



การเขียนโปรแกรมเชิงวัตถุ

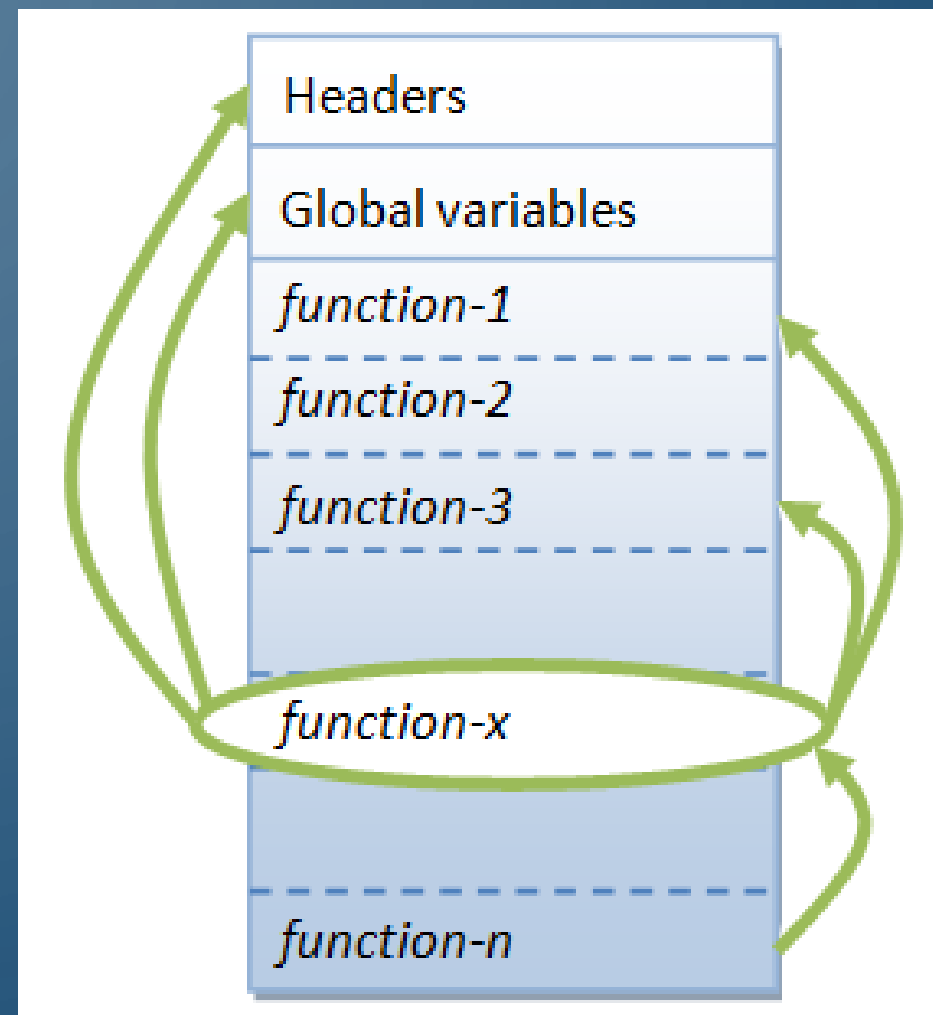
OBJECT ORIENTED PROGRAMMING (OOP)

SETHALAT RODHETBHAJ

DEPARTMENT OF COMPUTING • SILPAKORN UNIVERSITY

Procedural (*Function-based*) Programming

- โปรแกรมจะประกอบไปด้วยฟังก์ชันหลายฟังก์ชัน แต่ฟังก์ชันเหล่านี้มักจะไม่สามารถนำกลับมาใช้ได้อีกงานใหม่ (Reusable) เนื่องจากแต่ละฟังก์ชันจะไม่สามารถในการห่อหุ้ม (Encapsulate) หน่วยต่างๆ ภายในตัวเอง
- แยกระหว่างโครงสร้างข้อมูลกับการดำเนินการ (อัลกอริทึม) ออกจากกัน



Object Oriented Programming

- วิธีการเขียนโปรแกรมแบบ Object Oriented Programming (OOP) (เช่น ภาษา C++/Java) มีแนวคิดที่แตกต่างจาก Procedural programming (เช่น ภาษา C)
- Object มักจะถูกออกแบบโดยนำเสนอสิ่งที่ได้พบเห็นอยู่ทั่วไปในโลกแห่งความจริง โดยรวมโครงสร้างข้อมูลกับการดำเนินการ (อัลกอริทึม) เข้าไว้ด้วยกัน
- เนื่องจาก OOP มีความสามารถในการห่อหุ้ม (Encapsulate) หน่วยต่างๆ ภายในตัวเอง จึงทำให้สามารถนำกลับมาใช้ทำงานใหม่ (Reusable) ได้ง่าย

