

Classes that Work

Eric Berdahl

Channeling Sean Parent

June 26th, 2008



About this Talk

- Interoperability of software components increases productivity as well as quality, security, and performance
- This talk is from Adobe's Software Technology Lab which is responsible for developing better ways to write code
- This talk focuses on two of the basic operations of all types, copy and assignment, it is not the whole story but presents an interesting taste of the techniques being developed
- Proficiency with C++ and STL is assumed
- The proper use of header files, inline functions, and namespaces are ignored for clarity

client

library

cout

guidelines

defects

client

library

```
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

cout

Hello World!

client

library

cout

guidelines

defects

client

library

```
typedef int object_t;
```

```
void print(const object_t& x) { cout << x << endl; }
```

```
typedef vector<object_t> document_t;
```

```
void print(const document_t& x)
{
    cout << "<document>" << endl;
    for_each(x.begin(), x.end(), (void (*)(const object_t&))(&print));
    cout << "</document>" << endl;
}
```

cout

guidelines

defects

client

library

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

<document>

0

1

2

3

</document>

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

guidelines

- Write all code as a library.
 - Reuse increases your productivity.
 - Writing unit tests is simplified.

client

library

```
typedef int object_t;
```

```
void print(const object_t& x) { cout << x << endl; }
```

```
typedef vector<object_t> document_t;
```

```
void print(const document_t& x)
```

```
{
```

```
    cout << "<document>" << endl;
```

```
    for_each(x.begin(), x.end(), (void (*)(const object_t&))(&print));
```

```
    cout << "</document>" << endl;
```

```
}
```

cout

guidelines

defects

client

library

```
typedef vector<object_t> document_t;
```

```
void print(const document_t& x)
```

```
{
```

```
    cout << "<document>" << endl;
```

```
    for_each(x.begin(), x.end(), (void (*)(const object_t&))(&print));
```

```
    cout << "</document>" << endl;
```

```
}
```

cout

guidelines

defects

```
void print(const int& x) { cout << x << endl; }
```

```
class object_t {  
public:  
    explicit object_t(const int& x) : object_m(x)  
    { }
```

```
    friend void print(const object_t& x);  
private:  
    int object_m;  
};
```

```
void print(const object_t& x)  
{ print(x.object_m); }
```

```
typedef vector<object_t> document_t;
```

```
void print(const document_t& x)  
{  
    cout << "<document>" << endl;  
    for_each(x.begin(), x.end(), (void (*)(const object_t&))(&print));  
    cout << "</document>" << endl;  
}
```

client

library

```
void print(const int& x) { cout << x << endl; }

class object_t {
public:
    explicit object_t(const int& x) : object_m(x)
    { }

    friend void print(const object_t& x);
private:
    int object_m;
};

void print(const object_t& x)
{ print(x.object_m); }
```

typedef vector<object_t> document_t;

void print(const do

guidelines

- The compiler will supply member-wise copy and assignment operators.
 - Let the compiler do the work where appropriate.

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

<document>

0

1

2

3

</document>

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

guidelines

- Write classes that behave like *regular* objects to increase reuse.

client

library

```
void print(const int& x) { cout << x << endl; }
```

```
class object_t {  
public:
```

```
    explicit object_t(const int& x) : object_m(x)  
    { }
```

```
    friend void print(const object_t& x);
```

```
private:
```

```
    int object_m;
```

```
};
```

```
void print(const object_t& x)  
{ print(x.object_m); }
```

```
typedef vector<object_t> document_t;
```

```
void print(const document_t& x)  
{
```

```
    cout << "<document>" << endl;
```

```
    for_each(x.begin(), x.end(), (void (*)(const object_t&))(&print));
```

```
    cout << "</document>" << endl;
```

```
}
```

cout

guidelines

defects

client

library

+

```
class object_t {  
public:  
    explicit object_t(const int& x) : object_m(x)  
    { }  
  
    friend void print(const object_t& x);  
private:  
    int object_m;  
};
```

```
void print(const object_t& x)  
{ print(x.object_m); }
```

```
typedef vector<object_t> document_t;
```

```
void print(const document_t& x)  
{  
    cout << "<document>" << endl;  
    for_each(x.begin(), x.end(), (void (*)(const object_t&))(&print));  
    cout << "</document>" << endl;  
}
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:
```

```
        explicit object_t(const int& x) : object_m(x)  
        { }
```

```
        friend void print(const object_t& x);
```

```
    private:
```

```
        int object_m;
```

```
};
```

```
void print(const object_t& x)  
{ print(x.object_m); }
```

```
typedef vector<object_t> document_t;
```

```
void print(const document_t& x)  
{
```

```
    cout << "<document>" << endl;
```

```
    for_each(x.begin(), x.end(), (void (*)(const object_t&))(&print));
```

```
    cout << "</document>" << endl;
```

```
}
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:
```

+

```
typedef vector<object_t> document_t;
```

```
void print(const document_t& x)
```

+

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
  
        friend void print(const object_t& x);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};  
  
void print(const object_t& x)  
{ x.object_m->print_(); }
```

```
typedef vector<object_t> document_t;
```

```
void print(const document_t& x)
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }
```

```
friend void print(const object_t& x);  
private:  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
    int_model_t* object_m;  
};  
  
void print(const object_t& x)  
{ x.object_m->print_(); }  
  
typedef vector<object_t> document_t;
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }
```

```
    ~object_t() { delete object_m; }
```

```
    friend void print(const object_t& x);
```

```
private:
```

```
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }
```

```
        int data_m;
```

```
    };  
  
    int_model_t* object_m;
```

```
};  
  
void print(const object_t& x)  
{ x.object_m->print_(); }
```

```
typedef vector<object_t> document_t;
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        friend void print(const object_t& x);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

guidelines

- Do your own memory management - don't create garbage for your client to clean-up.

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }
```



```
        friend void print(const object_t& x);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

