บทที่ 2

การโปรแกรมเชิงวัตถุ

ความเป็นมาของการโปรแกรมเชิงวัตถุ

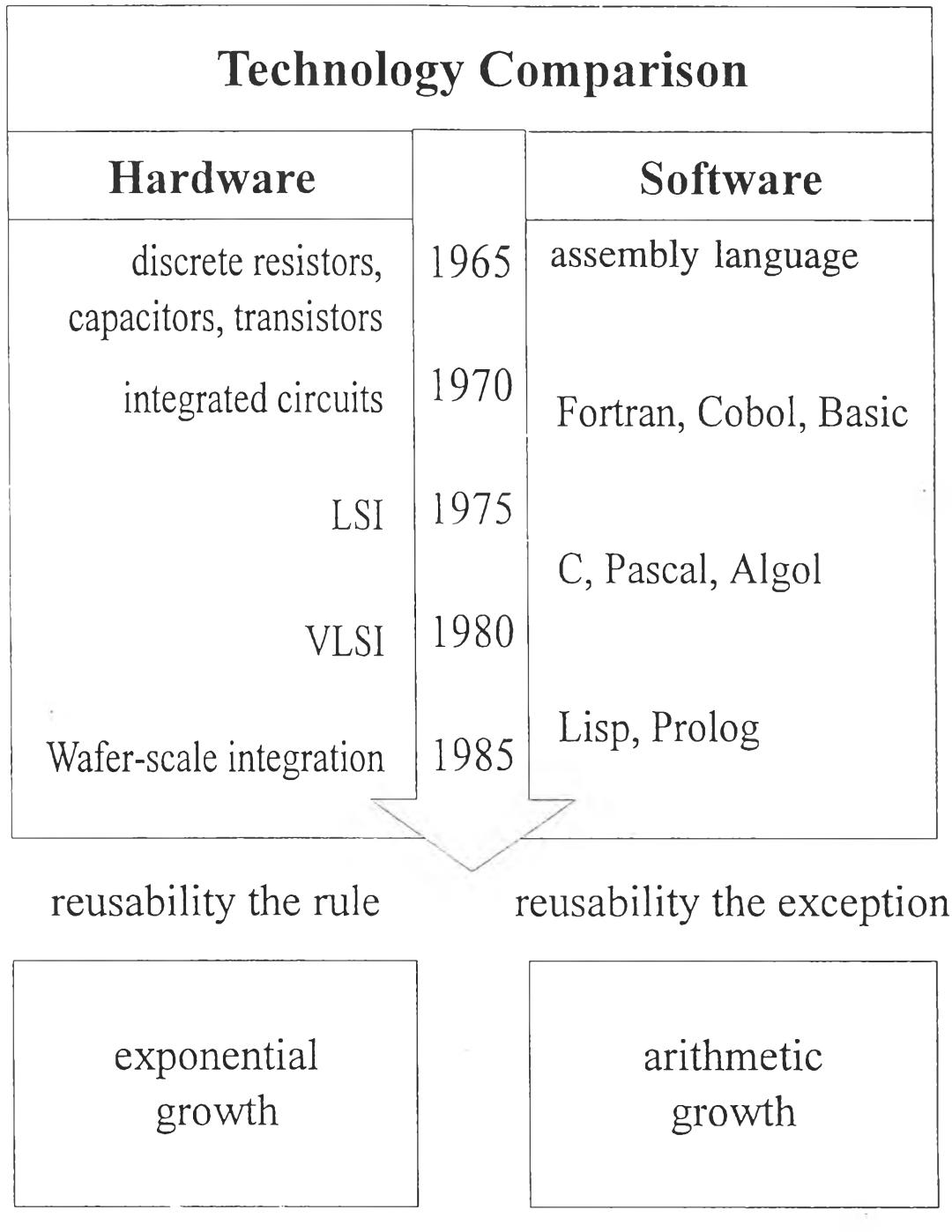
การโปรแกรมเชิงวัตถุนั้นเกิดขึ้นมาได้อย่างไร การโปรแกรมเชิงวัตถุเกิดขึ้นมาได้ก็มีสาเหตุมาจาก วิกฤตกาลของซอฟต์แวร์ (Software Crisis)

1.1 วิกฤตกาลของซอฟต์แวร์ เป็นปัญหาซึ่งเกิดขึ้นในการผลิตซอฟต์แวร์ขึ้นมา ซึ่งพอจะแจกแจงได้ดังนี้

- ซอฟต์แวร์มีราคาสูง เนื่องจาก
  - การแก้ไขบางจุดของซอฟต์แวร์จะทำให้กระทบไป ทั่วตัวซอฟต์แวร์
  - ไม่มีการนำบางส่วนกลับมาใช้ใหม่ (Software Reusability)
  - ฮาร์ดแวร์ราคาสูง ทำให้ต้นทุนในการผลิตซอฟต์แวร์ราคาสูงตามไปด้วยเนื่องจากซอฟต์แวร์ต้องทำงานอยู่บนฮาร์ดแวร์
- ซอฟต์แวร์มีความซับซ้อนเพิ่มมากขึ้น เมื่อเวลาผ่านไปซอฟต์แวร์มีขนาดใหญ่ขึ้น ทำให้เกิดความซับซ้อนยากต่อการทำความเข้าใจ จึงทำให้การบำรุงรักษากระทำได้ยาก ดังนั้นตัวซอฟต์แวร์จะมีคุณภาพต่ำลง

