Data Abstraction ด้วย C++

เทพพิทักษ์ การุญบุญญานันท์

Encapsulation

```
class Car {
public:
   Car();   // constructor
   ~Car(); // destructor

   void forward();
   void turnLeft();

private:
   int     maxSpeed;
   Vector curVelocity;
   Vector curPos;
};
```

Encapsulation

C++ : **`class`** คือ **`struct`** ที่

- มีสมาชิกเป็นฟังก์ชันได้ (ไม่ใช่แค่ข้อมูล)
- มีการควบคุมการเข้าถึง (access control):
  - **`private`** : เฉพาะฟังก์ชันของคลาสเข้าถึงได้
  - **`protected`** : เฉพาะฟังก์ชันของคลาสและ derived class เข้าถึงได้
  - **`public`** : เข้าถึงได้ทั่วไป
- หากไม่ระบุ access specifier จะหมายถึง **`private`**
- ฟังก์ชันสมาชิกพิเศษ:
  - Constructor (เช่น **`Car()`**) : เรียกเมื่อสร้างออบเจกต์
  - Destructor (เช่น **`~Car()`**) : เรียกเมื่อทำลายออบเจกต์

ตัวอย่าง Class

```
class Box {
  int left, top, right, bottom;

public:
  void set (int x1, int y1, int x2, int y2);
  int  area() { return (right - left) * (bottom - top); }
};

void Box::set (int x1, int y1, int x2, int y2)
{
  left = x1;  top = y1;
  right = x2; bottom = y2;
}
```

ตัวอย่าง Class

คลาส **Box**

- มีสมาชิกข้อมูล 4 ตัว เข้าถึงจากภายนอกไม่ได้
- มีสมาชิกฟังก์ชัน 2 ตัว เข้าถึงจากภายนอกได้
- การประกาศสมาชิกฟังก์ชันภายในการประกาศคลาส
  - ประกาศ prototype → ต้องกำหนด body ข้างนอก
  - ระบุ body ด้วย → เป็น inline function
- การอ้างถึงสมาชิกของคลาส → ใช้ *scope operator* `::` ระบุ class scope เช่น **`Box::set`**
- การอ้างถึงสมาชิกของออบเจกต์จากฟังก์ชันสมาชิก → อ้างชื่อได้โดยตรง เช่น **`left`**, **`top`**, **`right`**, **`bottom`**

ตัวอย่าง Class

การใช้งาน

```
int main ()
{
  Box b;
  b.set (0, 0, 5, 6);
  cout << "Area = " << b.area() << endl;
  return 0;
}
```

- **b** เป็นออบเจกต์ที่สร้างจากคลาส **Box**
- เข้าถึงสมาชิกของออบเจกต์ด้วย operator . เหมือนที่ใช้กับ **struct** เช่น **b.set** และ **b.area**

Constructor และ Destructor

- ตัวอย่างข้างต้น จะเกิดอะไรขึ้นถ้าเรียก **`b.area()`** ก่อน **`b.set()`** ?
- การสร้างออบเจกต์ควรมีการกำหนดค่าเริ่มต้นเสมอ
  $\rightarrow$ ใช้ constructor
- หากออบเจกต์ที่สร้างมีการสร้างออบเจกต์อื่นเพิ่มเติมล่ะ?
- เมื่อออบเจกต์หมดอายุขัย ควรทำลายออบเจกต์ที่เกี่ยวข้องให้เรียบร้อย
  $\rightarrow$ ใช้ destructor

ตัวอย่าง Constructor

```
class Box {
  int left, top, right, bottom;

public:
  Box (int x1, int y1, int x2, int y2);
  int  area() { return (right - left) * (bottom - top); }
};

Box::Box (int x1, int y1, int x2, int y2)
{
  left = x1;  top = y1;
  right = x2; bottom = y2;
}
```

ตัวอย่าง Constructor

การใช้งาน

```
int main ()
{
  Box b (0, 0, 5, 6);
  cout << "Area = " << b.area() << endl;
  return 0;
}
```

- constructor มีชื่อเดียวกับคลาส และไม่มีชนิดรีเทิร์น
- การประกาศอินสแตนซ์ของคลาส สามารถส่งพารามิเตอร์ให้ constructor ได้โดยเพิ่มวงเล็บและพารามิเตอร์ต่อท้าย

การโอเวอร์โหลด Constructor

```
class Box {
  int left, top, right, bottom;

public:
  Box() { left = top = right = bottom = 0; }
  Box (int x1, int y1, int x2, int y2);
  int  area() { return (right - left) * (bottom - top); }
};

Box::Box (int x1, int y1, int x2, int y2)
{
  left = x1;  top = y1;
  right = x2; bottom = y2;
}
```

