การเขียนโปรแกรมเชิงวัตถุ
Object-Oriented Programming

ทัศนวรรณ ศูนย์กลาง

ภาควิชาคอมพิวเตอร์ คณะวิทยาสตร์ มหาวิทยาลัยศิลปากร

หัวข้อที่จะเรียน

- Paradigm Evolution
- Abstract Data Types
- Categories of OOP Language
- Characteristics of OOP
- Imperative VS Object-oriented
- Object-oriented concept
- UML

Paradigm Evolution

- Procedural 1950s -1970s (procedural abstraction)
- Data-Oriented early 1980s (data-oriented/ADTs)
- Object-Oriented late 1980s (Inheritance & dynamic binding)
- Procedural abstraction
  - กระบวนการที่อนุญาตให้โปรแกรมเมอร์สนใจแค่ interface ระหว่างฟังก์ชันและรู้ว่ามันทำงานอะไร โดยไม่ต้องจำเป็นต้องสนใจรายละเอียดว่าทำงานได้อย่างไร
  - Sort(list, len)

Abstract Data Types (ADTs)

- เป็นชนิดข้อมูลที่สร้างขึ้นโดยผู้เขียนโปรแกรม (user defined data type)
- สร้างโดยใช้ structure ที่มีในภาษา imperative
- เพื่อให้ชนิดข้อมูลนี้ใช้งานได้อย่างปลอดภัย ควรมีลักษณะ
  - รายละเอียดของการเก็บข้อมูลต้องไม่มีความสำคัญต่อการใช้งานชนิดข้อมูล
  - ข้อมูลภายในจะไม่ถูกอ้างถึงได้โดยตรง แต่ผ่านทาง operations ที่กำหนดไว้
- ตัวอย่างเช่น ภาษา Modula-2, Ada --> object-based

Abstract Data Types

- Build-in type ก็คือ ADTs นั่นเอง
- ***float*** ในภาษา Java
  - มองไม่เห็นการแทนข้อมูล
  - Operation ทุกตัวเป็น build-in
  - ผู้ใช้สามารถกำหนด object เป็นชนิดข้อมูล ***float*** ได้
- User-defined ADTs จะต้องมีลักษณะแบบเดียวกับ Build-in ADTs

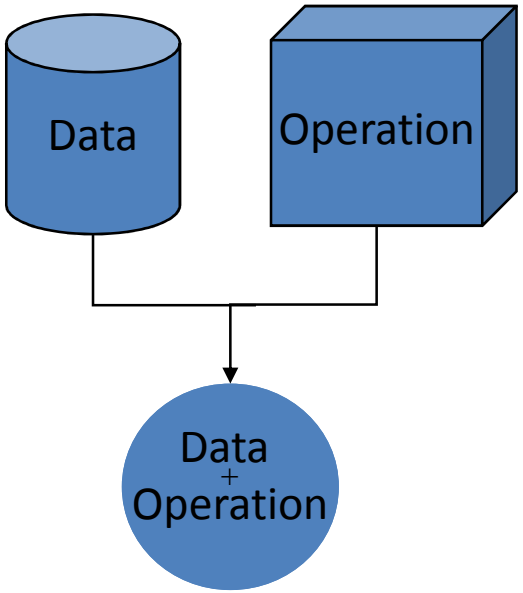
Abstract Data Types

- ภาษาที่สร้าง ADT ได้ต้องมีกลไก 2 อย่าง คือ
  - ***Encapsulation*** เพื่อนำข้อมูลกับ operations มาผูกติดกันเป็นหน่วยหนึ่ง
  - ***Information hiding*** เพื่อกำหนดขอบเขตในการอ้างถึงข้อมูลหรือ operations ของชนิดข้อมูลหนึ่ง ๆ
- ข้อดี
  - ง่ายต่อการจัดการโครงสร้างของโปรแกรม
  - แก้ไขเปลี่ยนแปลงได้สะดวก
  - คอมไพล์แยกส่วนได้
  - สามารถเปลี่ยนการแทนข้อมูลได้ โดยไม่มีผลกระทบต่อผู้ใช้
  - Reliability โดยการซ่อนการแทนข้อมูล ผู้ใช้ไม่สามารถเข้าถึงข้อมูลในวัตถุได้โดยตรง

Abstract Data Types

- ในการทำ encapsulation ใน Java/C++ ใช้ class
- ในการทำ information hiding ใช้ access control modifiers

  ตัวอย่างเช่น ใน Java
  - private สมาชิกถูกอ้างถึงได้จากภายใน class เท่านั้น
  - public สมาชิกถูกอ้างถึงได้จาก class ใดๆ ที่มองเห็น
  - protected สมาชิกถูกอ้างถึงได้จาก subclass หรือ class ใน package เดียวกัน
  - ไม่มีระบุ (default) สมาชิกถูกอ้างถึงจาก class ใดๆ ใน package เดียวกัน



Categories of OOP Language

- สนับสนุน OOP โดยเป็นส่วนเพิ่มเติมในภาษาที่อยู่แล้ว
  - C++ (สนับสนุน procedural และ data-oriented programming ด้วย)
  - Ada95 (สนับสนุน procedural และ data-oriented programming ด้วย)
  - CLOS (สนับสนุน functional programming ด้วย)
  - Scheme (สนับสนุน functional programming ด้วย)
- สนับสนุน OOP แต่ยังคงใช้โครงสร้างพื้นฐานแบบภาษา imperative
  - Eiffel (เป็นภาษาใหม่ไม่มีรากฐานมาจากภาษาใด)
  - Java (รากฐานจากภาษา C++)
- ภาษา OOP อย่างเดียว (Pure OOP) เช่น ภาษา Smalltalk

Characteristics of OOP

- Encapsulation
- Information hiding

} ADT = Class

- **Inheritance**
- Dynamic binding
  - Polymorphism

Imperative VS Object-oriented

Imperative

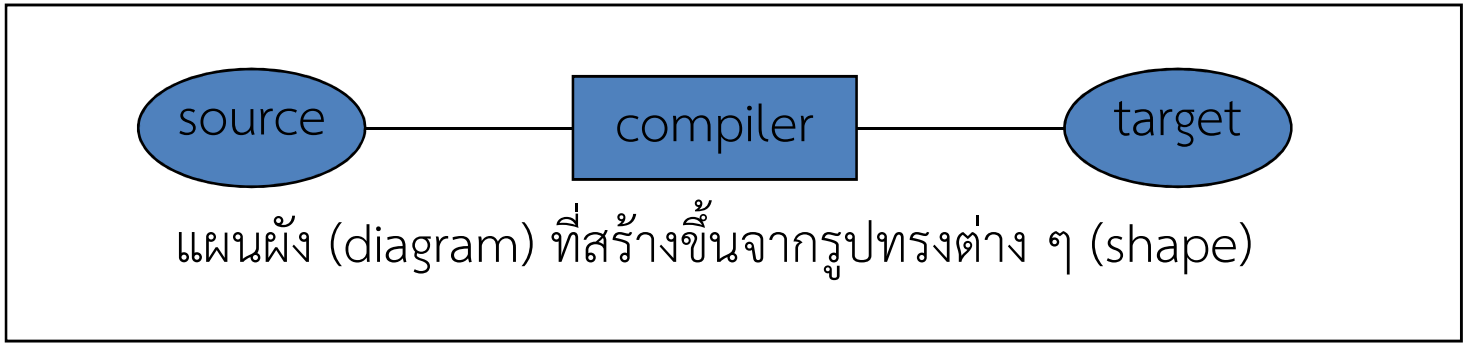
- Top-down design
- Procedure-oriented
- Algorithm
- Share global data
- Program-data separation
- Data transportation
- Single flow

