`CloningPtr` has a default constructor that creates a null pointer and a value constructor that allocates a copy of its argument on the heap. When a `CloningPtr` object is copied using the copy constructor or copy-assignment operator, a new copy of the source's managed object is allocated for the target `CloningPtr` to manage. Copy construction and copy assignment are potentially throwing operations because they (1) allocate memory and (2) call `T`'s copy constructor.

Now let's consider whether `CloningPtr` would benefit from defining move operations. A `CloningPtr` allocates a resource (the pointed-to object), and it can safely transfer that resource — via simple pointer moves — to the moved-to object without invoking any potentially throwing operations. We have, therefore, implemented a move constructor and move-assignment operator, both of which are decorated with the **`noexcept`** specifier. In both move operations, the `d_owned_p` pointer is copied from the moved-from object to the moved-to object, and then the moved-from object is set to null (to avoid having two `CloningPtr` objects attempting to own the same resource).

Note that `std::swap<T>` is declared such that when `T`'s move constructor and move-assignment operator are both **`noexcept`**, `std::swap<T>` is **`noexcept`** automatically:

```
namespace std {

template <typename T>
void swap(T& left, T& right)  // Note use of conditional noexcept syntax.
     noexcept(is_nothrow_move_constructible<T>::value &&
              is_nothrow_move_assignable<T>::value);

}  // close ~~std namespace~~
```

Thus, we need not provide a custom `swap` function for `CloningPtr` because the global one, provided by the Standard Library as the default, will do the job:

```
#include <string>       // std::string
#include <utility>      // std::swap
#include <type_traits>  // std::is_nothrow_move_constructible,
                        // std::is_nothrow_move_assignable
#include <cassert>      // standard C assert macro

void f1()
{
    typedef CloningPtr<std::string> PtrType;

    PtrType p1("hello");
    PtrType p2(p1);          // Clones the string owned by p1
    assert(*p1 == "hello");
    assert(*p2 == "hello");

    static_assert(std::is_nothrow_move_constructible<PtrType>::value, "");
    static_assert(std::is_nothrow_move_assignable<PtrType>::value, "");
```