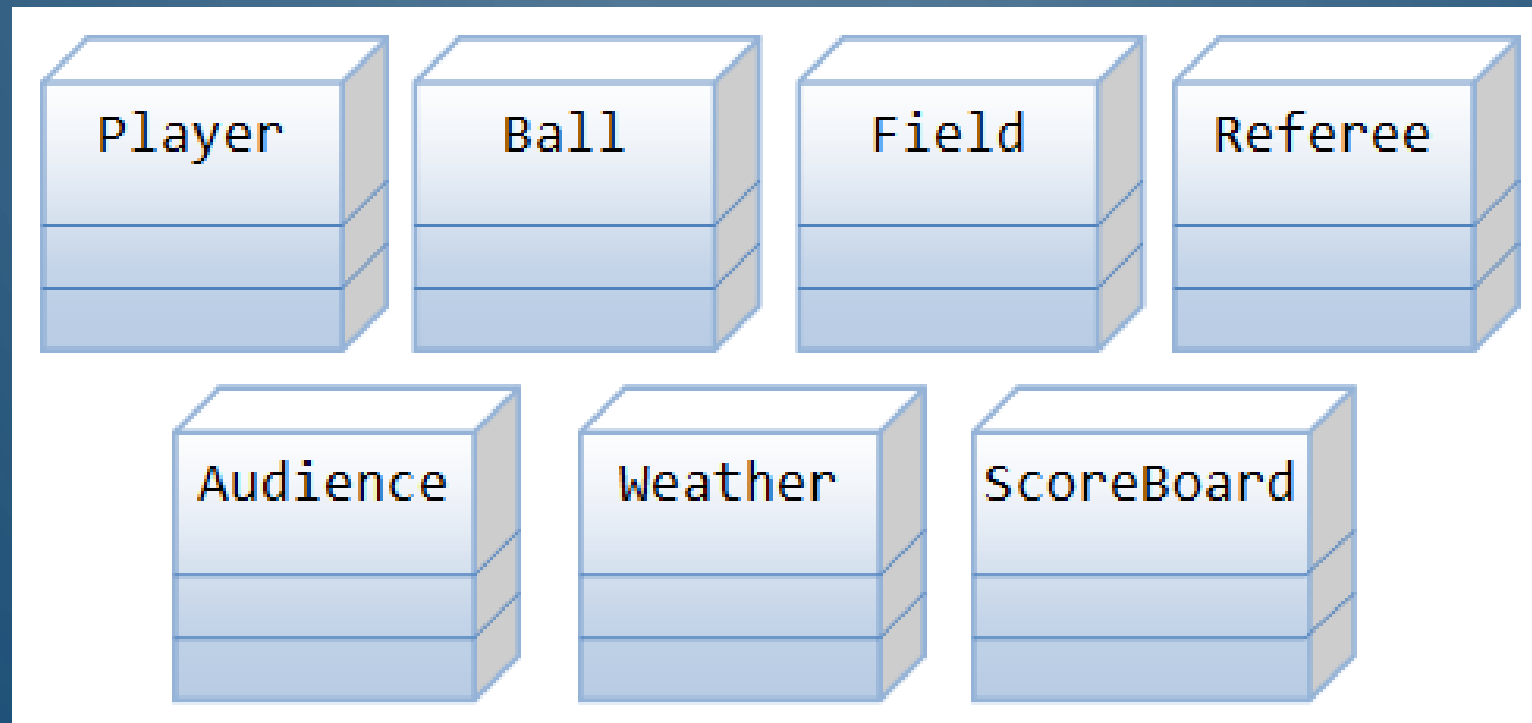
Object Oriented Programming

- การเขียนโปรแกรมแบบ OOP จะโดยมีแนวคิดเกี่ยวกับมองและการอธิบาย **คุณลักษณะ** (attribute / property / field / data / state / characteristic / variable) และ **พฤติกรรม** (method / function / action / behavior) ของ **วัตถุ (object)** ซึ่งพฤติกรรมของวัตถุอาจจะทำให้เกิดการเปลี่ยนแปลงคุณลักษณะของมันก็ได้ ทั้งนี้ จะมีการกำหนดนิยามคุณลักษณะและพฤติกรรมของวัตถุที่จัดเป็นประเภทเดียวกัน เรียกว่า **คลาส (class)**



