใช้รีโมทออบเจ็กต์บนนั้นได้ รีจิสตรีอาจถูกแชร์โดยเซิร์ฟเวอร์ทั้งหมดที่กำลังทำงานอยู่บนโฮส หรือ เซิร์ฟเวอร์หลาย ๆ ตัวโดยกระบวนการสร้างและใช้รีจิสตรีถ้าหากจำเป็น

(ดูที่ java.rmi.registry.LocateRegistry.createRegistry method)

เมธอด lookup จะคืนค่ารีโมทออบเจ็กต์ที่ร่วมกับชื่อนั้น NotBoundException จะเกิดขึ้น ถ้า ชื่อยังไม่ได้ถูกประกาศสำหรับออบเจ็กต์นั้น

เมธอด bind ได้ทำการรวมเอาชื่อไปไว้กับรีโมทออบเจ็กต์ จะทำการเรียก AlreadyBoundException ถ้าชื่อได้ถูก bound ไปยัง object แล้ว

เมธอด rebind จะทำการ bind ชื่อไปยังออบเจ็กต์เสมอ ถึงแม้ว่าชื่อนั้นจะถูก bind เอาไว้ก่อนแล้ว ส่วนค่าที่ bind ไว้แล้วจะหายไปหมด

เมธอด unbind จะทำการยกเลิกค่าระหว่างชื่อและรีโมทออบเจ็กต์ทั้งหมด มันจะเกิด NotBoundExceiption ถ้าไม่ได้มีการ bind เอาไว้ก่อน

เมธอด list คืนค่าเป็นอาเรย์ String ออบเจ็กต์ซึ่งประกอบด้วยองค์ย่อยของ URL ที่ได้ bind ไว้ในรีจิสตรีเป็นค่าของ host และ port ของ URL ซึ่งจำเป็นเพื่อที่จะทำการติดต่อกับรีจิสตรี สำหรับ list ส่วนที่เป็นระเบียนของ URL จะถูกละไป

2.6.3.3 Registry Interface

ระบบ RMI ใช้อินเตอร์เฟส java.rmi.registry.Registry และ คลาส java.rmi.registry.LocateRegistry เพื่อที่จะจัดบริการ bootstrap ที่เป็นที่รู้จักกันสำหรับการรับการจดทะเบียนออบเจ็กต์ด้วยชื่อที่ง่าย registry เป็นรีโมทออบเจ็กต์ซึ่งทำการแม็ปชื่อกับรีโมทออบเจ็กต์ กระบวนการที่เกิดทางฝั่งเซิร์ฟเวอร์ใด ๆ สามารถรองรับรีจิสตรีของมันหรือรีจิสตรีเดี่ยวใด ๆ สามารถถูกเรียกใช้ได้โดยโฮส เมธอดของ LocateRegistry ถูกใช้เพื่อที่จะรับเอากระบวนการรีจิสตรีบนโฮสที่ต้องการหรือเป็นชื่อโฮสและเบอร์พอร์ต ส่วนเมธอดของคลาส java.rmi.Naming ได้ทำการเรียกไปยังรีโมทเมธอดซึ่งทำการอิมพลีเมนต์ค่าของรีจิสตรีอินเตอร์เฟสโดยใช้เมธอด LocateRegistry.getRegistry ที่เหมาะสม

2.6.3.4 The Registry Interface

รีโมทอินเตอร์เฟส java.rmi.registry ได้จัดเตรียมเมธอดสำหรับทำ lookup , binding, rebinding, unbinding และ listing เนื้อความของ registry. คลาส java.rmi.Naming ใช้รีโมทอินเตอร์เฟส registry เพื่อที่จะจดชื่อแบบ URL

เมธอด lookup จะคืนค่ารีโมทออบเจ็กต์ที่รวบรวมไว้อยู่ในชื่อ name และ รีโมทออบเจ็กต์ได้ทำการอิมพลีเมนต์กลุ่มของรีโมทอินเตอร์เฟส ไคลเอนต์สามารถที่ทำการคาสรีโมทออบเจ็กต์เพื่อจะอ้างถึงรีโมทอินเตอร์เฟส (การคาสนี้สามารถจะล้มได้เช่นเดียวกับการล้มในภาษาจาวา)

เมธอด bind ทำการกำหนด name เข้ากับรีโมทออบเจ็กต์ , obj ถ้าชื่อได้ถูกกำหนดให้กับออบเจ็กต์เรียบร้อยแล้ว AlreadyBoundException จะถูก throw

เมธอด rebind ทำการกำหนด name เข้ากับรีโมทออบเจ็กต์ ,obj การกำหนดก่อนหน้านี้จะถูกยกเลิก

เมธอด unbind ทำการย้ายการกำหนดระหว่าง name และรีโมทออบเจ็กต์ ถ้าชื่อยังไม่ได้ถูกกำหนดให้กับออบเจ็กต์จะเกิด NotBoundException

เมธอด list จะคืนค่าเป็นอาร์เรย์ของสตริงซึ่งประกอบไปด้วยส่วนที่เป็น snapshot ของชื่อที่รวมไว้แล้วในรีจิสตรี ค่าที่คืนกลับมาจะประกอบด้วย snapshot ของเนื้อหาของรีจิสตรี

ไคลเอนต์สามารถเข้าถึงรีจิสตรีใด ๆ โดยอาจใช้ LocateRegistry และ Registry อินเตอร์เฟสหรือโดยการใช้เมธอดของคลาสที่ใช้ java.rmi.Naming เป็นพื้นฐาน รีจิสตรีรองรับการทำ bind, unbind, และ rebind จากไคลเอนต์ลบนโฮสเดียวกันกับเซิร์ฟเวอร์ การlookup สามารถทำได้ จากโฮสใด ๆ

2.6.4 Stubs and Skeletons

RMI ถูกใช้ในกระบวนการพื้นฐาน (ถูกใช้เวลาในระบบการทำงานของ RPC) สำหรับการติดต่อระหว่างของ Remote Object ทั้งสองคือ Stubs และ Skeletons สำหรับ Stub เป็น Remote Object ที่ทำหน้าที่เป็น Client ภายในที่แสดงให้เห็น หรือ proxy สำหรับ Remote Object ส่วนตัวผู้เรียกก่อให้เกิดวิธีการทำงานบน Stub ภายในที่รับผิดชอบต่อการรับภาระการเรียกไปยัง Remote Object ซึ่งใน RMI Stub ยังทำหน้าที่ในส่วนเพิ่มเติมของการสนับสนุน Remote Object ซึ่งในรูปแบบที่คล้ายกับการเชื่อมโยงของส่วนสนับสนุน Remote Object นั่นเอง

ส่วนทำงานของของ Stub จะเกิดขึ้นเมื่อ

- การติดต่อที่เริ่มขึ้นโดย Remote VM ซึ่งประกอบไปด้วย Remote Object
- การจัดรูปแบบการทำงานอย่างเหมาะสม (การเขียนบันทึกข้อมูลและการส่งข้อมูล) ในรูปแบบตัวแปรของ Remote VM
- การรอผลการทำงานของการติดต่อส่งข้อมูล
- ไม่สามารถจัดการทำงานได้เนื่องจาก การอ่านค่าที่ผิดพลาดไป และการส่งค่ากลับได้ไม่ถูกต้อง
- การส่งค่ากลับไปยังตัวผู้เรียก

สำหรับ Stub ยังทำการซ่อนตัวแปรที่ต่อเนื่องกันและการติดต่อสื่อสารในระดับเครือข่ายตามรูปแบบที่แสดงในการส่งข้อมูลเบื้องต้น

ในรูปแบบของ Remote VM ทั้งแต่ละ Remote Object จะมีการเรียก Skeleton โต้ตอบข้อมูลกัน (สำหรับ JDK 1.2 เท่านั้นที่มีตัวแปรทั่วไปเหมาะสม ซึ่งไม่มีการเรียกใช้ Skeleton) ใน Skeleton จะมีการการส่งข้อมูลจะเป็นไปอย่างรวดเร็วจากการเรียก ซึ่งใน Remote Object ตัวเพิ่มเติมอย่างแท้จริง
เมื่อ Skeleton ได้รับผลจากการทำงานดังนี้ คือ

- ไม่สามารถทำจัดข้อมูลการอ่านตัวแปรได้อย่างเหมาะสมในรูปแบบการเรียก
- รูปแบบการสนับสนุนของ Remote Object อย่างแท้จริง
- การจัดรูปแบบการทำงาน (ทั้งการเขียนและการส่งข้อมูล) รวมทั้งผลของข้อมูลที่แสดงค่าย้อนกลับจากผู้เรียก)

2.6.4.1 Thread Usage in Remote Method Invocations

ในรูปแบบการทำงานโดย RMI ที่ทำการส่งข้อมูลไปอย่างรวดเร็ว ซึ่งในเวลาการทำงานของ RMI จะไปยังการทำงานในลักษณะนอกเหนือจากที่กำหนดในการแยกแยะล่วงหน้าโดยจะมีการทำงานหรือไม่ สำหรับเวลาในการ run RMI จะไม่สามารถกำหนดอย่างแน่นอนในการตรวจสอบการจัดการ Remote Object อย่างล่วงหน้าได้ และตั้งแต่การใช้งาน Remote ทำการ โดยการใช้ remote ในรูปแบบเดียวกันที่จะได้ Object ที่ทำงานอย่างรวดเร็วและต่อเนื่องต่อไปได้ตามความต้องการและความปลอดภัยของหน่วยข้อมูล

2.6.4.2 Garbage Collection of Remote Objects

ในระบบ Distributed เริ่มต้นจากระบบภายใน ที่มีความต้องการจากการที่ข้อมูลสูญหายทั้งหมดโดยกะทันหัน ทั้งนี้เนื่องจากไม่มีการอ้างอิงถึง Client ตำแหน่งที่อยู่ออกไปอื่นๆ ได้ ดังนั้น นักเขียนโปรแกรมจึงมีความจำเป็นที่จะต้องเก็บรักษาส่วนที่เป็น Remote Object Client เพื่อว่ามันจะสามารถถูกทำลายได้อย่างถูกขั้นตอน RMI ยังถูกใช้ในการทำงานในลักษณะที่เป็นการอ้างอิงถึงตำแหน่งที่เป็นการคำนวณทาง Algorithm ซึ่งคล้ายกับรูปแบบเครือข่าย

เมื่อ Remote Object ไม่สามารถถูกอ้างโดย Client ซึ่งการ run RMI ได้ใช้การอ้างอิง ซึ่งการอ้างอิงจากการยอมรับของ JAVA virtual machine ซึ่งการพิจารณาจาก Object

2.6.4.3 Dynamic Class Loading

RMI จะสามารถยอมรับการทำงานของตัวแปร ที่จะส่งค่ากลับจากการเรียกผ่านไปยัง RMI ที่เป็น Object ที่สามารถมีความต่อเนื่องจากการจัดส่งข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่ง ซึ่งแต่ละส่วนจะประกอบไปด้วยการทำงานผ่านตำแหน่งหน่วยข้อมูลเพื่อว่าที่จะกำหนดลักษณะประเภทแฟ้มข้อมูลที่จะถูกโหลดเรียกมาใช้งานจากตัวรับข้อมูล

เมื่อตัวแปรเหล่านี้ถูกส่งค่ากลับจากการทำงานของการควบคุมและการจัดระเบียบการทำงานให้กลายเป็นรูปแบบการรับข้อมูล VM ซึ่งการกำหนดความต้องการทั้งหมดของประเภทวัตถุในงาน ซึ่งจากการจัดกระบวนการทำงานในระดับแรกที่เป็นความพยายามที่จะตัดสินใจโดยลำดับชื่อในคลาสภายในที่จะโหลด context (context class ตัวโหลดจากหน้าปัจจุบัน) RMI

remote ทั้งที่เป็นการเป็นโดยเราสามารถจำจาก Stub โดย ส่วนที่เป็นส่วนเสริมมาทำให้เรายังมีวิธีที่ทำการปฏิบัติโดย ทำการปฏิบัติเป็นโดยทั้งยังทำหน้าที่เป็นการทำให้เรา

2.6.5 The LocateRegistry Class

คลาส java.rmi.registry.LocateRegistry ถูกนำมาใช้เพื่อทำการอ้างอิงการทำ bootstrap สำหรับการรีจิสตรีรีโมทออบเจ็กต์บนโฮสที่ต้องการ(ไม่ได้รวมถึงโลคอลโฮส) หรือ เพื่อ สร้างรีจิสตรีของรีโมทออบเจ็กต์ซึ่งยอมรับการคอลล์บนพอร์ตที่ต้องการ

รีจิสตรีได้อิมพลีเมนต์การตั้งชื่อเป็นไวยากรณ์แบบธรรมดา ซึ่งรวมชื่อกับรีโมทออบเจ็กต์(เป็นสตริง) เพื่ออ้างไปยังรีโมทออบเจ็กต์ ทั้งชื่อและรีโมทออบเจ็กต์ที่ทำการไบด์ดิ้งแล้วจะไม่ได้จำเอาไว้เมื่อฝั่งเซิร์ฟเวอร์ได้ทำการรีสตาร์ทใหม่

จะสังเกตว่า getRegistry ที่เรียกจะไม่มีการเชื่อมต่อโดยตรงระหว่างชื่อกับรีโมทออบเจ็กต์ โดยปกติแล้วมันจะสร้างเพื่ออ้างแบบรีโมทไปที่รีโมทรีจิสตรี ผู้ใช้รู้สึกราวกับว่ากำลังทำงานอยู่บนรีโมทโฮสเลยทีเดียว

แต่อย่างไรก็ตาม สิ่งที่ตามมาในการเข้าถึงรีโมทนั้น จะคืนค่าเป็นผลลัพธ์ไปให้ซึ่งเมธอดนี้อาจจะล้มเหลวก็ได้ หนึ่งในสิ่งของเมธอด getRegistry จะคืนการอ้างถึงรีจิสตรีบนโฮสปัจจุบัน ที่พอร์ตที่ระบุไว้ รวมทั้งโฮสที่ระบุไว้ด้วย อะไรจะถูกคืนกลับมา ซึ่งก็คือสตับสำหรับรีจิสตรีด้วยค่าที่เป็นข้อมูลบทางโฮสและพอร์ต

เมธอดของ getRegistry อันที่ห้า(ซึ่งรวมเอา RmiClientSocketFactory รวมเข้าเป็นอาร์กิวเมนต์ไว้ด้วย) จะทำการคืนรีโมทสตับที่สร้างขึ้นทางฝั่งโลคอลไปยังรีโมทออบเจ็กต์ บนโฮสและพอร์ตที่กำหนด การสื่อสารกันด้วยรีโมทรีจิสตรี ซึ่งสตับได้ถูกสร้างขึ้นด้วยเมธอดนี้จะสร้าง RMIClientSocketFactory,csf, เพื่อสร้างซ๊อกเก็ตไปยังรีจิสตรีบนรีโมทโฮสและพอร์ต

เมธอด createRegistry สร้างและส่งรีจิสตรีบนโลคอลโฮสด้วยพอร์ตที่กำหนด เมธอดที่สอง CreateRegistry ก็คือ จะอนุญาตให้มีการสื่อสารที่ยืดหยุ่นกับรีจิสตรีมากกว่า การเรียกนี้จะสร้างและทำการส่ง Registry บนโลคอลโฮสซึ่งใช้ socket factory ที่ได้กำหนดไว้ก่อนแล้วสำหรับการสื่อสารกันด้วยรีจิสตรีนั้น รีจิสตรีนั้นจะถูกสร้างขึ้นมาเพื่อรับฟังสำหรับการเข้ามาของสัญญาณเรียกบนพอร์ตที่ให้โดยการใช้ ServerSocket สร้างจาก RMIServerSocketFActory ไคลเอนต์ซึ่งรับการอ้างถึงรีจิสตรีนี้จะใช้ Socket สร้างจาก RMIClientSocketFactory

2.6.6 The RemoteStub Class

คลาส java.rmi.server.RemoteStub เป็นโดยปกติแล้วเป็นซูเปอร์คลาสสำหรับรีโมทออบเจ็กต์ สตับออบเจ็กต์รองรับอย่างตรง ๆ กับกลุ่มของรีโมทอินเตอร์เฟสเดียวกับที่กำหนดไว้โดยการอิมพลีเมนต์รีโมทออบเจ็กต์ตรง ๆ

```
package java.rmi.server;
public abstract class RemoteStub extends java.rmi.RemoteObject {
```

```
    protected RemoteStub();
    protected RemoteStub(RemoteRef ref);
    protected static void setRef(RemoteStub stub, RemoteRef ref);
}
```

คอนสตรัคเตอร์แรกของ RemoteStub ได้สร้างสตรับด้วยค่าที่อ้างเป็นศูนย์ ส่วนคอนสตรัคเตอร์ที่สองสร้างสตับด้วยการอ้างถึงรีโมทที่ให้,ref
เมธอด setRef ได้ถูกรวมไว้ด้วยใน JDK1.2

2.6.6.1 Type Equivalency of Remote Objects with a Stub class

ไคลเอนต์ได้ทำการติดต่อกับสตับออบเจ็กต์ซึ่งมีรีโมทอินเตอร์เฟสเป็นกลุ่มเดียวกัน กำหนดโดยคลาสของรีโมทอินเตอร์เฟส ส่วนสตับคลาสไม่ได้ถูกรวมส่วนที่ไม่ใช่รีโมทของคลาสลำดับชั้นซึ่งประกอบไปด้วยออบเจ็กชนิดกราฟ ที่เป็นเช่นนี้เพราะว่าสตับคลาสนั้นถูกสร้างขึ้นมาจากการอิมพลิเมนต์สิ่งที่ละเอียดมากซึ่งทำการอิมพลีเมนต์หนึ่งหรือมากกว่าหนึ่งอินเตอร์เฟส ตัวอย่างเช่น ถ้า C ทำการขยาย B และ B ทำการขยาย A แต่ B เท่านั้นที่ได้ทำการอิมพลีเมนต์รีโมทอินเตอร์เฟส ดังนั้นสตับจะถูกสร้างขึ้นจาก B ไม่ใช่ C

และเพราะว่า สตับได้ทำการอิมพลีเมนต์กลุ่มของกลุ่มของรีโมทอินเตอร์เฟสที่เฟมือนกัน ของ รีโมทอินเตอร์เฟสดั่งเช่น เป็นคลาสของรีโมทออบเจ็กต์ จากมุมมองของจาวานี้จะพบว่าสตับนั้นมีไทป์เป็นรีโมทของ กราฟทางฝั่งของเซิร์ฟเวอร์ออบเจ็กต์ แต่อย่างไรก็ตาม ไคลเอนต์สามารถใช้การทำงานของจาวาที่ให้มาเพื่อที่จะตรวจสอบชนิดของออบเจ็กต์และเพื่อที่จะทำการคาสจากอินเตอร์เฟสหนึ่งไปยังอีกอันหนึ่ง
สตับจะถูกสร้างขึ้นโดยการใช้ rmic คอมไพเลอร์

2.6.6.2 The Semantics of Object Methods Declared final

เมธอดที่ได้รับการประการเป็น final ในคลาส java.lang.Object และไม่ได้ถูก โอเวอร์ไรด์โดยการอิมพลีเมนต์ใด ๆ

- getClass
- notify
- notifyAll
- wait

ค่าดั้งเดิมที่ทำการอิมพลีเมนต์ให้กับ getClass นั้นเป็นค่าที่เหมาะสำหรับออบเจ็กต์ของจาวาต่าง ๆ ทั้งโลคอลและรีโมท ดังนั้นเมธอดไม่จำเป็นต้องมีการอิมพลีเมนต์รีโมทออบเจ็กต์เพิ่มขึ้นอีก เมื่อใช้บนรีโมทสตับ เมธอด getClass จะทำการรายงานไทป์ที่แน่นอนของสตับออบเจ็กต์, ซึ่งถูกสร้างโดย rmic จะสังเกตว่า