Object-oriented

- Bottom-up
- Object-oriented
- Behavior
- Information hiding
- Encapsulation
- Communication with messages
- Multiple objects

Procedure-oriented VS Object-oriented

- ปัญหา : ต้องการวาดแผนผัง

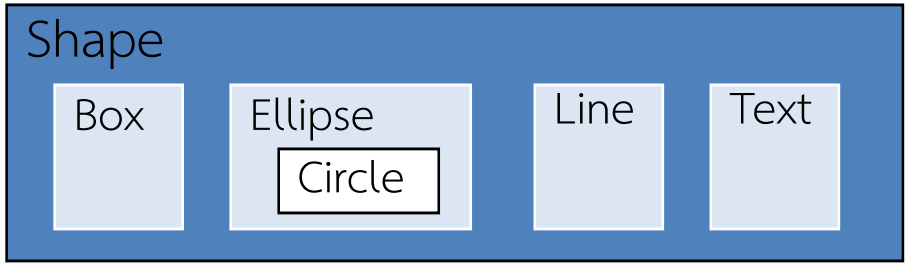


Procedural thinking

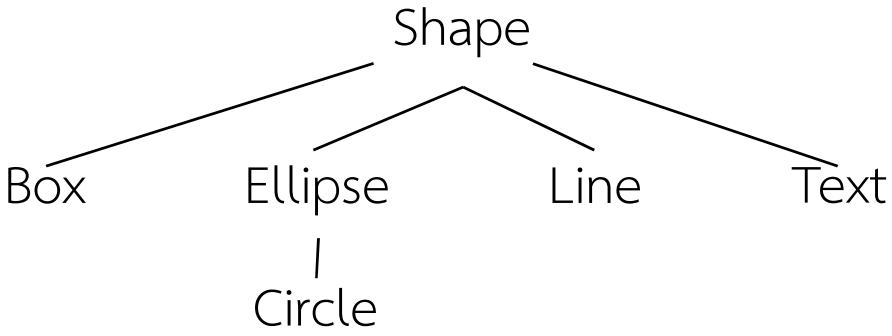
```
Procedure draw (diagram d);
begin
   for each shape s in diagram d do
   begin
        case s of
          box : code to draw a box;
          ellipse : code to draw an ellipse;
          .....

          arc :  code to draw an arc;
```

OO Thinking

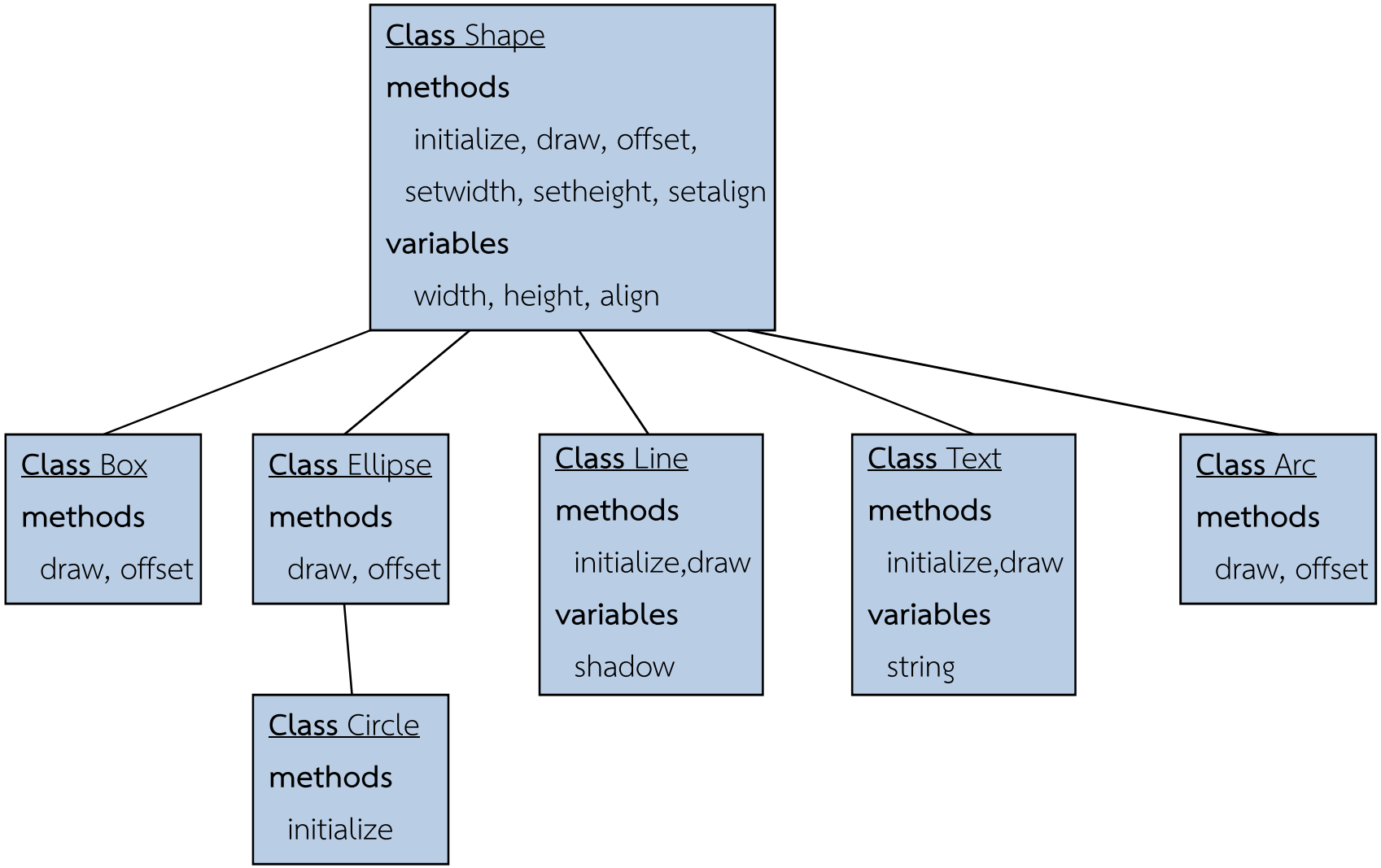


การจำแนกประเภทของรูปทรง



Class hierarchy ของ shape

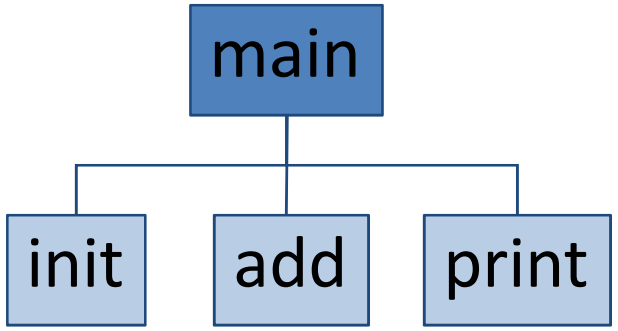
OO Thinking


Class Shape
methods
initialize, draw, offset,
setwidth, setheight, setalign
variables
width, height, align
Class Box
methods
draw, offset
Class Ellipse
methods
draw, offset
Class Circle
methods
initialize
Class Line
methods
initialize,draw
variables
shadow
Class Text
methods
initialize,draw
variables
string
Class Arc
methods
draw, offset

