Copyright Warning

ViSLAB
High-Performance Visualization, Communication, Computing

Object Oriented Design

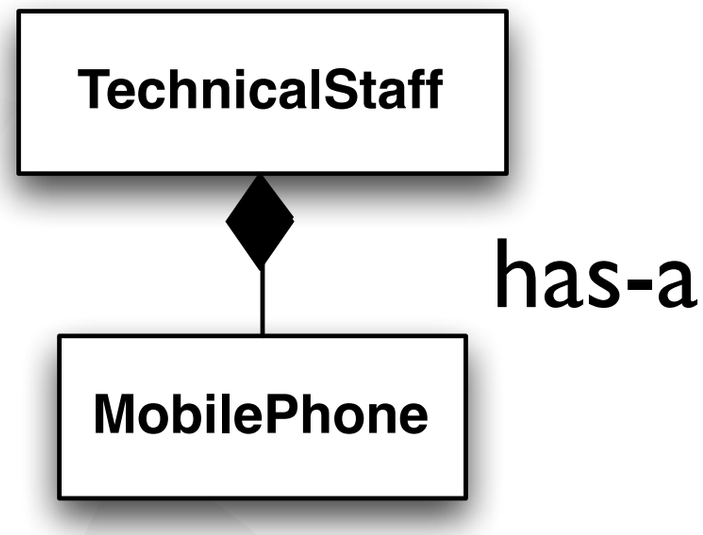
Week 7-2

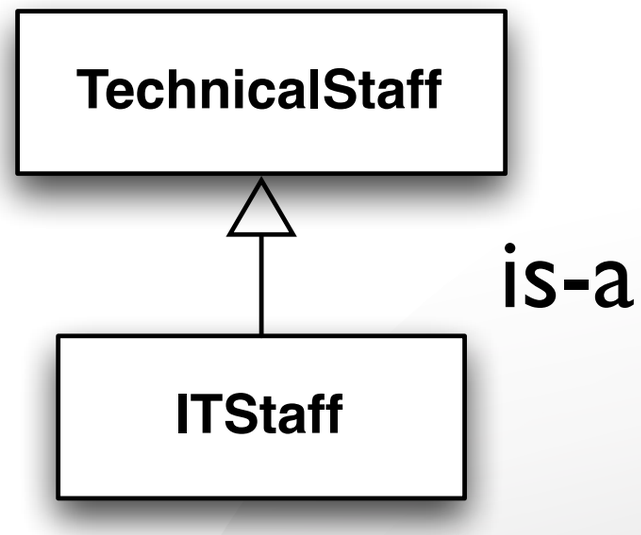
- Composition
- Exception
- Copy Constructor
- Friend Function

Composition

- member object
  - object as a member variable

```
class TechnicalStaff {
    MobilePhone mobile;  // Member Object
    ...
};
```





Composition

- know the timing of creation.

