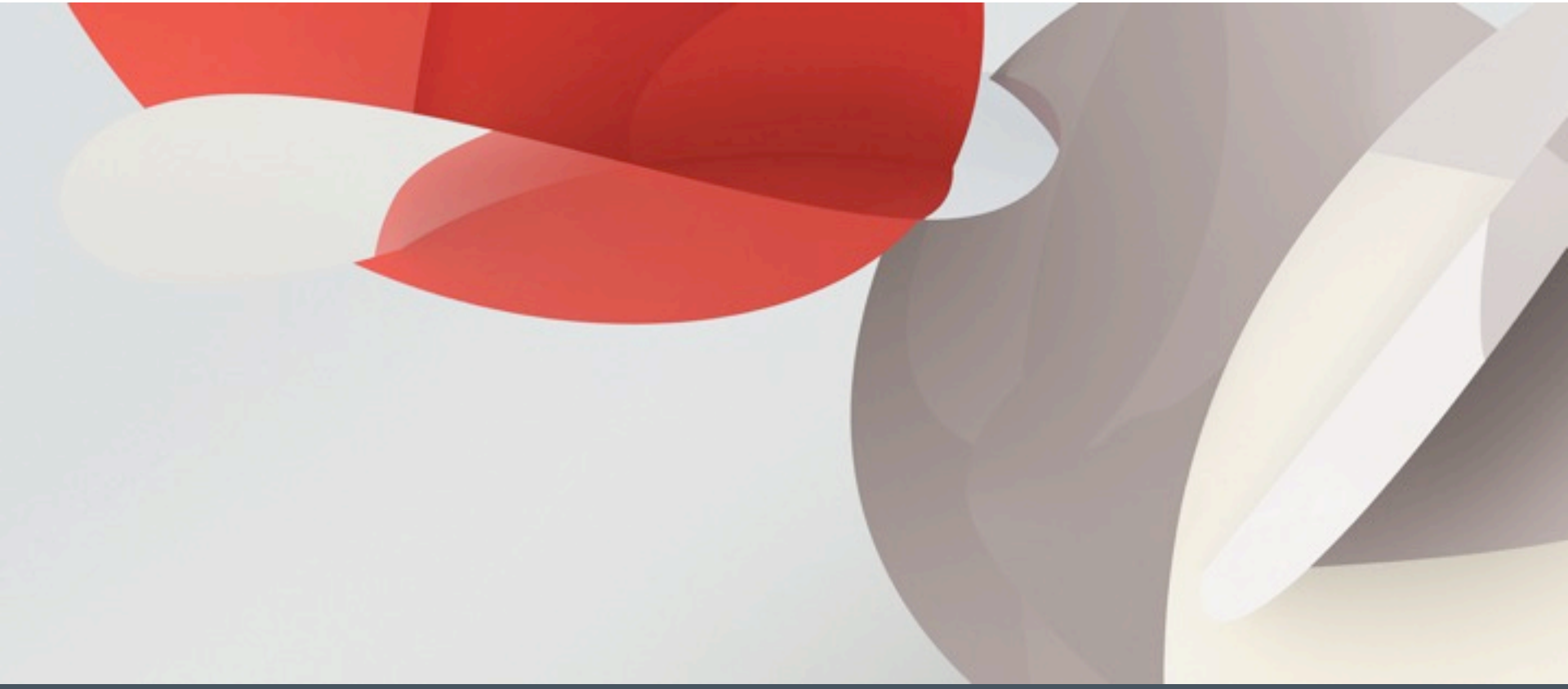
Value Semantics and Concept Based Polymorphism

Sean Parent | Principal Scientist



Outline of Talk

- Defining Value Semantics
- “Polymorphic Types”
- Demo of Photoshop History
- Implement History

Disclaimer

- In the following code, the proper use of header files, inline functions, and namespaces are ignored for clarity

client | library

cout | guidelines | defects

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

client | library

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

cout

Hello World!

```
using object_t = int;

void draw(const object_t& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

client | library

cout | guidelines | defects

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

cout

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

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

guidelines

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

Polymorphism

- What happens if we want the document to hold any drawable object?

```
class object_t {
  public:
    virtual ~object_t() { }
    virtual void draw(ostream&, size_t) const = 0;
};

using document_t = vector<shared_ptr<object_t>>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) e->draw(out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

client | library

cout | guidelines | defects

```
class my_class_t : public object_t
{
  public:
    void draw(ostream& out, size_t position) const
    { out << string(position, ' ') << "my_class_t" << endl; }
    /* ... */
};


int main()
{
    document_t document;

    document.emplace_back(new my_class_t());

    draw(document, cout, 0);
}
```

client | library

```
class my_class_t : public object_t
{
  public:
    void draw(ostream& out, size_t position) const
    { out << string(position, ' ') << "my_class_t" << endl; }
    /* ... */
};


int main()
{
    document_t document;

    document.emplace_back(new my_class_t());

    draw(document, cout, 0);
}
```

cout

```
<document>
 my_class_t
</document>
```

client | library

```
class my_class_t : public object_t
{
  public:
    void draw(ostream& out, size_t position) const
    { out << string(position, ' ') << "my_class_t" << endl; }
    /* ... */
};


int main()
{
    document_t document;

    document.emplace_back(new my_class_t());

    draw(document, cout, 0);
}
```

defects

- An instance of my_class_t will be allocated first
- Then the document will grow to make room
- If growing the document throws an exception, the memory from my_class_t is leaked

client | library

```
class my_class_t : public object_t
{
  public:
    void draw(ostream& out, size_t position) const
    { out << string(position, ' ') << "my_class_t" << endl; }
    /* ... */
};


int main()
{
    document_t document;

    document.emplace_back(make_shared<my_class_t>());

    draw(document, cout, 0);
}
```

defects

- An instance of my_class_t will be allocated first
- Then the document will grow to make room
- If growing the document throws an exception, the memory from my_class_t is leaked

