```
    {
        // Allocate memory using malloc and return address.
    }
};

static_assert(sizeof(ObjectCreator<int,OpNewCreator>) == sizeof(std::size_t),"");
static_assert(sizeof(ObjectCreator<int,MallocCreator>)== sizeof(std::size_t),"");
```

Since `OpNewCreator` and `MallocCreator` do not have any data members, inheriting from either of them does not increase the size of `ObjectCreator` on any compiler that implements the ~~empty base~~ optimization. If someone later decides to declare them as **`final`**, inheriting becomes impossible, even if just privately as an optimization:

```
template <typename T>
class OpNewCreator final { /*...*/ };   // subsequently declared final

template <typename T>
class MallocCreator final { /*...*/ };  //      "           "       "

template <typename T, template<typename> class CreationPolicy>
class ObjectCreator : CreationPolicy<T>  // Error, derivation is disallowed.
{ /*...*/ };
```

By declaring the empty bases class **`final`**, a valid use case is needlessly prohibited. Using composition instead of **private inheritance** consumes at least one extra byte in the footprint of `ObjectCreator`,[15] which will inevitably also come at the cost of additional padding imposed by alignment requirements:

```
template <typename T, template<typename> class CreationPolicy>
class LargeObjectCreator
{
    CreationPolicy<T> policy;  // now consumes an extra byte &
    std::size_t objectCount = 0;  // with padding 8 extra bytes

public:
    T* create()
    {
        ++objectCount;
```

[15] C++20 adds a new attribute, `[[no_unique_address]]`, that allows the compiler to avoid consuming additional storage for data objects of empty classes:

```
struct A final { /* no data members */ };
struct S {
    [[no_unique_address]] A a;  static_assert(sizeof(a) >= 1, "");
    int x;                      static_assert(sizeof(x) == 4, "");
};                              static_assert(sizeof(S) == 4, "");
```