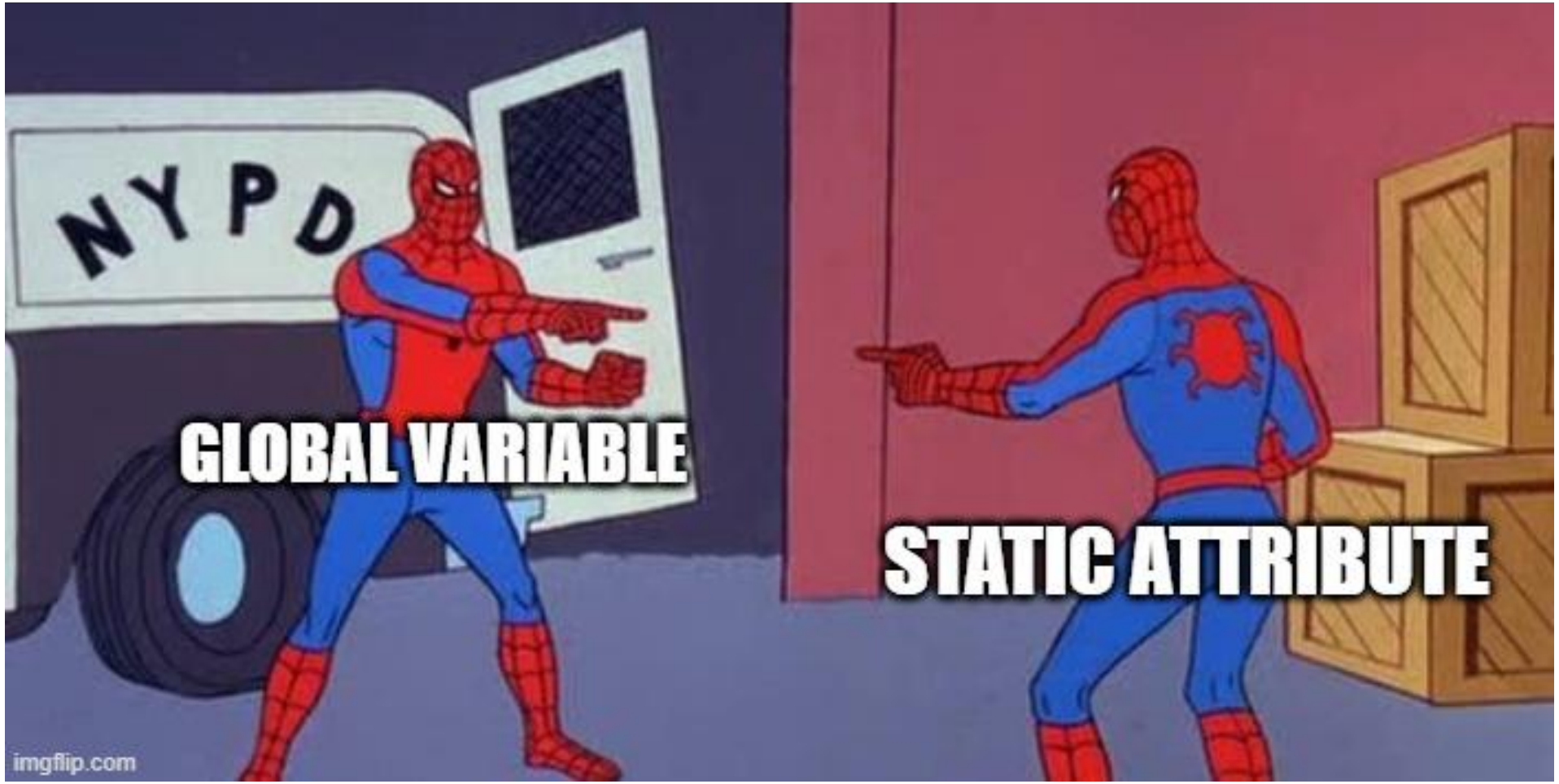
OOP & data struct

6. Utility

BY SOMSIN THONGKRAIRAT

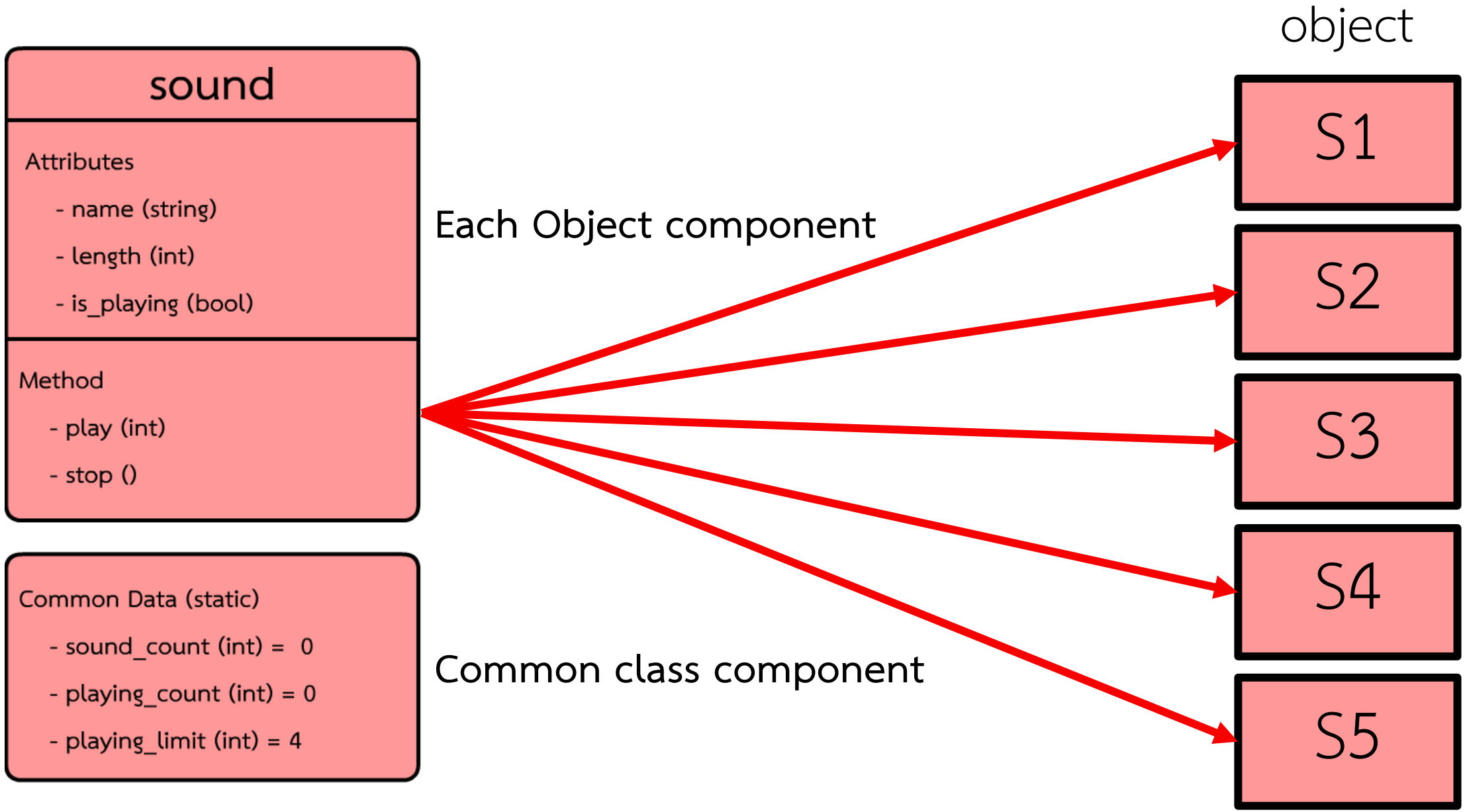

NYPD
GLOBAL VARIABLE
STATIC ATTRIBUTE
imgflip.com

Utility

- static variable
- Templates

static variable

- one variable for a class (like a global variable)
- เป็นตัวแปรตัวเดียวที่ใช้ได้กับทั้ง class (เหมือนกับ global variable)


object
sound
Attributes
- name (string)
- length (int)
- is_playing (bool)
Method
- play (int)
- stop ()
Common Data (static)
- sound_count (int) = 0
- playing_count (int) = 0
- playing_limit (int) = 4
Each Object component
Common class component
S1
S2
S3
S4
S5