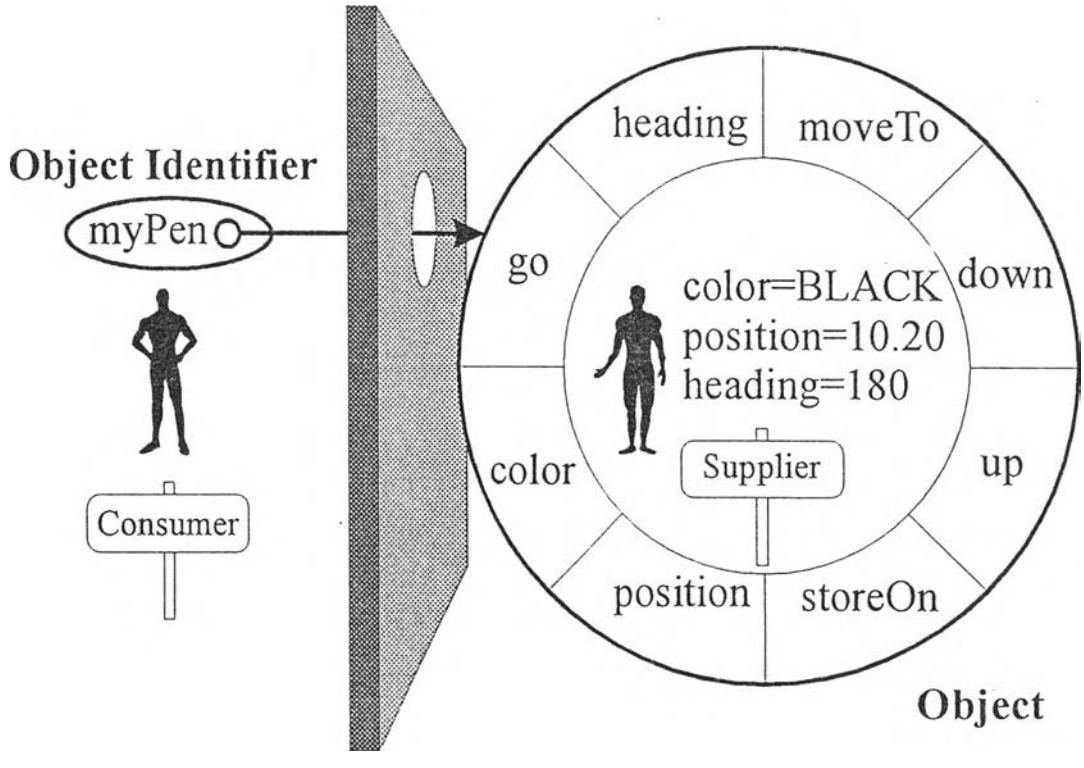
สรุป ในการพัฒนาระบบงานขนาดใหญ่ ปัญหาที่มักจะเกิดขึ้นก็คือสิ้นเปลืองกำลังคนและทรัพยากร ตลอดจนงบประมาณจำนวนมาก นอกจากนี้ยังมีความยากลำบากในการติดตามดูแลระบบให้ถูกต้องอยู่เสมอ และหากมีการพัฒนาระบบใหม่โดยมากแล้วเรามักจะไม่สามารถใช้ระบบที่เคยพัฒนาไว้ได้ ต้องทำการพัฒนาขึ้นใหม่ทั้งหมด

1.2 การแก้ปัญหาวิกฤตกาลของซอฟต์แวร์ การแก้ปัญหาในข้อนี้ก็คือเราต้องการซอฟต์แวร์ที่มีคุณสมบัติง่ายต่อการบำรุงรักษา และมีความสามารถในการนำกลับมาใช้ได้ใหม่สูง ซึ่งเมื่อเป็นไปตามนี้แล้วจะทำให้ต้นทุนในการผลิตซอฟต์แวร์มีคาคาถูกลง ดังนั้นจึงมีผู้พิจารณาว่าจะทำให้ได้ซอฟต์แวร์ที่มีคุณสมบัติตามนี้ได้อย่างไร ปัญหาก็กลับมาอยู่ที่วิธีการออกแบบซอฟต์แวร์ เมื่อเปรียบเทียบการพัฒนาระหว่างซอฟต์แวร์ และฮาร์ดแวร์ ดังรูปที่ 2.1

