บทที่ 1
ความรู้ทั่วไปเกี่ยวกับแนวคิดเชิงวัตถุ

ในบทนี้จะกล่าวถึงความรู้ทั่วไปเกี่ยวกับแนวคิดเชิงวัตถุ ได้แก่ องค์ประกอบของระบบคอมพิวเตอร์ ภาษาคอมพิวเตอร์ เหตุผลในการพัฒนาแนวคิดเชิงวัตถุ แนวคิดเชิงวัตถุขั้นพื้นฐาน รวมทั้งความแตกต่างระหว่างการพัฒนาโปรแกรมเชิงวัตถุและการพัฒนาโปรแกรมแบบเดิม

1.1 องค์ประกอบของระบบคอมพิวเตอร์

คอมพิวเตอร์ หมายถึง อุปกรณ์อิเล็กทรอนิกส์ที่สามารถแปลงคำสั่งและดำเนินการต่าง ๆ ตามที่ได้รับคำสั่ง เช่น การส่งข้อมูลเข้าสู่ระบบ การประมวลผลด้านการคำนวณ การเปรียบเทียบด้านตรรกะ และการแสดงผลลัพธ์ ทั้งนี้คำสั่งต่าง ๆ จะอยู่ในรูปของโปรแกรม นั่นคือคอมพิวเตอร์จะต้องได้รับโปรแกรมการสั่งงานก่อนจึงจะเริ่มปฏิบัติงานได้ องค์ประกอบพื้นฐานของระบบคอมพิวเตอร์ประกอบด้วย ฮาร์ดแวร์ ซอฟต์แวร์ และบุคลากร

1. ฮาร์ดแวร์ (Hardware) คือ ส่วนประกอบที่เป็นตัวเครื่องคอมพิวเตอร์ที่ประกอบด้วย หน่วยรับข้อมูล (Input Unit) หน่วยประมวลผลกลาง (Central Processing Unit : CPU) หน่วยแสดงผล (Output Unit) และหน่วยความจำ (Memory Unit)

2. ซอฟต์แวร์ (Software) หรือ โปรแกรม (Program) คือ ชุดคำสั่งที่สั่งให้เครื่องคอมพิวเตอร์ทำงานตามขั้นตอนที่กำหนดไว้ เพื่อให้ได้ผลลัพธ์ตามที่ผู้ใช้คอมพิวเตอร์ต้องการ ซอฟต์แวร์จะทำหน้าที่จัดการ ควบคุมการประมวลผลของคอมพิวเตอร์ตั้งแต่เปิดเครื่องจนกระทั่งปิดเครื่อง เครื่องคอมพิวเตอร์จำเป็นต้องมีซอฟต์แวร์เป็นตัวควบคุมการทำงาน ซอฟต์แวร์เป็นสิ่งที่มนุษย์สร้างขึ้นเพื่อการสั่งงานเครื่องคอมพิวเตอร์ โดยซอฟต์แวร์แบ่งได้ 3 ประเภท คือ ซอฟต์แวร์ระบบ (System Software) ซอฟต์แวร์ประยุกต์ (Application Software) และซอฟต์แวร์สำเร็จรูป (Package Software)

3. บุคลากร (People) บุคลากรจะเป็นผู้จัดการและควบคุมระบบคอมพิวเตอร์ให้สามารถปฏิบัติงานได้อย่างราบรื่น คอยแก้ปัญหาที่เกิดขึ้นกับระบบคอมพิวเตอร์ พัฒนาโปรแกรมประยุกต์ต่าง ๆ รวมไปถึงการใช้งานโปรแกรมประยุกต์ที่ถูกพัฒนาขึ้น

1.2 ภาษาคอมพิวเตอร์

โปรแกรมคอมพิวเตอร์คือ ชุดคำสั่งที่นักพัฒนาโปรแกรม (Programmer) พัฒนาขึ้นเพื่อสั่งให้เครื่องคอมพิวเตอร์ทำงานตามที่ต้องการ โดยการเขียนโปรแกรมคอมพิวเตอร์จะต้องใช้

ภาษาคอมพิวเตอร์ที่เหมาะสมกับงานที่ต้องการนำคอมพิวเตอร์ไปช่วยแก้ปัญหา ซึ่งแต่ละภาษาจะมีความสามารถในการรองรับการเขียนโปรแกรมควบคุมการทำงานของคอมพิวเตอร์ที่มีลักษณะแตกต่างกัน ภาษาคอมพิวเตอร์แบ่งออกเป็น 3 ประเภทคือ

1. ภาษาเครื่อง (Machine Language) เป็นภาษาที่เครื่องคอมพิวเตอร์สามารถเข้าใจได้ ประกอบด้วยคำสั่งที่เป็นชุดของเลขฐานสอง เช่น 01011010 ซึ่งจะถูกกำหนดโดยฮาร์ดแวร์ของเครื่องคอมพิวเตอร์ ทั้งนี้ภาษาเครื่องจะขึ้นอยู่กับชนิดของหน่วยประมวลผลกลางของเครื่องคอมพิวเตอร์แต่ละเครื่อง โดยปกติแล้วนักพัฒนาโปรแกรมไม่สามารถพัฒนาโปรแกรมโดยเขียนด้วยภาษาเครื่องโดยตรง เนื่องจากเป็นภาษาที่ใช้เลขฐานสองซึ่งไม่ใช่ชุดคำสั่งที่มนุษย์จะสามารถเข้าใจได้โดยง่าย

2. ภาษาแอสเซมบลี (Assembly Language) เป็นภาษาคอมพิวเตอร์ที่แทนชุดคำสั่งเลขฐานสองด้วยคำ สัญลักษณ์ที่เป็นภาษาอังกฤษ เช่น 10110011 แทนด้วย MOV เป็นต้น ทำให้นักพัฒนาโปรแกรมเขียนและเข้าใจโปรแกรมได้ง่ายขึ้น การทำงานของภาษาแอสเซมบลีจะต้องมีการแปลภาษาแอสเซมบลีให้เป็นภาษาเครื่องก่อนจึงจะทำงานได้ แม้ภาษาแอสเซมบลีจะมีชุดคำสั่งที่เป็นคำภาษาอังกฤษ แต่นักพัฒนาโปรแกรมภาษาแอสเซมบลีจะต้องมีความเข้าใจโครงสร้างฮาร์ดแวร์ของระบบคอมพิวเตอร์จึงจะสามารถเขียนโปรแกรมได้ จึงทำให้ภาษาแอสเซมบลีไม่เป็นที่นิยมใช้

3. ภาษาระดับสูง (High-level Language) การพัฒนาโปรแกรมโดยทั่วไปจะใช้โปรแกรมภาษาระดับสูง ซึ่งเป็นชุดคำสั่งที่คนทั่วไปเข้าใจได้ง่ายเหมือนการใช้คำสั่งในภาษาอังกฤษหรือการเขียนสมการคณิตศาสตร์ทั่วไป นักพัฒนาโปรแกรมไม่จำเป็นต้องเข้าใจหลักการทำงานของฮาร์ดแวร์ก็สามารถพัฒนาโปรแกรมภาษาคอมพิวเตอร์ระดับสูงได้ ซึ่งโปรแกรมภาษาระดับสูงจะต้องมีตัวแปลภาษาที่ทำหน้าที่เปลี่ยนชุดคำสั่งให้เป็นภาษาแอสเซมบลีหรือภาษาเครื่องจึงจะทำงานได้

1.2.1 **ภาษาคอมพิวเตอร์ระดับสูง**

ในปัจจุบันมีโปรแกรมภาษาคอมพิวเตอร์ระดับสูงหลายภาษา แต่ที่นิยมใช้และได้รับการยอมรับมีจำนวนไม่มาก ภาษาคอมพิวเตอร์ที่พัฒนาขึ้นในยุคแรก ยังมีข้อจำกัดในการที่จะพัฒนาโปรแกรมขนาดใหญ่และมีความซับซ้อน ได้แก่ ภาษาฟอร์แทน (FORTRAN : FORmula TRANslator) ภาษาโคบอล (COBOL : Common Business Oriented Language) และภาษาเบสิก (BASIC : Beginners All-purpose Symbolic Instructional Code) ในช่วงต้นปี ค.ศ.1970 จึงมีภาษาคอมพิวเตอร์ที่เป็นภาษาเชิงกระบวนการ (Procedural หรือ Structural Language) เกิดขึ้น ซึ่งมี

ความยืดหยุ่นในการพัฒนาโปรแกรม ทำให้สามารถแก้ไขและบำรุงรักษาได้ง่าย เนื่องจากโปรแกรมถูกแยกออกเป็นส่วน ๆ ภาษาคอมพิวเตอร์ที่เป็นภาษาเชิงกระบวนการที่สำคัญคือ

- ภาษาปาสคาล (Pascal) พัฒนาโดย Nicklaus Wirth ในปี ค.ศ. 1971 โดยมีวัตถุประสงค์เพื่อใช้ในการสอนการเขียนโปรแกรมเชิงกระบวนการในมหาวิทยาลัย แต่ด้วยภาษาปาสคาลไม่มีคุณลักษณะที่จะสนับสนุนการพัฒนาโปรแกรมด้านธุรกิจและอุตสาหกรรมจึงไม่ได้รับความนิยมมากนัก

