C++ Class Natures
in their Canonical Form
and how to find them

Speed Version

CPPCon 2019
Prof. Peter Sommerlad
@PeterSommerlad
peter.cpp@sommerlad.ch
What classes we write...

OR

virtual is overrated

MeetingC++ 2019
Prof. Peter Sommerlad
@PeterSommerlad
peter.cpp@sommerlad.ch
Remember: What Special Member Functions Do You Get?

<table>
<thead>
<tr>
<th>What you write</th>
<th>default constructor</th>
<th>destructor</th>
<th>copy constructor</th>
<th>copy assignment</th>
<th>move constructor</th>
<th>move assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>nothing</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
</tr>
<tr>
<td>any constructor</td>
<td>not declared</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
</tr>
<tr>
<td>default constructor</td>
<td>user declared</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
</tr>
<tr>
<td>destructor</td>
<td>defaulted</td>
<td>user declared</td>
<td>defaulted (!)</td>
<td>defaulted (!)</td>
<td>not declared</td>
<td>not declared</td>
</tr>
<tr>
<td>copy constructor</td>
<td>not declared</td>
<td>defaulted</td>
<td>user declared</td>
<td>defaulted (!)</td>
<td>not declared</td>
<td>not declared</td>
</tr>
<tr>
<td>copy assignment</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted (!)</td>
<td>user declared</td>
<td>not declared</td>
<td>not declared</td>
</tr>
<tr>
<td>move constructor</td>
<td>not declared</td>
<td>defaulted</td>
<td>deleted</td>
<td>deleted</td>
<td>user declared</td>
<td>not declared</td>
</tr>
<tr>
<td>move assignment</td>
<td>defaulted</td>
<td>defaulted</td>
<td>deleted</td>
<td>deleted</td>
<td>not declared</td>
<td>user declared</td>
</tr>
</tbody>
</table>

Howard Hinnant's Table: [https://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf](https://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf)

Note: Getting the defaulted special members denoted with a (!) is an unfixable bug in the standard.
These are most of the kinds of classes we (should) write (aka class natures)

- **value**
- **polymorphic**
- **relationship**
- **managing**
struct AssignDangle{
    int i;
}

A proposal to add l-value member function qualifier

Author: Bronek Kozicki <brok@rubikon.pl>
Doc. no.: N1784=05-0044
Date: 2005-04-15
Working Group: Evolution

// ... as in motivation example
class M
{
    struct K
    {
        int i;
        K(int i) : i (i) {}
        K& operator=(const K& rh) & // l-value qualifier
        {
            i = rh.i;
            return *this;
        }
    };
    // ... as in motivation example
A new Rule of Six (if you can not apply Rule of Zero for Value Types)

class Good {
public:
    Good() = default;
    ~Good() = default;
    Good(const Good &other) = default;
    Good(Good &&other) noexcept = default;
    Good& operator=(const Good &other) & = default;
    Good& operator=(Good &&other) & noexcept = default;
};

Is that better?
class OOBase {
public:
    virtual ~OOBase() = default;
    OOBase& operator=(OOBase &&other) = delete;
};

DesDeMovA
    Rule of if
    Destructor defined
    Deleted
    Move Assignment

Copying is prohibited, default construction remains
OO-Polymorphism is highly overrated! Think hard, if you need "virtual"

value $
polymorphic
relationship
managing

variant<...>
type erasure

requires heap allocation or s-o-o
Scoped managers are non-copyable

Scoped Manager

```cpp
class Scoped {
    Resource resource;

public:
    Scoped(); // acquire resource
    ~Scoped(); // release resource
    Scoped& operator=(Scoped &&other) = delete;
};
```

DesDeMovA
- Rule of if
  - Destructor defined
  - Deleted
  - Move Assignment
Unique managers require deliberate “moved-from” state

```cpp
class Unique {
    std::optional<Resource> resource;
    void release() noexcept; // release resource

public:
    Unique() = default;
    Unique(Params p); // acquire resource
    ~Unique() noexcept;
    Unique& operator=(Unique &&other) & noexcept;
    Unique(Unique &&other) noexcept;
};

void Unique::release() noexcept {
    if (resource) {
        // really release resource here
        resource.reset();
    }
}

Unique::~Unique() noexcept {
    this->release();
}

Unique::Unique(Unique &&other) noexcept:
    resource{std::move(other.resource)}{
    other.resource.reset();
}

Unique& Unique::operator=(Unique &&other) & noexcept {
    if (this != &other) {
        this->release();
        std::swap(this->resource, other.resource);
    }
    return *this;
}
```
Sharing mutable managers is hard, think thrice!

Shared Manager

use

shared_ptr<resource>

and concurrency control

if at all
Implement your own regular manager, like Vector?

Manager as Value

```cpp
class MValue {
public:
    MValue() = default;
    ~MValue();
    MValue(MValue &&other) noexcept ;
    MValue& operator=(MValue &&other) & noexcept;
    MValue(const MValue &other);
    MValue& operator=(const MValue &other) &;
};
```

💰 Hard, but rewarding!
### Natures and Canonical Form

<table>
<thead>
<tr>
<th></th>
<th>Some constructor</th>
<th>default constructor</th>
<th>destructor</th>
<th>copy constructor</th>
<th>copy assignment</th>
<th>move constructor</th>
<th>move assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>yes</td>
<td>none / =default</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
<td>defaulted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Relationship</td>
<td>yes</td>
<td>none / =default</td>
<td>defaulted</td>
<td>defaulted/ deleted</td>
<td>defaulted/ deleted</td>
<td>defaulted/ deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scope</td>
<td>typical</td>
<td>none</td>
<td>implemented</td>
<td>deleted</td>
<td>deleted</td>
<td>deleted</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>yes</td>
<td>defined / =default</td>
<td>implemented</td>
<td>deleted</td>
<td>deleted</td>
<td>implemented</td>
</tr>
<tr>
<td></td>
<td>OO - Base</td>
<td>protected</td>
<td>protected</td>
<td>virtual =default</td>
<td>deleted</td>
<td>deleted</td>
<td>deleted</td>
</tr>
</tbody>
</table>

- **Value**: Value
- **Relationship**: Relationship
- **Scoped**: Scoped
- **Unique**: Unique
- **OO - Base**: OO - Base

- **Some constructor**: none
- **default constructor**: none
- **destructor**: defaulted
- **copy constructor**: defaulted
- **copy assignment**: defaulted
- **move constructor**: defaulted
- **move assignment**: defaulted