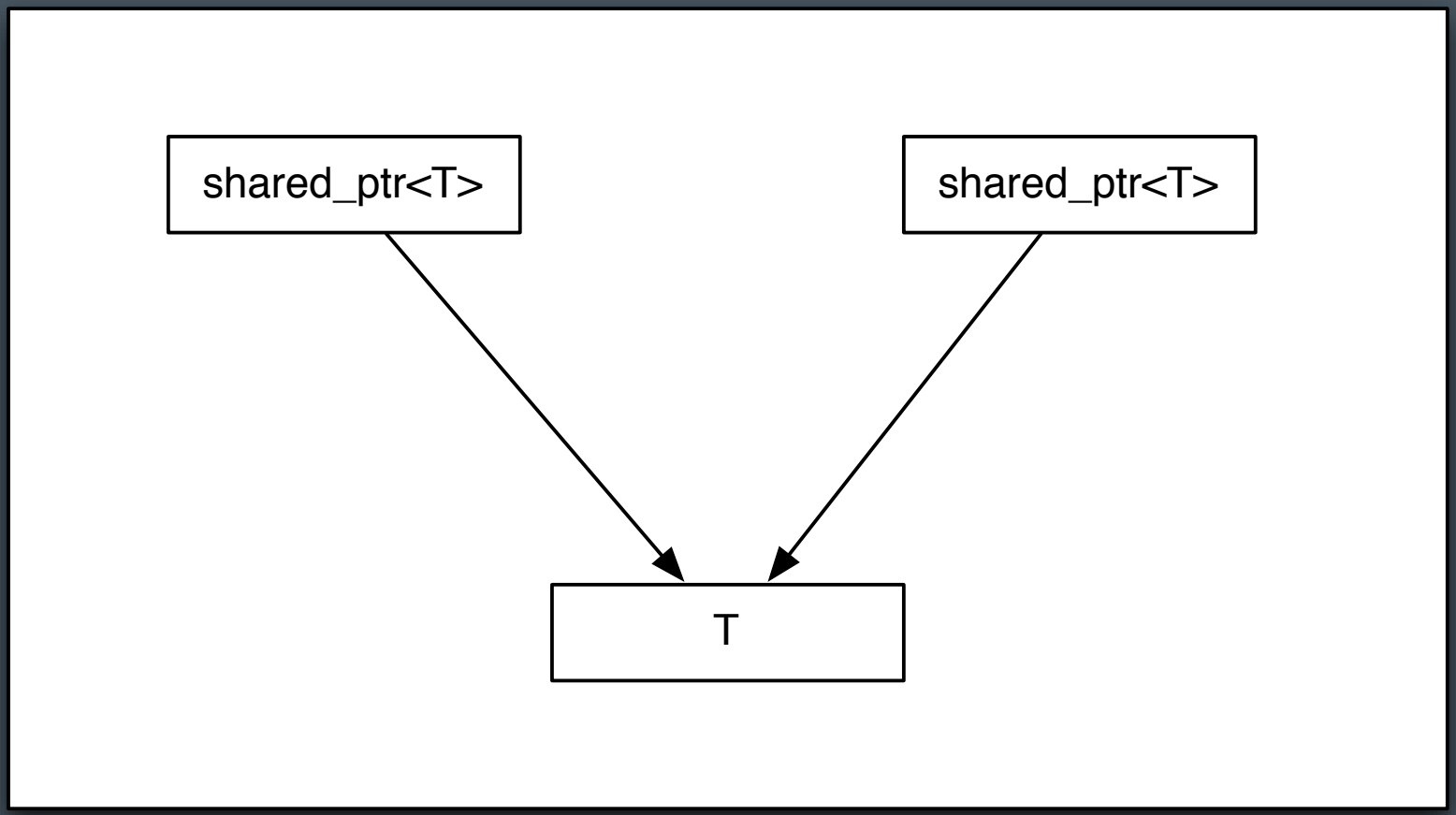
Deep problem #1

- Changed semantics of copy, assignment, and equality of my document
  - leads to incidental data structures
  - thread safety concerns

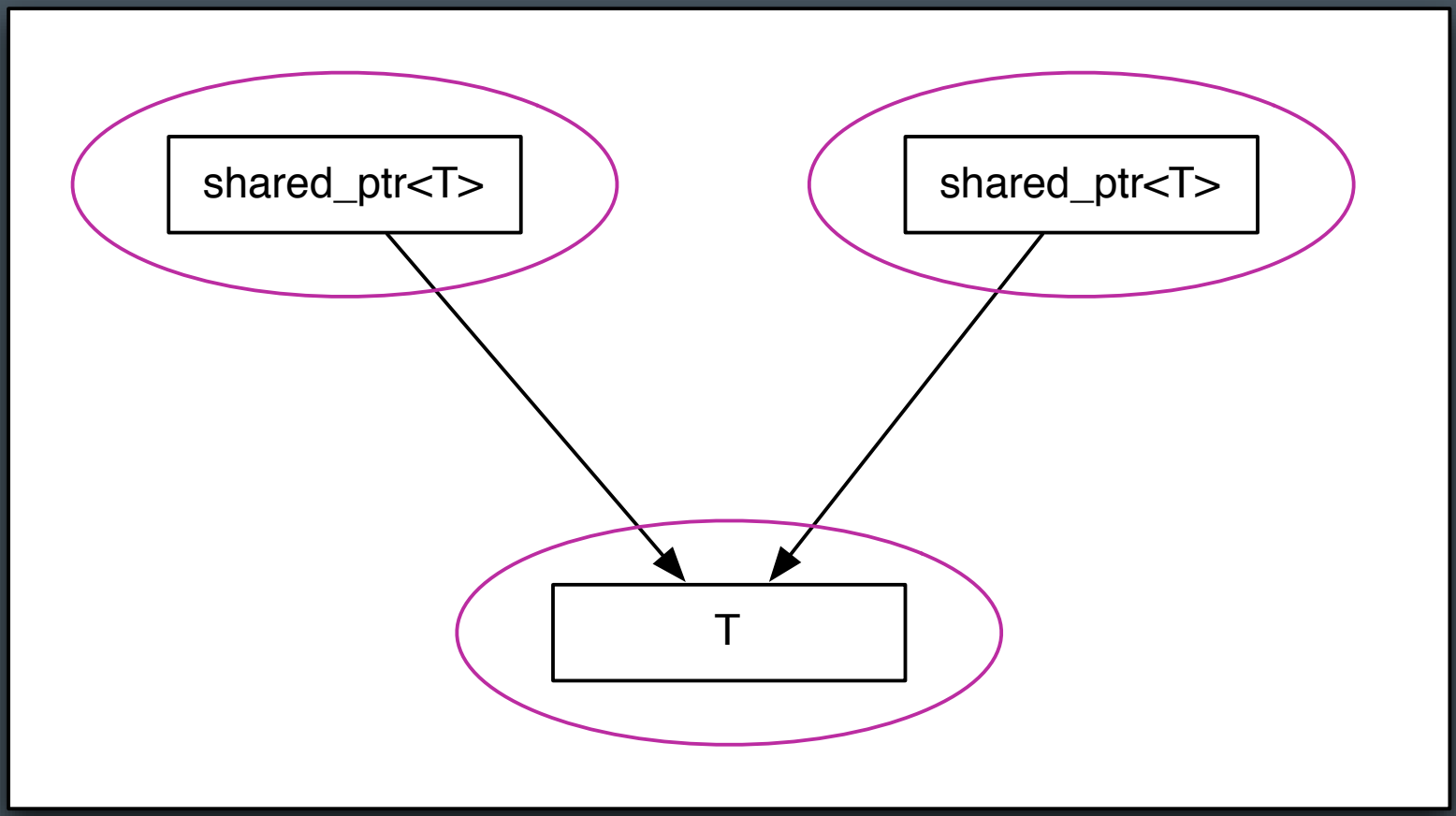
Semantics & Syntax

- We define an operation in terms of the operation's semantics:
  - "Assignment is a procedure taking two objects of the same type that makes the first object equal to the second without modifying the second."