```
void print(const object_t& x)  
{ x.object_m->print_(); }
```



cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }
```

```
    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
    { }
```

```
    friend void print(const object_t& x);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

```
void print(const object_t& x)  
{ x.object_m->print_(); }
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
    ~object_t() { delete object_m; }
```

```
    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
    { }
```

```
    friend void print(const object_t& x);  
private:  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };
```

guidelines

- The semantics of copy are to create a new object which is equal to, and logically disjoint from, the original.
- Copy constructor must copy your object. The compiler is free to elide copies so if your copy constructor does something else your code is incorrect.
- When a type manages *remote parts* it is necessary to supply your own copy constructor.
 - If you can, use an existing class (such as a vector) to manage remote parts.

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
};
```

```
friend void print(const object_t& x);  
private:  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
    int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)  
        { delete object_m; object_m = new int_model_t(*x.object_m); return *this; }  
  
        friend void print(const object_t& x);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)  
        { delete object_m; object_m = new int_model_t(*x.object_m); return *this; }  
  
        friend void print(const object_t& x);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print(const int& x) { print(data_m); }  
  
            int data_m;  
        }  
};
```

guidelines

- Assignment is consistent with copy. Generally:
 $T\ x; x = y;$ is equivalent to $T\ x = y;$

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

<document>

0

1

2

3

</document>

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

guidelines

- The Private Implementation (Pimpl), or Handle-Body, idiom is good for separating the implementation and reducing compile times.

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)  
        { delete object_m; object_m = new int_model_t(*x.object_m); return *this; }  
  
        friend void print(const object_t& x);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)  
        { delete object_m; object_m = new int_model_t(*x.object_m); return *this; }  
  
        friend void print(const object_t& x);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print() const { print(data_m); }  
  
            int data_m;  
        };  
};
```

defects

- If new throws an exception, the object will be left in an invalid state.
- If we assign an object to itself this will crash.

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)
```

```
friend void print(const object_t& x);  
private:  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
    int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)  
        { object_t tmp(x); swap(*this, tmp); return *this; }  
  
        friend void print(const object_t& x);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)  
        { object_t tmp(x); swap(*this, tmp); return *this; }  
};
```

```
friend void print(const object_t& x);  
private:  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print() const { print(data_m); }  
  
        int data_m;
```

guidelines

- Assignment should satisfy the *strong exception guarantee*.
 - Either complete successfully or throw an exception, leaving the object unchanged.
- Don't optimize for rare cases which impact common cases.
 - Don't test for self-assignment to avoid the copy.

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)  
        { object_t tmp(x); swap(*this, tmp); return *this; }  
  
        friend void print(const object_t& x);  
  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)  
        { object_t tmp(x); swap(*this, tmp); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

public:

```
explicit object_t(const int& x) : object_m(new int_model_t(x))  
{ }  
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }
```

```
object_t& operator=(const object_t& x)  
{ object_t tmp(x); swap(*this, tmp); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

private:

```
struct int_model_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```

```
    int data_m;  
};
```

```
int_model_t* object_m;
```

```
};
```

```
explicit object_t(const int& x) : object_m(new int_model_t(x))
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print_() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};

void print(const object_t& x)
```

client

library

+

```
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print_() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};

void print(const object_t& x)
{ x.object_m->print_(); }
```

+

cout

guidelines

defects

client

library

```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }
```

```
object_t& operator=(const object_t& x)  
{ object_t tmp(x); swap(*this, tmp); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
struct int_model_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```

```
    int data_m;  
};
```

```
int_model_t* object_m;  
};
```

```
void print(const object_t& x)  
{ x.object_m->print_(); }
```

cout

guidelines

defects

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print_() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};

void print(const object_t& x)
{ x.object_m->print_(); }

typedef vector<object_t> document_t;
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};

void print(const object_t& x)
{ x.object_m->print(); }

typedef vector<object_t> document_t;
```


client

library

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};
```

cout

guidelines

defects

client

library

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};
```

```
void swap(object_t& x, object_t& y)
{
    swap(x.object_m, y.object_m);
    cout << "swap" << endl;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));
```

```
print(document);
return 0;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    reverse(document.begin(), document.end());

    print(document);
    return 0;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    reverse(document.begin(), document.end());

    print(document);
    return 0;
}
```

cout

```
swap
swap
<document>
3
2
1
0
</document>
```

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    reverse(document.begin(), document.end());

    print(document);
    return 0;
}
```

guidelines

- Provide a non-throwing swap function in the same namespace as the class.
- Call swap unqualified allowing argument dependent lookup (ADL) to find the best swap.
- The standard library treats swap as an operator and calls it unqualified.*

**Microsoft Visual C++ does not currently make unqualified calls to swap.*

client

library

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print_() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};

void swap(object_t& x, object_t& y)
{
    swap(x.object_m, y.object_m);
    cout << "swap" << endl;
}
```

cout

guidelines

defects

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print_() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};

void swap(object_t& x, object_t& y)
{
    swap(x.object_m, y.object_m);
}
```


client

library

+

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print_() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};

void swap(object_t& x, object_t& y)
{
    swap(x.object_m, y.object_m);
}
```

+

cout

guidelines

defects

client

library

```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }
```

```
object_t& operator=(const object_t& x)  
{ object_t tmp(x); swap(*this, tmp); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

private:

```
struct int_model_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print() const { print(data_m); }
```

```
    int data_m;  
};
```

```
int_model_t* object_m;  
};
```

```
void swap(object_t& x, object_t& y)  
{  
    swap(x.object_m, y.object_m);
```

cout

guidelines

defects

client

library

