Code, Refactored

VG101 TA Group / Patrick Yao

Yao Yue 2016.7.17

Assembly

C++

Python

C

Controversial C++

A

C, No namespaces

add()

printf()

myPow()

max()

pow()

intMax()

_printf()

A

C++ Namespaces

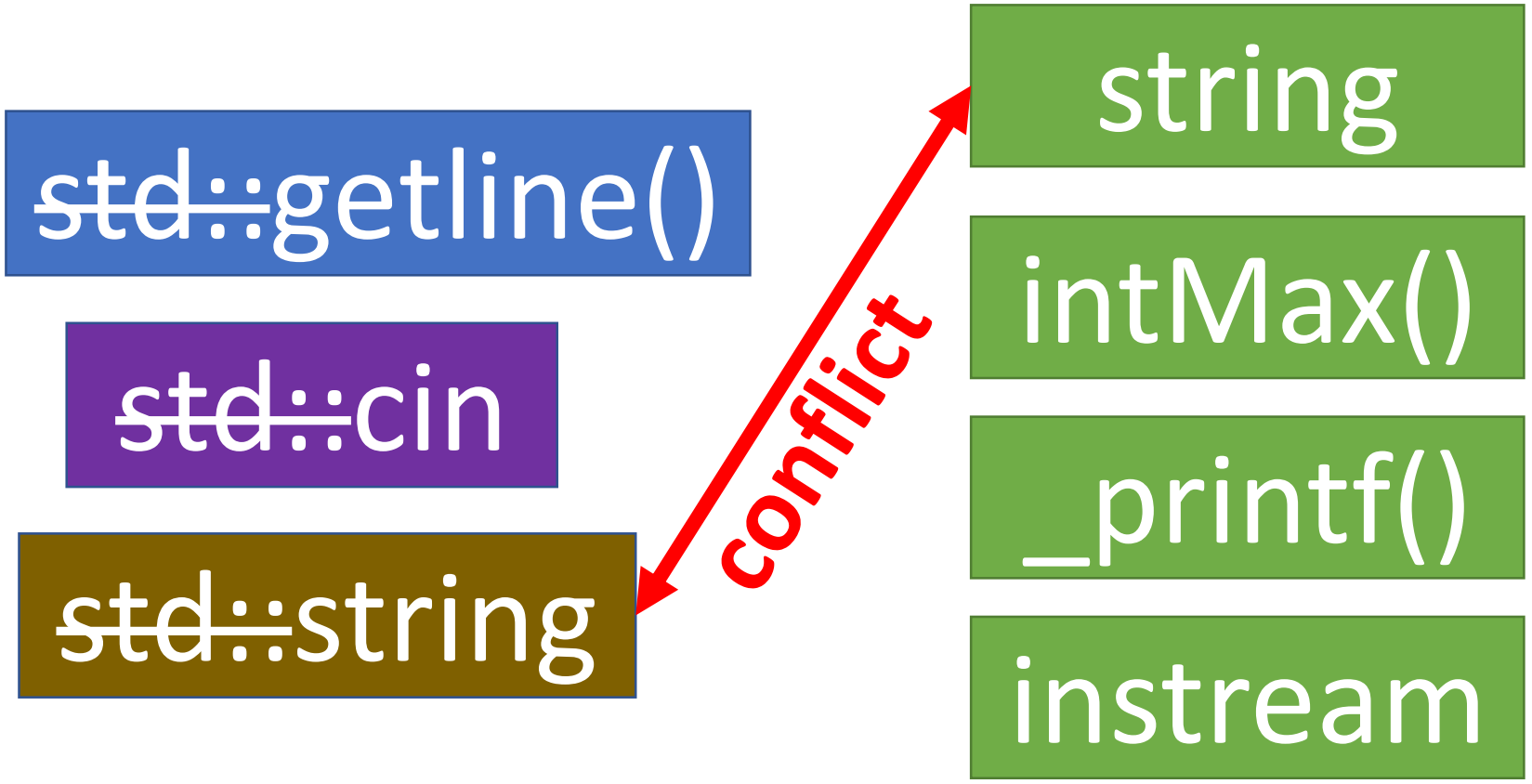
std::

- std::getline()
- std::cin
- std::string

- string
- intMax()
- _printf()
- instream

A

using namespace std;



A

Namespace, pros and cons

Namespace prevents name conflicts

But seem verbose in code

Using (import) a namespace is convenient

Brings back the potential conflict hazard

A

Cpp style IO: streams

cin, the standard input stream

Use with “>>”, the extractor.