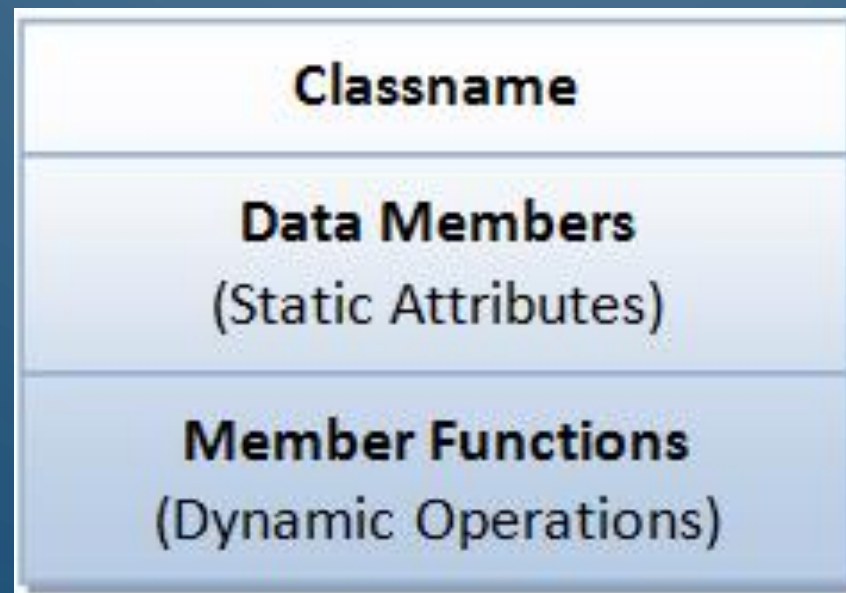
ตัวอย่าง

- คลาสต่างๆ ในสนามฟุตบอล



คลาส (Class)

- คลาส (Class) เป็นการกำหนดนิยามของวัตถุ (Object) ประเภทเดียวกัน
- หรือกล่าวอีกนัยหนึ่งว่า คลาสเป็นแบบพิมพ์เขียว (Blueprint)/แม่พิมพ์ (Template)/ต้นแบบ (Prototype) ของการกำหนดและบรรยายลักษณะของ Static Attribute และ Dynamic Behavior ของวัตถุทุกอันที่จัดเป็นประเภทเดียวกัน



ตัวอย่าง

Student
name grade
getName() printGrade()

Circle
radius color
getRadius() getArea()

SoccerPlayer
name number xLocation yLocation
run() jump() kickBall()

Car
plateNumber xLocation yLocation speed
move() park() accelerate()

วัตถุ (Object)

- วัตถุ (Object) หรือ อินสแตนซ์ (Instant) คือ สิ่งที่ถูกสร้างจากคลาส โดยที่ทุกวัตถุของคลาสจะมีคุณสมบัติคล้ายคลึงกันตามที่ได้ถูกนิยามเอาไว้ในคลาส

<u>paul:Student</u>
name="Paul Lee" grade=3.5
getName() printGrade()

<u>peter:Student</u>
name="Peter Tan" grade=3.9
getName() printGrade()

คลาส & วัตถุ

คลาส
(Class Definition)

Circle
-radius:double=1.0 -color:String="red"
+getRadius():double +getColor():String +getArea():double

วัตถุ
(Objects / Instants)

<u>c1:Circle</u>
-radius=2.0 -color="blue"
+getRadius() +getColor() +getArea()

<u>c2:Circle</u>
-radius=2.0 -color="red"
+getRadius() +getColor() +getArea()

<u>c3:Circle</u>
-radius=1.0 -color="red"
+getRadius() +getColor() +getArea()

คลาสใน C++

- การนิยาม class จะเริ่มต้นด้วยคำว่า *class*
- ส่วน body ของคลาสจะอยู่ในวงเล็บปีกกา { } ; (มี semi-colon ปิดท้าย)

```
class class_name {  
    ....  
    ....  
    ....  
};
```

ชื่อคลาส

ส่วน **body** ของคลาส (ประกอบด้วย **members** และ **methods**)

คลาสใน C++

- ภายใน body ของคลาส จะมีการกำหนดระดับของการเข้าถึงสมาชิกใน class เช่น **private:** และ **public:** (ค่าปกติจะกำหนดอัตโนมัติให้เป็น **private**)
 - **private:**
 - มีเฉพาะเพียง member function ของ class เท่านั้นที่จะเข้าถึงได้
 - Private members และ method จะสามารถใช้ภายใน class ได้เท่านั้น
 - **public:**
 - สามารถที่จะเข้าถึงจากนอก class ได้โดยตรง
 - สิ่งใน public เสมือนเป็น *interface*
- โดยทั่วไป ตัวแปรจะประกาศใน **private:** section และพวก function จะอยู่ในส่วน **public:** section