Syntax

Static variable declaration

```
static [variable declaration];
```

```
class sound{
public :
    static int playing_limit;
    static int playing_count;
    static int sound_count;
```

usage

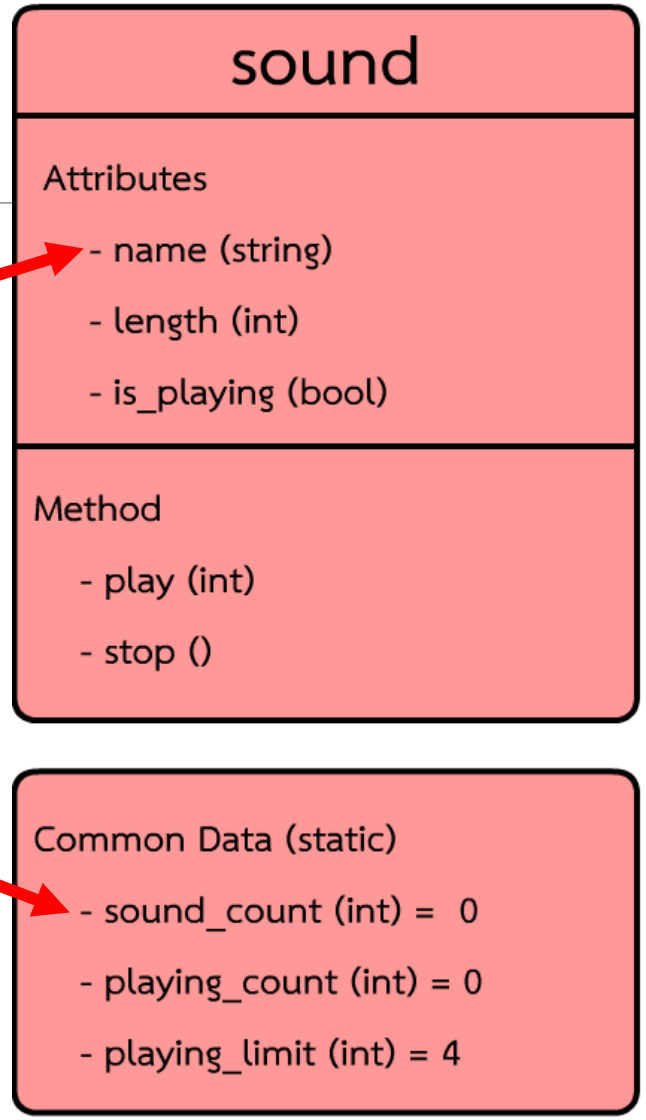
```
int sound::sound_count = 0; // init
int sound::playing_count = 0; // init
int sound::playing_limit = 4; // init

int main(){

    cout << sound::sound_count << endl;
```

Result : 0

- Must initialize outside of main
- ต้องกำหนดค่าเริ่มต้นนอกฟังก์ชัน main

Example

```
sound(string _name){
    name = _name;
    sound_count++;
}
```



- Unlike normal attribute static member call common data of its class
- ไม่เหมือนกับตัวแปรทั่วไป ตัวแปร static จะไปเรียกข้อมูลที่ใช่ร่วมกันใน class

```
cout << sound::sound_count << endl;

sound s1("Som san");
sound s2("Jai sang ma");
sound s3("s3");
sound s4("s4");
sound s5("s5");
sound s6("s6");

cout << sound::sound_count << endl;
```

Common Data (static)

- sound_count (int) = 0
- playing_count (int) = 0
- playing_limit (int) = 4

Common Data (static)

- sound_count (int) = 6
- playing_count (int) = 0
- playing_limit (int) = 4

Result : 0

6

```
sound(string _name){
    name = _name;
    sound_count++;
    cout << "this sound number : " << sound_count << endl;
}
```

