МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«РОССИЙСКИЙ БИОТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ (РОСБИОТЕХ)»

Институт промышленной инженерии, информационных технологий и мехатроники

Кафедра «Информатика и вычислительная техника пищевых производств»

Направление: 09.03.01 Информатика и вычислительная техника

**ЛАБОРАТОРНАЯ РАБОТА №**3-4

*на тему:*

«РАЗРАБОТКА КЛАССОВ В С++»

Вариант № 30

|  |  |  |
| --- | --- | --- |
| Выполнил:  ФИО |  | Студент 1 курса, гр. 24о-090301/**ИИ1**  Чудаков Стефан Алексеевич |
| Проверил: |  | Ящун Т.В. |

Москва, 2025

Цель работы: исследовать методы и принципы разработки классов в языке C++, систематизировать знания об инкапсуляции как фундаментальном механизме защиты и организации данных, а также освоить практическое применение различных типов конструкторов (по умолчанию, параметризированных, копирования) и деструкторов.

Геология 1. Опишите класс «Минерал» – Mineral. Класс должен содержать: • Поля для хранения названия минерала, его плотности и списка твердости при разной температуре. • Методы get и set для описанных полей. • Конструкторы (по умолчанию, параметризованные, копирования, преобразования, де легирующие). • Метод вычисления средней твердости. • Перегруженные операции: сравнение по средней твердости, сложение, присваивание, постфиксный и префиксный инкремент, ввод и вывод информации 2. Создайте массив объектов класса. 3. Протестируйте все созданные методы и операции. 4. Опишите функцию, возвращающую список минералов, отсортированных по средней твердости.

Код:

Chudakov\_LR3-4\_ClassMineral.h

#ifndef CHUDAKOV\_LR3\_4\_CLASSMINERAL\_H

#define CHUDAKOV\_LR3\_4\_CLASSMINERAL\_H

#include <string>

#include <vector>

#include <iostream>

class Mineral {

private:

    std::string name;

    double density;

    std::vector<double> hardness\_list;

public:

    // Конструкторы

    Mineral(); // По умолчанию

    Mineral(const std::string& n, double d, const std::vector<double>& h);

    Mineral(const Mineral& other); // Копирования

    explicit Mineral(const char\* n); // Преобразования из строки

    Mineral(const std::string& n, double d); // Делегирующий

    // Методы get

    std::string getName() const;

    double getDensity() const;

    std::vector<double> getHardnessList() const;

    // Методы set

    void setName(const std::string& n);

    void setDensity(double d);

    void setHardnessList(const std::vector<double>& h);

    // Метод вычисления средней твердости

    double calculateAverageHardness() const;

    // Перегруженные операции

    bool operator==(const Mineral& other) const;

    bool operator>(const Mineral& other) const;

    bool operator<(const Mineral& other) const;

    bool operator!=(const Mineral& other) const;

    bool operator>=(const Mineral& other) const;

    bool operator<=(const Mineral& other) const;

    Mineral operator+(const Mineral& other) const;

    Mineral& operator=(const Mineral& other);

    Mineral& operator++();    // Префиксный инкремент (увеличивает плотность)

    Mineral operator++(int);  // Постфиксный инкремент

    friend std::ostream& operator<<(std::ostream& os, const Mineral& m);

    friend std::istream& operator>>(std::istream& is, Mineral& m);

    ~Mineral() = default;

};

#endif // CHUDAKOV\_LR3\_4\_CLASSMINERAL\_H

Chudakov\_LR3-4\_ClassMineral.cpp

#include "Chudakov\_LR3-4\_ClassMineral.h"

#include <numeric>

// Конструкторы

Mineral::Mineral() : name("Unknown"), density(0.0), hardness\_list() {}

Mineral::Mineral(const std::string& n, double d, const std::vector<double>& h)

    : name(n), density(d), hardness\_list(h) {}

Mineral::Mineral(const Mineral& other)

    : name(other.name), density(other.density), hardness\_list(other.hardness\_list) {}

