# C++ OOP
# ครั้งที่ 4

# หัวข้อ

1. คุณสมบัติของการพ้องรูป (Polymorphism)
2. ประเภทของการพ้องรูป

# แนวคิดของ OOP

- กุญแจหลักของ OOP คือ abstraction ซึ่งประกอบด้วยแนวคิด 3 ประการ

1. Encapsulation เป็นสร้างวัตถุที่มีทั้งข้อมูลและวิธีการทำงานครบในตัวของออบเจกต์เอง อีกทั้งยังซ่อนรายละเอียดของแต่ละองค์ประกอบไว้ภายใน
2. Inheritance เป็นคุณสมบัติในการเขียนโปรแกรมเชิงวัตถุที่เรียกว่าคุณสมบัติการสืบทอด โดยที่คลาสสามารถสืบทอด attribute และ method จากคลาสหลัก (base class) ไปยังคลาสย่อย (derived class)
3. Polymorphism การกำหนดให้วัตถุสามารถมีได้หลายรูปแบบตามกรณีเฉพาะต่าง ๆ ซึ่งเกิดจากการสืบทอดจาก super class และมันยังคงรักษาสภาพและคุณสมบัติของ super class

https://linux.thai.net/~thep/docs/ และ http://marcuscode.com/lang/java/

# Polymorphism

- **Polymorphism หรือการพ้องรูป**
  - เป็นคุณสมบัติของ OOP ที่มีการตอบสนองด้วยแมสเสจเดียวกันมักพบในเรื่องการถ่ายทอดคุณสมบัติ (inheritance) หรือมี interface เดียวกัน
  - ซึ่งเกิดจากการสืบทอดจาก super class ไปยัง sub class แต่ยังคงรักษาสภาพและคุณสมบัติของ super class อยู่
- **สามารถแบ่งออกเป็น 2 ประเภท คือ**

1. Compile time Polymorphism
2. Runtime Polymorphism

https://www.geeksforgeeks.org/polymorphism-in-c/

# Polymorphism

1. Compile time polymorphism: แบ่งออกเป็น 2 ประเภทคือ overloading or operator overloading.
   - 1.1 Function Overloading: ในการทำงานของคลาสอาจจะมีเมธทอดจำนวนมากและบางครั้งเมธทอดนั้นจะมีชื่อเหมือนกันแต่มีความแตกต่างที่จำนวนของ arguments หรือ ชนิดของข้อมูลของ argument ที่ใช้ในเมธทอด

```
int square(int x) {
  return x*x;
}
double square(double x) {
  return x*x;
}
```

```
int rectangle(int x,int y) {
  return x*y;
}
double rectangle(double x) {
  return x*x;
}
```

https://www.geeksforgeeks.org/polymorphism-in-c/

```
1.  #include <iostream>
2.  #include <iomanip>
3.  using namespace std;
4.  class Number{
5.      public:
6.          void func(int num){
7.              cout << "value of num is " << num<< endl;
8.          }
9.          void func(double num) {
10.                 cout<<fixed<<setprecision(3);
11.             cout << "value of num is " << num << endl;
12.         }
13.         void func(int num, int val) {
14.             cout << "value of x and y is " << num << ", " << val << endl;
15.         }
16. };
17. main()
18. {
19.     Number obj1;
20.     obj1.func(5);
21.     obj1.func(5.5);
22.     obj1.func(20,40);
23. }
```



https://www.geeksforgeeks.org/polymorphism-in-c/

# Polymorphism

1. **Compile time polymorphism**: แบ่งออกเป็น 2 ประเภทคือ overloading or operator overloading.
   - 1.2 Operator Overloading
     - สำหรับภาษา C++ จะอนุญาติให้มี overload operators.
     - ตัวอย่าง
       - operator (‘+’) ใช้เชื่อม string class จำนวน 2 strings
       - operator (‘+’) ใช้ + หาผลบวกของ operand และเก็บค่าผลบวกนั้นที่ operand เดิม

https://www.geeksforgeeks.org/polymorphism-in-c/

# Overloadable/Non-overloadable Operators

## Overloadable Operators

| | | | | | |
|---|---|---|---|---|---|
| + | - | * | / | % | ^ |
| & | \| | ~ | ! | , | = |
| < | > | <= | >= | ++ | -- |
| << | >> | == | != | && | \|\| |
| += | -= | /= | %= | ^= | &= |
| \|= | *= | <<= | >>= | [] | () |
| -> | ->* | new | new [] | delete | delete [] |