คลาสใน C++

```
class class_name {
```

```
    private:
```

```
        ...
```

```
        ...
```

```
        ...
```

```
    public:
```

```
        ...
```

```
        ...
```

```
        ...
```

```
};
```

Private members หรือ methods

Public members หรือ methods

ตัวอย่าง Class

- ตัวอย่างของ class ที่แสดงให้เห็นข้อมูลของวงกลมที่ถูกสร้างเก็บไว้ใน package เดียว (unit หรือ class)

```
class Circle {  
    private:  
        double radius;  
    public:  
        void setRadius(double r);  
        double getDiameter();  
        double getArea();  
        double getCircumference();  
};
```

ไม่ต้องการให้ class อื่นมาเข้าถึงหรือดัดแปลงค่าได้โดยตรง

สามารถเข้าถึงได้จากภายนอก และยังสามารถที่จะเข้าถึงสมาชิกอย่าง radius

Constructor

- Constructor:
 - เป็น Public method ที่มีชื่อเดียวกับ class จะถูกเรียกเมื่อมี object ใหม่ถูกสร้างขึ้นมา (instantiated) เป็นการกำหนดค่าเริ่มต้นให้กับข้อมูลสมาชิก ไม่มีการ return type
 - Constructor สามารถกำหนดได้หลายรูปแบบ
 - Constructor/Method overloading

Constructor

```
class Circle {  
    private:  
        double radius;  
    public:  
        Circle();  
        Circle(double r);  
        void setRadius(double r);  
        double getDiameter();  
        double getArea();  
        double getCircumference();  
};
```

Constructor แบบไม่มี argument

Constructor แบบมี 1 argument

การประกาศนิยาม Member Method ของคลาส

1. Member methods ที่ประกาศภายใน class

```
class Circle {  
    private:  
        double radius;  
    public:  
        Circle() { radius = 0.0; }  
        Circle(double r);  
        void setRadius(double r) { radius = r; }  
        double getDiameter() { return radius * 2; }  
        double getArea();  
        double getCircumference();  
};
```

ประกาศภายใน class

การประกาศนิยาม Member Method ของคลาส

2. Member methods ที่ประกาศภายนอก class

- ใช้ Binary scope resolution operator (::)
- เชื่อมความสัมพันธ์ระหว่างชื่อ method กับชื่อ class
- สร้าง method ที่มีชื่อไม่ซ้ำกัน
- class ที่ต่างกันสามารถมี method ชื่อซ้ำกันได้
- รูปแบบ

```
ReturnType ClassName::MemberMethodName () {  
    ...  
}
```
- ตัวอย่าง

```
double Circle::getDiameter() { return radius *2; }
```

```
class Circle {  
    private:  
        double radius;  
    public:  
        Circle() { radius = 0.0; }  
        Circle(double r);  
        void setRadius(double r) { radius = r; }  
        double getDiameter() { return radius * 2; }  
        double getArea();  
        double getCircumference();  
};  
Circle::Circle(double r) {  
    radius = r;  
}  
double Circle::getArea() {  
    return radius * radius * M_PI;  
}  
double Circle::getCircumference() {  
    return 2 * radius * M_PI;  
}
```

ประกาศภายใน class

ประกาศภายนอก class

วิธีการสร้าง object ของคลาส

- การประกาศตัวแปรโดยอ้างถึงชนิดของ class ที่นิยามเอาไว้ทำให้เกิดตัวแปร object ขึ้นมาใหม่ (instance) เมื่อ object ถูกสร้างขึ้นมาก็จะมีการจองเนื้อที่ใน memory ใหม่เพื่อใช้เก็บข้อมูล
- สามารถสร้าง object ได้มากมายจาก class
 - ตัวอย่าง `Circle c;` หรือ `Circle *c;`

การเข้าถึง Class Member

- Operator ที่จะใช้ในการเข้าถึง class member
 - Dot operator (.)
 - Object
 - Reference to object
 - Arrow operator (->)
 - Pointers

