บทที่ 4

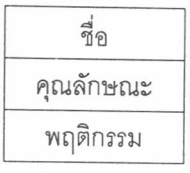
การเขียนโปรแกรมเชิงวัตถุ (Object-oriented programming)

วิทยานิพนธ์นี้เลือกภาษา C++ ซึ่งเป็นภาษาเชิงวัตถุสำหรับพัฒนาโปรแกรม ดังนั้นการออกแบบโปรแกรมจะต้องใช้หลักการเชิงวัตถุมาเป็นสำคัญ เพื่อให้เข้าใจความหมายของการพัฒนาโปรแกรมเชิงวัตถุ บทนี้จะอธิบายถึงความเป็นวัตถุและหลักการเชิงวัตถุสำคัญที่นำมาใช้ออกแบบโปรแกรม

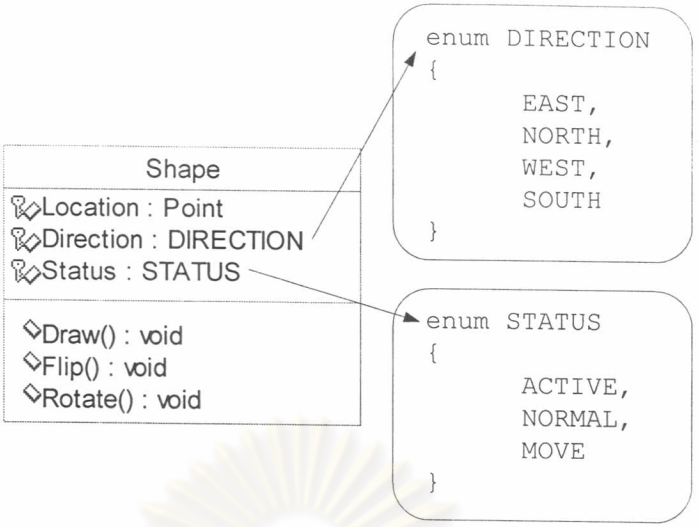
4.1 คลาส

ในการพัฒนาโปรแกรมด้วยภาษา C++ ผู้พัฒนาสามารถกำหนดชนิดของตัวแปรขึ้นใหม่ได้ ชนิดของตัวแปรที่กำหนดขึ้นมาใหม่นี้สามารถรวมเอาข้อมูล (data) หรือคุณลักษณะ (Attribute) และฟังก์ชัน (function) หรือพฤติกรรม (behaviour) เข้าไว้ด้วยกัน ทำให้ชนิดของตัวแปรที่กำหนดขึ้นใหม่นี้มีหน้าที่การทำงานเป็นของตัวเอง ทำให้ผู้พัฒนาสามารถแบ่งโปรแกรมที่พัฒนาขึ้นออกเป็นส่วนย่อยๆ ได้อย่างชัดเจน และง่ายต่อดูแลและปรับปรุงโปรแกรมให้มีประสิทธิภาพที่ดีขึ้น

คลาสคือการกำหนดชนิดของตัวแปรขึ้นใหม่ ผู้พัฒนาสามารถประกาศตัวแปรของคลาสและเรียกใช้ได้เหมือนตัวแปรธรรมดาทั่วไป คลาสจึงเปรียบเสมือนเป็นพิมพ์เขียวของกลุ่มข้อมูลที่เกี่ยวข้องกัน วัตถุ (Object) ก็คือตัวแปรของคลาส หรือจะเรียกว่าอินสแตนซ์ (Instance) ของคลาส ในการอธิบายการออกแบบของโปรแกรมจะใช้สัญลักษณ์ของคลาสดังแสดงในรูปที่ 4.1 ซึ่งประกอบไปด้วย 3 ส่วนได้แก่ชื่อ คุณลักษณะ และพฤติกรรม ตัวอย่างของการประยุกต์ใช้สัญลักษณ์คลาสแสดงได้ดังรูปที่ 4.2 โดยคลาสมีชื่อว่า Shape เป็นคลาสสำหรับจัดการกับการวาดรูปพื้นฐานของโปรแกรม คุณลักษณะของคลาสนี้จะมี Location เป็นตัวแปรเก็บตำแหน่งที่ใช้ในการวาด Direction เป็นตัวแปรที่บอกทิศทางของการวาด และ Status เป็นตัวแปรเก็บค่าสถานะของการวาด ส่วนพฤติกรรมของคลาสจะมี Draw เป็นฟังก์ชันที่ใช้สำหรับการวาดรูป Flip เป็นฟังก์ชันที่ใช้สำหรับกลับด้านของรูปที่จะวาดและ Rotate เป็นฟังก์ชันที่ใช้สำหรับการหมุนของรูป