+

```
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print_() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};

void swap(object_t& x, object_t& y)
{
```

+

cout

guidelines

defects

```
explicit object_t(const int& x) : object_m(new int_model_t(x))
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(const object_t& x)
{ object_t tmp(x); swap(*this, tmp); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct int_model_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }

    int data_m;
};

int_model_t* object_m;
};

void swap(object_t& x, object_t& y)
```



public:

```
explicit object_t(const int& x) : object_m(new int_model_t(x))  
{ }  
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }  
object_t& operator=(const object_t& x)  
{ object_t tmp(x); swap(*this, tmp); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

private:

```
struct int_model_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```

```
    int data_m;  
};
```

```
int_model_t* object_m;
```

```
};
```



```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(const object_t& x)  
        { object_t tmp(x); swap(*this, tmp); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
  
        object_t& operator=(const object_t& x)  
        { object_t tmp(x); swap(*this, tmp); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { cout << "ctor" << endl; }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { cout << "copy" << endl; }  
        object_t& operator=(const object_t& x)  
        { object_t tmp(x); swap(*this, tmp); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

cout

guidelines

defects

client

library

```
object_t function() { return object_t(5); }
```

```
int main()
{
    /*
     * Quiz: What will this print?
     */

    object_t x = function();

    return 0;
}
```

cout

guidelines

defects

client

library

```
object_t function() { return object_t(5); }
```

```
int main()
{
    /*
     * Quiz: What will this print?
     */

    object_t x = function();

    return 0;
}
```

cout

ctor

client

library

```
object_t function() { return object_t(5); }
```

```
int main()  
{  
    /*  
    Quiz: What will this print?  
    */
```

```
    return 0;  
}
```

cout

guidelines

defects

client

library

```
object_t function() { return object_t(5); }
```

```
int main()
{
    /*
     * Quiz: What will this print?
     */
```

```
    object_t x(0);
```

```
    x = function();
```

```
    return 0;
```

```
}
```

cout

guidelines

defects

client

library

```
object_t function() { return object_t(5); }
```

```
int main()
{
    /*
     * Quiz: What will this print?
     */

    object_t x(0);

    x = function();

    return 0;
}
```

cout

ctor
ctor
copy

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { cout << "ctor" << endl; }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { cout << "copy" << endl; }  
        object_t& operator=(const object_t& x)  
        { object_t tmp(x); swap(*this, tmp); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

```
class object_t {  
public:  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { cout << "ctor" << endl; }  
    ~object_t() { delete object_m; }  
  
    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
    { cout << "copy" << endl; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
    int_model_t* object_m;  
};
```




```
class object_t {
public:
    explicit object_t(const int& x) : object_m(new int_model_t(x))
    { cout << "ctor" << endl; }
    ~object_t() { delete object_m; }

    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
    { cout << "copy" << endl; }
    object_t& operator=(object_t x)
    { swap(*this, x); return *this; }

    friend void print(const object_t& x);
    friend void swap(object_t& x, object_t& y);
private:
    struct int_model_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print_() const { print(data_m); }

        int data_m;
    };

    int_model_t* object_m;
};
```

client

library

```
class object_t {  
public:  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { cout << "ctor" << endl; }  
    ~object_t() { delete object_m; }  
  
    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
    { cout << "copy" << endl; }  
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }  
  
    friend void print(const object_t& x);  
    friend void swap(object_t& x, object_t& y);  
private:  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print(const int_model_t& data_m); }  
};
```

guidelines

- Pass *sink* arguments by value and swap or move into place.
- A sink argument is any argument consumed or returned by the function.
 - The argument to assignment is a sink argument.

client

library

```
object_t function() { return object_t(5); }
```

```
int main()
{
    /*
     * Quiz: What will this print?
     */

    object_t x(0);

    x = function();

    return 0;
}
```

cout

guidelines

defects

client

library

```
object_t function() { return object_t(5); }
```

```
int main()
{
    /*
     * Quiz: What will this print?
     */

    object_t x(0);

    x = function();

    return 0;
}
```

cout

ctor
ctor

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { cout << "ctor" << endl; }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { cout << "copy" << endl; }  
        object_t& operator=(object_t x)  
        { swap(*this, x); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
  
        object_t& operator=(object_t x)  
        { swap(*this, x); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(object_t x)  
        { swap(*this, x); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct int_model_t {  
            explicit int_model_t(const int& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            int data_m;  
        };  
  
        int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library

```
class object_t {  
public:
```



```
explicit object_t(const int& x) : object_m(new int_model_t(x))  
{ }  
~object_t() { delete object_m; }  
  
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:
```



cout

guidelines

defects

client

library

```
class object_t {  
    public:  
        explicit object_t(const string& x) : object_m(new string_model_t(x))  
        { }  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(object_t x)  
        { swap(*this, x); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct string_model_t {  
            explicit string_model_t(const string& x) : data_m(x) { }  
            void print() const { print(data_m); }  
  
            string data_m;  
        };  
};
```

cout

guidelines

defects

```
class object_t {  
    public:  
        explicit object_t(const string& x) : object_m(new string_model_t(x))  
        { }  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(object_t x)  
        { swap(*this, x); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct string_model_t {  
            explicit string_model_t(const string& x) : data_m(x) { }  
            void print_() const { print(data_m); }  
  
            string data_m;  
        };  
};
```



public:

```
explicit object_t(const string& x) : object_m(new string_model_t(x))  
{ }
```

```
explicit object_t(const int& x) : object_m(new int_model_t(x))  
{ }
```

```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
```

```
friend void swap(object_t& x, object_t& y);
```

private:

```
struct string_model_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```