การโอเวอร์โหลด Constructor

การใช้งาน

```
int main ()
{
  Box b1;
  Box b2 (0, 0, 5, 6);
  cout << "Area #1 = " << b1.area() << endl;
  cout << "Area #2 = " << b2.area() << endl;
  return 0;
}
```

- constructor สามารถโอเวอร์โหลดได้เหมือนฟังก์ชันทั่วไป
- constructor จะถูกเลือกเรียกตามรูปแบบการประกาศ

การสร้างออบเจกต์ผ่าน new และ delete

```
int main ()
{
  Box* pBox1 = new Box;
  Box* pBox2 = new Box (0, 0, 5, 6);
  cout << "Area #1 = " << pBox1->area() << endl;
  cout << "Area #2 = " << pBox2->area() << endl;
  delete pBox1;
  delete pBox2;
  return 0;
}
```

พอยน์เตอร์ this

this เมื่ออ้างใน member function หมายถึงพอยน์เตอร์ไปยังตัวออบเจกต์ที่กำลังเรียก

```
class Box {
  int x1, y1, x2, y2;
public:
  Box() { x1 = y1 = x2 = y2 = 0; }
  Box (int x1, int y1, int x2, int y2) {
    this->x1 = x1;  this->y1 = y1;
    this->x2 = x2;  this->y2 = y2;
  }
  int  area() { return (x2 - x1) * (y2 - y1); }
};
```

Initialization List ใน Constructor

```
class Box {
  int x1, y1, x2, y2;

public:
  Box() : x1 (0), y1 (0), x2 (0), y2 (0) {}
  Box (int x1, int y1, int x2, int y2)
    : x1 (x1), y1 (y1), x2 (x2), y2 (y2) {}
  int  area() { return (x2 - x1) * (y2 - y1); }
};
```

การใช้ Default Argument ใน Constructor

```
class Box {
  int x1, y1, x2, y2;

public:
  Box (int x1 = 0, int y1 = 0, int x2 = 0, int y2 = 0)
    : x1 (x1), y1 (y1), x2 (x2), y2 (y2) {}
  int  area() { return (x2 - x1) * (y2 - y1); }
};
```

- constructor สามารถใช้ default argument ได้เหมือนฟังก์ชันทั่วไป
- constructor จะถูกเลือกเรียกตามรูปแบบการประกาศ

การ Initialize สมาชิกที่เป็นออบเจกต์

```
class String {
  // ...
public:
  String (const char* s);
};

class Box {
  int x1, y1, x2, y2;
  String label;
public:
  Box (int x1 = 0, y1 = 0, x2 = 0, y2 = 0,
       const char* label = "")
    : x1 (x1), y1 (y1), x2 (x2), y2 (y2), label (label) { }
  int  area() { return (x2 - x1) * (y2 - y1); }
};
```

Constructor ที่มีอาร์กิวเมนต์เดียว

```
class String {
  // ...
public:
  String (const char *s);
};

String s = "Hello"; // same as: String s ("Hello");

void PrintString (String s);
// same as: PrintString (String ("Hello"));
PrintString ("Hello");
```

- constructor ที่มีอาร์กิวเมนต์เดียว ใช้แปลงชนิดได้

การก๊อปปี้ออบเจกต์

- การ assign ออบเจกต์ = การก๊อปปี้ member รายตัว

```
Box a (0, 0, 10, 10);
Box b = a;
```

- หากต้องมีการกระทำพิเศษเพิ่มเติมในการก๊อปปี้
  → สร้าง Copy Constructor

```
class Box {
  // ...
public:
  Box (const Box& b); // copy constructor
  // ...
};
```

Destructor

- รูปแบบ: `~`⟨*ชื่อคลาส*⟩ `()`
  - ไม่มี return type
  - ไม่รับอาร์กิวเมนต์
  - ไม่มีการโอเวอร์โหลด
- เรียกทำงานขณะที่ออบเจกต์จะถูกทำลาย
  - สิ้นสุด scope การประกาศ (สำหรับออบเจกต์ในสแต็ก)
  - ถูกทำลายด้วย **`delete`** (สำหรับออบเจกต์ในฮีป)
  - เมื่อออบเจกต์ชั่วคราว (temporary object) ถูกทำลาย
- ใช้สำหรับทำลายออบเจกต์อื่นที่อาจมีการสร้างเพิ่มเติมในตัวออบเจกต์นั้น