Semantics & Syntax



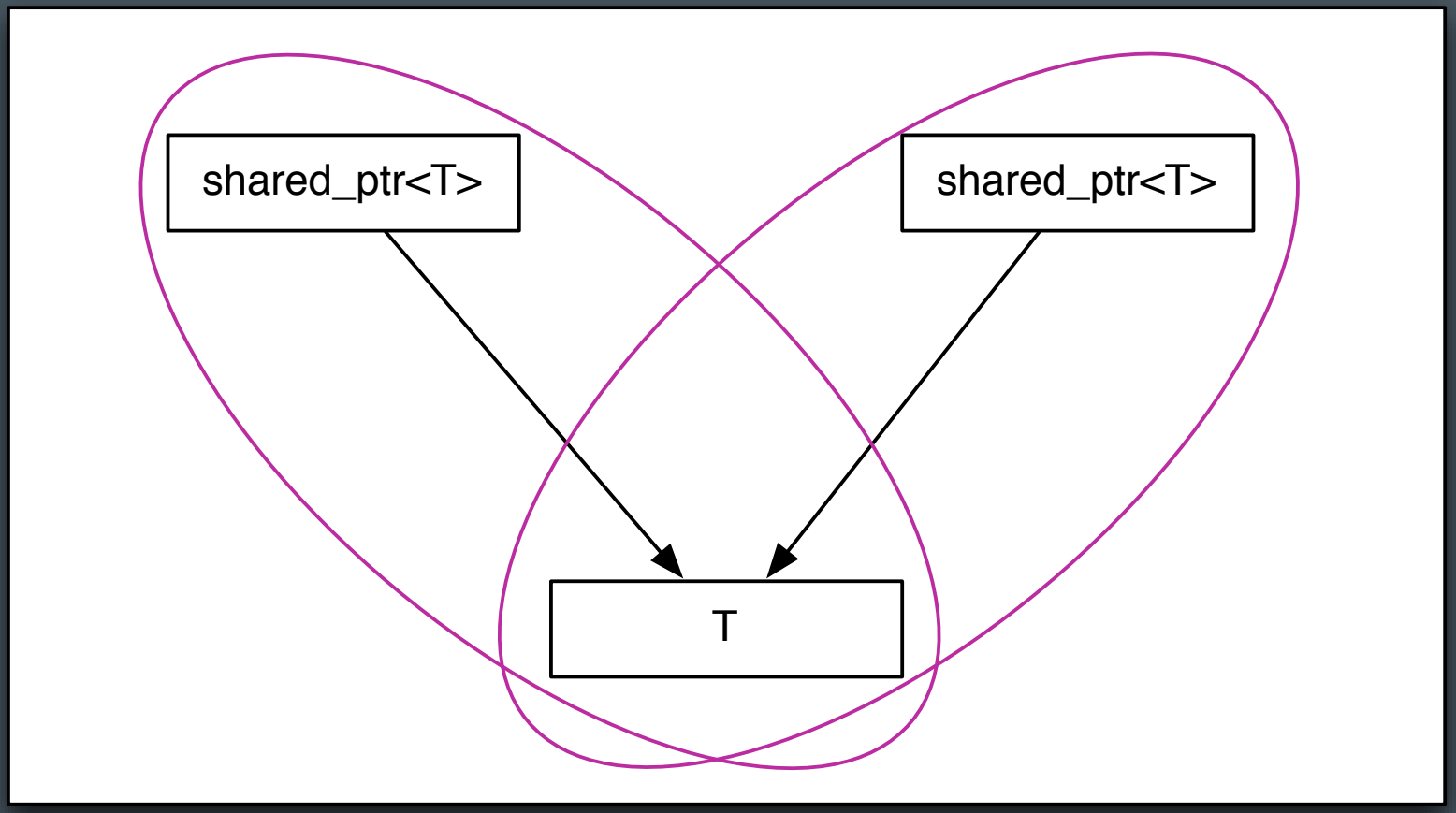
Semantics & Syntax



Semantics & Syntax

- Considered as individual types, assignment and copy hold their regular semantic meanings
  - However, this fails to account for the relationships (the arrows) which form an incidental data-structure. You cannot operate on T through one of the shared pointers without considering the effect on the other shared pointer

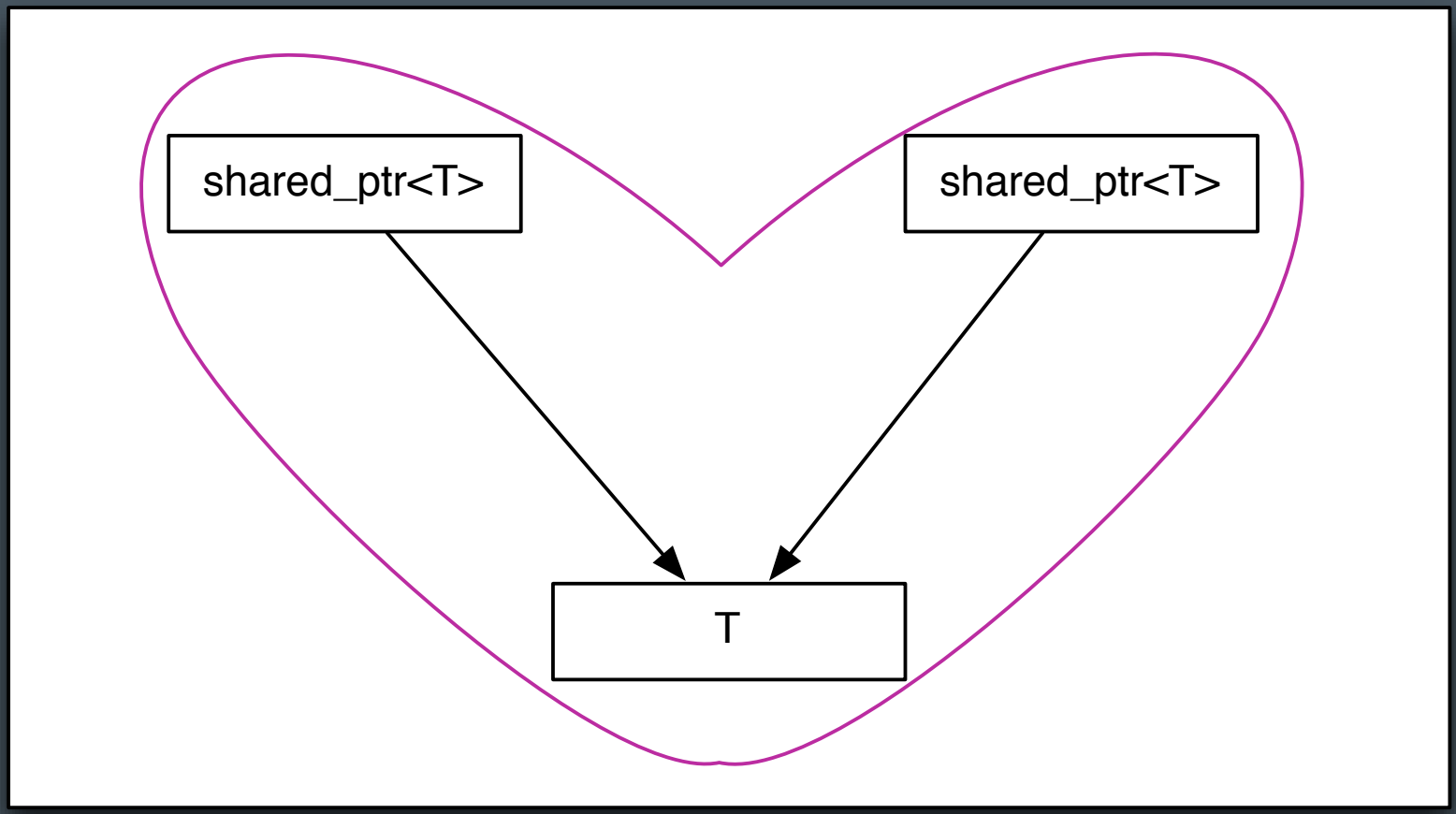
Semantics & Syntax



Semantics & Syntax

- If we extend our notion of object type to include the directly related part then we have intersecting objects which will interfere with each other