```
    string data_m;  
};
```

```
struct int_model_t {
```



```
explicit object_t(const string& x) : object_m(new string_model_t(x))
{ }
explicit object_t(const int& x) : object_m(new int_model_t(x))
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct string_model_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print() const { print(data_m); }

    string data_m;
};

struct int_model_t {
    explicit int_model_t(const int& x) : data_m(x) { }
```



```
{ }
explicit object_t(const int& x) : object_m(new int_model_t(x))
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct string_model_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};

struct int_model_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }
```





```
explicit object_t(const int& x) : object_m(new int_model_t(x))
{ }
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
```

private:

```
struct string_model_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};
```

```
struct int_model_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }
```



```
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct string_model_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};

struct int_model_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }

    int data_m;
```

```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
struct string_model_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```

```
    string data_m;  
};
```

```
struct int_model_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```

```
    int data_m;  
};
```



```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct string_model_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};

struct int_model_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }

    int data_m;
};
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct string_model_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};

struct int_model_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }

    int data_m;
};

int_model_t* object_m;
```

client

library

```
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct string_model_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};  
  
struct int_model_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    int data_m;  
};  
  
int_model_t* object_m;  
};
```

cout

guidelines

defects

client

library



```
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
    struct string_model_t {  
        explicit string_model_t(const string& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        string data_m;  
    };  
  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
    int_model_t* object_m;  
};
```



cout

guidelines

defects

client

library



```
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
    struct string_model_t {  
        explicit string_model_t(const string& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        string data_m;  
    };  
  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
};
```



cout

guidelines

defects

client

library

+

```
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
    struct string_model_t {  
        explicit string_model_t(const string& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        string data_m;  
    };  
  
    struct int_model_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };
```

```
concept_t* object_m;
```

```
};
```

+

cout

guidelines

defects

client

library

+

```
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
struct string_model_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```

```
    string data_m;  
};
```

```
struct int_model_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```

```
    int data_m;  
};
```

```
concept_t* object_m;  
};
```

+

cout

guidelines

defects

client

library



```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:
```

```
explicit string_model_t(const string& x) : data_m(x) { }  
void print_() const { print(data_m); }
```

```
string data_m;  
};
```

```
explicit int_model_t(const int& x) : data_m(x) { }  
void print_() const { print(data_m); }
```

```
int data_m;  
};
```

```
concept_t* object_m;  
};
```



cout

guidelines

defects

client

library



```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:
```

```
struct concept_t {  
    virtual ~concept_t() { }  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};
```

```
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    int data_m;  
};
```

```
concept_t* object_m;  
};
```



cout

guidelines

defects

client

library

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:
```

```
struct concept_t {  
    virtual ~concept_t() { }
```

```
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print() const { print(data_m); }
```

```
    string data_m;  
};
```

```
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print() const { print(data_m); }
```

```
    int data_m;  
};
```

```
concept_t* object_m;
```

cout

guidelines

defects

client

library

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };  
  
    struct string_model_t : concept_t {  
        explicit string_model_t(const string& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        string data_m;  
    };  
  
    struct int_model_t : concept_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
    concept_t* object_m;
```

cout

guidelines

defects

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };  
  
    struct string_model_t : concept_t {  
        explicit string_model_t(const string& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        string data_m;  
    };  
  
    struct int_model_t : concept_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
    concept_t* object_m;
```

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};

struct int_model_t : concept_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }

    int data_m;
};
```

client

library

```
{ swap(*this, x); return *this; }
```



```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};  
  
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    int data_m;  
};
```



cout

guidelines

defects



```
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};
```

```
struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};
```

```
struct int_model_t : concept_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }

    int data_m;
```



```
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};  
  
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```



```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct concept_t {
        virtual ~concept_t() { }
        virtual void print_() const = 0;
    };

    struct string_model_t : concept_t {
        explicit string_model_t(const string& x) : data_m(x) { }
        void print_() const { print(data_m); }

        string data_m;
    };

    struct int_model_t : concept_t {
        explicit int_model_t(const int& x) : data_m(x) { }
        void print_() const { print(data_m); }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};

struct int_model_t : concept_t {
    explicit int_model_t(const int& x) : data_m(x) { }
```



```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};
```

```
struct int_model_t : concept_t {
```





```
{ }  
~object_t() { delete object_m; }  
  
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};
```





```
explicit object_t(const int& x) : object_m(new int_model_t(x))
{ }
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};
```

```
struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};
```





```
{ }  
explicit object_t(const int& x) : object_m(new int_model_t(x))  
{ }  
~object_t() { delete object_m; }  
  
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;
```





```
explicit object_t(const string& x) : object_m(new string_model_t(x))
{ }
explicit object_t(const int& x) : object_m(new int_model_t(x))
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }
```





public:

```
explicit object_t(const string& x) : object_m(new string_model_t(x))  
{ }
```

```
explicit object_t(const int& x) : object_m(new int_model_t(x))  
{ }
```

```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
```

```
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```




```
class object_t {  
    public:  
        explicit object_t(const string& x) : object_m(new string_model_t(x))  
        { }  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
        { }  
        object_t& operator=(object_t x)  
        { swap(*this, x); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct concept_t {  
            virtual ~concept_t() { }  
            virtual void print_() const = 0;  
        };  
  
        struct string_model_t : concept_t {  
            explicit string_model_t(const string& x) : data_m(x) { }
```

```
class object_t {
public:
    explicit object_t(const string& x) : object_m(new string_model_t(x))
    { }
    explicit object_t(const int& x) : object_m(new int_model_t(x))
    { }
    ~object_t() { delete object_m; }

