실습 055 static 을 이용한 싱글톤 클래스 만들기

제출 마감일: 2023-04-09 23:59
업로드 가능한 파일 수: 3
제출 방식: 개인

목적

이 실습은 클래스의 static 멤버 변수와 함수를 사용하는 연습을 합니다.

설명

프로그램이 실행 중에 단 하나의 객체만 존재해야 한다면 어떻게 구현할 수 있을까요?

객체 생성 패턴 중 말도 많고 탈도 많은 싱글톤(Singleton) 이 심플하게 구현할 수 있는 대표적인 방법입니다.

static 키워드로 시작해 보겠습니다.

static SharedPreferences sharedPreference { };

static 변수는 컴파일러가 딱 하나만 만들어 준다고 배웠습니다.

하지만, static initialization order problem 발생할 수 있습니다.

이 문제는 x translation unit 과 y translation unit 이 누가 먼저 실행 되어 초기화 되는지 정의되지 않았서 발생합니다.

쉽게 말하면 x.cpp 가 y.cpp 에 있는 객체의 함수를 호출하는 코드를 실행할 때,

x 객체가 먼저 생성될 지, y 객체가 먼저 생성될 지 실행 때 마다 달라질 수 있다는 얘기입니다.

이를 방지하기 위한 심플한 방법은 the Construct On First Use 이디엄을 사용하는 것입니다.

즉, 객체를 생성하는 메서드를 제공하고, 그 메서드 내부에서 static 변수로 선언하라는 것입니다.

SharedPreferences& createSharedPreference() {

```
    static SharedPreferences sharedPreferences;

    return sharedPreferences;

}
```

이제 SharedPreferences 객체를 사용하는 개발자에게 해당 객체는 하나만 생성된다고 알려 줍시다.

일반적으로 생성자를 private 으로 선언해서 직접 생성할 수 없도록 조치 후, 객체를 하나만 생성하는 getInstance() 스태틱 함수를 정의합니다.

```
class SharedPreferences {
private:
    SharedPreferences () {}
public:
    static SharedPreferences& getInstance() {
        // thread-safety since C++11
        static SharedPreferences sharedPreferences;

        return sharedPreferences;

    }
}
```

코어 가이드 라인 - CP (Concurrency and parallelism) .1: Assume that your code will run as part of a multi-threaded program

부분을 참고해 보면, C++11 부터는 static 변수의 초기화는 thread-safety (쓰레드에 안전) 보장한다고 합니다.

문제

SharedPreferences 객체를 단 하나만 생성하는 클래스를 정의하시오.

<참고>

```
int main() {

    WorkerThread worker;

    worker.play();

}
```

//WorkerThread.h

```
class WorkerThread {
public:
    void play();
private:
    void readWorker();
    void writeWorker(int value);
    std::mutex mu;
    std::vector<SharedPreferences*> answers;
};
```

//WorkerThread.cpp

```
void WorkerThread::readWorker() {
    size_t size = SharedPreferences::getInstance().size();
    std::lock_guard<std::mutex> map_locker(mu);
    answers.emplace_back(&SharedPreferences::getInstance());
    //std::cout << "read by key: " << &SharedPreferences::getInstance() << " " << size << " " << SharedPreferences::getInstance().getInt("key") << std::endl;
}

void WorkerThread::writeWorker(int value) {
    //auto preferences = SharedPreferences::getInstance();
    SharedPreferences::getInstance().putInt("key", value);
    size_t size = SharedPreferences::getInstance().size();
    std::lock_guard<std::mutex> map_locker(mu);
    answers.emplace_back(&SharedPreferences::getInstance());
    //std::cout << "write by key: " << &SharedPreferences::getInstance() << " " << size << " " << value << std::endl;
}

void WorkerThread::play() {
    WorkerThread workerThread;
    // std::cerr << "h/w cores: " << std::thread::hardware_concurrency() << std::endl;
    std::thread threads[8];
    for(int i=0; i < 8; i+=2) {
        threads[i] = std::thread ([=] { writeWorker(i); });
        threads[i + 1] = std::thread ([=] { readWorker(); });
    }

    for (int i=0; i < 8; i++) {
        threads[i].join();
    }

    if(std::all_of(std::cbegin(answers), std::cend(answers), [=](const auto& e) {return e == &SharedPreferences::getInstance();}))
        std::cout << "PASS" << std::endl;
    else
        std::cout << "FAIL" << std::endl;
}
```

//SharedPreferences.h

```
class SharedPreferences;
class SharedPreferences {
private:
    std::map<std::string, int> preferences;
    SharedPreferences()= default;
    ~SharedPreferences()= default;
    SharedPreferences(const SharedPreferences&)= delete;
    SharedPreferences& operator=(const SharedPreferences&)= delete;
    friend class std::vector<SharedPreferences*>;
public:
    static SharedPreferences& getInstance();
    void putInt(std::string key, int value);
    int getInt(std::string key);
    size_t size();
};
```

입력

없음

출력

PASS (성공 시)

FAIL (실패 시)

제출파일

SharedPreferences.cpp

55.csv

입출력

입력	출력
	PASS