สตับไทป์นี้จะสะท้อนเพียงรีโมทอินเตอร์เฟสซึ่งทำการอิมพลีเมนต์โดยรีโมทออบเจ็กต์ ไม่ใช่โลคอลอินเตอร์เฟส

เมธอด wait และ notify ของ java.lang.Object จะเน้นถึงการคอยและการสังเกตในปริบทของแบบจำลองที่เป็นเธรดของภาษาจาวา ขณะที่การใช้เมธอดของรีโมทสตับเหล่านี้ไม่ได้ทำให้การทำงานของเธรดของจาวาหยุดชะงักแต่อย่างไร เมธอดเหล่านี้ไม่ได้มีความหมายเดียวกับการที่มันทำให้กับออบเจ็กต์ของจาวาโดยเฉพาะอย่างยิ่งการใช้เมธอดเหล่านี้ในการอ้างถึงบนโบคอลออบเจ็กต์ไปยังรีโมทออบเจ็กต์ (สตับ) ไม่เหมือนกับออบเจ็กต์ทางฝั่งของรีโมท

2.6.6.3 The Skeleton Interface

อินเตอร์เฟส Skeleton ถูกนำมาใช้โดยการอิมพลีเมนต์ของ skeleton ที่สร้างโดย rmic คอมไพล์เลอร์ ส่วน skeleton สำหรับรีโมทออบเจ็กต์นั้นเป็นส่วนหนึ่งของทางฝั่งเซิร์ฟเวอร์ซึ่ง ใช้เพื่อเรียกสำหรับการทำอิมพลีเมนต์รีโมทออบเจ็กต์อย่างแท้จริง

เมธอด dispatch ไม่ได้รวมอาร์กิวเมนต์ใด ๆ จาก input สตรีม ที่ได้จากการเรียกไปยังออบเจ็กต์ เรียกถึงเมธอดบนรีโมทออบเจ็กต์ที่ทำการอิมพลีเมนต์แล้ว , obj, และอาจทำการรวมค่ากลับหรือคืนค่าเป็น exception ถ้าเกิดสิ่งใดสิ่งหนึ่งขึ้นระหว่างการเรียก

2.6.7 The Interface DGC

DGC ที่มีอยู่ทางฝั่งเซิร์ฟเวอร์ถูกนำมาใช้เพื่อสร้างอัลกอลิทึ่มสำหรับการทำ garbage collection แบบกระจาย อินเตอร์เฟสนี้ประกอบไปด้วยเมธอดสองอันคือ dirty และ clean. ซึ่ง การเรียก dirty จะทำให้การอ้างถึงรีโมทมีการกระจายออกของข้อมูลในฝั่งไคลเอนต์(ทางฝั่งไคลเอนต์ใช้ VMID) ส่วนการเรียก clean จะเรียกเมื่อไม่มีการอ้างถึงรีโมทอีกแล้วในฝั่งไคลเอนต์ การเรียก dirty แล้วมีการผิดพลาดจะทำให้ต้องจัดระเบียบในการ clean เป็นอย่างมาก ดังนั้นตัวเลขที่เรียกเป็นลำดับสามารถนำมาเพื่อรักษาการเจอการเรียกอีกในโอกาสต่อไป

การอ้างถึงรีโมทออบเจ็กต์จะถูก leased สำหรับช่วงเวลาหนึ่งโดยไคลเอนต์เพื่อทำการอ้างอิงถึง ระยะเวลาในการทำการ lease จะเริ่มต้นเมื่อได้รับการเรียก dirty. ความรับผิดชอบของไคลเอนต์คือเพื่อ ทำการอ้าง lease อีกครั้ง โดยการเรียก dirty ในการอ้างถึงรีโมทซึ่งมันจะคอยจนกระทั่ง การ lease นั้นหมดเวลา ถ้าฝั่งไคลเอนต์ไม่ได้ทำการ lease ก่อนที่มันจะหมดเวลา distributed garbage collector จะสมมุติเองว่ารีโมทออบเจ็กต์ไม่มีการอ้างถึงที่ไคลเอนต์นั้น

เมธอด dirty จะร้องของการ lease สำหรับการอ้างถึงรีโมทออบเจ็กต์ที่ทำการผนวกด้วยการบ่งชี้ถึงออบเจ็กต์ที่รวมอยู่ในกลุ่มของอาร์กิวเมนต์ ids . สำหรับ lease จะบรรจุสิ่งที่เป็นลักษณะเอกเทศของvirtual machine identifier(VMID) และระยะเวลาที่ร้องของการ lease สำหรับการส่งออกของออบเจ็กต์แต่ละครั้งใน

โลคอลออบเจ็กต์ของ local virtual machine ตัว garbage collector นั้นจะรวมเอา reference list ซึ่งเป็นระเบียนของไคลเอนต์ซึ่งได้ทำการอ้างอิงไว้ก่อน ถ้า lease นั้นยอมรับ

garbage collector จะเพิ่มส่วนที่เป็นไคลเอนต์ไปยังรายการที่อ้างอิงถึงสำหรับรีโมทออบเจ็กต์แต่ละตัวว่า , ids, ส่วนพารามิเตอร์ sequenceNum เป็นตัวเลขที่เรียงกันใช้เพื่อตรวจพบและยกเลิกการเรียกที่ล่าช้าไปยัง garbage collector และตัวเลขที่เรียงกันนี้ควร เป็นลำดับเพิ่มขึ้นแต่ละครั้งหลังการเรียกใช้ garbage collector ไคลเอนต์บางตัวสามารถที่จะสร้าง VMID ที่เป็นเอกเทศ ที่เป็นเช่นนี้เพราะว่า VMID โดยทั่วไปแล้วจะมีตัวบ่งที่แต่ต่างกันอยู่แล้วถ้ามีการใส่แอดเดรสจริง ๆ ให้กับมัน และแอดเดรสซึ่งในบางไคลเอนต์ไม่สามารถที่จะจำกัดสิทธิ์ในการทำการป้องกัน

2.7 ภาษา UML

ภาษา UML เหมาะสำหรับการแสดงโครงสร้างและความสัมพันธ์ของระบบที่ซับซ้อน เพื่อจำลองการทำงานของโปรแกรมแบบระเบียบวิธีของวัตถุ (Object Oriented :OO) เริ่มใช้โดยกลุ่ม OMG ซึ่งเป็นกลุ่มผู้พัฒนาวิธีมาตรฐานสำหรับการพัฒนาโปรแกรมแบบ OOP

2.7.1 คำนิยาม

Object – มีโครงสร้างคล้ายคลังที่เก็บฟังก์ชันเอาไว้ภายใน ซึ่งมีองค์ประกอบด้วยกันหลายส่วน คือมีทั้ง Attribute Behavior และ State

Attribute – ในทาง OO จะหมายถึงข้อมูลที่ถูกห่อไว้ภายใน Object บางภาษาที่มีความสามารถทาง OOP อาจถือได้ว่าชนิดของข้อมูลเป็น Object เช่น Integer ถือว่าเป็น Object ด้วย แต่ในความเป็นจริงพวกตัวแปรต่างๆ ที่อยู่เป็นชนิดของข้อมูล เช่น Integer หรือ Float ยังไม่เป็น Object

Behavior – สามารถแบ่งออกเป็น 3 ชนิด คือ

1. Simple สามารถประมวลผลโดยไม่ขึ้นกับสถานะก่อนหน้านั้น เช่น ฟังก์ชัน sin(), cos() ,tan ()
2. Automaton เป็นลักษณะเป็นเครื่องจักรคำนวณ ซึ่งจะมีสถานะ ทำงานไปตามสถานะแต่ไม่ขึ้นอยู่กับเหตุการณ์ก่อนหน้าหลังใด ๆ
3. Continuous เป็นการทำงานแบบต่อเนื่อง ไม่มีขอบเขต สามารถรับข้อมูลเป็นสตรีม การทำงานขึ้นกับสถานะที่แล้ว อินพุตที่รับเข้ามา

Behavior ที่มีใช้อยู่ในออบเจ็กต์จะเป็น 3 ชนิดนี้รวมกันอยู่

Responsibility – บทบาทของออบเจ็กต์ที่มีต่อระบบ

Class – เป็นที่รวบรวมออบเจ็กต์ที่มีลักษณะคล้าย ๆ กันเอาไว้ด้วยกัน

2.7.2 ความสัมพันธ์ระหว่างคลาสและออบเจ็กต์

ในภาษา UML ความสัมพันธ์ระหว่าง คลาส มีอยู่ด้วยกัน 5 ชนิด คือ association, aggregation, composition, generalization, refinement

association แสดงเป็นเส้นตรงลากเชื่อมระหว่างออบเจ็กต์ ออบเจ็กต์สามารถสื่อสารกันได้ทั้ง 2 ทาง ถ้าหากเป็นทางเดียวจะมีหัวลูกศรเปิดกำกับฝ่ายที่เป็นผู้รับ

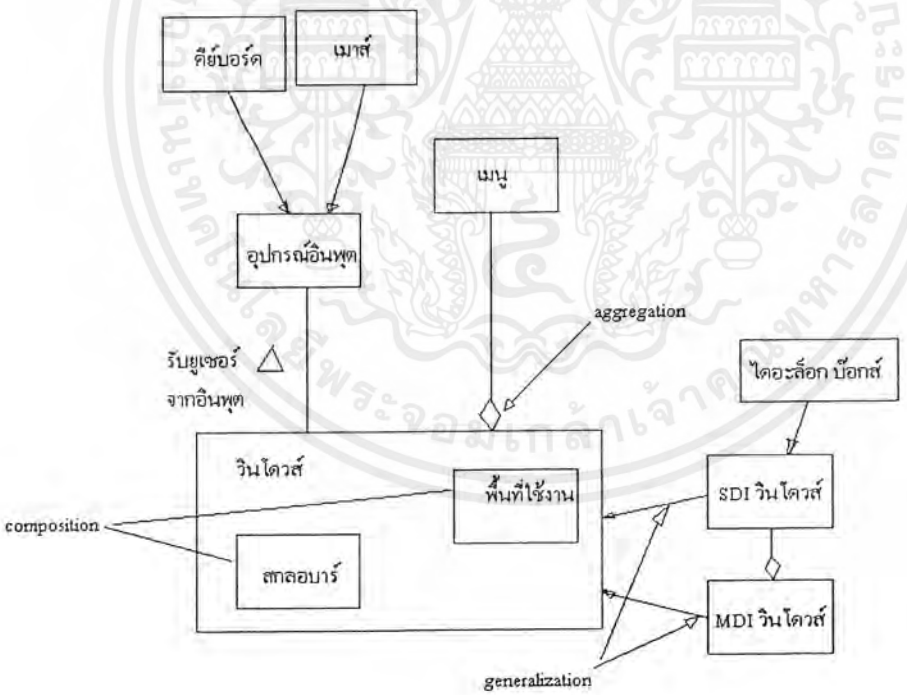
aggregation association แสดงเป็นเส้นตรงมีปลายสี่เหลี่ยมเพชรอยู่ที่ผู้ที่เป็นเจ้าของความสัมพันธ์

composition มีลักษณะของ aggregation รวมอยู่ด้วยแต่สามารถที่จะสร้างหรือทำลายองค์ประกอบที่แสดงความเป็นเจ้าของได้

generalization แสดงเป็นเส้นตรงปลายเป็นหัวลูกศรชี้ไปที่คลาสที่สืบทอดคุณสมบัติมา เป็นความสัมพันธ์ในรูปแบบหนึ่งคล้ายคุณสมบัติการสืบทอด

2.7.3 การใช้งาน UML

ตัวอย่าง ของวินโดวส์แสดงคลาสไดอะแกรม ดังรูป



รูปที่ 2-7 คลาสของวินโดวส์

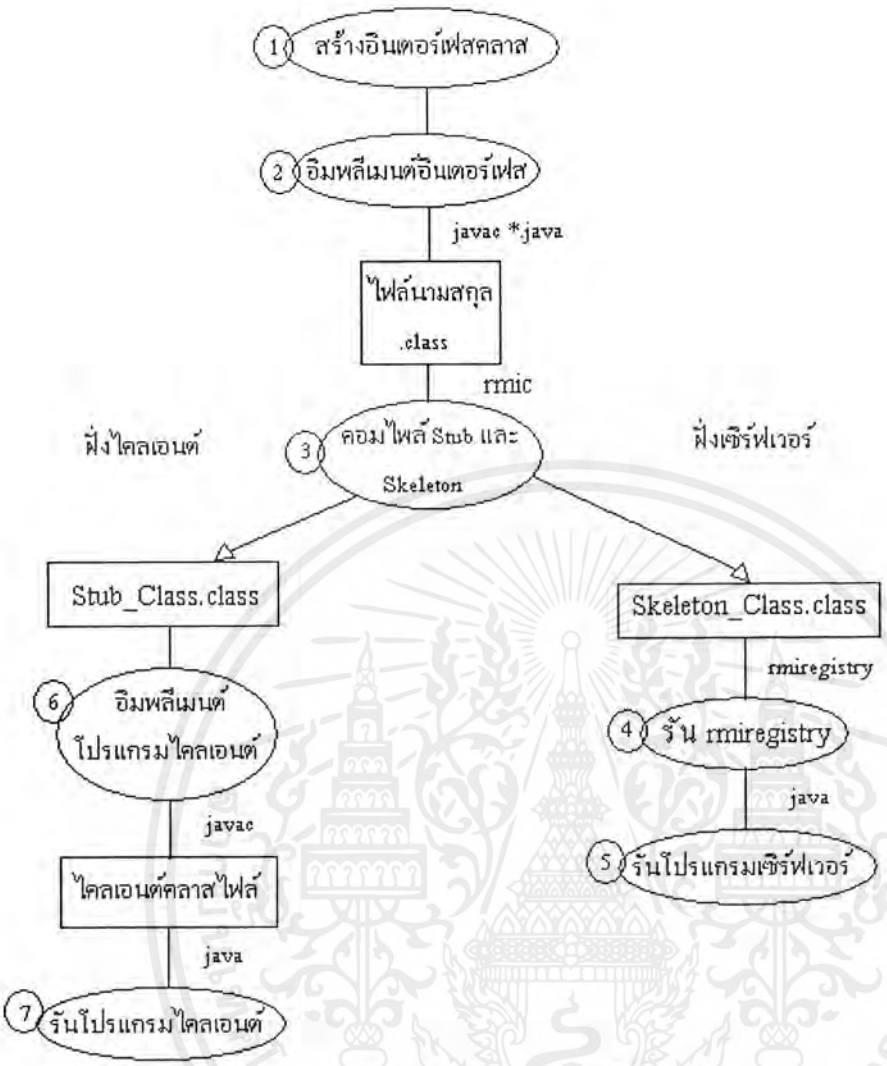
บทที่ 3
การออกแบบและการสร้าง

3.1 ขั้นตอนในการพัฒนาโปรแกรม RMI

เนื่องจากเป็นการพัฒนาโปรแกรมที่มีการเรียกใช้งานรีโมทฟังก์ชั่น ดังนั้นจึงต้องมีการระบุลักษณะของอินเตอร์เฟส และ การทำเอ็กเซปชั่นต่าง ๆ ซึ่งขั้นตอนในการพัฒนาจะดำเนินไปในลักษณะนี้

- ออกแบบคลาสโดยวิธีการของภาษา UML เพื่อกำหนดแอตตริบิวท์ และ วิธีการต่าง ๆ
- จัดเตรียมโปรแกรม Java Compiler เตรียมคอมไพเลอร์ดังนี้คือ VisualCafe 3.0
- Implement Program with Interface
  ออกแบบอินเตอร์เฟสของโปรแกรม
- Implement all those interface
  ทำการ Implement คลาสที่ได้กำหนดเป็นอินเตอร์เฟสไว้
- Compile Implement Class and stub
  ทำการคอมไพล์ทุกคลาสและคอมไพล์สตับ
  javac *.java
  rmic < Server และ Client ไฟล์ .java >
- Run rmiregistry
  ใช้คำสั่ง Start rmiregistry ที่ตำแหน่งที่เก็บ stub และ skeleton คลาสไฟล์
- Run program for SERVER
  รันโปรแกรมเซิร์ฟเวอร์เพื่อรอการเรียกใช้จากไคลเอนต์
- Implement program Client
  หลังจากสร้างโปรแกรมฝั่งเซิร์ฟเวอร์เสร็จ จึงมาทำฝั่งไคลเอนต์ต่อไป
- Run program for Client
  เรียกใช้โปรแกรมเซิร์ฟเวอร์ด้วยโปรแกรมไคลเอนต์

การพัฒนาโปรแกรมแบบ RMI มีลำดับขั้นและวิธีการเป็นลำดับ ซึ่งสามารถพัฒนาทางฝั่ง Server และฝั่ง Client ไปพร้อม ๆ กันได้ แต่เนื่องจาก Client จะเรียกใช้บางฟังก์ชั่นจากเซิร์ฟเวอร์ เพื่อการดีบักโปรแกรม ดังนั้นการพัฒนาทางฝั่งเซิร์ฟเวอร์จะสิ้นสุดก่อนทางฝั่งไคลเอนต์ ซึ่งสามารถแสดงเป็นแผนภาพในการพัฒนาโปรแกรม ดังรูป 3-1



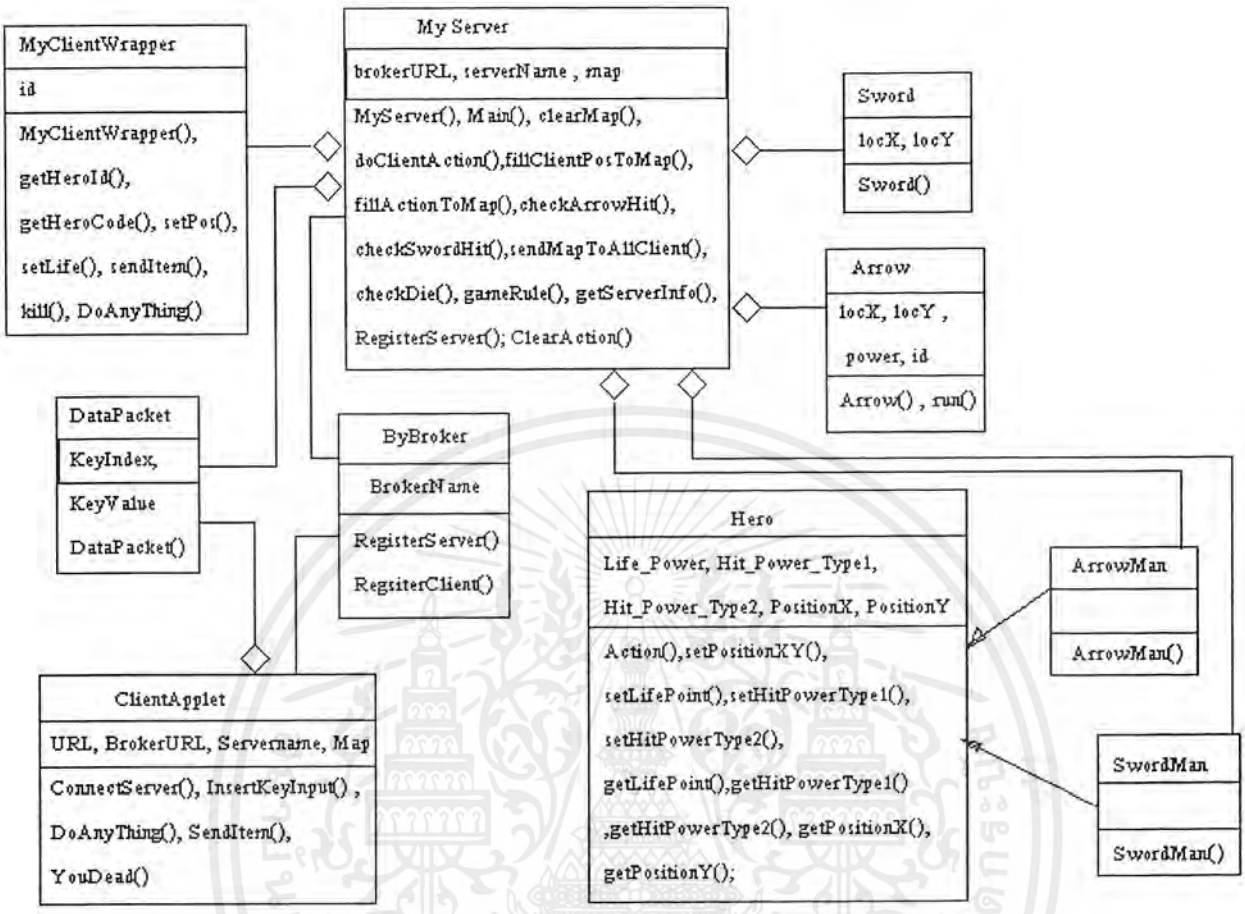
รูปที่ 3-1 ขั้นตอนการพัฒนาโปรแกรม

3.2 การออกแบบและการสร้างโปรแกรม RmiGame

ดังวิธีที่ได้นำเสนอทฤษฎีและหลักการของภาษา UML ซึ่งเป็นแบบจำลองสำหรับการสร้างโปรแกรมแบบระเบียบวิธีทางวัตถุ (Object Oriented) สามารถออกแบบคลาสที่จำเป็นสำหรับโปรแกรมได้