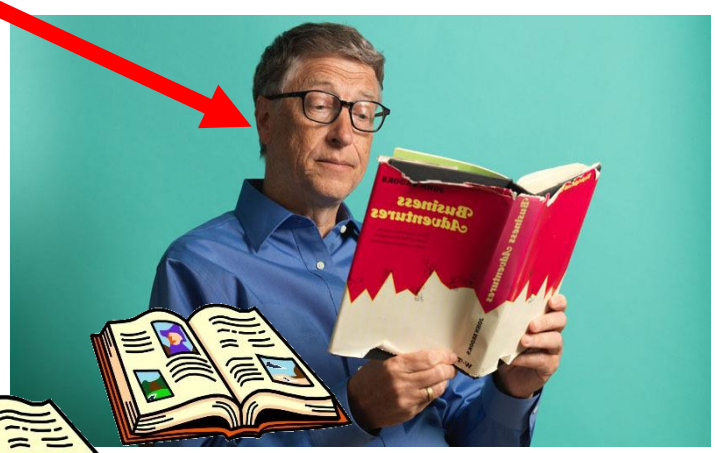
cout, the standard output stream

Use with “<<”, the insertion operator

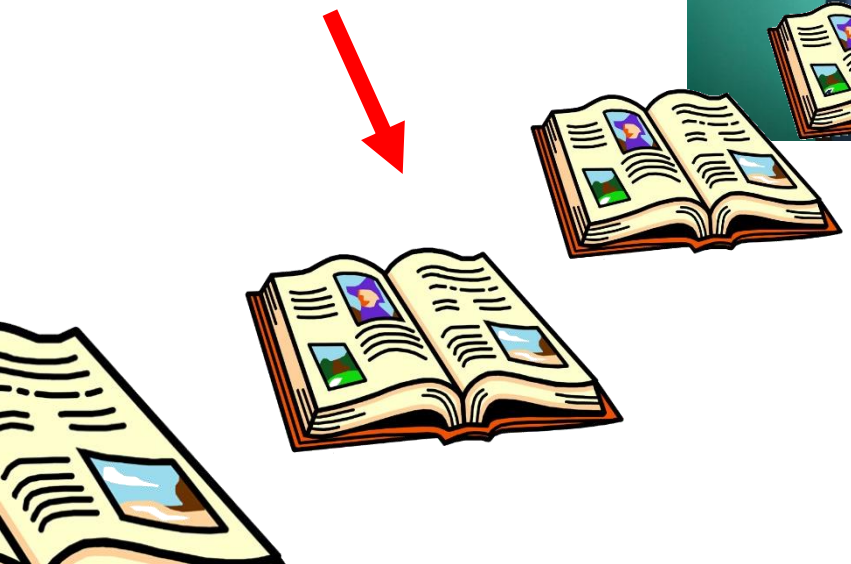
A

Visualize streams: input stream

cin



User Input



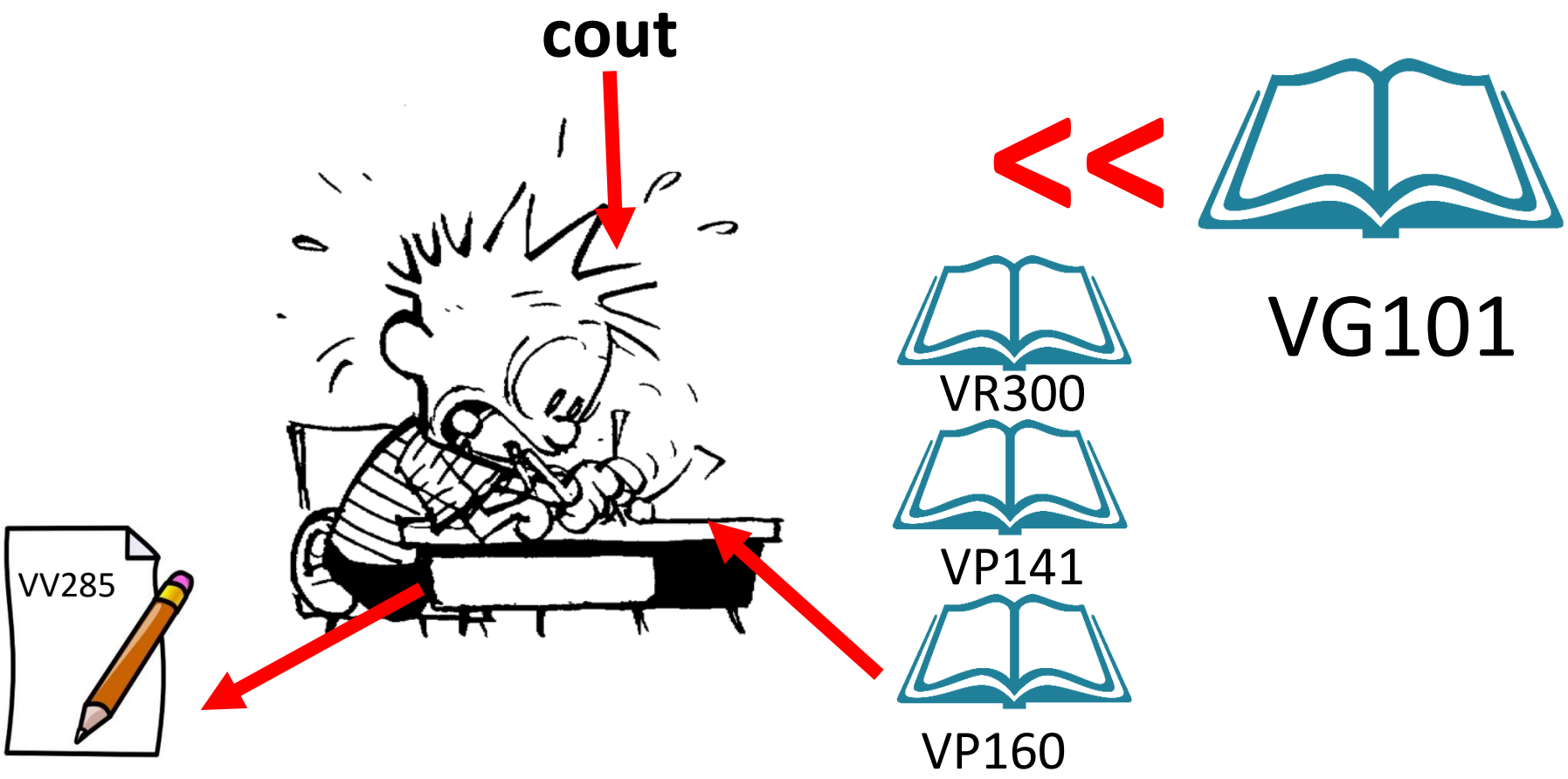
>>

"Modern Crypto"

Dr. Charlemagne

A

Visualize streams: output stream

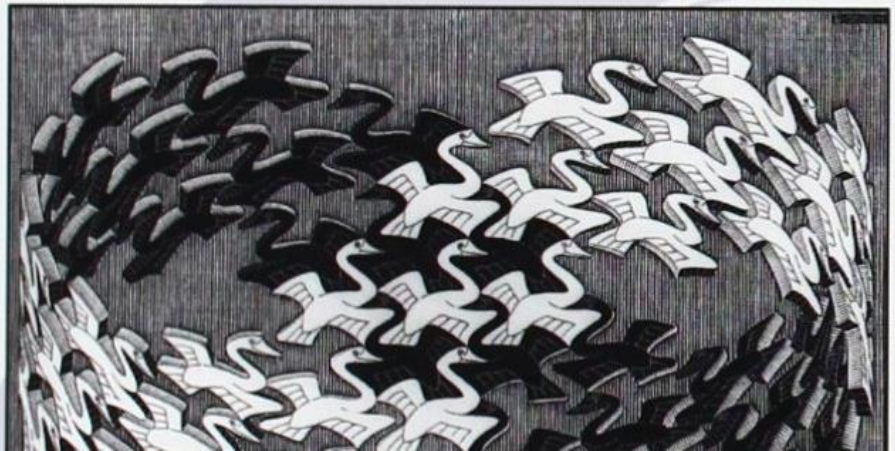


Class and Objects

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



ADDISON-WESLEY PROFESSIONAL COMPUTI

A What is computer code?

Code Manipulates data in memory!

The C programming language, Matlab, Procedural

A

What is computer code?

Code is Mathematical Functions!

Lisp, Scheme, Haskell, Methematica…..
(Functional programming)

Code is Mathematical Proof!

COQ, Formal Code Proving….

A

What is computer code?

Code models real life “things”.

Object Oriented Programming, C++, Java ….

A

Well, what about all?

A

Use code to model objects .

Data Representation

What data do we need? How to store it.

Allowed operations on (with) the data

What can we do with these data?

A

A problem in hand

An SJTU library counter

- Up Counts 1 if someone enters
- A Screen displays current count
- Should also display library name
- Clear to zero every night.

A The C-Style Solution

Design a data type to represent counter

Using structure

Design functions that use that data type

Functions are kind of “bind” to the datatype

A

```c
#include <stdio.h>

typedef struct {
    char* name;
    int count;
} Counter;

int  CounterDisplay (Counter* counter);
void CounterReset   (Counter* counter);
void CounterCountUp (Counter* counter);

int main() {
    Counter mainLibCounter =
            {"Main Library", 0};

    CounterReset(&mainLibCounter);
    for (int i = 0; i < 10; ++i) {
        CounterCountUp(&mainLibCounter);
    }
    CounterDisplay(&mainLibCounter);

    CounterReset(&mainLibCounter);
    for (int i = 0; i < 7; ++i) {
        CounterCountUp(&mainLibCounter);
    }
    CounterDisplay(&mainLibCounter);
}
```

```c
int CounterDisplay(Counter* counter) {
    printf("%s has %d people inside!\n",
           counter->name,
           counter->count);
    return counter->count;
}

void CounterReset(Counter* counter) {
    printf("%s counter has resetted!\n",
           counter->name);
    counter->count = 0;
}

void CounterCountUp(Counter* counter) {
    counter->count += 1;
    printf("We have %d people in %s\n",
           counter->count,
           counter->name);
}
```

A An Observation

A close relationship between datatype and functions

"What is counter" is answered by specifying what it can do.

Can we combine them?

Define them in a compact and clearer way?

A Solution

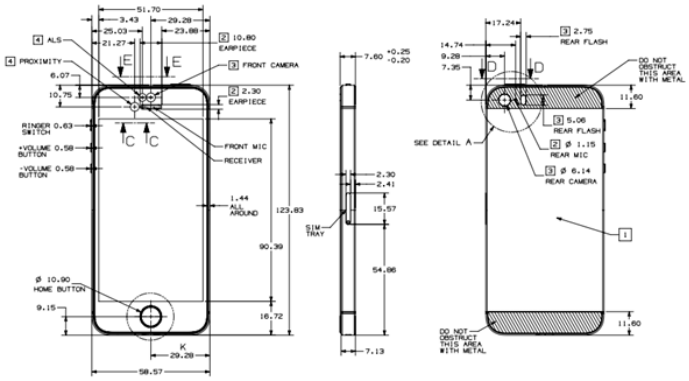
class

Datatype along with allowed functions

Instance (objects)

Instances of classes

A An analogy



Instantiate

class

Think of it as a schematic

instance

Things that built accordingly

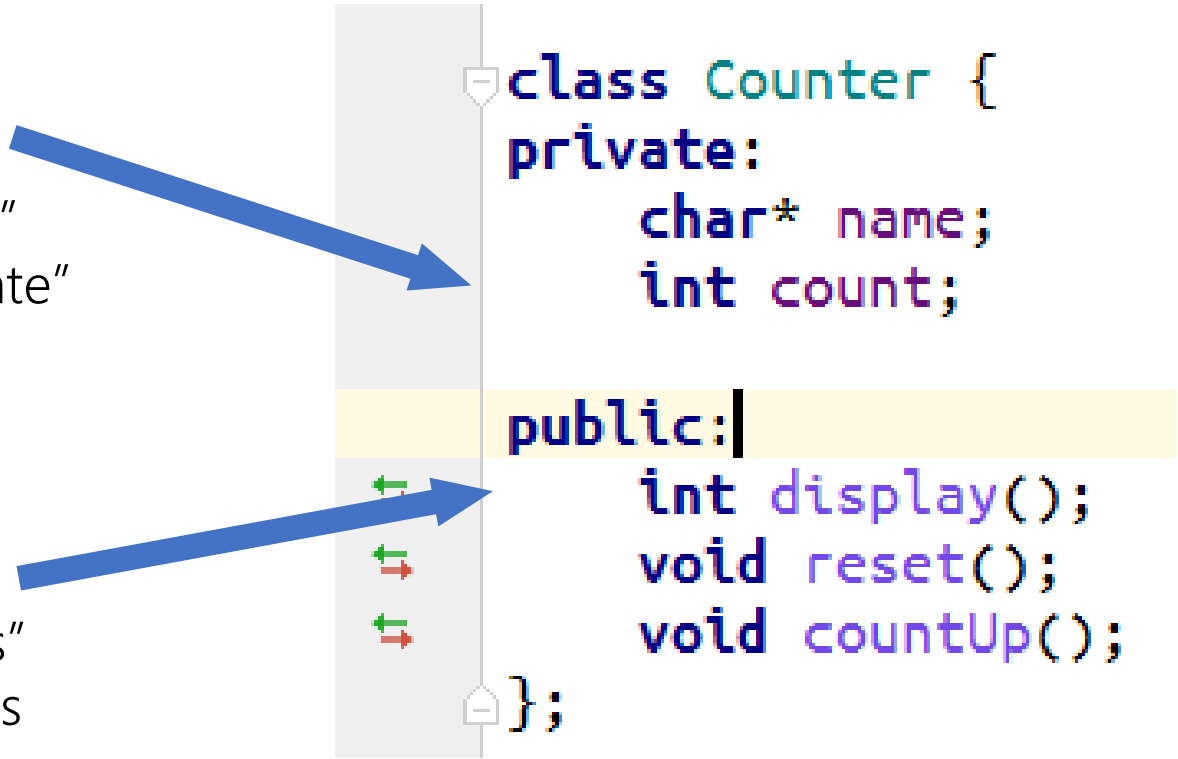
A Class definition

Attribute

"Member variables"
Stores data, or "State"

