STUDENT PRICE BOOK CENTER

DEV-C++ PROGRAMMING: ADVANCE

Object-Oriented Programming:

- Advanced OOP
- Templates
- Asception
- Exception Handling
- STL

C++

- Standard Template Library

STUDENT PRICE BOOK CENTER

คำนำ

เมื่อผู้เรียนก้าวข้ามระดับพื้นฐานและระดับกลางของการเขียนโปรแกรมมาแล้ว การพัฒนาทักษะในระดับสูงย่อมเป็นเป้าหมายต่อไปที่สำคัญอย่างยิ่ง โดยเฉพาะในภาษา C++ ซึ่งเป็นภาษาที่มีพลังและยืดหยุ่นสูงในเชิงโครงสร้าง หนังสือเล่มนี้จัดทำขึ้นเพื่อเป็นคู่มือในการเรียนรู้แนวคิดและเทคนิคระดับสูงที่จำเป็นต่อการพัฒนาโปรแกรมที่มีความซับซ้อน มีประสิทธิภาพ และสามารถดูแลรักษาได้ในระยะยาว

เริ่มต้นด้วย **แนวคิดเชิงวัตถุ (Object-Oriented Programming)** ซึ่งเป็นพื้นฐานในการออกแบบซอฟต์แวร์สมัยใหม่ โดยผู้เรียนจะได้ศึกษาการสร้าง class และ object การใช้งาน access modifiers เพื่อควบคุมการเข้าถึงข้อมูล การเขียน constructor และ destructor รวมถึงการใช้ this pointer เพื่ออ้างอิงวัตถุภายใน class ของตนเอง ก่อนจะต่อยอดไปยัง **OOP ขั้นสูง** ซึ่งประกอบด้วยแนวคิดการสืบทอด (Inheritance), พหุนิยม (Polymorphism), ฟังก์ชันเสมือน (Virtual Functions), และคลาสนามธรรม (Abstract Class) ตลอดจนการห่อหุ้มข้อมูล (Encapsulation) และการซ่อนรายละเอียดการทำงานภายในวัตถุ

เนื้อหาถัดไปคือ **เทมเพลต (Template)** ซึ่งเป็นเครื่องมือสำคัญของ C++ ที่ช่วยให้เขียนโค้ดแบบทั่วไป (Generic Programming) ได้อย่างมีประสิทธิภาพ ไม่ว่าจะเป็น function templates หรือ class templates ซึ่งช่วยให้สามารถนำโค้ดเดียวกันไปใช้ซ้ำกับหลายชนิดข้อมูลได้อย่างปลอดภัยและยืดหยุ่น ตามด้วย **การจัดการข้อผิดพลาด (Exception Handling)** ที่มีบทบาทสำคัญในการสร้างโปรแกรมที่เสถียร โดยผู้อ่านจะได้ฝึกใช้ try, catch, throw และการสร้าง exception class ของตนเองเพื่อรองรับสถานการณ์ที่ไม่คาดคิด

ท้ายที่สุด หนังสือเล่มนี้จะพาผู้อ่านสู่ **Standard Template Library (STL)** ซึ่งเป็นคลังโครงสร้างข้อมูลและอัลกอริธึมที่ทรงพลังใน C++ ครอบคลุม container ประเภทต่าง ๆ เช่น vector, list, map, set, stack และ queue พร้อมทั้งวิธีใช้ iterators และฟังก์ชันสำเร็จรูปอย่าง sort(), find() รวมถึงการประยุกต์ lambda function กับ STL เพื่อเขียนโค้ดที่กระชับ ทันสมัย และมีประสิทธิภาพสูง

ผู้เขียนหวังเป็นอย่างยิ่งว่าหนังสือเล่มนี้จะช่วยเสริมสร้างความเข้าใจเชิงลึกในแนวคิดระดับสูงของภาษา C++ และเปิดประตูสู่การพัฒนาโปรแกรมขนาดใหญ่ การทำงานร่วมกับทีม และการเข้าสู่โลกของซอฟต์แวร์ในระดับมืออาชีพอย่างมั่นใจ

ด้วยความปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 11
การเขียนโปรแกรมเชิงวัตถุ
(Object-Oriented Programming)

เนื้อหา

- แนวคิดเชิงทฤษฎีเบื้องต้น
- UML กับการเขียนโปรแกรมเชิงวัตถุ (OOP)
- ขยายเนื้อหา UML กับการเขียนโปรแกรมเชิงวัตถุ (OOP)
- การเขียนโปรแกรมเชิงวัตถุ (OOP) ใน C++
- การเขียนโปรแกรมเชิงวัตถุ (OOP) พร้อมแสดงตัวอย่าง UML
- การสร้าง class และ object แบบละเอียด พร้อมเพิ่ม UML Class Diagram
- การสร้าง Class และ Object ในเชิงลึก
- Access Modifiers ในการเขียนโปรแกรมเชิงวัตถุ
- ขยายความเชิงลึกเรื่อง Access Modifiers ใน C++
- Constructor และ Destructor ในการเขียนโปรแกรมเชิงวัตถุ
- อธิบายเชิงลึกเรื่อง Constructor และ Destructor ใน C++
- การใช้ this pointer ใน C++
- บูรณาการเนื้อหา การเขียนโปรแกรมเชิงวัตถุ (OOP)

บทนำ

การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming: OOP) เป็นแนวคิดที่ทรงพลังในการออกแบบและพัฒนาซอฟต์แวร์ โดยเน้นการมองปัญหาและทางแก้ไขผ่าน “วัตถุ” ซึ่งมีคุณสมบัติ (attributes) และพฤติกรรม (behaviors) เป็นของตนเอง แนวคิดนี้ช่วยให้โปรแกรมมีโครงสร้างชัดเจน แยกส่วนได้ดี และง่ายต่อการบำรุงรักษา อีกทั้งยังสนับสนุนการเขียนโค้ดแบบนำกลับมาใช้ซ้ำ (reusability) และการขยายระบบอย่างมีประสิทธิภาพ

การเริ่มต้นเรียนรู้ OOP ในภาษา C++ มักเริ่มจากการสร้าง **class** ซึ่งเป็นแม่แบบของวัตถุ และการสร้าง **object** ซึ่งเป็นอินสแตนซ์ของ class นั้น ๆ โดย class จะกำหนดโครงสร้างของข้อมูลและพฤติกรรมที่เกี่ยวข้อง เช่น ฟังก์ชันสมาชิก (member functions) หรือข้อมูลภายใน (data members) การใช้ class และ object เป็นจุดเริ่มต้นสำคัญในการออกแบบโปรแกรมในเชิงวัตถุ