Semantics & Syntax



Semantics & Syntax

- When we consider the whole, the standard syntax for copy and assignment no longer have their regular semantics.
  - This structure is still copyable and assignable but these operations must be done through other means

- The shared structure also breaks our ability to reason locally about the code
  - A shared pointer is as good as a global variable

Semantics & Syntax

- Choosing the same syntax for the same semantics enables code reuse and avoids combinatoric interfaces
  - If a type has a proper set of basis operations then it can be adapted to any alternative set of basis operations regardless of syntax

- C++ has defined semantics for operations on built-in types, including assignment, copy, equality, address-of
  - Using the same operator names to provide the same semantics on user types enables code reuse

Regular Types

"There is a set of procedures whose inclusion in the computational basis of a type lets us place objects in data structures and use algorithms to copy objects from one data structure to another. We call types having such a basis regular, since their use guarantees regularity of behavior and, therefore, interoperability." – *Elements of Programming, Section 1.5*

Semantics & Syntax

- Regular types where the regular operations are implemented with the standard names are said to have *value semantics*

- When objects are referred to indirectly, through a shared reference or pointer, the objects are said to have *reference semantics*

Semantics & Syntax

- The shared structure also breaks our ability to reason locally about the code
  - A shared pointer is as good as a global variable

Deep problem #2

- Inefficient
  - calls to draw() on my_class_t are *always* virtual as is the destructor
  - my_class_t is *always* heap allocated
  - access to my class_t must be synchronized

Deep problem #3

- Polymorphism is intrusive
  - Document can no longer hold a drawable integer

“Polymorphic Types”

- The requirement of a polymorphic type, by definition, comes from it's use
- There are no polymorphic types, only a *polymorphic use* of similar types

“Polymorphic Types”

- By using inheritance to capture polymorphic use, we shift the burden of use to the type implementation, tightly coupling components
- Inheritance implies variable size, which implies heap allocation
- Heap allocation forces a further burden to manage the object lifetime
- Indirection, heap allocation, virtualization impacts performance
- Object lifetime management leads to garbage collection or reference counting
- This encourages *shared* ownership and the proliferation of *incidental data-structures*
- Shared ownership leads to synchronization issues, breaks local reasoning, and further impacts performance

“Polymorphic Types”

- By using inheritance to capture polymorphic use, we shift the burden of use to the type implementation, tightly coupling components
- Inheritance implies variable size, which implies heap allocation
- Heap allocation forces a further burden to manage the object lifetime
- Indirection, heap allocation, virtualization impacts performance
- Object lifetime management leads to garbage collection or reference counting
- This encourages *shared* ownership and the proliferation of *incidental data-structures*
- Shared ownership leads to synchronization issues, breaks local reasoning, and further impacts performance

Inheritance is the base class of Evil

```
using object_t = int;

void draw(const object_t& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

```
using object_t = int;

void draw(const object_t& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

client | library

```
using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

```
void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { draw(x.self_, out, position); }

  private:
    int self_;
};
```

```
using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

client | library

```
void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { draw(x.self_, out, position); }

  private:
    int self_;
};

using document_t = vector<object_t>;

void draw(const d                tream& out, size_t position)
{
    out << string                << "<document>" << endl;
```

guidelines

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

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

cout

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

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

guidelines

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

```
void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { draw(x.self_, out, position); }

  private:
    int self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

```
{ out << string(position, ' ') << x << endl; }

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { draw(x.self_, out, position); }

  private:
    int self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { draw(x.self_, out, position); }

  private:
    int self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { draw(x.self_, out, position); }

  private:
    int self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { draw(x.self_, out, position); }

  private:
    int self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

client | library

```
class object_t {
  public:
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
```

cout | guidelines | defects

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

    unique_ptr<int_model_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
```

client | library

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

    unique_ptr<int
};
```

guidelines

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

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

    unique_ptr<int_model_t> self_;
};

using document_t = vector<object_t>;
```

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

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

    unique_ptr<int_model_t> self_;
};

using document_t = vector<object_t>;
```

client | library

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

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_
    };
```

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 the object. The compiler is free to elide copies so if the copy constructor does something else the code is incorrect.
- When a type manages *remote parts* it is necessary to supply a copy constructor.
  - If you can, use an existing class (such as a vector) to manage remote parts.

client | library

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

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

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

    unique_ptr<int_model_t> self_;
};
```

cout | guidelines | defects

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t& operator=(const object_t& x)
    { object_t tmp(x); self_ = move(tmp.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

    unique_ptr<int_model_t> self_;
};
```

client | library

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t& operator=(const object_t& x)
    { object_t tmp(x); self_ = move(tmp.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(da                    on); }

        int data
```

guidelines

- Assignment is consistent with copy. Generally:
  `T x; x = y;` is equivalent to `T x = y;`
- Assignment satisfying the *strong exception guarantee* is a nice property.
  - Either complete successfully or throw an exception, leaving the object unchanged.
- Assignment (like all other operations) must satisfy the basic exception guarantee.
- Don't optimize for rare cases which impact common cases.
  - Don't test for self-assignment to avoid the copy.

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

cout

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

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

guidelines

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

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

client | library

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

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

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

cout | guidelines | defects

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

client | library

cout | guidelines | defects

client | library

```
object_t func()
{
    object_t result = 5;
    return result;
}

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

    object_t x = func();
}
```

cout | guidelines | defects

client | library

```
object_t func()
{
    object_t result = 5;
    return result;
}

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

    object_t x = func();
}
```

cout

ctor

client | library

```
object_t func()
{
    object_t result = 5;
    return result;
}

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

}
```

cout | guidelines | defects

```
object_t func()
{
    object_t result = 5;
    return result;
}

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

    object_t x = 0;

    x = func();
}
```

client | library

```
object_t func()
{
    object_t result = 5;
    return result;
}

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

    object_t x = 0;

    x = func();
}
```

cout

```
ctor
ctor
copy
```

client | library

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t& operator=(const object_t& x)
    { object_t tmp(x); self_ = move(tmp.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

cout | guidelines | defects

client | library

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

cout | guidelines | defects

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t& operator=(object_t x) noexcept
    { self_ = move(x.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

client | library

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t& operator=(object_t x) noexcept
    { self_ = move(x.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(da                    on); }

        int data
```

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.