Method

"Member functions"
Allowed Operations



```
class Counter {
private:
    char* name;
    int count;

public:
    int display();
    void reset();
    void countUp();
};
```

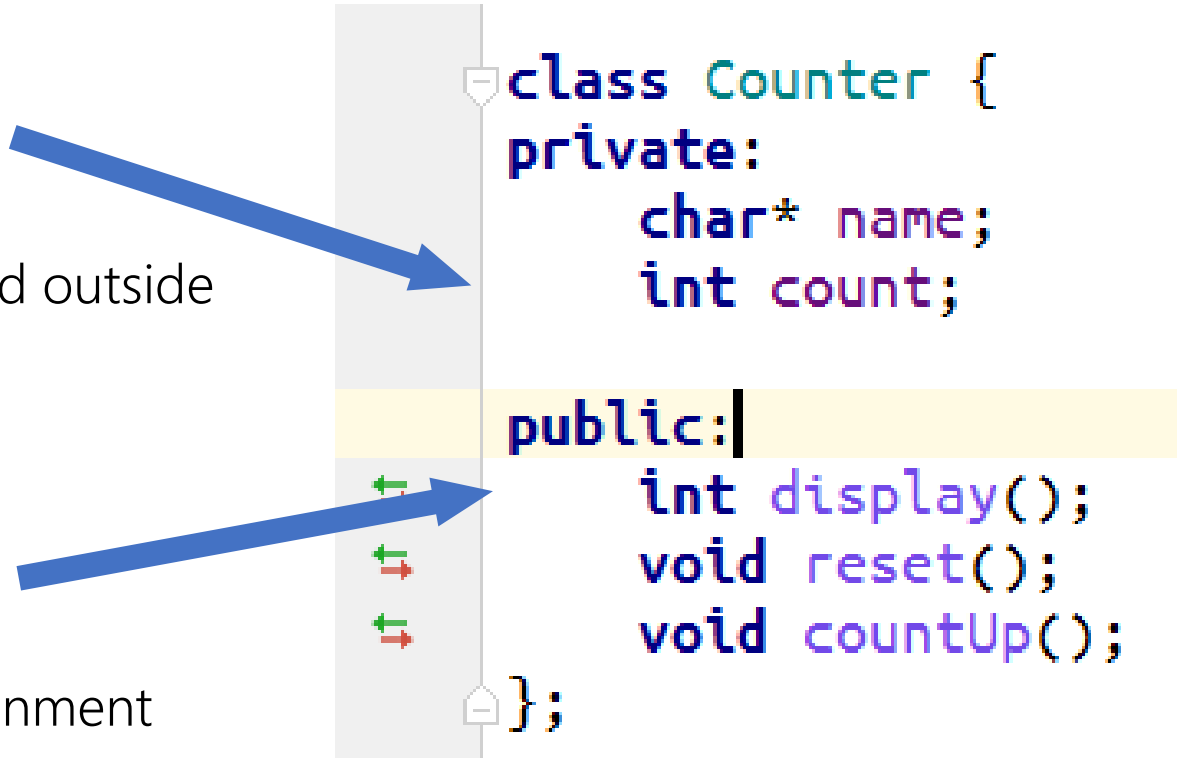
A Visibility of members

private

Internal design.
Cannot be accessed outside

public

Exposed interface
Interact with environment



```
class Counter {
private:
    char* name;
    int count;

public:
    int display();
    void reset();
    void countUp();
};
```

A Comparison of design

C Code

```c
#include <stdio.h>

typedef struct {
    char* name;
    int count;
} Counter;

int  CounterDisplay
        (Counter* counter);
void CounterReset
        (Counter* counter);
void CounterCountUp
        (Counter* counter);
```

C++ Code

```cpp
class Counter {
private:
    char* name;
    int count;

public:

    int display();
    void reset();
    void countUp();
};
```

A Implementing functions

C Code

```c
int CounterDisplay(Counter* counter) {
    printf("%s has %d people inside!\n",
            counter->name,
            counter->count);
    return counter->count;
}

void CounterReset(Counter* counter) {
    printf("%s counter has resetted!\n",
            counter->name);
    counter->count = 0;
}

void CounterCountUp(Counter* counter) {
    counter->count += 1;
    printf("We have %d people in %s\n",
            counter->count,
            counter->name);
}
```

C++ Code

```cpp
int Counter::display() {
    cout << this->name << " has "
        << this->count << " people inside!"
        << endl;
    return this->count;
}

void Counter::reset() {
    this->count = 0;
    std::cout << this->name
        << "counter reset!" << endl;
}

void Counter::countUp() {
    this->count += 1;
    cout << "We have " << this->count
        << " people!" << endl;
}
```

A This pointer can be dropped

C++ Code (before)

```
int Counter::display() {
    cout << this->name << " has "
        << this->count << " people inside!"
        << endl;
    return this->count;
}

void Counter::reset() {
    this->count = 0;
    std::cout << this->name
        << "counter reset!" << endl;
}

void Counter::countUp() {
    this->count += 1;
    cout << "We have " << this->count
        << " people!" << endl;
}
```

C++ Code (now)

```
int Counter::display() {
    cout << name << " has "
        << count << " people inside!"
        << endl;
    return count;
}

void Counter::reset() {
    count = 0;
    std::cout << name
        << " counter reset!" << endl;
}

void Counter::countUp() {
    count += 1;
    cout << "We have " << count
        << " people!" << endl;
}
```

A Play with the new toy

C Code

```c
int main() {
    Counter mainLibCounter =
            {"Main Library", 0};

    CounterReset(&mainLibCounter);
    for (int i = 0; i < 10; ++i) {
        CounterCountUp(&mainLibCounter);
    }
    CounterDisplay(&mainLibCounter);

    CounterReset(&mainLibCounter);
    for (int i = 0; i < 7; ++i) {
        CounterCountUp(&mainLibCounter);
    }
    CounterDisplay(&mainLibCounter);
}
```

C++ Code

Instantiate

```cpp
int main() {
    Counter newLibCounter;

    newLibCounter.reset();
    for (int i = 0; i < 10; ++i) {
        newLibCounter.countUp();
    }
    newLibCounter.display();

    newLibCounter.reset();
    for (int i = 0; i < 7; ++i) {
        newLibCounter.countUp();
    }
    newLibCounter.display();
}
```

A One more problem: Initialization

C Code

```c
int main() {
    Counter mainLibCounter =
            {"Main Library", 0};

    CounterReset(&mainLibCounter);
    for (int i = 0; i < 10; ++i) {
        CounterCountUp(&mainLibCounter);
    }
    CounterDisplay(&mainLibCounter);

    CounterReset(&mainLibCounter);
    for (int i = 0; i < 7; ++i) {
        CounterCountUp(&mainLibCounter);
    }
    CounterDisplay(&mainLibCounter);
}
```

C++ Code

```cpp
int main() {
    Counter newLibCounter;

    newLibCounter.reset();
    for (int i = 0; i < 10; ++i) {
        newLibCounter.countUp();
    }
    newLibCounter.display();

    newLibCounter.reset();
    for (int i = 0; i < 7; ++i) {
        newLibCounter.countUp();
    }
    newLibCounter.display();
}
```

A Solution: Constructor

Constructor:
Function called when class is being instantiated

Must have the same name as the class

Make sure the instance is valid. Provide initial states, etc.

No return type! Allow to take argument.

A constructor without argument called “default constructor”

A Constructor for the counter

Mostly public

No return type

Impose Parameter

Construct object

```
class Counter {
private:
    char* name;
    int count;

public:
    Counter(char*);

    int display();
    void reset();
    void countUp();
};

Counter::Counter(char *str) {
    this->name = str;
    this->count = 0;
}
```

A working version of counter

C Code

Initialization

```c
int main() {
    Counter mainLibCounter =
            {"Main Library", 0};

    CounterReset(&mainLibCounter);
    for (int i = 0; i < 10; ++i) {
        CounterCountUp(&mainLibCounter);
    }
    CounterDisplay(&mainLibCounter);

    CounterReset(&mainLibCounter);
    for (int i = 0; i < 7; ++i) {
        CounterCountUp(&mainLibCounter);
    }
    CounterDisplay(&mainLibCounter);
}
```

C++ Code

Invoke constructor

```cpp
int main() {
    Counter newLibCounter(
            (char*)"New library");

    newLibCounter.reset();
    for (int i = 0; i < 10; ++i) {
        newLibCounter.countUp();
    }
    newLibCounter.display();

    newLibCounter.reset();
    for (int i = 0; i < 7; ++i) {
        newLibCounter.countUp();
    }
    newLibCounter.display();
}
```

A

Everything can be an object

Classes are special datatypes

Instance of class is essentially a variable of special datatype

Datatype are special classes

You can use class syntax to define int, double, etc.