Destructor

```
class String {
public:
  String (const char* s) {
    str = new char[strlen (s) + 1];
    strcpy (str, s);
  }
  ~String() { delete[] str; }

  const char* get() { return str; }

private:
  char* str;
};
```

Destructor

```
int main ()
{
  String* p = new String ("Hello.");
  cout << p->get() << endl;
  delete p; // *p is destructed here

  String s ("Bye.");
  cout << s.get() << endl;

  cout << String("See you.").get() << endl;
  // temporary object is destructed here

  return 0; // s is destructed here
}
```

ออบเจกต์ชั่วคราว

ออบเจกต์ชั่วคราวเกิดขึ้นเมื่อ:

- เป็นค่าระหว่างการคำนวณนิพจน์

```
int x = a * b + c; // temp (a * b)
```

```
String s1 ("Good");
String s2 ("Morning");
// assuming operator '+' is overloaded
cout << (s1 + s2).get();
```

ออบเจกต์ชั่วคราว

ออบเจกต์ชั่วคราวเกิดขึ้นเมื่อ:

- สร้างจากการเรียก constructor ลอย ๆ

```
cout << String("See you.").get() << endl;
```

ออบเจกต์ชั่วคราว

ออบเจกต์ชั่วคราวเกิดขึ้นเมื่อ:

- เป็นอาร์กิวเมนต์ของฟังก์ชัน

```
int FindChar (String s, char c);

String s ("Hello");
int idx1 = FindChar (s, 'e');
int idx2 = FindChar ("Goodbye", 'e');
int idx3 = FindChar (String ("Goodbye"), 'e');
```

ออบเจกต์ชั่วคราว

ออบเจกต์ชั่วคราวเกิดขึ้นเมื่อ:

- เป็นค่ารีเทิร์นของฟังก์ชัน

```
String Head (const String& s, int n);

String s ("Hello");
cout << Head (s, 2) << endl;
```

ออบเจกต์ชั่วคราว

ออบเจกต์ชั่วคราวเกิดขึ้นเมื่อ:

- เป็นค่าเริ่มต้นให้กับ const reference

```
String Head (const String& s, int n);

const String& s = Head ("Hello", 2);
...
```

ออบเจกต์ชั่วคราว

อายุขัยของออบเจกต์ชั่วคราว:

- เมื่อเกิดระหว่างคำนวณนิพจน์: หมดอายุขัยเมื่อจบการคำนวณนิพจน์
- เมื่อเป็นค่าเริ่มต้นให้ const ref: อายุขัย = อายุขัยของ reference

Const Member Function

- **`const`** เป็น keyword ของ C และ C++ ที่ใช้บ่งชี้ข้อมูลหนึ่ง ๆ ว่าเป็นค่าที่จะไม่ถูกเปลี่ยนแปลง
- ถ้าคอมไพเลอร์พบความพยายามเปลี่ยนแปลงในข้อมูล **`const`** $\rightarrow$ error
- **`const`** จึงใช้ในการประกาศ:
  - พอยน์เตอร์ที่ชี้ข้อมูลแบบ read-only
  - reference ที่อ้างอิงข้อมูลแบบ read-only
  - member function ที่จะเข้าใช้ออบเจกต์แบบ read-only (**`this`** เป็น const pointer)

Const Member Function

```
class String {
public:
  String (const char* s) {
    str = new char[strlen (s) + 1];
    strcpy (str, s);
  }
  ~String() { delete[] str; }

  const char* get() const { return str; }

private:
  char* str;
};
```

Const Member Function

ตัวอย่างการใช้งาน:

```
void PrintString (const String& s)
{
  cout << s.get() << endl;
}
```

- อาร์กิวเมนต์ **s** อ้างอิงถึงออบเจกต์ที่เป็น **const**
- ถ้าฟังก์ชัน **String::get()** ไม่เป็น const member → error!

Static Member

Static Member Data:

- มีเพียง 1 copy ต่อ 1 คลาส
- ไม่เป็นสมาชิกในออบเจกต์ใดของคลาสนั้น
- ทุกออบเจกต์ของคลาสนั้นสามารถเข้าใช้ร่วมกันได้
- เรียกว่า *class variable*

การประกาศ:

- การประกาศในคลาสเป็นเพียง declaration ไม่มีการสร้างตัวแปร
- ต้อง define ตัวแปร ที่ใดที่หนึ่งนอกการประกาศคลาส

Static Member

```
class Transaction {
public:
  Transaction (int delta)
    : id (NextID++), delta (delta) {}
private:
  static int NextID;
private:
  int id;
  int delta;
};

int Transaction::NextID = 1;
```

Static Member

Static Member Function:

