Do you Remember: What Special Member Functions Do You Get?

What you get

What you write	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted (!)	defaulted (!)	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted (!)	not declared	not declared
copy assignment	defaulted	defaulted	defaulted (!)	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Table: https://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf
Note: Getting the defaulted special members denoted with a (!) is a bug in the standard.

Sane and less sane combinations

	Some constructor	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Aggregates	none	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
Simple Values	yes	none / =default	defaulted	defaulted	defaulted	defaulted	defaulted
Scope Manager	typical	none / =default	implemented	deleted	deleted	deleted	=delete 🙋
Unique Manager	typical	defined / =default	implemented	deleted	deleted	implemented	implemented
Value Manager	yes	defined / =default	implemented	implemented	implemented	implemented	implemented
OO - Base	may be	may be	=default virtual!	deleted	deleted	deleted	=delete 🙋
OO & Value type erasure*	yes	no	Expert Level - =default	Expert Level Implementation	Expert Level Implementation	Expert Level Implementation	Expert Level Implementation

* see Sean Parent's talks and slides: https://sean-parent.stlab.cc/papers-and-presentations/#better-code-runtime-polymorphism

Making a class T non-copyable: T& operator=(T&&) noexcept=delete;

What you get

What you write	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
any constructor	not declared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted (!)	defaulted (!)	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted (!)	not declared	not declared
copy assignment	defaulted	defaulted	defaulted (!)	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Table: https://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf
Note: Getting the defaulted special members denoted with a (!) is a bug in the standard.

Rule of DesDeMovA: T& operator=(T&&) noexcept=delete;

DesDeMovA

Rule of if
Destructor defined
Deleted
Move **A**ssigment

minimum amount of code to achieve desired non-copyable

What you write

	default constructor				move constructor	move assignment
nothing	defaulted			ulted	defaulted	defaulted
any constructor	not declared					
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted (!)	defaulted (!)	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted (!)	not declared	not declared
copy assignment	defaulted	defaulted	defaulted (!)	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Table: https://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf
Note: Getting the defaulted special members denoted with a (!) is a bug in the standard.

- **Common to Managing types**
  - define "interesting" destructor: `~manager() { /* clean up stuff */}`
- **0: scope - locally usable SBRM (e.g., `std::lock_guard`)**
  - Rule of **DesDeMovA**: `manager& operator=(manager &&) noexcept=delete;`
  - No movability implies also no copyability
  - C++17: can still return from factory if needed
- **1: unique - move-only type (e.g., `std::unique_ptr`)**
  - requires a **sane moved-from state** for transfer of ownership, copy operations implicitly deleted
- **N: value type (e.g., `std::vector`)**
  - requires duplicatable resource (aka memory)

Rule of DesDeMovA: T& operator=(T&&) noexcept=delete;

DesDeMovA

Rule of if
Destructor defined
Deleted
Move **A**ssigment

minimum amount of code to achieve desired non-copyable

What you write

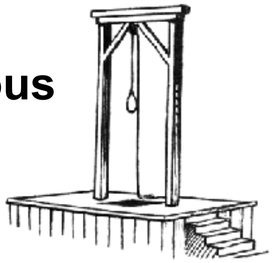
	default constructor				move constructor	move assignment
nothing	defaulted			ulted	defaulted	defaulted
any constructor	not declared					
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
destructor	defaulted	user declared	defaulted (!)	defaulted (!)	not declared	not declared
copy constructor	not declared	defaulted	user declared	defaulted (!)	not declared	not declared
copy assignment	defaulted	defaulted	defaulted (!)	user declared	not declared	not declared
move constructor	not declared	defaulted	deleted	deleted	user declared	not declared
move assignment	defaulted	defaulted	deleted	deleted	not declared	user declared

Howard Hinnant's Table: https://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf
Note: Getting the defaulted special members denoted with a (!) is a bug in the standard.

Influence of Member Variable Types on Special Member Functions

Member Varible Kind	Some constructor	default constructor	destructor	copy constructor	copy assignment	move constructor	move assignment
Value	none	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
T&[2]	yes	=delete	defaulted	defaulted	=delete[1]	defaulted	=delete[1]
Manager: [2] Scope	typical	none	defaulted	deleted	deleted	deleted	deleted 🙋
Manager: [2] Unique	typical	defined / =default	defaulted	deleted	deleted	defaulted	defaulted 🙋
Pot. Dangling[2,3]	typical	defined / =default	defaulted	defaulted	defaulted	defaulted	defaulted

- **1 - remedy through using std::reference_wrapper<T> instead**
- **2 - "contagious": your class becomes the same without further means**
- **3 - Regular Potentially Dangling Members make using your class type dangerous**



- **Rule of Zero**
  - for value types, for types with managing members
- **Rule of DesDeMovA**
  - for OO base classes, for SBRM classes

- **Adapted Rule of Three (destructor and move operations)**
  - for unique managing types define move operations, think of a sane moved-from state
- **Rule of Five**
  - for expert-level managing types (Containers like vector, Type Erasure, others)
- **Avoid members of potentially dangling types, otherwise**