Objective-C และ GNUstep

Objective-C และ GNUstep

by a suchness of philosophy

24 กันยายน พ.ศ. 2546

Copyright (c) 2002 Ching San Free Software School. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections. A copy of the license is included in the section entitled "GNU Free Documentation License".

สารบัญ

บทนำ

หลังจากที่ผมเขียนบทความเรื่อง Objective-C ครั้งแรกบน Project-ILE [1] ผมก็มานั่งนึกนอนนึกว่า จริงๆแล้วผมควรจะเขียนหนังสือขึ้นมาสักเล่ม ที่จะใช้แสดงความรู้สึกของผมเกี่ยวกับแนวคิดแบบซอฟท์แวร์เสรี[2] ความรู้สึกที่ว่านี้ คืออยากจะเห็นตำราสักเล่มที่ไม่ละเลยความสำคัญของแนวคิดดังกล่าว ซึ่งโดยส่วนตัวแล้วผมเชื่อว่าจะนำประโยชน์มาสู่มนุษยชาติได้มากกว่าน้อย ประกอบกับความปรารถนาที่จะเห็นคนหันมาสนใจภาษา Objective-C และใช้พัฒนาโปรแกรมอย่างกว้างขวาง ในทีแรกผมคิดจะเขียนหนังสือเล่มนี้เป็นภาษาฝรั่ง แต่กระนั้นผมก็คิดว่าตำราฝรั่งก็มีอยู่มากมายและดีอยู่แล้ว จึงตกลงใจว่าจะเขียนด้วยภาษาไทย หรือที่จริงอาจจะเป็นข้อแก้ตัวที่จากเหตุผลที่ผมไม่ค่อยจะเอาอ่าวกับภาษาฝรั่งก็ว่าได้ ด้วยเหตุและปัจจัยที่ได้เกริ่นไปในตอนต้นแล้ว แนวทางของหนังสือเล่มนี้จึงพยายามผสมผสานแนวคิดในเชิงการเมืองเข้ากับแนวคิดเรื่องเทคโนโลยีให้เป็นเนื้อเดียวกันให้มากที่สุดเท่าที่จะเป็นไปได้ โดยในแนวทางหนึ่งจะเน้นนำเสนอแต่เฉพาะเนื้อหาที่เป็นซอฟท์แวร์เสรีเท่านั้น หรือจะพยายามให้พาดพิงถึงซอฟท์แวร์แบบมีเจ้าของให้น้อยที่สุดเท่าที่จะเป็นไปได้ เพื่อให้ผู้อ่านตระหนักถึงความสำคัญของสังคมที่ควรจะใช้ซอฟท์แวร์เสรีเป็นทรัพยากรและเป็นค่านิยมพื้นฐาน และอีกแนวทางหนึ่ง คือการเสนอเนื้อหาของปรัชญาซอฟท์แวร์เสรีโดยตรง ไม่ว่าจะในส่วนของบทนำนี้หรือในเนื้อหาตอนท้ายของหนังสือในเรื่องบทบาทของซอฟท์แวร์เสรีและ GNU project[3]

มีเรื่องที่ผมอยากเรียนให้ผู้อ่านทราบเพิ่มเติมก่อนจะอ่านหนังอยู่ประการหนึ่ง ว่าด้วยเหตุที่หนังสือเล่มนี้เป็นหนังสือเล่มแรกที่ผมเขียนจนจบ และผมก็ไม่ได้มีพื้นเพมาจากนักเขียนแต่ประการใด จึงขอให้ผู้อ่านทำใจเอาไว้ล่วงหน้าด้วยว่าการนำเสนออาจจะน่าเบื่อหรือไม่ชวนติดตามไปบ้าง ซึ่งก็ต้องขออภัยในความบกพร่องดังกล่าวมา ณ ที่นี้ด้วย และสุดท้ายนี้ผมหวังว่าผู้อ่านจะได้รับความรู้จากหนังสือเล่มนี้และสามารถไปใช้ประโยชน์ได้อย่างจริงจัง หรือถ้าผู้อ่านมีข้อเสนอใดอันจะเกิดประโยชน์ต่อส่วนรวม ท่านสามารถเขียนมาติชมได้ตาม email address *id@project-ile.net* เพื่อที่ผมจะได้นำข้อเสนอของท่านมาปรับปรุงแก้ไขเพิ่มเติมใน edition ต่อไปของหนังสือเล่มนี้

ขอแสดงความนับถือ

a suchness of philosophy

---

[1] http://www.project-ile.net
[2] http://www.fsf.org
[3] http://www.gnu.org

บทที่ 1

ปรัชญาเบื้องต้นของการเขียนโปรแกรมแบบ Object-Oriented

1.1 การทำงานของโปรแกรมแบบ Object-Oriented

Object-Oriented หมายถึงการออกแบบโปรแกรมให้แบ่งเป็นหน่วยย่อยๆ โดยแต่ละหน่วยจะมีลักษณะของตัวเอง เราอาจจะเปรียบเทียบกับชีวิตประจำวัน เช่นที่ทำงานของเราซึ่งประกอบด้วยบุคคลต่างๆ แต่ละคนทำหน้าที่แตกต่างกันไป แต่ละคนมีบทบาทและความรับผิดชอบต่องานของตัวเอง มีคนทำความสะอาด มีคนพิมพ์เอกสาร แต่ละคนหรือแต่ละหน่วยก็จะมีข้อมูลเฉพาะของตัวเอง บางข้อมูลก็เปิดเผยให้หน่วยงานอื่นทราบได้ และต้องมีหน่วยที่คอยจัดการหรือบริหารหน่วยย่อยต่างๆเหล่านี้ให้สอดคล้องกัน และก้าวก่ายกันน้อยที่สุด เพื่อให้บรรลุวัตถุประสงค์ร่วมกัน

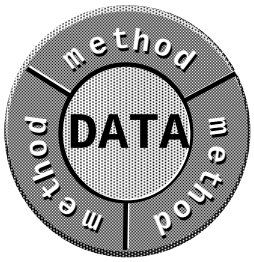
1.2 ลักษณะเฉพาะของ Object

การเขียนโปรแกรมแบบ Object-Oriented จะช่วยให้คุณสามารถจัดการกับการเขียนโปรแกรมขนาดใหญ่ได้สะดวกยิ่งขึ้น และสามารถกระจายงานออกไปให้หลายๆหน่วยทำงานร่วมกันโดยจะช่วยลดความซ้ำซ้อนของงานของคุณและทำให้งานเสร็จรวดเร็วยิ่งขึ้นและลดโอกาสที่จะเกิดข้อผิดพลาดลง หลักการนี้จะกระจายงานของคุณออกเป็นหน่วยย่อยๆที่เรียกว่า object แต่ละหน่วยย่อยจะติดต่อสื่อสารกันด้วยการส่ง message ถึงกัน และแต่ละ object ก็จะมีวิธีที่ตอบสนองต่อ message นั้นๆต่างกันไป ลักษณะที่สำคัญของ object ตามปรัชญาแบบ Object-Oriented จะประกอบด้วยสิ่งต่อไปนี้