A

Another problem with C

Consider a function max()

max() takes in 2 arguments, and return the max of them

What should be the type of the argument?

Both int? Both Float? One Float, one int?

A Another problem with C

Logically speaking datatype is irrelevant.

But due to datatype we have to write different functions!

C Code

```
int maxIntInt   (int a, int b);
int maxIntFoat  (int a, float b);
int maxFloatInt (float a, int b);
```

A Function overloading

(Function) Overload:
Functions have same return type, but with different input datatype, sharing a name.

Different overloads are still different functions!

You need to implement individually

Compiler choose overload by calling argument.

The overload with "most similar" function signature will be selected.

A

Function overloading

C++ Code

```
#define MAX(a, b) ((a) > (b) ? (a) : (b))

int max (int a, int b);
int max (int a, float b);
int max (float a, int b);

int max (int a, int b) {return MAX(a,b);}
int max (int a, float b) {return MAX(a,b);}
int max (float1 a, int b) {return MAX(a,b);}
```

A

Well, what else can be overloaded?

Operator “+” works on primitive datatypes.

It also make sense if used on Complex numbers (structure)

It makes sense to overload Operators!

Overloaded operators do not require same return type.

Int + int => int
float + float => float
Complex + Complex => Complex
std::string + std::string => std::string (concatenation)

A Overloaded Operators

Operator “>>”, “<<”

Overloaded for input/output streams

Operator “+”, “[ ]”

Overloaded for std::string

```
std::string str("Overload");
cout << str[1] << str[2] << endl; // Outputs "ve"
std::string str2(" + Operator");
cout << (str + str2) << endl; //Output "Overload + Operator"
```

A The C++ std::string class

```
using std::cout;
using std::endl;

int main() {
    std::string str1("New String1"); //Constructor
    std::string str2 = "String2";    //Assignment Construction

    cout << str1 + "&" + str2 << endl;  //"New String1&String2"

    cout << str1.substr(4, 6) << endl; //"String1"

    cout << "Length of str2: " << str2.length() << endl; //Get length

    cout << (str1 == str2) << endl;     //Overloaded "=="
    cout << (str1 == "Equal?") << endl; //Makes comparing easier

    char character = str1[1]; //Acess individual character

    const char *c_string = str1.c_str();
// Get a c_style string, you cannot modify it!!
}
```

A

File IO (Points of interests)

How to open/close a file?

Use ifstream/ofstream. #include <fstream>

How to get int/double/char input/output?

Use “<<“ and “>>”

How to input an entire line?

Use file.getline();

A

File IO (Example from mid2 rv.)

C Code

```c
int getNumberFromFile(char* file,
                      int* dest,
                      int size) {
    FILE* fp = fopen(file, "r");
    if (fp == NULL) {
        puts("Error");
        return -1;
    }
    int num = 0; int count = 0;
    while (fscanf(fp, "%d", &num) != EOF)
        if (count == size){
            fclose(fp);
            return -2;
        }
        dest[count] = num;
        count++;
    }
    fclose(fp);
    return count;
}
```

C++ Code

```cpp
int getNumberFromFile(char* file,
                      int* dest,
                      int size) {
    fstream inFile(file); // Opens the file
    if (!inFile.is_open()) { // Check file open
        cout << file << " not exist" << endl
        inFile.close();
        return -1;
    }
    int i = 0;
    while (inFile.good()) {
        //if last read is successfull
        if (i == size) break;
        inFile >> dest[i];
        if (inFile.good()) i++;
    }
    inFile.close();
    return i;
}
```

A

File Output

C++ Code

```
int writeNumberToFile(char* file, int* number, int size) {
    ofstream outFile(file);
    if (!outFile.is_open()) {
        cout << file << " can't open." << endl;
        outFile.close();
        return -1;
    }
    for (int i = 0; i < size; ++i) {
        outFile << "Number " << i << " number is " << number[i] << endl;
    }
    return 0;
}
```

A

Common tasks

- **Read formatted input, given format (e2_r)**
- **Solution: use “>>” according to format**
- **Read unknown number of inputs.**
- **Key: How to check if read has ended?**
- **Write to file given a format**
- **Key: use “<<“ according to format**
- **Pay attention to function prototype!**
- **How data is passed outside the function**

A Object Lifespan

Object: Instances, variables

Object should be understand in a broader sense

Lifespan: From creation to destruction

From “Constructed in memory” to “Erased from memory”
🡐 From “Memory Allocated” to “Memory Freed”

A Scope versus Lifespan

Local object

- Exist when it is constructed/initialized
- Vanishes when the function exits

Wrong Code!

```
1   #include <stdlib.h>
2   #include <time.h>
3
4   int *getRandomVector(int n) {
5       int a[100] = {0};
6       srand(time(NULL));
7       for (int i = 0; i < n; ++i) {
8           a[i] = rand();
9       }
10      return a;
11  }
12
13  int main() {
14      int* num = getRandomVector(10);
15      return 0;
16  }
```

Local

<= Return a "dead" object

A Scope versus Lifespan

Global object

Not Allowed!

- Exist from the moment the program begins
- Vanishes with the end of the program

```
#include <iostream>

using namespace std;

string globalString = "Global";

int main() {

    string another = "Local";

    cout << globalString;
}
```

Fun fact:
cout is a global object!

A Operator "new" and dynamic created objects

Operator new

- On the shared memory, allocate enough memory for the class
- Call the constructor to construct the object.
- Return a pointer to that object type
- Require delete/delete[] to call de-constructor and free (Life span!).

```
using namespace std;
int main() {
    string *strPtr =
            new string("Hello");

    int* a = new int(4); //

    double* b = new double[10];

    delete strPtr;
    delete a;
    delete[] b;
}
```

A

Notes on operator new

Similar to malloc():

The constructed object (allocated memory) life span never ends, until it is explicitly destroyed!

Different to malloc():

“new” invokes constructor, malloc() just allocate memory
“delete” invokes de-constructor, free() just free memory
“malloc()” returns void*, “new” returns the “casted” pointer

A

Notes on operator new

Similar to malloc():

The constructed object (allocated memory) life span never ends, until it is explicitly destroyed!

Different to malloc():

“new” invokes constructor, malloc() just allocate memory
“delete” invokes de-constructor, free() just free memory
“malloc()” returns void*, “new” returns the “casted” pointer

A

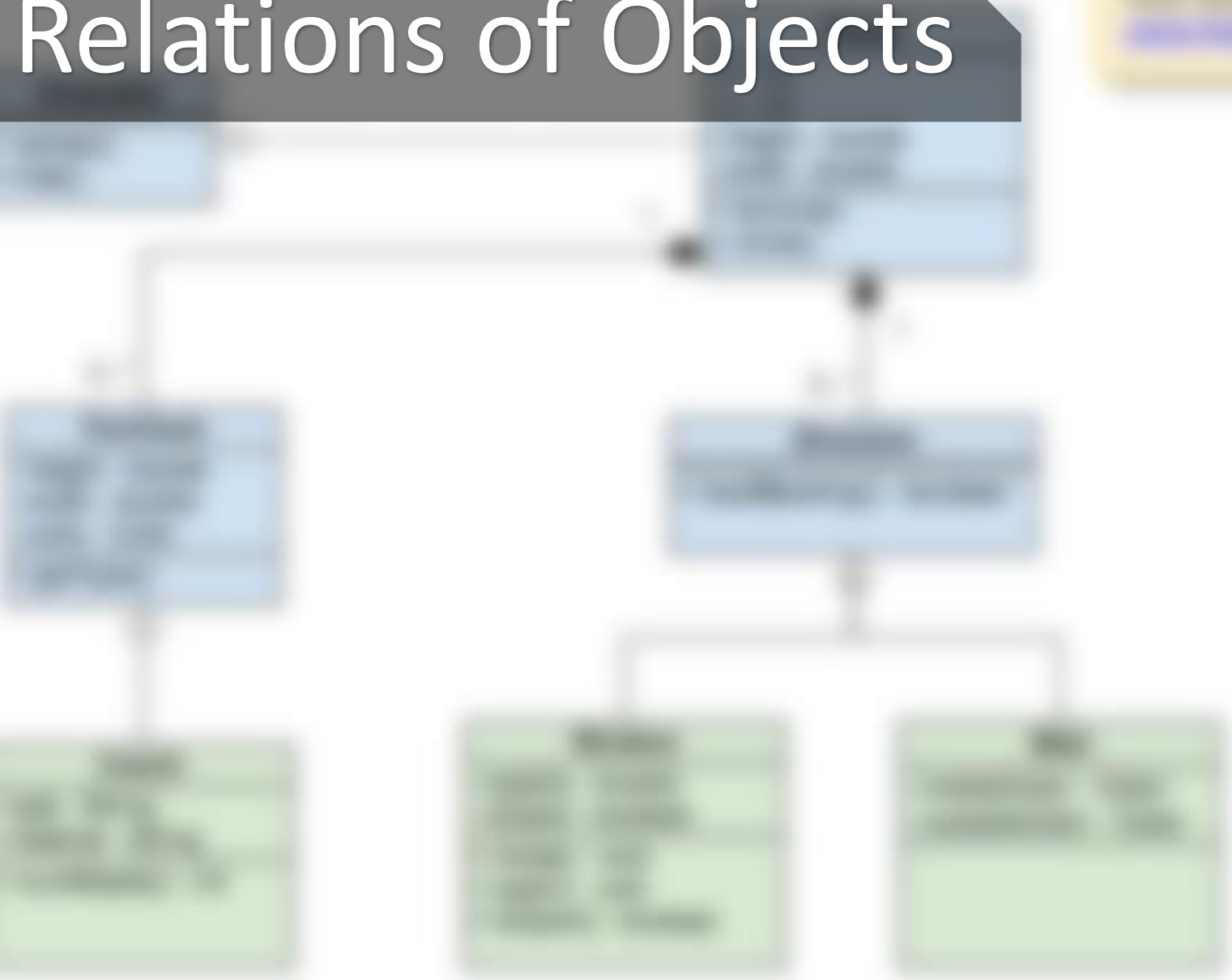
Example on the life span and "new"