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

```
class Circle {
    private:
        double radius;
    public:
        Circle() { radius = 0.0; }
        Circle(double r);
        void setRadius(double r) { radius = r; }
        double getDiameter() { return radius * 2; }
        double getArea();
        double getCircumference();
};

Circle::Circle(double r) {
    radius = r;
}

double Circle::getArea() {
    return radius * radius * M_PI;
}

double Circle::getCircumference() {
    return 2 * radius * M_PI;
}
```

constructor
ตัวที่ 1 ถูกเรียก

constructor
ตัวที่ 2 ถูกเรียก

```
int main() {
    Circle c1, c2(7);

    cout<<"The area of c1:"
        <<c1.getArea()<<"\n";

    //c1.radius = 5; //syntax error
    c1.setRadius(5);

    cout<<"The circumference of c1: "
        <<c1.getCircumference()<<"\n";

    cout<<"The Diameter of c2: "
        <<c2.getDiameter()<<"\n";
}
```

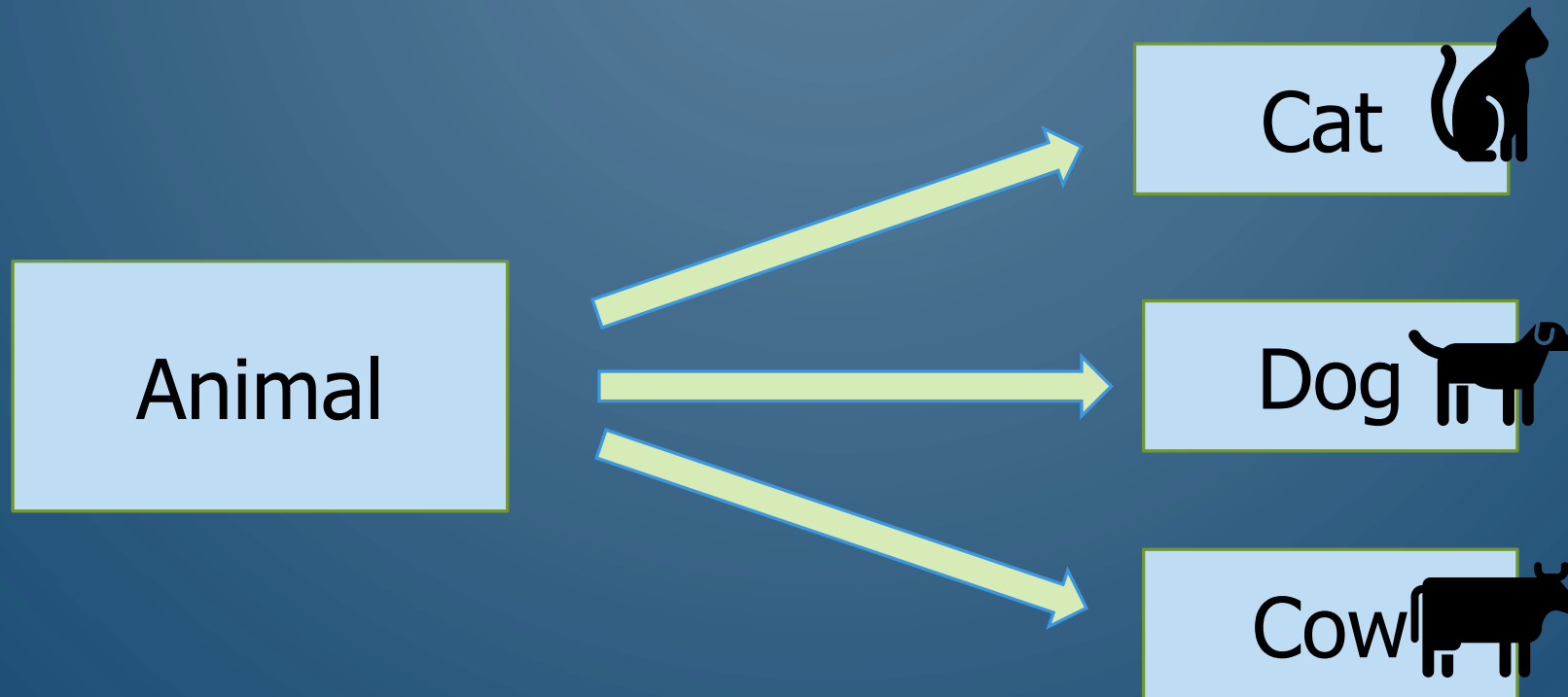
เนื่องจาก radius เป็นข้อมูล
ประเภท private

```
class Circle {  
    private:  
        double radius;  
    public:  
        Circle() { radius = 0.0; }  
        Circle(double r);  
        void setRadius(double r) { radius = r; }  
        double getDiameter() { return radius * 2; }  
        double getArea();  
        double getCircumference();  
};  
Circle::Circle(double r) {  
    radius = r;  
}  
double Circle::getArea() {  
    return radius * radius * M_PI;  
}  
double Circle::getCircumference() {  
    return 2 * radius * M_PI;  
}
```

```
int main() {  
    Circle c(7);  
    Circle *cp1 = &c;  
    Circle *cp2 = new Circle(7);  
  
    cout<<"The area of cp2: "<<cp2->getArea();  
}
```