```
object_t func()
{
    object_t result = 5;
    return result;
}

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

    object_t x = 0;

    x = func();
}
```

client | library

```
object_t func()
{
    object_t result = 5;
    return result;
}

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

    object_t x = 0;

    x = func();
}
```

cout

```
ctor
ctor
```

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

client | library

```
int main()
{
    document_t document;
    document.reserve(5);

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

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

    draw(document, cout, 0);
}
```

cout | guidelines | defects

client | library

```
int main()
{
    document_t document;
    document.reserve(5);

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);
```

cout

```
ctor
ctor
ctor
ctor
copy
copy
copy
copy
copy
copy
copy
<document>
 3
 2
 1
 0
</document>
```

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }

    object_t& operator=(object_t x) noexcept
    { self_ = move(x.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t(object_t&& x) noexcept : self_(move(x.self_)) { }
    object_t& operator=(object_t x) noexcept
    { self_ = move(x.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }

    object_t& operator=(object_t x) noexcept
    { self_ = move(x.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t(object_t&&) noexcept = default;
    object_t& operator=(object_t x) noexcept
    { self_ = move(x.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

client | library

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t(object_t&&) noexcept = default;
    object_t& operator=(object_t x) noexcept
    { self_ = move(x.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw
        { draw(da
```

guidelines

- Provide a move constructor and move assignment to avoid copies and get fast permutations
  - Prior to C++11, provide a swap() function.
- Use "= default" when possible
- Include the expected exception specification to catch mistakes

client | library

```
int main()
{
    document_t document;
    document.reserve(5);

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

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

    draw(document, cout, 0);
}
```

cout | guidelines | defects

client | library

```
int main()
{
    document_t document;
    document.reserve(5);

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

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

    draw(document, cout, 0);
}
```

cout

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

```
struct some_t {
    object_t member_;
};

some_t func() { return { 5 }; }

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

    some_t x = { 0 };

    x = func();
}
```

client | library

```
struct some_t {
    object_t member_;
};

some_t func() { return { 5 }; }

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

    some_t x = { 0 };

    x = func();
}
```

cout

```
ctor
ctor
copy
```

client | library

```
struct some_t {
    object_t member_;
};

some_t func() { return { 5 }; }

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

    some_t x = { 0 };

    x = func();
}
```

guidelines

- 12.8.23 - "A defaulted copy/move assignment operator for class X is defined as deleted if X has:

...

— for the move assignment operator, a non-static data member or direct base class with a type that does not have a **move assignment operator** and is not trivially copyable..."

- Core Issue 1402

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t(object_t&&) noexcept = default;
    object_t& operator=(object_t x)
    { self_ = move(x.self_); return *this; }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

client | library

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_mo
        int_model                    : data_(x) { }
        void draw                    size_t position) const
```

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.
  - *However, because of a language defect, you must write a move assignment operator.*

```
struct some_t {
    object_t member_;
};

some_t func() { return { 5 }; }

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

    some_t x = { 0 };

    x = func();
}
```

client | library

```
struct some_t {
    object_t member_;
};

some_t func() { return { 5 }; }

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

    some_t x = { 0 };

    x = func();
}
```

cout

```
ctor
ctor
```

Keypoint

- Returning objects from functions, passing read-only arguments, and passing rvalues as sink arguments do not require copying

- Understanding this can greatly improve the efficiency of your application

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { cout << "copy" << endl; }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

client | library

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

cout | guidelines | defects

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

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

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(const int& x) : data_(x) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

```
class object_t {
  public:
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {

        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

```
class object_t {
  public:
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(int x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

```
class object_t {
  public:
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct int_model_t {
        int_model_t(int x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

client | library

```
public:
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
```

cout | guidelines | defects

client | library

```
public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

private:
    struct string_model_t {
        string_model_t(string x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        string data_;
    };
```

cout | guidelines | defects

```
public:
  object_t(string x) : self_(new string_model_t(move(x)))
  { }
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
```

```
  object_t(string x) : self_(new string_model_t(move(x)))
  { }
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
```

```
  { }
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
```

```
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
```

```
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
```

```
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };
```

```
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };

 unique_ptr<int_model_t> self_;
```

```
    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct string_model_t {
        string_model_t(string x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        string data_;
    };
    struct int_model_t {
        int_model_t(int x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<int_model_t> self_;
};
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };

  unique_ptr<int_model_t> self_;
};
```

```
    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct string_model_t {
        string_model_t(string x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        string data_;
    };
    struct int_model_t {
        int_model_t(int x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

};
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct string_model_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };

  unique_ptr<concept_t> self_;
};
```

```
    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct string_model_t {
        string_model_t(string x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        string data_;
    };
    struct int_model_t {
        int_model_t(int x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

  unique_ptr<concept_t> self_;
};
```

```
    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct string_model_t {
        string_model_t(string x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        string data_;
    };

    struct int_model_t {
        int_model_t(int x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

```
    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
    };
    struct string_model_t : concept_t {
        string_model_t(string x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        string data_;
    };
    struct int_model_t : concept_t {
        int_model_t(int x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };
```

client | library

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
```

cout | guidelines | defects

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
```

```
  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
```

```
  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
```

```
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
```

```
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
```

```
  { }
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t(string x) : self_(new string_model_t(move(x)))
  { }
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
public:
  object_t(string x) : self_(new string_model_t(move(x)))
  { }
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(new int_model_t(*x.self_))
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
```

```
class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    struct string_model_t : concept_t {
        string_model_t(string x) : data_(move(x)) { }
```

```
class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    struct string_model_t : concept_t {
```

```
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
```

```
void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
```

client | library

```
void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
```

cout | guidelines | defects

```
void draw(const string& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }
```

```
void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
```

```
void draw(const string& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(new int_model_t(*x.self_))
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
```

```
void draw(const string& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
```

```
void draw(const string& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
```

```
void draw(const string& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
```

client | library

```
{ out << string(position, ' ') << x << endl; }

void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
```

cout | guidelines | defects

```
void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
```

```
void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
```

```
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
```

```
class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    struct string_model_t : concept_t {
```

client | library

```
class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    struct string_model_t : concept_t {
        string_model_t(string x) : data_(move(x)) { }
```

cout | guidelines | defects

```
public:
  object_t(string x) : self_(new string_model_t(move(x)))
  { }
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
```

```
  object_t(string x) : self_(new string_model_t(move(x)))
  { }
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  { }
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t(int x) : self_(new int_model_t(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
```

```
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
```

```
  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
```

```
  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
```

```
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
```

```
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
```

```
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };
```

```
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };
```

```
  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };

 unique_ptr<concept_t> self_;
```

```
    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    struct string_model_t : concept_t {
        string_model_t(string x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        string data_;
    };
    struct int_model_t : concept_t {
        int_model_t(int x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<concept_t> self_;
};
```

```
  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;

      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }

      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }

      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };
```

```
  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      concept_t* copy_() const { return new int_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };
```

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(1);
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

cout | guidelines | defects

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);

    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

cout | guidelines | defects

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

cout | guidelines | defects

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

cout

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

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(2);
    document.emplace_back(3);

    draw(document, cout, 0);
}
```

guidelines

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

client | library

```
void draw(const string& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

void draw(const int& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    object_t(string x) : self_(new string_model_t(move(x)))
    { }
    object_t(int x) : self_(new int_model_t(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
```

cout | guidelines | defects

client | library

```
class object_t {
  public:
    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
```