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

class Obj{
    const char* name;
public:
    Obj(const char* str): name(str) {
        cout << name << " is constructed!" << endl;
    };
    ~Obj() {
        cout << name << " is destructed" << endl;
    }
};

int main() {
    cout << "Program begin" <<endl;
    Obj objLocal("Local object1");
    Obj objLocal2("Local object2");
    Obj* objPtr = new Obj("Dynamic 1");
    Obj* leakPtr = new Obj("ObjLeak");
    cout << "This one will leak" << endl;
    delete objPtr;
    cout << "Function ends here" << endl;
}
```

Output

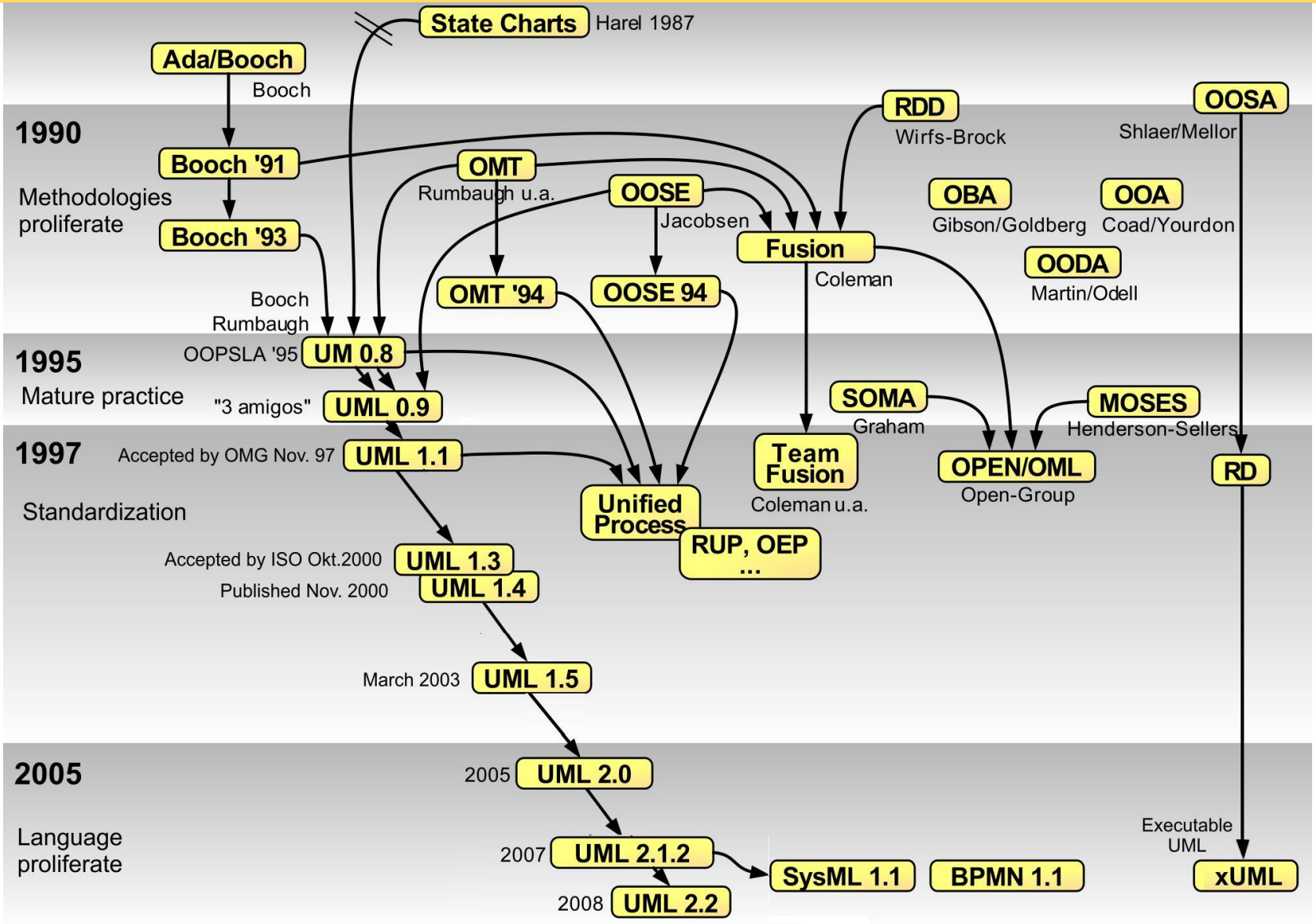
```
Program begin
Local object1 is constructed!
Local object2 is constructed!
Dynamic 1 is constructed!
ObjLeak is constructed!
This one will leak
Dynamic 1 is destructed
Function ends here
Local object2 is destructed
Local object1 is destructed
```

Relations of Objects



History of modeling notations

B



B

There are 3 characteristics of OOP

Class and encapsulation

Ability to “capture” data with allowed operations.

Inheritance

Ability to factor out similarity between classes

Polymorphism

Ability to allow objects behave differently in runtime

Relationships

Two main types:

- "Is a":

```
1 class Submarine : public Boat {
2   ...
3 }
```

- "Has a":

```
1 class Company {
2   public:
3     Boat *boats;
4     Trucks *trucks;
5 };
```

B

They are about relations

Inheritance

This class is a kind of A, but do more then A.

Polymorphism

This class do similar things as A, but in a different way

B

Problem: A new type of counter

An SJTU library counter V2

- Support all functions of original
- Plus be able to down count when someone leaves the building

B An analysis on the relation

CounterV2 is a special type of Counter

Because wherever you need Counter, CounterV2 will work

CounterV2 extends the Counter

CounterV2 support more functions then Counter

B

Translate the relation to program

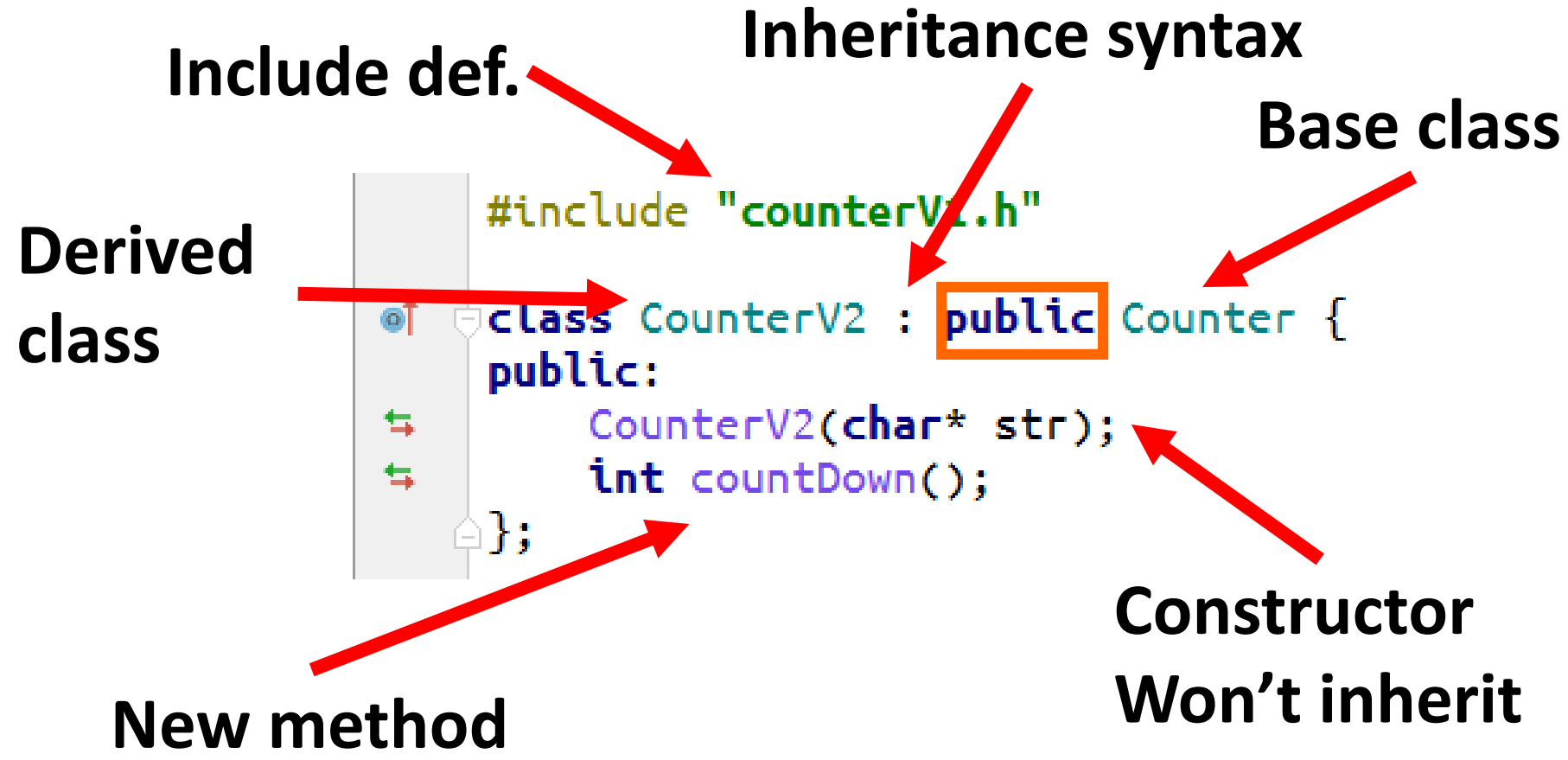
CounterV2 inherits everything from Counter

CounterV2 is in this way build upon the original Counter

CounterV2 has its own new methods

By defining new method CounterV2 can do more.

B Inheritance: the syntax



B Implementation, visibility?