- ภาษาซี (C) พัฒนาพร้อมภาษาปาสคาล ซึ่งได้นำเอาจุดเด่นของภาษา BCPL และภาษา B มาใช้และได้เพิ่มคุณลักษณะและชนิดข้อมูลอื่นเข้ามาร่วมด้วย

จุดด้อยของการพัฒนาโปรแกรมภาษาเชิงกระบวนการคือ จะมีต้นทุนในการพัฒนาโปรแกรมที่สูง เนื่องจากต้องมีการออกแบบโปรแกรมให้แยกออกเป็นส่วน ๆ ที่เรียกว่าโมดูล (Module) ซึ่งต้องเป็นอิสระจากกัน ซึ่งหากออกแบบมาไม่ดีจะทำให้การแก้ไขและบำรุงรักษาโปรแกรมเป็นไปได้ยาก ดังนั้นจึงมีการพัฒนาภาษาคอมพิวเตอร์ที่มีลักษณะการโปรแกรมเชิงวัตถุ (Object Oriented Programming) ที่เป็นการเลียนแบบการทำงานของออปเจ็กต์ เพื่อให้ง่ายต่อการพัฒนาโปรแกรมและสามารถนำโปรแกรมกับมาใช้ใหม่ (Reuse) ได้ดีกว่าภาษาเชิงกระบวนการ ภาษาเชิงวัตถุที่สำคัญ คือ

- ภาษาซีพลัสพลัส (C++) เป็นภาษาที่พัฒนามาจากภาษาซี โดยได้เพิ่มหลักการเชิงวัตถุขึ้นมาจากภาษาซี ซึ่งเป็นที่นิยมในปัจจุบัน

- ภาษาสมอลทอล์ค (Smalltalk) เป็นภาษาโปรแกรมเชิงวัตถุที่ได้ออกแบบในปี ค.ศ. 1970 ที่ Xerox PARC โดย อลันด์ เคย์ (ผู้ริเริ่มใช้คำเรียกว่า Object-oriented) Dan Ingalls Ted Kaehler Adele Goldberg และคนอื่นๆ ภาษาสมอลทอล์คยังคงมีการพัฒนาอย่างต่อเนื่อง และมีชุมชนผู้ใช้ที่เหนียวแน่น ภาษาสมอลล์ทอล์กเป็นภาษาที่มีกระบวนการจำแนกแบบยืดหยุ่น (Dynamic)

- ภาษาจาวา (JAVA) เป็นภาษาคอมพิวเตอร์ที่พัฒนาโดยนักวิจัยของบริษัทซันไม - โครซิสเต็มส์ โดยเริ่มมีการนำมาเผยแพร่เมื่อปี ค.ศ. 1995 ปัจจุบันเป็นภาษาเชิงวัตถุที่เป็นที่นิยมใช้กันมาก โดยได้รับการยอมรับในการพัฒนาโปรแกรมทางธุรกิจและอุตสาหกรรม และเป็นที่นิยมใช้เพื่อศึกษาหลักการเขียนโปรแกรมเชิงวัตถุอีกด้วย

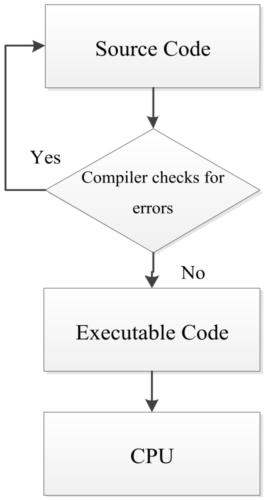
นอกจากนี้ในปัจจุบันยังเข้าสู่ภาษาในยุคที่5 (Fifth Generation Language : 5GL) เรียกได้ว่าเป็นภาษาธรรมชาติ (Natural Language) ด้วยมีความใกล้เคียงกับภาษามนุษย์มากที่สุด ภาษาในยุคนี้สามารถรองรับเทคโนโลยีปัญญาประดิษฐ์ (Artificial Intelligence : AI) เช่น การทำให้เครื่องคอมพิวเตอร์สามารถเข้าใจคำสั่งจากเสียงพูดและโต้ตอบได้

1.2.2 ตัวแปลภาษา

ภาษาคอมพิวเตอร์ระดับสูงจะต้องมีตัวแปลภาษา (Language translator) เพื่อแปลโปรแกรมที่เขียนขึ้นมา (Source code) ของภาษาคอมพิวเตอร์แต่ละภาษาให้เป็นภาษาแอสเซมบลีหรือภาษาเครื่องที่เครื่องคอมพิวเตอร์สามารถเข้าใจได้ ตัวแปลภาษาแบ่งออกได้เป็น 2 ประเภท คือ คอมไพเลอร์ และอินเตอร์พรีตเตอร์

1) คอมไพเลอร์ (Compiler)

เป็นการแปลภาษาด้วยการแปลชุดคำสั่งในซอร์สโค้ดทั้งหมดให้เป็นภาษาเครื่องที่เป็นรหัสกระทำการ (Executable code) ก่อนแล้วเก็บลงในหน่วยความจำ แล้วจึงสั่งให้เครื่องคอมพิวเตอร์ทำงาน ข้อดีคือ ทำงานได้รวดเร็วเนื่องจากแปลคำสั่งเพียงครั้งเดียวแล้วสามารถสั่งงานให้คอมพิวเตอร์ทำงานได้เลย ข้อเสียคือ ต้องการพื้นที่ของหน่วยความจำเพื่อเก็บข้อมูลที่ถูกแปลมาก การหาจุดผิดพลาดของโปรแกรมทำได้ยากและมีโครงสร้างของภาษาซับซ้อนกว่า ภาษาคอมพิวเตอร์ที่ใช้ตัวแปลภาษาประเภทคอมไพเลอร์คือ ภาษาซี C++ ฟอร์แทน และปาสคาล เป็นต้น ขั้นตอนการทำงานของคอมไพเลอร์ แสดงดังภาพที่ 1.1



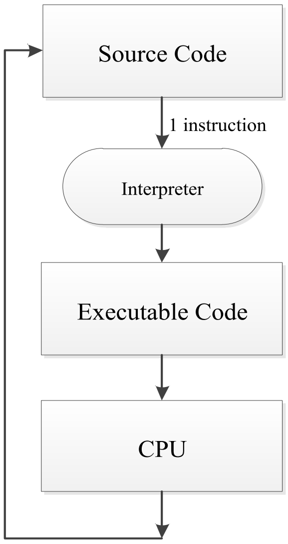
ภาพที่ 1.1 การทำงานของคอมไพเลอร์

จากภาพที่ 1.1 คอมไพเลอร์จะทำการแปลชุดคำสั่งที่เป็นซอร์สโค้ดให้เป็นโปรแกรมรหัสกระทำการทั้งโปรแกรม หากไม่พบข้อผิดพลาดของคำสั่งจากนั้นจะนำไปเก็บที่หน่วยความจำเพื่อเรียกใช้งาน แต่การสร้างโปรแกรมรหัสกระทำการ (Executable Code) ที่ได้จะขึ้นกับแพลตฟอร์ม(ฮาร์ดแวร์และระบบปฏิบัติการ) ซึ่งจะไม่สามารถนำไปใช้บนแพลตฟอร์มอื่นได้

หากนักพัฒนาโปรแกรมต้องการนำโปรแกรมที่พัฒนาขึ้นไปใช้บนแพลตฟอร์มอื่น จะต้องทำการแปลซอร์สโค้ดของโปรแกรมใหม่บนแพลตฟอร์มที่ต้องการ

2) อินเตอร์พรีเตอร์ (Interpreter)

เป็นการแปลภาษาด้วยการแปลจากชุดคำสั่งทีละบรรทัด หรือทีละคำสั่งให้เป็นภาษาเครื่องที่เป็นรหัสกระทำการ (Executable code) แล้วสั่งให้เครื่องคอมพิวเตอร์ทำงานตามคำสั่งที่แปลได้ทันทีจากนั้นจึงแปลคำสั่งถัดไปจนจบโปรแกรมหากมีข้อผิดพลาดจากการแปลจะหยุดการทำงานเพื่อให้แก้ไขข้อผิดพลาดก่อน ขั้นตอนการทำงานของอินเตอร์พรีเตอร์ แสดงดังภาพที่ 1.2