รูปที่ 4.1 สัญลักษณ์ของคลาส



รูปที่ 4.2 คลาส Shape

ในภาษา C++ เราสามารถกำหนดค่าของตัวแปรให้แคบลงได้ ยกตัวอย่างจากรูปที่ 4.2 กล่าวคือตัวแปร Direction นั้นจะมีค่าได้เพียง 4 ค่าเท่านั้นคือ EAST, NORTH, WEST และ SOUTH ซึ่งโดยปกติจะแทนค่า EAST เท่ากับ 0 NORTH เท่ากับ 1 WEST เท่ากับ 2 และ SOUTH เท่ากับ 3 เช่นเดียวกันกับตัวแปร Status ที่จะมีค่าได้เพียง 3 ค่าเท่านั้นคือ ACTIVE, NORMAL และ MOVE ซึ่งโดยปกติจะแทนค่า ACTIVE เท่ากับ 0 NORMAL เท่ากับ 1 และ MOVE เท่ากับ 2

4.2 แนวคิดเชิงวัตถุ

แนวคิดเชิงวัตถุเป็นแนวคิดสำหรับการออกแบบโปรแกรมอีกแนวคิดหนึ่ง ซึ่งแตกต่างจากการออกแบบโปรแกรมแบบกระบวนคำสั่ง (procedural programming design) เช่นในภาษา C กล่าวคือแนวคิดเชิงวัตถุนี้จะแบ่งโปรแกรมออกเป็นส่วนย่อย ซึ่งแต่ละส่วนย่อยนี้จะมีหน้าที่การทำงานที่ชัดเจน ผู้พัฒนาจะควบคุมการติดต่อกันระหว่างแต่ละส่วนย่อยเพื่อให้โปรแกรมทำงานตามต้องการ

หลักการของแนวคิดเชิงวัตถุเป็นหลักการที่ช่วยให้แบ่งโปรแกรมออกเป็นส่วนย่อยได้ง่ายขึ้น ส่วนย่อยที่กล่าวถึงนี้ก็คือคลาสนั่นเอง หลักการที่สำคัญของแนวคิดเชิงวัตถุได้แก่การสืบทอด (Inheritance) การพ้องรูป (Polymorphism) และการหุ้มห่อ (Encapsulation) ซึ่งอธิบายได้ดังนี้