Mineral::Mineral(const char\* n) : name(n), density(0.0), hardness\_list() {}

Mineral::Mineral(const std::string& n, double d) : Mineral(n, d, {}) {}

// Методы get

std::string Mineral::getName() const { return name; }

double Mineral::getDensity() const { return density; }

std::vector<double> Mineral::getHardnessList() const { return hardness\_list; }

// Методы set

void Mineral::setName(const std::string& n) { name = n; }

void Mineral::setDensity(double d) { density = d; }

void Mineral::setHardnessList(const std::vector<double>& h) { hardness\_list = h; }

// Средняя твердость

double Mineral::calculateAverageHardness() const {

    if (hardness\_list.empty()) return 0.0;

    return std::accumulate(hardness\_list.begin(), hardness\_list.end(), 0.0) / hardness\_list.size();

}

// Операторы

bool Mineral::operator==(const Mineral& other) const {

    return (calculateAverageHardness() == other.calculateAverageHardness());

}

bool Mineral::operator>(const Mineral& other) const {

    return (calculateAverageHardness() > other.calculateAverageHardness());

}

bool Mineral::operator<(const Mineral& other) const {

    return (calculateAverageHardness() < other.calculateAverageHardness());

}

bool Mineral::operator!=(const Mineral& other) const {

    return (calculateAverageHardness() != other.calculateAverageHardness());

}

bool Mineral::operator>=(const Mineral& other) const {

    return (calculateAverageHardness() >= other.calculateAverageHardness());

}

bool Mineral::operator<=(const Mineral& other) const {

    return (calculateAverageHardness() <= other.calculateAverageHardness());

}

Mineral Mineral::operator+(const Mineral& other) const {

    std::vector<double> combined\_hardness = hardness\_list;

    combined\_hardness.insert(combined\_hardness.end(), other.hardness\_list.begin(), other.hardness\_list.end());

    return Mineral(name + "+" + other.name, (density + other.density) / 2, combined\_hardness);

}

Mineral& Mineral::operator=(const Mineral& other) {

    if (this != &other) {

        name = other.name;

        density = other.density;

        hardness\_list = other.hardness\_list;

    }

    return \*this;

}

Mineral& Mineral::operator++() {

    density += 1.0;

    return \*this;

}

Mineral Mineral::operator++(int) {

    Mineral temp = \*this;

    ++(\*this);

    return temp;

}

std::ostream& operator<<(std::ostream& os, const Mineral& m) {

    os << "Mineral: " << m.name << ", Density: " << m.density << ", Hardness: [";

    for (size\_t i = 0; i < m.hardness\_list.size(); ++i) {

        os << m.hardness\_list[i];

        if (i < m.hardness\_list.size() - 1) os << ", ";

    }

    os << "]";

    return os;

}

std::istream& operator>>(std::istream& is, Mineral& m) {

Chudakov\_LR3-4\_Methods.h

#ifndef CHUDAKOV\_LR3\_4\_METHODS\_H

#define CHUDAKOV\_LR3\_4\_METHODS\_H

#include "Chudakov\_LR3-4\_ClassMineral.cpp"

#include <vector>

#include <algorithm>

extern std::vector<Mineral> minerals; // Глобальный вектор минералов

// Функции для работы с меню

void createDefaultMineral();

void createParameterizedMineral();

void createCopyMineral();

void showAllMinerals();

void testOperations();

void showSortedMinerals();

#endif // CHUDAKOV\_LR3\_4\_METHODS\_H

Chudakov\_LR3-4\_Methods.cpp

#include <iostream>

#include <functional>

#include <vector>

#include <random>

#include <algorithm>

#include <cstdlib>

#include "Chudakov\_LR3-4\_ClassMineral.cpp"

bool userInput(std::string input) {

    if (input.empty()) {

        return false;

    }

    try {

        int lastTry = stoi(input);

        if (lastTry < 0) {

            return false;

        }

    }

    catch (...) {

        return false;

    }

    return true;

}

bool userInputDouble(std::string input) {

    if (input.empty()) {

        return false;

    }

    try {

        double lastTry = stod(input);

        if (lastTry < 0) {

            return false;

        }

    }

    catch (...) {

        return false;

    }

    return true;