3.2.2 องค์ประกอบต่าง ๆ ของโปรแกรม

เมื่อทำการกำหนดแอตตริบิวต์ และ เมธอดที่เหมาะสม ใช้ขบวนการในการสร้างคลาส สามารถแสดงเป็นคลาสไดอะแกรมได้ ดังรูป 3-2



รูปที่ 3-2 แสดงคลาสไดอะแกรมของ RmiGame

จากคลาสไดอะแกรมอธิบายความสัมพันธ์ตามเส้นที่เชื่อมโยงได้

MyServer <----------> MyBroker <----------> ClientApplet

จะได้ว่า MyServer มีการสื่อสารกับ MyBroker และ ClientApplet มีการสื่อสารกับ MyBroker ซึ่ง MyBroker เป็นเสมือนศูนย์กลางในการสื่อสาร ซึ่งทั้ง MyServer และ ClientApplet มาทำการลงทะเบียนเพื่อขอใช้งานร่วมกันระหว่าง MyServer และ ClientApplet

หลังจากที่ MyBroker ติดตั้งเรียบร้อย จะต้องมีเซิร์ฟเวอร์เข้ามาลงทะเบียนเพื่อเก็บเป็นระเบียนของเซิร์ฟเวอร์ ต่อจากนั้นไคลเอนต์จึงเข้ามาลงทะเบียน MyBroker จะทำการส่งรายการชื่อของเซิร์ฟเวอร์ที่ได้ทำการลงทะเบียนกับ MyBroker ไปให้กับไคลเอนต์ที่เรียกเข้ามาลงทะเบียน

จากคลาส Hero มีหัวลูกศรแสดงว่า ArrowMan กับ SwordMan ต่างก็เป็นสับคลาสของ Hero

extend

ArrowMan -------> Hero

ได้ว่า ArrowMan เป็นสับคลาสของ Hero

extend

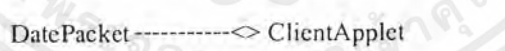
SwordMan ------ > Hero

SwordMan เป็นสับคลาสของ Hero

MyServer มี aggregate หรือส่วนประกอบ ดังนี้คือ MyClientWrapper, DataPacket, Sword, Arrow, ArrowMan, SwordMan ถ้าขาดส่วนประกอบใดไป MyServer จะไม่สามารถคอมไพล์ได้

สำหรับ MyClientWrapper เป็นคลาสที่เชื่อมต่อระหว่าง ArrowMan , SwordMan และ MyServer ให้สามารถส่งข้อมูลผ่านถึงกันได้ โดย MyClientWrapper จะทำหน้าที่เป็นตัวกลาง

คลาส Arrow สืบทอดมาจาก thread ซึ่งทำงานตามช่วงเวลาที่ได้กำหนดไว้ในคลาส และจะ dispose ตัวเองออกจากระบบได้



ClientApplet มี aggregate คือ DataPacket เป็นส่วนประกอบ ซึ่ง DatePacket นี้จะเป็นคลาสเดียวกันกับที่เรียกใช้โดย MyServer ซึ่ง DatePacket นี้เป็นออบเจ็กต์ที่ถูกส่งผ่านขณะสื่อสารระหว่างไคลเอนต์และเซิร์ฟเวอร์

การสร้างแอพพลิเคชั่นสามารถสร้างจากโปรแกรม VisualCafe 3.0 แล้วทำการคอมไพล์ทุกคลาสและคอมไพล์สตับ จะได้แอพเพล็ตสำหรับรันโปรแกรม

3.3 การสร้างโปรแกรม Listings

3.3.1 เขียนคลาสไดอะแกรมได้ ดังรูป 3-3

รูปที่ 3-3 คลาสไดอะแกรมของโปรแกรม Lisings

3.3.2 ขั้นตอนการเรียกใช้งานโปรแกรม Listings

- คอมไพล์ไฟล์นามสกุล .java ทุกไฟล์

java *.java

- คอมไพล์สตับไฟล์ด้วยสตับคอมไพเลอร์

rmic RMIServer

- เริ่มโปรแกรม rmiregistry

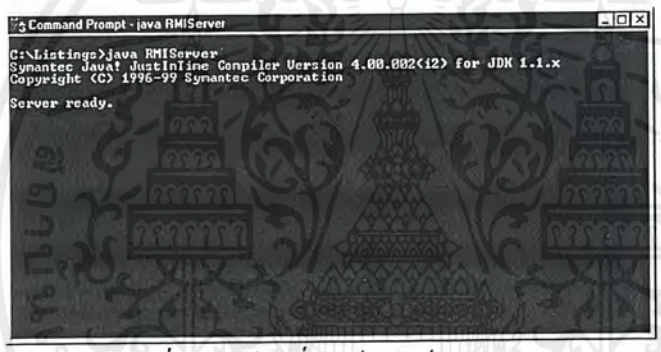
start rmiregistry

- เริ่มโปรแกรมเซิร์ฟเวอร์

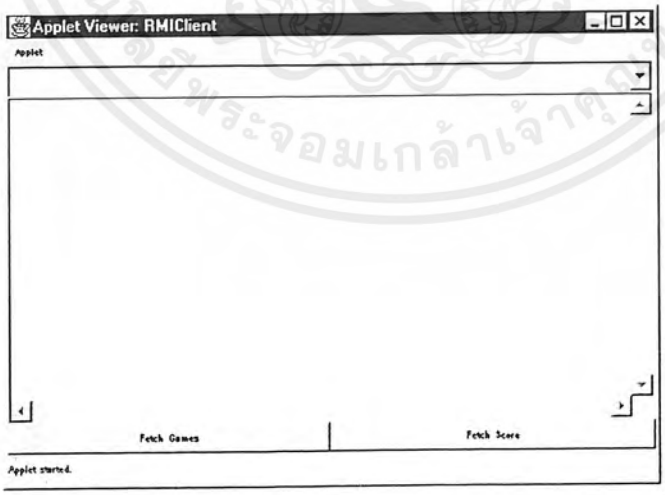
java RMIServer

- เรียกใช้โปรแกรมไคลเอนต์

appletviewer RMIClient.html



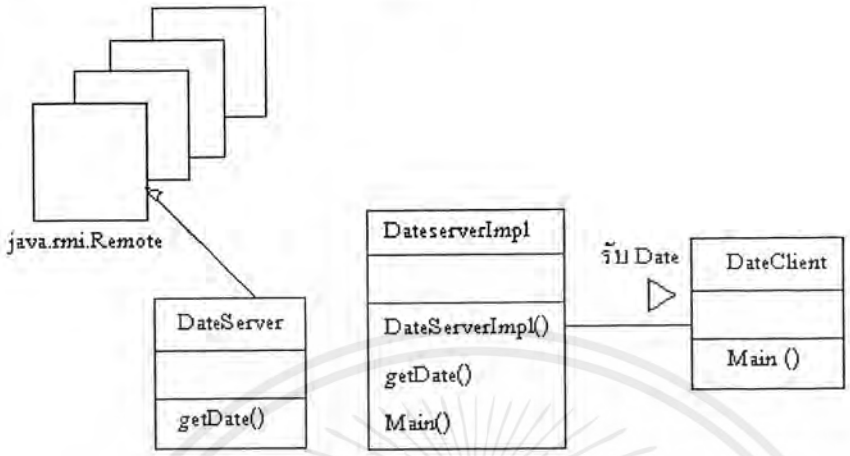
รูปที่ 3-4 หน้าจอฝั่งเซิร์ฟเวอร์ของ Listings



รูปที่ 3-5 หน้าจอไคลเอนต์แอพเพล็ตของ Listings

3.4 การสร้างโปรแกรม DateRMI

3.4.1 สามารถแสดงเป็นคลาสไดอะแกรมได้ ดังรูป 3-4



รูปที่ 3-6 คลาสไดอะแกรมของ DateRMI

จากคลาสไดอะแกรมจะพบว่า DateServer เป็นอินเตอร์เฟสคลาส ซึ่งทำการ extend มาจาก java.rmi.Remote ซึ่งเป็นมาตรฐานสำหรับไฟล์อินเตอร์เฟส ที่ใช้งานเพื่อเรียกใช้ออบเจ็กต์แบบรีโมท

DateserverImpl เป็นอิมพลีเมนต์ของ DateServer ซึ่งได้ระบุรายละเอียดของฟังก์ชั่นต่างๆ DateClient เป็นไคลเอนต์โปรแกรมที่สื่อสารแบบรีโมทกับ DateserverImpl ซึ่งเป็นการรับค่า Date จาก DateserverImpl เพื่อมาแสดงทางฝั่งไคลเอนต์

3.4.2 ขั้นตอนการเรียกใช้งานโปรแกรม DateRMI

1. คอมไพล์โปรแกรมที่มีนามสกุล .java ทั้งหมด

   java *.java
2. สร้างสตับโดยคำสั่ง rmic

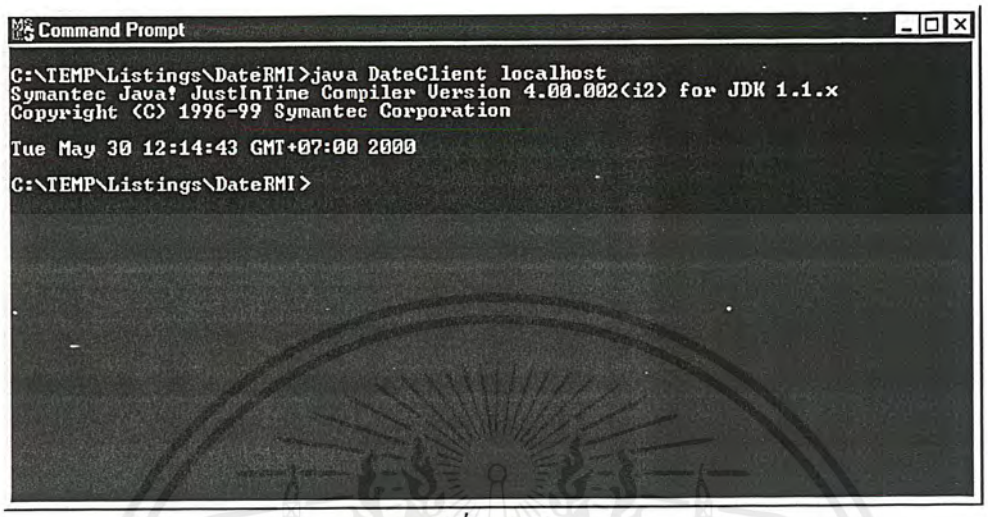
   rmic DateServerImpl
3. เรียก rmiregistry

   start rmiregistry
4. เรียก DateserverImpl

   java DateserverImpl
5. เรียก DateClient ตามด้วยหมายเลข IP หรือ localhost

   java DateClient <IP-Serverhost>

แสดงหน้าจอของโปรแกรม DateRMI จะแสดงวันที่และเวลาของเครื่องเซิร์ฟเวอร์ ส่งมาที่ไคลเอนต์โปรแกรม ดังรูป 3-7



รูป 3-7 แสดงวันที่ DateClient

บทที่ 4
การทดลองและผลการทดลอง

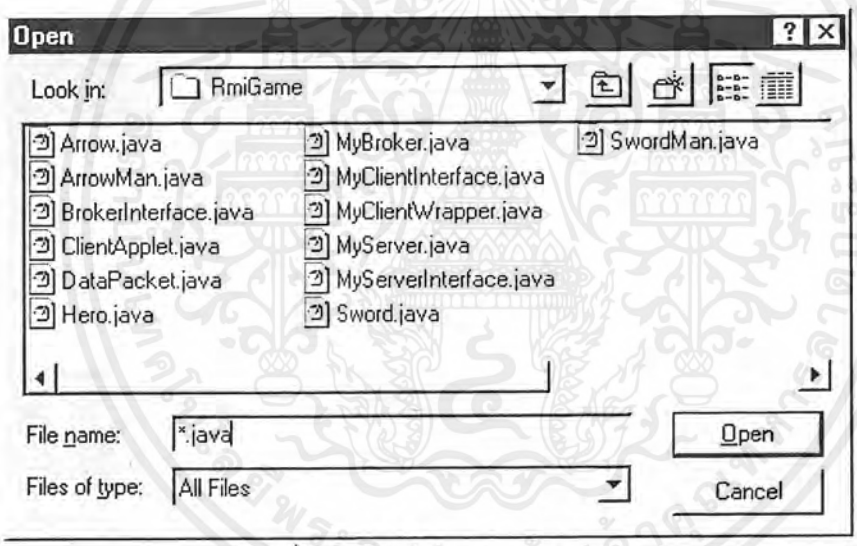
มีโปรแกรมทดสอบทั้งหมด 3 โปรแกรมคือ RmiGame, DateRmi และ Listings ซึ่งเขียนขึ้นโดยใช้ภาษาจาวา มีเครื่องมือที่ใช้คือ JDK 1.1.6

4.1 โปรแกรม RmiGame

เป็นการทดลองโดยระบุการทำงานเป็นโลคอลโฮสต์

4.1.1 ตรวจดูไฟล์ทั้งหมดของเกมส์

มีทั้งสิ้น 13 ไฟล์ เพราะว่าทุกไฟล์มีความสัมพันธ์กัน ดังนั้นก่อนที่จะรันโปรแกรมไฟล์ทั้งหมดจะต้องอยู่ในโฟล์เดอร์เดียวกัน



รูปที่ 4-1 ซอร์สโค้ดของ RmiGame

4.1.2 ทำการคอมไพล์โปรแกรม RmiGame

มีขั้นตอนดังต่อไปนี้

- ใช้จาวาคอมไพเลอร์คอมไพล์โปรแกรมด้วยคำสั่ง

  javac RmiGame*.java

- ใช้ RMI Stub Compiler เพื่อทำการสร้าง Stub class และ Skeleton class

  rmic RmiGame.MyBroker –d .

  rmic RmiGame.MyServer -d .

  rmic RmiGame.ClientApplet -d .

- เรียกใช้โปรแกรมตามลำดับดังต่อไปนี้

start rmiregistry

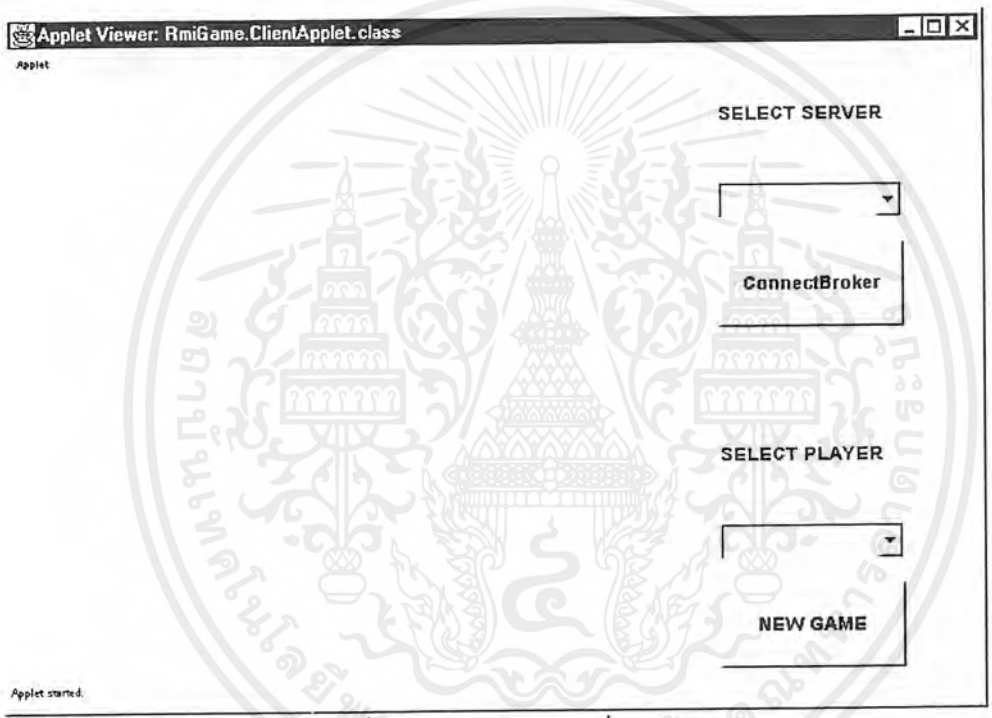
start java RmiGame.MyBroker

start java RmiGame.MyServer

start appletviewer ClientApplet.html

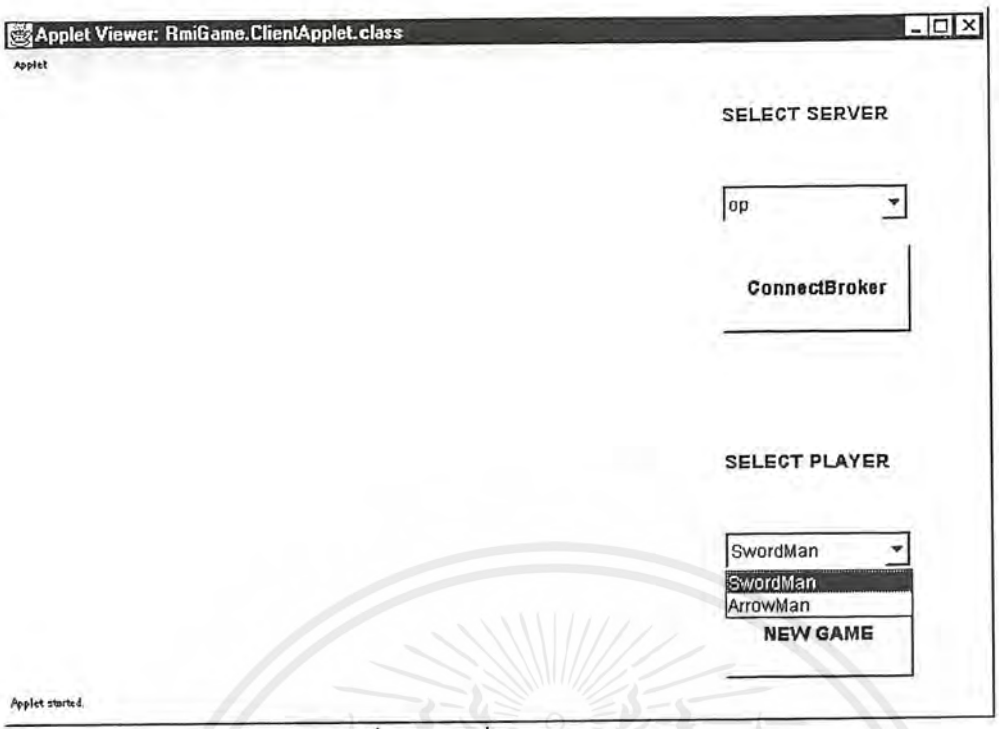
4.1.3 เข้าสู่การทำงานของโปรแกรม RmiGame

เมื่อเรียก Appletviewer ClientApplet.html จะขึ้นหน้าจอ โดยมีปุ่ม ConnectBroker เมื่อกดปุ่มนี้ โปรแกรมไคลเอนต์จะเรียกไปยัง broker เพื่อรายงานการเป็นไคลเอนต์กับ broker