การควบคุมการเข้าถึงข้อมูลภายใน class เป็นอีกหัวใจสำคัญของ OOP ซึ่งสามารถจัดการได้ผ่าน **access modifiers** ได้แก่ public, private, และ protected โดย public อนุญาตให้เข้าถึงได้จากภายนอก class ส่วน private จำกัดการเข้าถึงเฉพาะภายใน class และ protected ให้สิทธิ์กับ class ลูกที่สืบทอดมาเท่านั้น การควบคุมนี้มีบทบาทในการซ่อนรายละเอียด (information hiding) และสร้างความปลอดภัยให้กับข้อมูล

การสร้างและทำลาย object อย่างเหมาะสมยังเป็นอีกส่วนที่สำคัญ โดยมี **constructor** สำหรับกำหนดค่าเริ่มต้นเมื่อมีการสร้างวัตถุ และ **destructor** สำหรับทำความสะอาดเมื่อวัตถุหมดอายุการใช้งาน การออกแบบ constructor หลายรูปแบบ (overloading) ช่วยให้การสร้าง object มีความยืดหยุ่นและเหมาะสมกับสถานการณ์ที่หลากหลาย

ท้ายที่สุด การใช้ **this pointer** ซึ่งเป็นตัวชี้ไปยัง object ปัจจุบันภายใน class ช่วยให้สามารถจัดการกับข้อมูลภายใน object ได้อย่างชัดเจน โดยเฉพาะในกรณีที่ต้องการอ้างอิงตัวแปรหรือฟังก์ชันภายในตัวเอง บทนี้จะช่วยปูพื้นฐานแนวคิดและการใช้งานองค์ประกอบหลักของ OOP เพื่อเตรียมพร้อมสำหรับการพัฒนาโปรแกรมที่มีโครงสร้างดี รองรับการเติบโต และนำไปสู่การเรียนรู้ขั้นสูงต่อไป

แนวคิดเชิงทฤษฎีเบื้องต้น

---

1. Encapsulation (การห่อหุ้ม)

- ช่วย **ปกป้องข้อมูล** ไม่ให้ถูกแก้ไขโดยตรงจากภายนอก class เพื่อป้องกันความผิดพลาดหรือการใช้งานผิดวิธี
- ส่งเสริมการออกแบบ interface ที่ชัดเจนผ่านเมธอด เช่น get และ set หรือ accessor/mutator methods
- **ตัวอย่างข้อดี:**
    - สามารถเปลี่ยนแปลงโครงสร้างข้อมูลภายในได้โดยไม่กระทบโค้ดที่ใช้งาน class นั้น
    - ป้องกันการเข้าถึงข้อมูลที่ไม่เหมาะสม เช่น ห้ามตั้งค่าค่าในช่วงที่ไม่ถูกต้อง

---

2. Abstraction (การซ่อนรายละเอียด)

- **มุ่งเน้นให้ผู้ใช้โฟกัสกับ “อะไร” มากกว่า “อย่างไร”**
- ทำให้การใช้วัตถุเป็นเรื่องง่ายโดยไม่ต้องสนใจรายละเอียดภายในซับซ้อน
- เช่น ในภาษา C++ อาจใช้ abstract class หรือ interface เพื่อกำหนดฟังก์ชันที่ต้องใช้โดยไม่ต้องระบุรายละเอียดของการทำงาน

---

3. Inheritance (การสืบทอด)

- ลดการเขียนโค้ดซ้ำ (code duplication) เพราะ subclass ได้รับทุกอย่างจาก superclass

- สามารถเพิ่มพฤติกรรมใหม่ หรือ override พฤติกรรมเดิมได้ (method overriding)
- **ข้อควรระวัง:**
    - การสืบทอดที่ไม่เหมาะสมอาจทำให้โครงสร้างโปรแกรมซับซ้อนและยากต่อการดูแลรักษา (Inheritance Hell)
    - ควรออกแบบ class hierarchy อย่างรอบคอบ

---

4. Polymorphism (พหุรูปร่าง)

- ช่วยให้โปรแกรม “ยืดหยุ่น” มากขึ้น เช่น ผ่าน **virtual functions** ใน C++ ทำให้สามารถเรียกใช้ฟังก์ชันเดียวกันแต่มีพฤติกรรมต่างกันตาม object ที่เรียก
- มี 2 แบบหลัก
    - **Compile-time polymorphism** (Function overloading, Operator overloading)
    - **Run-time polymorphism** (Virtual functions, Abstract classes)
- **ประโยชน์:** สร้างโค้ดที่เขียนในรูปแบบ generic มากขึ้น ลดการเขียนโค้ดซ้ำ

---

5. Constructor และ Destructor

- **Constructor**
    - ฟังก์ชันพิเศษที่จะถูกเรียกโดยอัตโนมัติเมื่อสร้าง object
    - ใช้กำหนดค่าเริ่มต้น หรือเตรียมตัว resource ต่าง ๆ ที่ object ต้องใช้
    - สามารถมี constructor หลายแบบ (overloading) เพื่อรองรับการสร้าง object ที่ต่างกันได้
- **Destructor**
    - ฟังก์ชันพิเศษที่จะถูกเรียกเมื่อ object หมดอายุ หรือถูกลบ (เช่น ใช้ delete ใน C++)
    - ใช้สำหรับคืน resource เช่น memory, file handles, database connections

---

6. this Pointer

- ตัวชี้ที่ชี้ไปยัง object ปัจจุบันที่เรียกใช้งาน method
- ใช้เพื่อแก้ไขความคลุมเครือ เช่น ในกรณีที่ตัวแปร parameter ชื่อซ้ำกับ attribute ของ class
- ช่วยให้ method สามารถเข้าถึง object ตัวเองได้อย่างชัดเจน

---

7. การออกแบบ Class ที่ดี

- ควรมี **หน้าที่เดียว (Single Responsibility Principle)**
- ขนาดของ class ไม่ควรใหญ่มากจนเกินไป ควรแบ่ง class ให้มีหน้าที่ชัดเจน
- ใช้ **Access Modifiers** อย่างเหมาะสมเพื่อควบคุมการเข้าถึงข้อมูลและพฤติกรรม
- พิจารณาใช้ **Composition** แทนการสืบทอดเมื่อต้องการความยืดหยุ่นมากกว่า

8. ตัวอย่างแนวคิด OOP ในชีวิตจริง

- **Class:** รถยนต์ (Car)
- **Object:** รถยนต์คันนี้ (Car object เช่น Honda Civic)
- **Attributes:** สี, ยี่ห้อ, รุ่น, ปีผลิต
- **Methods:** ขับเคลื่อน, เบรก, เปิดไฟหน้า