ภาพที่ 1.2 การทำงานของอินเตอร์พรีเตอร์

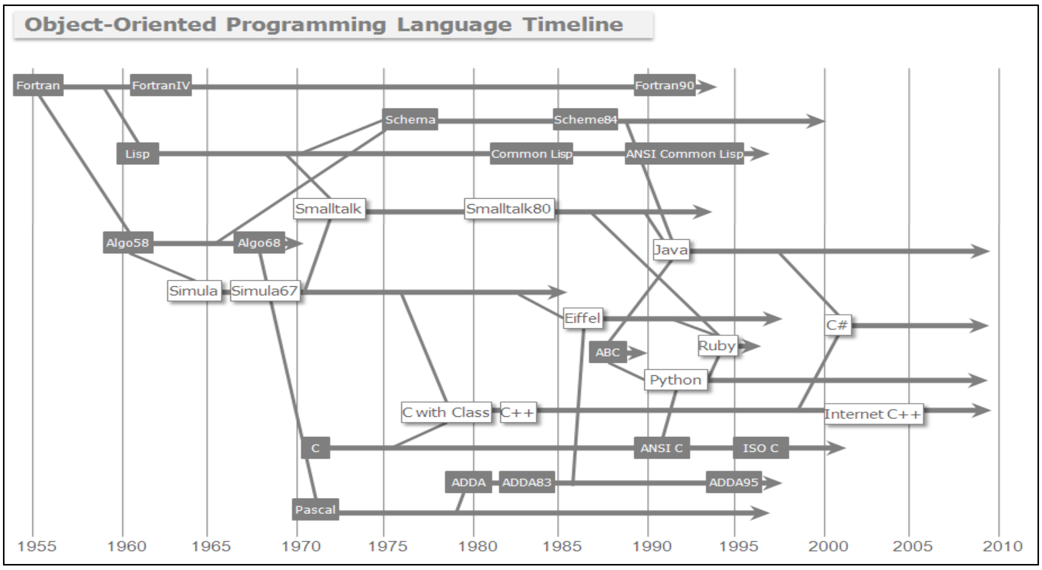
การแปลภาษาวิธีนี้จะทำงานได้ช้ากว่าโปรแกรมภาษาคอมพิวเตอร์ที่ใช้คอมไพเลอร์ เนื่องจากต้องหยุดรอการแปล มีข้อดีคือสามารถหาข้อผิดพลาดของโปรแกรมได้ง่ายและสามารถสั่งให้โปรแกรมทำงานเฉพาะจุดได้ นอกจากนี้โปรแกรมภาษาคอมพิวเตอร์ที่ใช้อินเตอร์พรีเตอร์จะสามารถใช้งานข้ามแพลตฟอร์มได้

1.3 **แนวคิดเชิงวัตถุ**

แนวคิดเชิงวัตถุ คือ การมองหรือการใช้วัตถุ (Object) เป็นตัวหลักในการพิจารณาความเป็นจริงในโลก โดยพิจารณาว่ากิจกรรมทั้งหมดที่เกิดขึ้นในโลกนั้น เกิดจากความสัมพันธ์ และการปฏิสัมพันธ์ของวัตถุต่าง ๆ ที่มีต่อกัน โดยจะมองทุกสิ่งทั้งที่เป็นรูปธรรม (สิ่งที่จับต้องได้) และนามธรรม (สิ่งที่จับต้องไม่ได้) ให้เป็นวัตถุ และใช้เป็นตัวแทนในการทำกิจกรรมบนคอมพิวเตอร์

เพื่อให้คอมพิวเตอร์สามารถแก้ปัญหาที่เกิดขึ้นจริงในโลกได้ ดังนั้นการเขียนโปรแกรมที่นำแนวคิดเชิงวัตถุมาใช้จึงถูกเรียกว่า “การโปรแกรมเชิงวัตถุ”

ภาษาการโปรแกรมเชิงวัตถุได้มีวิวัฒนาการมาตั้งแต่อดีตจนถึงปัจจุบัน โดยภาษาแรกที่นำแนวคิดเชิงวัตถุมาใช้คือ ภาษาซิมูลา (Simula) ซึ่งมีรากฐานมาจากภาษาอัลกอ 58 (Algo 58) และต่อมาได้กลายเป็นรากฐานของภาษาการโปรแกรมเชิงวัตถุอื่นอีกมากมาย เช่น ภาษาสมอลล์ทอล์ก (Smalltalk) ภาษาซีวิธคลาส (C with class) และภาษาไอเฟล (Eiffel) โดยภาษาที่เป็นที่รู้จักกันดีในยุค 2000 ก็คือ ภาษาจาวา (Java) นอกจากนี้ยังมีภาษาสคริปต์ที่นำแนวคิดเชิงวัตถุมาใช้ เช่น ภาษาไพธอน (Python) และภาษารูบี้ (Ruby) เป็นต้น ซึ่งวิวัฒนาการของภาษาการโปรแกรมเชิงวัตถุแสดงดังภาพที่ 1.3



ภาพที่ 1.3 วิวัฒนาการของภาษาการโปรแกรมเชิงวัตถุ

จากวิวัฒนาการของภาษาการโปรแกรมพบว่า ภาษาการโปรแกรมในยุคเก่าที่พัฒนาด้วยแนวคิดแบบเดิม คือเป็นภาษาการโปรแกรมเชิงโครงสร้างหรือโมดูล (Structured/Modular Programming) เช่น ภาษาฟอร์แทรน (Fortran) ภาษาซี และภาษาอัลกอ 58 เป็นต้น ได้ถูกนำมาพัฒนาเป็นภาษาการโปรแกรมเชิงวัตถุทั้งสิ้น นอกจากนี้ระบบสารสนเทศส่วนใหญ่ในปัจจุบันถูกพัฒนาโดยใช้ภาษาการโปรแกรมเชิงวัตถุเช่นกัน ซึ่งเหตุผลหลักที่ปัจจุบันระบบสารสนเทศมากมายได้ถูกพัฒนามาจากการใช้โปรแกรมเชิงวัตถุนั้นเนื่องมาจาก ระบบงานหรือขั้นตอนการทำงานของภาคธุรกิจในปัจจุบันมีความซับซ้อนเพิ่มมากขึ้น ทำให้การพัฒนาโปรแกรมแบบเดิมไม่สามารถรองรับการทำงานที่ซับซ้อนเหล่านั้นได้ รวมทั้งการเปลี่ยนแปลงแก้ไข และบำรุงรักษาซอฟแวร์ทำ

ได้ยากมาก การโปรแกรมเชิงวัตถุจึงถูกนำมาใช้เพื่อช่วยให้การพัฒนาระบบสารสนเทศมีประสิทธิภาพมากขึ้น

1.4 แนวคิดเชิงวัตถุขั้นพื้นฐาน

แนวคิดเชิงวัตถุขั้นพื้นฐานประกอบด้วย 4 แนวคิด ได้แก่ แนวคิดการมองแบบนามธรรม แนวคิดการห่อหุ้ม แนวคิดการทำโมดูล และแนวคิดการสร้างลำดับชั้น โดยแนวคิดพื้นฐานเหล่านี้เป็นแนวคิดที่จำเป็นในการโปรแกรมเชิงวัตถุเป็นอย่างยิ่ง ซึ่งรายละเอียดของแต่ละแนวคิดสามารถอธิบายได้ดังนี้

1.4.1 แนวคิดการมองแบบนามธรรม (Abstraction)

ก่อนที่จะเข้าสู่แนวคิดพื้นฐานแรกของหลักการเชิงวัตถุนั้น องค์ประกอบของแนวคิดเชิงวัตถุที่จำเป็นก็คือ วัตถุ (Object) และคลาส (Class) ที่เป็นส่วนสำคัญในแนวคิดเชิงวัตถุ มีลักษณะที่สำคัญ ดังนี้

1) วัตถุและคลาส

วัตถุเป็นตัวแทนของสิ่งที่สนใจ ซึ่งอาจจับต้องได้หรืออาจจับต้องไม่ได้ ตัวอย่างของวัตถุที่จับต้องได้ เช่น เครื่องคิดเลข บัญชีธนาคาร และคน เป็นต้น ส่วนวัตถุที่จับต้องไม่ได้ เช่น วิชาเรียน คะแนน และเที่ยวบิน เป็นต้น ซึ่งวัตถุทั้งสองชนิดนี้จะมีองค์ประกอบที่เหมือนกัน ได้แก่ คุณสมบัติหรือคุณลักษณะ (Property/Attribute) และพฤติกรรมหรือการกระทำ (Behavior/Operation) แสดงตัวอย่างองค์ประกอบของวัตถุที่สนใจ ดังตารางที่ 1.1

ตารางที่ 1.1 องค์ประกอบของวัตถุ

วัตถุ	คุณสมบัติ	พฤติกรรม
เครื่องคิดเลข	ตัวเลข เครื่องหมาย จุดทศนิยม	คำนวณผลลัพธ์
บัญชีธนาคาร	ชื่อเจ้าของบัญชี เลขที่บัญชี ยอดเงินคงเหลือ	ฝากเงิน ถอนเงิน โอนเงิน
คน	ชื่อ ความสูง น้ำหนัก อายุ	ฟัง พูด อ่าน เขียน

วัตถุเดียวกันอาจมีองค์ประกอบที่แตกต่างกันได้ ขึ้นอยู่กับมุมมองของบุคคลที่สนใจในวัตถุนั้น เช่น ถ้าวัตถุที่สนใจคือ “นักศึกษา” และผู้ที่สนใจคือ “อาจารย์” คุณสมบัติของนักศึกษาที่อาจารย์ต้องการ ได้แก่ ชื่อ-นามสกุล คะแนนสอบ จำนวนวันขาดเรียน และความประพฤติ เพราะอาจารย์ต้องการนำคุณสมบัติเหล่านั้นไปใช้ในการพิจารณาให้เกรดนักศึกษา แต่ถ้าผู้ที่สนใจคือ “เจ้าหน้าที่ฝ่ายทะเบียน” คุณสมบัติของนักศึกษาที่เจ้าหน้าที่ต้องการ ได้แก่ รหัสนักศึกษา วิชาที่เรียน