## Non-overloadable Operators

| | | | |
|---|---|---|---|
| :: | .* | . | ?: |

https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm

```
1.  #include<iostream>
2.  #include <iomanip>
3.  using namespace std;
4.
5.  class Complex {
6.  private:
7.      int real, imag;
8.  public:
9.      Complex(int r = 0, int i =0)  {real = r;   imag = i;}
10.
11.     // This is automatically called when '+' is used with
12.     // between two Complex objects
13.     Complex operator + (Complex const &obj) {
14.         Complex res;
15.         res.real = real + obj.real;
16.         res.imag = imag + obj.imag;
17.         return res;
18.     };
19.     void print() { cout << real << " + i" << imag << endl; }
20. };
21. main()
22. {
23.     Complex c1(10, 5), c2(2, 4);
24.     Complex c3 = c1 + c2; // An example call to "operator+"
25.     c3.print();
26. }
```

https://www.geeksforgeeks.org/polymorphism-in-c/

```
1.  #include <iostream>
2.  #include <iomanip>
3.  using namespace std;
4.  class Box {
5.    public:
6.      double getVolume(void) {
7.        return length * breadth * height;
8.      };
9.      void setLength( double len ) {
10.       length = len;
11.     };
12.     void setBreadth( double bre ) {
13.       breadth = bre;
14.     };
15.     void setHeight( double hei ) {
16.       height = hei;
17.     };
18.       // Overload + operator to add two Box objects.
19.     Box operator+(const Box& b) {
20.       Box box;
21.       box.length = this->length + b.length;
22.       box.breadth = this->breadth + b.breadth;
23.       box.height = this->height + b.height;
24.       return box;
25.     }
```

https://www.tutorialspoint.com/cplusplus/cpp_overloading.htm

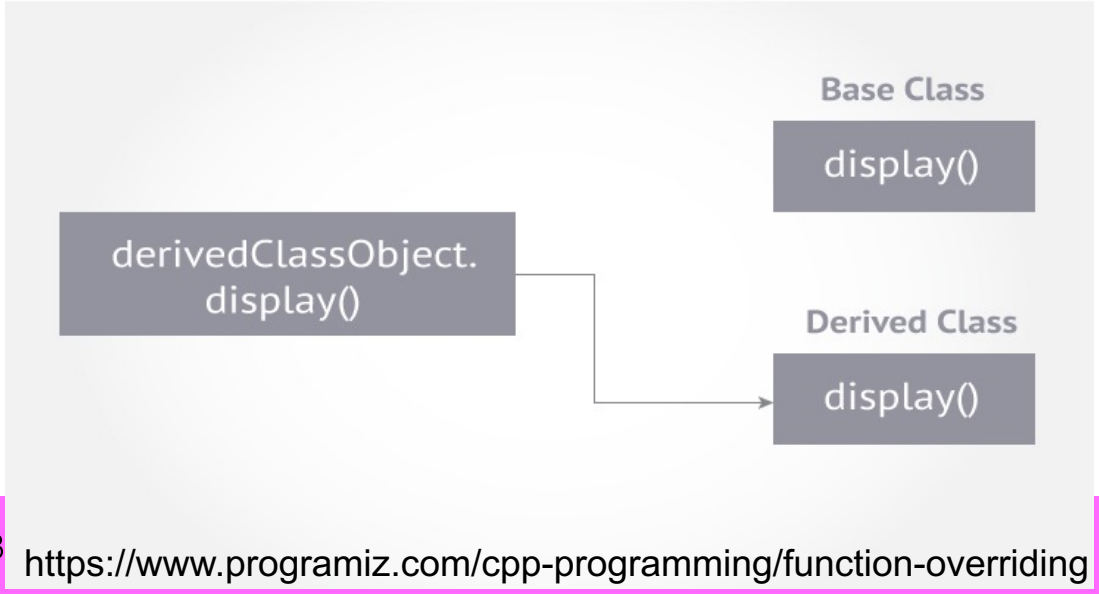
```
26.    private:
27.      double length;     // Length of a box
28.      double breadth;    // Breadth of a box
29.      double height;     // Height of a box
30.  };
31.  main() {
32.    Box Box1;   // Declare Box1 of type Box
33.    Box Box2;   // Declare Box2 of type Box
34.    Box Box3;   // Declare Box3 of type Box
35.    double volume = 0.0;
36.
37.    // box 1 specification
38.    Box1.setLength(6.0);
39.    Box1.setBreadth(7.0);
40.    Box1.setHeight(5.0);
41.
42.    // box 2 specification
43.    Box2.setLength(12.0);
44.    Box2.setBreadth(13.0);
45.    Box2.setHeight(10.0);
```

```
46.    cout<<fixed<<setprecision(3);
47.    // volume of box 1
48.    volume = Box1.getVolume();
49.    cout << "Volume of Box1 : " << volume <<endl;
50.
26.    // volume of box 2
27.    volume = Box2.getVolume();
28.    cout << "Volume of Box2 : " << volume <<endl;

29.    // Add two object as follows:
30.    Box3 = Box1 + Box2;

31.    // volume of box 3
32.    volume = Box3.getVolume();
33.    cout << "Volume of Box3 : " << volume <<endl;

34.  }
```