```
#include "counterV2.h"

CounterV2::CounterV2(char *str) :
        Counter(str) {

}

int CounterV2::countDown() {
    count -= 1;
    cout << "Leave 1 person, remains "
        << count << " people!" << endl;
}
```

private 'Counter::count' is inaccessible

Listing things to initialize

First construct base class

B

Visibility, revisited

What is the visibility of member count?

Who can access count?

Who should be able to access count?
Who shouldn’t be?

B

Visibility, revisited

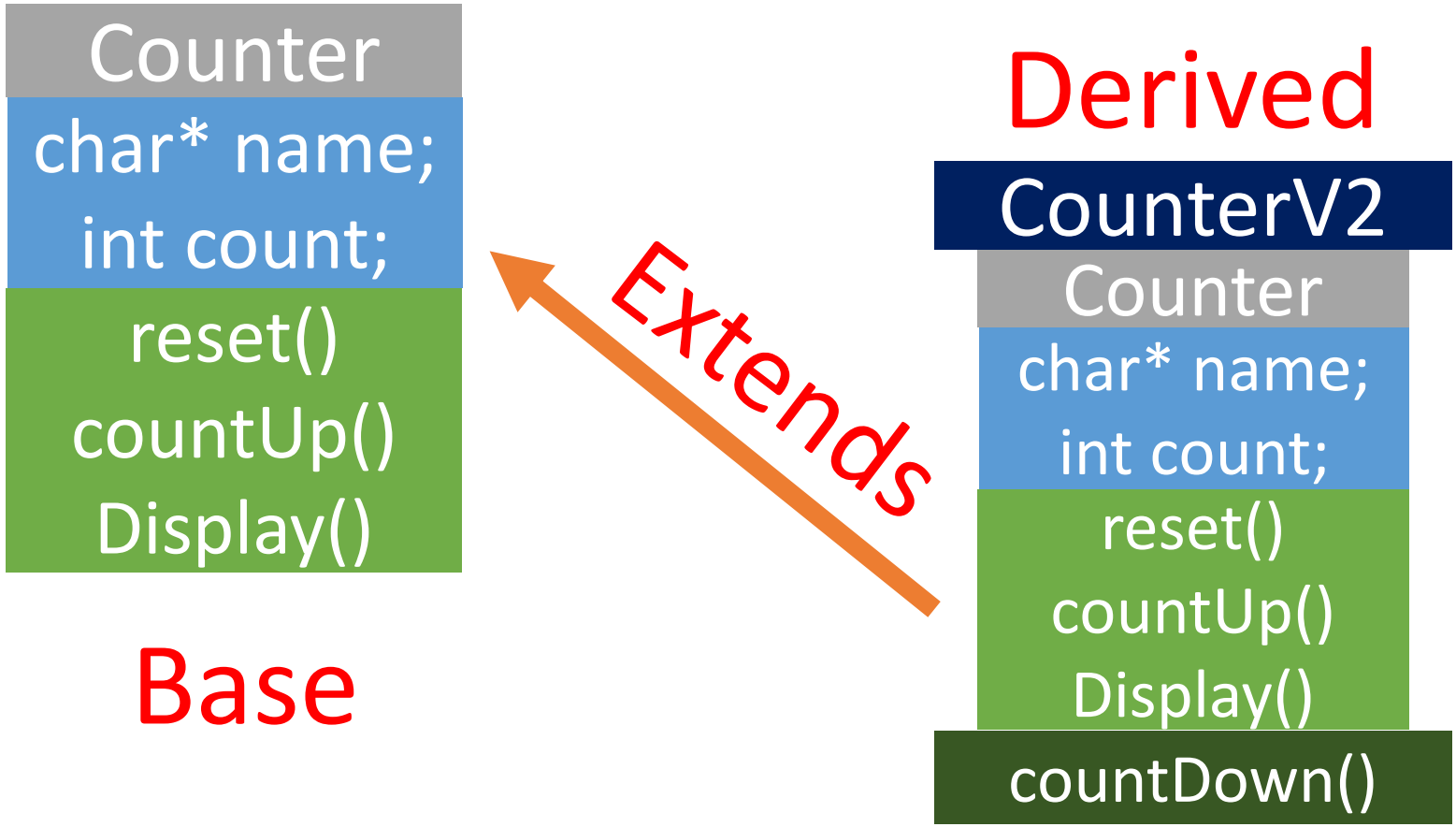
	Public	Private	Protected
Class	Yes	Yes	Yes
Derived	Yes	No	Yes
Others	Yes	No	No

B Derived class substitutes base class

```
int main() {
    CounterV2 newLibCounter(
            (char*)"New library");

    newLibCounter.reset();
    for (int i = 0; i < 10; ++i) {
        newLibCounter.countUp();
    }
    newLibCounter.display();
    for (int j = 0; j < 5; ++j) {
        newLibCounter.countDown();
    }
    newLibCounter.display();

    newLibCounter.reset();
    for (int i = 0; i < 7; ++i) {
        newLibCounter.countUp();
    }
    newLibCounter.display();
}
```