cout | guidelines | defects

```
template <typename T>
void draw(const T& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
```

```
template <typename T>
void draw(const T& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
```

```
void draw(const T& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
```

```
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    struct string_model_t : concept_t {
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    struct string_model_t : concept_t {
        string_model_t(string x) : data_(move(x)) { }
```

```
public:
  template <typename T>
  object_t(T x) : self_(new model<T>(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
```

```
  template <typename T>
  object_t(T x) : self_(new model<T>(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
```

```
  object_t(T x) : self_(new model<T>(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
```

```
  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
```

```
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
```

```
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      concept_t* copy_() const { return new int_model_t(*this); }
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      concept_t* copy_() const { return new int_model_t(*this); }
      void draw_(ostream& out, size_t position) const
```

```
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      concept_t* copy_() const { return new int_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      concept_t* copy_() const { return new int_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      concept_t* copy_() const { return new int_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
```

```
  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      concept_t* copy_() const { return new int_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };
```

```
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      concept_t* copy_() const { return new int_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };
```

```
private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  struct string_model_t : concept_t {
      string_model_t(string x) : data_(move(x)) { }
      concept_t* copy_() const { return new string_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      string data_;
  };
  struct int_model_t : concept_t {
      int_model_t(int x) : data_(move(x)) { }
      concept_t* copy_() const { return new int_model_t(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      int data_;
  };

 unique_ptr<concept_t> self_;
```

```
  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    struct string_model_t : concept_t {
        string_model_t(string x) : data_(move(x)) { }
        concept_t* copy_() const { return new string_model_t(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        string data_;
    };
    struct int_model_t : concept_t {
        int_model_t(int x) : data_(move(x)) { }
        concept_t* copy_() const { return new int_model_t(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

   unique_ptr<concept_t> self_;
};
```

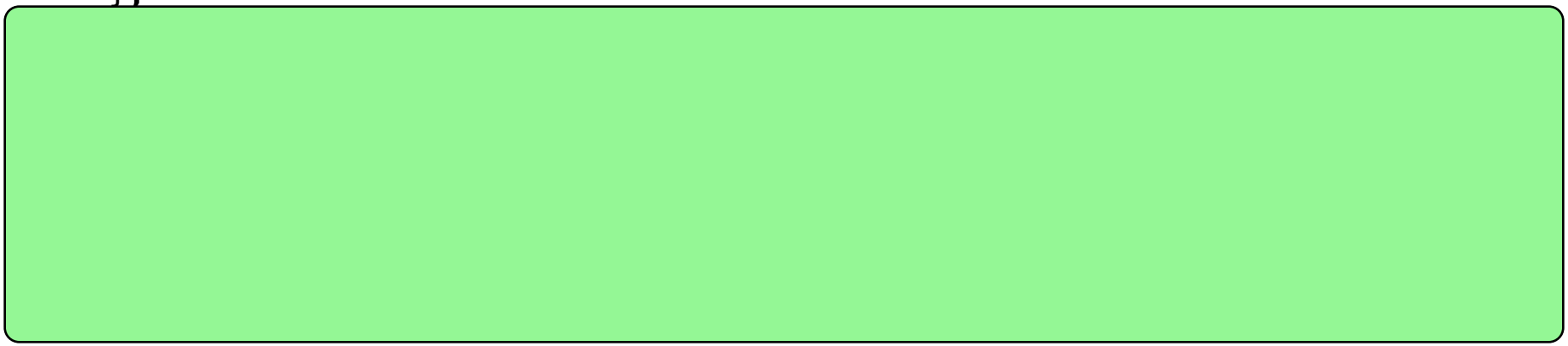
```
  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    struct string_model_t : concept_t {
        string_model_t(string x) : data_(move(x)) { }
        concept_t* copy_() const { return new string_model_t(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        string data_;
    };
    struct int_model_t : concept_t {
        int_model_t(int x) : data_(move(x)) { }
        concept_t* copy_() const { return new int_model_t(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        int data_;
    };

    unique_ptr<concept_t> self_;
};
```

```
  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
```



```
    unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
```

```
  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
```

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(2);

    draw(document, cout, 0);
}
```

cout | guidelines | defects

```
class my_class_t {
    /* ... */
};

void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(2);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

client | library

```
class my_class_t {
    /* ... */
};

void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(2);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

cout

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

client | library

```
class my_class_t {
    /* ... */
};

void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(2);
    document.emplace_back(my_class_t());

    draw(document, cout, 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 payed when needed.
  - Polymorphic types are used like any other types, including built-in types.

```
class my_class_t {
    /* ... */
};

void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));

    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

```
class my_class_t {
    /* ... */
};

void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

client | library

```
class my_class_t {
    /* ... */
};

void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());
```

cout

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

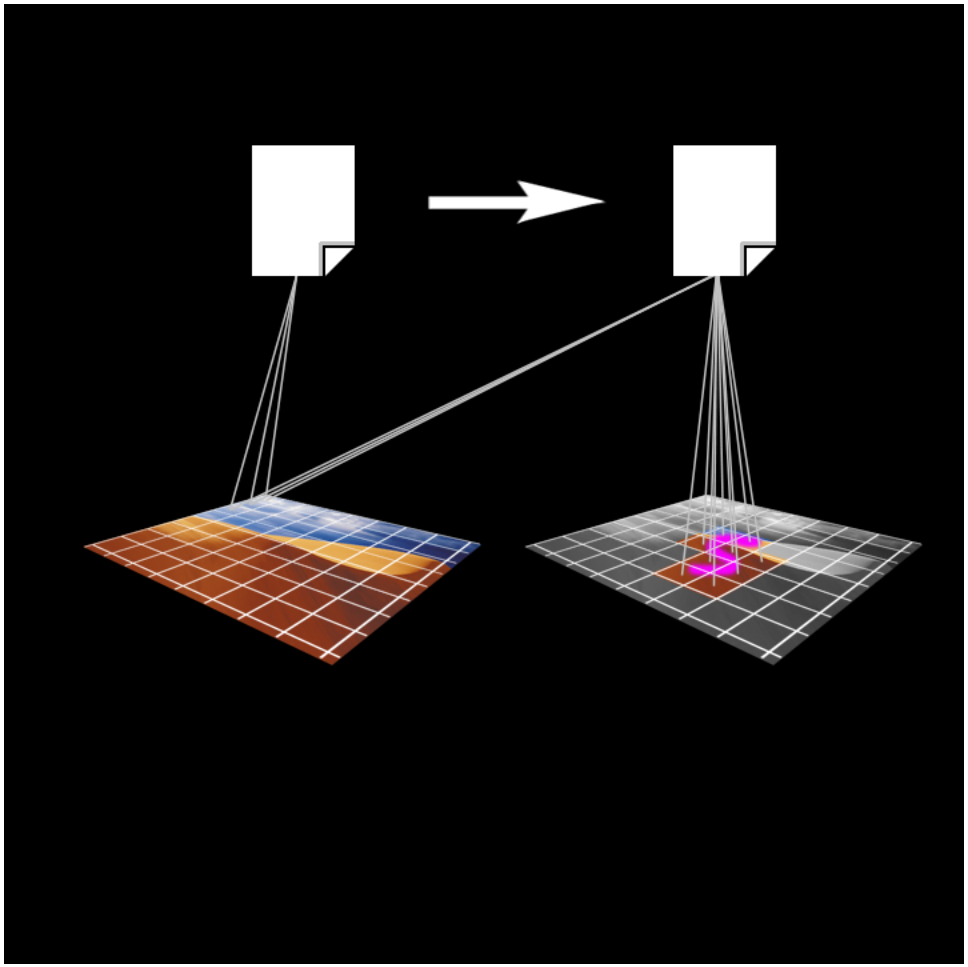
Polymorphic Use

- Shifting polymorphism from type to use allows for greater reuse and fewer dependencies
- Using regular semantics for the common basis operations, copy, assignment, and move helps to reduce shared objects
- Regular types promote interoperability of software components, increases productivity as well as quality, security, and performance
- There is no performance penalty to using regular semantics, and often times there are performance benefits from a decreased use of the heap