1.2.1 Object

object ใดๆก็คือกลุ่มของข้อมูลและชุดคำสั่งที่เกาะกลุ่มกัน ยกตัวอย่างจากในชีวิตประจำวันเช่น เราอาจสมมติว่าสุนัขตัวหนึ่งเป็น object การกระทำใดๆของสุนัข หรือการกระทำใดๆของเราต่อสุนัขตัวนั้นถือ

เป็นชุดคำสั่ง เช่นถ้าเราเรียกชื่อสุนัขตัวนั้น ก็หมายถึงเราส่ง message บอกสุนัขตัวนั้น เมื่อสุนัขได้รับ message จากเรา มันก็อาจจะสนอง message นั้น ในวิธีที่ต่างกัน เช่นถ้ามันอาจจะวิ่งมาหาเรา หรือถ้ามันขี้เกียจ มันก็อาจจะทำเป็นไม่สนใจ ในที่นี้ภาวะทางจิตของสุนัขก็ถือเป็นข้อมูลเฉพาะของสุนัขตัวนั้น เราไม่สามารถรู้ถึงความคิดของสุนัขตัวนั้นได้ สิ่งที่สำคัญสำหรับ object จึงแบ่งออกเป็นสองส่วนหลัก คือข้อมูล และชุดคำสั่งที่จะใช้ตอบสนอง message ซึ่ง message ไม่จำเป็นจะต้องถูกส่งมาจาก object อื่นๆเท่านั้น object สามารถส่ง message ให้กับตัวเองก็ได้ ดังเช่นรูป 1.1 จะแสดงถึงข้อมูล(data)และชุดคำสั่ง(method)



รูปที่ 1.1: Object ประกอบด้วยข้อมูล(data)และชุดคำสั่ง(method)

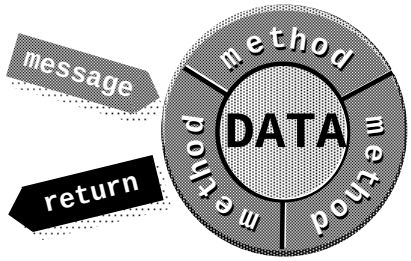
1.2.2 Class

ในแต่ละ object ย่อมมีลักษณะจำเพาะของตัวเอง บาง object อาจจะคล้ายคลึงกัน หรืออาจจะแตกต่างกันก็ได้ ลองเปรียบภาพของคนและสุนัข ทั้งสองอย่างมีลักษณะที่แตกต่างกัน แต่ทั้งสองอย่างก็มีจุดร่วมกันด้วย เช่นทั้งคนและสุนัข ต่างก็เป็นสัตว์เลี้ยงลูกด้วยนม ลักษณะที่แตกต่างกันนี้ ถูกแยกออกจากกันด้วย class หรืออาจกล่าวว่า นายพฤษภ์เป็น object ใน class คน ไอ้ต่าง เป็น object ที่อยู่ใน class สุนัข เป็นต้น

1.2.3 Encapsulation

หมายถึงการที่เรานำกลุ่มของข้อมูลและกลุ่มของชุดคำสั่งมาจับกลุ่มกันเป็น object หนึ่งๆ และเพื่อเป็นการกันไม่ให้ object อื่นๆสามารถเข้าถึง object นั้นๆได้โดยตรง ดังตัวอย่างที่เรากล่าวไปแล้วว่า เราไม่สามารถอ่านใจสุนัขได้โดยตรง เราอาจจะสังเกตว่ามันทำอะไร และคาดว่ามันจะรู้สึกอย่างไร การทำ encapsulation ก็มีความหมายในลักษณะนี้ซึ่งการ encapsulation นี้จะทำให้เราสามารถแยกย่อย object ต่างๆออกตามหน้าที่และบทบาทของมัน และจะทำให้เราสามารถทำ object มาใช้ใหม่ได้สะดวกยิ่งขึ้น

รูป 1.2 จะแสดงถึงการติดต่อกับ object ว่าจะต้องทำผ่าน message และ return โดยจะไม่มีการเข้าถึง data ของ object โดยตรง อย่างไรก็ตาม แม้ว่าการเข้าถึงข้อมูลของ object โดยตรงอาจจะผิดมารยาทของการเขียนโปรแกรมแบบ Object-Oriented แต่ตัวภาษาก็อาจจะยอมให้ชุดคำสั่งอื่นๆที่อยู่



รูปที่ 1.2: การติดต่อกับ Object ตามหลักการ Encapsulation

ภายนอก object เข้าถึงข้อมูลใน object โดยตรงเลยก็ได้ เพราะการเข้าถึงข้อมูลโดยผ่าน message จะทำให้โปรแกรมทำงานช้าลง

1.2.4 Message

การสื่อสารระหว่าง object หรือการติดต่อกับ object ใดๆ ทำได้ด้วยการส่ง message ไปยัง object นั้นๆ และเป็นหน้าที่ของ object นั้นๆที่จะต้องตัดสินใจว่าจะตอบสนอง message ที่ได้รับอย่างไร ขึ้นตอนในการตอบสนองนี้คือชุดคำสั่งหรือ method ที่ถูก encapsulation อยู่ใน object และในการเขียนโปรแกรม message ที่ได้รับ ก็จะทำให้ object เรียก method ที่ชื่อตรงกับ object ขึ้นมาทำงาน แต่ในบางกรณี object นั้น อาจจะไม่ method ที่ตรงกับ message ที่ได้รับก็ได้ ในภาษา Objective-C จะมีคุณลักษณะสำคัญประการหนึ่งที่เรียกว่า dynamic-binding ที่สามารถใช้จัดการกับ message ที่เราไม่รู้จักได้ เพื่อให้มองเห็นภาพของการส่ง message ชัดเจนขึ้น จะขอยกตัวอย่างที่ 1.2.1 ประกอบ