Code :

```
cout << sound::sound_count << endl;
sound s1("Som san");
sound s2("Jai sang ma");
sound s3("s3");
sound s4("s4");
sound s5("s5");
sound s6("s6");
cout << sound::sound_count << endl;
```

Result :

0

this sound number : 1

this sound number : 2

this sound number : 3

this sound number : 4

this sound number : 5

this sound number : 6

6

warning

```
cout << sound::sound_count << endl;
sound s1("Som san");
sound s2("Jai sang ma");
sound s3("s3");
if(true){
    sound s7("song yang bad");
}
sound s4("s4");
sound s5("s5");
sound s6("s6");
cout << sound::sound_count << endl;
```

```
0
this sound number : 1
this sound number : 2
this sound number : 3
this sound number : 4
this sound number : 5
this sound number : 6
this sound number : 7
7
```

Quiz

HOW TO SOLVE IT?

Play method

```
void play(){
        if(is_playing == true){
            cout << name << " is still playing " << endl;
            return;
        }
        else{
            is_playing = true;
            playing_count++;
            cout << "playing " << name << endl;
        }
    }
```

Stop method

```
void stop(){
    if(is_playing == false){
        cout << name << " is not playing " << endl;
    }
    else{
        is_playing = false;
        playing_count--;
    }
}
```

Another example

Sound mixer

- Can play only 4 sound at a time
- สามารถเล่นได้แค่ 4 เสียงในหนึ่งๆ เวลา

Solution

- create global variable and count (non - OOP)
- using data struct such as queue stack (not elegant!)
- using static variable to count playing sound (very elegant!)
- สร้างตัวแปร global เพื่อนับ (non - OOP)
- ใช้ data struct เช่น คิว หรือ stack (not elegant!)
- ใช้ตัวแปร static เพื่อนับจำนวนของเพลงที่เล่น (very elegant!)

```
int sound::sound_count = 0; // init
int sound::playing_count = 0; // init
int sound::playing_limit = 4; // init

int main(){

    cout << sound::sound_count << endl;
```

Set limit

Added to play

```
if(playing_count >= playing_limit){
    cout << "playlist is full, can't play " << name << endl;
}
```

```
void play(){
        if(is_playing == true){
            cout << name << " is still playing " << endl;
            return;
        }

        if(playing_count >= playing_limit){
            cout << "playlist is full, can't play " << name << endl;
        }
        else{
            is_playing = true;
            playing_count++;
            cout << "playing " << name << endl;
        }
    }
```

```
s1.play(); // playing Som san
s2.play(); // playing Jai sang ma
s3.play(); // playing s3
s4.play(); // playing s4
s5.play(); // playlist is full, can't play s5
s6.play(); // playlist is full, can't play s6
s1.stop();
s6.play(); // playing s6
s1.stop(); // Som san is not playing
s2.stop();
s2.stop(); // Jai sang ma is not playing
cout << sound::playing_count << endl; // 3
```

Static method

```
static void print(){
    cout << "total sound is : [" << sound_count << "] now playing ["
    << playing_count << "]" << endl;
}

static int get_playing_count(){
    return playing_count;
}

static int get_sound_count(){
    return sound_count;
}
```

And make static attribute as private !

เปลี่ยน static attribute ให้เป็น private ได้!

Static member

- can be both attribute or method and can modify as public or private
- the common component of whole class (like global variable or global function)

- สามารถมีได้ทั้ง attribute และ method และสามารถ modify ให้เป็น public หรือ private
- เป็นส่วนประกอบที่ใช้ร่วมกันทั้ง class (เหมือนตัวแปร global และ function)

Templates

- allow to create function that can custom datatype inside a function

- สามารถสร้าง function ที่กำหนดชนิดของข้อมูล datatype ที่ใช้ใน function ได้

Syntax

```
template <class identifier,…> function_declaration;

template <typename identifier,…> function_declaration;
```