# Polymorphism

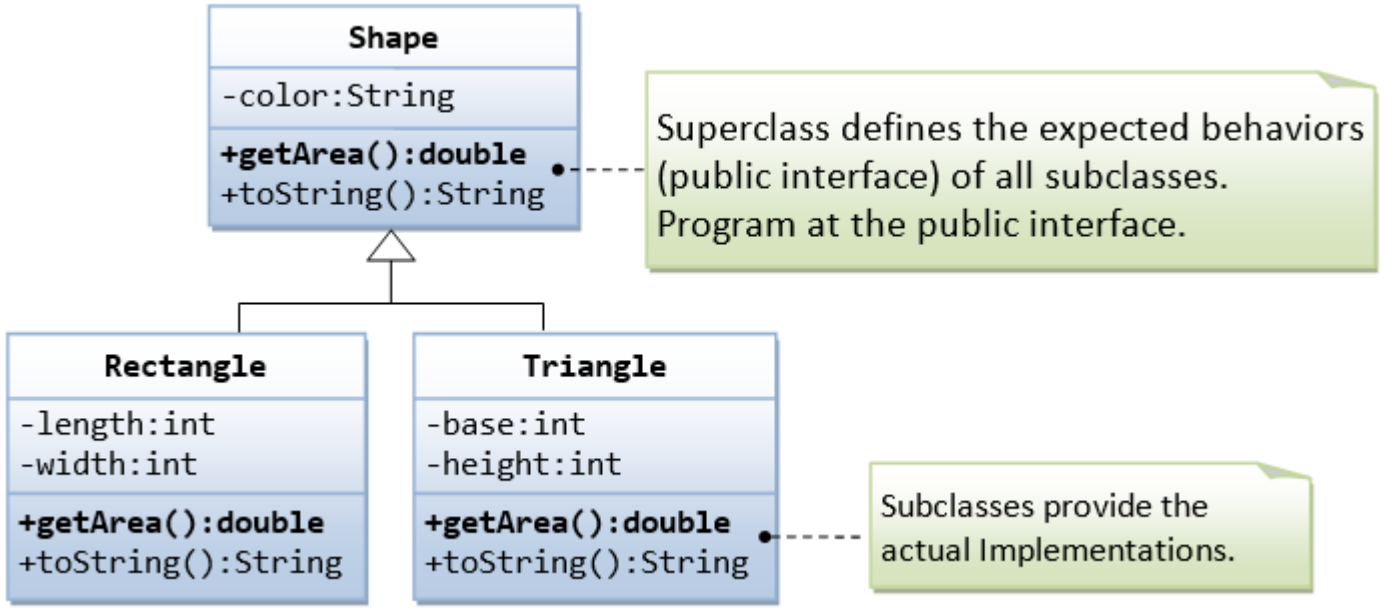
2. Runtime polymorphism: This type of polymorphism is achieved by Function Overriding.

- 1.Function overriding
  - พิจารณากรณีที่ sub class หรือ derived class มีการนิยามหรือมีเมธทอดเหมือนกับ super class หรือ base class นั่นเอง.
  - ดังนั้นการเกิดกรณีนี้จะทำให้เมธทอดของ base class ถูกทำซ้ำหรือเรียกว่า overridden