Demo

Photoshop History

Photoshop History



```
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

  unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

client | library

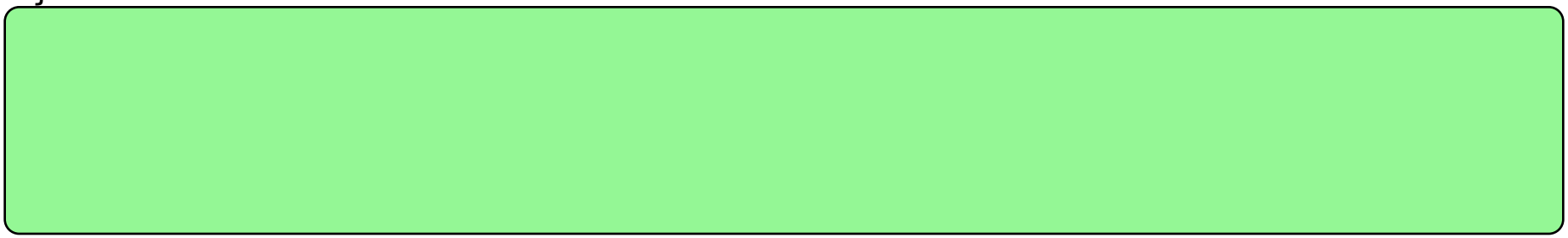
```
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```



cout | guidelines | defects

```
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

```
using history_t = vector<document_t>;

void commit(history_t& x) { assert(x.size()); x.push_back(x.back()); }
void undo(history_t& x) { assert(x.size()); x.pop_back(); }
document_t& current(history_t& x) { assert(x.size()); return x.back(); }
```

```
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;

void commit(history_t& x) { assert(x.size()); x.push_back(x.back()); }
void undo(history_t& x) { assert(x.size()); x.pop_back(); }
document_t& current(history_t& x) { assert(x.size()); return x.back(); }
```

```
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;

void commit(history_t& x) { assert(x.size()); x.push_back(x.back()); }
void undo(history_t& x) { assert(x.size()); x.pop_back(); }
```

```
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

  unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;

void commit(history_t& x) { assert(x.size()); x.push_back(x.back()); }
```

```
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

  unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;
```

```
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;
```

```
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

```
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

```
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
```

```
  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
```

```
  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
```

```
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
```

```
    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
```

```
    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;
```

```
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};

using document_t = vector<object_t>;
```

```
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};
```

```
    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   unique_ptr<concept_t> self_;
};
```

```
  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  template <typename T>
  struct model : concept_t {
      model(T x) : data_(move(x)) { }
      concept_t* copy_() const { return new model(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      T data_;
  };

 unique_ptr<concept_t> self_;
```

```
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  template <typename T>
  struct model : concept_t {
      model(T x) : data_(move(x)) { }
      concept_t* copy_() const { return new model(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      T data_;
  };
```

```
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  template <typename T>
  struct model : concept_t {
      model(T x) : data_(move(x)) { }
      concept_t* copy_() const { return new model(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      T data_;
  };
```

```
  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  template <typename T>
  struct model : concept_t {
      model(T x) : data_(move(x)) { }
      concept_t* copy_() const { return new model(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }

      T data_;
```

```
  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  template <typename T>
  struct model : concept_t {
      model(T x) : data_(move(x)) { }
      concept_t* copy_() const { return new model(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  template <typename T>
  struct model : concept_t {
      model(T x) : data_(move(x)) { }
      concept_t* copy_() const { return new model(*this); }
      void draw_(ostream& out, size_t position) const
      { draw(data_, out, position); }
```

```
  object_t(T x) : self_(new model<T>(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  template <typename T>
  struct model : concept_t {
      model(T x) : data_(move(x)) { }
      concept_t* copy_() const { return new model(*this); }
      void draw_(ostream& out, size_t position) const
```

```
  template <typename T>
  object_t(T x) : self_(new model<T>(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  template <typename T>
  struct model : concept_t {
      model(T x) : data_(move(x)) { }
      concept_t* copy_() const { return new model(*this); }
```

```
public:
  template <typename T>
  object_t(T x) : self_(new model<T>(move(x)))
  { }

  object_t(const object_t& x) : self_(x.self_->copy_())
  { }
  object_t(object_t&&) noexcept = default;

  object_t& operator=(const object_t& x)
  { object_t tmp(x); *this = move(tmp); return *this; }
  object_t& operator=(object_t&&) noexcept = default;

  friend void draw(const object_t& x, ostream& out, size_t position)
  { x.self_->draw_(out, position); }

private:
  struct concept_t {
      virtual ~concept_t() = default;
      virtual concept_t* copy_() const = 0;
      virtual void draw_(ostream&, size_t) const = 0;
  };
  template <typename T>
  struct model : concept_t {
      model(T x) : data_(move(x)) { }
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
```

client | library

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())

    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
```

cout | guidelines | defects

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { cout << "copy" << endl; }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
```

```
class my_class_t {
    /* ... */
};

void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

```
    /* ... */
};

void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

```
};

void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

```
void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

client | library

```
void draw(const my_class_t&, ostream& out, size_t position)
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

cout | guidelines | defects

client | library

```
{ out << string(position, ' ') << "my_class_t" << endl; }

int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

cout | guidelines | defects

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

cout | guidelines | defects

client | library

```
int main()
{
    document_t document;

    document.emplace_back(0);
    document.emplace_back(string("Hello!"));
    document.emplace_back(document);
    document.emplace_back(my_class_t());

    draw(document, cout, 0);
}
```

cout | guidelines | defects

client | library

```
int main()
{
}
```

cout | guidelines | defects

client | library

```
int main()
{
    history_t h(1);

    current(h).emplace_back(0);
    current(h).emplace_back(string("Hello!"));

    draw(current(h), cout, 0);
    cout << "------------------------" << endl;

    commit(h);

    current(h)[0] = 42.5;
    current(h)[1] = string("World");
    current(h).emplace_back(current(h));
    current(h).emplace_back(my_class_t());

    draw(current(h), cout, 0);
    cout << "------------------------" << endl;

    undo(h);

    draw(current(h), cout, 0);
}
```

cout | guidelines | defects

client | library

```
int main()
{
    history_t h(1);

    current(h).emplace_back(0);
    current(h).emplace_back(string("Hello!"));

    draw(current(h), cout, 0);
    cout << "------------------------" << endl;

    commit(h);

    current(h)[0] = 42.5;
    current(h)[1] = string("World");
    current(h).emplace_back(current(h));
    current(h).emplace_back(my_class_t());

    draw(current(h), cout, 0);
    cout << "------------------------" << endl;

    undo(h);

    draw(current(h), cout, 0);
}
```

cout | guidelines | defects

client

library

cout

```
<document>
 0
 Hello!
</document>
------------------------
copy
copy
copy
copy
<document>
 42.5
 World
 <document>
  42.5
  World
 </document>
 my_class_t
</document>
------------------------
<document>
 0
 Hello!
</document>
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    object_t(const object_t& x) : self_(x.self_->copy_())
    { cout << "copy" << endl; }
    object_t(object_t&&) noexcept = default;

    object_t& operator=(const object_t& x)
    { object_t tmp(x); *this = move(tmp); return *this; }
    object_t& operator=(object_t&&) noexcept = default;

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
```

client | library

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };
```

cout | guidelines | defects

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual concept_t* copy_() const = 0;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        concept_t* copy_() const { return new model(*this); }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

    unique_ptr<concept_t> self_;
};
```

client | library

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

};
```

cout | guidelines | defects

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

    shared_ptr<concept_t> self_;
};
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

};
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

    shared_ptr<const concept_t> self_;
};
```

client | library

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw
                                    size_t position) const
        { draw(da                   on); }
```

guidelines

- A shared pointer to an immutable (const) object has value semantics.
  - This is why passing arguments by const & works.
- Observation: Mutable polymorphic objects are the exception.
- Copy-on-write can be obtained using shared_ptr::unique().

