Special Member Functions in C++

Default constructor , Copy constructor, Copy assignment operator
Move constructor (since C++ 11), Move assignment operator (since C++ 11), Destructor

In many cases such as in *The Rule of Five* we talk about just **five** of these operations, because the default constructor is different in terms of semantics to the other **five** (or **three** before C++11) operations. The other **five** operations are usually not declared and have more complex dependencies.

forces

user declaration of	default constructor	**copy constructor**	**copy assignment**	**move constructor**	**move assignment**	**destructor**
nothing	defaulted	defaulted	defaulted	defaulted	defaulted	defaulted
any constructor	undeclared	defaulted	defaulted	defaulted	defaulted	defaulted
default constructor	user declared	defaulted	defaulted	defaulted	defaulted	defaulted
copy constructor	undeclared	user declared	defaulted	undeclared (fallback enabled)	undeclared (fallback enabled)	defaulted
copy assignment	defaulted	defaulted	user declared	undeclared (fallback enabled)	undeclared (fallback enabled)	defaulted
move constructor	undeclared	deleted	deleted	user declared	undeclared (fallback disabled)	defaulted
move assignment	defaulted	deleted	deleted	undeclared (fallback disabled)	user declared	defaulted
destructor	defaulted	defaulted	defaulted	undeclared (fallback enabled)	undeclared (fallback enabled)	user declared

Special Member Functions in C++

Default constructor , Copy constructor, Copy assignment operator
Move constructor (since C++ 11), Move assignment operator (since C++ 11), Destructor

- A default constructor is declared automatically if no other constructor is user-declared.
- The special copy member functions and the destructor disable move support. The automatic generation of special move member functions is disabled (unless the moving operations are also declared). However, still a request to move an object usually works because a copy member functions are used as a fallback (unless the special move member functions are explicitly deleted).
- The special move member functions disable the normal copying and assignment. The copying and other moving special member functions are deleted so that you can only move (assign) but not copy (assign) an object (unless the other operations are also declared).

Special Member Functions in C++

The Rule of Three, The Rule of Five, The Rule of Zero

1. The Rule of Three

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 of the three.

Because C++ copies and copy-assigns objects of user-defined types in various situations (passing/returning by value, manipulating container) these special member functions will be called, if accessible, and if they are not user-defined they are implicitly defined by the compiler.

The implicitly defined special member functions are typically incorrect if the class manages a resource whose handle is an object of non-class type (raw pointer, POSIX file descriptor, etc.) whose destructor does nothing and copy constructor/assignment operator performs “shallow” copy (copy the value of the handle without duplicating the underlying resource).
Classes that manage non-copyable resources through copyable handles may have to declare copy assignment and copy constructor private and not provide their definitions or define them as deleted. This is another application of the rule of three: deleting one and leaving the other to be implicitly-defined will most likely result in errors.

Special Member Functions in C++

The Rule of Three, The Rule of Five, The Rule of Zero

2. The Rule of Five

Because the presence of a user-defined destructor, copy-constructor, or copy-assignment operator prevents the implicit definition of the move constructor and the move-assignment operator, any class for which move semantics are desirable has to declare all five special member functions.
Unlike in The Rule of Three failing to provide move constructor and move assignment is usually not an error, but a missed optimization opportunity.

3. The Rule of Zero

Classes that have custom destructors, copy/move constructors or copy/move assignment operators should deal exclusively with ownership (which follows from the Single Responsibility Principle). Other classes should not have custom destructors, copy/move constructors or copy/move assignment operators. The rule also appears in the C++ Core Guidelines as C.20: if you can avoid defining default operations do.
When a base class is intended for polymorphic use, its destructor may have to be declared public and virtual. This blocks implicit moves (and deprecates implicit copies), and so the special member functions have to be declared as defaulted.
However, this makes the class prone to slicing, which is why polymorphic classes often define copy as deleted (see C.67: A polymorphic class should suppress copying in C++ Core Guidelines), which leads to the following generic wording for the Rule Of Five:
C21: If you define or =delete any default operation define or =delete them all