รูปที่ 4-2 หน้าจอก่อนการเชื่อมต่อ

หลังจากทางฝั่ง broker ได้รับการเรียกโดยไคลเอนต์แล้ว จะแสดงคำว่า "add client" ที่หน้าจอ หลังจากนั้นทางฝั่งไคลเอนต์จะมีการสร้างหัวข้อในเท็กซ์บ็อกซ์ ดังรูป



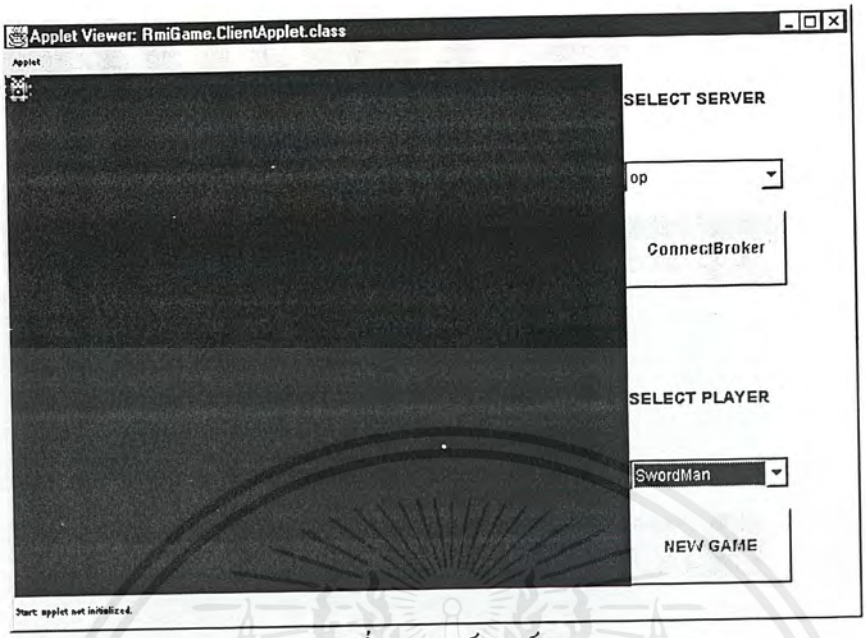
รูปที่ 4-3 การเชื่อมต่อกับ broker

หลังจากที่ broker ได้เชื่อมต่อเรียบร้อยแล้ว จะเกิดชื่อของตัวผู้เล่นขึ้นที่ช่อง SELECT PLAYER มีผู้เล่นอยู่ 2 ชนิดคือ เป็น Sword Man หรือ Arrow Man หลังจากนั้นให้กดปุ่ม NEW GAME จะทำการเริ่มเกมส์ใหม่

ขณะนี้ทางฝั่งเซิร์ฟเวอร์พร้อมที่จะให้บริการเรียกใช้โปรซีเยอร์แล้ว ทางฝั่งไคลเอนต์จะทำการตอบรับโดยการร้องขอข้อมูลเบื้องต้นของเกมส์ (Game Information) ซึ่งจะทำการ โหลดแผนที่ อยู่ที่เซิร์ฟเวอร์มาแสดงที่หน้าจอ ซึ่งการแสดงแผนที่ออกเป็นภาพนี้เป็นหน้าที่ของไคลเอนต์เอง

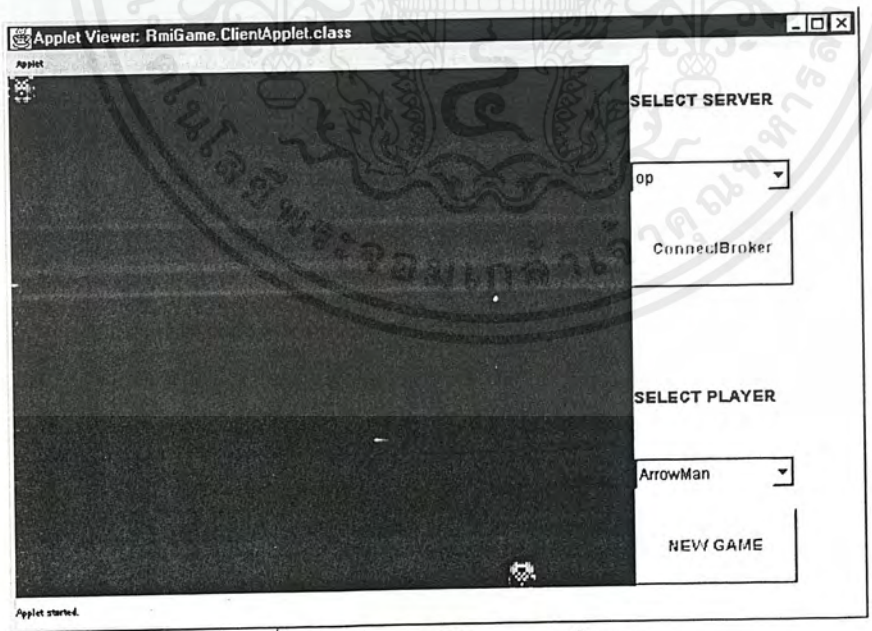
กรณีที่ไคลเอนต์เรียกใช้รีโมทโปรซีเยอร์ ถ้าหากไม่ได้ทำการรันโปรแกรมเซิร์ฟเวอร์ไว้ก่อน จะขึ้นข้อความแสดงข้อผิดพลาดขึ้นที่ DOS mode เพื่อบอกให้ใช้ที่เป็นไคลเอนต์ทราบ ให้ตรวจดูว่าที่เซิร์เวอร์มีข้อความว่า “Server Ready” ขึ้นถ้าไม่มีให้รันโปรแกรมเซิร์ฟเวอร์อีกครั้ง

แสดงหน้าจอเกมส์ โดยผู้เล่นชนิด Sword Man ขึ้นมาที่มุมหนึ่งของสนาม ดังรูปที่ 4-4



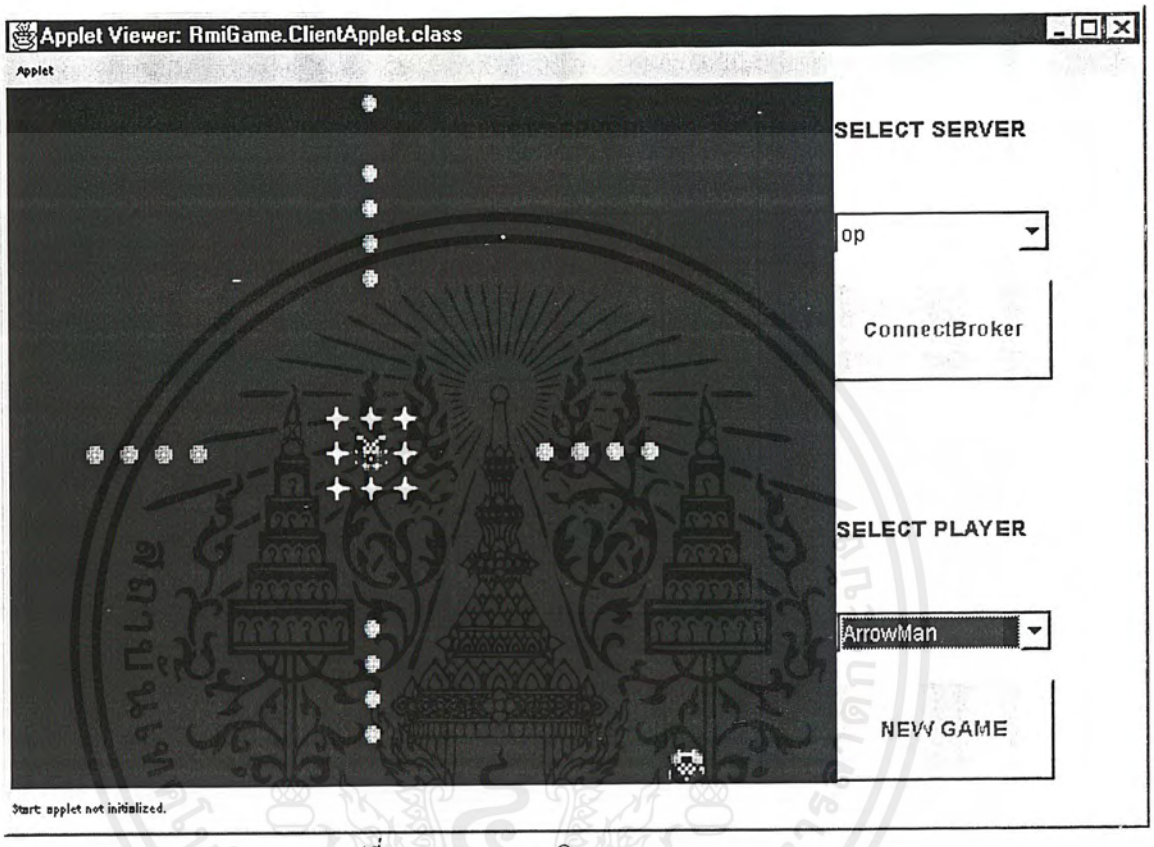
รูปที่ 4-4 เกมส์สตาร์ท

หลังจากเรียกโปรแกรมไคลเอนต์ดังรูป 4-4 ไคลเอนต์และเซิร์ฟเวอร์พร้อมสำหรับรันโปรแกรม สามารถมีผู้เล่นได้ตั้งแต่ 1 ตัวขึ้นไป ทำการรันไคลเอนต์โปรแกรมอีก จะแล้วเลือก SELECT PLAYER เป็น Arrow Man จะเป็นผู้เล่นทั้ง Sword Man และ Arrow Man ซึ่งต่างก็เป็นไคลเอนต์



รูปที่ 4-5 รันไคลเอนต์โปรแกรม 2 โพรเซส

ทดลองให้ผู้เล่น Sword man ปล่อยอาวุธ โดย เมื่อกด ปุ่ม 1 จะยิง ออกเป็น 4 แฉก รอบตัวแต่หากกดปุ่ม 2 จะยิง ออกเป็น 8 แฉกรอบตัว ดังรูป 4-6



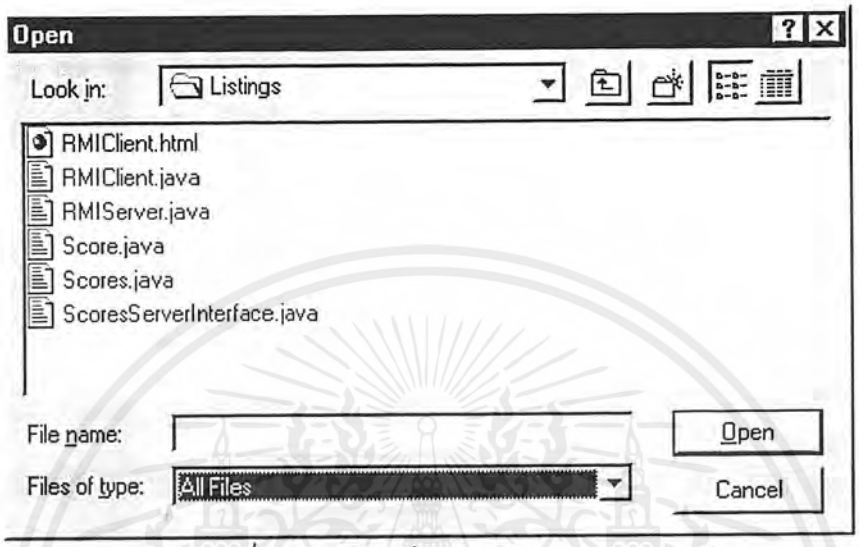
รูปที่ 4-6 ทดสอบการยิง

ถ้าฝ่ายใดฝ่ายหนึ่งค่าชีวิตหมด ผู้เล่นของฝ่ายนั้นก็จะหายไปจากแผนที่ คงเหลือผู้ที่ยังมีค่าชีวิตอยู่บนแผนที่

4.2 โปรแกรม Listings

เป็นโปรแกรมที่แสดงการเรียกข้อมูลจากเซิร์ฟเวอร์ที่อยู่ต่างเครื่องมาแสดงบนแอพเพล็ต ของเครื่องที่เป็นไคลเอนต์ เป็นตัวอย่างการรัน RMI ระหว่างเครื่องที่อยู่ในเครือข่ายเน็ตเวิร์ก

4.2.1 ไฟล์ทั้งหมดของโปรแกรม



รูปที่ 4-7 แสดงไฟล์ของ Listings

4.2.2 ทำการคอมไพล์โปรแกรม

- คอมไพล์โปรแกรมนามสกุล . java ทั้งหมด ใช้คำสั่ง

  javac *.java

- สร้างคลาสให้กับ RMIServer.class โดยใช้ rmic (Stub Compiler)

  rmic RMIServer

- ทำการเริ่มโปรแกรมรีจิสตรี

  start rmiregistry

- เริ่มเซิร์ฟเวอร์

  java RMIServer

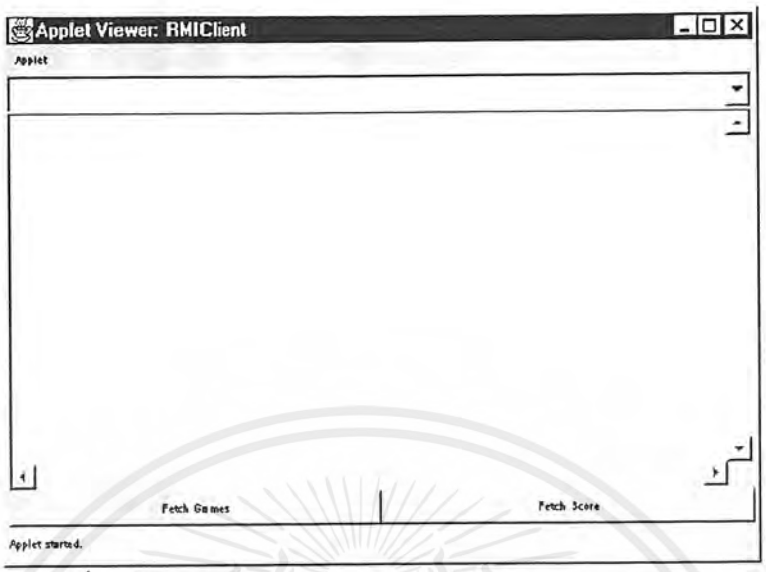
- เริ่มไคลเอนต์

  appletviewer RMIClient.html

หลังจากเริ่มโปรแกรมไคลเอนต์แล้ว ถ้าไม่สามารถเรียกข้อมูลจากทางเซิร์ฟเวอร์มาแสดงได้ ให้ตรวจดูที่ค่า IP ของโปรแกรมไคลเอนต์ว่าตรงกับค่า IP ของเครื่องที่รันโปรแกรมเซิร์ฟเวอร์หรือไม่ ถ้าไม่ตรง จะต้องคอมไพล์โปรแกรมอีกครั้ง แล้วจะต้องรันโปรแกรมเซิร์ฟเวอร์ไว้พร้อมก่อนที่จะเรียกโปรแกรมไคลเอนต์เสมอ

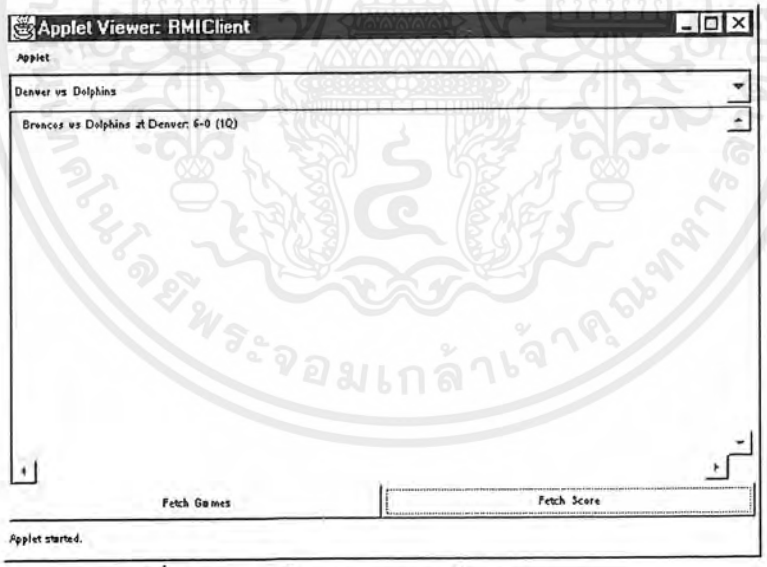
4.2.3 เริ่มต้นโปรแกรม

เมื่อเริ่มโปรแกรมจะขึ้นหน้าจอโปรแกรมแอพเพล็ต



รูปที่ 4-8 ไคลเอนต์แอพเพล็ตของโปรแกรม Listings

เมื่อต้องการให้แสดงข้อมูลให้กดปุ่ม Fetch Games จะขึ้นข้อมูลที่เรียกมาจากฝั่งเซิร์ฟเวอร์ขึ้นในเท็กซ์เอเรีย ซึ่งแสดง score ของแต่ละทีม เมื่อต้องการให้อัพเดตคะแนนให้กดปุ่ม Fetch Score อีกครั้ง



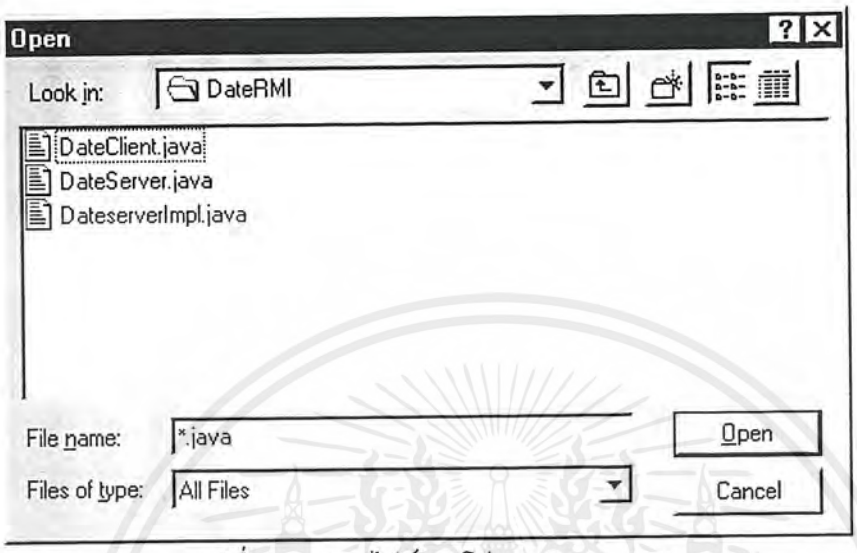
รูปที่ 4-9 แสดงข้อมูลบนแอพเพล็ต RMIClient