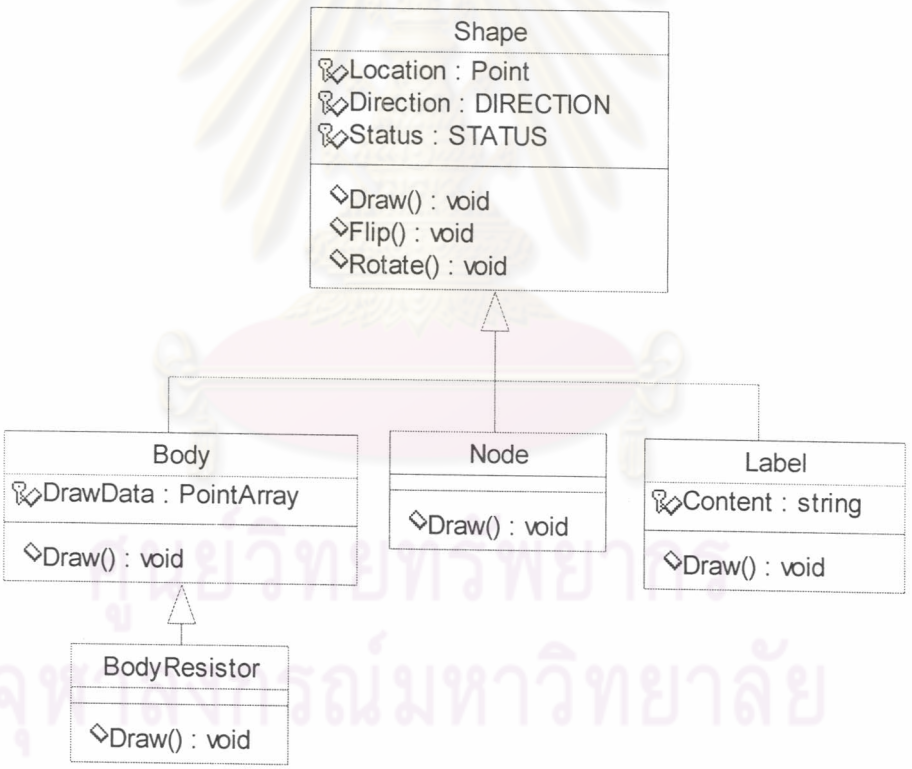
4.2.1 การสืบทอด (Inheritance)

การสืบทอดเป็นการเป็นการสร้างลำดับความสำคัญของการเกี่ยวข้องกันระหว่างคลาส ซึ่งเป็นการช่วยนำรหัส (code) ที่สร้างไว้แล้วกลับมาใช้ใหม่ คลาสที่มีการสืบทอดมาจากอีกคลาสหนึ่งจะยังคงคุณสมบัติเหมือนคลาสต้นฉบับ ผู้พัฒนาสามารถทำการแก้ไขหรือเพิ่มความสามารถ

ให้กับคลาสที่สืบทอดมาใหม่นี้ โดยที่ไม่กระทบถึงคลาสต้นฉบับเดิม เพื่อให้เข้าใจความหมายได้ดีขึ้นเราจะเรียกคลาสต้นฉบับว่าเป็นคลาสแม่ และเรียกคลาสที่สืบทอดมาว่าเป็นคลาสลูก สัญลักษณ์ที่ใช้แทนการสืบทอดแสดงได้ดังรูปที่ 4.3

รูปที่ 4.3 สัญลักษณ์แทนการสืบทอด

เนื่องจากคลาสลูกยังคงคุณสมบัติเช่นเดียวกันกับคลาสแม่ ดังนั้นเราจะเรียกได้ว่าคลาสลูกนี้เป็นชนิดเดียวกันกับคลาสแม่ ยกตัวอย่างดังรูปที่ 4.4 จะเห็นว่าคลาส Body, Node และ Label ทำการสืบทอดมาจากคลาส Shape ดังนั้นจะสามารถเรียกได้ว่าวัตถุจากคลาส Body, Node และ Label ก็เป็นวัตถุจากคลาส Shape ด้วยเหมือนกัน เช่นเดียวกันกับคลาส BodyResistor ที่วัตถุจากคลาสนี้ก็เป็นวัตถุจากคลาส Body และคลาส Shape เหมือนกัน



รูปที่ 4.4 ลำดับชั้นของคลาส Shape

จากรูปที่ 4.4 จะเห็นว่าคลาส Body, Node, Label และ BodyResistor ได้นิยามฟังก์ชัน Draw ใหม่ ดังนั้นวัตถุจากคลาส Body, Node, Label และ BodyResistor จะมีการวาดรูปที่แตกต่างจากคลาส Shape แต่การทำงานอื่นยังคงทำงานเหมือนคลาส Shape ซึ่งเป็นข้อดีของการ