    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))
    { }
    object_t& operator=(object_t x)
    { swap(*this, x); return *this; }

    friend void print(const object_t& x);
    friend void swap(object_t& x, object_t& y);
private:
    struct concept_t {
        virtual ~concept_t() { }
        virtual void print_() const = 0;
    };

    struct string_model_t : concept_t {
```

```
void print(const int& x) { cout << x << endl; }
```

```
class object_t {  
public:  
    explicit object_t(const string& x) : object_m(new string_model_t(x))  
    { }  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }  
    ~object_t() { delete object_m; }  
  
    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
    { }  
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }  
  
    friend void print(const object_t& x);  
    friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };  
};
```



client

library

```
void print(const int& x) { cout << x << endl; }
```

```
class object_t {  
public:  
    explicit object_t(const string& x) : object_m(new string_model_t(x))  
    { }  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }  
    ~object_t() { delete object_m; }  
  
    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
    { }  
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }  
  
    friend void print(const object_t& x);  
    friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };  
};
```

+

cout

guidelines

defects

client

library

```
void print(const string& x) { cout << x << endl; }
```

```
void print(const int& x) { cout << x << endl; }
```

```
class object_t {
```

```
public:
```

```
    explicit object_t(const string& x) : object_m(new string_model_t(x))  
    { }
```

```
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }
```

```
    ~object_t() { delete object_m; }
```

```
    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
    { }
```

```
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }
```

```
    friend void print(const object_t& x);
```

```
    friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };
```



cout

guidelines

defects

```
void print(const string& x) { cout << x << endl; }  
void print(const int& x) { cout << x << endl; }
```

```
class object_t {  
public:  
    explicit object_t(const string& x) : object_m(new string_model_t(x))  
    { }  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }  
    ~object_t() { delete object_m; }
```

```
    object_t(const object_t& x) : object_m(new int_model_t(*x.object_m))  
    { }
```

```
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }
```

```
    friend void print(const object_t& x);  
    friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };
```



client

library

```
void print(const string& x) { cout << x << endl; }  
void print(const int& x) { cout << x << endl; }
```

```
class object_t {  
public:  
    explicit object_t(const string& x) : object_m(new string_model_t(x))  
    { }  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }  
    ~object_t() { delete object_m; }
```

```
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};
```



cout

guidelines

defects

```
void print(const string& x) { cout << x << endl; }  
void print(const int& x) { cout << x << endl; }
```

```
class object_t {  
public:  
    explicit object_t(const string& x) : object_m(new string_model_t(x))  
    { }  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }  
    ~object_t() { delete object_m; }
```

```
    object_t(const object_t& x) : object_m(x.object_m->copy_())  
    { }
```

```
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }
```

```
    friend void print(const object_t& x);  
    friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };
```




```
void print(const string& x) { cout << x << endl; }  
void print(const int& x) { cout << x << endl; }
```

```
class object_t {  
public:  
    explicit object_t(const string& x) : object_m(new string_model_t(x))  
    { }  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }  
    ~object_t() { delete object_m; }  
  
    object_t(const object_t& x) : object_m(x.object_m->copy_())  
    { }  
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }  
  
    friend void print(const object_t& x);  
    friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };  
};
```

```
void print(const int& x) { cout << x << endl, }
```

```
class object_t {  
public:  
    explicit object_t(const string& x) : object_m(new string_model_t(x))  
    { }  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }  
    ~object_t() { delete object_m; }  
  
    object_t(const object_t& x) : object_m(x.object_m->copy_())  
    { }  
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }  
  
    friend void print(const object_t& x);  
    friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };  
};
```

```
class object_t {
public:
    explicit object_t(const string& x) : object_m(new string_model_t(x))
    { }
    explicit object_t(const int& x) : object_m(new int_model_t(x))
    { }
    ~object_t() { delete object_m; }

    object_t(const object_t& x) : object_m(x.object_m->copy_())
    { }
    object_t& operator=(object_t x)
    { swap(*this, x); return *this; }

    friend void print(const object_t& x);
    friend void swap(object_t& x, object_t& y);
private:
    struct concept_t {
        virtual ~concept_t() { }
        virtual void print_() const = 0;
    };

    struct string_model_t : concept_t {
```

```
class object_t {  
    public:  
        explicit object_t(const string& x) : object_m(new string_model_t(x))  
        { }  
        explicit object_t(const int& x) : object_m(new int_model_t(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(x.object_m->copy_())  
        { }  
        object_t& operator=(object_t x)  
        { swap(*this, x); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct concept_t {  
            virtual ~concept_t() { }  
            virtual void print_() const = 0;  
        };  
  
        struct string_model_t : concept_t {  
            explicit string_model_t(const string& x) : data_m(x) { }
```



public:

```
explicit object_t(const string& x) : object_m(new string_model_t(x))  
{ }
```

```
explicit object_t(const int& x) : object_m(new int_model_t(x))  
{ }
```

```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(x.object_m->copy_())  
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
```

