Special Member Function

Rule of Three, Five, Zero and Five Defaults

Terminology

- “deleted” - generated or declared with = delete; - deleted members participate in overload resolution
- “not declared” - do **not** generated and do **not** participate in overload resolution.
- “defaulted” - generated by compiler or declared with = default;
- “disables” - either deleting or not declaring something
- “copy/move operation” - copy/move assignment and construction
- “user declared” - can be deleted, defaulted or user-defined

What are the special member functions?

- The special member functions are the ones that C++ is willing to generate on its own.
- C++98 has four such functions: **the default constructor, the destructor, the copy constructor,** and **the copy assignment operator**.
- C++11 adds two more - **the move constructor** and **the move assignment operator.**

What are the special member functions?

default constructor	X();
destructor	~X();
copy constructor	X(X const&);
copy assignment	X& operator=(X const&);
move constructor	X(X&&);
move assignment	X& operator=(X&&);

What are the special member functions?

- These functions are generated only if they’re needed, i.e., if some code uses them
- only if something illegal does not happen
- If copy/move operations are generated they perform “memberwise copies/moves” on the non-static data members of the class.
- They are inline and public. Destructor is virtual if base class destructor is virtual

Default constructor

- Generated only if the class contains no user-declared constructors
- not declared if *any* other constructor is declared including move and copy constructors

What declaring a default constructor disables?

- Nothing
- Declaring a default constructor does not imply resource ownership

Destructor

- inline public member
- virtual only if a base class destructor is virtual
- =delete; if has a non-static or has a direct or virtual base class that can not be destructed (has deleted or inaccessible destructor)

What declaring the destructor disables?

- **move constructor** and **move assignment** are not declared
- **copy constructor** and **copy assignment** are declared but it's deprecated

Copy operations

- memberwise copy construction/assignment of non-static data members.
- Generated only if the class lacks a user declared the other copy operation.
- Deleted if the class declares a **move** operation
- Generation of this functions in a class with a user-declared copy operation operator or destructor is deprecated.

What declaring the copy operations disables?

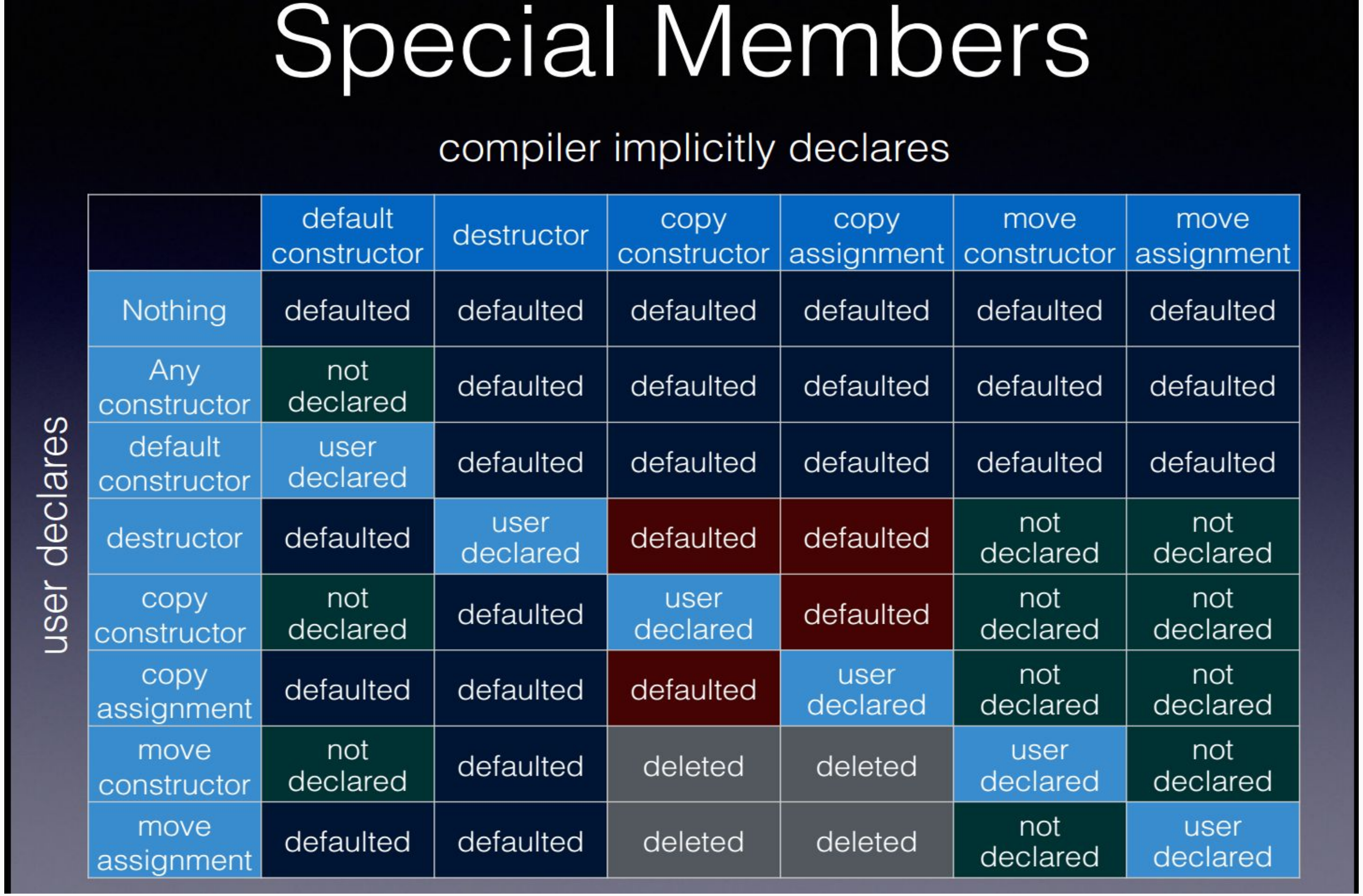
- default constructor is not declared
- the other copy operation is declared but it's deprecated
- move operations are not declared and if requested will be replaced with copy operations

Move operations

- Generated only if the class contains **no** user declared
  - copy operations
  - move operations
  - destructor

What declaring a move operation disables?

- deletes copy operations
- the other move operation is not declared



Special Members

compiler implicitly declares

user declares

	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

http://accu.org/content/conf2014/Howard_Hinnant_Accu_2014.pdf

Rule of Three (C++98/03)

If a class requires a user-defined destructor, a user-defined copy constructor, or a user-defined copy assignment operator it almost certainly requires all three.

Rule of Five (C++11)

If a class requires destructor, move or copy operation it almost certainly requires all five of them.

Rule of Zero

Classes that have custom destructors, copy/move constructors or copy/move assignment operators should deal exclusively with ownership.

Other classes should not have custom destructors, copy/move constructors or copy/move assignment operators.

Rule of Zero

Rule of five defaults

Always declare destructor, move and copy operations as =default;

A Concern about the Rule of Zero

Special Member Function

Thank you