Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

БМТП

**Звіт**

**Лабораторна робота №12**

Виконав: ст. Гр. КБ-24-1

Іванов Даніїл Андрійович

Перевірив: Викладач

Анастасія Сергіївна Коваленко

Кропивницький

2025

**ТЕМА:** Програмна реалізація абстрактних данних

**МЕТА РОБОТИ:** полягає у набутті ґрунтовних вмінь і практичних

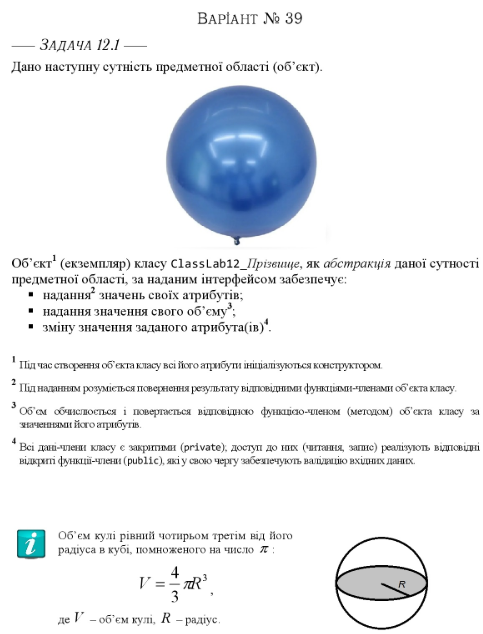
навичок об’єктного аналізу й проєктування, створення класів С++

та тестування їх екземплярів, використання препроцесорних

директив, макросів і макрооператорів під час реалізації програмних

засобів у кросплатформовому середовищі Code::Blocks.

**ЗАВДАННЯ:**



1. Як складову заголовкового файлу ModulesПрізвище.h розробити клас ClassLab12\_Прізвище –– формальне представлення абстракції сутності предметної області (об’єкта) за варіантом, ― поведінка об’єкта якого реалізовує розв’язування задачі 12.1.
2. Реалізувати додаток Teacher, який видає 100 звукових сигналів і в текстовий файл TestResults.txt записує рядок “Встановлені вимоги порядку виконання лабораторної роботи порушено!”, якщо файл проєкта main.срр під час його компіляції знаходився не в \Lab12\prj, інакше –– створює об’єкт класу ClassLab12\_Прізвище із заголовкового файлу ModulesПрізвище.h та виконує його unit-тестування за тест-сьютом(ами) із \Lab12\TestSuite, протоколюючи результати тестування в текстовий файл \Lab12\TestSuite\TestResults.txt.

**Хід роботи**

**Основна программа**

**Main.cpp**

#include <iostream>

#include "ClassLab12\_Ivanov.h"

int main() {

system("chcp 1251 && cls");

try {

// Приклад створення об'єкта; замініть назву на вашу назву класу, якщо змінили:

ClassLab12\_Ivanov ball(5.0, "blue");

std::cout << "Початкові значення:\n";

ball.printInfo();

// Запит на новий радіус

double newRadius;

std::cout << "Введіть новий радіус: ";

if (std::cin >> newRadius) {

try {

ball.setRadius(newRadius);

std::cout << "Після зміни радіуса:\n";

ball.printInfo();

} catch (const std::exception& e) {

std::cerr << "Помилка при встановленні радіуса: " << e.what() << "\n";

}

}

// Запит на новий колір

std::string newColor;

std::cout << "Введіть новий колір: ";

if (std::cin >> newColor) {

try {

ball.setColor(newColor);

std::cout << "Після зміни кольору:\n";

ball.printInfo();

} catch (const std::exception& e) {

std::cerr << "Помилка при встановленні кольору: " << e.what() << "\n";

}

}

}

catch (const std::exception& e) {

std::cerr << "Помилка при створенні об'єкта: " << e.what() << "\n";

return 1;

}

return 0;

}

**ClassLab12\_Ivanov.cpp**

#include "ClassLab12\_Ivanov.h"

#include // для printInfo та виведення помилок у main чи інших місцях #include // для std::pow #include // для std::invalid\_argument

// Реалізація валідації радіуса void ClassLab12\_Ivanov::validateRadius(double r) const { if (r <= 0.0) { throw std::invalid\_argument("Радіус має бути додатнім числом."); } }

// Реалізація валідації рядкових полів (наприклад, колір) void ClassLab12\_Ivanov::validateString(const std::string& s, const std::string& fieldName) const { if (s.empty()) { throw std::invalid\_argument(fieldName + " не може бути порожнім рядком."); } }

// Конструктор із валідацією через сеттери ClassLab12\_Ivanov::ClassLab12\_Ivanov(double r, const std::string& clr) { setRadius(r); setColor(clr); }