หลังจากที่เรียกข้อมูลมาแสดงบนไคลเอนต์แล้ว ข้อมูลทางฝั่งเซิร์ฟเวอร์จะถูกอัพเดตต่อไป แต่ยังไม่มีผลมายังไคลเอนต์ ต้องกด Fetch Score อีกครั้งเพื่ออัพเดตข้อมูลจากเซิร์ฟเวอร์

4.3 โปรแกรม DateRMI

แสดงวันที่และเวลา โดยสามารถใส่เลข IP เพื่อต่อไปยังเครื่องปลายทางได้

4.3.1 ไฟล์ทั้งหมดของโปรแกรม



รูปที่ 4-10 แสดงไฟล์ของโปรแกรม DateRMI

4.3.2 การคอมไพล์โปรแกรม DateRMI

ให้คอมไพล์โปรแกรมที่ดอสโหมด

- คอมไพล์โปรแกรมนามสกุล . java ทั้งหมด ใช้คำสั่ง

javac *.java

- สร้างคลาสให้กับ RMIServer.class โดยใช้ rmic (Stub Compiler)

rmic DateserverImpl

- ทำการเริ่มโปรแกรมรีจิสตรี

start rmiregistry

- เริ่มเซิร์ฟเวอร์

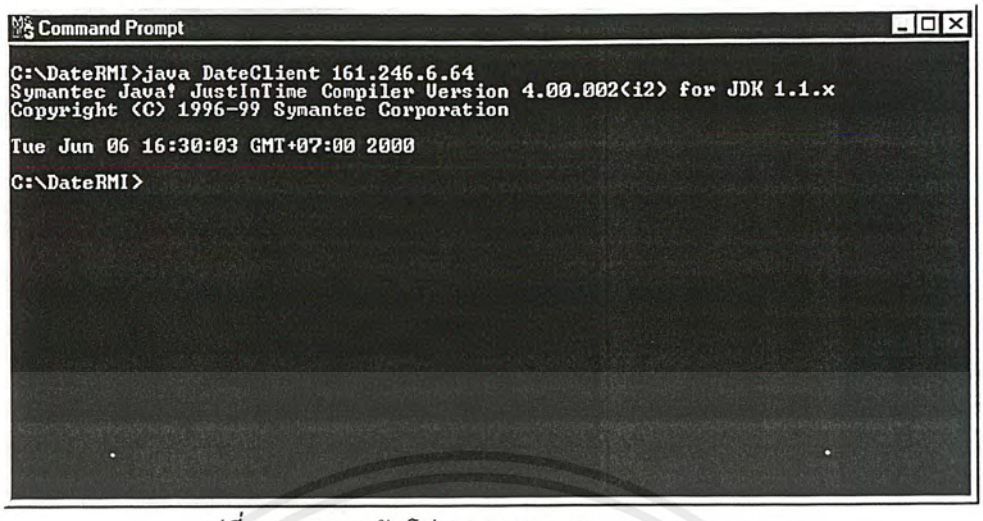
java DateserverImpl

- เริ่มไคลเอนต์

java DateClient <ip>

ให้ใส่ค่า IP เป็น localhost หรือ IP Address ของเครื่องที่รันโปรแกรมเซิร์ฟเวอร์

4.3.3 แสดงโปรแกรม DateRMI



รูปที่ 4-11 ผลการรันโปรแกรม DateClient

บทที่ 5
บทสรุปและวิจารณ์

5.1 สรุปผลการดำเนินงาน

สามารถสรุปผลงานแต่ละส่วนในโครงงานได้ดังนี้

5.1.1 ส่วนของการรันโปรแกรมโดยวิธี RMI

สรุปความสามารถได้ดังนี้

- สามารถทำการรันโปรแกรมผ่านเครื่องอื่นๆ ได้อย่างถูกต้อง และมีประสิทธิภาพ
- สามารถส่งข้อมูลของโปรแกรมและสื่อสาร ได้อย่างเหมาะสมและถูกต้อง
- สามารถนำข้อมูลที่ได้จากการสื่อสารมาจัดข้อมูลและใช้งานได้อย่างถูกต้อง

5.1.2 ส่วนของการค้นหาข้อมูลจากผู้ใช้

สรุปความสามารถได้ดังนี้

- สามารถรองรับการใช้งานจากผู้ใช้ได้หลาย ๆ คนในเวลาเดียวกันอย่างมีประสิทธิภาพ
- สามารถติดต่อกับฐานข้อมูลได้อย่างมีประสิทธิภาพโดยมีเครื่องเซิร์ฟเวอร์เป็นตัวกลาง
- สามารถรองรับรูปแบบโปรแกรมตรงกับความต้องการผู้ใช้ข้อมูล

5.1.3 ส่วนของรูปร่างเกมส์

ซึ่งในที่นี้ได้พัฒนาเกมขึ้น โดยสรุปความสามารถได้ดังนี้

- สามารถจัดการการเล่นเกมระหว่างผู้เล่นได้อย่างมีประสิทธิภาพในระดับหนึ่ง
- สามารถเล่นได้พร้อมกันได้ โดยมีเครื่องเซิร์ฟเวอร์คอยจัดการอย่างมีประสิทธิภาพ
- สามารถเล่นเกมส์ในเครื่องคอมพิวเตอร์ที่จัดสรรทรัพยากรอย่างเหมาะสม

5.2 แนวทางในการพัฒนาต่อ

โครงงานนี้สามารถที่จะพัฒนาขีดความสามารถได้เพิ่มเติมต่อไปอีกดังนี้

5.2.1 ส่วนของการรันโปรแกรมโดยวิธี RMI

- พัฒนาหลักการในการเทียบค่าและคำนวณค่าความสำคัญให้เหมาะสมยิ่งขึ้น
- พัฒนาในส่วนของการ update ข้อมูลในฐานข้อมูลผู้เล่นให้มีประสิทธิภาพเพิ่มขึ้น
- พัฒนาในรูปแบบการทำงานบนโปรแกรม Java ในแต่ละ version ได้ดียิ่งขึ้น

5.2.2 ส่วนของการค้นหาข้อมูลจากผู้ใช้

- พัฒนาการค้นหาข้อมูลจากเครื่องเซอร์เวอร์ให้มีประสิทธิภาพเพิ่มขึ้น
- พัฒนาการรองรับจากผู้ใช้ในเวลาเดียวกันได้อย่างมีประสิทธิภาพ
- พัฒนารูปแบบในการใช้โปรแกรมเพื่อให้ตรงกับความต้องการของผู้ใช้มากขึ้น

5.2.3 ส่วนของรูปร่างเกมส์

- พัฒนาประสิทธิภาพในการรองรับการเล่นพร้อมกันของผู้ใช้มากขึ้นได้
- พัฒนารูปแบบของการนำเสนอเกมส์ให้ตรงกับความต้องการของผู้ใช้มากขึ้น
- พัฒนาประสิทธิภาพของโปรแกรมให้เหมาะสมกับทรัพยากรในแต่ละเครื่องได้

ภาคผนวก ก

โค้ดของโปรแกรม Listings มีดังต่อไปนี้

Program 1### Score.java ###

```
import java.io.Serializable;

class Score implements Serializable
{
   String theHomeCity;
   String theHomeTeam;
   String theVisitorTeam;
   int    theHomeScore;
   int    theVisitorScore;
   int    theQuarter;

   public Score(String homecity, String hometeam,
    String visitorteam, int homescore, int visitorscore,
    int quarter)
   {
      theHomeCity    = homecity;
      theHomeTeam    = hometeam;
      theVisitorTeam  = visitorteam;
      theHomeScore    = homescore;
      theVisitorScore = visitorscore;
      theQuarter     = quarter;
   }

   public String getKey()
   {
      return theHomeCity + " vs " + theVisitorTeam;
   }
```

```
  public String toString()
  {
    String s = theHomeTeam;
    s += " vs " + theVisitorTeam;
    s += " at " + theHomeCity;
    s += ": "   + theHomeScore;
    s += "-"    + theVisitorScore;
    s += " ("   + theQuarter + "Q)";
    return s;
  }
}
```

Program 1### Scores.java ###

```
import java.io.Serializable;
import java.util.*;

public class Scores implements Serializable
{
  Hashtable theScores = new Hashtable();

  public String [] getGames()
  {
    int elemcount = theScores.size();
    String [] games = new String [elemcount];
    Enumeration elements = theScores.elements();
    for (int i = 0; i < elemcount; i++)
    {
      Score score = (Score) elements.nextElement();
      games [i] = score.getKey();
    }
    return games;
  }
```

```
    public void putScore(Score score)
    {
        String key = score.getKey();
        theScores.remove(key);
        theScores.put(key, score);
    }

    public Score getScore(String key)
    {
        return (Score) theScores.get(key);
    }
}
```

Program 1### ScoresServerInterface.java ###

```
import java.rmi.*;

public interface ScoresServerInterface extends Remote
{
    public String [] fetchGames() throws RemoteException;
    public Score fetchScore(String game) throws RemoteException;
}
```

Program 1### RMIServer.java ###

```
import java.rmi.*;
import java.rmi.server.*;

public class RMIServer extends UnicastRemoteObject
  implements ScoresServerInterface
{
    static final String SERVERNAME = "scores";

    Scores theScores = new Scores();
```

```
public static void main(String [] args)
{
   System.setSecurityManager(new RMISecurityManager());
   try
   {
      RMIServer server = new RMIServer();
      Naming.rebind(SERVERNAME, server);
      System.out.println("Server ready.");
   }
   catch (Exception ex)
   {
      ex.printStackTrace();
   }
}

public RMIServer() throws RemoteException
{
   theScores.putScore(new Score("Denver", "Broncos",
     "Dolphins", 6, 0, 1));
   theScores.putScore(new Score("Dallas", "Cowboys",
     "Packers", 6, 6, 2));
}

public String [] fetchGames() throws RemoteException
{
   return theScores.getGames();
}

public Score fetchScore(String game) throws RemoteException
{
   return theScores.getScore(game);
}
}
```

Program 1### RMIClient.java ###

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.rmi.*;
import java.rmi.server.*;

public class RMIClient extends Applet
{
   static final String URL = "rmi://161.246.6.64/scores";

   Button   theFetchGames  = new Button("Fetch Games");
   Button   theFetchScore  = new Button("Fetch Score");
   Choice   theGames       = new Choice();
   TextArea theScore       = new TextArea(20, 64);

   FetchScoreHandler theFetchScoreHandler =
    new FetchScoreHandler();

   public void init()
   {
      setLayout(new BorderLayout());
      Panel p = new Panel();
      add(theGames, BorderLayout.NORTH);
      add(theScore, BorderLayout.CENTER);
      add(p, BorderLayout.SOUTH);
      p.setLayout(new GridLayout(1, 0));
      p.add(theFetchGames);
      p.add(theFetchScore);
      theScore.setEditable(false);
      theScore.setFont(new Font(
       "Monospaced", Font.BOLD, 12));
      theGames.addItemListener(
```

```
    new GameHandler());
  theFetchGames.addActionListener(
    new FetchGamesHandler());
  theFetchScore.addActionListener(
    theFetchScoreHandler);
}

public void fatalError(Exception ex)
{
  ex.printStackTrace();
}

class GameHandler implements ItemListener
{
  public void itemStateChanged(ItemEvent evt)
  {
    String choice = theGames.getSelectedItem();
    theFetchScoreHandler.fetchScore(choice);
  }
}

class FetchGamesHandler implements ActionListener
{
  public void actionPerformed(ActionEvent evt)
  {
    try
    {
      theGames.removeAll();
      ScoresServerInterface server =
        (ScoresServerInterface) Naming.lookup(URL);
      String [] games = server.fetchGames();
      for (int i = 0; i < games.length; i++)
      {
```

```
            theGames.addItem(games[i]);
          }
        }
        catch (Exception ex) { fatalError(ex); }
      }
    }

    class FetchScoreHandler implements ActionListener
    {
      public void actionPerformed(ActionEvent evt)
      {
        fetchScore(theGames.getSelectedItem());
      }

      public void fetchScore(String game)
      {
        try
        {
          ScoresServerInterface server =
           (ScoresServerInterface) Naming.lookup(URL);
          Score score = server.fetchScore(game);
          theScore.setText(score.toString());
        }
        catch (Exception ex) { fatalError(ex); }
      }
    }
}
```

โค้ดโปรแกรมที่ 1### RMIClient.html ###

```
<APPLET
        CODE=RMIClient WIDTH=500 HEIGHT=300>
</APPLET>
```

โค้ดของโปรแกรม RMIDate มีดังนี้

Program 2### DateServer.java ###

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Date;

public interface DateServer extends Remote {
   public Date getDate () throws RemoteException;
}
```

Program 2### DateserverImpl.java ###

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Date;

public class DateserverImpl exteimport java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Date;

public interface DateServer extends Remote {
   public Date getDate () throws RemoteException;
}
nds UnicastRemoteObject implements DateServer {
 ]


 ]
   public DateserverImpl () throws RemoteException {
   }

   public Date getDate () {
      return new Date();
   }
```

```
    public static void main(String args[])throws Exception {
        DateserverImpl dateServer = new DateserverImpl();
        Naming.bind("Date Server",dateServer);
    }
}
```

Program 2### DateClient.java ###

```
import java.rmi.Naming;
import java.util.Date;

public class DateClient {
    public static void main (String args[]) throws Exception {
        if (args.length != 1)
            throw new IllegalArgumentException ("Syntax: DateClient <hostname>");
        DateServer dateServer = (DateServer) Naming.lookup("rmi://"+ args[0] +"/Date Server");
        Date when = dateServer.getDate();
        System.out.println(when);
    }
}
```

โปรแกรม RMIGame

Class : Arrow

Attribute: locx, locy , power, id

Operation: Arrow() , run()

Class: ArrowMan extends Arrow

Attribute:

Operation: ArrowMan()

Class: BrokerInterface extends Remote

Attribute:

Operation : RegisterServer() , RegisterClient

Class: ClientApplet extends Applet

Attribute: URL, BrokerURL, Servername, Map ,

Operation : ConnectServer(), InsertKeyInput() , DoAnyThing(), SendItem(), YouDead()

Class: DataPacket

Attribute: KeyIndex, KeyValue

Operation: DataPacket()

Class : Hero

Attribute : Life_Power, Hit_Power_Type1, Hit_Power_Type2, PositionX, PositionY

Operation : Action(),setPositionXY(), setLifePoint(),setHitPowerType1(),setHitPowerType2(),

getLifePoint(),getHitPowerType1(),getHitPowerType2(), getPositionX(), getPositionY();

Class : MyBroker extends UnicastRemoteObject

Attribute: brokername

Operation: Main(), RegisterServer(), RegisterClient();

Class : MyClientInterface extend remote

Attribute :

Operation: DoAnyThing(), SendItem(), YouDead()

Class: MyClientWrapper

Attribute : id

Operation: MyClientWrapper(), getHeroId(), getHeroCode(), setPos(), setLife(), sendItem(), kill(),

DoAnyThing()

Class: MyServer extends UnicastRemoteObject

Attribute : brokerURL, serverName , map

Operation: MyServer(), Main(), clearMap(), doClientAction(),fillClientPosToMap(),fillActionToMap

(),checkArrowHit(),checkSwordHit(),sendMapToAllClient(), checkDie(), gameRule(), getServerInfo(),

RegisterServer(); ClearAction()

Class: MyServerInterface extends remote

Attribute:

Operation: getServerInfo(), RegisterServer()

Class: Sword

Attribute: locX, locY

Operation: Sword()

Class: Swordman extends Hero

Attribute:

Operation: SwordMan()

โค้ดโปรแกรม DateRMI

โปรแกรม ### DateServer.java ###

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.Date;

public interface DateServer extends Remote {
   public Date getDate () throws RemoteException;
}
```

โปรแกรม ### DateserverImpl.java ###

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Date;
```

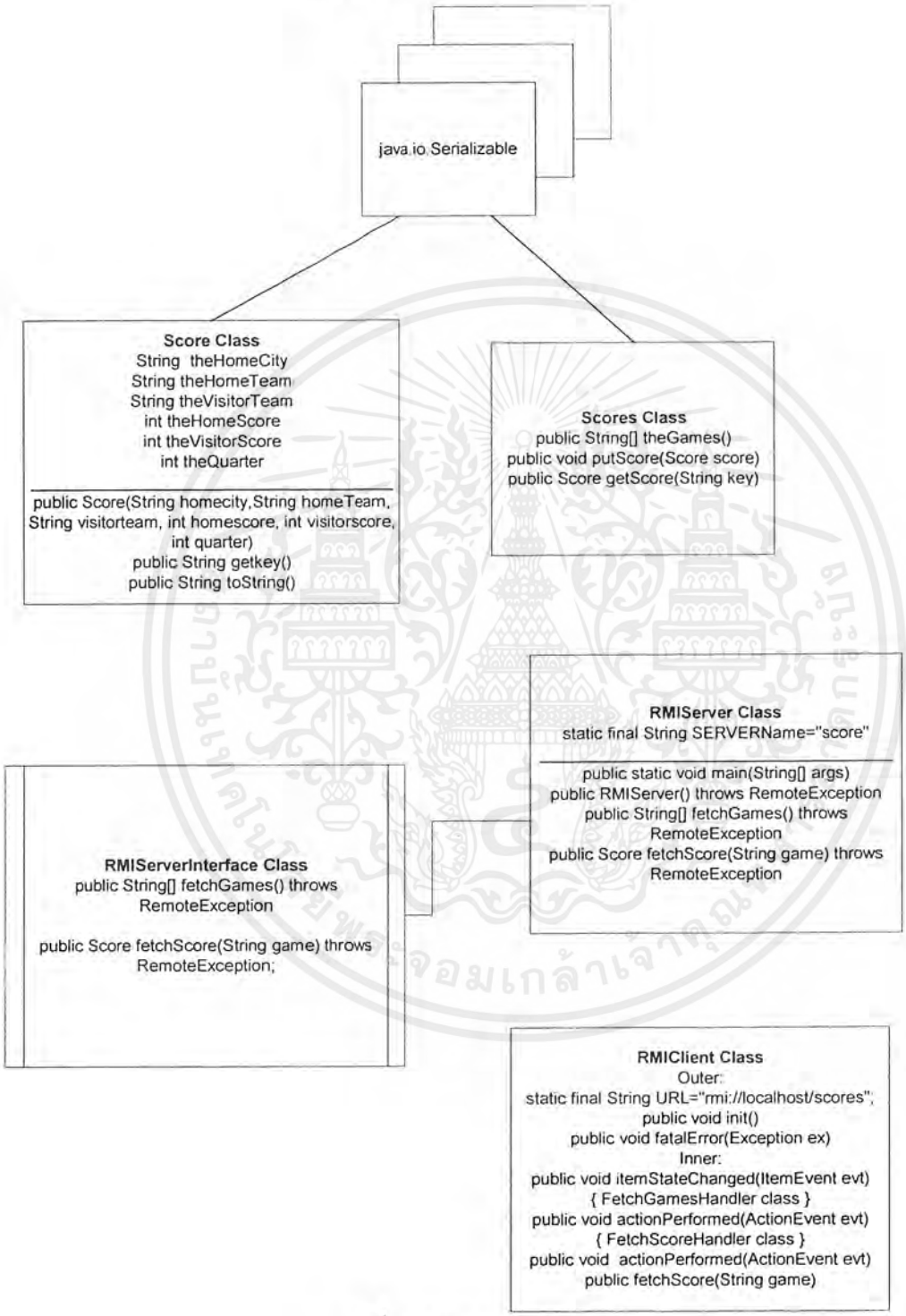
```
public class DateserverImpl extends UnicastRemoteObject implements DateServer {