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(new model<T>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};
```

```
class object_t {
  public:
    template <typename T>
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(make_shared<model<T>>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};
```

client | library

```
int main()
{
    history_t h(1);

    current(h).emplace_back(0);
    current(h).emplace_back(string("Hello!"));

    draw(current(h), cout, 0);
    cout << "------------------------" << endl;

    commit(h);

    current(h)[0] = 42.5;
    current(h)[1] = string("World");
    current(h).emplace_back(current(h));
    current(h).emplace_back(my_class_t());

    draw(current(h), cout, 0);
    cout << "------------------------" << endl;

    undo(h);

    draw(current(h), cout, 0);
}
```

cout | guidelines | defects

client

library

cout

```
<document>
 0
 Hello!
</document>
------------------------
<document>
 42.5
 World
 <document>
  42.5
  World
 </document>
 my_class_t
</document>
------------------------
<document>
 0
 Hello!
</document>
```

client | library

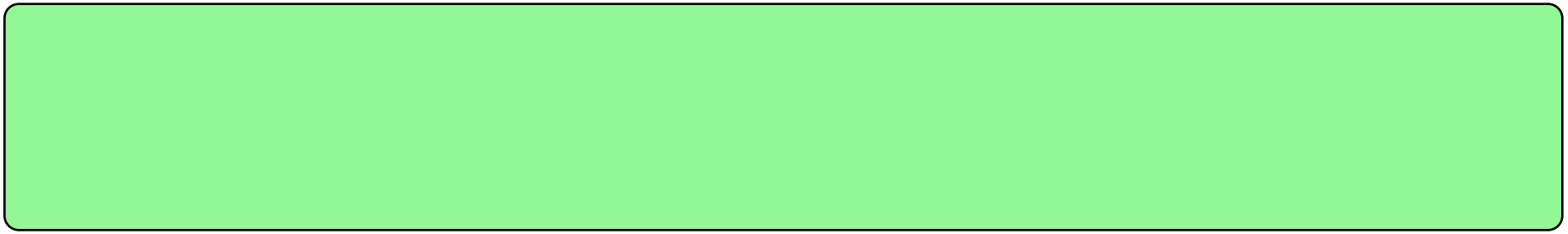
```
int main()
{
    history_t h(1);

    current(h).emplace_back(0);
    current(h).emplace_back(string("Hello!"));

    draw(current(h), cout, 0);
    cout << "-------------------------" << endl;

    commit(h);

    current(h)[0] = 42.5;
```



```
    current(h)[1] = string("World");
    current(h).emplace_back(current(h));
    current(h).emplace_back(my_class_t());
```

cout | guidelines | defects

client | library

```
int main()
{
    history_t h(1);

    current(h).emplace_back(0);
    current(h).emplace_back(string("Hello!"));

    draw(current(h), cout, 0);
    cout << "------------------------" << endl;

    commit(h);

    current(h)[0] = 42.5;

    auto document = current(h);
    auto saving = async([=]() {
        this_thread::sleep_for(chrono::seconds(3));
        cout << "-------- 'save' --------" << endl;
        draw(document, cout, 0);
    });

    current(h)[1] = string("World");
    current(h).emplace_back(current(h));
    current(h).emplace_back(my_class_t());
```

cout | guidelines | defects

```
<document>
 0
 Hello!
</document>
------------------------
<document>
 42.5
 World
 <document>
  42.5
  World
 </document>
 my_class_t
</document>
------------------------
<document>
 0
 Hello!
</document>
-------- 'save' --------
<document>
 42.5
 Hello!
</document>
```

Compared To Inheritance Based Design

- More flexible
  - Non-intrusive design doesn't require class wrappers
- More efficient
  - Polymorphism is only paid for when needed
- Less error prone
  - Client doesn't do any heap allocation, worry about object ownership or lifetimes
  - Exception safe
- Thread safe

```
template <typename T>
void draw(const T& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(make_shared<model<T>>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
```

client | library

```
void draw(const T& x, ostream& out, size_t position)
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(make_shared<model<T>>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };
```

cout | guidelines | defects

```
{ out << string(position, ' ') << x << endl; }

class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(make_shared<model<T>>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(make_shared<model<T>>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
```

```
class object_t {
  public:
    template <typename T>
    object_t(T x) : self_(make_shared<model<T>>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};
```

```
  public:
    template <typename T>
    object_t(T x) : self_(make_shared<model<T>>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};
```

```
    template <typename T>
    object_t(T x) : self_(make_shared<model<T>>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;
```

```
    object_t(T x) : self_(make_shared<model<T>>(move(x)))
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;
```

```
    { }

    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
```

```
    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
```

```
    friend void draw(const object_t& x, ostream& out, size_t position)
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
```

```
    { x.self_->draw_(out, position); }

  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
```

```
  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
```

```
  private:
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

```
    struct concept_t {
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}
```

```
        virtual ~concept_t() = default;
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;
```

```
        virtual void draw_(ostream&, size_t) const = 0;
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;
```

```
    };
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;

void commit(history_t& x) { assert(x.size()); x.push_back(x.back()); }
```

client | library

```
    template <typename T>
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;

void commit(history_t& x) { assert(x.size()); x.push_back(x.back()); }
void undo(history_t& x) { assert(x.size()); x.pop_back(); }
```

cout | guidelines | defects

```
    struct model : concept_t {
        model(T x) : data_(move(x)) { }
        void draw_(ostream& out, size_t position) const
        { draw(data_, out, position); }

        T data_;
    };

   shared_ptr<const concept_t> self_;
};

using document_t = vector<object_t>;

void draw(const document_t& x, ostream& out, size_t position)
{
    out << string(position, ' ') << "<document>" << endl;
    for (const auto& e: x) draw(e, out, position + 2);
    out << string(position, ' ') << "</document>" << endl;
}

using history_t = vector<document_t>;

void commit(history_t& x) { assert(x.size()); x.push_back(x.back()); }
void undo(history_t& x) { assert(x.size()); x.pop_back(); }
document_t& current(history_t& x) { assert(x.size()); return x.back(); }
```

Concluding Remarks

- As we increasingly move to heavily threaded systems using promises, reactive programming, and task queues, value semantics becomes critical to avoid locking and to reason about code

- It is my hope that the language (and libraries) will evolve to make creating polymorphic types with value semantics easier

- Thanks to Alex Stepanov, Howard Hinnant, and Dave Abrahams


Adobe