Complex number in C

```
#include <stdio.h>
Typedef struct { double  r,  i; }  Complex;
void init (Complex  *a, double r, double i) {
      a –> r = r;
      a –> i  = i;
}
void printComplex (Complex a) {
      printf(“%1.1f + i%1.1f”, a.r, a.i);
}
Complex add(Complex a, Complex b) {
      Complex c;
      c.r = a.r + a.r;
      c.i = a.i + b.i;
      return c;
}
```

```
void main() {
      Complex   x, y, z;
      init(&x, 1.0, 2.0);
      init(&x, 1.0, 2.0);
      z  =  add(x, y);
      printComplex(z);
}
```



Complex number in Java

```
#include <stdio.h>
class  Complex {
      private double r, i;
      Complex (double x, double y) {
              r =  x;   i = y;
      }
      public void add(Complex c) {
              r += c.r,  i += c.i;
      }
       public void print()  {
              system.out.println(r+”+i“+i);
      }
}
```

```
class ComplexTest {
   public static void main(String args[])  {
      Complex  a = new Complex(1.0, 2.0);
      Complex  b = new Complex(3.0, 4.0);
      Complex  c = new Complex(0.0, 0.0);
      c.add(a);
      c.add(b);
      c.print();
   }
}
```

Complex Class
- r : double - i : double
+ Complex() : void + add() : void + print() : void

OOP Concepts

- Class
- Object, instance
- Subclass, derived class
- Superclass, parent class
- Method, behavior
- Message
- Inheritance
- Polymorphism
- Overriding
- Encapsulation

Object-oriented concept

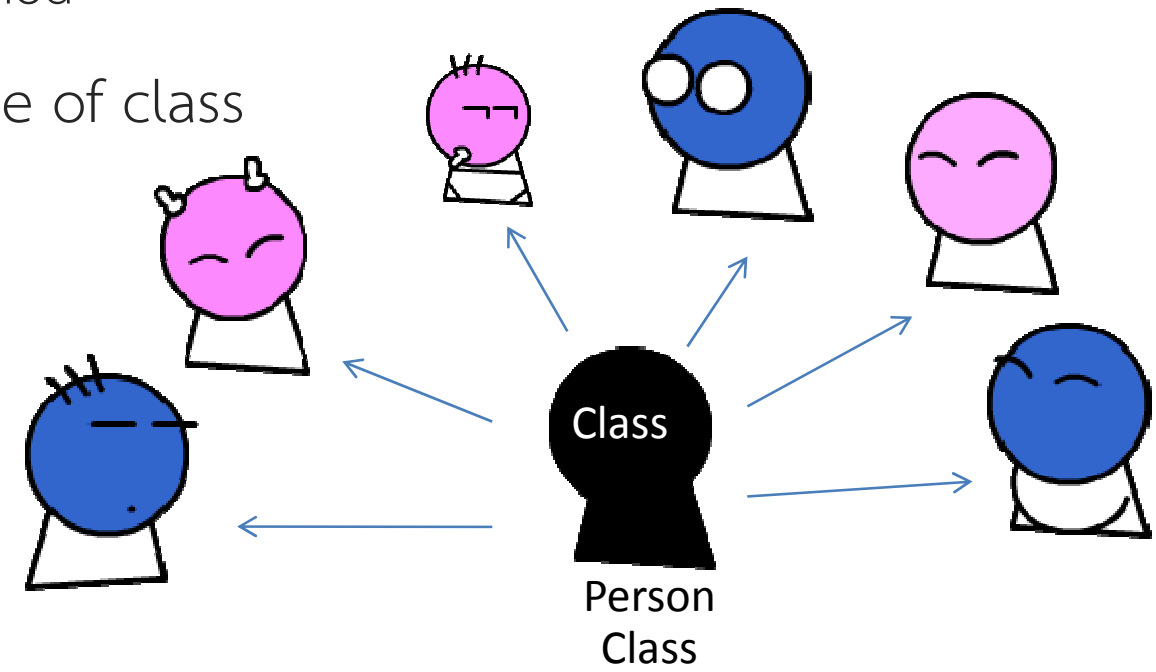
- วัตถุ (Object) ทั่วไปจะเก็บสถานะ (state) เป็นข้อมูลของตัวเองได้ และสามารถทำกิจกรรมบางอย่างได้ เช่น คำนวณ เปลี่ยนข้อมูล ติดต่อหรือโต้ตอบกับวัตถุอื่น
- นักเรียน --> วัตถุ
  - ข้อมูล : ชื่อ ที่อยู่ ผลการเรียน
  - กิจกรรม : ตอบคำถามต่าง ๆ เกี่ยวกับตัวเองได้, ติดต่อเพื่อดูผลสอบ
- ตัวแปรในภาษา Imperative ไม่สามารถจำลองพฤติกรรมและการทำงานของวัตถุเช่นนี้ได้

Class & Instance

- *Class* เป็นกลไกสำหรับกำหนดโครงสร้างและพฤติกรรมของวัตถุ
- *Instance* จะถูกสร้างขึ้นเพื่อจำลองการทำงานของวัตถุใน class นั้น
  - มีพื้นที่หน่วยความจำสำหรับเก็บข้อมูลของวัตถุ
  - มี method สำหรับกระทำต่อข้อมูล หรือโต้ตอบกับ instance อื่น
  - สามารถสร้าง instance ของ class ได้หลาย instance โดยแต่ละตัวจะมีโครงสร้างเหมือนกัน แต่มีข้อมูลที่ต่างกันไป
  - เมื่อสร้าง instance แล้ว มันจะรออยู่เฉย ๆ จนกว่าจะมีใครเรียกใช้
- Class -> Type
- Instance -> Variable

Class

- Class เป็นต้นแบบ (blueprint) ในการสร้าง object ประกอบด้วย
  - Attribute == state == data/variable
  - Behavior == method
- Object เป็น instance of class



Attribute & Behavior

- Class ประกอบด้วยสมาชิก 2 ประเภท คือ
  - data member คือสมาชิกที่เป็นข้อมูล อาจเป็นค่าคงที่ ตัวแปรของชนิดข้อมูลพื้นฐาน หรือตัวแปรที่เป็น object -> Attribute
  - method member คือสมาชิกที่เป็นฟังก์ชัน ซึ่งตอบสนองต่อ message เรียกสั้น ๆ ว่า *method* -> Behavior

Person Class

Person Class

+ name : string
+ height : float
+ weight : float
+ age : int
+ sex : string

+ walkAhead() : void
+ turnLeft() : void
+ turnRight() : void

Method

ประเภทของ method

- Constructor กำหนดค่าเริ่มต้นให้ object ชื่อ method คือชื่อ class ไม่มี return type
- Destructor ใช้ในการคืนพื้นที่ในหน่วยความจำ ไม่มี return type
- Accessor ใช้อ่านค่าของ data member นิยมตั้งชื่อขึ้นต้นด้วย get ตามด้วยชื่อ data member เช่น getName() มี return type เป็นชนิดข้อมูลของ data member นั้นๆ
- Mutator ใช้เปลี่ยนแปลงค่าของ data member นิยมตั้งชื่อขึ้นต้นด้วย set ตามด้วยชื่อ data member และรับพารามิเตอร์เป็นค่าที่จะกำหนดให้ data member ไม่มี return type

Method Examples in C++

```
class Human {
    private:
        int Age;
    public:
        Human()  { Age = 1; }
        int CalSum(int,int);
        void SetAge (int Num)  { Age = Num; }
        int GetAge() const  { Age++;   return Age;  }
         ~Human();
};
```