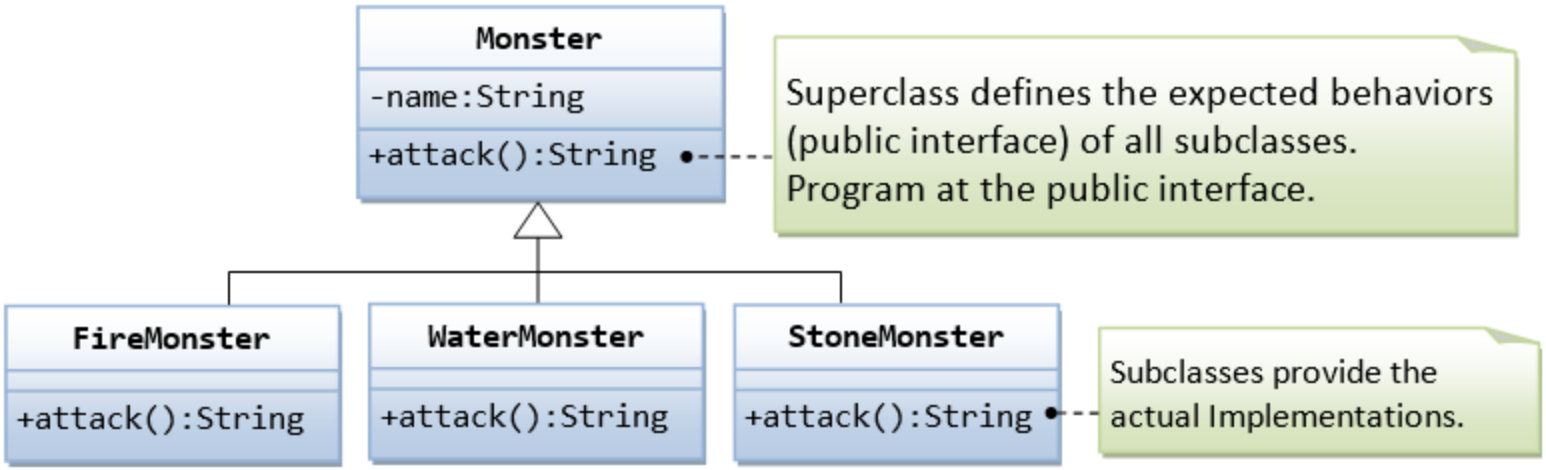

https://www.programiz.com/cpp-programming/function-overriding

# UML Class Diagram




https://www.ntu.edu.sg/home/ehchua/programming/java/J3b_OOPInheritancePolymorphism.html

# UML Class Diagram




https://www.ntu.edu.sg/home/ehchua/programming/java/J3b_OOPInheritancePolymorphism.html

```
1.  #include <iostream>
2.  using namespace std;
3.  class Parent// Base class
4.  {
5.      public:
6.      void print()
7.      {
8.          cout << "The Parent print function was called" << endl;
9.      };
10. };
26. // Derived class
27. class Child : public Parent
28. {
29.     public:
30. // definition of a member function already present in Parent
31.     void print()
32.     {
33.         cout << "The child print function was called" << endl;
34.     };
35. };
```

```
21. main()
22. {
26.     //object of parent class
27.     Parent obj1;
28.
29.     //object of child class
30.     Parent obj2 = Child();
31.
32.     // obj1 will call the print function in Parent
33.     obj1.print();
34.
35.     // obj2 will override the print function in Parent
36.     // and call the print function in Parent
37.     obj2.print();
38. }
```

```
#include <iostream>
#include <string>
using namespace std;
// Base class
class Animal {
  public:
    void soundAnimal() {
      cout << "The animal makes a sound \n";
    }
};
// Derived class
class Pig : public Animal {
  public:
    void soundAnimal() {
      cout << "The pig says: wee wee \n";
    }

};
// Derived class
class Dog : public Animal {
  public:
    void soundAnimal() {
      cout << "The dog says: bow wow \n";
    }
};
```

.....

```
int main() {
  Animal myAnimal;
  Animal dog = Dog();
  Pig myPig;
  Dog myDog;

  myAnimal.soundAnimal();
  myPig.soundAnimal();
  myDog.soundAnimal();
  dog.soundAnimal();
  return 0;
}
```

```
1.  #include <iostream>
2.  using namespace std;
3.  class Base {
4.  public:
5.          void display() {cout << "Called (Base) Class function\n";}
6.          void print(){ cout << "Called (Base) print function\n\n";}
7.  };
8.  class Child : public Base {
9.  public:
10.         void display() { cout << "Called (Child) Display Function\n"; }
11.         void print() { cout << "Called (Child) print Function\n\n"; }
12.         void plus(){ cout << "Called (Child) plus Function\n\n";}
13.         Child(){cout << "Coonstruct Child\n";}
14. };
```

https://www.geeksforgeeks.org/cpp-polymorphism/

```
15.  int main()
16.  {
17.          Base base;
18.          Child child;
19.          Base newBase = Child();

20.          base.display();
21.          base.print();
22.
23.          child.display();
24.          child.print();
25.          child.plus();
26.
27.          newBase.display();
28.          newBase.print();
29.  }
```



https://www.geeksforgeeks.org/cpp-polymorphism/

# คำถาม

```
string getColor() {   // Member function (Getter)
   return color;
}
```

```
" Color=" << c1.getColor() <
```

1) จงเขียนโปรแกรมคำนวณพื้นที่ของรูปทรง โดยรายละเอียดของแต่ละคลาสให้เป็นตาม UML ด้านล่าง