9. ประโยชน์ของ OOP ในงานพัฒนาซอฟต์แวร์

- ง่ายต่อการทำงานเป็นทีมเพราะแต่ละคนสามารถพัฒนาคลาสต่าง ๆ ที่แยกกันได้
- เพิ่มความน่าเชื่อถือของโค้ดเพราะ encapsulation ช่วยป้องกันข้อผิดพลาดจากการแก้ไขข้อมูลโดยตรง
- ง่ายต่อการแก้ไขและเพิ่มเติมฟีเจอร์ใหม่ ๆ ในระบบที่มีความซับซ้อน

UML กับการเขียนโปรแกรมเชิงวัตถุ (OOP)

1. UML คืออะไร?

- **UML** คือ ภาษามาตรฐานสำหรับการวาดแบบจำลอง (modeling language) เพื่อช่วยในการออกแบบและวิเคราะห์ระบบซอฟต์แวร์
- UML ถูกใช้ในการแสดงภาพโครงสร้างและพฤติกรรมของระบบในรูปแบบกราฟิก ทำให้ทีมพัฒนาสามารถเข้าใจภาพรวมของระบบได้ง่ายขึ้น
- UML ไม่ได้ผูกกับภาษาโปรแกรมใดภาษาหนึ่ง และช่วยให้การสื่อสารระหว่างนักพัฒนา นักวิเคราะห์ และผู้เกี่ยวข้องเป็นไปอย่างมีประสิทธิภาพ

2. ความสัมพันธ์ของ UML กับ OOP

- UML ถูกออกแบบมาเพื่อช่วยในการออกแบบ **ระบบเชิงวัตถุ (Object-Oriented System)**
- สามารถใช้ UML เพื่อสร้างแบบจำลองคลาส (class models) ซึ่งแสดงถึงคลาสต่าง ๆ ที่มีในระบบและความสัมพันธ์ระหว่างคลาส
- UML ยังช่วยแสดงพฤติกรรมของวัตถุผ่าน diagram ชนิดต่าง ๆ เช่น Use Case Diagram, Sequence Diagram, Activity Diagram เป็นต้น

3. ประเภทของ UML Diagrams ที่ใช้บ่อยใน OOP

ประเภท Diagram	จุดประสงค์หลัก	ใช้ใน OOP อย่างไร

ประเภท Diagram	จุดประสงค์หลัก	ใช้ใน OOP อย่างไร
Class Diagram	แสดงโครงสร้างคลาสและความสัมพันธ์	แสดงคลาส, attributes, methods และความสัมพันธ์ (inheritance, association)
Object Diagram	แสดง snapshot ของ object ในเวลาหนึ่ง	แสดงวัตถุจริงที่ถูกสร้างขึ้นจากคลาสในระบบ
Use Case Diagram	แสดงความต้องการของระบบจากมุมมองผู้ใช้	ระบุบทบาท (actor) และฟังก์ชันที่ระบบต้องทำ
Sequence Diagram	แสดงลำดับการส่งข้อความระหว่าง object	แสดงการโต้ตอบระหว่างวัตถุในเวลาที่เกิดขึ้น
Activity Diagram	แสดงลำดับของการทำงานหรือกิจกรรม	แสดง flow ของการทำงานในระบบ

---

4. องค์ประกอบสำคัญใน UML Class Diagram

- **Class**: เป็นสัญลักษณ์สี่เหลี่ยมที่แบ่งออกเป็น 3 ส่วน
    1. ชื่อคลาส (Class Name)
    2. คุณสมบัติ (Attributes) เช่น ตัวแปรของคลาส
    3. เมธอด (Operations/Methods) เช่น ฟังก์ชันในคลาส
- **Access Modifiers** แสดงโดยสัญลักษณ์ต่อหน้าชื่อ
    - + = public
    - - = private
    - # = protected
- **ความสัมพันธ์ระหว่างคลาส** เช่น
    - **Inheritance (Generalization)** : ใช้ลูกศรทิศทางจาก subclass ไป superclass
    - **Association** : ความสัมพันธ์ปกติ (เช่น คลาส A ใช้งานคลาส B)
    - **Aggregation** : ความสัมพันธ์แบบ "has-a" โดยวัตถุสามารถมีอยู่ได้เอง (empty diamond)
    - **Composition** : ความสัมพันธ์แบบ "part-of" ที่ส่วนประกอบไม่มีตัวตนหากไม่มีเจ้าของ (filled diamond)

---

5. ตัวอย่าง UML Class Diagram

+-----------------+

```
|    Person       |
+-----------------+
| - name: string  |
| - age: int      |
+-----------------+
| + getName(): string |
| + getAge(): int     |
+-----------------+

          ^
          |
+-----------------+
|    Student      |
+-----------------+
| - studentID: int|
+-----------------+
| + getID(): int  |
+-----------------+
```

- แสดงว่าคลาส Student สืบทอดจากคลาส Person
- - แสดงว่าช่องข้อมูลเป็น private
- + แสดงว่าฟังก์ชันเป็น public

---

6. การใช้ UML ช่วยใน OOP อย่างไร?

- ช่วยให้ออกแบบโครงสร้างคลาสได้อย่างชัดเจนก่อนลงมือเขียนโปรแกรม
- ช่วยตรวจสอบความสัมพันธ์และหน้าที่ของแต่ละคลาส (เช่น inheritance, association)
- ลดความซับซ้อนของโปรแกรมโดยการแยกแยะหน้าที่ของแต่ละคลาส
- ใช้เป็นเอกสารสำหรับทีมพัฒนาหรือสำหรับการบำรุงรักษาระบบในอนาคต

---

7. สรุป

หัวข้อ	สรุป
UML คือ	ภาษาสำหรับการวาดแบบจำลองระบบโดยเฉพาะระบบเชิงวัตถุ
ใช้กับ OOP	แสดงโครงสร้างคลาส ความสัมพันธ์ และพฤติกรรมของวัตถุในระบบ
ประโยชน์	ทำให้เข้าใจระบบง่ายขึ้น ออกแบบและวางแผนได้ดี ลดข้อผิดพลาด

หัวข้อ	สรุป
ตัวอย่างสำคัญ	Class Diagram, Use Case Diagram, Sequence Diagram

ขยายเนื้อหา UML กับการเขียนโปรแกรมเชิงวัตถุ (OOP)