และค่าลงทะเบียน เพราะเจ้าหน้าที่ต้องการนำคุณสมบัติเหล่านั้นไปใช้ในการพิจารณาการชำระค่าลงทะเบียนเรียนของนักศึกษา เป็นต้น ดังแสดงในภาพที่ 1.4

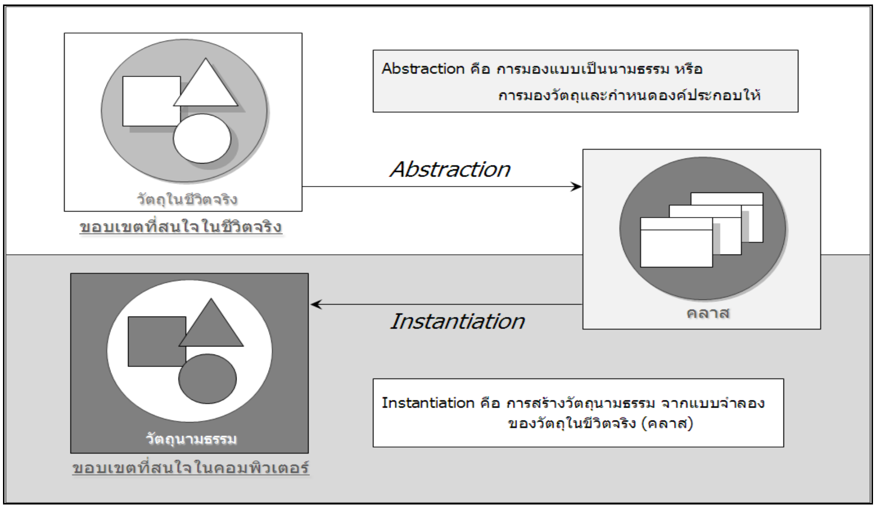
ภาพที่ 1.4 มุมมองที่มีต่อวัตถุ

ในการโปรแกรมเชิงวัตถุนั้น วัตถุจะถูกแบ่งเป็น 2 ประเภท ได้แก่ วัตถุในชีวิตจริง (Real-World Object) และวัตถุนามธรรม (Abstract Object หรือ Instance) การโปรแกรมเชิงวัตถุจะนำวัตถุในชีวิตจริง (อาจจะเป็นวัตถุที่จับต้องได้หรือไม่ได้) ที่อยู่ในขอบเขตที่สนใจเท่านั้นมาพิจารณาและเปลี่ยนให้เป็นวัตถุนามธรรมเพื่อเป็นตัวแทนในการทำกิจกรรมต่างๆ บนระบบคอมพิวเตอร์ หรือการพัฒนาโปรแกรมเชิงวัตถุ โดยวิธีการเปลี่ยนจากวัตถุในชีวิตจริงเป็นวัตถุนามธรรมนั้นมี 2 กระบวนการดังนี้

กระบวนการแรก คือ การมองวัตถุในชีวิตจริงและใส่ความคิดลงไป เรียกว่า กระบวนการมองแบบนามธรรม (Abstraction) สิ่งที่ได้จากการมองแบบนามธรรมก็คือ “*คลาส*” โดยคลาสจะเปรียบเสมือน “*แบบจำลองของวัตถุในชีวิตจริง*” ที่อยู่ในขอบเขตที่เราสนใจ ดังนั้นคลาสที่ได้จะมีองค์ประกอบ (ได้แก่ คุณสมบัติและพฤติกรรม) เช่นเดียวกับวัตถุที่ใช้สร้างคลาสนั้น ตัวอย่างเช่น คลาสนักศึกษาซึ่งถูกสร้างโดยเจ้าหน้าที่ฝ่ายทะเบียน จะมีคุณสมบัติและพฤติกรรมดังนี้

คลาส →	นักศึกษา
คุณสมบัติ →	- รหัสนักศึกษา - วิชาที่เรียน - ค่าลงทะเบียน
พฤติกรรม →	- กำหนดรหัสนักศึกษา - แสดงรหัสนักศึกษา - เพิ่มวิชาเรียน - แสดงวิชาเรียน - แสดงค่าเทอมทั้งหมด

กระบวนการที่สอง คือ กระบวนการสร้างวัตถุนามธรรม (Instantiation) โดย "*คลาส*" จะถูกนำมาใช้เป็น "*แม่พิมพ์ในการสร้างวัตถุนามธรรม*" ดังนั้นวัตถุนามธรรมที่สร้างขึ้นมาจึงมีคุณสมบัติและพฤติกรรมเหมือนกับแบบจำลองหรือคลาสที่เป็นแม่พิมพ์ของมันทุกประการ ซึ่งจุดประสงค์ของการสร้างวัตถุนามธรรมดังที่กล่าวมาแล้ว ก็เพื่อเป็นตัวแทนของคลาสในการทำกิจกรรมต่าง ๆ ในการพัฒนาโปรแกรมเชิงวัตถุ โดยคุณสมบัติของคลาสก็คือ ตัวแปร (Variables) หรือแอททริบิวต์ (Attributes) ในภาษาโปรแกรมเชิงวัตถุ และพฤติกรรมของคลาสก็คือ เมธอด (Method) ในภาษาโปรแกรมเชิงวัตถุ



ภาพที่ 1.5 กระบวนการสร้างคลาสและวัตถุนามธรรม

ภาพที่ 1.5 แสดงการเปลี่ยนวัตถุในชีวิตจริงให้เป็นวัตถุนามธรรม เพื่อเป็นตัวแทนในการทำกิจกรรมต่าง ๆ บนระบบคอมพิวเตอร์ ซึ่งมี 2 กระบวนการตามที่กล่าวมาแล้ว โดยกระบวนการมองแบบนามธรรม (Abstraction) ก็คือ กระบวนการสร้างคลาสจากวัตถุในชีวิตจริงที่อยู่ในขอบเขตที่สนใจ โดยการมองวัตถุแล้วใส่ความคิดลงไป และกระบวนการสร้างวัตถุนามธรรม

ตัวอย่างกระบวนการเปลี่ยนวัตถุในชีวิตจริงให้เป็นวัตถุนามธรรมของระบบการสั่งซื้อสินค้า มีดังนี้

1) *วัตถุในชีวิตจริง* ที่อยู่ในขอบเขตของระบบการสั่งซื้อสินค้า ได้แก่ สินค้า ผู้ซื้อและใบสั่งซื้อ

2) *คลาส* ของวัตถุในชีวิตจริงที่ได้จากการกระบวนการมองแบบนามธรรม

3) *วัตถุนามธรรม* ที่ได้จากกระบวนการสร้างวัตถุนามธรรมจะเป็นตัวแทนของคลาส ซึ่งมีคุณสมบัติและพฤติกรรมทั้งหมดตามที่ประกาศไว้ในคลาสนั้นทุกประการ คุณสมบัติเปรียบเสมือน "แอททริบิวต์" และพฤติกรรมเปรียบเสมือน "เมธอด" ในภาษาโปรแกรมเชิงวัตถุนั้น

และการสร้างวัตถุนามธรรมขึ้นมานั้น โดยปกติจะมีการกำหนดค่าแอททริบิวต์เริ่มต้นไว้แล้ว และถ้าต้องการเปลี่ยนค่าของแอททริบิวต์เหล่านั้น ต้องใช้เมธอดที่มีอยู่เป็นตัวเปลี่ยนค่าแอททริบิวต์

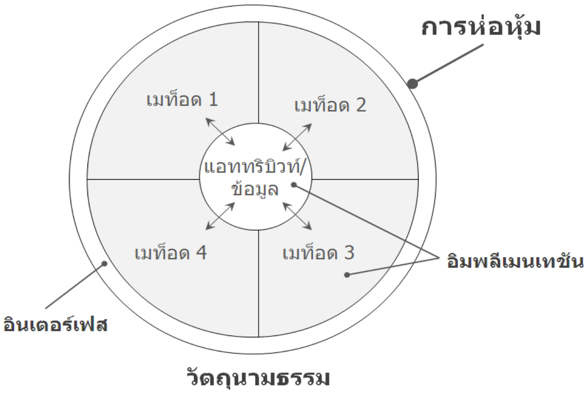
สรุปแนวคิดการมองแบบนามธรรม ก็คือ การมองวัตถุแล้วใส่ความคิดลงไปเพื่อกำหนดคุณสมบัติและพฤติกรรมของวัตถุนั้น และได้คลาสเพื่อเป็นแม่พิมพ์ในการสร้างวัตถุนามธรรมที่ใช้เป็นตัวแทนในการทำงานต่างๆ บนคอมพิวเตอร์ หรือการพัฒนาโปรแกรมเชิงวัตถุนั่นเอง