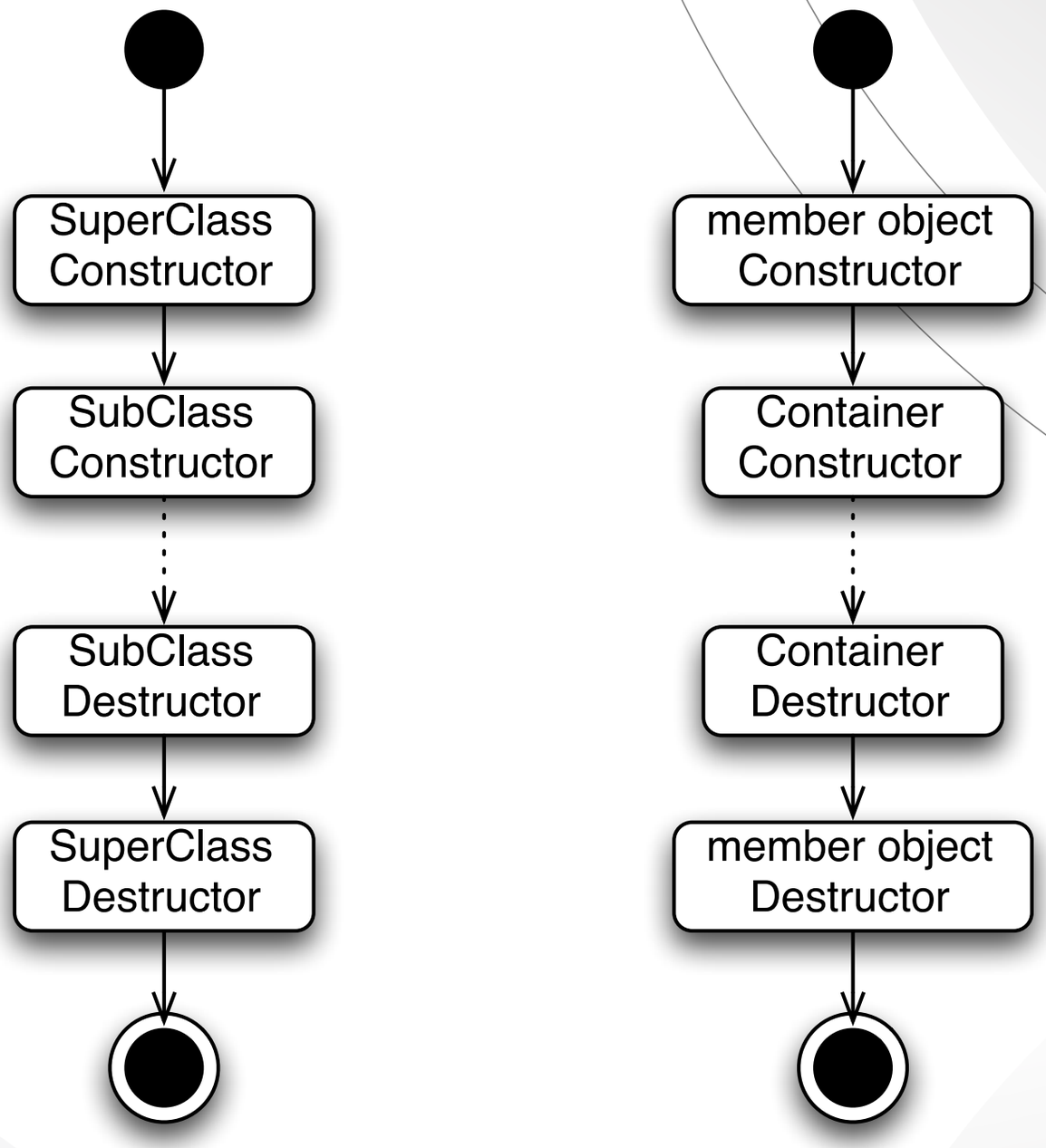
```
class MobilePhone {
public:
    char number[15]; // phone number
    bool smart;      // whether smart-phone or not
};

class TechnicalStaff{
public:
    int staffNo;         // staff id
    char name[64];       // name
    MobilePhone mobile; // mobile phone
    void displayData(); // display details
};

TechnicalStaff mystaff;

TechnicalStaff *mystaff = new TechnicalStaff();
```

Composition



Composition

- know the timing of creation.

```
class MobilePhone {
public:
    char number[15]; // phone number
    bool smart;      // whether smart-phone or not
    MobilePhone():smart(false) {
        strcpy(number, "no number");
    };
    MobilePhone(char *num, bool s):smart(s) {
        strcpy(number, b);
    };
};

class TechnicalStaff{
public:
    int staffNo;         // staff id
    char name[64];       // name
    MobilePhone mobile; // mobile phone
    void displayData(); // display details
    TechnicalStaff(int id, char *name, char *number, bool s):mobile(number, s) {
        staffNo = id;
        strcpy(this->name, name);
    };
};
```

Exception

- What do you normally do for error handling?
  - Java
  - C
  - etc.

Error handling (example)

```
   char *buff;
buff = (char *) malloc(xxxx);
if (buff == NULL) {
    fprintf(stderr, "%s, %d: failed to allocate memory\n", __FILE__, __LINE__);
}
```

```
char *buff;
if ((buff = (char *)malloc(xxxx)) == NULL) {
    fprintf(stderr, "%s, %d: failed to allocate memory\n", __FILE__, __LINE__);
}
```

- __FILE__, __LINE__ : preprocessor's macro

```
FILE *fp;
if ((fp = fopen(argv[1], “r”)) == NULL) {
    fprintf(stderr, "%s, %d: Could not open %s\n", __FILE__, __LINE__, argv[1]);
}
```

Error handling

- In C
  - check the return value of a function
  - use “if” statement to check it

Error handling (C++ ver.)

```
    char *buff;
if ((buff = new char[xxxx]) == NULL) {
     fprintf(stderr, "%s, %d: failed to allocate memory\n", __FILE__, __LINE__);
}
```

- ?

Error handling (C++ ver.)

```
try {
    char *buff;
    buff = new char[xxxx];
} catch (std::bad_alloc &ex) {
    fprintf(stderr, "%s, %d: failed to allocate memory\n", __FILE__, __LINE__);
}
```

- in OOP lang, “error”/”exception” is thrown!
- you need to “catch” it after you “try” the code.