8. รายละเอียดเชิงลึกของ UML ใน OOP

8.1 Class Diagram - โครงสร้างคลาสอย่างละเอียด

- **Attributes** (คุณสมบัติ) ใน UML จะบอกชนิดข้อมูลและ access modifier ตัวอย่างเช่น
- - id: int
- - name: string
- **Operations** (เมธอด) จะบอกชื่อฟังก์ชัน, พารามิเตอร์, และชนิดข้อมูลผลลัพธ์ เช่น
- + setName(name: string): void
- + getId(): int
- การกำหนดความสัมพันธ์
    - **Association**: ความสัมพันธ์ระหว่างคลาสสองตัว เช่น Person กับ Car ที่ Person อาจมี Car หลายคัน
    - **Multiplicity**: จำนวนความสัมพันธ์ เช่น 1..*, 0..1 แสดงว่ามี 1 หรือมากกว่า, หรือมีได้ 0 หรือ 1 ตัว
    - **Navigability**: ลูกศรบอกทิศทางของความสัมพันธ์ ว่าคลาสไหนสามารถเข้าถึงอีกคลาสได้

8.2 Use Case Diagram - มุมมองของผู้ใช้กับระบบ

- ใช้เพื่อแสดงฟังก์ชันหลัก ๆ ที่ระบบต้องรองรับ
- แสดง **actor** (ผู้ใช้งาน เช่น ผู้ใช้ ระบบอื่น)
- แสดง **use case** (ฟังก์ชันที่ระบบให้บริการ)
- ช่วยในการวางแผนและกำหนดขอบเขตของระบบ

8.3 Sequence Diagram - แสดงกระบวนการทำงานเชิงลำดับ

- แสดงการส่งข้อความ (method calls) ระหว่างวัตถุแต่ละตัว
- ช่วยในการออกแบบ flow ของโปรแกรมโดยละเอียด
- ตัวอย่างเช่น แสดงลำดับการ login ของผู้ใช้ในระบบ

8.4 Activity Diagram - แสดง flow ของกระบวนการ

- คล้าย flowchart แสดงขั้นตอนและทางเลือกของระบบ
- ช่วยวางแผนการทำงานโดยละเอียด เช่น ขั้นตอนการสั่งซื้อสินค้า

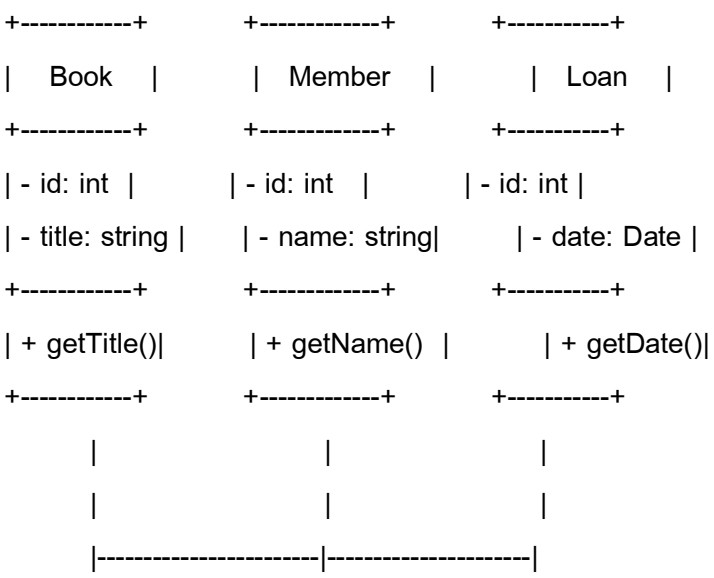
---

9. ตัวอย่าง UML ในการออกแบบโปรแกรมเชิงวัตถุ

สมมติว่าเราจะออกแบบระบบจัดการห้องสมุด โดยมีคลาสหลัก 3 ตัว คือ

- Book
- Member
- Loan

Class Diagram ตัวอย่าง



- Loan เป็นตัวกลางเชื่อม Member กับ Book เพื่อเก็บข้อมูลการยืม

---

10. ตัวอย่าง Use Case Diagram สำหรับระบบนี้

- Actor: Librarian, Member
- Use cases: Borrow Book, Return Book, Search Book, Register Member

---

11. ข้อดีของการใช้ UML กับ OOP

ข้อดี	อธิบาย
ช่วยวางแผน	ทำให้เห็นภาพรวมระบบก่อนเขียนโค้ดจริง
ลดข้อผิดพลาด	วางแผนโครงสร้างและความสัมพันธ์ได้ดี
สื่อสารได้ง่าย	ให้ทีมพัฒนาและผู้มีส่วนเกี่ยวข้องเข้าใจตรงกัน

ข้อดี	อธิบาย
รองรับการขยาย	UML ช่วยเตรียมระบบสำหรับการเพิ่มฟีเจอร์ใหม่

---

12. เครื่องมือวาด UML ที่นิยมใช้

- **StarUML**
- **Visual Paradigm**
- **PlantUML** (ใช้ภาษา text-based เขียน UML แล้วแปลงเป็นภาพ)
- **Microsoft Visio**
- **Lucidchart**

การเขียนโปรแกรมเชิงวัตถุ (OOP) ใน C++

---

11. การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming)

1. การสร้าง Class และ Object

- **Class** คือโครงสร้างข้อมูลที่ประกอบด้วยสมาชิกข้อมูล (Attributes) และฟังก์ชัน (Methods) เพื่อจัดการข้อมูลนั้น ๆ
- **Object** คืออินสแตนซ์ของ Class หรือสิ่งที่สร้างขึ้นจาก Class

ตัวอย่างการประกาศ Class และสร้าง Object:

```
#include <iostream>
using namespace std;

class Car {
public:   // สมาชิกที่เข้าถึงได้จากภายนอก
    string brand;
    int year;

    void showInfo() {
        cout << "Brand: " << brand << ", Year: " << year << endl;
    }
};

int main() {
    Car myCar;        // สร้าง object ชื่อ myCar จาก class Car
```

```
    myCar.brand = "Toyota";
    myCar.year = 2020;

    myCar.showInfo(); // เรียกใช้เมธอดแสดงข้อมูล

    return 0;
}
```

ผลการรัน:

Brand: Toyota, Year: 2020

---

2. Access Modifiers: public, private, protected

- **public** : สมาชิกสามารถเข้าถึงได้จากภายนอก class
- **private** : สมาชิกสามารถเข้าถึงได้เฉพาะภายใน class เท่านั้น
- **protected** : สมาชิกสามารถเข้าถึงได้ใน class และ class ที่สืบทอด (inheritance)