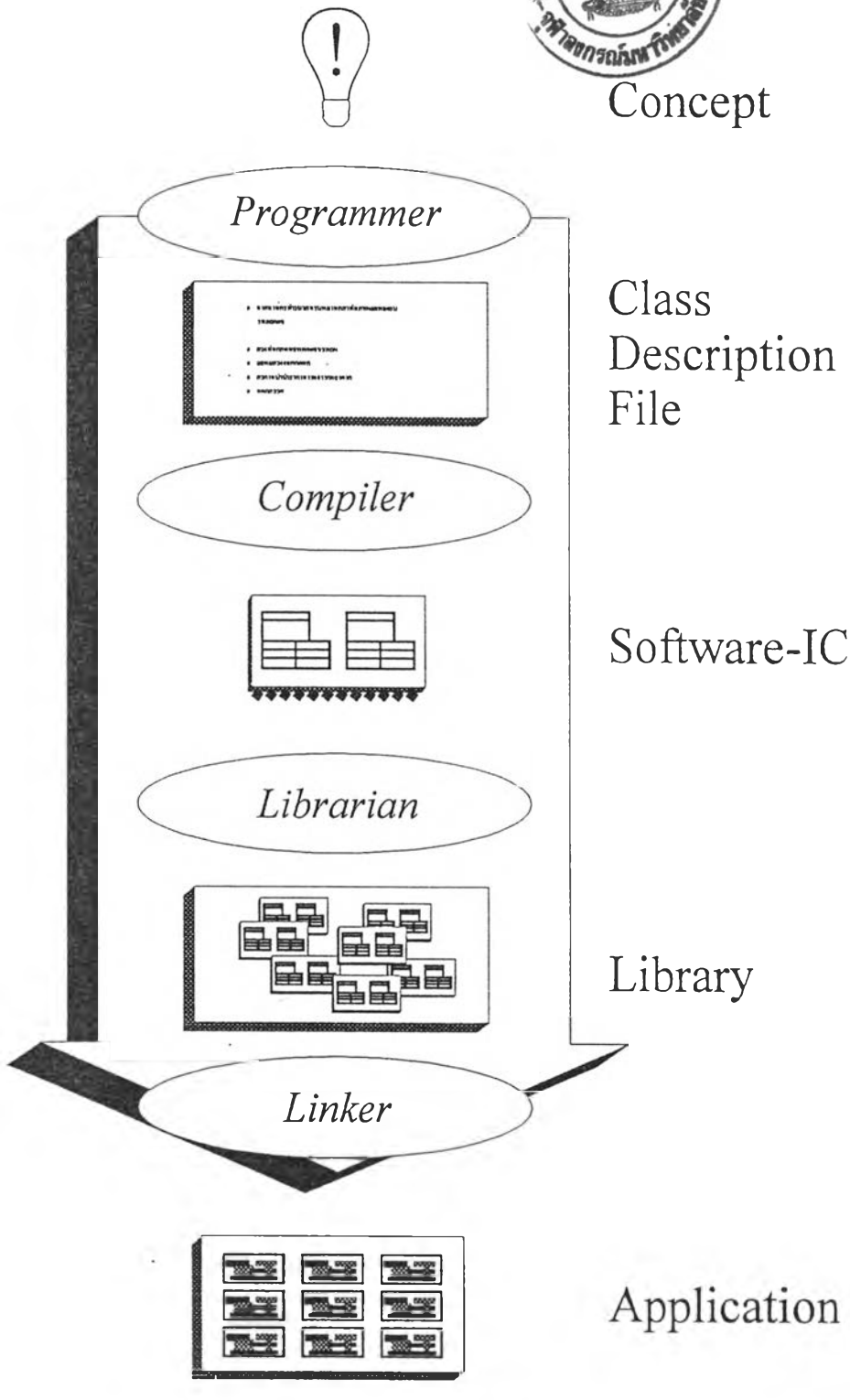


รูปที่ 2.1 เปรียบเทียบการพัฒนาของฮาร์ดแวร์กับซอฟต์แวร์

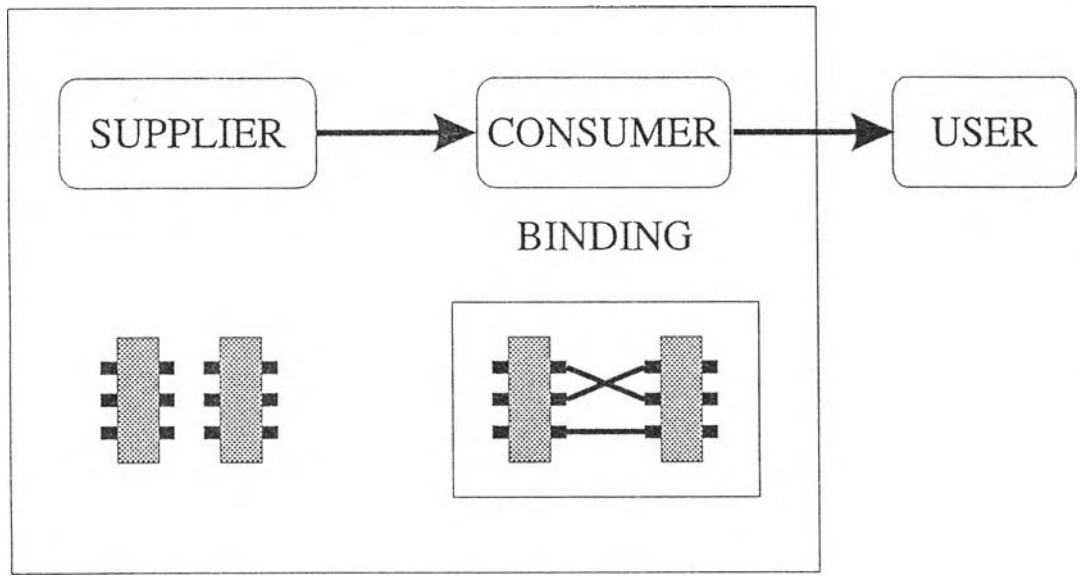
จากรูปจะเห็นว่าการพัฒนาซอฟต์แวร์นั้นไม่สามารถนำสิ่งที่สร้างขึ้นมาแล้ว กลับมาใช้ใหม่ได้ ต้องทำการสร้างใหม่ทั้งหมด จึงทำให้การเจริญเติบโตของซอฟต์แวร์นั้นเป็นไปได้ช้า แทนที่จะเติบโตแบบทวีคูณเหมือนฮาร์ดแวร์ ซึ่งสามารถนำส่วนที่ออกแบบไว้นำกลับมาใช้ได้อีกในการพัฒนาฮาร์ดแวร์รุ่นต่อไป ดังนั้นจึงมีแนวความคิดที่จะประยุกต์ให้การผลิตซอฟต์แวร์นั้นเหมือนกับการผลิตฮาร์ดแวร์จึงมีหลักการเกี่ยวกับ Software IC ขึ้น คือ ในการพัฒนานั้นเน้นที่จะให้มีการนำกลับมาใช้ใหม่เป็นเรื่องปกติ เช่นเดียวกับฮาร์ดแวร์ และ ผู้ที่นำส่วนของซอฟต์แวร์นั้นกลับมาใช้ใหม่ไม่จำเป็นที่จะต้องรู้ถึงองค์ประกอบภายใน แต่ใช้งานผ่านส่วนเชื่อมต่อ (Interface) โดยดูข้อมูลที่ผู้ผลิตบอกมา (data Book) ดังรูปที่ 2.2 โดยรวมถึงการดึงซอฟต์แวร์บางส่วนออก และนำส่วนใหม่เข้ามาใส่แทนที่ได้โดยไม่ต้องแก้ไขซอฟต์แวร์ส่วนอื่น เช่นเดียวกับการเปลี่ยนองค์ประกอบของฮาร์ดแวร์ ซึ่งส่วนนี้เป็นหลักการของการ Encapsulation เป็นการใช้งานผ่านส่วนเชื่อมต่อที่กำหนด โดยซ่อนรายละเอียดการทำงานไว้ภายใน ซึ่งจะกล่าวในรายละเอียดต่อไป หลักการของการพัฒนา Software IC สามารถแสดงรายละเอียดให้ดูได้ดังรูปที่ 2.3 และในอนาคตวิธีการพัฒนาซอฟต์แวร์ก็จะคล้ายกับการพัฒนาฮาร์ดแวร์ดังรูปที่ 2.4