ตัวอย่างที่ 1.2.1 *การส่งและการรับ message*

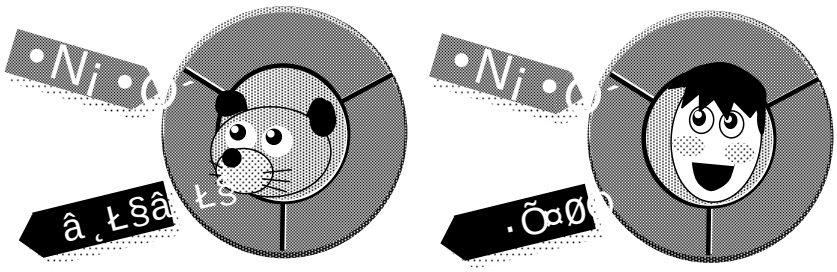
```
voice = [dog bark];
```

จากตัวอย่างจะเป็นการส่ง message `bark` ให้กับ object `dog` ซึ่งเมื่อ object `dog` ได้รับ message `bark` ก็จะเรียก method `bark` ขึ้นมาทำงาน และ method ก็ส่งค่ากลับมาแทนค่า `voice` ค่าที่ส่งกลับมานี้ คือค่าที่ method ที่ถูกเรียกขึ้นมาทำงานส่งกลับมาให้ ซึ่งอาจจะอยู่ในรูปแบบใดก็ได้ขึ้นกับ method นั้นๆ

1.2.5 Polymorphism

หมายถึงการที่ object แต่ละตัวสามารถที่จะตอบสนอง message ได้รับอย่างเดียวกันในวิธีที่แตกต่างกัน เช่นถ้าเรามี object จาก class **หมอ** กับ object จาก class **นักร้อง** แล้วส่ง message **ทำงาน** ให้กับทั้งสอง object แล้ว object **หมอ** ก็จะ**ทำงาน**ด้วยการรักษาคนไข้ ในขณะที่ object **นักร้อง** จะ**ทำงาน**ด้วยการร้องเพลงเป็นต้น การที่ object แต่ละ object จะตอบสนอง message อย่างไรขึ้นอยู่กับ method

ของแต่ละ object นั้นๆ method อาจจะมีชื่อซ้ำกันแต่อาจจะทำงานแตกต่าง หรืองานอย่างเดียวกันก็ได้ ลักษณะการตอบสนอง message ชนิดเดียวกัน แต่แตกต่างกันไปใน class ที่ต่างกันเช่นนี้ เรียกว่า **polymorphism**



รูปที่ 1.3: Polymorphism คือการตอบสนองของ object ต่อ message แบบเดียวกันในวิธีที่ต่างกัน

1.2.6 Dynamism

จะประกอบด้วยคุณลักษณะย่อยๆสามประการได้แก่

Dynamic Binding

เป็นลักษณะที่สำคัญมากประการหนึ่งของภาษา Objective-C ซึ่งเป็นลักษณะที่จะพบได้ในบางภาษาจากภาษาที่เป็นแบบ Object-Oriented ทั้งหมด ภาษาอื่นๆที่เป็นภาษาแบบ dynamic binding เช่น และภาษา Smalltalk-80 ซึ่งเป็นภาษาต้นแบบของภาษา Objective-C เป็นต้น ลักษณะที่ว่านี้คือการจัดการตีความ message ขณะที่โปรแกรมทำงานอยู่แทนที่จะตีความในระหว่างการ compile เช่นในภาษา C++ การตีความ message แบบ dynamic binding หมายถึงเมื่อ object ได้รับ message มา object จะทำการเปรียบเทียบ message ที่ได้กับ method ในขณะที่โปรแกรมกำลังทำงานอยู่ ดังนั้นโปรแกรมจึงสามารถ compile และทำงานได้โดยไม่จำเป็นจะต้องมี method ที่ตรงกับ message นั้นจริงๆ นอกจากนี้ยังทำให้การทำ Distributed Object หรือการที่โปรแกรมหนึ่งส่ง message ให้อีกโปรแกรมซึ่งไม่จำเป็นจะต้องทำงานอยู่บนเครื่องเดียวกัน เป็นไปได้ง่ายอย่างยิ่ง

Dynamic Typing

คือการที่โปรแกรมจะสามารถทราบ class ของ object ที่คุณส่ง message ไปได้โดยไม่จำเป็นต้องประกาศไว้อย่างเจาะจงในตอน source code ข้อโปรแกรม คุณสามารถกำหนดตัวแปรอิสระที่สามารถจะชี้ไปยัง class อะไรก็ได้ และเมื่อคุณส่ง message ไปยัง object นั้น โปรแกรมก็จะสามารถเลือก method ที่ถูกต้องหรือตรงกับ class นั้นๆขึ้นมาทำงานโดยที่คุณไม่ต้องบอกว่า message นั้นถูกส่งไปยัง class ใด

ในบางภาษาเช่นภาษา C คุณจะต้องบอกให้โปรแกรมทราบก่อน compile ว่าคุณต้องการจะจัดการกับข้อมูลแบบใด หรือที่เรียกว่าการ *casting* นั่นเอง

Dynamic Loading

หมายถึงการที่คุณสามารถ load object ขึ้นมาจาก file system เมื่อคุณต้องการจะใช้งาน โดยไม่จำเป็นต้อง link ไว้ล่วงหน้า ลักษณะการทำงานเช่นนี้ก็เช่นเดียวกับการใช้ shared library หรือการใช้ function `dlopen()` ในภาษา C เพื่อเรียกชุดคำสั่งที่คุณต้องการใช้ขึ้นมาจาก file system โดยไม่จำเป็นต้องทำการ link ไว้ก่อน

1.2.7 การสืบสกุล (Inheritance)

class บาง class จะมีลักษณะร่วมกัน เช่น class คน กับ class สุนัข ต่างก็มีลักษณะของ class สัตว์เลี้ยงลูกด้วยนม จากลักษณะดังกล่าวสามารถเรียกได้ว่า class คนและ class สุนัข ต่างก็เป็น sub-class ของ class สัตว์เลี้ยงลูกด้วยนม และ class สัตว์เลี้ยงลูกด้วยนม เป็น parent class ของ class คน และ class สุนัข เพื่อความเข้าใจยิ่งขึ้น เราอาจเขียนแผนผัง class ต่างๆประกอบด้วยข้อมูลและชุดคำสั่งดังตัวอย่างที่ ex:classlist