ตัวอย่างการใช้ Access Modifiers:

```
#include <iostream>
using namespace std;

class BankAccount {
private:
    double balance;    // private ไม่สามารถเข้าถึงจากภายนอก

public:
    BankAccount() {    // Constructor กำหนดค่าเริ่มต้น
        balance = 0.0;
    }

    void deposit(double amount) {
        if (amount > 0)
            balance += amount;
    }

    double getBalance() {
        return balance;
```

```
    }
};

int main() {
    BankAccount acc;
    acc.deposit(1500.50);
    // acc.balance = 1000; // ERROR! balance เป็น private

    cout << "Balance: " << acc.getBalance() << endl;

    return 0;
}
```

ผลการรัน:

```
Balance: 1500.5
```

---

3. Constructor และ Destructor

- **Constructor** คือฟังก์ชันพิเศษที่ชื่อเหมือน class ทำงานเมื่อสร้าง object เพื่อกำหนดค่าเริ่มต้น
- **Destructor** คือฟังก์ชันพิเศษที่ชื่อเป็น ~ClassName() เรียกใช้เมื่อลบ object เพื่อจัดการทรัพยากร (เช่น ปล่อยหน่วยความจำ)

ตัวอย่าง:

```
#include <iostream>
using namespace std;

class Person {
private:
    string name;

public:
    // Constructor
    Person(string n) {
        name = n;
        cout << "Constructor: Created Person " << name << endl;
    }
```

```
    // Destructor
    ~Person() {
        cout << "Destructor: Destroying Person " << name << endl;
    }

    void sayHello() {
        cout << "Hello, I am " << name << endl;
    }
};

int main() {
    Person p1("Alice");
    p1.sayHello();

    {
        Person p2("Bob");
        p2.sayHello();
    }  // p2 ถูกทำลายที่นี่ (Destructor เรียก)

    cout << "End of main function\n";

    return 0;
}
```

ผลการรัน:

```
Constructor: Created Person Alice
Hello, I am Alice
Constructor: Created Person Bob
Hello, I am Bob
Destructor: Destroying Person Bob
End of main function
Destructor: Destroying Person Alice
```

---

4. การใช้ this Pointer

- this คือ pointer ภายใน class ที่ชี้ไปยัง object ปัจจุบัน (instance)
- ใช้เพื่อเข้าถึงสมาชิกของ object ปัจจุบัน เช่น แก้ไขความสับสนระหว่างชื่อพารามิเตอร์และสมาชิก class

ตัวอย่าง:

```
#include <iostream>
using namespace std;

class Rectangle {
private:
    int width, height;

public:
    // Constructor ใช้ this เพื่อแยกสมาชิกและพารามิเตอร์
    Rectangle(int width, int height) {
        this->width = width;
        this->height = height;
    }

    int area() {
        return width * height;
    }

    void setWidth(int width) {
        this->width = width;  // แก้ไขสมาชิก width ของ object
    }
};

int main() {
    Rectangle rect(10, 5);
    cout << "Area: " << rect.area() << endl;

    rect.setWidth(20);
    cout << "New area: " << rect.area() << endl;
```

```
    return 0;
}
```

ผลการรัน:

```
Area: 50
New area: 100
```

การเขียนโปรแกรมเชิงวัตถุ (OOP) พร้อมแสดงตัวอย่าง UML

---

11. การเขียนโปรแกรมเชิงวัตถุ (OOP)

11.1 การสร้าง Class และ Object

- **Class** คือ แม่แบบ (template) ที่กำหนดโครงสร้างข้อมูล (attributes) และพฤติกรรม (methods หรือ functions) ของวัตถุ
- **Object** คือ อินสแตนซ์ (instance) ของคลาสนั้น ๆ ซึ่งเก็บข้อมูลจริงและสามารถเรียกใช้เมธอดได้

ตัวอย่าง UML Class Diagram ของ Car

```
+-------------------+
|       Car         |
+-------------------+
| - brand: string   |
| - model: string   |
| - year: int       |
+-------------------+
| + Car()           |
| + Car(brand, model, year) |
| + displayInfo()   |
+-------------------+
```

- เครื่องหมาย - แปลว่า **private**
- เครื่องหมาย + แปลว่า **public**

โค้ด C++ ตัวอย่าง

```
#include <iostream>
using namespace std;
```

```
class Car {
private:
    string brand;  // private attribute
    string model;
    int year;

public:
    // Constructor แบบไม่มีพารามิเตอร์
    Car() {
        brand = "Unknown";
        model = "Unknown";
        year = 0;
    }

    // Constructor แบบมีพารามิเตอร์
    Car(string b, string m, int y) {
        brand = b;
        model = m;
        year = y;
    }

    // ฟังก์ชันแสดงข้อมูล
    void displayInfo() {
        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << endl;
    }
};

int main() {
    Car car1;  // สร้าง object โดยใช้ default constructor
    Car car2("Toyota", "Camry", 2022);  // สร้าง object โดยใช้ constructor แบบมีพารามิเตอร์

    car1.displayInfo();
    car2.displayInfo();
```

```
    return 0;
}
```

11.2 Access Modifiers: public, private, protected

Modifier	การเข้าถึง
private	เข้าถึงได้เฉพาะในคลาสเท่านั้น (ข้อมูลส่วนตัว)
public	เข้าถึงได้จากทุกที่
protected	เข้าถึงได้ในคลาสและคลาสลูก (inheritance)

- ค่าเริ่มต้นของ access modifier ใน class คือ **private**
- ใน struct ค่าเริ่มต้นเป็น **public**

11.3 Constructor และ Destructor

- **Constructor** คือฟังก์ชันพิเศษที่ถูกเรียกเมื่อสร้าง object ใช้สำหรับกำหนดค่าเริ่มต้น
- **Destructor** คือฟังก์ชันพิเศษที่ถูกเรียกเมื่อ object ถูกทำลาย ใช้สำหรับจัดการหน่วยความจำหรือ resource อื่น ๆ

ตัวอย่างโค้ด Constructor และ Destructor

```
class Example {
public:
    Example() {  // Constructor
        cout << "Constructor called" << endl;
    }
    ~Example() {  // Destructor
        cout << "Destructor called" << endl;
    }
};

int main() {
    Example obj;  // สร้าง object --> Constructor ถูกเรียก
    // เมื่อ main สิ้นสุด obj จะถูกทำลาย --> Destructor ถูกเรียก
    return 0;
}
```

11.4 การใช้ this pointer

- this คือ pointer ที่ชี้ไปยัง object ปัจจุบัน
- ใช้เพื่ออ้างถึงสมาชิกของ object ในเมธอด เช่น กรณีที่พารามิเตอร์ชื่อเดียวกับ attribute เพื่อแก้ความสับสน

