МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

“НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

Лабораторная работа №1 по дисциплине «Технологии и методы программирования»

### Рассмотрение принципов Singleton, относительно паттернов, используемых в ООП.

Группа: АИ-02

Студент: Григорьев Д.А, Белкин Н.М.

Преподаватель: Медведев М.А.

Новосибирск 2022

**Цели и задачи работы:** Рассмотреть сильные и слабые стороны внедрения паттерна Singleton. Выполнить реализацию, если бы в задаче использовался паттерн SOLID. Рассмотреть проблематику тестирования программного кода и связанности кода.

**Методика выполнения работы:**

1. Разработать алгоритм решения задачи по индивидуальному заданию.
2. Написать и отладить программу решения задачи в паре.
3. Протестировать работу программы на различных исходных данных. Контрольный пример  с результатами ручных расчетов представить в отчете.
4. Сравнить программную реализацию 2 паттернов проектирования.

**Задание**

Рассмотреть предметную область на основании изначальных данных, построить модели программного кода с использованием паттерна singleton и SOLID. Реализовать задачу и выстроить аргументированную защиту/критику кода оппонента. Результатом работы программного кода должна быть имитация процесса задания.

**Текст программы:**

***Файл first\_med\_SOLID.cpp***

#include <iostream>

#include <memory>

class System {

public:

System() {};

~System() {};

virtual void getVolume() {

std::cout << "Volume is " << value << std::endl;

};

friend class Mute;

friend class Unmute;

protected:

int value = 0, muteVolume = false;

};

class Mute {

public:

Mute() {};

~Mute() {};

void mute(std::shared\_ptr<System> sys) {

sys->muteVolume = true;

std::cout << "Volume is muted\n";

}

};

class Unmute {

public:

Unmute() {};

~Unmute() {};

void unmute(std::shared\_ptr<System> sys) {

sys->muteVolume = false;

std::cout << "Volume is unmuted\n";

}

};

class Volume : public System {

public:

Volume() {};

~Volume() {};

void increaseVolume(int val) {

if (muteVolume == true) {

std::cout << "Current volume is 0 because volume is muted\n";

}

else if (value < 100) {

value += val;

if (value > 100) {

value = 100;

std::cout << "Volume is maxed out, making it equal to one hundred\n";

}

else {

std::cout << "Current volume is " << value << ", volume is increased by " << val << "\n";

}

}

}

void decreaseVolume(int val) {

if (muteVolume == true) {

std::cout << "Current volume is 0 because volume is muted\n";

}

else if (value > 0) {

value -= val;

if (value < 0) {

value = 0;

std::cout << "Volume is below zero, making it equal to zero\n";

}

else {

std::cout << "Current volume is " << value << ", volume is decreased by " << val << "\n";

}

}

}

};

void main() {

auto system = std::make\_shared<Volume>();

auto muteVol = std::make\_unique<Mute>();

auto unmuteVol = std::make\_unique<Unmute>();

system->getVolume();

system->decreaseVolume(10);

muteVol->mute(system);

system->increaseVolume(30);

system->decreaseVolume(20);

unmuteVol->unmute(system);

system->increaseVolume(30);

}

***Файл source\_4\_singleton.cpp***

#include <iostream>

class Volume {

public:

static Volume& GetInstance() {

static Volume\* singleton = new Volume();

singleton->getVolume();

return \*singleton;

}

void increaseVolume(int val) {

value += val;

if (value > 100) {

value = 100;

std::cout << "Volume is maxed out\n";

}

else if (muteVolume == true) {

std::cout << "Current volume is 0 because volume is muted\n";

}

else {

std::cout << "Current volume is " << value << ", volume is increased by " << val << "\n";

}

}

void decreaseVolume(int val) {

value -= val;

if (value < 0 && muteVolume == false) {

value = 0;

std::cout << "Volume is below zero\n";

}

else if (muteVolume == true) {

std::cout << "Current volume is 0 because volume is muted\n";

}

else {

std::cout << "Current volume is " << value << ", volume is decreased by " << val << "\n";

}

}

void mute() {

muteVolume = true;

value = 0;

}

void unmute() {

muteVolume = false;

value = 0;

}

private:

int value = 35, muteVolume = false;

Volume() = default;

void getVolume() {

std::cout << "Volume is " << value << std::endl;

}

Volume(const Volume&) = delete;

Volume& operator=(const Volume&) = delete;

Volume(Volume&&) = delete;

Volume& operator=(Volume&&) = delete;

};

void main() {

Volume& system = Volume::GetInstance();

system.increaseVolume(40);

system.decreaseVolume(10);

}

**Выводы**

* Сильные стороны паттерна Singleton:  
  Позволяет контролировать создание экземпляров класса, ограничивая их количество одним. Предоставляет доступ к свойству через глобально доступный метод.
* Слабые стороны паттерна Singleton:  
  Синглтон нарушает SRP (Single Responsibility Principle) — помимо своих обязанностей, он контролирует создание экземпляров класса. Неявная зависимость подсистем друг от друга введу наличия глобального состояния, изменения которого сложно контролировать. Это порождает ещё одну проблему — сложность тестирования, потому что появляется сильная связанность, и может оказаться так, что несколько тестов будут зависеть друг от друга.
* Сильные стороны паттерна SOLID:

Позволяет легко поддерживать/изменять код и так же сам код лёгок для понимания человек, впервые сталкивающимся с ним.

* Слабые стороны паттерна SOLID:

Увеличеное время на поддержку и не всегда понятное соблюдение принципов. Если речь идёт про бизнес, возможно и так что SOLID могут в какой-то момент вообще отбросить, уступив дорогу трате времени на проработку фич.