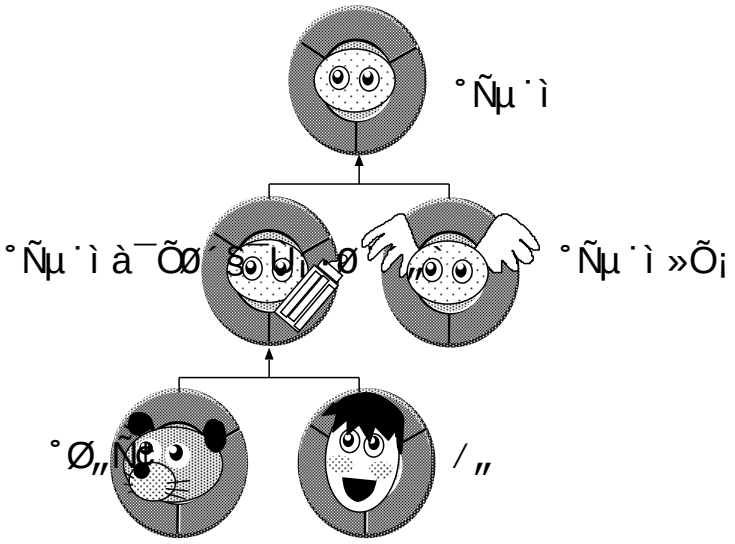
ตัวอย่างที่ 1.2.2 *ตัวอย่างของการลำดับ Class และความสัมพันธ์*

```
class:MAMMAL
parent class:ANIMAL
data:gender
method:eat,sleep

class:DOG
parent class:MAMMAL
data:name
method:bark

class:HUMAN
parent class:MAMMAL
data:name
method:say
```

จากตัวอย่างที่ ?? class MAMMAL หรือสัตว์เลี้ยงลูกด้วยนม จะเป็น subclass ของ class สัตว์ มีข้อมูลอย่างเดียวคือเพศ และมี method สอง method เพื่อใช้ในการกิน(eat) และนอน (sleep) ในขณะที่ class คน (HUMAN) และสุนัข (DOG) ไม่จำเป็นต้องประกาศข้อมูลและชุดคำสั่งที่ปรากฏใน parent class ซ้ำ ลักษณะเช่นนี้เรียกว่าการสืบสกุลหรือ **inheritance** และเมื่อเป็นการสืบสกุล

ทั้ง class คนและสุนัขจะประกอบด้วย method eat, sleep และข้อมูล gender โดยที่ไม่จำเป็นจะต้องประกาศซ้ำ เพราะทั้งสอง class จะได้สืบทอด (derive) ข้อมูลและชุดคำสั่งดังกล่าวมาจาก parent class โดยอัตโนมัติ อย่างไรก็ตาม อาจมี class บาง class ที่ไม่ได้สืบสกุลมาจาก class ใดๆเลยหรือไม่มี parent class เราเรียก class ประเภทนี้ว่า root class

การสืบสกุล ทำให้เราสามารถนำ code เก่ามาใช้ใหม่ได้สะดวกขึ้น เราอาจจะเคยสร้าง class ไว้แล้ว แต่เมื่อเราต้องการเพิ่มเติมความสามารถให้กับ class เดิม เราก็เพียงแต่สร้าง class ใหม่โดยให้สืบสกุลจาก class เดิม แล้วเพิ่มเติมชุดคำสั่งและข้อมูลที่เราต้องการลงไป ซึ่งจะทำให้ไม่กระทบต่อการทำงานของส่วนอื่นๆของโปรแกรมซึ่งใช้ class นั้นอยู่ นอกจากนี้ เรายังอาจสร้าง class ขนาดเล็กๆไว้จำนวนหนึ่ง เพื่อให้ class อื่นๆนำใช้งานร่วมกันตามความเหมาะสมได้อีกด้วย เราจะพบได้ว่า Object-Oriented frame work ต่างๆ ไม่ว่าจะใช้ภาษาอะไรก็ตาม จะประกอบด้วย class ขนาดเล็กจำนวนมาก เมื่อเราใช้งาน frame work นั้นๆ ก็เพียงแต่ดึง class ย่อยๆเหล่านั้นมาใช้งาน หรืออาจจะสืบสกุลจาก class เหล่านั้นก็ได้ และถ้า frame work นั้นๆออกแบบมาให้ทุก class สืบสกุลมาจาก class เพียง class เดียวหรือมี root class ร่วมกันเพียง class เดียว เราจะเรียกการออกแบบนั้นว่าใช้หลักการแบบ *single object hierarchy* ซึ่งจะช่วยให้เราลดความซับซ้อนในการควบคุมและจัดการ object จำนวนมากลงได้



รูปที่ 1.4: เราสามารถสืบทอดข้อมูลและชุดคำสั่งจาก parent class ผ่านการสืบสกุล

1.3 แนวคิดเรื่องการออกแบบโปรแกรมแบบ Object-Orieted

1.3.1 การแบ่งงานออกเป็นหน่วยย่อย

การเขียนโปรแกรมแบบเดิมมักเกิดปัญหาเมื่อ source code มีขนาดใหญ่ เช่นปัญหาในการทำนุบำรุงหรือปรับปรุงแก้ไขเพิ่มเติม และหากมีผู้พัฒนาหลายคนก็อาจจะเกิดความสับสนโดยเฉพาะเมื่อผู้พัฒนาไม่สามารถทำงานพร้อมๆกันและสามารถติดต่อกันได้อย่างสะดวก นอกจากนี้ การนำแนวคิดแบบ Object-Oriented จะช่วยลดปัญหาเหล่านี้ได้มาก เพราะการแบ่งโปรแกรมออกเป็นส่วนย่อยๆที่อิสระจากกันจะช่วยลดความสับสนในการจัดการ การพัฒนาโปรแกรมในแบบ Object-Oriented จึงเน้นไปที่การแบ่ง object ต่างๆเพื่อให้ทำงานร่วมกันได้อย่างมีประสิทธิภาพที่สุด เราเรียกกลุ่มของ object ที่ใช้ทำงานร่วมกันโดยมีจุดมุ่งหมายร่วมกันว่า object frame work นอกจากนี้การออกแบบโปรแกรมแบบ Object-Oriented จะทำให้คุณสามารถนำ code ที่เขียนแล้วมาใช้ใหม่ หรือเพิ่มเติมความสามารถได้สะดวกยิ่งขึ้นกว่าการเขียนด้วยเทคนิคแบบ procedural ซึ่งคุณจะต้องเปิดให้หน่วยต่างๆจัดการกับข้อมูลได้โดยตรง และสร้างความสับสนเมื่อคุณต้องทำงานเป็นคณะใหญ่ๆ เพราะเมื่อคุณเปลี่ยนแปลงจุดหนึ่ง ก็จะส่งผลกระทบต่อจุดอื่นๆทั้งหมดไปด้วย