1.4.2 **แนวคิดการห่อหุ้ม (Encapsulation)**

แนวคิดการห่อหุ้ม (Encapsulation) มีรากศัพท์มาจากคำว่า แคปซูล (Capsule) ดังนั้นแนวคิดการห่อหุ้มจึงหมายถึง การซ่อนข้อมูลและการทำงานของวัตถุไว้ภายใน (Information Hiding) เพื่อป้องกันการมองเห็นและการเข้าถึงข้อมูลจากวัตถุภายนอก โดยการห่อหุ้มนี้จะมองวัตถุเฉพาะภายนอกเท่านั้น ไม่ต้องรู้ว่าวัตถุนั้นเกิดขึ้นได้อย่างไร หรือมีการทำงานภายในอย่างไร เช่น การใช้อุปกรณ์ควบคุมในการเปลี่ยนช่องสัญญาณโทรทัศน์ สิ่งที่เห็นจะเป็นเพียงผลลัพธ์ที่ได้จากการเปลี่ยนแปลงคือ รายการที่เปลี่ยนไปตามช่องสัญญาณที่กำหนด แต่จะไม่รู้เลยว่าโทรทัศน์นั้นทำงานอย่างไร และมีคุณสมบัติหรือข้อมูลภายในอะไรที่เปลี่ยนแปลงไปบ้าง ดังภาพที่ 1.6

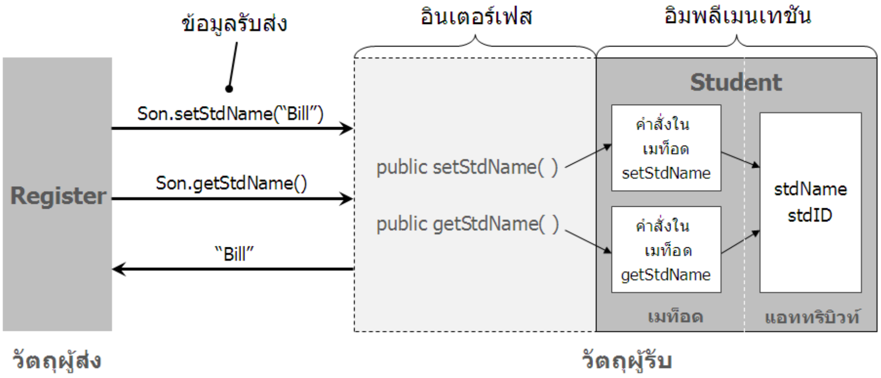
ภาพที่ 1.6 การซ่อนข้อมูลและการทำงานของวัตถุไว้ภายใน

แนวคิดการห่อหุ้มนี้พัฒนามาจากแนวคิดการมองแบบนามธรรม เมื่อได้คลาสที่เป็นแม่พิมพ์ในการสร้างวัตถุนามธรรมเพื่อนำไปใช้ในการพัฒนาโปรแกรมเชิงวัตถุ ถ้ามองในมุมมองของแนวคิดการห่อหุ้ม วัตถุนามธรรมที่ได้จากคลาสใด ๆ นั้นจะห่อหุ้มคุณสมบัติ หรือข้อมูล และวิธีการทำงานภายใน หรืออิมพลีเมนเทชัน (Implementation) ของเมธอดไว้ วัตถุภายนอกจะสามารถติดต่อกับวัตถุนั้นได้เฉพาะส่วนที่วัตถุนั้นใช้เพื่อการติดต่อเท่านั้น ซึ่งเรียกส่วนนี้ว่า อินเตอร์เฟส (Interface) ตัวอย่างการห่อหุ้มวัตถุนามธรรม ดังแสดงดังภาพที่ 1.5



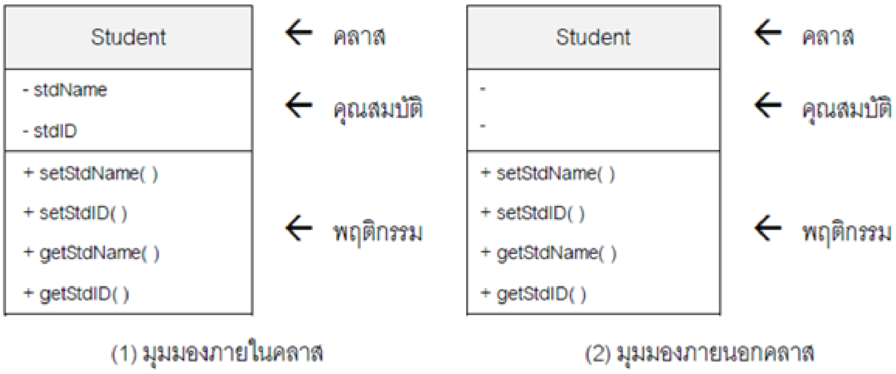
ภาพที่ 1.5 แผนภาพการห่อหุ้มของวัตถุนามธรรม

โดยสมมติว่าวัตถุผู้ส่ง คือ เจ้าหน้าที่ฝ่ายทะเบียน (Register) และวัตถุนามธรรมที่ใช้แทนคลาสนักศึกษา (Student) คือ บุตรชาย (Son) เจ้าหน้าที่ฝ่ายทะเบียนต้องการที่จะตั้งชื่อบุตรชายและแสดงผลการตั้งชื่อนั้น



ภาพที่ 1.6 การติดต่อผ่านอินเตอร์เฟสของวัตถุนามธรรม

จากภาพที่ 1.6 จะเห็นว่า เจ้าหน้าที่ฝ่ายทะเบียนจะส่งข้อมูล ได้แก่ การเรียกเมท็อดตั้งชื่อนักศึกษา "public setStdName()" และชื่อนักศึกษา (หรือแอททริ-บิวต์ "stdName") ที่ต้องการกำหนดคือ "Bill" ติดต่อกับอินเตอร์เฟสของบุตรชาย ซึ่งมีเพียงชื่อ เมท็อดต่างๆ ที่มีในคลาสเท่านั้น หลังจากนั้นบุตรชายจะกำหนดชื่อให้กับเจ้าหน้าที่ฝ่ายทะเบียนเอง (แอททริบิวต์: [ชื่อนักศึกษา] จะมีค่าใหม่คือ [ชื่อนักศึกษา = "Bill"]) โดยเจ้าหน้าที่ฝ่ายทะเบียนไม่จำเป็นต้องรู้การทำงานภายในว่าบุตรชายมีวิธีการตั้งชื่ออย่างไร เป็นต้น และการติดต่อระหว่างวัตถุผู้ส่งและวัตถุผู้รับ เพื่อให้เข้าใจได้ชัดเจนยิ่งขึ้น อาจมองแผนภาพของคลาสด้วยมุมมองที่ต่างกัน ได้แก่ การมองภายในคลาสของตนเอง และการมองภายนอกคลาสของตนเอง ดังแสดงในภาพที่ 1.7



ภาพที่ 1.7 มุมมองที่ต่างกันเมื่อคลาสใช้แนวคิดการห่อหุ้ม

จากภาพที่ 1.7 กำหนดให้คลาสนักเรียนมีคุณสมบัติ 2 ประการ คือ ชื่อนักศึกษา (stdName) และรหัสนักศึกษา (stdID) และมีพฤติกรรม 4 ประการ คือ ตั้งชื่อนักศึกษา (setStdName) กำหนดรหัสนักศึกษา (setStdID) แสดงชื่อนักศึกษา (getStdName) และแสดงรหัสนักศึกษา (getStdID) ถ้ามองในมุมมองภายในคลาส จะเห็นคุณสมบัติและพฤติกรรมทั้งหมดของคลาสนั้น ดังภาพที่ 1.7 (1) แต่ถ้ามองจากภายนอกเข้ามา จะเห็นเพียงอินเตอร์เฟสหรือชื่อของพฤติกรรมทั้งสี่เท่านั้น แต่ไม่เห็นคุณสมบัติภายในคลาส และวิธีการทางานภายในหรือพฤติกรรมเหล่านั้น ดังแสดงในภาพที่ 1.7 (2)

สรุปแนวคิดการห่อหุ้ม คือ การรวมแอททริบิวต์และเมธอดทั้งหลายที่เกี่ยวข้องกันและทำงานร่วมกันเอาไว้ในวัตถุหนึ่งๆ พร้อมทั้งซ่อนรายละเอียดของการทำงานของเมธอดต่างๆ ในวัตถุเอาไว้ และไม่ให้วัตถุอื่นๆ สามารถที่จะเข้าไปแก้ไขข้อมูลได้โดยตรง การติดต่อสื่อสารกันระหว่างวัตถุจะต้องติดต่อกันผ่านอินเตอร์เฟสที่กำหนดเอาไว้เท่านั้น ซึ่งก่อให้เกิดความสะดวกในการเปลี่ยนแปลงแก้ไขข้อมูลหรือการทำงานภายในของวัตถุต่างๆ ที่มีการห่อหุ้มไว้ เพราะการเปลี่ยนแปลงของวัตถุหนึ่งจะไม่ไปกระทบกับวัตถุอื่น