}

std::function<void()> enterNumber(int& variousLink, std::string label) {

    return [&variousLink, label]() {

        std::string rawInput;

        std::cout << label;

        std::getline(std::cin, rawInput);

        while (!userInput(rawInput)) {

            std::cout << label;

            std::getline(std::cin, rawInput);

        }

        variousLink = stoi(rawInput);

        };

}

std::function<void()> enterDouble(double& variousLink, std::string label) {

    return [&variousLink, label]() {

        std::string rawInput;

        std::cout << label << " - ";

        std::getline(std::cin, rawInput);

        while (!userInputDouble(rawInput)) {

            std::cout << label << " - ";

            std::getline(std::cin, rawInput);

        }

        variousLink = stod(rawInput);

        };

}

std::function<void()> enterString(std::string& variousLink, std::string label) {

    return [&variousLink, label]() {

        std::cout << label << " - ";

        std::getline(std::cin, variousLink);

    };

}

bool user\_comparison\_operator(std::string various\_link) {

    if (various\_link == "==" || various\_link == "!=" || various\_link == ">" || various\_link == "<" || various\_link == ">=" || various\_link == "<=") {

        return true;

    }

    return false;

}

std::function<void()> enter\_the\_сcomparison\_operator(std::string& varlink, std::string label) {

    return [&varlink, label]() {

        std::string rawinput;

        std::cout << label << " - ";

        std::getline(std::cin, rawinput);

        while(!user\_comparison\_operator(rawinput)) {

            std::cout << label << " - ";

            std::getline(std::cin, rawinput);

        }

        varlink = rawinput;

    };

}

std::vector<Mineral> minerals;

double fRand(double fMin, double fMax) {

    double f = (double)rand() / RAND\_MAX;

    return fMin + f \* (fMax - fMin);

}

template<typename T>

T random(std::vector<T> const &vector) {

    auto it = vector.cbegin();

    int random = rand() % vector.size();

    std::advance(it, random);

    return \*it;

}

void createDefaultMinercal() {

    std::vector<std::string> names = {"Quartz", "Diorite", "Andezit", "Stone", "Gold", "Diamod", "Sapphire"};

    std::string name = random(names);

    double density = fRand(5.0, 20.0);

    std::vector<double> hardness\_list = {};

    for (int start = 0; start < 3; start++) {

        hardness\_list.push\_back(fRand(2.0, 15.0));

    }

    Mineral m(name, density, hardness\_list);

    minerals.push\_back(m);

    std::cout << "Created defolt mineral: " << m << std::endl;

}

void createParameterizedMineral() {

    std::string m\_name;

    double m\_density;

    std::vector<double> m\_hardness;

    enterString(m\_name, "Input the name of the mineral: ")();

    enterDouble(m\_density, "Input the density of the mineral: ")();

    for (int start = 0; start < 3; start++) {

        double hardness = 0.0;

        enterDouble(hardness, "Input the hardness of the mineral: ")();

        m\_hardness.push\_back(hardness);

    }

    Mineral m(m\_name, m\_density, m\_hardness);

    minerals.push\_back(m);

    std::cout << "Created default mineral: " << m << std::endl;

}

void createCopyMineral() {

    if (!minerals.empty()) {

        int index = 0;

        enterNumber(index, "Enter the number of the mineral what copy you wanna do - ")();

        while (index > minerals.size()) {

            std::cout << "The number of the mineral what you wanna copy does not exist..." << std::endl;

            enterNumber(index, "Enter the number of the mineral what copy you wanna do - ")();

        }

        Mineral m = minerals[index - 1];

        minerals.push\_back(m);

        std::cout << "Created copy mineral: " << m << std::endl;

    } else {

        std::cout << "There are no minerals to copy..." << std::endl;

    }

}

void showAllMinerals() {

    if (minerals.empty()) {

        std::cout << "No minerals available." << std::endl;

        return;

    }

    for (const auto& m : minerals) {

        std::cout << m << " - Avg Hardness: " << m.calculateAverageHardness() << std::endl;

    }

}

void testOperations() {

    if (minerals.size() < 2) {

        std::cout << "Need at least 2 minerals!" << std::endl;

        return;