```
New library counter reset!
We have 1 people!
We have 2 people!
We have 3 people!
We have 4 people!
We have 5 people!
We have 6 people!
New library has 6 people inside!
Leave 1 person, remains 5 people!
Leave 1 person, remains 4 people!
Leave 1 person, remains 3 people!
New library has 3 people inside!
New library counter reset!
We have 1 people!
We have 2 people!
New library has 2 people inside!
```

B Visualize Inheritance (Important!)



A

Problem: Another new type of counter

SJTU library CounterPro

- Support all functions of Counter
- And with one extra switch, when turn on, the counter counts. When turned off, the counter stops counting.

A

Analyzing the relation

CounterPro is a another type of Counter

Because wherever you need Counter, CounterPro will work

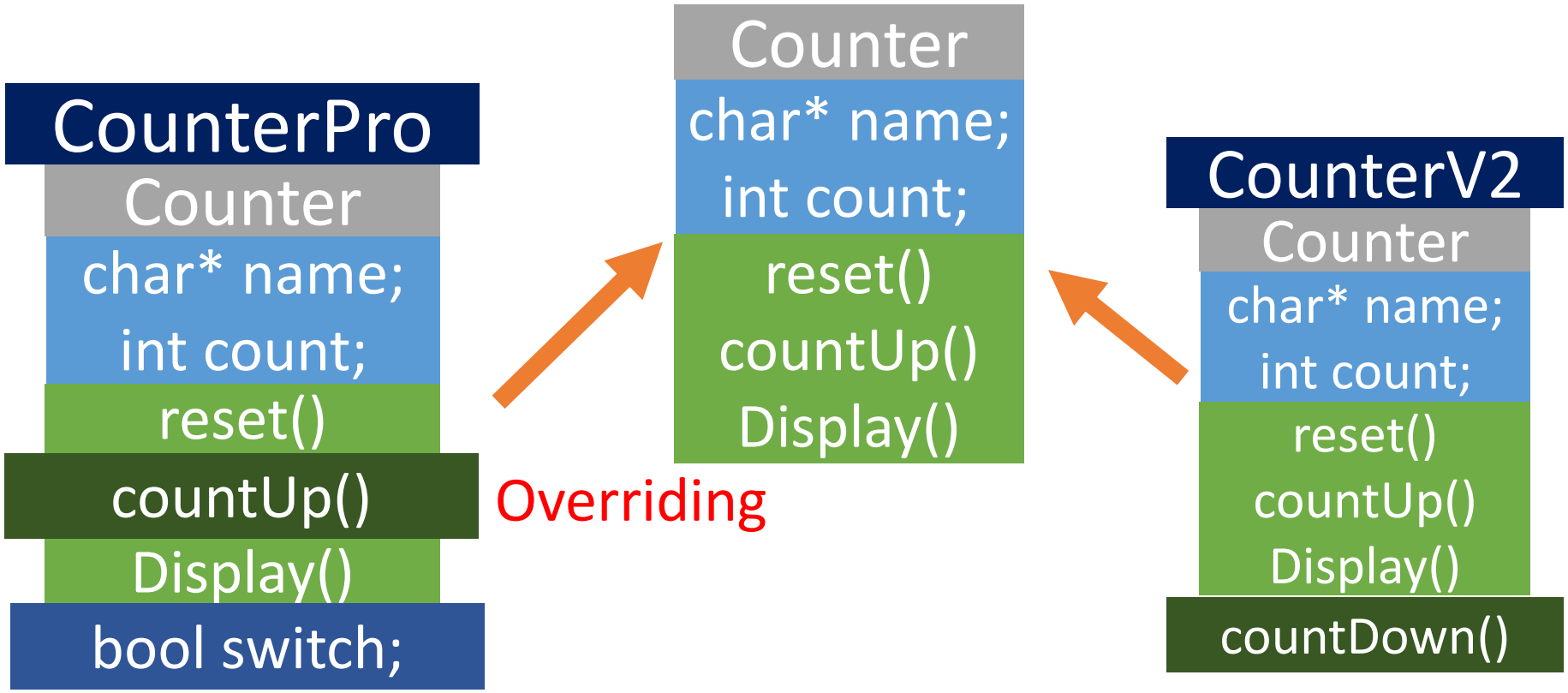
Add an attribute, switch

This attribute should be public.

We need to rewrite countUp()

The CounterPro still counts, but do this in a different way.

B Overriding the original function



A Naïve approach (Will not work)

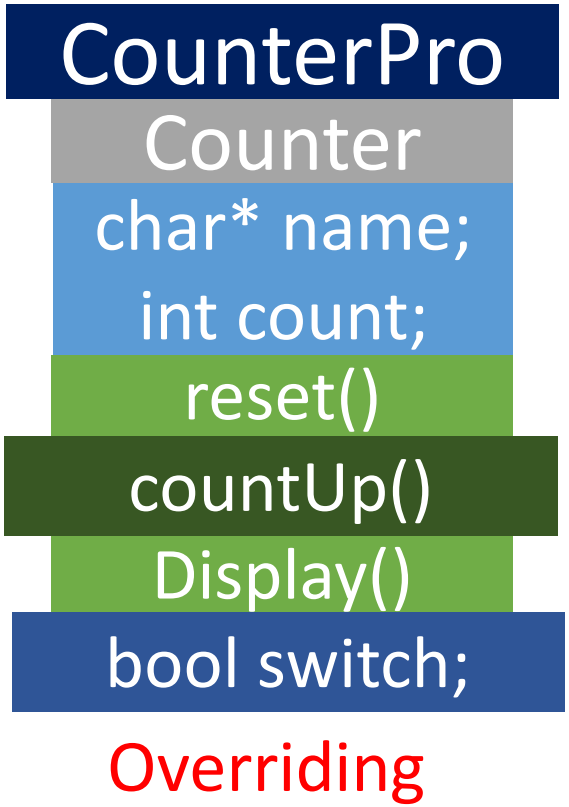
C++ Code

```
#include "counterV1.h"

class CounterPro : Counter {
public:
    bool isPowerOn;
public:
    CounterPro(char* str,
               bool power);
    void countUp();
};
```

```
#include "counterPro.h"

#include <iostream>
using namespace std;

CounterPro::CounterPro(char *str,
                       bool power) :
        Counter(str) {
    isPowerOn = power;
}

void CounterPro::countUp() {
    if (isPowerOn) {
        count += 1;
        cout << "We have " << count
        << " people!" << endl;
    } else {
        cout << "No power!" << endl;
    }
}
```

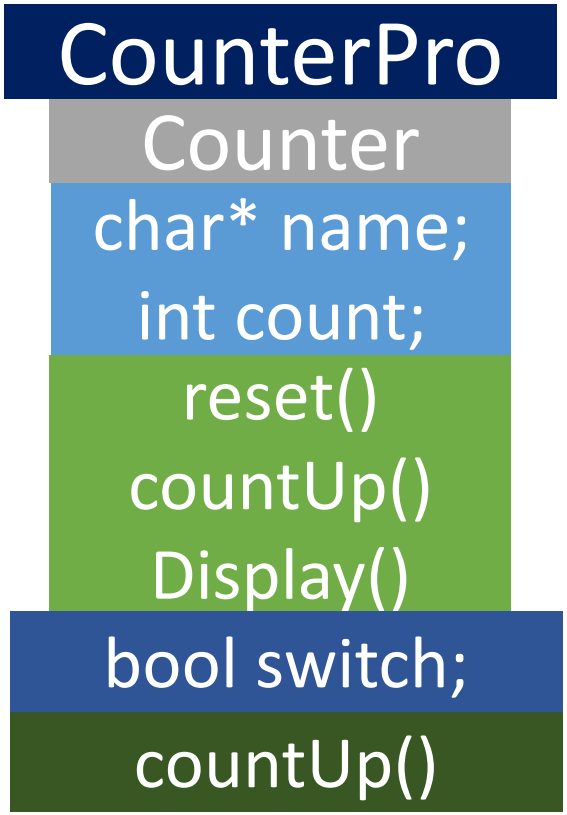
A Name Hiding

What you want



Overriding

What you write



Name Hiding

A

Fix: using the virtual keyword

virtual, a method modifier
Signaling that the method could be override in a derived class?

Which method is being overridden?

Which class is overriding the method?

A Fixing the situation

The base class

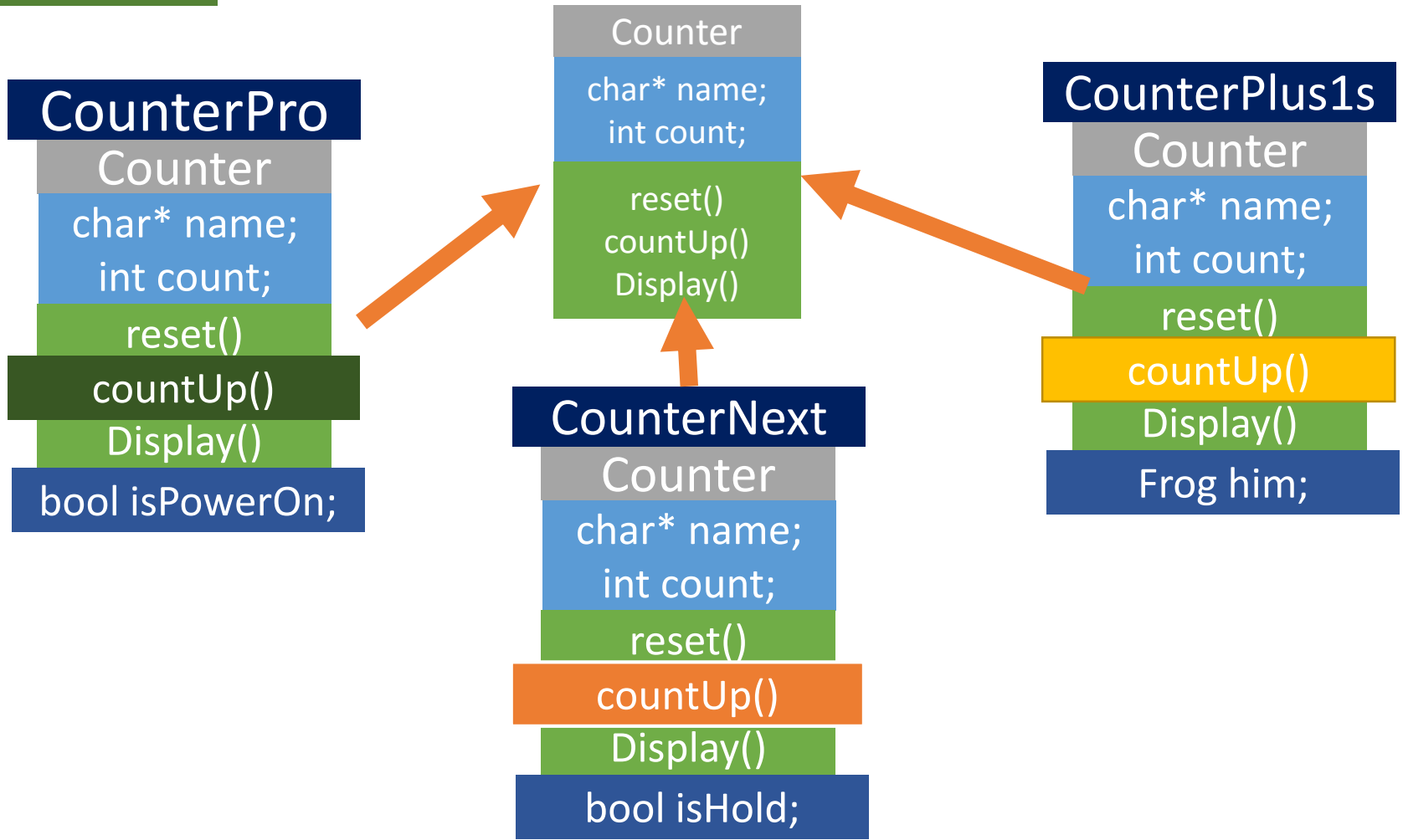
```
class Counter {
protected:
    char* name;
    int count;

public:
    Counter(char* name);
    int display();
    void reset();
    virtual void countUp();
};
```

The derived class