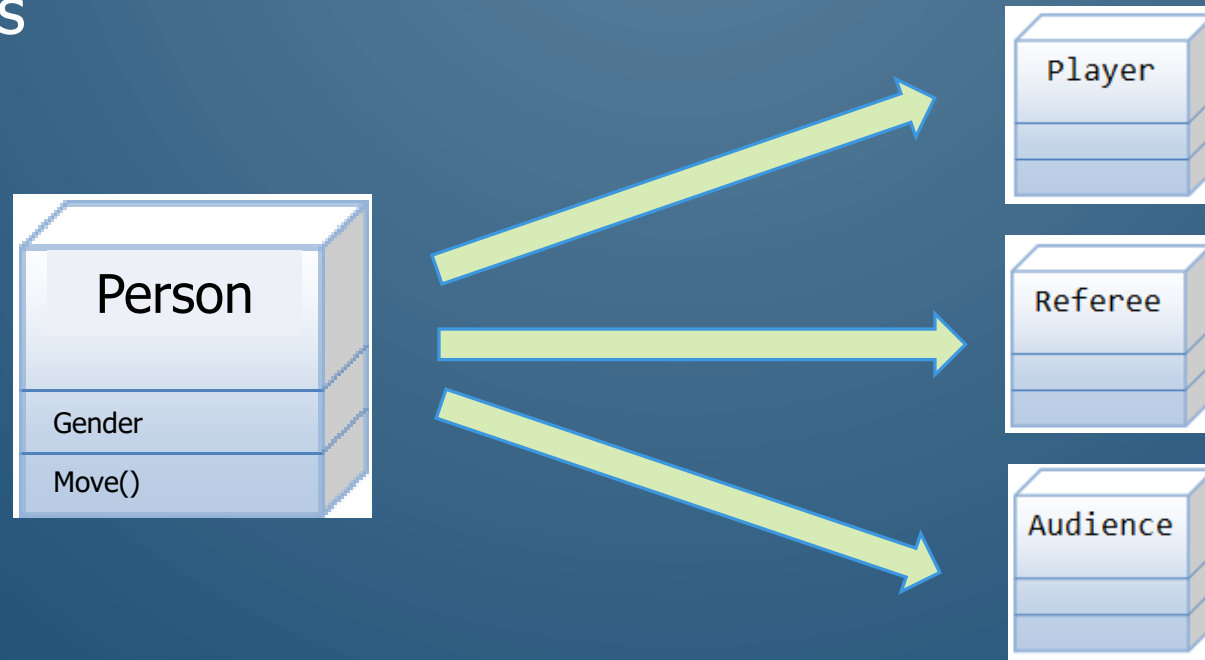

Inheritance

- การสืบทอดคุณสมบัติและความสามารถต่อมาจาก class เดิม



Sub Class & Super Class

- Sub class
- Super class



```

class CCircle: public Circle {
private:
    string color;
public:
    CCircle() { color="white"; }
    CCircle(string c) { color=c; }
    CCircle(double r);
    CCircle(double r,string c);
    string getColor();
    void setColor(string c) { color=c; }
};

CCircle::CCircle(double r):Circle(r) {}

CCircle::CCircle(double r,string c):Circle(r) {
    color=c;
}

string CCircle::getColor() {
    return color;
}

```

```

int main() {
    Circle c1,c2(7);
    CCircle cc3,cc4(6),cc5("green"),cc6(4,"red");

    cout<<"The color of cc5: "
         <<cc5.getColor()<<"\n";

    cc3.setColor("pink");
    cout<<"The color of cc3: "
         <<cc3.getColor()<<"\n";

    cout<<"The area of cc6: "
         <<cc6.getArea()<<"\n";

    cc4.setRadius(5);
    cout<<"The Diameter of cc4: "
         <<cc4.getDiameter()<<"\n";

}

```

```

class Time {
    private:
        int hour, minute, second;
    public:
        Time();
        Time(int h, int m, int s);
        void printTime();
        void setTime(int h, int m, int s);
        int getHour(){return hour;}
        int getMinute(){return minute;}
        int getSecond(){return second;}
        void setHour(int h){hour = h;}
        void setMinute(int m){minute = m;}
        void setSecond(int s){second = s;}
};

Time::Time() {
    hour = minute = second = 0;
}

Time::Time(int h, int m, int s) {
    hour = h; minute = m; second = s;
}

