environments, such as a small organization overseeing a closed-source codebase where clients are able to request timely code changes, encouraging liberal use of **`final`** might not be problematic. Instead of promising everything up front, even when much of what is offered is not immediately useful, the default development approach might reasonably be to provide only what is immediately necessary and then quickly expose more if and as needed.

For other organizations, however, request-based code changes might not be a viable option and can result in unacceptable delays in responding to client needs. Systemic use of **`final`** *inherently* prevents opportunistic reuse that involves inheritance. Consequently, clients wishing to adapt an immutable component are often forced to wrap it or else create a redundant copy. Gratuitously forbidding clients from doing what they deem appropriate — and what they would otherwise be able to do for free — might well be perceived as unnecessary nannyism.[8]

Consider, for example, the Standard Template Library (STL) and, in particular, `std::vector`. One might argue that `std:vector` was designed to facilitate generic programming, has no virtual functions, and therefore should be specified as **`final`** to ensure its “proper” use and no other. Suppose, on the other hand, that teachers wanting to teach their students the value of **defensive programming**[9] were to create an exercise to implement a `CheckedVector<T>`, derived publicly from `std::vector<T>`.[10] By inheriting constructors (see Section 2.1.“Inheriting ~~Ctors~~” on page 535), it is simple to implement this derived class with an alternate implementation for just **`operator[]`**:

```
#include <vector>   // std::vector
#include <cassert>  // standard C assert macro

template <typename T>
class CheckedVector : public std::vector<T>
{
public:
    using std::vector<T>::vector;  // Inherit all ctors of std::vector<T>.

    using reference       = typename std::vector<T>::reference;
    using const_reference = typename std::vector<T>::const_reference;
    using size_type       = typename std::vector<T>::size_type;
```

[8]“Unnecessary nannyism” is a phrase Bjarne Stroustrup used to characterize his initial decision to restrict operators `[]`, `()`, and `->` to be members; see **stroustrup94**, Chapter 3, section 3.6.2, “Members and Friends,” pp. 81–83, particularly p. 83.

[9]See **lakos14a** and **lakos14b**.

[10]Bjarne Stroustrup stated (via email, April 11, 2021) that he himself has employed a class exercise in which the only two functions in the derived type (that he typically calls “Vector”) hide the **`operator[]`** overloads of `std::vector`. These implementations perform additional checking so that if they are ever called out of their valid range, instead of resulting in **undefined behavior**, they do something sensible, e.g., throw an exception or, even better, print an error message and then call `abort` to terminate the program. By not employing `assert`, as we do in our example, Stroustrup avoids using conditional compilation, which is not essential to the didactic purpose of this exercise. When necessary, Stroustrup removes `Vector` from production use.