1.3.2 การนำมาใช้ซ้ำได้อย่างสะดวก

ในการแบ่งงานออกเป็นหน่วยย่อย มีหลักสำคัญที่ต้องระลึกควบคู่ไปด้วยเสมอ คือเราควรจะเน้นออกแบบให้สามารถนำ object มาใช้ซ้ำได้อย่างสะดวก ต้องคิดว่าในอนาคต เราจะมีแนวโน้มในการใช้ object นั้นๆอย่างไร ต้องพยายาอมออกแบบการสืบสกุลอย่างเหมาะสม พยายามออกแบบ message ให้เป็นกลาง หรือให้สามารถเข้าใจได้ง่าย และต้องพยายามลดความซ้ำซ้อนของ object ใน frame work ด้วย และสิ่งสำคัญที่จะละเลยมิได้คือการทำเอกสารประกอบ object นั้นๆให้ชัดเจนและเข้าใจได้ง่าย

1.3.3 การใช้ Dia เพื่อสร้าง UML อย่างง่าย

ในการออกแบบ object นี้ เราอาจใช้เครื่องมือต่างๆเพื่อช่วยในการออกแบบ เช่นโปรแกรม Dia [1] ซึ่งเป็นซอฟท์แวร์เสรีที่สามารถนำมาใช้ในการเขียน UML ก็ได้

การออกแบบ object ด้วย UML

การใช้ Dia สร้าง UML

[1] http://www.lysator.liu.se/ alla/dia/

บทที่ 2

ภาษา Objective-C

2.1 ภาพรวมและประวัติของภาษา Objective-C

2.2 การสร้างและการใช้งาน Class ในภาษา Objective-C

2.2.1 รูปแบบการประกาศ Class และการสร้าง Object โดยทั่วไป

การเขียนโปรแกรมด้วยภาษา Objective-C จะประกอบด้วยการประกาศ class และการนำ class ที่ประกาศมาใช้งาน ในขั้นแรก จะขอยกตัวอย่างประกอบการเขียนโปรแกรมภาษา Objective-C แบบง่ายๆดังตัวอย่างที่ 2.2.1

ตัวอย่างที่ 2.2.1 *ตัวอย่างโปรแกรมภาษา Objective-C อย่างง่าย*

```
#import <objc/Object.h>
@interface Person : Object
{
    char *firstname;
    char *lastname;
    int age;
}
- setName:(char *)newname lastName:(char*)newlastname;
- setAge:(int)newage;
- introduce;
- (int) age;
@end
```

```
@implementation Person
- setName:(char *)newname lastName:(char*)newlastname
{
    lastname = newlastname;
    firstname = newname;
}
- setAge:(int)newage
{
    age = newage;
}
- (int) age
{
    return age;
}
- introduce
{
    printf("I am %s %s.\n",firstname,lastname);
}
@end

int main() {
    Person *aPerson;
    aPerson = [Person new];
    [aPerson setName:"Pruet" lastName:"Boonma"];
    [aPerson setAge:40];
    [aPerson introduce];
    printf("He is %d year old.\n", [aPerson age]);
}
```

ถ้าคุณใจร้อน อยากทดลองโปรแกรมนี้อย่างรวดเร็ว ให้สร้างแฟ้มข้อมูลโดยสมมติว่าชื่อ `file.m` โดยมีเนื้อหาตามตัวอย่าง แล้ว compile ด้วยคำสั่ง

```
gcc -o test file.m -lobjc -lpthread
```
[1]

คุณจะได้โปรแกรมชื่อ test ออกมา ให้ทดลองเรียกขึ้นมาทำงาน

```
./test
```

การนิยาม class ในโปรแกรมภาษา Objective-C จะประกอบด้วยส่วนสำคัญหลักสองส่วนคือส่วน `@interface` และส่วน `@implementation` ใน class หนึ่งๆจะประกอบด้วยสองส่วนนี้ โดยทั่วไปเรามักจะเก็บส่วน `@interface` ไว้ในแฟ้มสกุล `.h` และส่วน `@implementation` ในแฟ้มสกุล

[1] ในกรณีใช้ gcc รุ่นหลังๆ คุณอาจไม่ต้องใส่ `-lpthread` ก็ได้

`.m` จากตัวอย่างจะเป็นการประกาศ class ชื่อ `Person` โดยให้สืบสกุลจาก class `Object` ซึ่งเป็น class ชั้นบนสุด หรือ root class

Interface

จะใช้นิยาม class ในส่วนของข้อมูลของ object และเป็นการประกาศให้ class อื่นได้ทราบว่าสามารถส่ง message อะไรมาให้ได้บ้าง ที่จริงแล้วในการประกาศ `@interface` เราไม่จำเป็นจะต้องประกาศในส่วนของ message ด้วยก็ได้เพราะภาษา Objective-C จะใช้ dynamic-binding ในการเลือก method ขึ้นมาทำงานตาม message ที่ได้รับในเวลาที่โปรแกรมทำงานจริง ดังนั้น compiler จะทำเพียงแค่ตักเตือนเราเท่านั้นหากเราไม่มีส่วนที่ประกาศ message ใน `@interface` แต่ประกาศ method ไว้ในส่วนของ `@implementation` ซึ่งเราจะได้กล่าวถึงต่อไป

จากตัวอย่างที่ 2.2.1 ในส่วนของ `@interface` จะใช้ประกาศว่า class ต้องการข้อมูลสามตัวเพื่อนิยามสถานะของบุคคลคนหนึ่ง ข้อมูลเหล่านี้ประกอบด้วยชื่อ นามสกุล และอายุ เป็นการประกาศเช่นเดียวกับการประกาศตัวแปรในภาษา C ตามปรกติ ข้อมูลเหล่านี้ยังสามารถเข้าถึงได้โดยตรงจาก method ของ class Person นอกจากนั้น เรายังประกาศถึง method สี่ตัวซึ่งมีหน้าที่ต่างๆกันไป (เราจะกล่าวถึงรูปแบบของชื่อ method ในเรื่องของ message และ selector) มีข้อสังเกตว่าการประกาศตัวแปรใน class จะต้องไม่ซ้ำกับตัวแปรใน parent class หรือ class ที่อยู่สูงขึ้นไปตามลำดับของการสืบสกุล แต่ในส่วนของชื่อ method นั้น สามารถใช้ชื่อเดียวกับตัวแปรได้เช่น `age`

รูปแบบของการใช้ `@interface` จะเริ่มต้นด้วย `@interface` ตามด้วยชื่อของ class ซึ่งนิยมให้นำหน้าด้วยตัวอักษรใหญ่ตามด้วยเครื่องหมาย `":"` แล้วตามด้วยชื่อ class ที่เราสืบสกุลมา หรือหากไม่มี class ที่สืบสกุลมาก็ไม่ต้องใส่เครื่องหมายและชื่อ parent class ซึ่งถ้าไม่มี parent class ก็จะถือว่า class นั้นเป็น root class แต่ในการเขียนโปรแกรมโดยทั่วไปแล้วเราจะต้องมี parent class เสมอ ถ้าไม่รู้จะใช้ parent class ใด ก็ควรใส่ root class ลงไป ไม่ควรประกาศ root class ตัวอื่นเพิ่มอีก เว้นแต่จะมีเหตุผลที่สมควรจริงๆซึ่งเราจะได้ยกตัวอย่างในบทเรียนขั้นสูงของหนังสือเล่มนี้