   public DateserverImpl () throws RemoteException {
   }

   public Date getDate () {
      return new Date();
   }

   public static void main(String args[])throws Exception {
      DateserverImpl dateServer = new DateserverImpl();
      Naming.bind("Date Server",dateServer);
   }
}
```

โปรแกรม ### DateClient.java ###

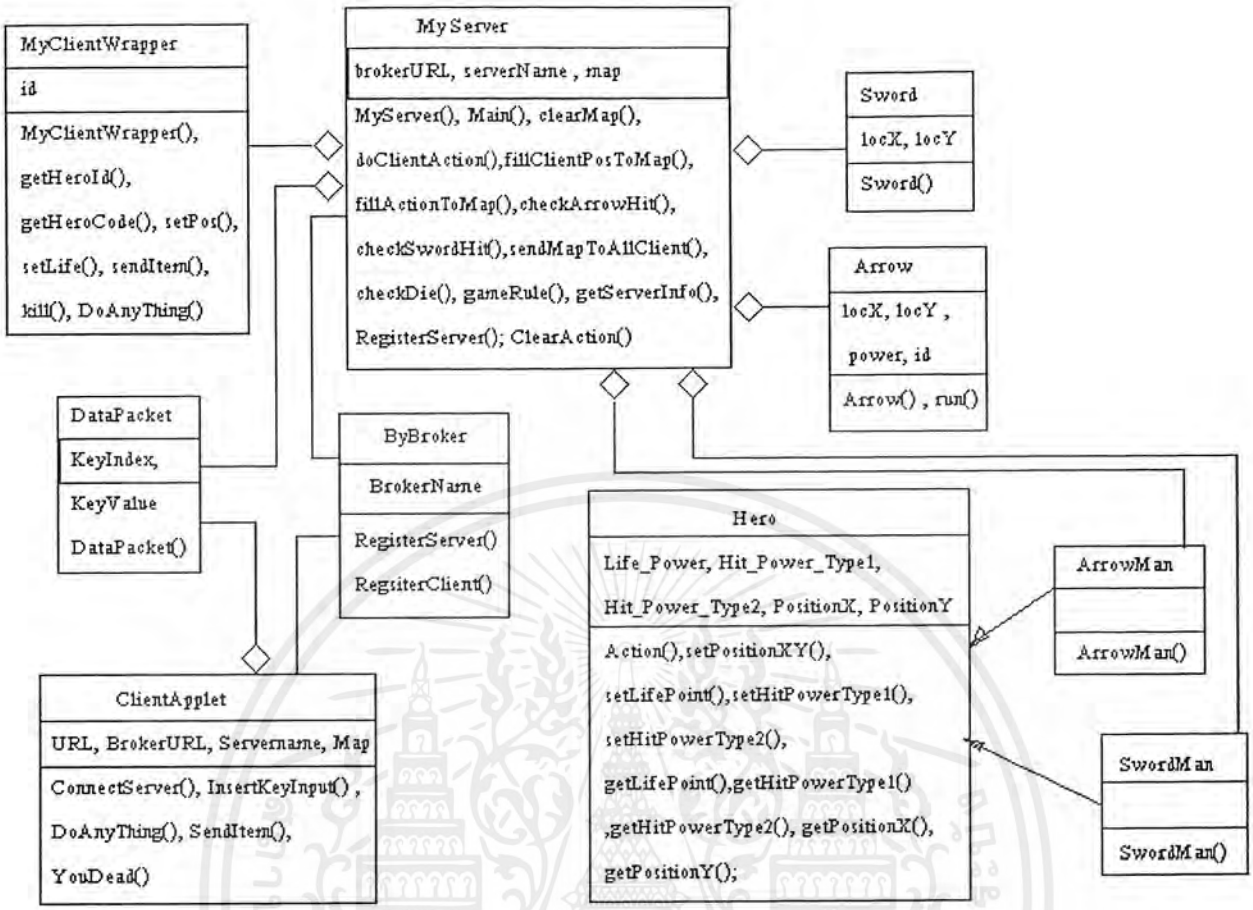
```
import java.rmi.Naming;
import java.util.Date;

public class DateClient {
   public static void main (String args[]) throws Exception {
      if (args.length != 1)
         throw new IllegalArgumentException ("Syntax: DateClient <hostname>");
      DateServer dateServer = (DateServer) Naming.lookup("rmi://"+ args[0] +"/Date Server");
      Date when = dateServer.getDate();
      System.out.println(when);
   }
}
```

ภาคผนวก ข

แผนภาพการทำงานต่าง ๆ



รูปที่ ข -1 แผนภาพ Listings



รูปที่ ข-2 แผนภาพ RmiGame

ภาคผนวก ค

คู่มือการใช้งานสำหรับผู้ใช้

ตัวอย่างโปรแกรมที่ 1

1. Compile all the .java files. You can do this in a single step:

   java *.java
2. Process the RMIServer.class file through the rmic tool:

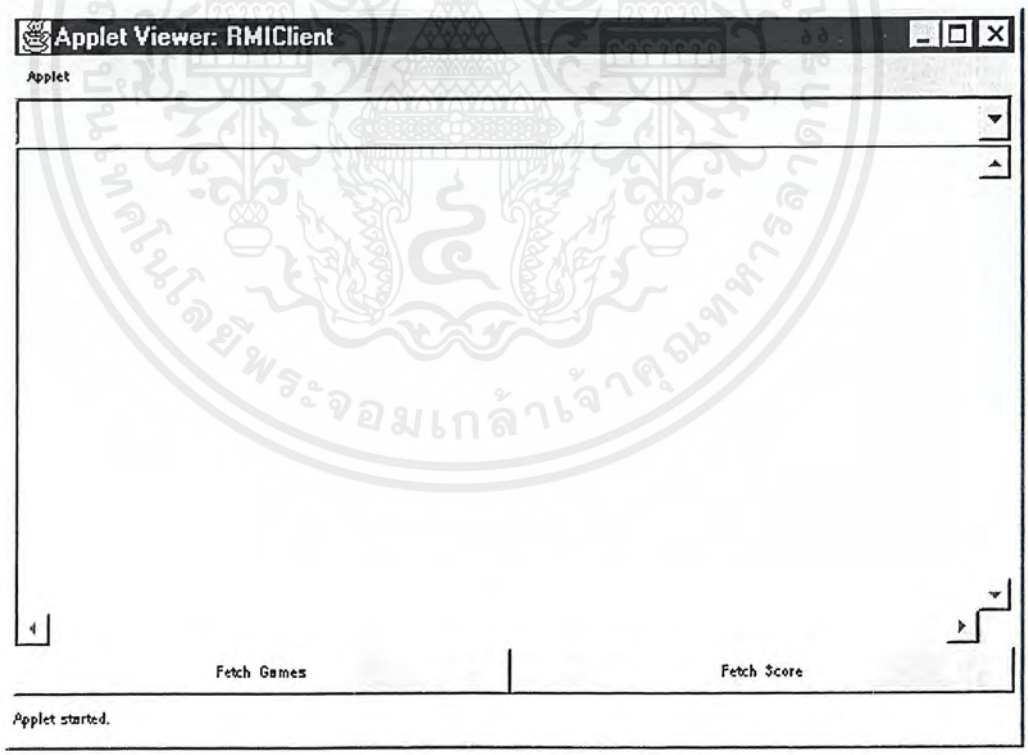
   rmic RMIServer
3. Start the registry:

   start rmiregistry
4. Start the server:

   java RMIServer
5. Open a second DOS window and start the client:

   appletviewer RMIClient.html

หน้าจอโปรแกรมตัวอย่าง



รูป ค-1 RMIClient

ตัวอย่างที่ 2

1. คอมไพล์โปรแกรมที่มีนามสกุล .java ทั้งหมด

   java *.java

2. สร้างสตับโดยคำสั่ง rmic

   rmic DateServerImpl

3. เรียก rmiregistry

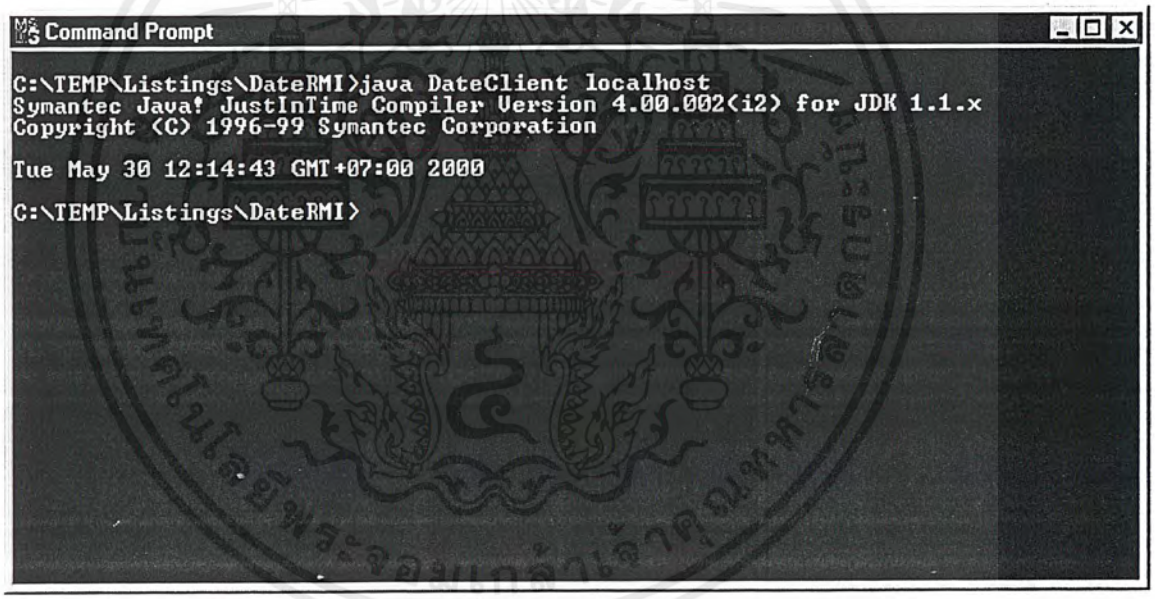
   start rmiregistry

4. เรียก DateserverImpl

   java DateserverImpl

5. เรียก DateClient ตามด้วยหมายเลข IP หรือ localhost

   java DateClient <host>



รูป ค-2 DateClient

ภาคผนวก ง
ภาษาจาวาเบื้องต้น

1. JAVA กับรูปแบบ เชิงวัตถุ

Java เป็นโปรแกรมภาษาโปรแกรมหนึ่งที่ทำงานในลักษณะของ **OOP** การเขียนโปรแกรมเชิงวัตถุ " *Object Oriented Programming* " เป็นวิธีการพัฒนางานและจัดระเบียบการเขียนโปรแกรมให้อยู่ในรูปแบบที่มององค์ประกอบต่างๆ ให้เป็นวัตถุ(Object)

1.1 Object

Object หมายถึง Entity หรือสิ่งที่มีตัวตน และนำไปใช้ในการประมวลผลข้อมูล โดยใน Object จะต้องมีข้อมูล (Data) ที่ใช้ในการอธิบายตนเองและการกระทำต่างๆ (Action) ที่ Object สามารถกระทำได้

1.2 Classes

Class นำไปใช้ในการสร้าง Object โดย Class จะประกอบไปด้วยลักษณะเฉพาะของ Data ใน Object พร้อมทั้งรายละเอียดของ Action ที่ Object นั้นๆ มีลักษณะเฉพาะของ Data คือ Attributes หรือ Variables

ชนิดของ Classes

ตามทฤษฎีของ **OO** จะแบ่ง Class ออกเป็น 3 กลุ่ม ได้แก่

1. **Entity Class**

   เป็น Class ที่ใช้แสดงข้อมูลของ Class Entity Class เป็นไปได้ทั้งข้อมูลที่จับต้องได้และไม่ได้ Entity Class แบ่งออกเป็น 3 กลุ่ม ได้แก่

   1.1 ***Concrete Classes*** เป็น Class ที่สามารถมองเห็นข้อมูลใน Class ได้

   1.2 ***Conceptual Classes*** ข้อมูลใน Class จะเป็นแนวความคิด (Concept) ของมนุษย์ รวมทั้งมโนภาพ (Abstraction)

   1.3 ***Event/State Classes*** เป็นเหตุการณ์ต่างๆ ที่เกิดขึ้นทั่วไป

   1.4 ***Interface Classes*** เป็น Class ที่ช่วยในการติดต่อระหว่าง User กับ Entity, Object

2. **Control Classes** เป็น Class ที่ช่วยในการควบคุม Object ต่างๆ ให้เป็นไปตามเงื่อนไขที่กำหนด

มีประโยชน์อยู่ 2 ประการที่สำคัญในการแบ่ง Class ของ OOP ได้แก่

1. การแบ่งออกเป็น *Class* ทำให้โปรแกรมที่สร้างขึ้นมามีความง่ายในการแก้ไข เนื่องจากแต่ละ Class จะไม่เกี่ยวข้องกัน (ถึงแม้จะทำงานรวมกัน) การแก้ไขสิ่งใดๆ ใน Class หนึ่ง ก็จะไม่มีผลกระทบต่อ Class อื่นๆ ที่เหลือ
2. เมื่อแต่ละ Class ไม่มีผลอย่างใดกับ Class อื่น เมื่อมีการแก้ไข ดังนั้น จึงสามารถนำ Class ต่างๆ มาใช้กับ โปรแกรมอื่นๆ ได้ตามใจชอบ แนวความคิดนี้เรียกว่า " *Reuseabillity* " หรือนำกลับมาใช้ใหม่นั่นเอง

1.3 Method

คำจำกัดความสำหรับ "Method" ก็คือ ระเบียบวิธีในการทำงานของงานใดงานหนึ่ง นอกจากนั้น ภายในแต่ละ Method ก็จะมีขั้นตอนการทำงานที่แตกต่างกัน และการนำขั้นตอนเหล่านี้หลายๆคำสั่งต่างๆ ของ Java มาประกอบกัน

1.4 Instantiation

เป็นการสร้าง Object ใดๆ ขึ้นมาใหม่ เพื่อให้สามารถใช้งาน Method ใดๆ ภายใน Class ได้ เมื่อสร้าง Object เสร็จแล้ว จะสามารถเรียกใช้ Method ต่างๆ ภายใน Class ได้ การเรียกใช้ Method จะใช้ในรูปแบบคำสั่ง

1.5 Data Encapsulation

สามารถที่จะซ่อนข้อมูลภายในแต่ละ Object ไว้ ไม่ให้ Object อื่นมองเห็นได้ ถ้ามีความต้องการใช้ข้อมูลเกิดขึ้น Object ที่ต้องการใช้ข้อมูลจะทำการร้องขอไปยัง Object ที่มีข้อมูล เพื่อให้ส่งข้อมูลกลับมาให้ตามที่ต้องการเท่านั้น โดยขบวนการต่างๆ จะถูกซ่อนไว้ทั้งหมด

1.6 Inheritance

OOP มีสัมพันธ์กับโครงสร้างแบบชั้น (Hierachy) หรือเรียกอีกอย่างหนึ่งว่า "การสืบทอดคุณสมบัติ" โดยสิ่งที่อยู่ด้านบนถ่ายทอดคุณสมบัติต่าง ๆ ลงมาให้กับสิ่งที่อยู่ด้านล่าง

2. พื้นฐานการเขียน Method

Method ของจาวา เปรียบเสมือน Procedure หรือ Function โดยเปรียบเทียบได้ว่าเป็น Method ที่ส่งผลลัพธ์กลับมาได้ เรียกว่า " *Function* " ถ้าเป็น Method ที่ไม่ส่งค่ากลับ (void) จะเรียกว่า " *Procedure* " ทั้งนี้โปรแกรมจาวาจะประกอบไปด้วย class ต่างๆ (ที่เรียกว่า " *public class* ") ภายในแต่ละ class จะมี Method อย่างน้อย 1 Method (ที่เรียกว่า " *main* ") เพื่อใช้ในการปฏิบัติงาน และจะต้องมี Method ชื่อ "main" เพื่อใช้ในการเริ่มต้นโปรแกรมเสมอ จากนั้นภายใน main Method ใดๆ นั้น

2.1 โครงสร้างของ Method

คำจำกัดความสำหรับ "Method" ก็คือ ระเบียบวิธีในการทำงานของงานใดงานหนึ่ง

2.2 Method Abstraction

เป็นแนวความคิดในการแบ่งโปรแกรมที่ต้องการพัฒนาออกเป็นส่วนๆ ตามหน้าที่การทำงาน โดยแต่ละส่วนที่แบ่งแยกกันออกไปนี้จะถูกเรียกใช้ซึ่งกันและกันได้ ลักษณะเช่นนี้เรียกว่า "Method Abstraction" โดยโปรแกรมที่แบ่งออกเป็นส่วนๆ นี้แต่ละส่วนจะเรียกว่า "Method"

2.3 ประเภทของ Method

จาวาแบ่งเป็น Method ออกเป็น 2 ประเภท ได้แก่

1. **Method ที่สร้างขึ้นมาเอง**

   ผู้เขียนเขียน Method ต่างๆใช้งานเองตามต้องการ Method ที่สร้างขึ้นมาเองนี้อาจเป็น Method ที่อยู่ใน Class เดียวกัน หรือต่าง Class ก็ได้

2. **Method ที่มีอยู่แล้ว**

   จะได้มาพร้อม Package ของ JDK โดย Methods เหล่านี้จะอยู่ใน Class Libraries ต่างๆ ที่ต้อง Import เข้ามาใช้ตอนต้นของโปรแกรม ด้วยคำสั่ง " *Import* " (เรียก Import เหล่านี้ว่า " *Packages* ")

2.4 การเรียกใช้ Method

2.4.1 Method ที่เราสร้างขึ้นเอง

1. **กรณีที่มีเพียง 1 Class ใน 1 โปรแกรม**

   ให้เรียกใช้จากชื่อของ Method พร้อมกับส่งค่าพารามิเตอร์ปให้กับ Method ตามที่ Method นั้นๆ ต้องการ

2. **กรณีที่ต้องการเรียกใช้ Method ที่อยู่ต่าง Class กัน**

โปรแกรมที่มี Class มากกว่า 1 Class หรือเรียกใช้ Method ในโปรแกรมอื่นๆ การเรียกใช้ Method ที่อยู่ต่าง Class กัน ต้องระบุชื่อ Class ไว้ร่วมกับ Method โดยคั่นระหว่าง Class และ Method ด้วยจุด (.)

2.4.2 Method จาก Class Libraries

เรียกใช้โดยระบุชื่อ Package ชื่อ class และชื่อ Method ตามลำดับ โดยคั่นเครื่องหมายจุด (.)

2.5 การแสดงผลทางจอภาพ (Console Output)

ใช้ Method ที่ชื่อว่า "*println*" ซึ่งอยู่ใน "*System.out*" คำสั่งนี้ต้องการข้อมูลที่เป็น String เพื่อนำไปแสดงบนจอภาพ แต่สามารถส่งข้อมูลที่เป็น Integer ร่วมด้วยโดยใช้เครื่องหมาย + มาเรียงเชื่อมต่อกัน โดยคำสั่งนี้ จะแปลงข้อมูลทั้งหมดให้เป็น String โดยอัตโนมัติ

2.6 การรับและส่งค่าระหว่าง Arguments และ Parameters

แต่ละ Method จะมีการรับส่งค่าให้กันในระหว่างการเรียกใช้โดย Method ที่เป็นตัวเรียกจะต้องส่งค่าไปให้กับ Method ที่ถูกเรียก เช่น Method ชื่อ "*Main*" นอกจากการส่ง Parameter เป็นค่าคงที่แล้ว ยังสามารถส่งค่า Parameter เป็นตัวแปรได้ แต่การส่งของ Method นั้นจะไม่สามารถส่งตัวแปรไป แต่จะส่งค่าที่อยู่ในตัวแปรไปแทน

2.7 การใช้คำสั่ง Return

คำสั่ง Return มีลักษณะดังต่อไปนี้

- ใช้จบการทำงานของ Method คือ เมื่อโปรแกรมพบ Return เมื่อใด จบการทำงานของ Method นั้น
- ถ้าหลัง Return มีตัวแปรใดตามมา ก่อนจบจะส่งค่าตัวแปรนั้นกลับไปให้กับ Method ที่เรียกใช้มา
- ถ้า Method นั้นมีการระบุ Keyword "*void*" ไว้ที่ Header ด้วย แสดงว่าไม่มีการส่งค่าใดๆ กลับ ดังนั้นก็ไม่จำเป็นต้องใส่ Return ไว้หลังสุดของ Method ก็ได้
- จะวาง Return ไว้ในส่วนใดของ Method ก็ได้ ถ้าต้องการให้จบการทำงานที่ส่วนนั้นมักพบว่าลักษณะนี้ ในคำสั่งควบคุมหรือคำสั่งเงื่อนไข เช่น IF, WHILE, FOR ... NEXT
- ถ้ามีการส่งค่ากลับ ชนิดของข้อมูล จะต้องเป็นชนิดเดียวกับที่ระบุไว้ใน data_type ของ Header ของ Method นั้น

2.8 การตรวจสอบข้อผิดพลาดของโปรแกรม