Back to our friend

```
int multiply_int_int(int a,int b){
    int c = a * b;
    return c;
}

float multiply_int_float(int a,float b){
    float c = a * b;
    return c;
}

float multiply_float_int(float a,int b){
    float c = a * b;
    return c;
}
```

Turn it into

```
template <typename myType> myType multiple(myType a ,myType b){
    myType c = a * b;
    return c;
}
```

Usage :

```
int a    = multiple<int>(3,4);
float b  = multiple<float>(3,5);
double c = multiple<double>(3,6);

cout << a << endl;
cout << b << endl;
cout << c << endl;
```

Result :

12

15

18

```
template <typename myType> myType multiple(myType a ,myType b){
    myType c = a * b;
    return c;
}
```

Replace “myType” with input type (int) / เปลี่ยน “myType. ให้เป็น type ที่ใส่เข้าไป (int)

```
int a  = multiple<int>(3,4);
```

```
int multiply(int a,int b){
    int c = a * b;
    return c;
}
```

```
float multiply(float a,float b){
    float c = a * b;
    return c;
}
```

```
float b  = multiple<float>(3,5);
double c = multiple<double>(3,6);
```

```
float multiply(double a,double b){
    float c = a * b;
    return c;
}
```

2 type name no problem!

```
template <typename T,typename U> T
multiple(T a ,U b){
    T c = a * b;
    return c;
}
```

```
int d     = multiple<float, int>(3,7);
float e   = multiple<float, float>(3,8);
double f = multiple<double, float>(3,9);

cout << d << endl;
cout << e << endl;
cout << f << endl;
```

Result :

21

24

27

Another example

```
template <typename T> void selection_sort(T
a[],int n){
    for(int i=0;i<n;i++){
        int min_idx = i;
        for(int j=i;j<n;j++){
            if(a[j] < a[min_idx]){
                min_idx = j;
            }
        }
        T tmp = a[min_idx];
        a[min_idx] = a[i];
        a[i] = tmp;
    }
}
```

```
template <typename T> void print_array(T a[],int n){
    for(int i=0;i<n;i++){
        cout << a[i] << " ";
    }
    cout << endl;
}
```