```

```

void Time::setTime(int h, int m, int s) {
    hour = h; minute = m; second = s;
}

string twoDigit(int t) {
    if(t < 10)
        return "0" + to_string(t);
    else
        return to_string(t);
}

void Time::printTime() {
    cout << "The time : ("
        << twoDigit(hour) << ":" << twoDigit(minute)
        << ":" << twoDigit(second) << ")" << endl;
}

int main() {
    Time t(3, 55, 54);
    t.printTime();
    t.setHour(7);
    t.setMinute(17);
    t.setSecond(43);
    t.printTime();
}

```

Output:

The time : (03:55:54)

The time : (07:17:43)

Polymorphism


- หลักการเขียนโปรแกรมเพื่อ
 - ให้ฟังก์ชันสามารถรองรับข้อมูลนำเข้าได้หลายรูปแบบ
 - ตัวแปรชื่อเดียวสามารถแทนข้อมูลได้หลายประเภท
- **Overloading**
 - เมธอดชื่อเดียวกัน แต่มี parameter แตกต่างกัน เมื่อเรียกใช้งานเมธอดโปรแกรมจะไปเรียกเมธอดที่มี parameter ตรงกัน
- **Overriding**
 - การ implement เมธอดขึ้นมาแทนใหม่ โดยทั้งชื่อเมธอด, modifier, return type, parameter จะยังคงเหมือนเดิม มีแค่การดำเนินการข้างในเท่านั้นที่จะถูกเปลี่ยนแปลงไป

Overloading

```
int sum(int a,int b) {  
    return (a+b);  
}
```

```
int sum(int a,int b,int c) {  
    return (a+b+c);  
}
```

```
int main() {  
    int sum1,sum2;  
    sum1=sum(15,25);  
    sum2=sum(10,20,30);  
}
```




Overriding

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

```
class Animal {
private:
    int leg=0;
public:
    int getLeg() { return leg; }
    void setLeg(int l) { leg=l; }
    string getType() { return "animal"; }
};
```

```
class Cat: public Animal {
private:
    string color;
public:
    Cat() { setLeg(4); }
    Cat(string c) { setLeg(4); color=c; }
    string getColor() { return color; }
    string getType() { return "cat"; }
};
```



```
int main() {
    Animal a;
    Cat c;
    cout<<a.getType()<<" has "<<a.getLeg()<<" leg(s).\n";
    cout<<c.getType()<<" has "<<c.getLeg()<<" leg(s).\n";
}
```


Destructors

- เป็น member method พิเศษ ใช้เพื่อให้ระบบคืน memory ของ object
 - สามารถนำ memory ไปให้กับ object ที่สร้างใหม่ได้
 - ใช้เพื่อปลดปล่อยตำแหน่งของ dynamic memory ที่ถูกสร้างขึ้น
- ชื่อเดียวกับ class
 - เริ่มต้นด้วย tilde (~)
- ไม่มี arguments
- ไม่มีการ return value
- ไม่มี overloaded

```

class Time {
    private:
        int *hour,*minute,*second;
    public:
        Time();
        Time(int h,int m,int s);
        void printTime();
        void setTime(int h,int m,int s);
        int getHour(){return *hour;}
        int getMinute(){return *minute;}
        int getSecond(){return *second;}
        void setHour(int h){*hour = h;}
        void setMinute(int m){*minute = m;}
        void setSecond(int s){*second = s;}
        ~Time();
};

Time::Time() {
    hour = new int; minute = new int;
    second = new int;
    *hour = *minute = *second = 0;
}

```

```

Time::Time(int h,int m,int s) {
    hour = new int; minute = new int;
    second = new int;
    *hour = h; *minute = m; *second = s;
}

void Time::setTime(int h,int m,int s) {
    *hour = h; *minute = m; *second = s;
}

string twoDigit(int t) {
    if(t<10) return "0"+to_string(t);
    else return to_string(t);
}

void Time::printTime() {
    cout<<"The time : (" <<twoDigit(*hour)
        <<":"<<twoDigit(*minute)<<":"
        <<twoDigit(*second)<<")\n";
}

Time::~~Time() {
    delete hour; delete minute; delete second;
}

int main() {
    Time *t;
    t = new Time(3,55,54); t->printTime();
    t->setHour(7); t->setMinute(17);
    t->setSecond(43); t->printTime();
    delete t;
}

```

Output:

The time : (03:55:54)

The time : (07:17:43)

EXERCISES

Exercises I (Rectangle)