ตัวอย่างการใช้ this

```
class Person {
private:
    string name;

public:
    void setName(string name) {
        this->name = name;  // this->name คือ attribute, name คือพารามิเตอร์
    }
    void printName() {
        cout << "Name: " << name << endl;
    }
};

int main() {
    Person p;
    p.setName("John");
    p.printName();
    return 0;
}
```

สรุป UML + OOP

Concept	UML Symbol/Notation	C++ Code Element
Class	Rectangle with class name	class ClassName { ... };
Attributes	- name: type (private)	private: string name;
Operations (Methods)	+ method(params): returnType (public)	public: void method();
Constructor	<> method	ClassName() { ... }
Destructor	<> method	~ClassName() { ... }
Access Modifiers	-, +, # for private, public, protected	private:, public:

Concept	UML Symbol/Notation	C++ Code Element
Association	Line between classes	Object references / members

การสร้าง class และ object แบบละเอียด พร้อมเพิ่ม UML Class Diagram

การสร้าง Class และ Object พร้อม UML

1. Class คืออะไร?

- **Class** คือแบบแผน (template) หรือแม่แบบที่กำหนดโครงสร้างข้อมูล (attributes) และพฤติกรรม (methods) ของวัตถุ (object)
- เป็น "พิมพ์เขียว" ที่ใช้สร้าง object หลายตัวที่มีลักษณะและพฤติกรรมเหมือนกัน

2. Object คืออะไร?

- **Object** คืออินสแตนซ์ (instance) ของ class
- เป็นตัวแทนข้อมูลจริง ๆ ที่ถูกสร้างขึ้นจาก class
- แต่ละ object จะมีข้อมูลของตัวเองและสามารถเรียกใช้ method ของ class ได้

3. UML Class Diagram

```
+--------------------+
|       Person       |
+--------------------+
| - name: string     |
| - age: int         |
+--------------------+
| + display(): void  |
+--------------------+
```

คำอธิบาย UML:

- ชื่อคลาส Person อยู่บนสุด
- Attributes (ตัวแปรข้อมูล) แสดงในช่องกลาง ได้แก่ name (ชนิด string) และ age (ชนิด int)
- Methods (ฟังก์ชัน) แสดงในช่องล่าง ได้แก่ display() ซึ่งไม่มีการคืนค่า (void)
- เครื่องหมาย + หมายถึง public (เข้าถึงได้จากภายนอก)

4. ตัวอย่างโค้ด C++ พร้อมคำอธิบาย

```
#include <iostream>
```

```
using namespace std;

// ประกาศคลาส Person
class Person {
public:  // กำหนดให้สมาชิกที่ตามเป็น public เข้าถึงได้จากภายนอกคลาส
    string name;  // attribute ชื่อ name ประเภท string
    int age;      // attribute ชื่อ age ประเภท int

    // method แสดงข้อมูลชื่อและอายุ
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Person p1;       // สร้าง object p1 จากคลาส Person
    p1.name = "Alice";  // กำหนดชื่อให้ p1
    p1.age = 25;        // กำหนดอายุให้ p1
    p1.display();       // เรียกใช้ method display() ของ p1

    Person p2;       // สร้าง object p2 อีกตัว
    p2.name = "Bob";   // กำหนดชื่อ
    p2.age = 30;       // กำหนดอายุ
    p2.display();      // เรียก method display() ของ p2

    return 0;
}
```

คำอธิบายโค้ดทีละบรรทัด

- class Person { ... }; — ประกาศคลาสชื่อ Person
- public: — กำหนดสมาชิกที่ตามเป็น public เพื่อให้สามารถเข้าถึงได้จากภายนอกคลาส
- string name; — กำหนด attribute ชื่อ name ประเภท string
- int age; — กำหนด attribute ชื่อ age ประเภท int
- void display() — เมธอดแสดงข้อมูลของ object

- ใน main() — สร้าง object p1 และ p2 จาก class Person
- กำหนดค่า attribute ของแต่ละ object (name, age)
- เรียกใช้เมธอด display() เพื่อแสดงข้อมูลแต่ละ object

ผลลัพธ์ที่ได้จากโปรแกรม

Name: Alice, Age: 25

Name: Bob, Age: 30

สรุป

- Class คือแบบแผนของวัตถุที่กำหนด attribute และ method
- Object คืออินสแตนซ์ของ class ที่มีข้อมูลและพฤติกรรมของตัวเอง
- การสร้าง object ใน C++ ใช้ชื่อ class ตามด้วยชื่อตัวแปร
- UML ช่วยแสดงภาพโครงสร้างของ class ได้ชัดเจน

การสร้าง Class และ Object ในเชิงลึก

---

1. การซ่อนข้อมูล (Encapsulation)

- หลักการสำคัญของ OOP คือ **Encapsulation** คือการปกปิดข้อมูล (data hiding)
- ทำได้โดยกำหนด **access modifiers** เช่น
    - private: สมาชิกของคลาสที่ไม่อนุญาตให้เข้าถึงจากภายนอก คลาสอื่น ๆ หรือแม้แต่ object
    - public: สมาชิกที่อนุญาตให้เข้าถึงได้จากทุกที่
    - protected: สมาชิกที่อนุญาตให้เข้าถึงได้เฉพาะภายในคลาสและคลาสที่สืบทอด (inheritance)

เหตุผล:

ช่วยป้องกันการแก้ไขข้อมูลโดยตรงจากภายนอก คลาสควบคุมการเข้าถึงและการแก้ไขข้อมูลผ่าน method (getter/setter)

---

2. Constructor และ Destructor

- **Constructor** คือ ฟังก์ชันพิเศษในคลาสที่ถูกเรียกใช้เมื่อสร้าง object เพื่อกำหนดค่าเริ่มต้น
    - มีชื่อเดียวกับคลาส
    - ไม่มีชนิดผลลัพธ์ (void, int ฯลฯ)
    - สามารถมี parameter เพื่อรับค่าเริ่มต้นได้
- **Destructor** คือ ฟังก์ชันพิเศษที่ถูกเรียกใช้เมื่อ object ถูกทำลาย (เช่น เมื่อออกจาก scope)

- ชื่อเหมือนคลาสแต่มีเครื่องหมาย ~ ข้างหน้า
- ใช้เพื่อจัดการทรัพยากร เช่น ปล่อยหน่วยความจำ

---

3. การใช้ this pointer

- ภายใน method ของ class เราสามารถใช้ pointer ชื่อ this เพื่ออ้างถึง object ปัจจุบัน
- ประโยชน์ เช่น เมื่อต้องการแยกความแตกต่างระหว่างพารามิเตอร์กับ attribute ที่ชื่อเดียวกัน

