class ThreadWrapper{  
private:  
 std::thread thread\_;  
public:  
 ThreadWrapper() = default;  
 ~ThreadWrapper() = default;  
  
 ThreadWrapper(const ThreadWrapper& other) = delete;  
 ThreadWrapper& operator = (const ThreadWrapper& other) = delete;  
  
 ThreadWrapper(ThreadWrapper&& other) noexcept :  
 thred\_()  
 ThreadWrapper& operator = (const ThreadWrapper& other) noexcept {  
 if (thread\_.joinable()){  
 thread\_.join();  
 }  
 thread\_ = std::move(other.thread\_);  
 return \*this;  
 }  
  
 template<typename Fn, typename ...Args>  
 explicit ThreadWrapper(Fn&& cb, Args&&... args):  
 thread\_{cb, std::forward<Args>(args)...} {}  
  
 template <typename Fn, typename ...Args>  
 ThreadWrapper operator()(Fn&& cb, Args&&... args){  
 ThreadWrapper thread(cb, std::forward<Args>(args)...);  
 return thread;  
  
   
 }  
 std::thread& GetThread(){  
 return thread\_;  
 }  
};

int main() {  
 ThreadWrapper th{test, "Lol"};  
 ThreadWrapper th2 = ThreadWrapper("git");  
 th.GetThread().join();  
 std::cout << "Hello, World!" << std::endl;  
 return 0;  
}