Declare Attribute of Person Class

Person Class

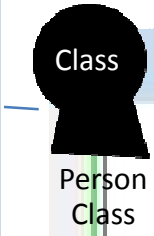
+ name : string
+ age : int
+ sex : string
+ height : float
+ weight : float
+ x : int

+ Person(name:string, age:int, sex:string) : void

+ WalkAhead(distance:int) : void

+ CanDrink() : bool

+ ToString() : string




```
class Person
{
    public string name;
    public int age;
    public string sex;
    public float height;
    public float weight;
    public int x;

    public Person(string name, int age, string sex)
    {
        this.name = name;
        this.age = age;
        this.sex = sex;
        this.height = 160;
        this.weight = 50;
        this.x = 0;
    }

    public void WalkAhead(int distance)
    {
```

Constructor of Person Class

Person Class

+ name : string
+ age : int
+ sex : string
+ height : float
+ weight : float
+ x : int

+ Person(name:string, age:int, sex:string) : void

+ WalkAhead(distance:int) : void

+ CanDrink() : bool

+ ToString() : string


```
class Person
{
    public string name;
    public int age;
    public string sex;
    public float height;
    public float weight;
    public int x;

    public Person(string name, int age, string sex)
    {
        this.name = name;
        this.age = age;
        this.sex = sex;
        this.height = 160;
        this.weight = 50;
        this.x = 0;
    }

    public void WalkAhead(int distance)
    {
```

WalkAhead() Method

Person Class

+ name : string
+ age : int
+ sex : string
+ height : float
+ weight : float
+ x : int

+ Person(name:string, age:int, sex:string) : void

+ WalkAhead(distance:int) : void

+ CanDrink() : bool

+ ToString() : string


```
public float weight;
public int x;

public Person(string name, int age, string sex)
{
    this.name = name;
    this.age = age;
    this.sex = sex;
    this.height = 160;
    this.weight = 50;
    this.x = 0;
}

public void WalkAhead(int distance)
{
    x = x + distance;
}

public bool CanDrink()
{
```

CanDrink() Method

Person Class

+ name : string
+ age : int
+ sex : string
+ height : float
+ weight : float
+ x : int

+ Person(name:string, age:int, sex:string) : void

+ WalkAhead(distance:int) : void

+ CanDrink() : bool

+ ToString() : string


```
        {
            this.name = name;
            this.age = age;
            this.sex = sex;
            this.height = 160;
            this.weight = 50;
            this.x = 0;
        }

        public void WalkAhead(int distance)
        {
            x = x + distance;
        }

        public bool CanDrink()
        {
            if (age > 20)
                return true;
            return false;
        }
```

ToString() Method

Person Class

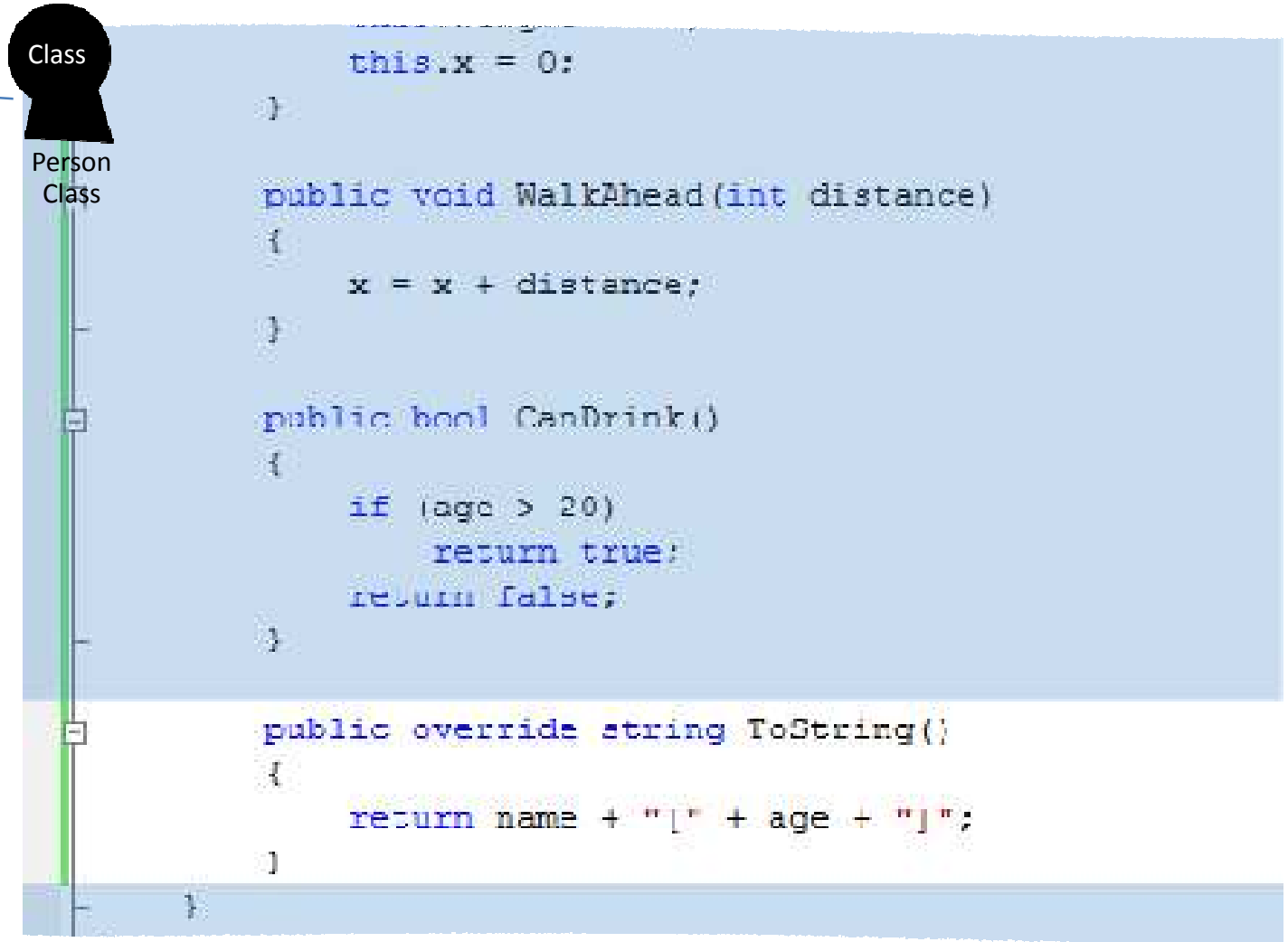
+ name : string
+ age : int
+ sex : string
+ height : float
+ weight : float
+ x : int

+ Person(name:string, age:int, sex:string) : void

+ WalkAhead(distance:int) : void

+ CanDrink() : bool

+ ToString() : string



Object/Instance of Class

Person Class

Person Class

+ name : string
+ height : float
+ weight : float
+ age : int
+ sex : string

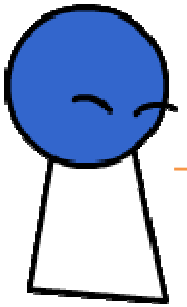
+ walkAhead() : void
+ turnLeft() : void
+ turnRight() : void

p1

new Person()

+ name = “Eve”
+ height = 160
+ weight = 199
+ age =19
+ sex = “female”

+ walkAhead() : void
+ turnLeft() : void
+ turnRight() : void



new Person()