จาวามี Utility ในการตรวจสอบข้อผิดพลาดให้ด้วย ปัจจุบันมีอยู่หลาย Utilities ที่ใช้กับจาวาได้ ที่นิยมใช้กันคือ **JDB (Java Debuger)** และ **Jbuilder**

2.9 Packages

Packages เป็นที่เก็บรวบรวม "*Classes*" ต่างๆ เอาไว้มากมาย Classes เหล่านี้จะถูกสร้างเอาไว้เรียบร้อยแล้ว เพื่อเตรียมเรียกใช้งานได้ทันที ภายในแต่ละ Class ก็จะมี Methods มากมาย Packages ถูกเรียกใช้ได้ด้วยคำสั่ง import ซึ่งจะต้องวางไว้ตอนต้นสุดของโปรแกรมเสมอ

Packages ของ JDK1.1 บางครั้งจะเรียกว่า "*API*" (Application Programming Interface) ซึ่งก็หมายถึง class ต่างๆ ที่มีมาให้พร้อมกับ JDK โดยใน JDK1.1 จะมี Package ทั้งหมด 25 Packages ซึ่งมี Class รวมกันได้ 477 Classes

2.10 การเรียกใช้ Command – Line Arguments

บางครั้งอาจจำเป็นต้องส่งข้อมูลบางอย่างไปพร้อมๆ กับการเรียกใช้โปรแกรมจาก Command Line หรือ DOS (ด้วยคำสั่ง Java) วิธีการนี้จะทำได้โดยการส่งค่า "*Arguments*" ที่ต้องการไปยัง "*main*" method

ส่วน Comand – Line Argument คือ การส่งค่าของข้อมูลไปประมวลผลพร้อมๆ กันกับการเรียกใช้โปรแกรมจาวา โดยมีตัวแปรที่ใช้ในการอ้างอิงคือ **args**

2.11 การรับข้อมูลทางจอภาพ

เราสามารถรับข้อมูลที่ป้อนเข้ามาทางจอภาพโดยแป้นพิมพ์ (Keyboard) ได้หลายวิธี เช่น

- การใช้ System.in.Read
- การใช้ BufferedReader
- การใช้ AWT Package

ในการเขียนโปรแกรมเพื่อรับข้อมูลทางจอภาพนี้ จะต้องใช้คำสั่ง "*throws IOException*" และ Package ชื่อ "*java.io*"

2.12 Recursion

Recursion หรือการทำงานแบบเรียกซ้ำ การนิยามการทำงานของ Method หนึ่ง โดยใช้นิยามการทำงานของอีก Method หนึ่ง ก็คือ Recursion เป็นลักษณะของการสั่งให้โปรแกรม

ทำงานซ้ำๆ กัน โดยการเรียกตัวเองซ้ำ ซึ่งการทำงานครั้งที่สองจะขึ้นกับผลการทำงานของครั้งแรก และการทำงานครั้งที่สามก็จะขึ้นกับผลการทำงานครั้งที่สอง ซึ่งจะเป็นเช่นนี้เรื่อยๆ ไป

3. การเขียน Method แบบ oop

การเขียน Method ในรูปแบบของ Object-Oriented Programming (oop) ซึ่งเป็นหัวใจหลักของการเขียนโปรแกรม Java

3.1 ความหมายของ Object และ Classes

ในการเขียนโปรแกรมแบบ OOP นั้นเป็นการเขียนโปรแกรมโดยใช้ Object เป็นหลัก Object หนึ่งๆ ก็จะมีคุณสมบัติ (Property) ที่ใช้ในการนิยามความหมายของ Object ว่าประกอบไปด้วยอะไรบ้างเรียก Property ว่า Data และพฤติกรรม (Behavior) ที่ใช้ในการอธิบายว่า Object นั้นทำอะไรได้บ้าง ในภาษาจาวา จะเรียก Behavior ว่า Methods

3.2 <u>การสร้าง Object</u>

การสร้าง Object ทำได้ 2 วิธีคือ

วิธีที่ 1 มีขั้นตอนคือ

1. กำหนดว่า Object นี้ จะให้อยู่ใน Class ใดๆ

รูปแบบ Class name Objectname

2.สร้าง Object เพื่อจัดสรรหน่วยความจำให้ Object นี้

รูปแบบ ObjectName = new Class Name ( )

วิธีที่ 2 วิธีลัด โดยใช้คำสั่งเดียว

รูปแบบ ClassName ObjectName = new ClassName ()

เมื่อสร้าง Object เสร็จแล้ว เมื่อต้องการเรียกใช้ data หรือ method ของ Object ให้ใช้คำสั่งในรูปแบบดังนี้

Data

รูปแบบ ObjectName.data

รูปแบบ ObjectName.method

3.3 Overloading Method

การสร้าง Method ขึ้นมาใหม่ โดยใช้ชื่อ Method เดียวกัน แต่สร้างให้ Method มีชนิดของตัวแปรใน Parameter List ต่างกัน โดยจะสร้าง Method ใหม่ขึ้นมาเป็นจำนวนเท่ากับตัวแปรที่

เป็นไปได้ เมื่อมีการใช้ Method ชื่อนี้ โปรแกรมจะเรียก Method ที่มีชนิดตัวแปรตรงกับที่ต้องการโดยอัตโนมัติ

3.4 Constructor Method

เมื่อ Object ใดๆ ถูกสร้างขึ้นมาภายใต้ Class หนึ่ง โปรแกรมจะต้องไปเรียกใช้ Method มีชื่อเดียวกันกับชื่อ Class ทันที (ถ้ามี) การกำหนดการทำงานลักษณะเช่นนี้เรียกว่า " *Constructor* " และเรียก Method ที่มีชื่อเดียวกับ Class ว่า " *Constructor Method* "

ทั้งนี้มีข้อจำกัดคือ

- **Constructor Method** ต้องมีชื่อเดียวกับชื่อ Class
- **Header** ของ Constructor Method ต้องไม่มีค่า return_data_type หรือแม้กระทั่ง Keyword " **void** "

3.5 Data Encapsulation

Data Encapsulation เป็นคุณสมบัติอย่างหนึ่งของการเขียนโปรแกรมแบบ Object Oriented โดยการจะเป็นการซ่อนข้อมูลไว้เพื่อเรียกใช้งานเฉพาะ Object หนึ่งๆ เท่านั้น ซึ่งจะทำให้ข้อมูลของ Object นั้นไม่เกิดการเปลี่ยนแปลงและเป็นอิสระจากกัน คุณสมบัติเหล่านี้จะส่งผลให้โปรแกรมมีประสิทธิภาพมากยิ่งขึ้น และเพื่อรองรับการขยายตัวของโปรแกรมในอนาคตด้วย การนำเสนอ Data Encapsulation นี้นิยมใช้การสร้าง Get Method และ Set Method ขึ้นใน Class ที่ใช้งาน

3.6 ชนิดของ Accessibillity

ตัวแปร Method และ Class สามารถมีค่าของ Accessibillity ได้ ดังนี้คือ public, protected, private, static, void หรือไม่กำหนดค่าใดๆ ไว้ก็ได้

3.7 การส่งผ่าน Arguments

การส่งผ่าน Arguments จะมีอยู่ 2 ลักษณะ คือ การส่งโดยใช้ค่าผ่านของ Arguments และการส่งโดยการอ้างอิง

การส่งผ่านด้วยการอ้างอิง Arguments

เป็นการส่งค่าผ่านโดยใช้ Object เป็นการรับส่งข้อมูล โดยการใช้ pointer ชี้ไปยังตัวแปรที่อ้างอิงถึงกัน ถ้าค่าของ Object ต้นฉบับเปลี่ยนแปลงไป ค่าของ Object ที่อ้างอิงถึง ก็จะเปลี่ยนแปลงตามไปด้วยเช่นกัน การส่งค่าโดยใช้ Object นั้น ตัวแปรทุกตัวที่เป็น Object นั้น จะสามารถอ้างอิงถึงกันหมดได้

3.8 ตัวแปรแบบ Instance และตัวแปรแบบ Class

ตัวแปรแบบ " *Instance* " เป็นตัวแปรที่ใช้เฉพาะ Object ใด Object หนึ่งเท่านั้น ไม่สามารถนำมาใช้ร่วมกัน (Share) ระหว่าง Object ใน Class เดียวกันได้ ซึ่งตัวแปรนี้จะถูกเก็บอยู่ใน Memory ต่าง Address กัน

ดังนั้นการเรียกใช้ตัวแปรจาก Object ที่ต่างกัน ก็จะได้ค่าที่ต่างกัน เนื่องจากตัวแปรแต่ละตัวถูกเก็บไว้ในหน่วยความจำต่าง Address กัน ตัวแปรลักษณะนี้จะนำมาใช้ร่วมกันภายใน Class ไม่ได้ เรียกตัวแปรเหล่านี้ว่า " *Instance Variable* "

ตัวแปรอีกประเภทหนึ่ง จะสามารถนำไปใช้งานร่วมกันระหว่าง Class ได้ (โปรแกรมเดียวกัน) ไม่ว่าจะกล่าวอ้างถึงในส่วนใดของโปรแกรม (ต่าง Class ได้) ซึ่งนี้จะให้ค่าเดิมเสมอ เนื่องจากตัวแปรนี้จะถูกเก็บใน Memory ที่ Address หนึ่งเพียง Address เดียว จะสร้างขึ้นด้วยการใช้ Keyword " **static** "

3.9 Instance Method และ Class Method

- Instance Method

ก็คือ Method ทั่วไปที่สร้างอยู่ใน Class และถ้าต้องการเรียกใช้ Method ชนิดนี้ ก็จะเรียกใช้โดยการสร้าง Object ขึ้นมาใหม่ก่อน แล้วจึงเรียกใช้

- Class Method

เรียกอีกอย่างหนึ่งว่า " *Static Method* " เป็น Method ที่เรียกใช้ได้โดยไม่จำเป็นต้องสร้าง Object ขึ้นมารองรับ แต่สามารถเรียก Method โดยการระบุชื่อ Class ที่ Method นั้นๆ อยู่แทนการระบุชื่อ Object

4. โครงสร้างของ Method (Method Structure)

4.1 ลักษณะโดยทั่วไปของ Method

โดยทั่วไปรูปแบบของ Method จะเป็นดังนี้

accessibility return_data_type methodName (parameter_list)

{ // เนื้อหาของ method }

คำอธิบาย

บรรทัดแรกของ Method เรียกว่า " *Header* " ใช้แสดงชื่อของ Method รวมทั้งค่าต่าง ๆ ดังนี้

- accessibility

กำหนดชนิดของการเรียกใช้ สำหรับแต่ละ Method นี้ แบ่งออกเป็น private, public, protect, static และ void

- return_data_type

กำหนดชนิดของผลลัพธ์ที่ Method นั้นจะส่งค่ากลับใน 1 Method จะต้องส่งผลลัพธ์อย่างน้อยที่สุด 1 ค่า กลับไปยัง Method ที่เรียกใช้มา (ยกเว้นกำหนด Accessibility ไว้เป็น " *void* ")

- parameter_list

เป็นชื่อของตัวแปรที่ใช้รับข้อมูลที่ส่งมาจาก Method ที่เรียกใช้งาน Method นี้ ถ้าไม่มีการส่งค่ามาให้ ก็ไม่จำเป็นต้องมี parameter_list ก็ได้ กรณีที่มี parameter_list มากกว่า 1 ตัว ให้คั่นแต่ละตัวด้วยเครื่องหมาย comma(,)

4.2 Accessibility

ค่าขอบเขตของการใช้งาน ตัวแปร, Method หรือ Class นั้น เรียกว่า " *Accessibility* " แบ่งออกเป็น

static ใช้นิยามตัวแปรและ Method ที่ต้องการให้มีค่าคงที่ สามารถนำไปใช้ได้กับทุก ๆ ส่วนของ Class โดยค่านั้นจะไม่มีการเปลี่ยนแปลงไม่ว่ากรณีใด ๆ

public ใช้นิยามตัวแปร, Method และ Class ใด ๆ เพื่อให้สามารถนำไปใช้กับ Class หรือโปรแกรมอื่น ๆ ได้

private ใช้นิยามตัวแปรหรือ Method เพื่อให้เรียกใช้ได้เฉพาะภายใน Class ที่สร้างตัวแปร หรือ Method นั้น ๆ ขึ้นมาเท่านั้น

protected ใช้นิยามตัวแปรหรือ Method ที่ใช้ได้เฉพาะ Class ที่สร้างขึ้นมาด้วยวิธีการสืบทอด (Inheritance) เท่านั้น

void ใช้นิยาม Method โดยแบ่งเป็นการกำหนดให้ไม่มีการส่งค่าใด ๆ กลับมาให้กับ Method นี้ ถ้าไม่มีการระบุ accessibility ไว้ จะหมายถึง ตัวแปร ใช้ได้เฉพาะใน Method ที่นิยามตัวแปรนั้นไว้ Method ใช้ได้เฉพาะภายใน Class นั้นเท่านั้น Class อื่นจะมาเรียกใช้ไม่ได้

Class ใช้ได้เฉพาะภายในโปรแกรม (File) นี้เท่านั้น โปรแกรมอื่นจะมาเรียกใช้ Method หรือ ตัวแปรต่าง ๆ ใน Class นี้ไม่ได้เลย

4.3 Overloading

งานบางงานอาจต้องให้มีการแยกแยะด้วยว่า ส่งค่าตัวแปรชนิดใดมา เช่น ถ้าส่งค่าตัวแปรมาเป็น Integer ก็ให้ทำงานใน Method ที่ 1 ถ้าส่งมาเป็น String ก็ให้ทำงานใน Method ที่ 2 โดยทั้ง 2 Method จะทำงานเหมือนกัน เพียงแต่รับค่า Input มาต่างกัน ลักษณะเช่นนี้เรียกว่า " *Overloading* " ข้อจำกัดของ Overloading คือชื่อ Method ที่ใช้ (เช่น Method1, Method2) จะต้องเป็นชื่อเดียวกัน เพียงแต่ parameter_list ต่างกัน