- เป็นฟังก์ชันที่ใช้ร่วมกันทั้งคลาส
- ไม่เป็น method สำหรับกระทำกับออบเจกต์ใดของคลาสนั้น (ไม่มีพอยน์เตอร์ **`this`**)
  → ไม่สามารถเข้าถึง non-static member ได้

Static Member

```
class Transaction {
public:
  Transaction (int delta)
    : id (NextID++), delta (delta) {}
private:
  static int NextID;
  static int GetCount() { return NextID - 1; }
private:
  int id;
  int delta;
};

int Transaction::NextID = 1;
```

กรณีตัวอย่าง: Time

```
class Time {
public:
  Time (int hour, int minute, int second);

  int getHour() const;
  int getMinute() const;
  int getSecond() const;

  Time& addHour (int hours);
  Time& addMinute (int minutes);
  Time& addSecond (int seconds);

  void print() const; // HH:MM:SS
};
```

กรณีตัวอย่าง: Time

Utility functions:

```
int DiffSeconds (Time a, Time b); // b - a in seconds
```

กรณีตัวอย่าง: Time

ตัวอย่างการใช้งาน

```
int main()
{
  Time start (8, 15, 30);
  Time end = start;
  end.addHour(8).addMinute(30).addSecond(2);

  cout << "Start: "; start.print();
  cout << "End: ";   end.print();
  cout << "Diff = " << DiffSeconds (start, end) << endl;

  return 0;
}
```

กรณีตัวอย่าง: Time

```
class Time {
  // ...
private:
  int hour;
  int minute;
  int second;
};

inline Time::Time (int hour, int minute, int second)
  : hour (hour), minute (minute), second (second) {}

inline int Time::getHour() const   { return hour; }
inline int Time::getMinute() const { return minute; }
inline int Time::getSecond() const { return second; }
```

กรณีตัวอย่าง: Time

```
Time& Time::addHour (int hours)
{
  hour += hours;
  return *this;
}
```

กรณีตัวอย่าง: Time

```
Time& Time::addMinute (int minutes)
{
  minute += minutes;
  if (minute >= 60) {
    addHour (minute / 60);
    minute %= 60;
  }
  return *this;
}
```

กรณีตัวอย่าง: Time

```
Time& Time::addSecond (int seconds)
{
  second += seconds;
  if (second >= 60) {
    addMinute (second / 60);
    second %= 60;
  }
  return *this;
}
```

กรณีตัวอย่าง: Time

```
void Time::print() const
{
  cout << hour << ':' << minute << ':' << second << endl;
}

int DiffSeconds (Time a, Time b)
{
  return (b.getHour() - a.getHour()) * 3600
         + (b.getMinute() - a.getMinute()) * 60
         + (b.getSecond() - a.getSecond());
}
```

การบ้าน: Date

```
class Date {
public:
  enum Month { JAN, FEB, MAR, APR, MAY, JUN,
               JUL, AUG, SEP, OCT, NOV, DEC };
  Date (int day, Month month, int year);

  int   getDay() const;
  Month getMonth() const;
  int   getYear() const;
  Date& addDay (int days);
  Date& addMonth (int months);
  Date& addYear (int years);
  int   daysUpTo (Date d);
  void  print() const; // e.g. 18 Sep 2015
};
```

การบ้าน: Date

ตัวอย่างการใช้งาน

```
int main()
{
  Date start (18, Date::SEP, 2015);
  Date end = start;
  end.addYear(7).addMonth(5).addDay(2);

  cout << "Start: "; start.print();
  cout << "End: ";   end.print();
  cout << "Diff = " << start.daysUpTo (end) << endl;

  return 0;
}
```

การบ้าน: Date

ข้อควรระวังในการบวกวัน:

- แต่ละเดือนมีจำนวนวันไม่เท่ากัน
  (ไม่เหมือน **`Time`** ที่ทุกนาทีมีจำนวนวินาทีเท่ากัน)
- เดือนกุมภาพันธ์ของแต่ละปีมีจำนวนวันไม่เท่ากัน
- เงื่อนไขของปีอธิกสุรทิน (กุมภาพันธ์มี 29 วัน):

ค.ศ. หารด้วย 4 ลงตัว แต่หารด้วย 100 ไม่ลงตัว
แต่ถ้าหารด้วย 400 ลงตัว ถือเป็นปีอธิกสุรทิน

- ตัวอย่างของปีอธิกสุรทิน: 1980, 2000, 2012, 2400, 2404
- ตัวอย่างของปีปกติสุรทิน: 1900, 1998, 2015, 2100, 2200
- การบวกปีและเดือน ให้บวกโดยไม่ต้องขยับวันที่