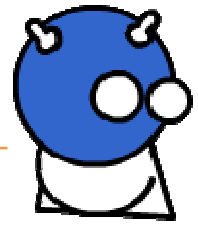
+ name = “Oak”
+ height = 210.3
+ weight = 55
+ age =20
+ sex = “male”

+ walkAhead() : void
+ turnLeft() : void
+ turnRight() : void

new Person()

+ name = “mint”
+ height = 173.4
+ weight = 89
+ age = 99
+ sex = “male”

+ walkAhead() : void
+ turnLeft() : void
+ turnRight() : void



Reference Variable

Primitive

```
int a=5;
int b=3;

a=b;
```

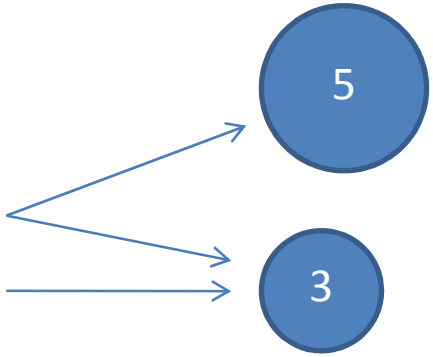
In Memory
a = 3
b = 3

Reference

```
Circle a= new Circle( 5 );
Circle b= new Circle( 3 );

a=b;
```

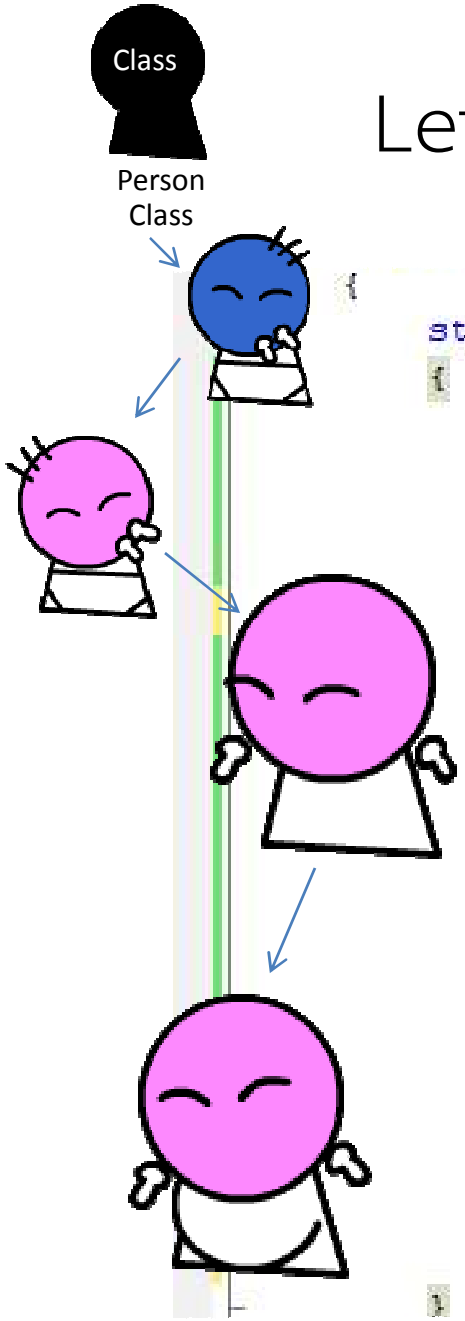
In Memory
a
b



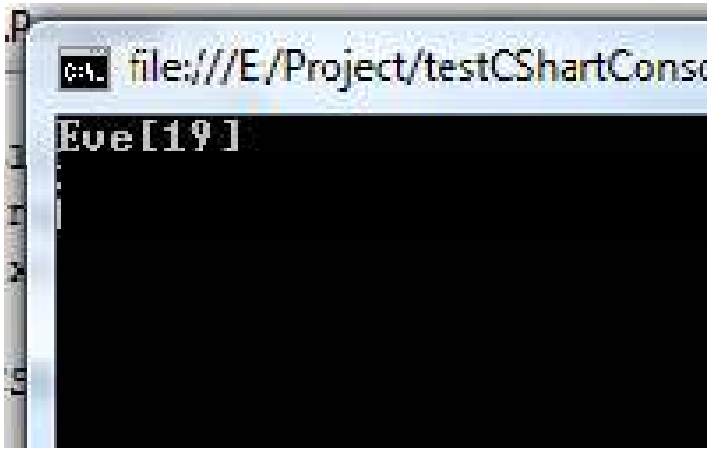
Message

- Class เป็นตัวระบุคุณสมบัติต่าง ๆ ของ object
- เมื่อมีการกำหนดพื้นที่สำหรับ object หนึ่งจะมีการสร้าง instance ขึ้น
- Object แต่ละตัวจะติดต่อกันโดยการรับส่ง message
- เมื่อ object ได้รับ message จะ execute method ที่เหมาะสมเพื่อตอบสนอง
- Message ประกอบด้วย 2 ส่วน คือ ชื่อ method และ destination object

Let's Use Person Class



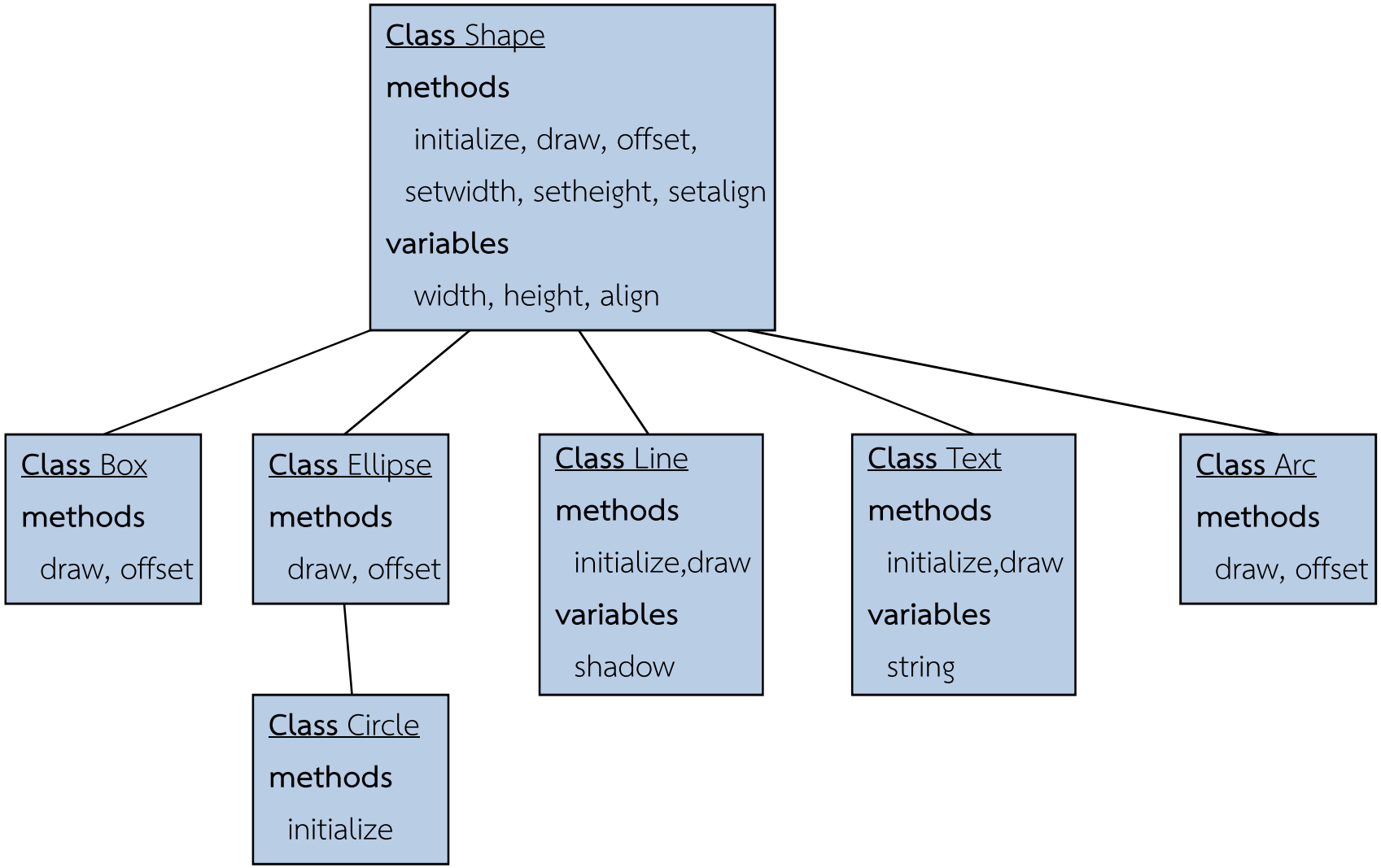

```
{
    static void Main(string[] args)
    {
    }
```