Implementation

จะใช้ในการนิยามส่วน method ของ class ตามที่ได้ประกาศไว้ในส่วน `@interface` แต่เราก็สามารถประกาศ method โดยไม่จำเป็นต้องมีอยู่ในส่วน `@interface` ก็ได้ดังที่ได้กล่าวไปแล้ว ลักษณะของการประกาศ method ก็เหมือนกับการประกาศ function ในภาษา C ทั่วไป แต่เราสามารถอ้างอิงถึงตัวแปรที่อยู่ในส่วน `@interface` ได้โดยตรง นอกจากนี้ยังสามารถอ้างถึงข้อมูลที่อยู่ใน parent class หรือสูงขึ้นไปตามลำดับได้โดยตรงเช่นกัน การอ้างถึงข้อมูลใน object นี้ตามปรกติเราจะไม่สามารถอ้างถึงข้อมูลภายใน object จาก method ที่ไม่ได้เป็นส่วนหนึ่งของ object หรือถูกประกาศไว้ใน class ทั้งนี้เป็นไปตามหลักการ Encapsulation การอ้างถึงตัวแปรภายใน object จึงควรทำผ่าน method เท่านั้น เช่น method `-age` ซึ่งจะ `return` ค่าอายุหรือค่าของตัวแปร `int age` แต่อย่างไรก็ตาม เราสามารถระบุขอบเขตของการเข้าถึงตัวแปรต่างๆใน object ได้ด้วยการใช้ keyword 3 ตัวได้แก่

`@private` ทำให้เข้าถึงได้เฉพาะ method ใน class เดียวกัน

`@protected` ทำให้เข้าถึงได้เฉพาะ method ใน class นั้นรวมถึง class ที่สืบสกุล

`@public` ทำให้เข้าถึงได้อย่างอิสระ

ซึ่งจะได้กล่าวถึงในรายละเอียดต่อไป

2.2.2 หลักการทำงานพื้นฐานของการส่ง Message, Selector และ Method

การติดต่อกับ object ทำได้ด้วยการส่ง message ไปยัง object ซึ่งในภาษา Objective-C หมายถึงการส่ง selector ไปยัง object ซึ่ง selector ก็คือข้อมูลรูปแบบหนึ่ง เราสามารถประกาศข้อมูลแบบ selector ได้ด้วยการใช้ keyword `SEL` เช่น `SEL aSelector;` เช่นเดียวกับการประกาศข้อมูลแบบอื่นๆ การทำงานของ dynamic binding ในภาษา Objective-C ก็เกิดขึ้นได้ด้วย selector นี่เอง เมื่อ object ได้รับ message ที่ถูกส่งมาในรูปของ selector แล้ว object ก็จะนำ selector ที่ได้มาเปรียบเทียบกับ method ที่ถูกประกาศไว้ใน class ซึ่งถ้าไม่พบ object ก็สามารถนำ selector ที่ได้รับมาตรวจสอบหรือจัดการต่อไป ไม่ว่าจะทำเป็นไม่สนใจ message นั้น หรือจะทำตัวเป็น proxy ส่ง selector นั้นต่อไปยัง object อื่นๆก็ได้ กระบวนการทั้งหมดนี้เกิดขึ้นโดยอัตโนมัติ เราสามารถเพิ่มเติม method เพื่อจัดการกับ message ที่เราไม่รู้จักได้ เช่นเราอาจจะเปรียบเทียบ selector ที่ได้รับมาด้วยการประกาศ method พิเศษบางกลุ่ม ซึ่งจะถูกส่งมาให้เราเมื่อ object ไม่รู้จัก message ที่ได้รับพร้อมกับผ่านค่า selector ที่ object นั้นไม่รู้จักมาด้วย โดยเราจะกล่าวถึงรายละเอียดในส่วนของ dynamic binding

message ที่ถูกส่งมายัง object จะอยู่ในรูปแบบ `[object message];` และถ้า method มีการ `return` ค่าใดๆกลับมา เราก็สามารถรับค่าที่ถูกส่งกลับมาได้เช่นเดียวกับการเรียกใช้ function ในภาษา C เช่น `return_value = [object message];` การอ้างถึงชื่อ message จะมีลักษณะดังนี้คือ

> message จะเป็นข้อความติดต่อกันไปโดยจะนำหน้าด้วยตัวอักษรประกอบสัญลักษณ์ "`_`" และ "`:`"
>
> สัญลักษณ์ "`:`" จะใช้สำหรับการแทรกข้อมูลระหว่าง message ชื่อหรือชนิดของข้อมูลจะไม่มีผลกับชื่อ message

ดังนั้น จากตัวอย่างที่ 2.2.1 การประกาศถึง message

```
setName:(char *)newname lastName:(char*)newlastname;
```

ในลักษณะนี้จะหมายความว่า message นี้คือ `setName:lastName:` ส่วนที่อยู่หลังเครื่องหมาย "`:`" จะประกอบด้วยชนิดของข้อมูลที่จะถูกส่งเป็นตัวแปรมากับ message นั้นภายในวงเล็บ ตามด้วยชื่อของตัวแปร และเมื่อพ้นจากชื่อของตัวแปรก็จะนับเป็นส่วนหนึ่งของชื่อ message ต่อไป ซึ่งในบางกรณีเราอาจส่ง message ให้กับ object ในรูปแบบของการส่ง selector ได้ด้วย เช่นแทนที่เราจะส่ง message `introduce` ให้กับ object `aPerson` ตามตัวอย่าง เราก็อาจจะแทนได้ด้วย `[aPerson perform:@selector(introduce)];` ซึ่งก็จะให้ผลลัพธ์อย่างเดียวกัน แต่อย่างไรก็ตาม การส่ง selector ให้ object โดยผ่าน message `perform:` ตามลักษณะที่กล่าวไปนั้น จะแตกต่างจากการเขียนโปรแกรมด้วย GNUstep Foundation ซึ่งจะใช้ message `performSelector:` แทน โดยจะได้กล่าวในรายละเอียดต่อไปในส่วนของ Foundation