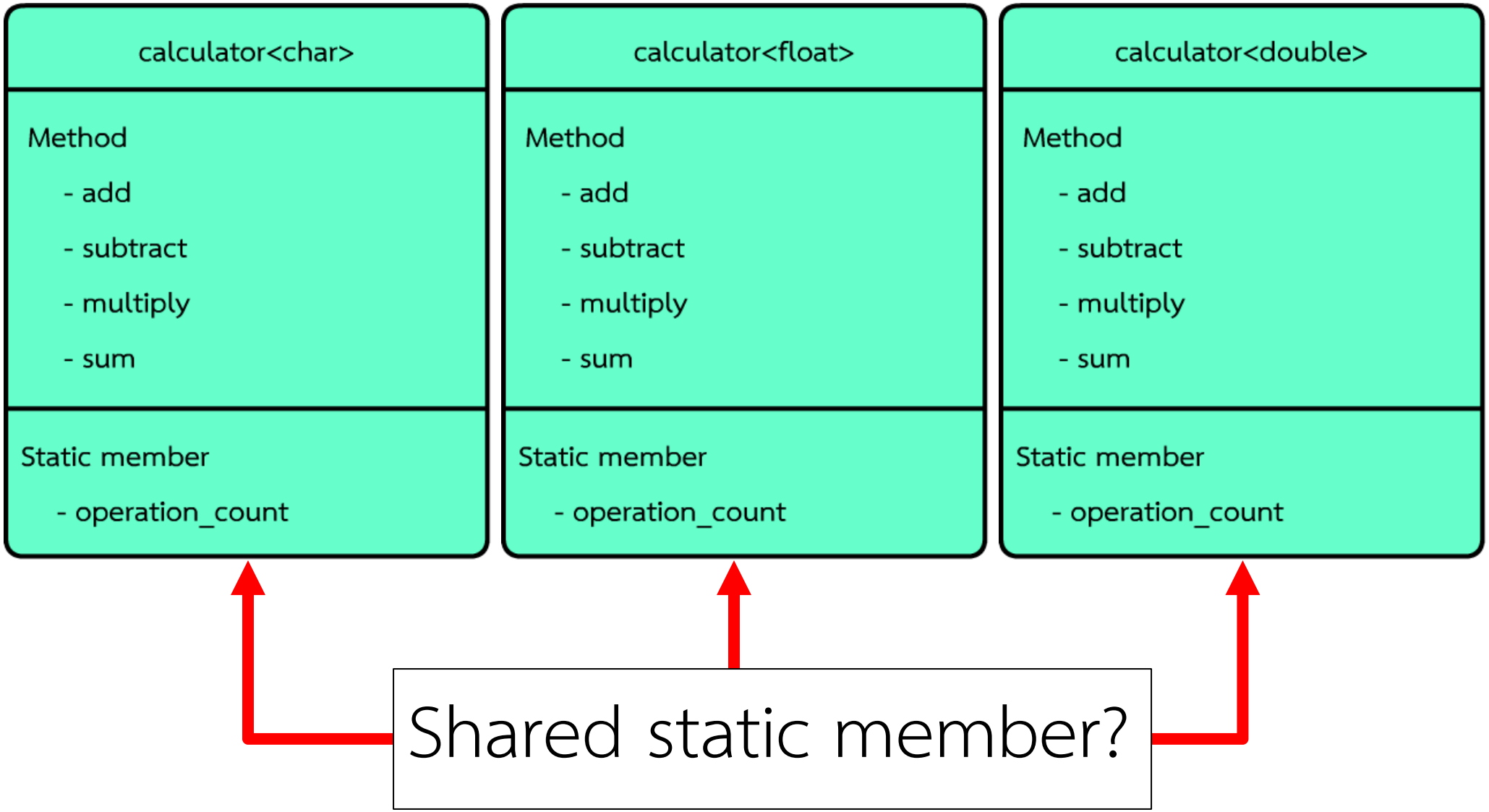
- Sort and print function can handle all of data type
- ฟังก์ชัน sort และ print สามารถรองรับตัวแปรได้ทุกชนิด

```
int my_int[] = {5,6,8,4,1,3,9,11,2};
char my_char[] = {'e','a','i','u','o'};
double my_double[] = {5.3,6.4,8.3,6.5,9.3,4.4,1.2,2.6};

selection_sort<int>(my_int,sizeof(my_int)/sizeof(int));
print_array<int>(my_int,sizeof(my_int)/sizeof(int));

selection_sort<char>(my_char,sizeof(my_char)/sizeof(char));
print_array<char>(my_char,sizeof(my_char)/sizeof(char));

selection_sort<double>(my_double,sizeof(my_double)/sizeof(double));
print_array<double>(my_double,sizeof(my_double)/sizeof(double));
```

```
1 2 3 4 5 6 8 9 11

a e i o u

1.2 2.6 4.4 5.3 6.4 6.5 8.3 9.3
```

interested question

```
template <typename T> class prototype_calculator{

    static int operate_count;
public :

     virtual T add(T x,T y)  = 0;

     virtual T subtract(T x,T y)  = 0;

     virtual T multiply(T x,T y)  = 0;

     virtual T sum(T x1,T x2=0,T x3=0,T x4=0,T x5=0) = 0;


 };
```


calculator<char>
Method
- add
- subtract
- multiply
- sum
Static member
- operation_count
calculator<float>
Method
- add
- subtract
- multiply
- sum
Static member
- operation_count
calculator<double>
Method
- add
- subtract
- multiply
- sum
Static member
- operation_count
Shared static member?

C++

- NO!
- ไม่

```
universal_calculator<int> int_calculator;
universal_calculator<float> float_calculator;
universal_calculator<double> double_calculator;
universal_calculator<char> char_calculator;
```

- Each object is in **<u>individual</u>** class
- object แต่ละตัวอยู่คนละ class กัน

```
universal_calculator<int>::operator_count
universal_calculator<float>::operator_count
universal_calculator<double>::operator_count
universal_calculator<char>::operator_count
```

- Individual item

- คนละตัวแปร

LAB