รูปที่ 2.2 แสดงการใช้งานผ่านส่วนเชื่อมต่อ



รูปที่ 2.3 หลักการของการพัฒนา Software IC



รูปที่ 2.4 การพัฒนาซอฟต์แวร์ในอนาคต

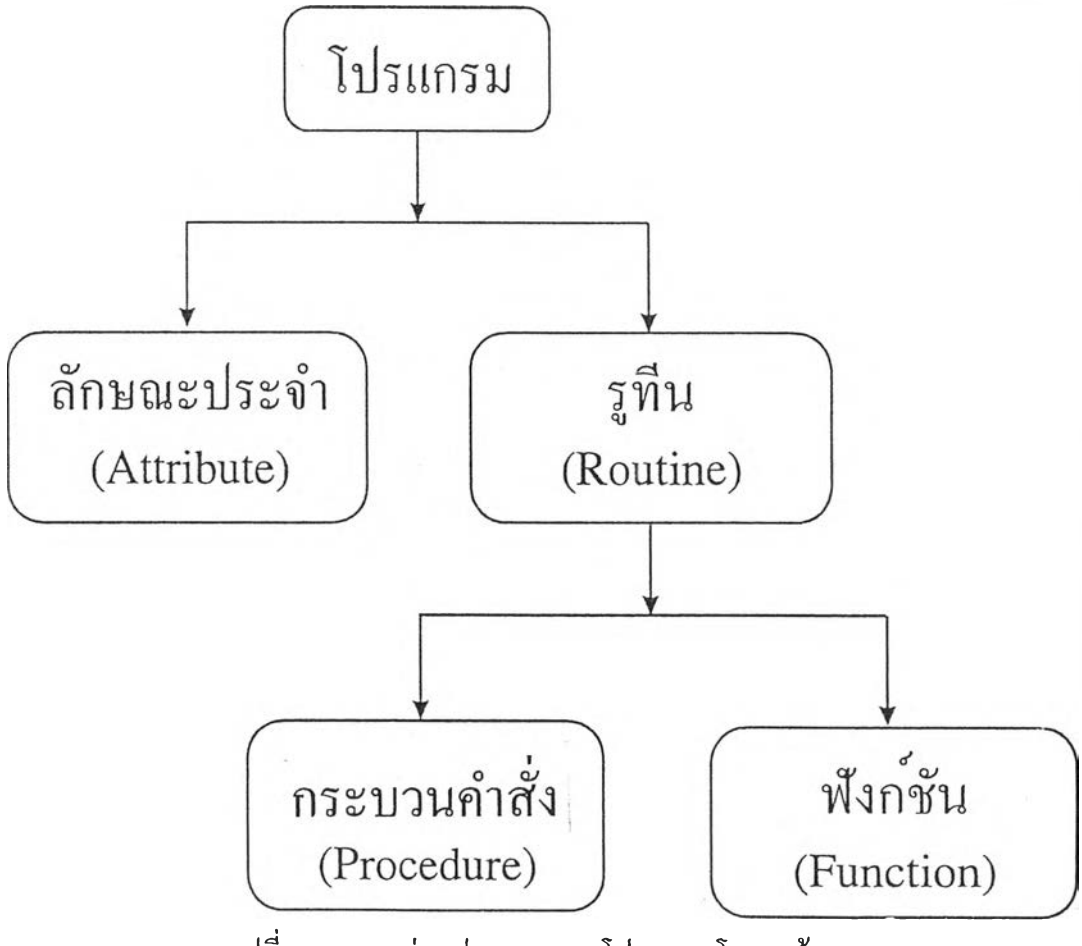
<u>วัตถุประสงค์ของการโปรแกรมเชิงวัตถุ</u>

การใช้หลักการของการโปรแกรมเชิงวัตถุ มีวัตถุประสงค์คือ

1. เพื่อให้เกิดการทำงานแบบเป็นมอดูล (Modularization) คือสามารถแยกการทำงานเป็นหน่วยจำเพาะย่อย ๆ ซึ่งแต่ละหน่วยสามารถทำงานได้โดยอิสระ แล้วจึงทำการรวมแต่ละหน่วยเข้าด้วยกันเป็นโปรแกรมใหญ่
2. ความเข้ากันได้ (Compatibility) เป็นประเด็นสำคัญ เพราะโดยทั่วไปแล้วเป็นการยากที่จะเข้าถึงข้อมูลที่มีโครงสร้างต่าง ๆ กันได้ การใช้โปรแกรมเชิงวัตถุจึงเป็นแนวทางหนึ่งในการแก้ปัญหา
3. การนำกลับมาใช้ใหม่ (Reusability) สำหรับโปรแกรมขนาดใหญ่ ซึ่งมีโครงสร้างข้อมูลขนาดใหญ่ ทำให้เกิดการกระจัดกระจายของตัวแปร และข้อมูลต่าง ๆ การนำตัวแปร และข้อมูลกลับมาใช้ใหม่ จึงเกิดความสับสน การโปรแกรมเชิงวัตถุสามารถที่จะแก้ปัญหานี้ได้ โดยที่ผู้ใช้ไม่จำเป็นต้องรู้โครงสร้างทั้งหมดของโปรแกรม
4. ความต่อเนื่อง (Continuity) ซึ่งสามารถใช้คุณสมบัติการโปรแกรมเชิงวัตถุในแง่ของการสรุป (Abstraction)

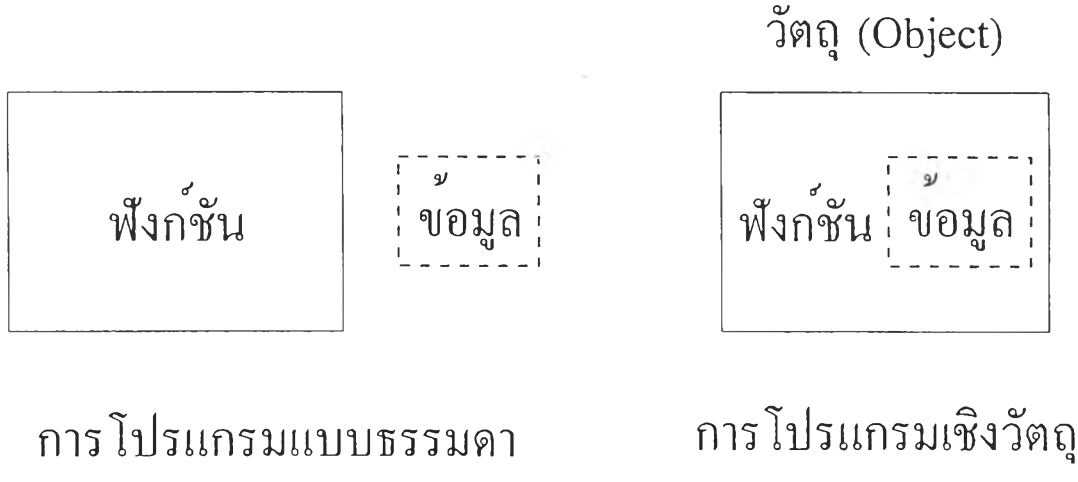
พัฒนาการแนวความคิดการโปรแกรมเชิงวัตถุ

เท่าที่ผ่านมามีการใช้วิธีการโปรแกรมแบบธรรมดา ที่นิยมใช้กันมากคือการโปรแกรมแบบกระบวนคำสั่ง (Procedural Programming) ซึ่งเน้นการออกแบบขั้นตอน และวิธีการทำงานของโปรแกรมอย่างมีระบบเป็นการโปรแกรมโครงสร้าง (Struture Programming) และการออกแบบจากบนลงล่าง (Topdown Design) ซึ่งมีจุดอ่อนเมื่อมีการเปลี่ยนการทำงานของ module หรือ data ถ้าการออกแบบดังที่กล่าวมานั้นการทำงานเกี่ยวกับข้อมูลบางส่วนของข้อมูลจะเคลื่อนที่เข้าหาฟังก์ชัน ในขณะที่บางส่วนเคลื่อนที่ออกจากฟังก์ชันตามที่เรากำหนด ภาษาที่นิยมใช้ในการโปรแกรมแบบกระบวนคำสั่งนี้ก็ได้แก่ ภาษาซี (C Language) หรือภาษาปาสคาล (Pascal Language) ซึ่งเป็นภาษาโครงสร้างในยุคแรก ๆ โปรแกรมมีขนาดเล็ก วิธีการแบบกระบวนคำสั่งก็ไม่ค่อยมีปัญหา แต่ต่อมาเมื่อโปรแกรมมีขนาดใหญ่ขึ้นทุกทีก็เริ่มมีปัญหา จึงต้องมีการจัดระบบของโคด (code) ให้ดี ดังนั้นจึงมีแนวความคิดเกี่ยวกับการโปรแกรมเชิงวัตถุ ซึ่งถูกคิดขึ้นมาเพื่อเป็นแนวทางที่จะเขียนซอฟต์แวร์ที่มีประสิทธิภาพกับปัญหาชุดหนึ่ง โดยจะรวมโครงสร้างข้อมูลและฟังก์ชันเข้าด้วยกันเป็นวัตถุ