Inheritance

- เราสามารถสร้าง class ใหม่ได้ โดยนำ class ที่ถูกสร้างและใช้งานอยู่แล้วมาปรับปรุงเปลี่ยนแปลงให้เหมาะสมกับหน้าที่ใหม่ที่ต้องการ
- class เดิมจะยังคงใช้งานได้เช่นเดิม เรียกว่าเป็น superclass หรือ parent class ของ class ใหม่
- class ใหม่จะได้สมาชิกของ class เดิมมาพร้อมกับมีการเพิ่มสมาชิกใหม่หรือเปลี่ยนแปลงสมาชิกที่มีอยู่เดิมให้ต่างไป จะเรียกว่าเป็น subclass หรือ derived class ของ class เดิม
- กลไกการสร้าง class แบบนี้เรียกว่า การสืบทอดคุณสมบัติ หรือ inheritance

Superclass & Subclass



Inheritance in C++

- class ใดๆ ไม่จำเป็นต้องเป็น subclass ของ class ใด class หนึ่ง
- การเข้าถึงสมาชิก (Access modifier) ใน class มี 3 รูปแบบ
  - Private สมาชิกถูกอ้างถึงได้เฉพาะภายใน class และ friends เท่านั้น (ทำให้ subclass ไม่เป็น subtype)
  - Public สมาชิกถูกอ้างถึงได้จาก class ใดๆ ที่มองเห็น class นั้นได้หรือ client
  - Protected สมาชิกถูกอ้างถึงได้ภายใน class และ subclass ยกเว้น client

Inheritance Example in C++

```
class A  {
  public:
     int x;
     char f( );
      A( );
};
class D : public A {
     int x;
     int g( );
};
```

class D เป็น subclass ของ class A

Object ของ class D มีสมาชิก 4 ตัวคือ

- x เป็น data member ที่สืบทอด (inherit) มาจาก class A (A::x)
- f เป็น method ที่สืบทอดมาจาก class A
- x เป็น data member ที่ประกาศเพิ่มใน class D
- g เป็น method ที่ประกาศเพิ่มใน class D

Encapsulation & Inheritance

Inheritance Example in C++

```
class List {
  private :
     cell * rear;
  public :
     List( );
     int     empty( );
  protected:
     void  add(int);
     void  push(int);
     int     get( );
};
class Queue : public List {
public :
     Queue ();
     int     get( ) { return List::get(); }
     void  put  ( int x) { List::add(x); }
};
```

สมาชิกของ object ใน class Queue

Public function

Queue	added(constructor)
get	added
put	added
List::empty	inherited

Protected function

add	inherited
push	inherited
List ::get	inherited

Information Hiding

Inheritance Example in C++

```
class Student: public Human
{
    private:
        int Score;
    public:
       void SetScore(int Num){
            Score = Num;
       }
       int GetScore()  {
            return Score;
       }
};
```

```
int main()   {
    Student Miki;

    Miki.SetAge(40);
    Miki.SetScore(88);

    cout << "Age :" << Miki.GetAge();
    cout << endl;
    cout << "Score :" << Miki.GetScore();

    return 0;
}
```

Inheritance in C++

- การสืบทอดคุณสมบัติสามารถกำหนด access controls (private หรือ public ทำให้สามารถเปลี่ยนการเข้าถึงสมาชิกใน subclasses ได้
  - Private derivation สมาชิก (ทั้ง Member Function และ Data Member) ในส่วนของ public และ protected ใน class แม่จะกลายเป็นสมาชิกของ class ลูกในลักษณะ private
  - Public/Protected derivation สมาชิกของ class แม่จะกลายเป็นสมาชิกของ class ลูกที่มีลักษณะเป็น public/protected
- derived class จะไม่สามารถเข้าถึงส่วน private ของ parent class ได้ทั้งสามรูปแบบ

Inheritance Example in C++

```
class base_class {
  private:
    int a;
    float x;
  protected:
    int b;
    float y;
  public:
    int c;
    float z;
};

class subclass_1 : public base_class { … };
//     In this one, b and y are protected and
//     c and z are public

class subclass_2 : private base_class { … };
//    In this one, b, y, c, and z are private,
//    and no derived class has access to any
//    member of base_class
```

Reexportation in C++

- สมาชิกที่ไม่สามารถเข้าถึงได้ใน subclass เนื่องจากการทำ private derivation สามารถทำให้มองเห็น/เข้าถึงได้โดยการใช้ scope resolution operator (::)
- ตัวอย่างเช่น

```
class subclass_3 : private base_class {
        base_class :: c;
        …
}
```

Example 1

```
class single_linked_list {
  private:
    class node {
       public:
        node *link;
        int contents;
    };
    node *head;
  public:
    single_linked_list() { head = 0 }
    void inseart_at_head(int);
    void insert_at_tail(int);
    void remove_at_head();
    int empty();
};
```

Example 2

```
class stack : public single_linked_list {
  public:
    stack();
    void push(int value) {
        single_linked_list ::inseart_at_head(int value);
    }
    int pop() {
       return single_linked_list::remove_at_head();
    }
};
class queue : public single_linked_list {
  public:
    queue();
    void enqueue(int value) {
        single_linked_list ::inseart_at_tail(int value);
    }
    int dequeue() {
       return single_linked_list::remove_at_head();
    }
};
```

Example 3

```
class stack : private single_linked_list {
  public:
    stack();
    void push(int value) {
        single_linked_list ::insert_at_head(int value);
    }
    int pop() {
       return single_linked_list::remove_at_head();
    }
    single_linked_list::empty();
};
class queue : private single_linked_list {
  public:
    queue();
    void enqueue(int value) {
        single_linked_list ::inseart_at_tail(int value);
    }
    int dequeue() {
       return single_linked_list::remove_at_head();
    }
    single_linked_list::empty();
};
```