ในกรณีที่เราต้องการให้ object ส่ง message ถึงตัว object เอง เราจะใช้ keyword `self` เป็นผู้รับ message และเราสามารถส่งผ่าน message ไปยัง parent class ได้ด้วยตรงด้วยการอ้างผ่าน keyword `super` ดังตัวอย่างที่ 2.2.2

ตัวอย่างที่ 2.2.2 *การส่ง message จาก method ของ object ถึง object เอง*

```
- myMethod
{
    [self myAnotherMethod];
}

- myAnotherMethod
{
    [super aMethodInParentClass];
}
```

แม้ว่าในภาษา Objective-C จะมี message อยู่เพียงประเภทเดียวประกอบด้วย method 2 ประเภท ได้แก่ class method และ instance method ซึ่งจะไม่มีความแตกต่างในลักษณะของการรับส่ง message แต่จะแตกต่างกันในแง่ของการประกาศและผู้รับ เราจะขึ้นต้นการประกาศ class method ด้วยเครื่องหมายบวก "+" และจะใช้เครื่องหมายลบ "-" ขึ้นต้นการประกาศ instance method ซึ่งความแตกต่างของ method ทั้งสองประเภทนี้ก็คือ class method และ instance method จะหมายถึง method สำหรับ class object และ instance object ตามลำดับ ในส่วนต่อไป เราจะได้กล่าวถึงความแตกต่างระหว่าง class object และ instance object

2.2.3 Class Object และ Object Instance

เรื่องความแตกต่างระหว่าง class object และ object instance อาจจะสร้างความสับสนให้กับผู้เริ่มต้นศึกษาโปรแกรมแบบ Object-Oriented ได้ไม่มากก็น้อย เพื่อให้เข้าใจหลักการนี้โดยง่าย ให้ลองเปรียบเทียบ class object เป็นโรงงานที่ผลิตสินค้าอย่างใดอย่างหนึ่ง และให้ object instance (หรือต่อไปจะเรียกโดยย่อว่า instance) คือผลิตภัณฑ์ที่ได้จากโรงงานนั้นๆ จากลักษณะเช่นนี้ทำให้บางครั้งเราก็เรียก class object ว่า object factory ดังนั้น จากตัวอย่างที่ 2.2.1 เราจะพบว่าในส่วน `main()` เราได้ประกาศตัวแปรชื่อ `aPerson` โดยให้ชี้ไปยัง class `Person` ตัวแปร `aPerson` นี้ก็คือ instance และ Person คือ class object จะสังเกตได้ว่าเรากำหนดค่าของ `aPerson` ด้วยคำสั่ง `aPerson = [Person new];` ซึ่งหมายถึงการส่ง message `+new` ให้กับ class `Person` เราจะไม่พบการประกาศ method `+new` ใน Person เพราะที่จริงแล้วมันจะถูกประกาศไว้ใน class `Object` ซึ่งเป็น parent class ของ class `Person`

class จะเข้าใจแต่ class method เช่นเดียวกับ instance ที่จะเข้าใจแต่ instance method เท่านั้น และในการประกาศ class หนึ่งๆ เราจะประกาศทั้ง class method และ instance method ร่วมกันดังตัวอย่างที่ 2.2.3 สิ่งที่แตกต่างกันระหว่าง class method และ instance method ก็คือ class method

ไม่สามารถเข้าถึงตัวแปรที่ประกาศใน `@interface` ได้ ตัวแปรที่ถูกประกาศอยู่ใน `@interface` เรียกว่า instance variable ซึ่งจะถูกเข้าถึงได้จาก instance method เท่านั้น

ที่จริงแล้ว การส่ง message `+new` ไปยัง object `Person` ตามตัวอย่างที่ 2.2.1 จะเป็นรูปแบบโดยย่อของการสร้าง instance ใหม่ เราได้เรียนรู้หลักการ inheritance มาแล้วว่าถ้า class นั้นไม่รู้จัก message ที่ได้รับ message นั้นจะถูกผ่านขึ้นไปตามลำดับของการสืบสกุล และเราจะรู้ว่า instance นั้นรู้จัก message ก็ต่อเมื่อ message นั้นถูกผ่านขึ้นไปจนถึง root class ซึ่งในที่นี้ก็คือ class `Object` ดังนั้นเมื่อเราส่ง `+new` ไปยัง class `Person` มันก็จะผ่าน message ต่อไปยัง class `Object` ซึ่งจะเปลี่ยนมันให้เป็น 2 message ต่อเนื่องกัน คือ `+alloc` และ `-init` หรือเราอาจบอกได้ว่า `aPerson = [Person new];` มีความหมายเช่นเดียวกับ `aPerson = [[Person alloc] init];` หรือ `aPerson = [Person alloc]; [aPerson init];` ซึ่งหมายความว่าเมื่อ class `Person` ได้รับ message `+alloc` มันก็จะจัดการจองหน่วยความจำขนาดเท่าที่ class นั้นต้องการ เมื่อจองแล้วก็จะ `return` pointer ชี้ไปยังหน่วยความจำที่จองไว้ออกมาในรูปของ instance จากนั้นส่ง message `-init` ให้กับ instance เพื่อตั้งค่าพื้นฐานตามความต้องการของ class นั้นๆ ถ้าเราเปิด source code ส่วน `@implementation` ของ object `Object` ขึ้นมา เราก็จะพบว่า message `+new` ถูกนิยามไว้ดังนี้

```
+ (id) new
{
    return [[self alloc] init];
}
```

จะเห็นว่าการใช้ new เพื่อสร้าง instance ใหม่จะไม่มีความแตกต่างจากการใช้ `+alloc` ตามด้วย `-init` เลย แต่การใช้ `+alloc` ตามด้วย `-init` ย่อมทำงานได้เร็วกว่าเพราะ object ไม่ต้องเสียเวลาในการเปรียบเทียบถึงสองขั้นตอน

ตามปรกติเราจะใช้ class method สำหรับจองหน่วยความจำและตระเตรียมค่าเบื้องต้นต่างๆใน instance ที่ถูกสร้างขึ้นใหม่ แต่เรายังสามารถใช้ class method สำหรับปรับค่าพื้นฐานของ class ได้อีกด้วย แต่เนื่องจากในภาษา Objective-C ไม่มี class variable และมีแต่ class method เท่านั้น การจะประกาศตัวแปรสำหรับ class จะใช้การประกาศตัวแปรแบบ `static` ภายในส่วน `@implementation` เช่นการประกาศตัวแปร `fake_class_variable` ตามตัวอย่างที่ 2.2.3

ตัวอย่างที่ 2.2.3 *การประกาศ instance method และ class method*

```
@interface MyObject : Object
{
    int x;
}
- myInstanceMethod;
+ myClassMethod;
@end
```

```
@implementation MyObject
static int fake_class_variable;
- myInstanceMethod
{
}
+ myClassMethod
{
}
@end
```