รูปที่ 2.5 แสดงส่วนประกอบของโปรแกรมโครงสร้าง

(Object) ซึ่งเราเรียกวิธีการนี้ว่าการซ่อนข้อมูล (Data Hiding) หรือ เอ็นแคปซูเลชัน (Encapsulation) เป็นการออกแบบโปรแกรมโดยคำนึงถึงตัวข้อมูล เพื่อให้วัตถุสามารถปกครองตัวเองสามารถจะตัดสินใจได้ด้วยตัวเอง และมีความเป็นอิสระในตัวเองสูงสุดเท่าที่จะทำได้ ในรูปที่ 2.5 ได้แสดงส่วนประกอบของโปรแกรมโครงสร้าง และรูปที่ 2.6 ได้แสดงข้อแตกต่างของการโปรแกรมแบบธรรมดา และการโปรแกรมเชิงวัตถุ



รูปที่ 2.6 แสดงข้อแตกต่างของการโปรแกรมแบบธรรมดา และการโปรแกรมเชิงวัตถุ

เมื่อแนวความคิดของการโปรแกรมเชิงวัตถุได้เกิดขึ้น การจะทำให้โปรแกรมถูกพัฒนาขึ้นตามแนวความคิดนี้จำเป็นจะต้องมีภาษาคอมพิวเตอร์ ซึ่งสนับสนุนหลักการอันนี้ ภาษาการโปรแกรมเชิงวัตถุ (Object-Oriented Programming Language) เป็นภาษาซึ่งมีโครงสร้างของภาษาเอื้ออำนวย และสนับสนุนการพัฒนาโปรแกรมในแนวทางของการโปรแกรมเชิงวัตถุ ได้แก่ SMALL TALK, EIFFEL ซึ่งเป็นภาษาของการโปรแกรมเชิงวัตถุ โดยสมบูรณ์ และ OBJECT PASCAL, C++, OBJECT C ซึ่งเป็นภาษาโครงสร้าง และภาษาเชิงวัตถุผสมกันอยู่ สำหรับในงานวิจัยได้เลือกใช้ภาษา C++ เนื่องจากสามารถหาตัวแปลภาษาได้ง่าย โดยได้เลือกใช้ตัวแปลภาษาของบริษัท BORLAND

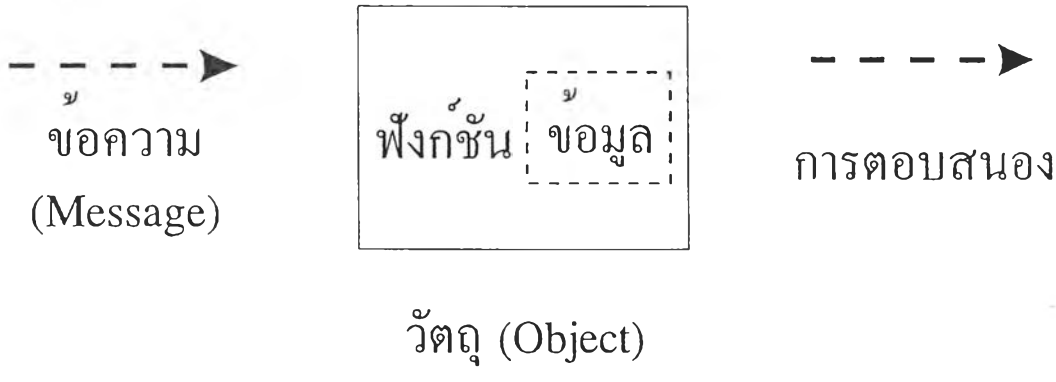
<u>หลักการโปรแกรมเชิงวัตถุ</u>

การโปรแกรมเชิงวัตถุ คือ เทคนิคในการพัฒนาโปรแกรมโดยมององค์ประกอบของข้อมูลต่างๆ ในรูปของวัตถุ (Object) และความสัมพันธ์ระหว่างวัตถุต่างๆ ซึ่งประสานงานกัน

โดยมุ่งที่จะเพิ่มประสิทธิภาพในการผลิตซอฟต์แวร์ วิธีนี้ทำให้การออกแบบ และการเขียนซอฟต์แวร์มีความใกล้ชิดกันมากขึ้นโดย

- มุ่งเน้นประมวลผลข้อมูลไม่ใช่หน้าที่การทำงาน
- จัดแบ่งข้อมูลตามแบบชนิดข้อมูลนามธรรม (Abstract Data Type)
- ใช้โปรแกรมที่มีอยู่กลับมาใช้ในโปรแกรม
- นิยามข้อมูลนามธรรมใหม่ สำหรับข้อมูล ซึ่งแยกออกจากข้อมูลอื่น

การทำงานต่างๆ จะถูกกระทำได้โดยการตอบสนองต่อ Message ต่างๆ ที่ถูกส่งให้แก่วัตถุ รูปที่ 2.7 แสดงลักษณะการกระตุ้นให้วัตถุทำงาน ดังกล่าว



รูปที่ 2.7 แสดงลักษณะการกระตุ้นให้วัตถุทำงาน

หลักการพื้นฐานสำคัญ 3 ประการ ของการโปรแกรมเชิงวัตถุ คือ Encapsulation, Inheritance และ Polymorphism แต่ก่อนที่จะกล่าวถึงหลักการ 3 ประการ ซึ่งเป็นหลักพื้นฐานของการโปรแกรมเชิงวัตถุนี้ จะอธิบายศัพท์ที่ใช้กันทั่วไป เมื่อพูดถึงเรื่อง การโปรแกรมเชิงวัตถุก่อน เพื่อเป็นพื้นฐานก่อนที่จะเข้าไปถึงเนื้อแท้ของการโปรแกรมเชิงวัตถุ

แบบชนิดข้อมูลนามธรรม (Abstract Data Type) จะรวม Type และ ชุดของการทำงาน (Operation) เข้าไว้ด้วยกัน ซึ่งการทำงานจะกำหนดคุณสมบัติของ Type นั้น