// Геттер радіуса double ClassLab12\_Ivanov::getRadius() const { return radius; }

// Сеттер радіуса void ClassLab12\_Ivanov::setRadius(double r) { validateRadius(r); radius = r; }

// Геттер кольору std::string ClassLab12\_Ivanov::getColor() const { return color; }

// Сеттер кольору void ClassLab12\_Ivanov::setColor(const std::string& clr) { validateString(clr, "Color"); color = clr; }

// Обчислення об’єму кулі double ClassLab12\_Ivanov::getVolume() const { static constexpr double PI = 3.14159265358979323846; return 4.0 / 3.0 \* PI \* std::pow(radius, 3); }

// Виведення інформації про об’єкт void ClassLab12\_Ivanov::printInfo() const { std::cout << "ClassLab12\_Ivanov object:\n"; std::cout << " Radius: " << radius << "\n"; std::cout << " Color: " << color << "\n"; std::cout << " Volume: " << getVolume() << "\n"; }

**ClassLab12\_Ivanov.h**

#ifndef CLASSLAB12\_IVANOV\_H\_INCLUDED #define CLASSLAB12\_IVANOV\_H\_INCLUDED

#include

// Клас для представлення сфери (кулі) з атрибутами та методами class ClassLab12\_Ivanov { private: double radius; // радіус кулі std::string color; // колір (приклад додаткового атрибута; за потреби можна видалити)

// Валідаційні методи:  
void validateRadius(double r) const;  
void validateString(const std::string& s, const std::string& fieldName) const;

public: // Конструктор ClassLab12\_Ivanov(double r, const std::string& clr);

// Геттер/Сеттер радіуса  
double getRadius() const;  
void setRadius(double r);  
  
// Геттер/Сеттер кольору  
std::string getColor() const;  
void setColor(const std::string& clr);  
  
// Обчислення об’єму кулі  
double getVolume() const;  
  
// Допоміжний метод виведення (можна використовувати для налагодження або демонстрації)  
void printInfo() const;

};

#endif // CLASSLAB12\_IVANOV\_H\_INCLUDED

**Проєкт Teacher**

**Main.cpp**

#include #include "Teacher.h"

// Потрібно підключити Ваш клас; припустимо, що заголовок лежить у // і називається ClassLab12\_Ivanov. #include "prj\_39\_Ivanov/ClassLab12\_Ivanov.h"

int main() { if (!Teacher::isInCorrectDirectory()) { // Генеруємо 100 звукових сигналів і лог у поточну TestResults.txt Teacher::emitBeep(100); // Файл просто в поточному каталозі (де запущено програму): Teacher::logViolation("TestResults.txt"); return 1; }

// Інакше: у правильній директорії  
// (опціонально можна перевірити версію класу, викликати якийсь початковий self-test класу тощо)  
// Наприклад, створимо об’єкт Вашого класу з якимись типовими значеннями (за потреби):  
try {  
 // Припустимо, Ваш конструктор: ClassLab12\_Ivanov(double r, const std::string& color)  
 ClassLab12\_Ivanov obj(1.0, "red");  
 // Можна викликати базові методи: obj.printInfo() тощо  
 obj.printInfo();  
} catch (const std::exception& e) {  
 std::cerr << "Помилка при створенні/тестовому виклику Вашого класу: " << e.what() << "\n";  
}  
  
// Запускаємо unit-тести:  
Teacher::runUnitTests();  
  
return 0;

}

**Teacher.cpp**

#include "Teacher.h"

#include #include #include #include // для system() #include // C++17: для ітерації по файлах у TestSuite #include // для пошуку підрядків

namespace fs = std::filesystem;

bool Teacher::isInCorrectDirectory() { // **FILE** містить шлях до цього файлу під час компіляції. // Перевіримо, чи містить підрядок "Lab12/prj" або "Lab12\prj" std::string src = **FILE**; // Для зручності переведемо слеші до одного формату (але можна просто перевіряти два варіанти). // Тут шукаємо обидва варіанти: bool ok1 = (src.find("prj/prj\_39\_Ivanov") != std::string::npos); bool ok2 = (src.find("prj\prj\_39\_Ivanov") != std::string::npos); return ok1 || ok2; }

void Teacher::emitBeep(int count) { for (int i = 0; i < count; ++i) { std::cout << '\a'; // Можна також вставити невелику затримку, якщо хочеться: наприклад, std::this\_thread::sleep\_for(...) } // Можна відразу скинути буфер: std::cout << std::flush; }