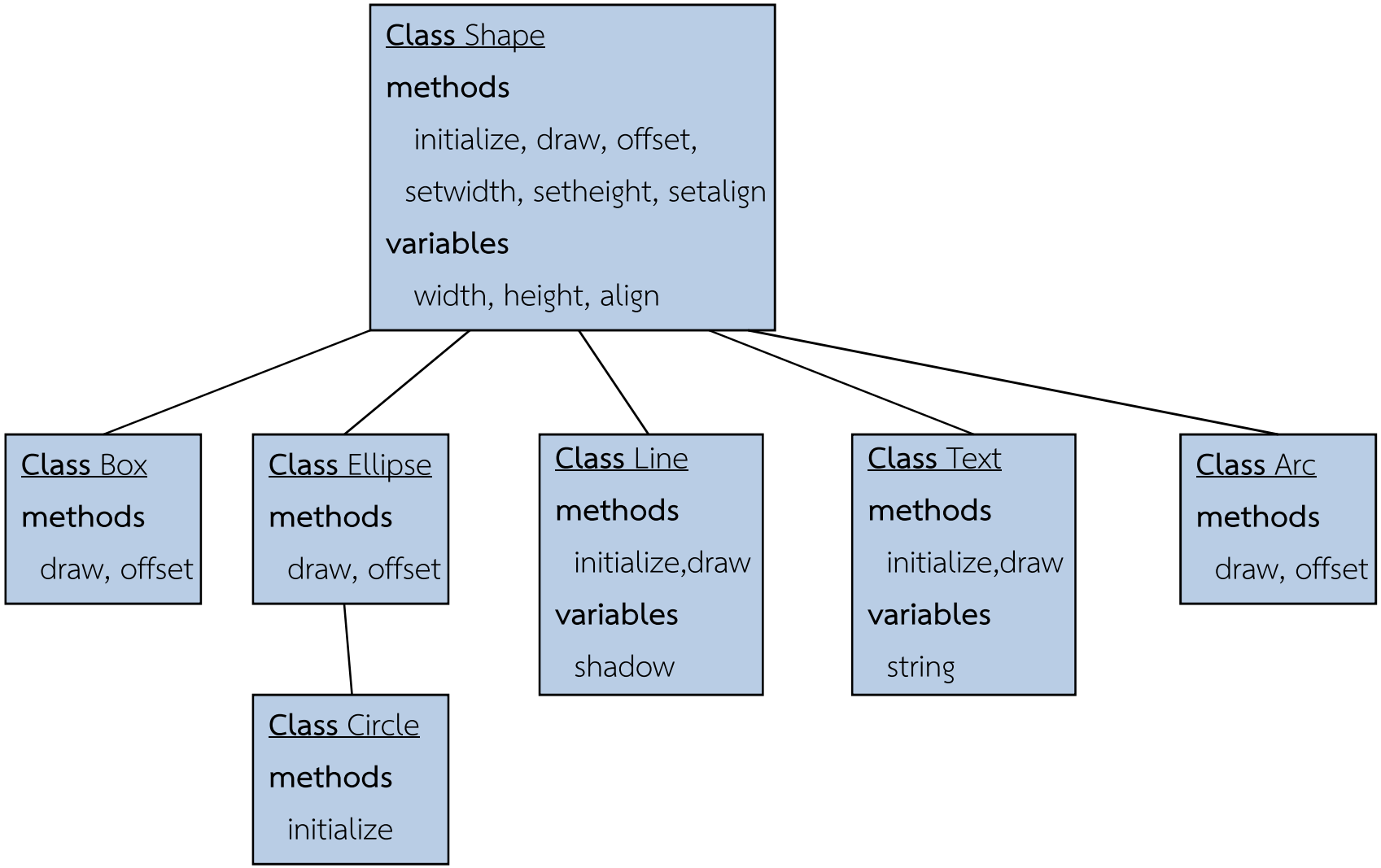
Dynamic Binding & Polymorphism

- Method **overriding** เป็นการกำหนดชื่อของ method ใน subclass ให้มีชื่อเหมือนกับ method ใน parent class เพื่อเปลี่ยนแปลงการทำงาน method ใน parent class ให้ทำหน้าที่อื่นใน subclass
- การ binding ว่าชื่อ method ที่ instance ส่งมาเป็น method ตัวใด จะทำแบบ **dynamic binding**
- แนวคิดในการจัดการกับ instance ของ class ที่ต่างกันด้วย method เดียวกันเรียกว่า **polymorphism**
- polymorphism เป็นการสนับสนุนการออกแบบโปรแกรมเพื่อให้ใช้งานร่วมกัน หรือการนำกลับมาใช้ได้อีก

Abstract Class

- method ที่ทำการ overriding ต้องมี protocol ที่เหมือนกัน
- method ที่กำหนดเป็น virtual จะสามารถเรียกใช้ผ่าน polymorphic variables และการ binding ของ message จะเป็นแบบ dynamic binding
- pure virtual function จะไม่มีการนิยามใดๆ
- class ใดๆ ที่มี pure virtual function อย่างน้อยหนึ่ง function จะเรียกว่าเป็น *abstract class*

Framework


Class Shape
methods
initialize, draw, offset,
setwidth, setheight, setalign
variables
width, height, align
Class Box
methods
draw, offset
Class Ellipse
methods
draw, offset
Class Circle
methods
initialize
Class Line
methods
initialize,draw
variables
shadow
Class Text
methods
initialize,draw
variables
string
Class Arc
methods
draw, offset

Polymorphism Example in C++

```
class shape {
  public:
    virtual void draw() = 0;
   …
}
class circle : public shape{
  public:
    virtual void draw() {…}
  …
}
class rectangle : public shape{
  public:
    virtual void draw() {…}
  …
}
class square : public shape{
  public:
    virtual void draw() {…}
  …
}
```

```
square s;
rectangle r;
shape &ref_shape = s;
//a reference to the square s

ref_shape.draw();
//dynamic binding to draw in square

r.draw();
//static binding to draw in rectangle
```

Polymorphism & Dynamic binding

Method Overloading

- Method **overloading** เป็นการเรียก method หนึ่งด้วย argument ที่แตกต่างกันไปในการเรียกแต่ละครั้ง และจะต้องมีหลาย ๆ method สำหรับการเรียกแต่ละแบบ
- เช่น method ในการบวกอาจประกอบด้วย 3 method ซึ่งมี argument ต่างกัน คือ
  - function add(x:integer; y:integer)
  - function add(x:real; y:real);
  - function add(x:integer; y:real);