สืบทอด กล่าวคือผู้พัฒนาสามารถที่จะนำเอาการทำงานพื้นฐานไว้ที่คลาสแม่ เมื่อต้องการเปลี่ยนแปลงการทำงาน ผู้พัฒนาเพียงสร้างคลาสใหม่ที่สืบทอดจากคลาสแม่แล้วนิยามส่วนที่ต้องการเปลี่ยนแปลงขึ้นใหม่ ซึ่งจะเป็นการช่วยลดเวลาในการพัฒนาโปรแกรมโดยรวมได้

4.2.2 การพ้องรูป (Polymorphism)

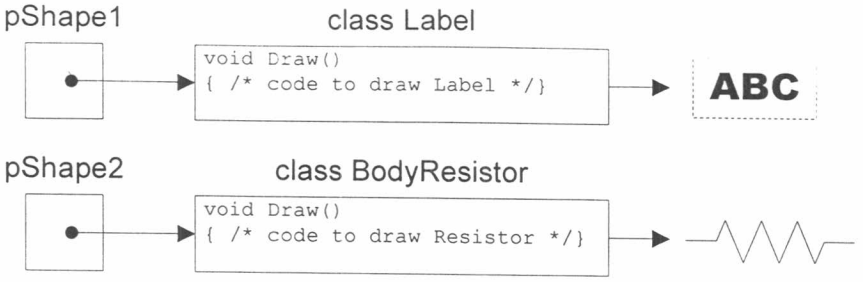
ในภาษา C++ จะมีการใช้ตัวชี้ (pointer) ซึ่งเป็นตัวแปรที่เก็บค่าแอดเดรส (address) ของข้อมูล โดยปกติแล้วตัวแปรจะเก็บค่าเฉพาะของข้อมูลที่ระบุไว้ แต่ตัวชี้จะเก็บค่าแอดเดรสของตัวแปรที่เก็บค่าเฉพาะนั้น กล่าวคือตัวแปรจะเป็นการเข้าถึงข้อมูลทางตรง แต่ตัวชี้จะเป็นการเข้าถึงข้อมูลทางอ้อม

จากหัวข้อ 4.2.1 ได้อธิบายว่าวัตถุจากคลาสลูกถือว่าเป็นชนิดเดียวกันกับคลาสแม่ ในภาษา C++ จึงอนุญาตให้ตัวแปรที่กำหนดให้เป็นตัวชี้ไปยังข้อมูลของคลาสแม่ สามารถชี้ไปยังข้อมูลที่เป็นวัตถุจากคลาสลูกได้ ยกตัวอย่างจากโครงสร้างคลาสในรูปที่ 4.4 เราสามารถใช้รหัสดังนี้

```
Shape *pShape1, *pShape2;
pShape1 = new Label();
pShape2 = new BodyResistor();
```

จากรหัสนี้เราได้ประกาศตัวแปร pShape1 และ pShape2 ให้เป็นตัวชี้ไปยังคลาส Shape จากนั้นจึงจองพื้นที่หน่วยความจำสำหรับคลาส Label แล้วให้ตัวชี้ pShape1 อ้างอิงถึงวัตถุนี้ และทำการจองพื้นที่หน่วยความจำสำหรับคลาส BodyResistor แล้วให้ตัวชี้ pShape2 อ้างอิงถึงวัตถุนี้ ในรูปที่ 4.4 จะเห็นว่าคลาส Label และคลาส BodyResistor มีการนิยามการทำงานของฟังก์ชัน Draw() ใหม่ ดังนั้นหากเราใช้ตัวชี้ pShape1 และ pShape2 ให้เรียกใช้ฟังก์ชัน Draw() เนื่องจากเป็นพื้นที่หน่วยความจำของข้อมูลในคลาส Label และคลาส BodyResistor ตามลำดับ ดังนั้นเมื่อ pShape1 เรียกใช้ฟังก์ชัน Draw() ก็จะเป็นการวาดรูปของคลาส Label และเมื่อ pShape2 เรียกใช้ฟังก์ชัน Draw() ก็จะเป็นการวาดรูปของคลาส BodyResistor เช่นกัน