1.4.3 **แนวคิดการทำเป็นโมดูล (Modularity)**

แนวคิดการทำเป็นโมดูล หมายถึง การแบ่งปัญหาใหญ่ที่สลับซับซ้อนออกเป็นส่วนย่อย โดยส่วนย่อยจะมีเนื้อหาสมบูรณ์ด้วยตัวมันเอง เพราะการทำงานจะเสร็จสิ้นภายในหนึ่งโมดูล ทำให้สามารถแยกส่วนการทำงานและแก้ไขโปรแกรมขนาดใหญ่ได้ง่าย โดยแต่ละโมดูลจะสามารถจัดการสร้างคำสั่งหรือทำกิจกรรมใดๆ ก็ได้ภายในโมดูล แต่ต้องมีการกำหนดอินเตอร์เฟส (ส่วนนำเข้าและส่วนผลลัพธ์) ไว้ชัดเจน เพื่อให้แต่ละโมดูลสามารถเชื่อมต่อกันได้ การทำเป็นโมดูลจะช่วยให้เกิดความคล่องตัวในการแก้ไข เพราะการแก้ไขการทำงานภายในโมดูลหนึ่งจะไม่ส่งผลกระทบกับโมดูลอื่น ดังนั้นการทำเป็นโมดูลจึงเหมาะกับระบบงานขนาดใหญ่ที่มีความซับซ้อน เช่น

ระบบลงทะเบียน ซึ่งเป็นระบบที่ยุ่งยากและซับซ้อน สามารถแบ่งระบบเป็น 3 โมดูลย่อย เช่น โมดูลการจัดการเกี่ยวกับการเงิน โมดูลการจัดการเกี่ยวกับข้อมูลนักศึกษา และโมดูลการจัดการเกี่ยวกับแผนการเรียน เป็นต้น และหากโมดูลย่อยเหล่านี้ยังคงสลับซับซ้อนก็สามารถแบ่งเป็นโมดูลย่อยที่เล็กลงไปได้อีก การออกแบบเชิงโมดูลนั้นมีหลักการดังนี้

1) หลักการ “Decomposability” คือ ความสามารถในการแตกออกเป็นโมดูลย่อย โดยการพัฒนาโปรแกรมแบบโมดูลนั้น จะแยกองค์ประกอบของซอฟต์แวร์ทั้งหมด ออกเป็นหน้าที่การทำงานย่อย ซึ่งแต่ละส่วนจะมีความสัมพันธ์กันในเชิงของการรับส่งค่าหรือการเรียกใช้กัน โดยในแต่ละส่วนยังสามารถแบ่งย่อยออกไปได้อีกตามต้องการ ถ้าส่วนใดยังมีขนาดใหญ่และมีการทำงานที่ซับซ้อนอยู่

2) หลักการ “Composability” คือ ความสามารถในการรวมกันของโมดูลเพื่อทำหน้าที่ใดหน้าที่หนึ่ง การพัฒนาโปรแกรมแบบโมดูลที่ดี จะต้องสามารถนำเอาโมดูลย่อยที่ออกแบบไว้ นำมารวมกันให้เป็นระบบใหม่ที่รวมการทำงานทั้งหมดของโมดูลย่อยไว้ด้วยกัน โดยโมดูลย่อยเหล่านั้นจะทำหน้าที่ตามที่ตนได้รับการออกแบบไว้

3) หลักการ “Understandability” คือ ความสามารถในการเข้าใจโมดูลใดโมดูลหนึ่งได้ โดยไม่ต้องพิจารณาองค์ประกอบอื่นของโปรแกรมเพื่อให้เข้าใจโมดูลที่กำลังพิจารณาอยู่นั้น นั่นก็คือ แต่ละโมดูลจะต้องมีความหมายอยู่ในตัวเอง ไม่จำเป็นต้องอ้างถึงโมดูลตัวอื่น

4) หลักการ “Continuity” คือ การออกแบบเชิงโมดูลให้รองรับการเปลี่ยนแปลงการทำงานของโปรแกรมเพียงบางส่วน โดยผลที่เกิดขึ้นคือ การเปลี่ยนแปลงโมดูลเพียงตัวเดียวหรือบางตัว ไม่จำเป็นต้องเปลี่ยนแปลงโครงสร้างโปรแกรมใหม่ทั้งหมด

5) หลักการ “Protection” คือ การออกแบบเชิงโมดูลซึ่งเมื่อเกิดความผิดพลาดในการทำงานภายในโมดูลใดโมดูลหนึ่ง ผลที่ผิดพลาดจะต้องไม่ทำความเสียหายต่อการทำงานหรือการออกแบบโดยรวม โดยผลที่เกิดขึ้นควรควบคุมได้ภายในโมดูลนั้น หรือโมดูลที่เกี่ยวข้องเท่านั้น

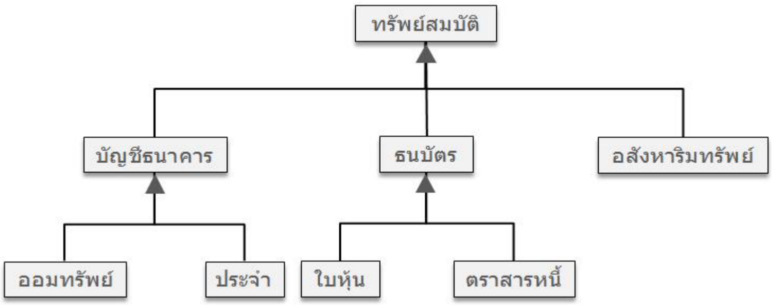
สรุปแนวคิดการทำเป็นโมดูล คือ การแบ่งระบบใหญ่ที่ซับซ้อนให้เป็นส่วนย่อยหรือโมดูลย่อยหลายส่วน และให้การทำงานเสร็จสิ้นภายในหนึ่งโมดูล หลังจากนั้นจึงนำโมดูลย่อยเหล่านั้นมาประกอบกันเป็นระบบที่สมบูรณ์ ซึ่งการทำเป็นโมดูลจะช่วยให้เราสามารถดูแลระบบได้ง่ายขึ้น เพราะโมดูลเหล่านี้สามารถแก้ไขเปลี่ยนแปลงการทำงานโดยอิสระไม่ขึ้นต่อกัน จะมีเพียงอินเตอร์เฟสเท่านั้นที่ใช้ติดต่อสื่อสารกันระหว่างโมดูลเหล่านั้น

1.4.4 **แนวคิดการสร้างลำดับชั้น (Hierarchy)**

แนวคิดการสร้างลำดับชั้น คือ การจัดกลุ่มสิ่งต่างๆ เป็นลำดับชั้นตามแนวคิดการมองแบบนามธรรม โดยมองว่าวัตถุต่างๆ มีความสัมพันธ์กันเป็นลำดับชั้นแบบโครงสร้างต้นไม้ ซึ่งการ

มองด้วยแนวคิดที่ต่างกันจะทำให้ได้ลำดับชั้นโครงสร้างต้นไม้ที่ต่างกัน ขึ้นอยู่กับมุมมองของแต่ละบุคคลเช่นเดียวกับการกำหนดคุณสมบัติและพฤติกรรมของวัตถุนั้น เช่น ยานพาหนะสามารถแบ่งเป็นยานพาหนะทางอากาศ ทางบก หรือทางน้ำ แต่ถ้าพิจารณายานพาหนะในแง่ของระบบเครื่องยนต์ อาจแบ่งยานพาหนะได้เป็น ยานพาหนะเครื่องยนต์เบนซิน เครื่องยนต์ดีเซล หรือเครื่องยนต์ไอน้ำ เป็นต้น

การจัดกลุ่มเป็นลำดับชั้นมีแนวคิดดังนี้ 1) วัตถุที่อยู่ในโครงสร้างต้นไม้เดียวกัน ต้องมีความสัมพันธ์กันในแง่ใดแง่หนึ่ง 2) วัตถุที่อยู่ในลำดับชั้นที่ต่ำกว่าจะมีความหมายที่เฉพาะเจาะจงกว่าวัตถุที่อยู่ในลำดับชั้นที่สูงกว่า และ 3) วัตถุที่อยู่ในลำดับชั้นเดียวกันของโครงสร้างต้นไม้ ต้องมีแนวความคิดอยู่ในระดับเดียวกัน แต่เป็นแนวคิดที่แตกต่างกันอย่างสิ้นเชิง ตัวอย่างดังภาพที่ 1.8



ภาพที่ 1.8 การสร้างลำดับชั้นแบบโครงสร้างต้นไม้

จากภาพที่ 1.8 แสดงการสร้างลำดับชั้นแบบโครงสร้างต้นไม้ โดยวัตถุที่อยู่ในขอบเขตที่สนใจ คือ ทรัพย์สมบัติ ซึ่งประกอบไปด้วย บัญชีธนาคาร ธนบัตร และอสังหาริมทรัพย์ ที่ส่วนประกอบนี้ควรจะจัดอยู่ในระดับเดียวกัน เพราะเป็นแนวคิดในระดับเดียวกัน และเป็นสิ่งที่ต่างกันอย่างสิ้นเชิง ในขณะที่บัญชีธนาคาร อาจแบ่งประเภทเป็นบัญชีออมทรัพย์และบัญชีประจำ ซึ่งควรแยกเป็นระดับชั้นใหม่ที่อยู่ต่ำกว่าบัญชีธนาคาร ไม่ควรจัดอยู่ในระดับชั้นเดียวกับบัญชีธนาคาร เพราะมีความหมายเฉพาะเจาะจงมากกว่า เช่นเดียวกับใบหุ้นและตราสารหนี้ ซึ่งเป็นประเภทต่างๆ ของธนบัตร จึงควรแยกเป็นระดับชั้นล่างของธนบัตร เป็นต้น