- in C++, use new (rather than malloc)....const/destructor
- also to catch memory allocation “error”

Throwing Error

```
try {
    char *buff;
    buff = new char[xxxx];
} catch (std::bad_alloc &ex) {
    fprintf(stderr, "%s, %d: failed to allocate memory\n", __FILE__, __LINE__);
}
```

- Error/Exception .... object
- Control move to “catch” block with corresponding type
- can have multiple “catch” block

Throwing Error

```
try {
    char *buff;
    buff = new char[xxxx];
} catch (std::bad_alloc &ex) {
    fprintf(stderr, "%s, %d: failed to allocate memory\n", __FILE__, __LINE__);
}
```

- looks like same as checking the return value....
- looks like more trouble than “if (...)”

- “delegate error/exception handling beyond functions”

Handling Error/Exception

- Extreme case:
  - Catch everything in main() !

```
int main() {
    try {
        ...
    } catch (...) {
        ...
    }
    return 0;
}
```

```
   #include <stdio.h>
#include <new>

void data_alloc() {
   char *data = new char[40000000000000000];
}

int main() {
   try {
      data_alloc();
   } catch (std::bad_alloc &ex) {
      fprintf(stderr, "%s, %d : failed to allocate memory\n", __FILE__, __LINE__);
   }
   return 0;
}
```

catch it where appropriate

- You should have “try-catch” where it’s appropriate.
  - especially, if you program might be able to recover from an error/exception.

- Your function’s return value can have more meaningful value. (instead of error code)

```
    #include <stdio.h>
#include <new>

int throw_bad_alloc() {
    throw std::bad_alloc();
    return 0;
}

void data_alloc() {
    char *data = new char[40000000000000000];
}

int main() {
    _set_new_handler(throw_bad_alloc);

    try {
        data_alloc();
    } catch (std::bad_alloc &ex) {
        fprintf(stderr, "%s, %d : failed to allocate memory\n", __FILE__, __LINE__);
    }
    return 0;
}
```

Copy Constructor

```
#include <iostream>
#include <string>

using namespace std;

class Staff {
public:
    int staffID;
    string name;
    Staff();
    ~Staff();
};

Staff::Staff() : staffID(0), name("un-named") {
    cout << "Staff::Staff() is called" << endl;
}

Staff::~Staff() {
    cout << "Staff::~Staff() is called" << endl;
}

void DisplayInfo(Staff obj) {
    cout << "Staff ID : " << obj.staffID << endl;
    cout << "Name : " << obj.name << endl;
}
```

```
int main() {
    Staff ben;

    ben.staffID = 201101;
    ben.name = "Ben";
    cout << "calling DisplayInfo()" << endl;
    DisplayInfo(ben);
    cout << "back from DisplayInfo()" << endl;

    return 0;
}
```

```
Staff::Staff() is called
calling DisplayInfo()
Staff ID : 201101
Name : Ben
Staff::~Staff() is called
back from DisplayInfo()
Staff::~Staff() is called
```

- When a function is called...pass by value.
- object is copied.
- Constructor will not be called!
  - Language Spec.

```
void DisplayInfo(Staff obj) {
   cout << "Staff ID : " << obj.staffID << endl;
   cout << "Name : " << obj.name << endl;
}

   DisplayInfo(ben);
```

- Destructor will be called.

```cpp
#include <iostream>
#include <string>

using namespace std;

class Staff {
public:
    int staffID;
    string name;
    char *note;
    Staff();
    ~Staff();
};

Staff::Staff() : staffID(0), name("un-named") {
    note = new char[256];
    cout << "Staff::Staff() is called" << endl;
}

Staff::~Staff() {
    delete []note;
    cout << "Staff::~Staff() is called" << endl;
}

void DisplayInfo(Staff obj) {
    cout << "Staff ID : " << obj.staffID << endl;
    cout << "Name : " << obj.name << endl;
    cout << "Note : " << obj.note << endl;
}
```

```cpp
int main() {
    Staff ben;

    ben.staffID = 201101;
    ben.name = "Ben";
    strcpy(ben.note, "This is a note for Ben.");
    cout << "calling DisplayInfo()" << endl;
    DisplayInfo(ben);
    cout << "back from DisplayInfo()" << endl;

    cout << "Note2 : " << ben.note << endl;

    return 0;
}
```

```
Staff::Staff() is called
calling DisplayInfo()
Staff ID : 201101
Name : Ben
Note : This is a note for Ben.
Staff::~Staff() is called
back from DisplayInfo()
Note2 :
```

```cpp
#include <iostream>
#include <string>

using namespace std;

class Staff {
public:
    int staffID;
    string name;
    char *note;
    Staff();
    ~Staff();
};

Staff::Staff() : staffID(0), name("un-named") {
    note = new char[256];
    cout << "Staff::Staff() is called" << endl;
}

Staff::~Staff() {
    delete []note;
    cout << "Staff::~Staff() is called" << endl;
}

void DisplayInfo(Staff *obj) {
    cout << "Staff ID : " << obj->staffID << endl;
    cout << "Name : " << obj->name << endl;
    cout << "Note : " << obj->note << endl;
}
```

```cpp
int main() {
    Staff ben;

    ben.staffID = 201101;
    ben.name = "Ben";
    strcpy(ben.note, "This is a note for Ben.");
    cout << "calling DisplayInfo()" << endl;
    DisplayInfo(&ben);
    cout << "back from DisplayInfo()" << endl;

    cout << "Note2 : " << ben.note << endl;

    return 0;
}
```

```
Staff::Staff() is called
calling DisplayInfo()
Staff ID : 201101
Name : Ben
Note : This is a note for Ben.
back from DisplayInfo()
Note2 : This is a note for Ben.
Staff::~Staff() is called
```