Multiple Inheritance

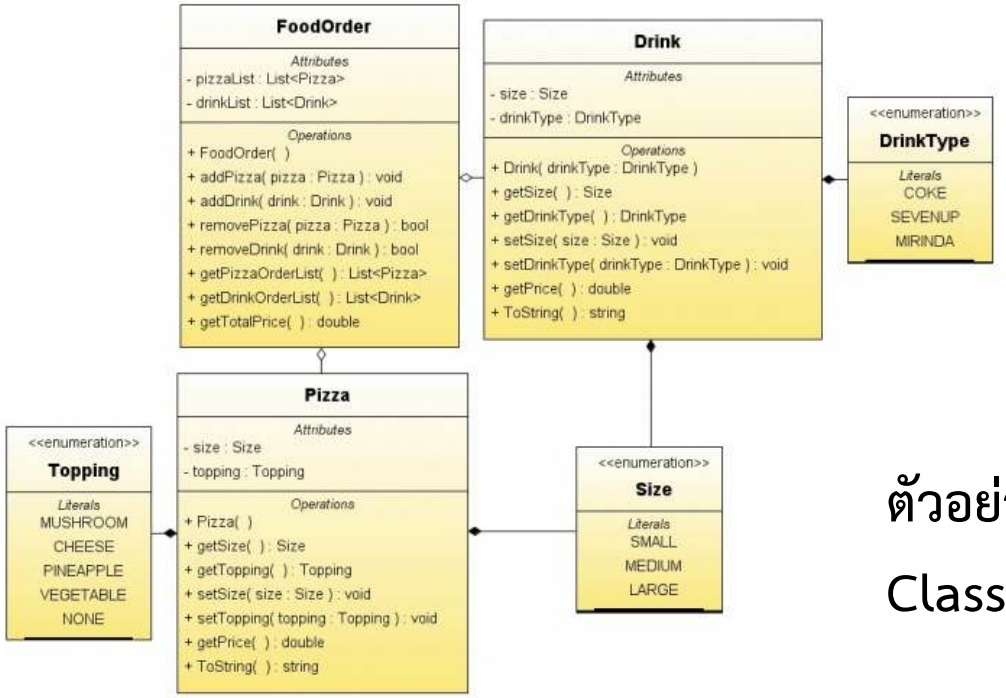
- Multiple Inheritance เป็นการสร้าง class ใหม่จาก superclass มากกว่าหนึ่ง class
- ถ้าสมาชิกชื่อซ้ำกันใน superclass ที่สืบทอดมา ใช้ scope resolution operator (::) ในการระบุสมาชิกที่อ้างถึง เช่น A::getN(), B::x:getN()

```
class A {...};
class B {...};
class C : public A, public B {...};
```

UML

- Unified Modeling Language

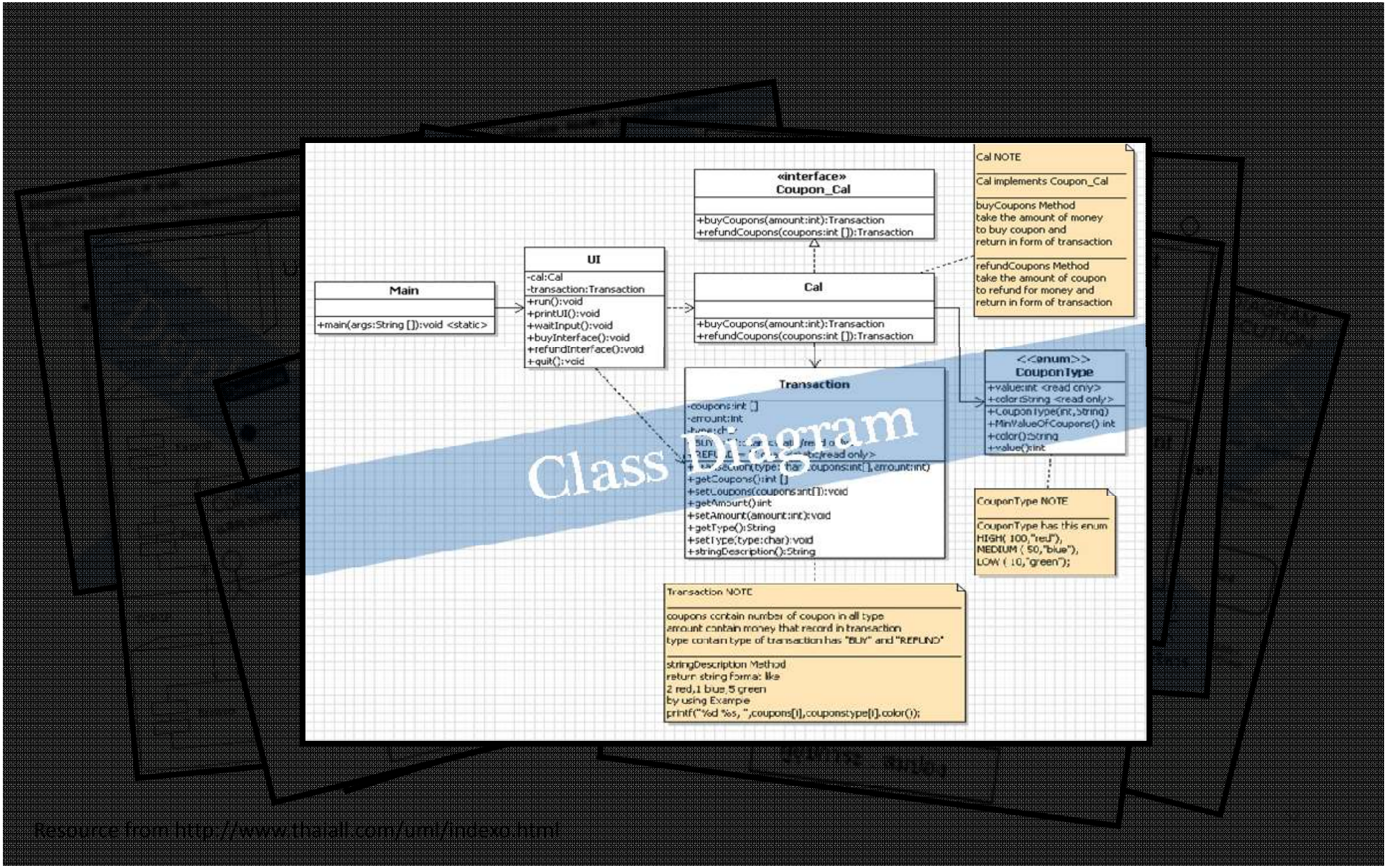
  UML ถูกใช้ในการ*สื่อสาร*กันภายในทีม เพื่อให้เข้าใจตรงกัน รวมถึงช่วยในการ *แบ่งงาน แบ่งส่วน* ของโปรแกรม



ตัวอย่างของ UML Class diagram

Resource from http://www.thaiall.com/uml/indexo.html

Many Types of UML



Resource from http://www.thaiall.com/uml/index.html

ข้อดีของ OOP

- เข้าใจง่าย เพราะการทำงานเปรียบเสมือนการจำลองเหมือนในโลกจริง โดยอาศัยการมองทุกอย่างเป็น object ซึ่งแต่ละตัวมีหน้าที่และความหมายในตัว
- บำรุงรักษา และแก้ไขโปรแกรมได้ง่าย
- มีความปลอดภัยสูง เพราะจัดการกับ error ได้ดี
- นำกลับมาใช้ใหม่ได้ (reusability) ลดขั้นตอนในการเขียนโปรแกรม
- โปรแกรมมีคุณภาพสูง ใช้ได้หลาย platform

ข้อเสียของ OOP

- เข้าใจยาก สำหรับผู้เริ่มต้นเขียนโปรแกรม หรือถนัดเขียนโปรแกรมแบบ procedural
- ทำงานได้ช้ากว่าภาษาโปรแกรมอื่น
- ภาษามีความกำกวม ถ้ามีลักษณะ multiple inheritance

อ้างอิงจาก

- Slide เรื่อง Object-oriented Programming โดย Oakvale ในงาน exceed camp project ม.เกษตรศาสตร์
- หนังสือ Fundamental of Java Programming Volume I โดย ดร.วีรศักดิ์ ซึงถาวร
- หนังสือแนวคิดเชิงวัตถุ Object-oriented Concept โดย จตุรพิธ สมหอม และ พิชชานันท์ พิงคประเสริฐ
- หนังสือ Concepts of Programming Languages โดย Robert Sebesta
- แบบเรียน ภาษา C++ Online โดย devzilla เข้าถึงได้จาก http://devzilla.exteen.com