---

4. การประกาศและกำหนดค่า Attribute

- Attribute ในคลาสสามารถกำหนดค่าเริ่มต้นได้ (ตั้งแต่ C++11 ขึ้นไป)
- การเข้าถึง attribute โดยตรงควรระมัดระวัง ควรใช้ setter/getter เพื่อควบคุม

---

5. การสร้าง Object

- Object สร้างได้ 2 แบบหลัก
  - **แบบ stack:**

    Person p1;

    สร้าง object ที่ stack memory, ถูกทำลายเมื่อออกจาก scope
  - **แบบ heap (dynamic allocation):**

    Person* p2 = new Person();

    สร้าง object บน heap, ต้องลบด้วย delete p2; เพื่อป้องกัน memory leak

---

6. การออกแบบ Class ที่ดี

- แยกส่วนรับผิดชอบ (Single Responsibility Principle)
- ใช้ access modifiers ควบคุมข้อมูล
- เขียน constructor และ destructor ให้เหมาะสม
- ใช้ getter/setter เพื่อควบคุมการเข้าถึง attribute
- พยายามทำ class ให้มี interface ที่ชัดเจนและใช้งานง่าย

ตัวอย่างโค้ดเชิงลึกที่แสดง constructor, destructor, this pointer และ encapsulation

```
#include <iostream>
using namespace std;

class Person {
private:
    string name;  // ซ่อนข้อมูลไม่ให้เข้าถึงโดยตรง
```

```
    int age;

public:
    // Constructor
    Person(string name, int age) {
        this->name = name;  // ใช้ this เพื่อแยก attribute กับ parameter
        this->age = age;
        cout << "Constructor called for " << name << endl;
    }

    // Destructor
    ~Person() {
        cout << "Destructor called for " << name << endl;
    }

    // Getter และ Setter
    void setName(string name) {
        this->name = name;
    }
    string getName() {
        return name;
    }

    void setAge(int age) {
        if (age >= 0) this->age = age;  // ป้องกันค่าที่ไม่สมเหตุสมผล
    }
    int getAge() {
        return age;
    }

    // Method แสดงข้อมูล
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
```

```
};

int main() {
    Person p1("Alice", 25);
    p1.display();

    // เปลี่ยนค่าโดยใช้ setter
    p1.setAge(26);
    p1.display();

    // สร้าง object แบบ dynamic
    Person* p2 = new Person("Bob", 30);
    p2->display();

    delete p2;  // ต้องลบ object ที่สร้างแบบ dynamic เพื่อป้องกัน memory leak

    return 0;
}
```

ผลการรันตัวอย่างนี้

```
Constructor called for Alice
Name: Alice, Age: 25
Name: Alice, Age: 26
Constructor called for Bob
Name: Bob, Age: 30
Destructor called for Bob
Destructor called for Alice
```

สรุปเพิ่ม

- private ช่วยซ่อนข้อมูล ป้องกันแก้ไขโดยตรงจากภายนอก
- constructor ช่วยกำหนดค่าเริ่มต้นเมื่อสร้าง object
- destructor ช่วยจัดการทรัพยากรเมื่อ object ถูกทำลาย
- this pointer ใช้เพื่ออ้างถึง object ปัจจุบันใน method
- การสร้าง object แบบ dynamic ต้องลบ (delete) เพื่อป้องกัน memory leak

ตัวอย่างโปรแกรม **Class Person** ที่มีการใช้

- การสร้าง class และ object
- Access modifiers (private, public)
- Constructor และ Destructor
- การใช้ this pointer

พร้อมกับ **แผนภาพ UML Class Diagram** และ **ผลการรันโปรแกรม** ให้ครบถ้วน

---

1. UML Class Diagram ของ Person

```
+--------------------+
|       Person       |
+--------------------+
| - name: string     |   // private attribute
| - age: int         |   // private attribute
+--------------------+
| + Person(name, age) |   // constructor (public)
| + ~Person()        |   // destructor (public)
| + setName(name)    |   // public method
| + getName(): string |
| + setAge(age)      |
| + getAge(): int    |
| + display()        |
+--------------------+
```

- - หมายถึง private
- + หมายถึง public

---

2. ตัวอย่างโค้ด C++ พร้อมคอมเมนต์

```
#include <iostream>
using namespace std;

class Person {
private:
    string name;  // ซ่อนข้อมูล ไม่ให้เข้าถึงโดยตรงจากภายนอก
    int age;
```

```
public:
    // Constructor: รับค่าชื่อและอายุมาเริ่มต้น attribute
    Person(string name, int age) {
        this->name = name;  // ใช้ this เพื่อแยก attribute กับ parameter
        this->age = age;
        cout << "Constructor called for " << name << endl;
    }

    // Destructor: เรียกใช้เมื่อ object ถูกทำลาย
    ~Person() {
        cout << "Destructor called for " << name << endl;
    }

    // Setter สำหรับ name
    void setName(string name) {
        this->name = name;
    }

    // Getter สำหรับ name
    string getName() {
        return name;
    }

    // Setter สำหรับ age พร้อมตรวจสอบเงื่อนไขอายุ >=0
    void setAge(int age) {
        if (age >= 0) this->age = age;
    }

    // Getter สำหรับ age
    int getAge() {
        return age;
    }
```

```
    // แสดงข้อมูลของ Person
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    // สร้าง object p1 โดยเรียก constructor
    Person p1("Alice", 25);
    p1.display();

    // เปลี่ยนอายุโดยใช้ setter แล้วแสดงใหม่
    p1.setAge(26);
    p1.display();

    // สร้าง object แบบ dynamic
    Person* p2 = new Person("Bob", 30);
    p2->display();

    // ลบ object dynamic เพื่อเรียก destructor
    delete p2;

    return 0;
}
```

---

3. ผลการรันโปรแกรม

```
Constructor called for Alice
Name: Alice, Age: 25
Name: Alice, Age: 26
Constructor called for Bob
Name: Bob, Age: 30
Destructor called for Bob
Destructor called for Alice
```

สรุป

- การสร้าง class Person มี attribute ซ่อนอยู่ (private)
- ใช้ constructor กำหนดค่าเริ่มต้นตอนสร้าง object
- ใช้ this pointer ใน constructor และ setter เพื่ออ้างอิง attribute
- มี method getter/setter สำหรับเข้าถึงข้อมูลอย่างปลอดภัย
- มี destructor ที่ทำงานเมื่อลบ object หรือ object หมด scope
- สร้าง object ทั้งแบบ stack (p1) และ heap (p2) พร้อมลบ object แบบ dynamic

---

ตัวอย่างโปรแกรม 3 ตัวอย่าง พร้อม

- อธิบายรายละเอียดการทำงานของโค้ดทีละบรรทัด
- แสดง UML Diagram แบบข้อความ
- แสดงผลการรันตัวอย่าง