Copy Constructor

- special constructor
- called when a function takes an instance.
- ClassName(const ClassName &arg_name)

```cpp
#include <iostream>
#include <string>

using namespace std;

class Staff {
public:
    int staffID;
    string name;
    char *note;
    Staff();
    Staff(const Staff &obj);
    ~Staff();
};

Staff::Staff() : staffID(0), name("un-named") {
    note = new char[256];
    cout << "Staff::Staff() is called" << endl;
}

Staff::Staff(const Staff &obj) {
    note = new char[256];

    staffID = obj.staffID;
    name = obj.name;
    strcpy(note, obj.note);
}

Staff::~Staff() {
    delete []note;
    cout << "Staff::~Staff() is called" << endl;
}
```

```cpp
void DisplayInfo(Staff obj) {
    cout << "Staff ID : " << obj.staffID << endl;
    cout << "Name : " << obj.name << endl;
    cout << "Note : " << obj.note << endl;
}

int main() {
    Staff ben;

    ben.staffID = 201101;
    ben.name = "Ben";
    strcpy(ben.note, "This is a note for Ben.");
    cout << "calling DisplayInfo()" << endl;
    DisplayInfo(ben);
    cout << "back from DisplayInfo()" << endl;

    cout << "Note2 : " << ben.note << endl;

    return 0;
}
```

```
Staff::Staff() is called
calling DisplayInfo()
Staff ID : 201101
Name : Ben
Note : This is a note for Ben.
Staff::~Staff() is called
back from DisplayInfo()
Note2 : This is a note for Ben.
Staff::~Staff() is called
```

Copy Constructor

- It is only for "pass by value" (function call).
- not for "="

```
Staff ben;
Staff jack;
```

```
ben.staffID = 201101;
ben.name = "Ben";
strcpy(ben.note, "This is a note for Ben.");

jack = ben;
```

- for "=", overload "operator="

Friend Function

- allow a standalone function to allow full access to members.
- private:
- protected:
- public:

```
#include <iostream>
#include <string>

using namespace std;

class Staff {
    int staffID;
    string name;
    char *note;
public:
    Staff(int id, string n, char *no):
staffID(id), name(n) {
        note = new char[256];
        strcpy(note, no);
    };
    ~Staff();
    friend void showStaff(Staff obj);
};

Staff::~Staff() {
    delete []note;
    cout << "Staff::~Staff() is called" << endl;
}
```

```
void DisplayInfo(Staff obj) {
    cout << "Staff ID : " << obj.staffID << endl;
    cout << "Name : " << obj.name << endl;
    cout << "Note : " << obj.note << endl;
}

void showStaff(Staff obj) {
    cout << "Staff ID : " << obj.staffID << endl;
    cout << "Name : " << obj.name << endl;
    cout << "Note : " << obj.note << endl;
};

int main() {
    Staff ben(201101, "Ben", "This is a note for
Ben.");

    cout << "calling showStaff()" << endl;
    DisplayInfo(ben);
    showStaff(ben);
    cout << "back from showStaff()" << endl;

    return 0;
}
```

Friend Function

- is not a member function
  - can declare

"friend void showStaff(Staff obj);" in

private:

- friend function(s) cannot be inherited.
  - You would need to repeat in the subclass.

Friend Function

```
class YourClass;

class MyClass {
    int value;
public:
    MyClass(int v) : value(v) {};
    friend int compare(MyClass *a, YourClass *b);
}

class YourClass {
    int number;
public:
    YourClass(int v) : number(v) {};
    friend int compare(MyClass *a, YourClass *b);
}

int compare(MyClass *a, YourClass *b) {
    if (a->value > b->number)
        return 1;
    else if (a->value < b->number)
        return -1;
    else
        return 0;
}
```