    }

    int index1;

    enterNumber(index1, "Input the number of the first mineral what you wanna compare - ")();

    while (index1 > minerals.size()) {

        std::cout << "The number of the mineral what you wanna compare does not exist..." << std::endl;

        enterNumber(index1, "Input the number of the first mineral what you wanna compare - ")();

    }

    int index2;

    enterNumber(index2, "Input the number of the second mineral what you wanna compare - ")();

    while (index2 > minerals.size()) {

        std::cout << "The number of the mineral what you wanna compare does not exist..." << std::endl;

        enterNumber(index2, "Input the number of the first mineral what you wanna compare - ")();

    }

    Mineral m1 = minerals[index1 - 1];

    Mineral m2 = minerals[index2 - 1];

    std::string choice\_operation;

    enter\_the\_сcomparison\_operator(choice\_operation, "Input the comparison operator what you wanna apply to")();

    if (choice\_operation == "==") {

        std::cout << "m1 + m2: " << (m1 + m2) << std::endl;

        std::cout << "m1 == m2: " << (m1 == m2 ? "true" : "false") << std::endl;

    } else if (choice\_operation == "!=") {

        std::cout << "m1 + m2: " << (m1 + m2) << std::endl;

        std::cout << "m1 != m2: " << (m1 != m2 ? "true" : "false") << std::endl;

    } else if (choice\_operation == ">") {

        std::cout << "m1 + m2: " << (m1 + m2) << std::endl;

        std::cout << "m1 > m2: " << (m1 > m2 ? "true" : "false") << std::endl;

    } else if (choice\_operation == "<") {

        std::cout << "m1 + m2: " << (m1 + m2) << std::endl;

        std::cout << "m1 < m2: " << (m1 < m2 ? "true" : "false") << std::endl;

    } else if (choice\_operation == ">=") {

        std::cout << "m1 + m2: " << (m1 + m2) << std::endl;

        std::cout << "m1 >= m2: " << (m1 >= m2 ? "true" : "false") << std::endl;

    } else {

        std::cout << "m1 + m2: " << (m1 + m2) << std::endl;

        std::cout << "m1 <= m2: " << (m1 <= m2 ? "true" : "false") << std::endl;

    }

}

void showSortedMinerals() {

Chudakov\_LR3-4\_Main.cpp

#include <map>

#include <locale>

#include "Chudakov\_LR3-4\_Methods.cpp"

//переопределить 6 оператров сравнения \*

//продемонстрировать операоры сравнения \*

//Рассказать про операторы ++ Mineral& Mineral::operator++() \*

//в конструктор по умолчанию добавить рандомные значения для default mineral \*

//void testOperations запрашивать номера элементов при демонстрации операторов \*

struct MenuItem {

    std::string title;

    void (\*action)();

};

int main() {

    std::srand(std::time(NULL));

    setlocale(LC\_ALL, "ru");

    std::map<int, MenuItem> menu = {

        {1, {"Create default mineral", createDefaultMinercal}},

        {2, {"Create parameterized mineral", createParameterizedMineral}},

        {3, {"Create copy mineral", createCopyMineral}},

        {4, {"Show all minerals", showAllMinerals}},

        {5, {"Test operations", testOperations}},

        {6, {"Show sorted minerals", showSortedMinerals}}

    };

    int choice;

    while (true) {

        std::cout << "\nMenu:\n";

        for (const auto& item : menu) {

            std::cout << item.first << ". " << item.second.title << std::endl;

        }

        enterNumber(choice, "0. Exit\nEnter choice: ")();

        if (choice == 0) break;

        if (menu.find(choice) != menu.end()) {

            menu[choice].action();

        } else {

            std::cout << "Invalid choice!" << std::endl;

        }

    }

    return 0;

}

Git:



Тест:

Menu:

1. Create default mineral

2. Create parameterized mineral

3. Create copy mineral

4. Show all minerals

5. Test operations

6. Show sorted minerals

0. Exit

Enter choice: 1

Created defolt mineral: Mineral: Sapphire, Density: 16.314, Hardness: [6.32765, 4.68554, 3.14618]

Menu:

1. Create default mineral

2. Create parameterized mineral

3. Create copy mineral

4. Show all minerals

5. Test operations

6. Show sorted minerals

0. Exit

Enter choice: 2

Input the name of the mineral: - Bronze

Input the density of the mineral: - 4.56

Input the hardness of the mineral: - 3.54

Input the hardness of the mineral: - 2.98

Input the hardness of the mineral: - 9.58

Created default mineral: Mineral: Bronze, Density: 4, Hardness: [3, 2, 9]

Menu:

1. Create default mineral

2. Create parameterized mineral

3. Create copy mineral

4. Show all minerals

5. Test operations

6. Show sorted minerals

0. Exit

Enter choice: 3

Enter the number of the mineral what copy you wanna do - 6

The number of the mineral what you wanna copy does not exist...

