dXXXX - Safe and Sane C++ Classes

Peter Sommerlad

2018-07-10

Document Number:	dXXXX
Date:	2018-07-10
Project:	Programming Language C++ / Programming Language Vulnerabilities C++
Audience:	SG12 / ISO SC22 WG23

1 Introduction

A guiding principle should be "easy to use, hard to misuse". An additional principle should be clear separation of concern and clear separation of purpose. Mixing categories, like values and objects, in a single type can be doomed, or at least hard-to-get-right expert territory.

Value Types - Regular types, built-in

Empty Classes - Tags, (CRTP-)super-Mix-ins and sub-classing-Adapters

Managing Classes - RAII, Manager objects, Containers

OO-Polymorphic Classes - better non-copyable!

Referring Types - References, (smart) pointers, string_view, span

combinations - expert territory, e.g., manager as value type, usage should be clear

less sane variations - e.g., copyable OO hierarchy types (deep or shallow), referring types as members, subclassing adapters

TODO: Howard's table

rule of zero and related

context of audience.

forces to be covered.

Properties to be discussed.

Potential dangers.

Table 1 — Howard Hinnant's special member functions table

	What the compiler provides for class X						
user declares	`X()`	`~X()`	`X(X const&)`	`=(X const&)`	`X(X &&)`	`=(X &&)`	OK?
nothing	`=default`	`=default`	`=default`	`=default`	`=default`	`=default`	OK
`X(T)`	not decl	`=default`	`=default`	`=default`	`=default`	`=default`	OK
`X()`	*declared*	`=default`	`=default`	`=default`	`=default`	`=default`	(OK)
`~X()`	`=default`	*declared*	`=default`	`=default`	not decl	not decl	**BAD**
`X(X const&)`	not decl	`=default`	*declared*	`=default`	not decl	not decl	**BAD**
`=(X const&)`	`=default`	`=default`	`=default`	*declared*	not decl	not decl	**BAD**
`X(X&&)`	not decl	`=default`	`=delete`	`=delete`	*declared*	not decl	**BAD**
`=(X&&)`	`=default`	`=default`	`=delete`	`=delete`	not decl	*declared*	(BAD)

Table 2 — Safe and Sane combinations of Special Member Functions (TODO)

	declared or provided						
type category	`X()`	`~X()`	`X(X const&)`	`=(X const&)`	`X(X &&)`	`=(X &&)`	
value/aggregate	`=default`	`=default`	`=default`	`=default`	`=default`	`=default`	OK
value	not decl	`=default`	`=default`	`=default`	`=default`	`=default`	OK
`X()`	*declared*	`=default`	`=default`	`=default`	`=default`	`=default`	(OK)
OO	`=default`	*declared*	`=default`	`=default`	not decl	not decl	**BAD**
`X(X const&)`	not decl	`=default`	*declared*	`=default`	not decl	not decl	**BAD**
`=(X const&)`	`=default`	`=default`	`=default`	*declared*	not decl	not decl	**BAD**
`X(X&&)`	not decl	`=default`	`=delete`	`=delete`	*declared*	not decl	**BAD**
`=(X&&)`	`=default`	`=default`	`=delete`	`=delete`	not decl	*declared*	(BAD)

1.1 Items to be discussed

Things I am unsure

— Are there further useful and safe exceptions?

2 Categories of safe user-defined classes

2.1 Plain Value Types

2.2 Monomorphic Object Types (better name) - Encapsulation Types

2.3 Polymorphic Object Types - Class Hierarchies

2.4 Resource Managing Types

3 Bibliography

Core Guidelines

MISRA

Rule of Zero

Howard's table