Class คือ ชนิด หรือกลุ่มของ Object ซึ่งมีคุณสมบัติร่วมกัน

การนิยาม Class (Class Definition) ซึ่งจะกำหนดการทำงานของ Abstract Data Type สามารถทำได้ด้วยการนิยามว่า จะเรียกใช้การทำงานของ Type นั้นอย่างไร การนิยาม Class ยังกำหนดโครงสร้างข้อมูลของ Type ด้วย ตามปกติแล้ว โครงสร้างข้อมูลนี้จะเข้าถึงได้เฉพาะภายใน Class ซึ่งจะเรียกว่าเป็น Type แบบ Private แต่ถ้าทุกส่วน หรือ บางส่วนของ Type นั้น เข้าถึงได้จากภายนอก Class จะเรียกว่าเป็น Type แบบ Public

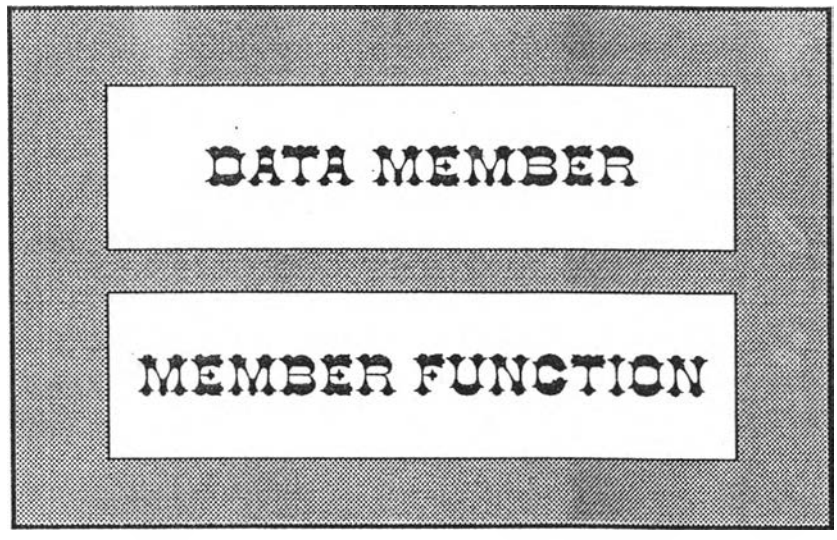
การทำงานของ Type ก็สามารถแบ่งเป็น 2 แบบ คือ การทำงานแบบ Public และ Private การทำงานแบบ Public สามารถเรียกใช้ภายนอก Class ได้ ส่วนการทำงานแบบ Private จะเรียกใช้ได้ภายใน Class เราเรียกการทำงานทั้ง 2 แบบนี้ว่า Method ซึ่งก็คือฟังก์ชันในภาษาที่ไม่ใช่ภาษา Object-Oriented นั้นเอง

Object คือ ตัวแปรของ Class โดยจะรวมข้อมูลทุกตัวไว้ใน Object เอง ซึ่งก็คือ รวมข้อมูลทุกตัว (ทั้ง Private และ Public) ที่นิยามไว้ในการนิยาม Class เพื่อให้เกิด การทำงานกับ Object การเรียกใช้ทำได้โดยการส่ง Message ให้ Object นั้น Message จะประกอบด้วยพารามิเตอร์ต่างๆ เหมือนกับ พารามิเตอร์ของฟังก์ชัน ในภาษาที่ไม่ใช่ภาษา Object-Oriented การเรียกใช้ Method โดยปกติจะทำให้ข้อมูลภายใน Object เปลี่ยนแปลงไป

วัตถุตอบสนองต่อ Message โดยกระทำการตามขั้นตอนดังนี้

- เลือกฐานการทำงานที่เหมาะสมถูกต้องกับ Message นั้น
- กระทำตามการทำงานที่เหมาะสมนั้น
- คืนการควบคุมสู่ผู้เรียก Message

การจัดลำดับชั้นของ Class มีวิธีในการจัดโดยใช้ Subclass ซึ่ง Subclass จะลอกเลียนเอาคุณสมบัติของ Parent Class มาทุกประการ และ ยังมีคุณสมบัติที่สร้างเพิ่มที่ไม่มีใน Parent Class ด้วย ต้นทุน และ ความซับซ้อนของการพัฒนาซอฟต์แวร์ จะลดลง โดยวิธีการสร้าง Subclass ขึ้นมา องค์ประกอบของแต่ละ Class นั้น ประกอบด้วย Data Member และ Member Function ดังแสดงในรูปที่ 2.8