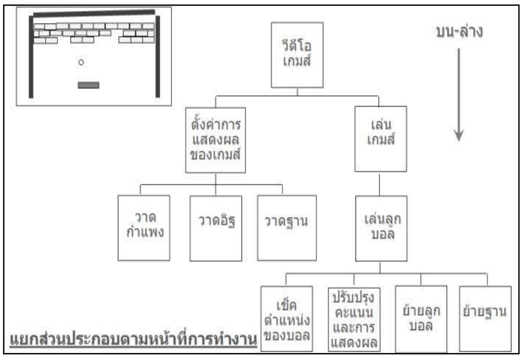
1.5 โปรแกรมเชิงโครงสร้าง และโปรแกรมเชิงวัตถุ

การเขียนโปรแกรมโดยใช้ภาษาคอมพิวเตอร์ระดับสูงในปัจจุบันมีแนววิธีการคิดอยู่ 2 รูปแบบที่สำคัญ คือ แนวคิดเชิงกระบวนการ และแนวคิดเชิงวัตถุ

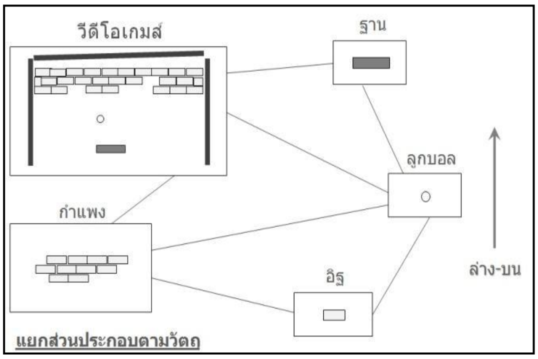
การเขียนโปรแกรมโดยใช้ภาษาเชิงกระบวนการ หรือบางครั้งอาจพบการใช้คำว่าการเขียนโปรแกรมแบบเดิมหรือที่เรียกว่าโปรแกรมเชิงโครงสร้าง (Structured Programming) ซึ่งรวมถึงการ

โปรแกรมแบบโมดูล (Modular programming) ที่เริ่มต้นการเขียนโปรแกรมจากการวิเคราะห์ปัญหา โดยพิจารณาจากกระบวนการและลำดับการทำงาน และจะพยายามแบ่งโปรแกรมออกเป็นส่วน ๆ ตามฟังก์ชันของการทำงาน (Functional Decomposition) โดยพิจารณาปัญหาและองค์ประกอบต่างๆ จากบนลงล่าง (Top-Down) คือเน้นภาพรวมโครงสร้างการทำงานทั้งหมดก่อน และกำหนดโมดูลหลักเพื่อเริ่มต้นการทำงานนั้น หากพบความซับซ้อนเกิดขึ้นในการทำงานของโมดูลหลักจะแตกโมดูลหลักนั้นออกเป็นโมดูลย่อยเพื่อกระจายหรือแบ่งส่วนการทำงานให้เล็กลง เพื่อให้ง่ายต่อการพัฒนา การแก้ไข และการปรับปรุงเปลี่ยนแปลงในภายหลัง และถ้าโมดูลย่อยเหล่านั้นยังคงซับซ้อนอยู่ ก็จะแบ่งย่อยลงไปอีกเรื่อยๆ จนกระทั่งผู้พัฒนาสามารถควบคุมการทำงานทั้งหมดได้อย่างมีประสิทธิภาพ เช่น การพัฒนาโปรแกรมระบบทะเบียนนักศึกษาที่พิจารณาแบ่งโปรแกรมตามลำดับการทำงาน โดยอาจมีฟังก์ชันในการทำงานสำหรับการลงทะเบียนรายวิชา การชำระเงิน การเพิ่มหรือถอนรายวิชา เป็นต้น ซึ่งฟังก์ชันแต่ละส่วนจะมีตัวแปรที่ส่งผ่านข้อมูลระหว่างกัน การออกแบบโปรแกรมที่ดีจะต้องพยายามแบ่งฟังก์ชันให้เป็นอิสระต่อกันให้มากที่สุด และต้องพยายามเขียนขั้นตอนหรือลำดับการทำงานให้สมบูรณ์ที่สุดตั้งแต่ขั้นตอนการออกแบบ จึงทำให้การปรับปรุงหรือแก้ไขโปรแกรมทำได้ยาก ทั้งนี้เนื่องจากโปรแกรมนั้นไม่สามารถที่จะออกแบบให้สมบูรณ์ตั้งแต่ต้นโดยไม่มีการนำกลับมาแก้ไขปรับปรุงอีกได้

การพัฒนาโปรแกรมเชิงวัตถุ (Object-Oriented Programming) จะพิจารณาปัญหาในมุมกลับจากการพัฒนาโปรแกรมเชิงกระบวนการ โดยมองจากล่างขึ้นบน (Bottom-Up) คือ พิจารณาส่วนการทำงานย่อยก่อน แล้วค่อยพัฒนาไปเป็นภาพรวมโครงสร้างการทำงานทั้งหมด โดยการโปรแกรมเชิงวัตถุจะนำแนวคิดของการแยกส่วนประกอบของโปรแกรมตามวัตถุ (Object-Oriented Decomposition) หรือมองระบบให้อยู่ในรูปของการปฏิสัมพันธ์กันระหว่างวัตถุที่เป็นองค์ประกอบทำให้เกิดการทำงานของระบบทั้งหมด โดยแต่ละวัตถุจะประกอบไปด้วยส่วนประกอบที่มีหน้าที่การทำงานเฉพาะอย่างที่แตกต่างกัน ขึ้นอยู่กับคุณสมบัติและพฤติกรรมของวัตถุเหล่านั้น



(1) การพัฒนาโปรแกรมแบบเดิม



(2) การพัฒนาโปรแกรมเชิงวัตถุ

จากภาพที่ 1.8 แสดงการเปรียบเทียบโครงสร้างของโปรแกรมที่พัฒนาด้วยแนวคิดเชิงกระบวนการและแบบเชิงวัตถุ เป็นการพัฒนาโปรแกรมวีดีโอเกมส์ ซึ่งในโครงสร้างของโปรแกรมแบบเดิม จะมองที่การทำงานโดยรวมก่อน และกำหนดโมดูลหลักคือ วีดีโอเกมส์ เพื่อเริ่มต้นการทำงาน เมื่อพบความซับซ้อนเกิดขึ้นในการทำงานของโมดูลหลัก จะแตกโมดูลหลักนั้นออกเป็นโมดูลย่อย ได้แก่ โมดูลตั้งค่าการแสดงผลของเกมส์ และโมดูลเล่นเกมส์ เพื่อกระจายหรือแบ่งส่วนการทำงานให้เล็กลง ช่วยให้ง่ายต่อการพัฒนา การแก้ไข และการปรับปรุงเปลี่ยนแปลงในภายหลัง ต่อจากนั้น จะพิจารณาโมดูลย่อยที่สร้างขึ้นมาว่ายังคงซับซ้อนอยู่หรือไม่ ถ้าซับซ้อนอยู่จะแบ่งย่อยลงไปอีกจนกระทั่งผู้พัฒนาสามารถควบคุมการทำงานทั้งหมดได้อย่างมีประสิทธิภาพ ตามแนวคิดการทำเป็นโมดูล ในขณะที่โครงสร้างของโปรแกรมเชิงวัตถุนั้น จะพิจารณาว่าวีดีโอเกมส์ประกอบด้วยวัตถุอะไรบ้าง หลังจากนั้นจะกำหนดคุณสมบัติและพฤติกรรมของวัตถุเหล่านั้นเพื่อให้ทำงานปฏิสัมพันธ์กับวัตถุอื่น ทำให้เกิดระบบทั้งหมดขึ้นมานั่นเอง ซึ่งวัตถุที่ประกอบขึ้นเป็นวีดีโอเกมส์ ได้แก่ กำแพง อิฐ ลูกบอล และฐาน ซึ่งวัตถุเหล่านี้จะมีหน้าที่การทำงานเฉพาะอย่างที่ต่างกัน ขึ้นอยู่กับคุณสมบัติและพฤติกรรมของวัตถุนั้น โดยวัตถุทั้งหมดจะติดต่อประสานกันผ่านอินเตอร์เฟสของแต่ละวัตถุ ซึ่งอาจจะประกอบไปด้วยแอททริบิวต์และโมดูลย่อยมากมายภายใน ทำให้เกิดการทำงานของระบบวีดีโอเกมส์ทั้งหมดขึ้นมา