Rectangle

- height:double=1.0
- width:double=1.0
- color:string="black"

+Rectangle(height:double,width:double,color:string)
+Rectangle(height:double,width:double)
+Rectangle(side:double)
+Rectangle(color:string)
+Rectangle()

+getHeight():double
+getWidth():double
+getColor():string
+getArea():double
+getPerimeter():double
+setHeight(double):void
+setWidth(double):void
+setSide(double):void
+setColor(string):void
+isSimilar(Rectangle):boolean

// Similar : same color, height&width (swappable)

Exercises II (Triangle)

Triangle

- **height**:double=1.0
- **base**:double=1.0
- **color**:string="black"
- **equilateral**:boolean=false

+**Triangle**(base:double,height:double,color:string)
+**Triangle**(base:double,height:double)
+**Triangle**(side:double,color:string)
+**Triangle**(side:double)
+**Triangle**(color:string)
+**Triangle**(triangle:Triangle)
+**Triangle**()

+**getHeight**():double
+**getBase**():double
+**getColor**():string
+**getArea**():double
+**getPerimeter**():double
+**isEquilateral**():boolean
+**setHeight**(double):void
+**setBase**(double):void
+**setColor**(string):void
+**isColor**(string):boolean
+**isSameColor**(Triangle):boolean
+**isBigger**(Triangle):boolean
+**isSmaller**(Triangle):boolean
+**compareTo**(Triangle):double
+**copyFrom**(Triangle):void

// return 0.0 as perimeter if equilateral==false

Exercises III (Fraction Number)

Fraction

- `num:int=0` *// numerator*
- `denom:int=1` *// denominator*
- + `Fraction(n:int,d:int)`
- + `Fraction()`
- + `setFraction(n:int,d:int):void`
- + `getFraction(&n:int,&d:int):void`
- + `getNum():int`
- + `getDenom():int`
- + `add(other:Fraction):Fraction`
- + `sub(other:Fraction):Fraction`
- + `mul(other:Fraction):Fraction`
- + `div(other:Fraction):Fraction`
- + `isEqualTo(other:Fraction):boolean`
- + `print():void`
- + `str():string`

```
int main() {  
    Fraction f1(2,3),f2(4,7),f3,f4;  
    int x,y;  
    f2.getFraction(x,y);  
    f3.setFraction(x,y);  
    f4=f3.mul(f1);  
    f1.print(); cout<<" x ";  
    f3.print(); cout<<" = ";  
    f4.print(); cout<<endl;  
    Fraction *f5=new Fraction(1,2);  
    cout<<f4.str()+" / "+f5->str();  
    cout<<" = "+f4.div(*f5).str()+"\n";  
    if(f2.isEqualTo(f3))  
        cout<<"Equal.\n";  
    else  
        cout<<"Not equal.\n";  
}
```

$2/3 \times 4/7 = 8/21$
 $8/21 / 1/2 = 16/21$
Equal.

Exercises IV (Animal)

Animal

- `type:string=""`
- `color:string=""`
- `weight:double=0.0`

- + `Animal(type:string,color:string,weight:double)`
- + `getType():string`
- + `getColor():string`
- + `getWeight():double`
- + `setType(type:string):void`
- + `setColor(color:string):void`
- + `setWeight(weight:double):void`
- + `print():void`

```
10
cat 5 green
dog 9 brown
cat 2 gray
dog 4 black
cat 3 white
bird 0.5 blue
cat 3 gray
dog 6 brown
bird 0.7 blue
bird 0.9 white
cat gray
2
green cat 5Kg.
gray cat 2Kg.
white cat 3Kg.
gray cat 3Kg.
Total weight : 13Kg.
```

Animals

- `animals:vector<Animal>`
- + `addAnimal(a:Animal):void`
- + `getAnimal(i:int):Animal`
- + `getAnimals(type:string):Animals`
- + `count():int`
- + `count(type:string):int`
- + `count(type:string,color:string):int`
- + `sumWeight():double`
- + `print():void`

```
int main() {
    int no; Animals zoo;
    cin>>no;
    for(int i=0;i<no;i++) {
        string typ,col; double wt;
        cin>>typ>>wt>>col;
        Animal a(typ,col,wt);
        zoo.addAnimal(a);
    }
    zoo.print();
    string ftyp,fcol;
    cin>>ftyp>>fcol;
    cout<<zoo.count(ftyp,fcol)<<"\n";
    Animals va=zoo.getAnimals(ftyp);
    for(int i=0;i<va.count();i++)
        va.getAnimal(i).print();
    cout<<"Total weight : "<<va.sumWeight()
        <<" G.\n";
}
```