```
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```





```
explicit object_t(const string& x) : object_m(new string_model_t(x))
{ }
explicit object_t(const int& x) : object_m(new int_model_t(x))
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(x.object_m->copy_())
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }
```





```
{ }  
explicit object_t(const int& x) : object_m(new int_model_t(x))  
{ }  
~object_t() { delete object_m; }  
  
object_t(const object_t& x) : object_m(x.object_m->copy_())  
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;
```





```
explicit object_t(const int& x) : object_m(new int_model_t(x))
{ }
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(x.object_m->copy_())
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};
```

```
struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};
```





```
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(x.object_m->copy_())
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};
```





```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(x.object_m->copy_())  
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};
```

```
struct int_model_t : concept_t {
```



```
object_t(const object_t& x) : object_m(x.object_m->copy_())
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct concept_t {
        virtual ~concept_t() { }
        virtual void print_() const = 0;
    };

    struct string_model_t : concept_t {
        explicit string_model_t(const string& x) : data_m(x) { }
        void print_() const { print(data_m); }

        string data_m;
    };

    struct int_model_t : concept_t {
        explicit int_model_t(const int& x) : data_m(x) { }
```

```
object_t(const object_t& x) : object_m(x.object_m->copy_())
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};

struct int_model_t : concept_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }
```

```
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};  
  
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }
```



```
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};
```

```
struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};
```

```
struct int_model_t : concept_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }

    int data_m;
```



client

library

```
{ swap(*this, x); return *this; }
```



```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print_() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};  
  
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print_() const { print(data_m); }  
  
    int data_m;  
};
```



cout

guidelines

defects

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    void print_() const { print(data_m); }

    string data_m;
};

struct int_model_t : concept_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    void print_() const { print(data_m); }

    int data_m;
};
```



```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };  
  
    struct string_model_t : concept_t {  
        explicit string_model_t(const string& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        string data_m;  
    };  
  
    struct int_model_t : concept_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
    concept_t* object_m;
```

```
friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual void print_() const = 0;  
    };  
  
    struct string_model_t : concept_t {  
        explicit string_model_t(const string& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        string data_m;  
    };  
  
    struct int_model_t : concept_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        void print_() const { print(data_m); }  
  
        int data_m;  
    };  
  
    concept_t* object_m;  
};
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual void print() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    void print() const { print(data_m); }  
  
    string data_m;  
};
```

```
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    void print() const { print(data_m); }  
  
    int data_m;  
};
```

```
concept_t* object_m;  
};
```

client

library

+

private:

```
struct concept_t {  
    virtual ~concept_t() { }
```

```
virtual void print_() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }
```

```
void print_() const { print(data_m); }
```

```
string data_m;  
};
```

```
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }
```

```
void print_() const { print(data_m); }
```

```
int data_m;  
};
```

+

cout

guidelines

defects

client

library

+

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy() const = 0;  
    virtual void print() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    concept_t* copy() const { return new string_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    string data_m;  
};
```

```
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    concept_t* copy() const { return new int_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    int data_m;  
};
```

+

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(1));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(string("Hello")));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

guidelines

defects

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(string("Hello")));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

cout

```
<document>
0
Hello
2
3
</document>
```

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(string("Hello")));
    document.push_back(object_t(2));
    document.push_back(object_t(3));

    print(document);
    return 0;
}
```

guidelines

- Don't allow polymorphism to complicate the client code.
 - Polymorphism is an implementation detail.

client

library

```
void print(const string& x) { cout << x << endl; }  
void print(const int& x) { cout << x << endl; }
```

```
class object_t {  
public:
```

```
    explicit object_t(const string& x) : object_m(new string_model_t(x))  
    { }  
    explicit object_t(const int& x) : object_m(new int_model_t(x))  
    { }  
    ~object_t() { delete object_m; }
```

```
    object_t(const object_t& x) : object_m(x.object_m->copy_())  
    { }  
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }
```

```
    friend void print(const object_t& x);  
    friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual concept_t* copy_() const = 0;  
        virtual void print_() const = 0;
```



cout

guidelines

defects

client

library

```
class object_t {  
public:
```

```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(x.object_m->copy_())  
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy_() const = 0;  
    virtual void print_() const = 0;  
};
```



cout

guidelines

defects

client

library

```
template <typename T>
void print(const T& x) { cout << x << endl; }
```

```
class object_t {
public:
```

```
    template <typename T>
    explicit object_t(const T& x) : object_m(new model<T>(x))
    { }
```

```
    ~object_t() { delete object_m; }
```

```
    object_t(const object_t& x) : object_m(x.object_m->copy_())
    { }
```

```
    object_t& operator=(object_t x)
    { swap(*this, x); return *this; }
```

```
    friend void print(const object_t& x);
    friend void swap(object_t& x, object_t& y);
```

```
private:
```

```
    struct concept_t {
        virtual ~concept_t() { }
        virtual concept_t* copy_() const = 0;
        virtual void print_() const = 0;
    };
};
```



cout

guidelines

defects

```
template <typename T>
void print(const T& x) { cout << x << endl; }

class object_t {
public:
    template <typename T>
    explicit object_t(const T& x) : object_m(new model<T>(x))
    { }
    ~object_t() { delete object_m; }

    object_t(const object_t& x) : object_m(x.object_m->copy_())
    { }
    object_t& operator=(object_t x)
    { swap(*this, x); return *this; }

    friend void print(const object_t& x);
    friend void swap(object_t& x, object_t& y);
private:
    struct concept_t {
        virtual ~concept_t() { }
        virtual concept_t* copy_() const = 0;
        virtual void print_() const = 0;
    };
};
```

```
void print(const T& x) { cout << x << endl; }
```

```
class object_t {  
public:  
    template <typename T>  
    explicit object_t(const T& x) : object_m(new model<T>(x))  
    { }  
    ~object_t() { delete object_m; }  
  
    object_t(const object_t& x) : object_m(x.object_m->copy_())  
    { }  
    object_t& operator=(object_t x)  
    { swap(*this, x); return *this; }  
  
    friend void print(const object_t& x);  
    friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual concept_t* copy_() const = 0;  
        virtual void print_() const = 0;  
    };  
};
```