```
public String find(Int ID-numb)
{      // เนื้อหาของ method      }

public String find (String name)
{      // เนื้อหาของ method      }
```

คำอธิบาย

ทั้ง 2 Methods มีชื่อเดียวกันคือ " *find* " และมีชนิดของ Return_data_type เป็น " *String* " เหมือนกัน ถ้า Method ที่เรียกใช้งาน ส่งค่า Parameter มาเป็นชนิด Integer(ตัวเลข) โปรแกรมจะเลือกทำงานตาม " Method บน" แต่ถ้าส่ง Parameter มาเป็นชนิด String(ข้อความ) โปรแกรมก็จะทำงานตาม "Method ล่าง" แต่ไม่ว่าจะทำงานตาม Method ใด ก็ตาม ผลลัพธ์ที่ส่งกลับไปให้ Method ที่เรียกใช้งาน จะต้องเป็น " **String** " เสมอ

4.4 Constructors

เมื่อ Object ใด ๆ ถูกสร้างขึ้นมา จะต้องไปเรียกใช้ Method ที่ระบุนี้ขึ้นมาใช้งานทันที การกำหนดการทำงานลักษณะเช่นนี้เรียกว่า " *Constructor* " ดังนั้น Constructor ก็คือ Method ทั่ว ๆ ไปของจาวา แต่มีข้อจำกัดดังนี้

1. ชื่อของ Constructor จะต้องเป็นชื่อเดียวกันกับชื่อ Class ที่ใช้สร้าง Object
2. Header ของ Constructor จะต้องไม่มีค่า Return_data_type

ดังนั้นถ้าชื่อ Method ใด ๆ ใน Class เป็นชื่อเดียวกันกับ Class ด้วย แสดงว่า Method นั้นเป็น " *Constructor* " Method โดยทั่วไปโปรแกรมใหญ่ ๆใน Class หนึ่ง ๆ มักจะมีการกำหนด Constructor ไว้มากกว่า 1 Constructor โดยจะขึ้นอยู่กับค่าของ Parameter ที่ส่งมา ดังนั้นจึงสามารถนำลักษณะที่เรียกว่า " *Overloading* " มาใช้งานร่วมกับ Constructors ได้

3. Public Class

Class ที่มีค่า Accessibility เป็น " *public* " จะมีได้เพียง Class เดียวใน 1 โปรแกรมเท่านั้น และชื่อ Class จะต้องเป็นชื่อเดียวกันกับชื่อโปรแกรมด้วย

4. main Method

main Method จะแตกต่างจาก Method อื่น ๆ ได้แก่

- มีเพียง 1 main Method ใน 1 โปรแกรมเท่านั้น
- จะต้องมีค่า Accessibility เป็น "static" เสมอ
- จะไม่มีการส่งค่ากลับคืน ดังนั้น จึงใช้ Accessibility " *Void* " ร่วมด้วยเสมอ
- Parameter_list จะต้องเป็นรูปแบบใดรูปแบบหนึ่งเสมอ คือ (String args[]) หรือ (String[] args)

4.5 การเรียกใช้ Class Libraries

ดังที่กล่าวมาในบทต้น ๆ ว่า Class Libraries ก็คือ โปรแกรมย่อย ๆ ที่เขียนขึ้นมาพร้อมให้ใช้ได้เลย เรียกโปรแกรมย่อย ๆ ว่า " *Package* " การเรียกใช้นั้น จะกำหนดคำสั่งไว้ด้านบนสุดของโปรแกรม โดยใช้คำสั่ง import แล้ว ตามด้วยชื่อ Package และ Class ซึ่งแยกกันด้วยเครื่องหมาย (.) เช่น

import java.lang

เป็นการเรียกใช้ Class ชื่อ lang ใน Package ชื่อ java

import java.;*

เรียกใช้ทุก Class ของ Package ชื่อ " **java** " เป็นการสะดวกกว่าการเรียกใช้ทีละ Class แต่จะสิ้นเปลืองหน่วยความจำไปส่วนหนึ่ง

4.6 การสร้าง Object

ตามที่กล่าวมาในบทแรก ๆ ว่า Class เปรียบเสมือนต้นแบบ (Template) สำหรับสร้าง Object ใด ๆ ขึ้นมา Object ใดถูกสร้างใน Class ใด ก็จะมีโครงสร้างเหมือนกับที่ระบุใน Class นั้น (ถ้ามีแต่ Class แต่ไม่มี Object ก็จะไม่มีการทำงานใด เกิดขึ้น เปรียบเสมือนกับห้องเรียนที่ไม่มีครูและนักเรียน ก็จะไม่มีการเรียนการสอนเกิดขึ้น ดังนั้นเมื่อสร้าง Class ขึ้นมา ก็จะต้องสร้าง Object ขึ้นมาด้วยเสมอ)

เมื่อมีการสร้าง Object ใด ๆ ขึ้นมา โปรแกรมจะทำงานตามขั้นตอนดังนี้

- จัดสรรที่ว่างใน Main Memory (หน่วยความจำหลัก) ให้กับตัวแปรต่าง ๆ ของ Object
- เชื่อม Object นั้นกับ Method ต่าง ๆ ใน Class
- ถ้า Class นั้นมีการใช้ " *Constructor* " ให้ทำงานตาม Constructor ก่อน

ส่งค่าของที่อยู่(Address) ของจุดเริ่มต้นของ Object ไปให้กับโปรแกรม เพื่อให้โปรแกรมทราบว่า Object นั้นเก็บอยู่ที่ใดใน Main Memory (เพื่อให้เรียกใช้ได้ถูก)

4.8 การเรียกใช้ Method

การเรียกใช้ Method ใด ๆ นั้นจะต้องอ้างอิงถึง Object ใน Method นั้น ๆ ด้วยเสมอ โดยการเรียกใช้ Method ใด ๆ มีรูปแบบดังนี้

Object_variable.Method_name(argument_list)

Object_variable คือ ชื่อ Object (หรือตัวแปร) ที่สร้างขึ้นมาด้วยคำสั่ง new

Method_name คือ ชื่อ Method ที่ Class นั้น ๆ มีอยู่

Argument_list คือ รายชื่อตัวแปรที่ใช้ส่งไปให้ Method ที่เรียกใช้ (Method_name)

4.9 การใส่หมายเหตุ (Comment)

เป็นส่วนหนึ่งของโปรแกรมที่ต้องการใช้อธิบายสิ่งใด ๆ ก็ตามให้โปรแกรม โดย Compiler จะข้ามการแปลข้อความของบรรทัดที่กำหนดเป็นหมายเหตุไว้ การสร้าง หมายเหตุทำได้ 2 รูปแบบ คือ

ใช้ /* และ */

เหมาะสำหรับข้อความที่ยาวมากกว่า 1 บรรทัด โดยโปรแกรมจะถือว่าข้อความที่ตามหลัง /* จะเป็นหมายเหตุ ไปจะกว่าจะพบเครื่องหมาย */ จึงจะแสดงว่าจบหมายเหตุแล้วใช้

// เหมาะสำหรับข้อความสั้น ๆ 1 บรรทัด โดยบรรทัดใดขึ้นต้นด้วย // บรรทัดนั้นจะถือเป็น หมายเหตุ และ Compiler จะข้ามบรรทัดนั้นไป

5. Packages และ Interfaces

Packages เป็นไฟล์เก็บรายชื่อของ Class ที่ต้องการนำมาใช้ในงานเดียวกันหรือมีวัตถุประสงค์ในการทำงานที่คล้ายกัน สามารถกำจัดปัญหาเกี่ยวกับชื่อ Class ที่ซ้ำกันได้

Interfaces ใช้ในการระบุกลุ่มของ Methods ที่ใช้กับ Class จำนวนตั้งแต่ 1 Class ขึ้นไป แต่มีรายละเอียดภายใน Method แตกต่างกันไปตามวัตถุประสงค์การใช้งาน

5.1 Packages

ข้อกำหนดของการตั้งชื่อ Class ในจาวาชื่อหนึ่งคือ ชื่อจะต้องไม่ซ้ำกันในแต่ละโปรแกรม ซึ่งถ้ามีการสร้างโปรแกรมมากขึ้น ชื่อ Class ก็อาจมีโอกาสซ้ำกันมากขึ้น เพื่อหลีกเลี่ยงปัญหานี้ จาวาจึงได้สร้างคำสั่ง Package ขึ้นมาเพื่อใช้เก็บรายชื่อ Class ต่างๆที่ใช้ทำงานประเภทเดียวกันไว้

ด้วยกัน เมื่อต้องการใช้งาน Class ใดๆก็สามารถเรียกใช้ Package ที่เก็บ Class นั้นขึ้นมาในโปรแกรม (ด้วยคำสั่ง Import) จากนั้นจึงเรียกใช้ Mathod ต่างๆใน Class นี้อีกที ด้วยวิธีการนี้สามารถตั้งชื่อ Class หรือ Method ซ้ำกันได้ ถ้าเก็บไว้ต่าง Package กัน และไม่เรียกใช้พร้อมกัน ถ้ามีรายการเรียกใช้พร้อมกันในโปรแกรมเดียวกัน เมื่อ Compile โปรแกรมจะไม่มีผลใดๆ แต่เมื่อเรียกใช้งานโปรแกรมนี้เมื่อใดจะเกิด Error ฟ้องมาทันที

5.2 การป้องกันใน Package

ในการเรียกใช้ Method ต่างๆของแต่ละ Class หรือแต่ละ Package นั้น จะมีการกำหนดค่า Accessibility ไว้ด้วย ทั้งนี้จะแยก Class ออกเป็น 5 กลุ่มได้แก่

1. class เดียวกัน
2. Subclass ใน Package เดียวกัน
2. ไม่เป็น Subclass แต่อยู่ใน Package เดียวกัน
3. Subclass ที่อยู่ต่าง Package กัน
4. Class ที่อยู่ต่าง package กันและไม่ได้เป็น Subclass

สำหรับค่า Accessibility แบ่งออกเป็น 4 ชนิดได้แก่

1. Public เรียกใช้ได้ไม่ว่าจะอยู่ class หรือ package ใดก็ตาม
2. Private เรียกใช้ได้เฉพาะภายใน class เท่านั้น class อื่นเรียกใช้ไม่ได้
3. Protect ใช้ได้เหมือน public รวมทั้ง class ที่เป็น subclass แต่กรณีที่อยู่ต่าง package กันจะใช้ไม่ได้
4. Default กรณีที่ไม่กำหนดค่า accessibility ไว้ จะหมายถึงเป็นค่า default โดยจะใช้ได้เฉพาะ class และ subclass ที่อยู่ใน Package เดียวกัน (ต่าง package จะใช้ไม่ได้)

5.3 Interfaces

การนิยาม Interfaceนั้น จะเหมือนกับการนิยาม class เพียงแต่ภายใน Interfaces จะไม่มีชื่อตัวแปร จะมีเฉพาะรายชื่อ Methods แต่ไม่มีเนื้อหา(Body)ของ methods ทั้งนี้เมื่อโปรแกรมใดก็ตามมีการนิยาม "Interfaces " ไว้ด้วย จะหมายถึงกำหนดว่าทุก class ในโปรแกรมนั้นจะต้องสร้าง Method ที่มีชื่อตามที่ระบุไว้ใน Interfaces ด้วย โดยเนื้อหาของ Method อาจแตกต่างกันไปในแต่ละ class ขึ้นอยู่กับความต้องการในการใช้งาน(แต่ชื่อ Method ต้องเป็นชื่อเดียวกัน) ลักษณะนี้เรียกว่า "มีเพียง 1 ชื่อ แต่ทำงานได้หลายอย่าง (One tnterface ,Multiple Methods)"ในทางobject-Oriented programming จะเรียกลักษณะนี้ว่า " *Polymorphism* "

ประโยชน์ของการนำ Interface มาใช้นั้น ก็เพื่อหลีกเลี่ยงปัญหาของการ Inherite ระหว่าง classes สรุป Interface ใช้ในการระบุกลุ่มของ Methods ที่ใช้ได้กับ class จำนวนตั้งแต่ 1 class ขึ้นไป

5.3.1 การนิยาม Interface

โดยการระบุคำว่า " **Interface** " (แทนการระบุคำว่า " **class** ") เช่น กำหนด interface ชื่อว่า " **callback** " และกำหนดว่าในทุกๆ class จะต้องสร้าง Method ชื่อ " callback "เอาไว้ด้วย (แต่รายละเอียดของ Method อาจแตกต่างกันไปได้)

5.3.2 การสร้าง Method ที่ระบุใน Interface

โดยการระบุคำว่า " **Implements** " แล้วตามด้วยชื่อ Interface ไว้ในบรรทัดของการกำหนดชื่อ class จากนั้นจึงกำหนดชื่อ Method พร้อมรายละเอียดไว้ใน class นั้น

6. การสร้าง Applets

Applets จะนำมาใช้ร่วมกับการสร้าง Web Page แต่ Applet ก็มีขีดความสามารถและข้อกำหนดต่างๆ ที่จำกัดกว่า Application รวมทั้งทางด้านการป้องกันด้านความปลอดภัยของข้อมูลและปัญหาของไวรัสก็ยังไม่สามารถทำได้

6.1 โครงสร้างของ Applet

โปรแกรม Java หรือที่เรียกว่า " *Java Application* " นั้นจะเริ่มต้นทำงานโดยการเรียกใช้ Method ชื่อ " ***main*** " โดยอัตโนมัติ ดังนั้นทุกโปรแกรม Java จึงจำเป็นต้องมี main method เสมอ แต่ Java Applet นั้นไม่ได้มีลักษณะเช่นนี้ แต่จะมีโครงสร้างเป็นของตนเอง

Applet ไม่มี Main Method เหมือนใน Application แต่จะทำงานโดยการเรียกใช้ Method ชื่อ " **int** " ทั้งนี้โครงสร้างของ Applet จะต้องประกอบไปด้วย Method ดังนี้คือ int (), start (), stop () และ destroy () ซึ่งปกติ Method เหล่านี้จะมีแต่ชื่อ ไม่มีการระบุขั้นตอนการทำงานไว้ด้วย เพียงแต่เป็นชื่อ Method Applet ใช้เรียกปฏิบัติงานตามลำดับ ทั้งนี้จะต้องเขียนหรือสร้างรายละเอียดของแต่ละ Method ให้เหมาะสมกับงานเอง แต่อาจไม่ใช้โครงสร้างดังกล่าวนี้ต่อไปได้ แต่จะต้องกำหนดรายละเอียดของ Method และเรียกใช้งานให้ถูกต้องด้วยตนเอง

int ()

เป็น method แรกที่ถูกเรียกใช้งานทันทีเมื่อ load applet เข้ามาใช้งาน โดย Web Browser

start ()

จะถูกเรียกใช้ได้งานก็ต่อเมื่อ init method ถูกเรียกใช้งานแล้วเท่านั้น โดยเมื่อเข้าสู่การทำงานของ Web Page ใด start method ก็จะถูกเรียกใช้งานด้วย และเมื่อไปที่ Web Page หน้าอื่นๆ และมีการกลับมายังหน้าเดิม start method ก็จะถูกกเรียกใช้งานอีกครั้งหนึ่ง

stop ()

เรียกใช้งานเมื่อมีการย้าย Web Page ไปแสดงที่หน้าอื่นๆ

destroy ()

เรียกใช้เมื่อต้องการออกจาก Browser โดยจะเป็นการบอก Browser ด้วยว่าให้ปล่อย Resource ต่างๆ (เช่น Memory) ที่ Applet ยึดครองอยู่ออกไป ทั้งนี้ stop method จะต้องถูกเรียกใช้งานก่อน destroy method เสมอ

paint ()

เป็น method ที่ใช้กับการทำงานแบบกราฟฟิกของ Applet จะเรียกใช้เมื่อต้องการแสดงผลลัพธ์บนจอภาพแบบกราฟฟิก

6.2 ข้อจำกัดของ Applet

โดยทั่วไป Applet ถูกโหลดจากระบบเครือข่าย ซึ่งไม่อนุญาตให้มีการเข้าถึงเครื่องคอมพิวเตอร์ Local ดังนั้น จึงเกิดข้อจำกัดบางประการ ดังนี้

- Applet ไม่สามารถเรียกใช้ไฟล์และโปรแกรมต่างๆ จากเครื่องที่เป็น Client ได้
- Applet ไม่สามารถสร้างไฟล์บนเครื่องที่เป็น Client ได้
- Applet ไม่สามารถทำการติดต่อเข้ากับเครือข่ายได้ ยกเว้นย้อนกลับไปยังเครื่อง Host ที่ Applet เข้ามา
- Applet ไม่สามารถเรียกใช้โปรแกรมต่างๆ บนเครื่องที่เป็น Client ได้
- Applet ไม่สามารถโหลดไลบารี่ได้
- Applet ไม่สามารถหยุดการปฏิบัติงานของ Interpreter ได้ (เรียก System.exit () ไม่ได้)

ข้อจำกัดดังกล่าวเป็นสิ่งจำเป็นเพื่อป้องกันอันตรายให้กับ Applet จากการกระทำบางอย่างข้ามเครือข่ายมา คือ

1. การอ่านและการดาวน์โหลดไฟล์ส่วนตัวจากเครื่องคอมพิวเตอร์
2. การเขียนไวรัสคอมพิวเตอร์ หรือการทำลายข้อมูลบนเครื่องคอมพิวเตอร์

3. การรันโปรแกรมเพื่อทำลายโปรแกรมอื่นจากที่ใดที่หนึ่งบนเครือข่าย

4. การติดต่อโดยตรงกับเครื่อง Client และการเลือกทำการเปลี่ยนแปลงบนเครื่อง Client

เมื่อ Applet ทำการป้องกันการกระทำดังกล่าว ถ้าเกิดต้องการปฏิบัติในสิ่งที่ Applet ป้องกัน ให้ทำโดยการเขียน Java Application แทนซึ่ง Java Application มีความสามารถติดต่อกับระบบเครือข่ายเรียกใช้ไฟล์ข้อมูลรันโปรแกรม เรียก Native Method โหลดไลบารี่ และเรียก System.exit ()ได้

6.3 การนำ Applet ใส่ในเอกสาร HTML

เมื่อต้องการใส่ Applet ลงในเอกสารเพื่อประกอบกับข้อมูลอื่น ต้องนำ Applet ไปไว้ในไฟล์ HTML ซึ่งทำให้สามารถเปิดดูได้ด้วย Web Browser หรือรันโปรแกรม Appletviewer

6.3.1 คำสั่ง <APPLET>

ใช้รูปแบบ ดังนี้

```
<APPLET [Codebase=[path|URL]]code=someclass.class
height=applet_height
width=applet_width>
<APPLET>
```

พารามิเตอร์ Codebase

เป็นพารามิเตอร์ที่ใช้ระบุตำแหน่งที่อยู่ของไฟล์คลาส ค่าของ Codebase เป็นไปได้ทั้งชื่อ path ที่สัมพันธ์กับไฟล์ HTML หรือเป็น URL (Uniform Resource Location) ที่ระบุที่อยู่ของคลาสพารามิเตอร์ Codebase

การใช้งานพารามิเตอร์ Codebase ทำได้ 2 ลักษณะ คือ

1. การใช้กับ Path บน Local Host

Codebase เมื่อจะใช้ลักษณะนี้ได้ หมายความว่าไฟล์คลาสต้องอยู่ในไดเรกทอรี่ย่อย ซึ่งอยู่ในไดเรกทอรี่เดียวกันกับไฟล์ HTML

2. การใช้แบบกำหนด URL ของคลาสบน Remote Host

การใช้งานลักษณะนี้เป็นการเข้าถึงคลาสบนเครื่องควบคุม โดยใช้พารามิเตอร์ Codebase ระบุ URL ของไดเรกทอรี่ ซึ่งมีคลาสที่ต้องการอยู่

6.3.2 การส่งผ่านพารามิเตอร์ที่ผู้ใช้กำหนดขึ้นด้วยคำสั่ง <PARAM>

Applet รับค่าพารามิเตอร์สำหรับเริ่มต้นปฏิบัติงานได้จากการร้องขอไฟล์ HTML คำสั่ง <PARAM> ใช้สำหรับการกำหนดพารามิเตอร์เพิ่มเติมแล้วผ่านค่าให้กับ Applet นอกเหนือจากกำหนดด้วยคำสั่ง <APPLET>

ไวยกรณ์คำสั่ง

<PARAM>

```
<APPLET [Codebase=[path|URL]]code=someclass.class
height=applet_height
width=applet_width>
{[<PARAM name=parameter_name value=parameter_value>]}
<APPLET>
```

ภายในคำสั่ง <PARAM> ส่วนที่เป็น Case Sensitive จะมีเพียง Parameter_Value เท่านั้น (ตัวอักษร UpperCase และ LowerCase ถือว่าแตกต่างกัน) ส่วนอื่นๆ ไม่เป็น Case Sensitive ดังนั้นจะใช้เป็นตัวอักษรแบบใดก็ได้

คำสั่ง <PARAM> จำเป็นต้องอยู่ภายในระหว่างคำสั่ง <APPLET> และ </APPLET> ซึ่งใช้คำสั่งจำนวนมากน้อยเท่าใดก็ได้ตามแต่ผู้ใช้ต้องการในการกำหนดพารามิเตอร์ให้กับ Applet สนองตอบ

สำหรับค่า parameter ที่ใช้ได้นี้คือ <PARAM> ได้แก่

- name
- text

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ไม่อาจสำเร็จไปด้วยดีได้ หากไม่ได้รับความช่วยเหลือและร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นบุคคลสำคัญที่ทำให้ปริญญานิพนธ์นี้สำเร็จได้ด้วยดีคือ อาจารย์ บรรจง ปียธำรง อาจารย์ที่ปรึกษาที่ให้ความกรุณาเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

ขอขอบคุณคณะอาจารย์ทุกท่านที่ได้ให้การอบรมสั่งสอนและให้ความรู้ความสามารถนำความรู้ที่ได้รับมาทั้งหมดมาใช้ได้

ขอบขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์ที่ได้ติดต่อประสานงานอาจารย์ทุกท่าน และอำนวยสถานที่ในการทำโครงงานนี้

ขอขอบคุณพี่ ๆ ที่มีรายงานดี ๆ ให้ศึกษา

ขอขอบคุณเพื่อน ๆ ชาว 4D ทุกคนที่ช่วยเป็นกำลังใจและแนะนำความรู้ในการทำงาน

ขอบคุณเจ้าหน้าที่ประจำห้องภาคที่อนุญาตให้ใช้เครื่องในการทำโปรเจคที่ห้องภาค

ขอบคุณอ๊อบ ในฐานะที่ให้คำปรึกษาที่ดี

ขอบคุณเพื่อน ๆ ทุกคนที่มีส่วนร่วมช่วยกันทำรายงานฉบับนี้

และต้องขอขอบพระคุณบุคคลที่สำคัญที่สุดในชีวิตนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้จัดทำ พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และคอยให้กำลังใจเอาใจใส่เสมอมา ในทุก ๆ ด้านอันหาที่เปรียบมิได้ ผู้จัดทำขอกราบขอบพระคุณมา ณ ที่นี้

รณชาติ ผ่องบุรุษ

รีโอ จับบาง

บรรณานุกรม

1. Bill McCarty : " Java Distributed Objects ", SAMS, First edition, 1989
2. Y. Daniel Liang : " An Introduction to Java Programming ", Que E&T , First edition, 1998
3. Gray Cornell, Cay s.Horstmann,Sun Microsystems : " Core Java ",Sun Microsystems, Second editon, 1997
4. Mertin Hughes, Michale Shoffner, Derek Hamner,Derek : " Java Network Programming " , Manning Publications, Second edition, 1997
5. Janes Gosling, Frank Yellin : "Java application programming interface ", Addison-wesley, First edition, 1996
6. www.java.sun.com