Enter the number of the mineral what copy you wanna do - 2

Created copy mineral: Mineral: Bronze, Density: 4, Hardness: [3, 2, 9]

Menu:

1. Create default mineral

2. Create parameterized mineral

3. Create copy mineral

4. Show all minerals

5. Test operations

6. Show sorted minerals

0. Exit

Enter choice: 4

Mineral: Sapphire, Density: 16.314, Hardness: [6.32765, 4.68554, 3.14618] - Avg Hardness: 4.71979

Mineral: Bronze, Density: 4, Hardness: [3, 2, 9] - Avg Hardness: 4.66667

Mineral: Bronze, Density: 4, Hardness: [3, 2, 9] - Avg Hardness: 4.66667

Menu:

1. Create default mineral

2. Create parameterized mineral

3. Create copy mineral

4. Show all minerals

5. Test operations

6. Show sorted minerals

0. Exit

Enter choice: 5

Input the number of the first mineral what you wanna compare - 9

The number of the mineral what you wanna compare does not exist...

Input the number of the first mineral what you wanna compare - 4

The number of the mineral what you wanna compare does not exist...

Input the number of the first mineral what you wanna compare - 1

Input the number of the second mineral what you wanna compare - 3

Input the comparison operator what you wanna apply to - >=

m1 + m2: Mineral: Sapphire+Bronze, Density: 10.157, Hardness: [6.32765, 4.68554, 3.14618, 3, 2, 9]

m1 >= m2: true

Menu:

1. Create default mineral

2. Create parameterized mineral

3. Create copy mineral

4. Show all minerals

5. Test operations

6. Show sorted minerals

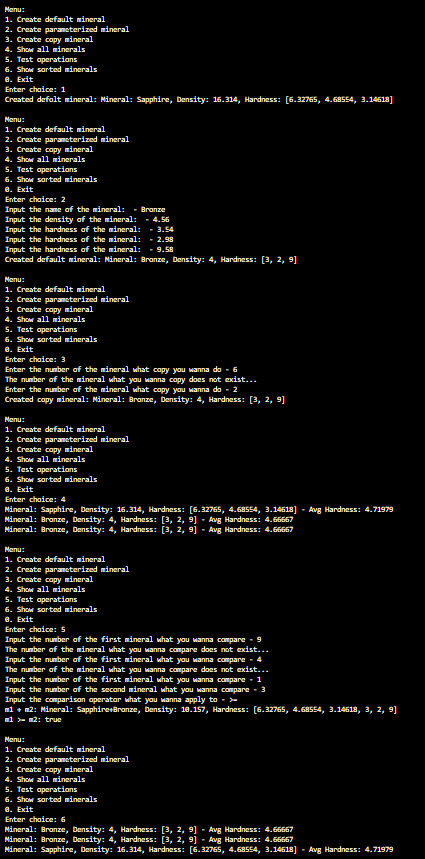
0. Exit

Enter choice: 6

Mineral: Bronze, Density: 4, Hardness: [3, 2, 9] - Avg Hardness: 4.66667

Mineral: Bronze, Density: 4, Hardness: [3, 2, 9] - Avg Hardness: 4.66667

Mineral: Sapphire, Density: 16.314, Hardness: [6.32765, 4.68554, 3.14618] - Avg Hardness: 4.71979



<https://github.com/Stephanchik/Chudakov_LR3-4_Mineral>

Вывод: Научился исследовать методы и принципы разработки классов в языке C++ и систематизировать знания об инкапсуляции как фундаментальном механизме защиты и организации данных, а также освоил практическое применение различных типов конструкторов и деструкторов.