```
class object_t {
public:
    template <typename T>
    explicit object_t(const T& x) : object_m(new model<T>(x))
    { }
    ~object_t() { delete object_m; }

    object_t(const object_t& x) : object_m(x.object_m->copy_())
    { }
    object_t& operator=(object_t x)
    { swap(*this, x); return *this; }

    friend void print(const object_t& x);
    friend void swap(object_t& x, object_t& y);
private:
    struct concept_t {
        virtual ~concept_t() { }
        virtual concept_t* copy_() const = 0;
        virtual void print_() const = 0;
    };

    struct string_model_t : concept_t {
```



```
class object_t {  
    public:  
        template <typename T>  
        explicit object_t(const T& x) : object_m(new model<T>(x))  
        { }  
        ~object_t() { delete object_m; }  
  
        object_t(const object_t& x) : object_m(x.object_m->copy_())  
        { }  
        object_t& operator=(object_t x)  
        { swap(*this, x); return *this; }  
  
        friend void print(const object_t& x);  
        friend void swap(object_t& x, object_t& y);  
    private:  
        struct concept_t {  
            virtual ~concept_t() { }  
            virtual concept_t* copy_() const = 0;  
            virtual void print_() const = 0;  
        };  
  
        struct string_model_t : concept_t {  
            explicit string_model_t(const string& x) : data_m(x) { }
```

public:

```
template <typename T>
explicit object_t(const T& x) : object_m(new model<T>(x))
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(x.object_m->copy_())
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {
    virtual ~concept_t() { }
    virtual concept_t* copy_() const = 0;
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    concept_t* copy_() const { return new string_model_t(data_m); }
```



```
template <typename T>
explicit object_t(const T& x) : object_m(new model<T>(x))
{ }
~object_t() { delete object_m; }

object_t(const object_t& x) : object_m(x.object_m->copy_())
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual concept_t* copy_() const = 0;
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    concept_t* copy_() const { return new string_model_t(data_m); }
    void print_() const { print(data_m); }
```





```
explicit object_t(const T& x) : object_m(new model<T>(x))  
{ }  
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(x.object_m->copy_())  
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy_() const = 0;  
    virtual void print_() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    concept_t* copy_() const { return new string_model_t(data_m); }  
    void print_() const { print(data_m); }
```



```
{ }  
~object_t() { delete object_m; }  
  
object_t(const object_t& x) : object_m(x.object_m->copy_())  
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy_() const = 0;  
    virtual void print_() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    concept_t* copy_() const { return new string_model_t(data_m); }  
    void print_() const { print(data_m); }  
  
    string data_m;
```

client

library



```
~object_t() { delete object_m; }
```

```
object_t(const object_t& x) : object_m(x.object_m->copy_())  
{ }
```

```
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy_() const = 0;  
    virtual void print_() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    concept_t* copy_() const { return new string_model_t(data_m); }  
    void print_() const { print(data_m); }  
  
    string data_m;  
};
```



cout

guidelines

defects

```
object_t(const object_t& x) : object_m(x.object_m->copy_())
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual concept_t* copy_() const = 0;
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    concept_t* copy_() const { return new string_model_t(data_m); }
    void print_() const { print(data_m); }

    string data_m;
};
```

```
object_t(const object_t& x) : object_m(x.object_m->copy_())
{ }
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }

friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
    struct concept_t {
        virtual ~concept_t() { }
        virtual concept_t* copy_() const = 0;
        virtual void print_() const = 0;
    };

    struct string_model_t : concept_t {
        explicit string_model_t(const string& x) : data_m(x) { }
        concept_t* copy_() const { return new string_model_t(data_m); }
        void print_() const { print(data_m); }

        string data_m;
    };

    struct int_model_t : concept_t {
```



```
{ }  
object_t& operator=(object_t x)  
{ swap(*this, x); return *this; }  
  
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy() const = 0;  
    virtual void print() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    concept_t* copy() const { return new string_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    string data_m;  
};  
  
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }
```

client

library



```
object_t& operator=(object_t x)
{ swap(*this, x); return *this; }
```

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {
    virtual ~concept_t() { }
    virtual concept_t* copy() const = 0;
    virtual void print() const = 0;
};
```

```
struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    concept_t* copy() const { return new string_model_t(data_m); }
    void print() const { print(data_m); }

    string data_m;
};
```

```
struct int_model_t : concept_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    concept_t* copy() const { return new int_model_t(data_m); }
```



cout

guidelines

defects

client

library

```
{ swap(*this, x); return *this; }
```

+

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy() const = 0;  
    virtual void print() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    concept_t* copy() const { return new string_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    string data_m;  
};
```

```
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    concept_t* copy() const { return new int_model_t(data_m); }  
    void print() const { print(data_m); }
```

+

cout

guidelines

defects

```
friend void print(const object_t& x);
friend void swap(object_t& x, object_t& y);
private:
struct concept_t {
    virtual ~concept_t() { }
    virtual concept_t* copy_() const = 0;
    virtual void print_() const = 0;
};

struct string_model_t : concept_t {
    explicit string_model_t(const string& x) : data_m(x) { }
    concept_t* copy_() const { return new string_model_t(data_m); }
    void print_() const { print(data_m); }

    string data_m;
};

struct int_model_t : concept_t {
    explicit int_model_t(const int& x) : data_m(x) { }
    concept_t* copy_() const { return new int_model_t(data_m); }
    void print_() const { print(data_m); }
```

```
friend void print(const object_t& x);  
friend void swap(object_t& x, object_t& y);  
private:  
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy() const = 0;  
    virtual void print() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    concept_t* copy() const { return new string_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    string data_m;  
};  
  
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    concept_t* copy() const { return new int_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    int data_m;
```