1.6 **จุดเด่นของการพัฒนาโปรแกรมเชิงวัตถุ**

การพัฒนาโปรแกรมเชิงกระบวนการจะมองระบบเป็นงาน ซึ่งถ้างานมีขนาดใหญ่ก็จะแบ่งส่วนการทำงานให้เล็กลง โดยข้อมูลจะถูกแยกจากกระบวนการทำงานนั้น ซึ่งการทำแบบนี้จะไม่สอดคล้องกับความเป็นจริงในโลก ทำให้ระบบยากต่อการเข้าใจ การรักษาข้อมูลให้ถูกต้องทำได้ยาก และขาดเสถียรภาพ เนื่องจากการปรับปรุงหรือเปลี่ยนแปลงการทำงานส่วนใดส่วนหนึ่งจะกระทบกับการทำงานส่วนอื่นที่สัมพันธ์กัน นอกจากนี้ความสามารถในการนำซอฟต์แวร์บางส่วนกลับมาใช้ใหม่ (Reusability) ยังทำได้ยาก เนื่องจากข้อมูลที่ต้องใช้ร่วมกันในหลายโมดูล จะอยู่ภายนอกโมดูลเหล่านั้น หรือที่เรียกว่า ข้อมูล/ตัวแปรโกลบอล ซึ่งโดยส่วนใหญ่การพัฒนาโปรแกรมแบบเดิมจะนำซอฟต์แวร์กลับมาใช้ใหม่เฉพาะฟังก์ชั่นการทำงานเท่านั้น ไม่ได้รวมตัวแปรโกลบอลมาด้วย

การพัฒนาโปรแกรมเชิงวัตถุจะใช้แนวคิดที่ใกล้เคียงกับความเป็นจริงในโลกมากกว่าการพัฒนาโปรแกรมเชิงกระบวนการ คือ การพิจารณาว่าระบบเกิดจากความสัมพันธ์ และการมีปฏิสัมพันธ์กันของวัตถุ มองทุกสิ่งให้เป็นวัตถุ (รวมทั้งปัญหาที่เป็นนามธรรม) เพื่อให้คอมพิวเตอร์สามารถแก้ปัญหาได้ โดยจะรวมกระบวนการทำงานและข้อมูลไว้ด้วยกันเป็นส่วนประกอบของ

วัตถุนั้น ซึ่งการทำแบบนี้จะสามารถอธิบายกระบวนการทำงานและข้อมูลที่ใช้ร่วมกันได้ถูกต้องมากกว่า สอดคล้องกับความเป็นจริงในโลก ช่วยให้ง่ายต่อการเข้าใจและการบำรุงรักษา นอกจากนั้นการที่วัตถุหรือคลาสได้รวมข้อมูลและกระบวนการทำงานไว้ด้วยกัน ทำให้ความสามารถในการนำโปรแกรมกลับมาใช้ใหม่ยังทำได้ง่าย รวมทั้งการเปลี่ยนแปลงการทำงานหรือข้อมูลบางส่วนก็สามารถทำได้โดยไม่ส่งผลกระทบต่อระบบโดยรวม ทำให้มีเสถียรภาพมากกว่าการพัฒนาโปรแกรมเชิงกระบวนการ เป็นผลทำให้แนวคิดเชิงวัตถุได้รับความนิยมมากในปัจจุบัน

บทสรุป

ในบทนี้ได้กล่าวถึงเหตุผลในการพัฒนาแนวคิดเชิงวัตถุ ซึ่งเหตุผลหลักที่ปัจจุบันระบบสารสนเทศมากมายได้ถูกพัฒนามาจากการใช้โปรแกรมเชิงวัตถุนั้นเนื่องมาจาก ระบบงานหรือขั้นตอนการทำงานของภาคธุรกิจในปัจจุบันมีความซับซ้อนเพิ่มมากขึ้น ทำให้การพัฒนาโปรแกรมแบบเดิมไม่สามารถรองรับการทำงานที่ซับซ้อนเหล่านั้นได้ รวมทั้งการเปลี่ยนแปลงแก้ไข และบำรุงรักษาซอฟแวร์ทำได้ยาก การโปรแกรมเชิงวัตถุจึงถูกนำมาใช้เพื่อช่วยให้การพัฒนาระบบสารสนเทศมีประสิทธิภาพมากขึ้น

นอกจากนี้ยังกล่าวถึงแนวคิดเชิงวัตถุขั้นพื้นฐาน 4 แนวคิด ได้แก่ แนวคิดการมองแบบนามธรรม (Abstraction) แนวคิดการห่อหุ้ม (Encapsulation) แนวคิดการทำโมดูล (Modularity) และแนวคิดการสร้างลำดับชั้น (Hierarchy) โดยแนวคิดพื้นฐานเหล่านี้ เป็นแนวคิดที่จำเป็นในการโปรแกรมเชิงวัตถุเป็นอย่างยิ่ง ซึ่งกล่าวโดยสรุปแล้ว แนวคิดเชิงวัตถุก็คือ การมองหรือการใช้วัตถุเป็นตัวหลักในการพิจารณาความเป็นจริงต่างๆ ในโลก โดยพิจารณาว่ากิจกรรมทั้งหมดที่เกิดขึ้นในโลกนั้น เกิดจากความสัมพันธ์ และปฏิสัมพันธ์ของวัตถุต่างๆ ที่มีต่อกัน โดยจะมองทุกสิ่งทั้งที่เป็นรูปธรรมและนามธรรมให้เป็นวัตถุ เพื่อให้คอมพิวเตอร์สามารถแก้ปัญหาได้ ดังนั้นจึงต้องมีกระบวนการในการเปลี่ยนวัตถุในชีวิตจริงให้เป็นวัตถุนามธรรม ซึ่งในบทนี้ได้อธิบายกระบวนการต่างๆ เหล่านั้น ได้แก่ กระบวนการ Abstraction หรือ กระบวนการสร้างคลาสจากวัตถุในชีวิตจริงที่อยู่ในขอบเขตที่สนใจ โดยการมองวัตถุแล้วใส่ความคิดลงไป และกระบวนการ Instantiation หรือกระบวนการสร้างวัตถุนามธรรมจากคลาส โดยใช้คลาสเป็นแม่พิมพ์ในการสร้างนั่นเอง

แบบฝึกหัดท้ายบท

นักศึกษาจงตอบคำถามต่อไป

1) แนวคิดเชิงวัตถุคืออะไร
2) ภาษาการโปรแกรมใดบ้างที่นำแนวคิดเชิงวัตถุมาใช้อย่างเต็มรูปแบบ
3) วัตถุในชีวิตจริงและวัตถุนามธรรมแตกต่างกันอย่างไร
4) จงยกตัวอย่างของวัตถุที่สนใจมา 3 วัตถุ พร้อมทั้งกำหนดคุณสมบัติและพฤติกรรมของวัตถุเหล่านั้น
5) แนวคิดเชิงวัตถุพื้นฐานมีอะไรบ้าง
6) จงให้คำจำกัดความของคำว่า Encapsulation
7) แนวคิดการทำเป็นโมดูลมีประโยชน์อย่างไร
8) การจัดกลุ่มวัตถุเป็นลำดับชั้นมีแนวทางในการทำอย่างไร
9) จงอธิบายความแตกต่างระหว่างการพัฒนาโปรแกรมเชิงกระบวนการและแบบเชิงวัตถุ
10) จงบอกจุดแข็งของการพัฒนาโปรแกรมแบบเชิงวัตถุ

เอกสารอ้างอิง

กิตติ ภักดีวัฒนะกุล, และศิริวรรณ อัมพรดนัย. (2544). **Object-Oriented ฉบับพื้นฐาน.** กรุงเทพฯ : เคทีพี คอมพ์ แอนด์ คอนซัลท์.

รุ่งโรจน์ โพนคำ, และปราลี มณีรัตน์. (2545). **Java Programming**. กรุงเทพฯ: ซัคเซค มีเดีย บจก.

วรรณิกา เนตรงาม. (2545). **คู่มือการเขียนโปรแกรมภาษาจาวา ฉบับผู้เริ่มต้น.** นนทบุรี: อินโฟเพรส บจก.

วีระศักดิ์ ซึงถาวร. (2543). **Java Programming Volume I**. กรุงเทพฯ: ซีเอ็ดยูเคชั่น บจก.

วีระศักดิ์ ซึงถาวร. (2545). Java Programming Volume II. กรุงเทพฯ: ซีเอ็ดยูเคชั่น บจก.

สุรางคนา ระวังยศ. (2555). การเขียนโปรแกรมเชิงวัตถุ. ค้นเมื่อ 23 สิงหาคม 2560, จาก http://www.kmitl.ac.th/~s3010819/MyFile/My%20Ebook/JAVA/ %BA%B7%B7%D5%E8%201%20%20%BA%B7%B9%D3.pdf.

About the java technology. (2012). Retrieved August 28, 2016, from: http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

Anban Pillay. (2007). Object Oriented Programming using Java. University of KwaZuluNatal.

Armstrong, E. (2002). The J2EE™ 1.4 Tutorial. Network Circle, CA: Sun Microsystem, Inc.

Java download (2016). Retrieved August 28, 2015, from: http://java.sun.com/javase/downloads/index.jsp