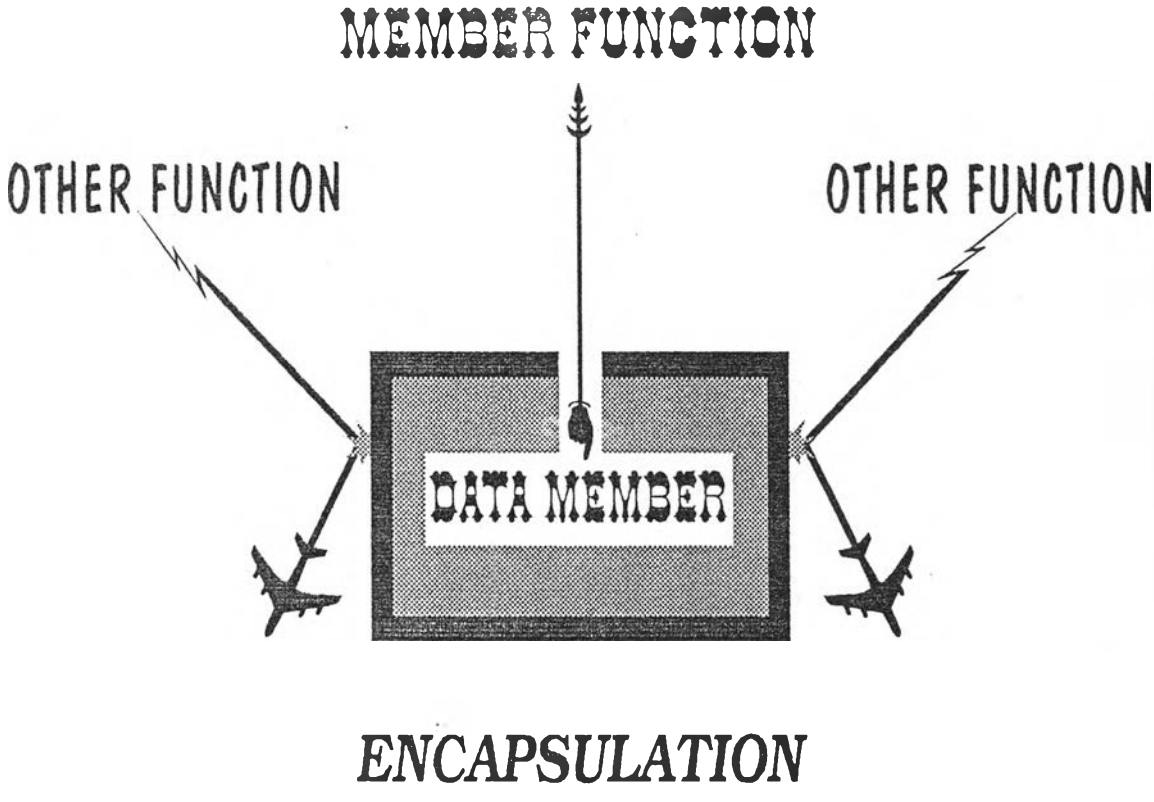
รูปที่ 2.8 แสดงองค์ประกอบของ Class

Encapsulation

Encapsulation เป็นคุณสมบัติของ Object ซึ่งมีลักษณะ ดังนี้

1. กำหนดขอบเขตที่ชัดเจนให้กับ Object
2. กำหนดส่วนอินเตอร์เฟส ซึ่งหมายความว่า Object นั้นจะติดต่อกับ Object อื่นอย่างไร
3. ส่วนอิมพลีเมนต์ ไม่สามารถเข้าถึงได้ภายนอกขอบเขตของ Class ที่ผลิต Object นั้น

หลักการของ Encapsulation เป็นการกำหนดให้ข้อมูลได้รับการป้องกันไม่ให้ ถูกเปลี่ยนแปลง มีเพียงฟังก์ชันที่กำหนดให้เข้าถึงข้อมูลได้เท่านั้น จึงจะเข้าไปประมวลผลข้อมูลที่กำหนดไว้ได้ ฟังก์ชันในที่นี้ เราเรียกว่า Member Function หรือ Method หลักการนี้อธิบายให้เข้าใจได้ดังใน รูปที่ 2.9



รูปที่ 2.9 แสดงหลักการของ Encapsulation

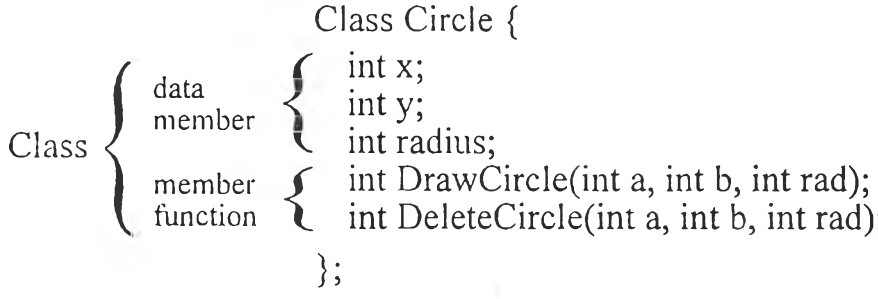
ประโยชน์ของหลักการ Encapsulation นี้ ถ้าสังเกตโปรแกรมต่อไปนี้

ส่วนที่ 1
```
Struct data_items
{
   int a;
   int b;
   int c;
};
```

ส่วนที่ 2
```
void manipulate_data(int x, int y, int z)
{
   data_items.a = data_items.a + x;
   data_items.b = data_items.b + y;
   data_items.c = data_items.c + z;
}
```

จะเห็นว่าข้อมูลใน Struct data_items สามารถถูกเข้าถึง หรือทำการเปลี่ยนแปลงโดยฟังก์ชันใดก็ได้ และ จะพบอีกว่าในเวลาแปลโปรแกรม (Compile) ส่วนที่ 1 และ ส่วนที่

2 จะแยกกันอยู่ ซึ่งทั้งสองส่วนนี้อาจจะอยู่ห่างกันมากในโปรแกรม ทำให้ยากต่อการติดตามแก้ไข หลักการของ Encapsulation จะแก้ปัญหานี้ หากเราสังเกต Class ต่อไปนี้

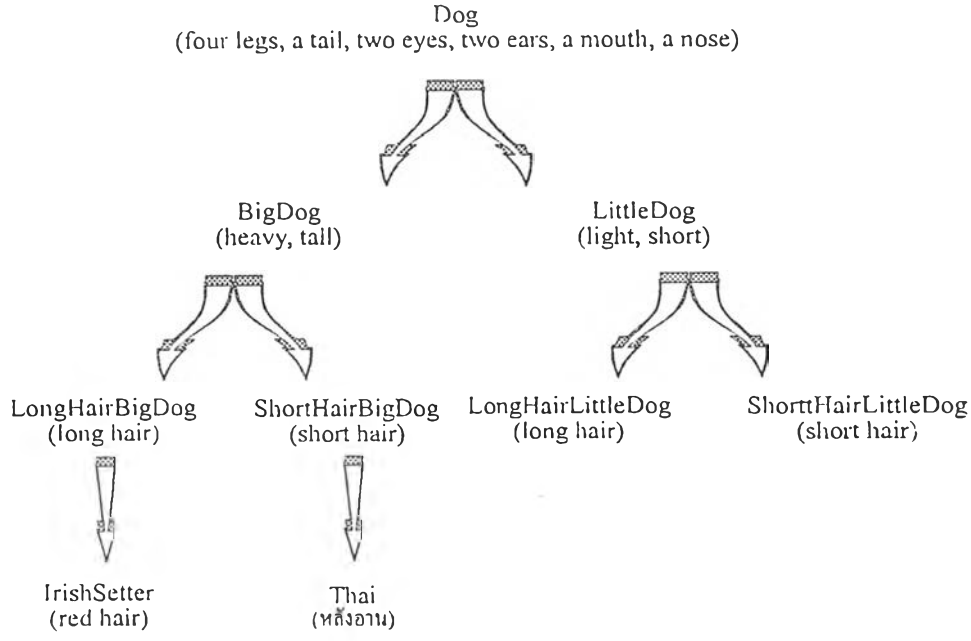


จากโครงสร้าง Class ข้างต้นอนุญาตให้รวมข้อมูล และ ฟังก์ชันไว้ใน Class และคุณสมบัติของ Object ซึ่งอยู่ใน Class ไม่สามารถเข้าถึงโดยตรงจากภายนอก Object พฤติกรรมของ Object สามารถถูกเรียก โดยส่ง Message ไปยัง Object เท่านั้น จากนิยามอันนี้ การทำงานของ Object ไม่สามารถเห็น หรือ เข้าถึงได้จากภายนอก