```
friend void swap(object_t& x, object_t& y);  
private:  
    struct concept_t {  
        virtual ~concept_t() { }  
        virtual concept_t* copy() const = 0;  
        virtual void print() const = 0;  
    };  
  
    struct string_model_t : concept_t {  
        explicit string_model_t(const string& x) : data_m(x) { }  
        concept_t* copy() const { return new string_model_t(data_m); }  
        void print() const { print(data_m); }  
  
        string data_m;  
    };  
  
    struct int_model_t : concept_t {  
        explicit int_model_t(const int& x) : data_m(x) { }  
        concept_t* copy() const { return new int_model_t(data_m); }  
        void print() const { print(data_m); }  
  
        int data_m;  
    };
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy() const = 0;  
    virtual void print() const = 0;  
};  
  
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    concept_t* copy() const { return new string_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    string data_m;  
};  
  
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    concept_t* copy() const { return new int_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    int data_m;  
};
```

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy() const = 0;  
    virtual void print() const = 0;  
};
```

```
struct string_model_t : concept_t {  
    explicit string_model_t(const string& x) : data_m(x) { }  
    concept_t* copy() const { return new string_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    string data_m;  
};
```

```
struct int_model_t : concept_t {  
    explicit int_model_t(const int& x) : data_m(x) { }  
    concept_t* copy() const { return new int_model_t(data_m); }  
    void print() const { print(data_m); }  
  
    int data_m;  
};
```


client

library

+

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy() const = 0;  
    virtual void print() const = 0;  
};
```

```
concept_t* object_m;
```

```
};
```

```
void swap(object_t& x, object_t& y)  
{  
    swap(x.object_m, y.object_m);  
}
```

+

cout

guidelines

defects

private:

```
struct concept_t {  
    virtual ~concept_t() { }  
    virtual concept_t* copy() const = 0;  
    virtual void print() const = 0;  
};
```

```
template <typename T>  
struct model : concept_t {  
    explicit model(const T& x) : data_m(x) { }  
    concept_t* copy() const { return new model(data_m); }  
    void print() const { print(data_m); }  
  
    T data_m;  
};
```

```
concept_t* object_m;  
};  
  
void swap(object_t& x, object_t& y)  
{  
    swap(x.object_m, y.object_m);  
}
```

client

library

```
int main()
{
    document_t document;

    document.push_back(object_t(0));
    document.push_back(object_t(string("Hello")));
    document.push_back(object_t(2));
```

```
    print(document);
    return 0;
}
```

cout

guidelines

defects

client

library

```
class my_class_t {  
    /* ... */  
};  
  
void print(const my_class_t& x)  
{ cout << "my_class_t" << endl; }
```

```
int main()  
{  
    document_t document;  
  
    document.push_back(object_t(0));  
    document.push_back(object_t(string("Hello")));  
    document.push_back(object_t(2));  
    document.push_back(object_t(my_class_t()));  
  
    print(document);  
    return 0;  
}
```

cout

guidelines

defects

client

library

```
class my_class_t {  
    /* ... */  
};  
  
void print(const my_class_t& x)  
{ cout << "my_class_t" << endl; }
```

```
int main()  
{  
    document_t document;  
  
    document.push_back(object_t(0));  
    document.push_back(object_t(string("Hello")));  
    document.push_back(object_t(2));  
    document.push_back(object_t(my_class_t()));  
}
```

cout

```
<document>  
0  
Hello  
2  
my_class_t  
</document>
```

guidelines

client

library

```
class my_class_t {  
    /* ... */  
};  
  
void print(const my_class_t& x)  
{ cout << "my_class_t" << endl; }
```

```
int main()  
{  
    document_t document;  
  
    document.push_back(object_t(0));  
    document.push_back(object_t(string("Hello")));  
    document.push_back(object_t(2));  
    document.push_back(object_t(my_class_t()));  
  
    print(document);  
    return 0;  
}
```

guidelines

- The runtime-concept idiom allows polymorphism when needed without inheritance.
 - Client isn't burdened with inheritance, factories, class registration, and memory management.
 - Penalty of runtime polymorphism is only paid when needed.
 - Polymorphic types are used like any other types, including built-in types.

client

library

```
class my_class_t {  
    /* ... */  
};  
  
void print(const my_class_t& x)  
{ cout << "my_class_t" << endl; }  
  
int main()  
{  
    document_t document;  
  
    document.push_back(object_t(0));  
    document.push_back(object_t(string("Hello")));  
  
    document.push_back(object_t(my_class_t()));  
  
    print(document);  
    return 0;  
}
```

cout

guidelines

defects

client

library

```
class my_class_t {  
    /* ... */  
};  
  
void print(const my_class_t& x)  
{ cout << "my_class_t" << endl; }  
  
int main()  
{  
    document_t document;  
  
    document.push_back(object_t(0));  
    document.push_back(object_t(string("Hello")));  
    document.push_back(object_t(document));  
    document.push_back(object_t(my_class_t()));  
  
    print(document);  
    return 0;  
}
```

cout

guidelines

defects

client

library

```
class my_class_t {  
    /* ... */  
};  
  
void print(const my_class_t& x)  
{ cout << "my_class_t" << endl; }  
  
int main()  
{  
    document_t document;  
  
    document.push_back(object_t(0));  
    document.push_back(object_t(string("Hello")));  
    document.push_back(object_t(document));  
}
```

cout

```
<document>  
0  
Hello  
<document>  
0  
Hello  
</document>  
my_class_t  
</document>
```

Concluding Remarks

- Interoperability of software components increases productivity as well as quality, security, and performance
- Focusing on the properties of the types instead of the types themselves allows us to create interoperable components with less work than specialized components - with no loss (and often a gain) in performance and code size
- There is even more to say about copy and assignment - which are just two of many properties of *regular* types
- See <http://stlab.corp.adobe.com/> for links and contact information for the Software Technology Lab



Adobe