chapter 7 .

Stack
LIFO

```
#include <exception>
#include <stack>
#include <mutex>
#include <memory>
#include <iostream>

struct empty_stack: std::exception
{
    const char* what() const throw()
    {
        return "empty stack";
    }

};

template<typename T>
class threadsafe_stack
{
private:
    std::stack<T> data;
    mutable std::mutex m;
public:
    threadsafe_stack(){}
    threadsafe_stack(const threadsafe_stack& other)
    {
        std::lock_guard lock{other.m};
        data=other.data;
    }
    threadsafe_stack& operator=(const threadsafe_stack&) = delete;

    void push(T new_value)
    {
        std::lock_guard lock{m};
        data.push(new_value);
    }
    std::shared_ptr<T> pop()
    {
        std::lock_guard lock{m};
        if(data.empty()) throw empty_stack();
        auto const res(std::make_shared<T>(data.top()));
        data.pop();
        return res;
    }
```

(4)

shared pointer
unique pointer
weak pointer

C++17 ← smart pointer

Keeps track how many pointers point to a memory

shared_ptr | new - delete

```
    void pop(T& value)
    {
        std::lock_guard lock{m};
        if(data.empty()) throw empty_stack();  (2)
        value=data.top();
        data.pop();
    }
    bool empty() const
    {
        std::lock_guard lock{m};
        return data.empty();
    }
};

int main()
{
    try
    {
        threadsafe_stack<int> si;
        si.push(5);  (0)
        std::cout << *si.pop() << std::endl;
        // if(!si.empty())
        {
            int x;
            si.pop(x); (1)
        }
    }
    catch(std::exception const& e)    (3)
    {  /* LOG */
       throw;
    }
    catch(...) // Catch anything else.
    {  /* LOG */
       throw;
    }
}
```

threadsafe_stack	stack
- pop allow you know what you popped	- pop you don't know what you popped