หลักการของการพ้องรูปก็คือการนิยามพฤติกรรมของคลาสลูกที่แตกต่างจากพฤติกรรมของคลาสแม่ จากนั้นในการทำงานจริงผู้ใช้จะสามารถเลือกได้ว่าต้องการการทำงานจากคลาสใด หลักการนี้มีประโยชน์ในการพัฒนาโปรแกรมเป็นอย่างมาก เนื่องจากในขณะพัฒนาโปรแกรมนั้นผู้พัฒนาจะไม่สามารถทราบได้ว่าผู้ใช้ต้องการการทำงานแบบใด แต่ผู้พัฒนาสามารถคาดเดาได้ว่าการทำงานที่ผู้ใช้ต้องการนั้นจะอยู่ในรูปแบบหนึ่งของคลาสแม่ที่ผู้พัฒนากำหนดไว้ ผู้พัฒนาเพียงประกาศตัวชี้ที่ชี้ไปยังข้อมูลของคลาสแม่ แล้วผู้ใช้จะเป็นคนเลือกการทำงานที่ตรงกับความต้องการเอง รูปแบบการทำงานของหลักการการพ้องรูปแสดงได้ดังรูปที่ 4.5



รูปที่ 4.5 หลักการการพ้องรูป

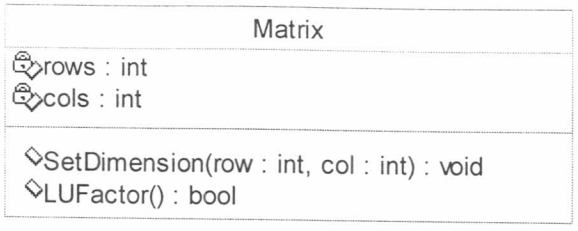
4.2.3 การหุ้มห่อ (Encapsulation)

ในโลกแห่งความเป็นจริง เราสามารถมองสิ่งของต่างๆ เป็นวัตถุได้ และการใช้งานสิ่งของนั้นเปรียบเสมือนเป็นการส่งคำร้อง (Message) เพื่อให้วัตถุนั้นทำงานตามที่ต้องการ ยกตัวอย่างเช่น การใช้งานรถยนต์ เราสามารถที่จะเร่งเครื่องด้วยการเหยียบคันเร่ง หยุดรถด้วยการเหยียบเบรค หรือแม้กระทั่งฟังเพลงในรถด้วยการเปิดวิทยุ การใช้งานเหล่านี้เราไม่จำเป็นต้องรู้ว่าคันเร่งทำงานอย่างไร เบรคทำงานอย่างไร หรือวิทยุทำงานอย่างไร สิ่งที่ต้องการคือผลลัพธ์ที่ได้เท่านั้น กล่าวได้ว่ารายละเอียดของการทำงานเหล่านี้ถูกซ่อนจากมุมมองของผู้ใช้

แนวคิดเชิงวัตถุได้นำหลักการนี้มาใช้โดยเรียกว่าการหุ้มห่อ ซึ่งถือได้ว่าเป็นหลักการสำคัญของแนวคิดเชิงวัตถุ การหุ้มห่อเป็นการซ่อนข้อมูลการทำงานที่ผู้ใช้ไม่จำเป็นต้องรู้ สิ่งที่ผู้ใช้จะรู้มีเพียงว่าคลาสนี้ทำงานอย่างไร และให้ผลอย่างไร ในภาษา C++ จะมีคำหลักที่ใช้กำหนดระดับขั้นของการหุ้มห่อได้แก่ public, private และ protected รายละเอียดของคำหลักมีดังนี้

- public: ข้อมูลที่อยู่ภายใต้ส่วน public สามารถเข้าถึงได้โดยตรงจากตัววัตถุ ผู้ใช้วัตถุหรือคลาสลูกอื่นๆ
- private: ข้อมูลที่อยู่ภายใต้ส่วน private จะถูกใช้ได้เพียงตัววัตถุเท่านั้น ผู้ใช้วัตถุหรือคลาสลูกจะไม่สามารถยุ่งเกี่ยวกับข้อมูลในส่วนนี้ได้
- protected: ข้อมูลที่อยู่ภายใต้ส่วน protected จะถูกใช้ได้จากตัววัตถุและคลาสลูกเท่านั้น ผู้ใช้ไม่สามารถยุ่งเกี่ยวกับข้อมูลในส่วนนี้ได้