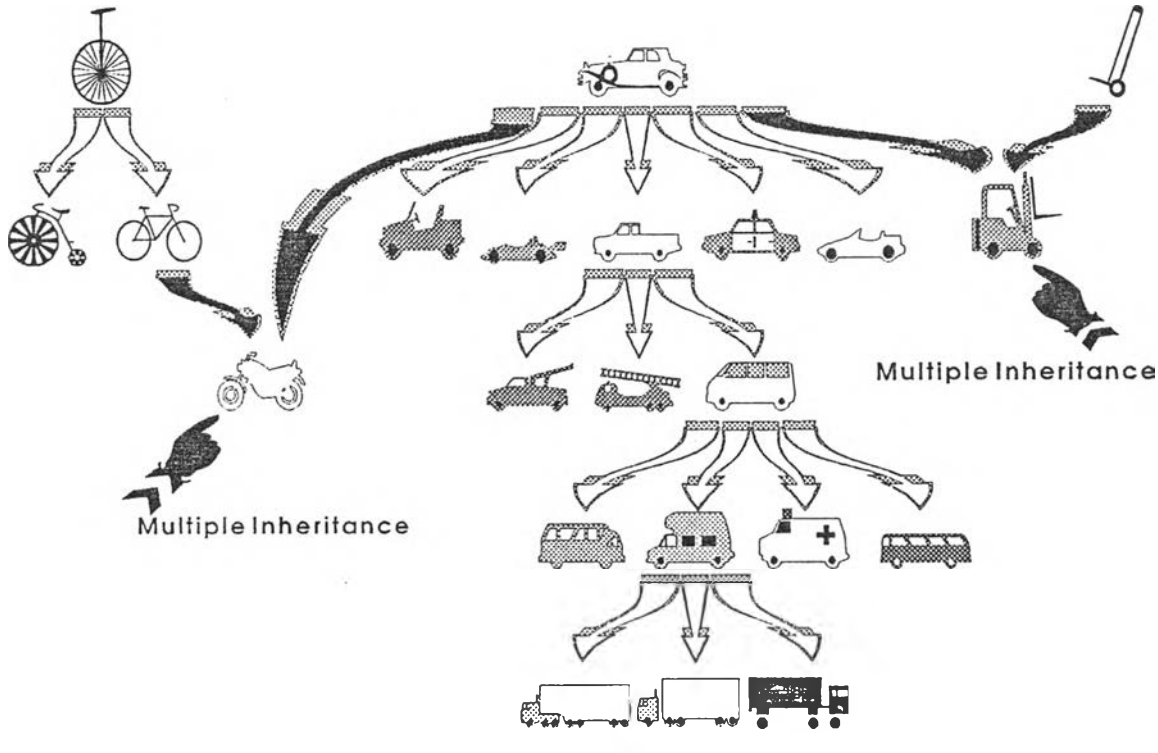
Inheritance

Inheritance เป็นการจัดลำดับชั้นของ Class ในการจัดลำดับชั้นนั้นจะมี Class จำนวนหนึ่ง ซึ่งเรียกว่า Subclass หรือ Derived Class ซึ่งเป็น Class ระดับที่ต่ำลงไปที่สืบทอดคุณลักษณะ บางประการจากระดับสูง ซึ่งเรียกว่า Base Class

เมื่อพิจารณาให้ดีจะเห็นว่าระดับที่ต่ำลงไปของการจัดลำดับขั้นจะทำให้ Class นั้นมีความ เฉพาะเจาะจงเพิ่มขึ้น ตัวอย่างแสดงภาพการสืบทอดคุณลักษณะของ Class ในชีวิตจริงแสดงใน รูปที่ 2.10 และ รูปที่ 2.11 ซึ่งแสดงการ Multiple Inheritance เป็นการสืบสายพันธ์ของ Class ซึ่งต่างสายพันธ์กัน โดยมีคุณลักษณะของทั้งสองสายพันธ์รวมกัน



รูปที่ 2.10 แสดงการ Inheritance ของ Class



รูปที่ 2.11 แสดงการ Inheritance ของ Class ที่ต่างสายพันธุ์กัน

Polymorphism

Polymorphism ทำให้ Object แต่ละ Object ตอบสนองต่อ Message อันเดียวกันในลักษณะที่เหมาะสมกับ Class ของตัวเอง ตัวอย่างเช่น Method Print ถูกนิยาม โดยให้แสดงค่าของข้อมูลบางตัวของ Object ทุก Object ของ Class และ Subclass ที่ทำให้ Object นั้นรู้จัก Message Print และ Object ของ Class ต่างกันจะตอบสนองต่อ Message นี้ต่างกันไป ความสามารถในการใช้ Message ที่เหมือนกันส่งให้ Object ต่างชนิดกันได้ จะมีความคล้ายคลึงกับวิธีการคิดของมนุษย์ คือ เป็นไปตามธรรมชาติ ถ้าต้องการแสดงค่าของตัวเลข Integer, ตัวเลข Floating Point, ตัวอักษร, ข้อความ และ ระเบียนของข้อมูล จะทำได้โดยมีวิธีการเรียกเหมือนกัน

ตัวอย่าง Function Prototypes ซึ่งนำหลักการของ Polymorphism มาใช้

```
int         square(int value);
float       square(float value);
double      square(double value);
```

จากตัวอย่าง Function Prototypes ข้างต้น เมื่อเราเรียกใช้ฟังก์ชัน square ตัวแปลโปรแกรม (Compiler) จะรู้ โดยอัตโนมัติว่าเราต้องการเรียกใช้ฟังก์ชัน square ฟังก์ชันใด ฟังก์ชันเหล่านี้มีชื่อเรียกอีกชื่อหนึ่งว่า Virtual Function ใช้สนับสนุนหลักการของ Polymorphism ซึ่งเกี่ยวข้องกับโครงสร้างต้นไม้ของ Parent Class และ Subclass ของมัน แต่ละ Subclass สามารถรับ message เดียวกัน โดยการทำงานจะขึ้นกับรายละเอียดของ Subclass นั้น ทำให้เราไม่ต้องกังวลต่อการทำงานที่เกิดขึ้นว่าจะถูกต้องหรือไม่

ดังนั้น Polymorphism คือวิธีการที่การกระทำอันหนึ่งถูกใช้ร่วมกันในกลุ่มของ Object หลายชนิด โดย Object แต่ละตัวจะกระทำไปตามวิธีการที่เหมาะสมกับ Object นั้น

สรุป การโปรแกรมเชิงวัตถุ ถูกออกแบบมา เพื่อช่วยลดความซับซ้อนของโปรแกรมขนาดใหญ่ โดยใช้หลักการของ Encapsulation, Inheritance และ Polymorphism ทำให้ตัวโปรแกรมที่ถูกพัฒนาขึ้นมา มีคุณสมบัติเป็นมอดูลาร์ (modular) มากขึ้น ดังนั้นจึงสามารถนำกลับมาใช้ใหม่ (reusable) และ สามารถบำรุงรักษา (maintainable) ได้ง่ายขึ้น