2.2.4 การ Override Method

text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text

2.2.5 การทำ Dynamic Binding ในภาษา Objective-C

text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text

2.2.6 Dynamic Typing และ Static Typing

text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text

2.2.7 Dynamic Loading

2.2.8 Categories และ Protocol

2.3 GNU Objective-C Compiler

text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text

2.3.1 การ Compile และการ Link

text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text

2.3.2 การเรียกใช้งาน GC (gabbage collector) ใน GNU Objective-C Compiler

text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text

2.3.3 Objective-C++

text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text text abc [2] def [3]

[2] http://www.areax.net/objcpp.pdf
[3] http://www.hey.com

2.4 ข้อเปรียบเทียบภาษา Objective-C กับภาษาแบบ Object-Oriented อื่นๆ

ภาษา Objective-C มีจุดเด่นอยู่ที่การเป็นภาษาที่เพิ่มเติมเพียงเล็กน้อยจากภาษา C ธรรมดาเพื่อให้มีความสามารถในเรื่อง Object-Oriented สามารถใช้งานร่วมกับ library ต่างๆในภาษา C ได้เป็นอย่างดี เรียนรู้ได้ง่าย และนอกจากนี้ความเป็น dynamic binding ทำให้การพัฒนาโปรแกรมทำได้รวดเร็วยิ่งขึ้น โดยเฉพาะกับกลุ่มทำงานขนาดใหญ่ นอกจากนี้ยังมี frame work ที่ใช้เพิ่มความสามารถของภาษาอย่างเช่น GNUstep ที่ช่วยให้การพัฒนาโปรแกรมลดความยุ่งยากลง ตลอดจนการพัฒนาโปรแกรมแบบ GUI ก็ทำได้ง่าย และไม่ยุ่งยากซับซ้อนอีกด้วย

บทที่ 3

GNUstep Foundation

3.1 ภาพรวมของ Foundation Framework

class ใน foundation เป็น class พื้นฐานที่ไม่เกี่ยวข้องกับการเขียนโปรแกรมแบบ GUI (graphic user interface) เราสามารถใช้ class ต่างๆเหล่านี้ในการสร้างโปรแกรมที่ไม่ จำเป็นต้องแสดงผลแบบหน้าต่าง เช่นโปรแกรมแบบ daemon หรือ internet service ต่างๆ

class ใน foundation จะแตกต่างจาก class ที่เราใช้ใน Objective-C แบบทั่วไป แต่ ยังมีหลักการเดียวกัน ส่วนที่แตกต่างกันที่ต้องกล่าวถึงเป็นจุดแรกคือ root class ซึ่งใน Objective-C จะใช้ root class เป็น `Object` แต่ใน GNUstep foundation จะใช้ `NSObject` เป็น root class (root class ใน foundation อีกตัวหนึ่งคือ NSProxy ซึ่งเป็นกรณีพิเศษและจะถูกใช้งานเกี่ยวกับ distributed objects และเราจะยัง ไม่นำมากล่าวถึง จนกว่าจะถึงบทที่เกี่ยวข้อง) นอกจากนั้น class ใน foundation และใน application kit จะขึ้นต้นด้วย *NS* ทั้งหมด (หมายถึง NeXTstep) โดยที่ class ใน foundation เหล่านี้ จะถูกแยกย่อยออกตามหน้าที่ที่แตกต่างกันไป เพื่อให้ใช้งาน ได้อย่างอิสระและตรงตามวัตถุประสงค์ ซึ่งเราจะได้กล่าวถึงต่อไปในรายละเอียดของแต่ ละ class

3.2 Class ในแบบ Foundation

3.2.1 การประกาศ Class เพื่อสืบสกุลจาก Class ใน Foundation

3.2.2 การสร้างและยกเลิก Object Instance แบบพื้นฐาน

3.2.3 การบริหารหน่วยความจำอัตโนมัติ (NSAutoreleasePool)

3.3 Class และการใช้ Selector โดยทั่วไปในแบบ Foundation

3.3.1 การใช้งาน Selector

respondsToSelector:

3.4 บางส่วนของ Class ที่สำคัญและการใช้งานจริงเบื้องต้น

3.4.1 NSObject

3.4.2 String

3.4.3 Array

3.4.4 Dictionary

3.4.5 เวลา

3.5 หลักการของ Class แบบ Container

3.6 ลำดับและการสืบสกุลของ Class ทั้งหมดใน Foundation

3.7 บทบาทหน้าที่ของ Class ต่างๆใน Foundation

บทที่ 4

GNUstep Application Kits

บทที่ 5

การพัฒนาโปรแกรมด้วย GNUstep ในสถานการณ์จริง

5.1 เทคนิคพิเศษที่สำคัญ

5.1.1 การทำ Distributed Objects

5.1.2 thread

5.1.3 การทำ Multiple Inheritance ใน Objective-C ด้วยการ Forward

5.1.4 การเร่งความเร็วของการส่ง message ในจุดที่เป็นคอขวด

5.1.5 การทำ Bundle

5.2 การใช้ ProjectCenter

5.3 การใช้ GORM (GNUstep Object Relationship Modeler)

5.4 การใช้งาน GSXML เพื่อจัดการกับ XML

5.5 การพัฒนา Web Application ด้วย GNUstepWeb

5.6 การพัฒนา Application 3 มิติด้วย GNU 3DKit

บทที่ 6

ประชาคม GNUstep

6.1 บทบาทของซอฟท์แวร์เสรีและ GNU project

6.1.1 ความหมายของเสรีภาพ

6.1.2 โครงการอื่นๆที่เกี่ยวข้องกับ GNU

6.1.3 สถาบันทางวัฒนธรรมของ Free Software Hacker

6.2 แหล่งสารสนเทศเกี่ยวกับ GNUstep บน Internet

6.2.1 Web Site

6.2.2 Usenet, Mailing List

บทที่ 7

GNU Free Documentation License

Version 1.1, March 2000

Copyright ©2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License

principally for works whose purpose is instruction or reference.

7.1 Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modifica-

tion. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

7.2 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

7.3 Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

7.4 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).

State on the Title page the name of the publisher of the Modified Version, as the publisher.

Preserve all the copyright notices of the Document.

Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

Include an unaltered copy of this License.

Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.

Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties -- for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

7.5 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled “History” in the various original documents, forming one section entitled “History”; likewise combine any sections entitled “Acknowledgements”, and any sections entitled “Dedications”. You must delete all sections entitled “Endorsements.”

7.6 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7.7 Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an “aggregate”, and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

7.8 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

7.9 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

7.10 Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright CYEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.