การกำหนดระดับขั้นของการหุ้มห่อ นอกเหนือจากเป็นการซ่อนข้อมูลที่ผู้ใช้ไม่จำเป็นต้องรู้แล้ว ยังเป็นการป้องกันไม่ให้มีการแก้ไขข้อมูลที่ไม่ถูกต้องด้วย

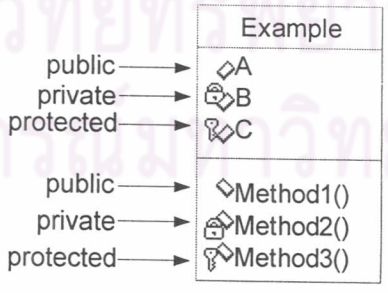


รูปที่ 4.6 คลาส Matrix

ยกตัวอย่างคลาส Matrix ในรูปที่ 4.6 คลาสนี้มีหน้าที่เก็บข้อมูลในรูปแบบเมทริกซ์ และสามารถแยกตัวประกอบแอล-ยูได้ โดยจะมีตัวแปร rows เป็นตัวแปรที่บอกจำนวนแถวและตัวแปร cols ที่บอกจำนวนคอลัมน์ในเมทริกซ์ โดยปกติจะให้ผู้ใช้กำหนดขนาดของเมทริกซ์ผ่านฟังก์ชัน SetDimension(int row, int col) ซึ่งจะทำการจองพื้นที่หน่วยความจำให้ตรงกับที่ผู้ใช้ต้องการ และมีฟังก์ชัน LUFactor ที่ใช้สำหรับแยกตัวประกอบแอล-ยู จะเห็นว่าคลาสนี้ไม่อนุญาตให้เข้าถึงข้อมูลจำนวนแถวและจำนวนคอลัมน์ได้โดยตรง เนื่องจากการแยกตัวประกอบแอล-ยูนั้นจะมีการยุ่งเกี่ยวกับข้อมูลในส่วนนี้ ถ้าให้ผู้ใช้สามารถแก้ไขข้อมูลจำนวนแถวและจำนวนคอลัมน์เองได้โดยไม่ผ่านฟังก์ชัน SetDimension(int row, int col) แล้ว ผู้ใช้อาจกำหนดจำนวนแถวและคอลัมน์ให้เกินจากพื้นที่หน่วยความจำที่จองไว้ได้ ซึ่งจะทำให้เกิดข้อผิดพลาดขณะทำงาน ยกตัวอย่างรหัสดังนี้

```
Matrix matrix;
matrix.SetDimension(3,3); // Allocate memory for 9 variable
matrix.rows = 4;    // Let user change number of row in Matrix
matrix.cols = 4;    // Let user change number of column in Matrix
matrix.LUFactor(); // Error! access to unallocate memory
                   // (variable 10 - 16)
```

สัญลักษณ์ที่บอกระดับขั้นของการหุ้มห่อแสดงได้ดังรูปที่ 4.7



รูปที่ 4.7 สัญลักษณ์แสดงระดับขั้นของการหุ้มห่อ

4.3 สรุปท้ายบท

ในบทนี้ได้นำเสนอหลักการของการพัฒนาโปรแกรมเชิงวัตถุ และหลักสำคัญของการเขียนโปรแกรมเชิงวัตถุ ในการนำหลักเหล่านี้มาประยุกต์ใช้จะช่วยทำให้ง่ายต่อการพัฒนาในระยะยาว ผู้พัฒนาสามารถที่จะเพิ่มความสามารถของโปรแกรมโดยที่ไม่กระทบถึงส่วนเดิมของโปรแกรมที่มีอยู่ รายละเอียดของการประยุกต์ใช้ในวิทยานิพนธ์จะนำเสนอในบทต่อไป