ตัวอย่างที่ 1: Class Car

UML Diagram

```
+---------------------+
|        Car          |
+---------------------+
| - brand: string     |
| - year: int         |
+---------------------+
| + Car(brand, year)  |
| + ~Car()            |
| + getBrand(): string |
| + getYear(): int    |
| + setBrand(string)  |
| + setYear(int)      |
| + display(): void   |
+---------------------+
```

โค้ด C++

```
#include <iostream>
using namespace std;

class Car {
```

```
private:
    string brand;  // ยี่ห้อรถ
    int year;      // ปีที่ผลิต

public:
    // Constructor: กำหนดค่าเริ่มต้นให้ brand และ year เมื่อสร้าง object
    Car(string brand, int year) {
        this->brand = brand;
        this->year = year;
        cout << "Car created: " << brand << " (" << year << ")" << endl;
    }

    // Destructor: เรียกเมื่อ object ถูกทำลาย
    ~Car() {
        cout << "Car destroyed: " << brand << endl;
    }

    // Getter สำหรับ brand
    string getBrand() {
        return brand;
    }

    // Getter สำหรับ year
    int getYear() {
        return year;
    }

    // Setter สำหรับ brand
    void setBrand(string brand) {
        this->brand = brand;
    }

    // Setter สำหรับ year (ตรวจสอบให้ปีมากกว่า 1885 ก่อน)
    void setYear(int year) {
```

```
        if (year > 1885)
            this->year = year;
    }

    // แสดงข้อมูลรถ
    void display() {
        cout << "Brand: " << brand << ", Year: " << year << endl;
    }
};

int main() {
    Car c1("Toyota", 2020);  // สร้าง object c1
    c1.display();            // แสดงข้อมูล c1

    c1.setYear(2021);        // เปลี่ยนปีเป็น 2021
    c1.display();            // แสดงข้อมูลใหม่

    return 0;
}
```

อธิบายโค้ดทีละบรรทัด

- class Car { ... }; — ประกาศคลาสชื่อ Car
- private: — ระบุว่า members ต่อไปนี้เป็นส่วนตัว (ไม่สามารถเข้าถึงจากภายนอก)
- string brand; และ int year; — ตัวแปรข้อมูลของคลาส
- public: — members ต่อไปนี้สามารถเข้าถึงได้จากภายนอก
- Car(string brand, int year) — Constructor ใช้กำหนดค่าเริ่มต้นตอนสร้าง object
- ~Car() — Destructor เรียกเมื่อ object ถูกลบ
- string getBrand() และ int getYear() — ฟังก์ชัน getter สำหรับดึงค่าตัวแปร
- void setBrand(string brand) และ void setYear(int year) — ฟังก์ชัน setter สำหรับแก้ไขค่าตัวแปร (setYear ตรวจสอบปีให้ถูกต้องก่อน)
- void display() — ฟังก์ชันแสดงข้อมูลรถ
- ใน main(), สร้าง object c1 จากคลาส Car และเรียกใช้ฟังก์ชันต่าง ๆ

ผลการรัน

```
Car created: Toyota (2020)
Brand: Toyota, Year: 2020
Brand: Toyota, Year: 2021
Car destroyed: Toyota
```

ตัวอย่างที่ 2: Class Rectangle

UML Diagram

```
+----------------------+
|      Rectangle       |
+----------------------+
| - width: double      |
| - height: double     |
+----------------------+
| + Rectangle(w,h)     |
| + ~Rectangle()       |
| + setWidth(double)   |
| + setHeight(double)  |
| + getWidth(): double |
| + getHeight(): double|
| + area(): double     |
| + display(): void    |
+----------------------+
```

โค้ด C++

```
#include <iostream>
using namespace std;

class Rectangle {
private:
    double width;  // ความกว้าง
    double height;  // ความสูง

public:
    // Constructor กำหนดค่าเริ่มต้น
```

```
Rectangle(double width, double height) {
    this->width = width;
    this->height = height;
}

// Destructor
~Rectangle() {
    cout << "Rectangle object destroyed" << endl;
}

// Setter กำหนดความกว้าง (ถ้าค่ามากกว่า 0)
void setWidth(double width) {
    if (width > 0)
        this->width = width;
}

// Setter กำหนดความสูง (ถ้าค่ามากกว่า 0)
void setHeight(double height) {
    if (height > 0)
        this->height = height;
}

// Getter ดึงค่าความกว้าง
double getWidth() {
    return width;
}

// Getter ดึงค่าความสูง
double getHeight() {
    return height;
}

// คำนวณพื้นที่
double area() {
```

```
        return width * height;
    }

    // แสดงข้อมูล
    void display() {
        cout << "Width: " << width << ", Height: " << height
             << ", Area: " << area() << endl;
    }
};

int main() {
    Rectangle r1(5.0, 3.0);  // สร้าง object r1
    r1.display();

    r1.setWidth(7.5);        // เปลี่ยนความกว้าง
    r1.setHeight(4.2);       // เปลี่ยนความสูง
    r1.display();

    return 0;
}
```

อธิบายโค้ดทีละบรรทัด

- ประกาศคลาส Rectangle พร้อมตัวแปรส่วนตัว width, height
- Constructor รับพารามิเตอร์กำหนดค่าความกว้างและสูง
- Destructor แจ้งเมื่อ object ถูกทำลาย
- ฟังก์ชัน setter และ getter สำหรับความกว้างและสูง พร้อมตรวจสอบค่าก่อนตั้ง
- ฟังก์ชัน area() คำนวณพื้นที่
- ฟังก์ชัน display() แสดงข้อมูลทั้งหมด
- ใน main() สร้าง r1 และแสดงผลก่อนและหลังเปลี่ยนขนาด

ผลการรัน

```
Width: 5, Height: 3, Area: 15
Width: 7.5, Height: 4.2, Area: 31.5
Rectangle object destroyed
```

ตัวอย่างที่ 3: Class Student

UML Diagram

```
+-----------------------+
|        Student        |
+-----------------------+
| - name: string        |
| - id: int             |
+-----------------------+
| + Student(name, id)   |
| + ~Student()          |
| + setName(string)     |
| + getName(): string   |
| + setId(int)          |
| + getId(): int        |
| + display(): void     |
+-----------------------+
```

โค้ด C++

```
#include <iostream>
using namespace std;

class Student {
private:
    string name;  // ชื่อนักเรียน
    int id;       // รหัสนักเรียน

public:
    // Constructor กำหนดชื่อและรหัส
    Student(string name, int id) {
        this->name = name;
        this->id = id;
    }

    // Destructor
```