```
#include "counterV1.h"

class CounterPro : Counter {
public:
    bool isPowerOn;
public:
    CounterPro(char* str,
               bool power);
    void countUp();
};
```

B

Visualizing Polymorphism



B Polymorphism

- **Objects inherits a common base, exibiting different behavior for the same method**

- **Counter Counts**
- **CounterPro Counts**

- **=> In a Different Way**

B

When inherit?

Objects sharing a common “trait”

All cars are cars. They should have an engine, a door ….

A Tank is a special Car => Tank inherit Car

The “is a” inheritance

A Jet is a improved Plane => Jet inherit Plane

Jet and Plane both flies, but in fly in an improved way

B

Why do we need inheritance.

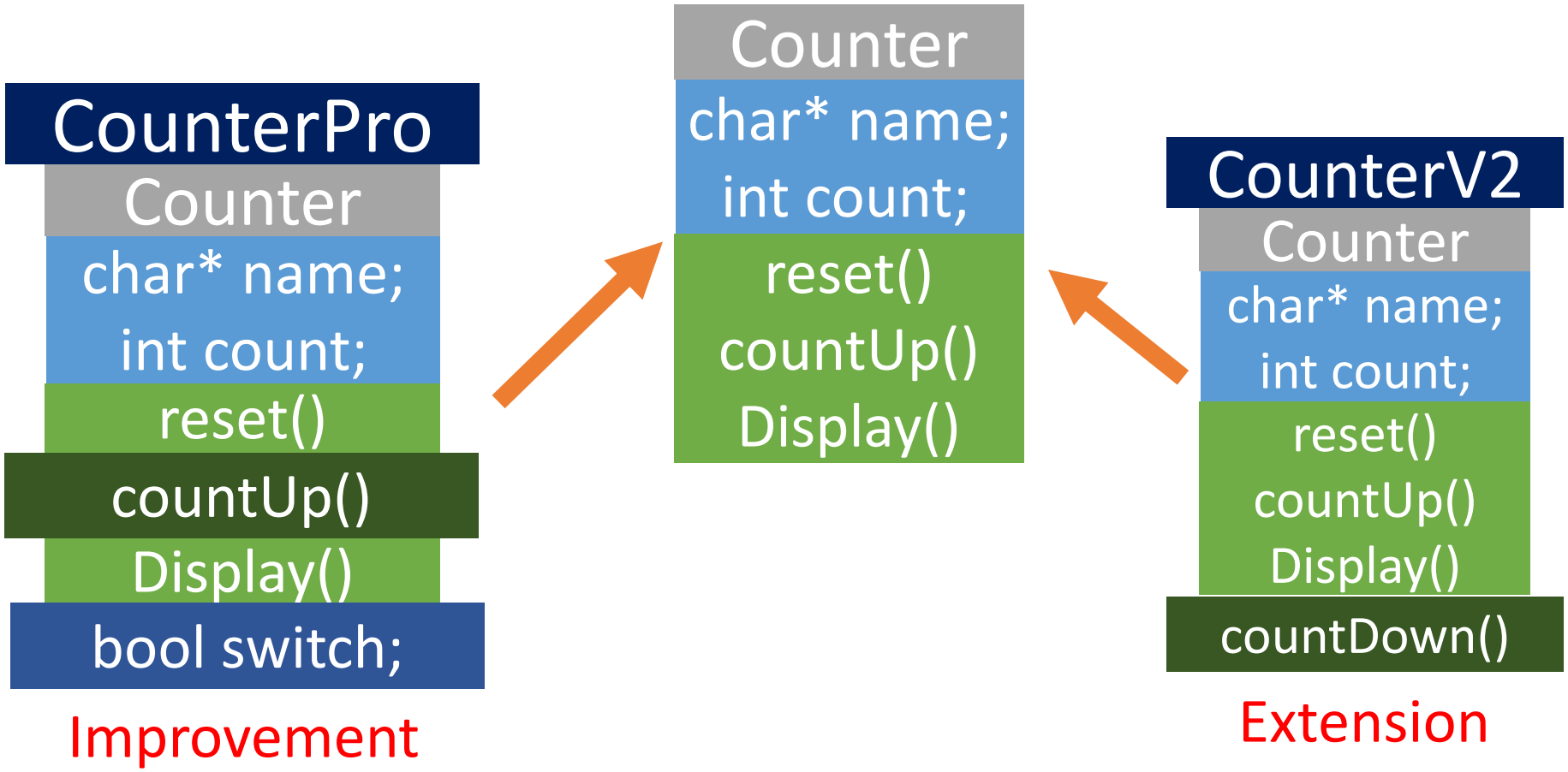
We need to add to the original class

A derived class is an base class with more functions!

We need to improve the original class

Rewrite some base class method, you need virtual

B Visualize class relationship



Z

C++ is about “Design” s

Pure virtual base class

How would you model abstract ideas, such as “an animal”

Aspect oriented programming

What if of all a sudden I need to log all class activities

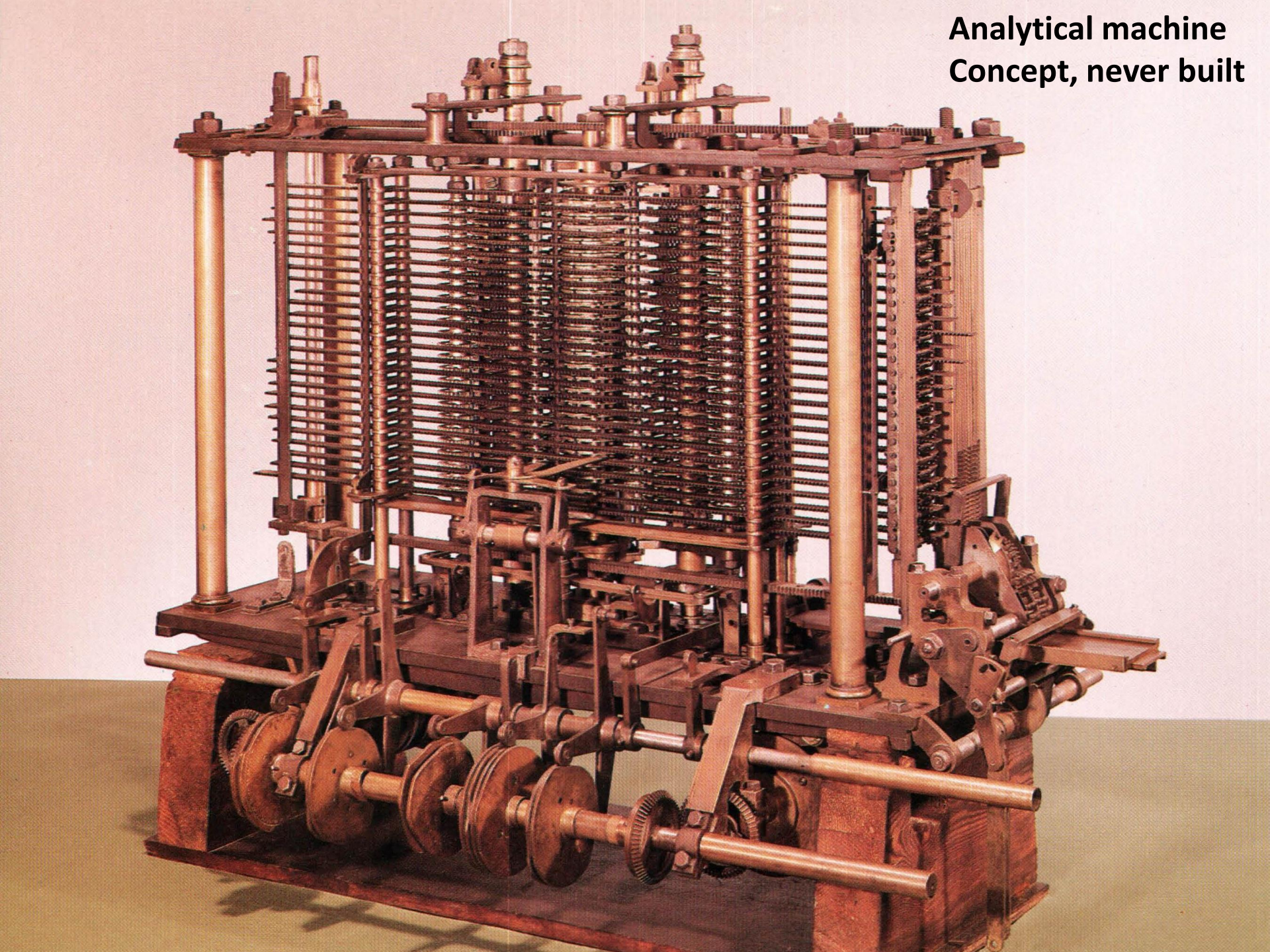
Y-Combinator

Well, programming is about combining objects together.

Ap.

Appendix: Inheritance visibility

	[I]Public	[I]Private	[I]protected
Public	Public	Private	Protected
Private	Private	No access	No access
Protected	Protected	Private	Protected


Analytical machine
Concept, never built