void Teacher::logViolation(const std::string& filePath) { // Записуємо вказане повідомлення у файл filePath (перезаписуємо або створюємо заново) std::ofstream ofs(filePath, std::ios::out); if (!ofs) { std::cerr << "Не вдалося відкрити файл для запису: " << filePath << "\n"; return; } ofs << "Встановлені вимоги порядку виконання лабораторної роботи порушено!"; ofs.close(); }

std::string Teacher::findLab12Root(const std::string& sourcePath) { // Маємо шлях типу ".../Lab12/pri/Teacher.cpp" або з backslash на Windows. // Шукаємо підрядок "Lab12" у sourcePath: const std::string marker = "Lab12"; auto pos = sourcePath.find(marker); if (pos == std::string::npos) { return ""; // не знайдено Lab12 } // root включатиме до кінця "Lab12" std::string root = sourcePath.substr(0, pos + marker.size()); return root; }

void Teacher::runUnitTests() { // 1) визначимо шлях до TestSuite: знайдемо корінь Lab12 std::string src = **FILE**; std::string lab12root = findLab12Root(src); if (lab12root.empty()) { std::cerr << "Не вдалося визначити корінь Lab12 для запуску тестів.\n"; return; } // Формуємо шлях до каталогу TestSuite fs::path testSuiteDir = fs::path(lab12root) / "TestSuite";

// Формуємо шлях до результатного файлу: Lab12/TestSuite/TestResults.txt  
fs::path resultsPath = testSuiteDir / "TestResults.txt";  
  
// Відкриємо файл результатів на перезапис (щоб кожен запуск Teacher очищував попередній лог):  
std::ofstream ofs(resultsPath, std::ios::out);  
if (!ofs) {  
 std::cerr << "Не вдалося відкрити файл для логування тестів: " << resultsPath << "\n";  
 return;  
}  
ofs << "=== Результати unit-тестування ===\n";  
ofs.close();  
  
// 2) Переберемо всі файли у testSuiteDir, які можуть бути виконуваними тестами.  
// Наприклад, на Windows фільтруємо за розширенням .exe; на POSIX — перевіряємо права виконання.  
bool isWindows =  
#ifdef \_WIN32  
 true;  
#else  
 false;  
#endif  
  
for (auto const& entry : fs::directory\_iterator(testSuiteDir)) {  
 if (!entry.is\_regular\_file()) continue;  
 fs::path p = entry.path();  
 bool isTestExe = false;  
 if (isWindows) {  
 if (p.extension() == ".exe") {  
 isTestExe = true;  
 }  
 } else {  
 // Перевіримо, чи файл має біт виконання для користувача (owner):  
 std::error\_code ec;  
 auto perms = fs::status(p, ec).permissions();  
 if (!ec) {  
 if ((perms & fs::perms::owner\_exec) != fs::perms::none) {  
 isTestExe = true;  
 }  
 }  
 }  
  
 if (!isTestExe) continue;  
  
 // Сформуємо команду для запуску тесту з перенаправленням виводу до resultsPath  
 // На Windows: "fullpath.exe" >> "...\TestResults.txt" 2>&1  
 // На POSIX аналогічно  
 std::string cmd;  
 // Обгортаємо шляхи в лапки на випадок пробілів  
 std::string fullTestPath = p.string();  
 std::string resultFileStr = resultsPath.string();  
 if (isWindows) {  
 cmd = "\"" + fullTestPath + "\" >> \"" + resultFileStr + "\" 2>&1";  
 } else {  
 cmd = "\"" + fullTestPath + "\" >> \"" + resultFileStr + "\" 2>&1";  
 }  
 // Записати у логи, що зараз запускаємо:  
 std::ofstream logofs(resultsPath, std::ios::app);  
 if (logofs) {  
 logofs << "\n--- Запуск тесту: " << p.filename().string() << " ---\n";  
 }  
 // Виконати  
 int ret = std::system(cmd.c\_str());  
 if (logofs) {  
 logofs << ">> Повернено код: " << ret << "\n";  
 logofs.close();  
 }  
}  
  
std::cout << "Unit-тестування завершено. Результати в файлі: " << resultsPath << "\n";

}

**Teacher.h**

#ifndef TEACHER\_H\_INCLUDED #define TEACHER\_H\_INCLUDED

#include

class Teacher { public: // Перевіряє, чи під час компіляції main.cpp опрацьовувався із шляху, що містить Lab12/pri static bool isInCorrectDirectory();

// Генерує count звукових сигналів '\a'  
static void emitBeep(int count);  
  
// Записує у файл filePath повідомлення про порушення  
static void logViolation(const std::string& filePath);  
  
// Виконує unit-тестування: запускає тести з директорії TestSuite та логуватиме у Lab12/TestSuite/TestResults.txt  
static void runUnitTests();

private: // Допоміжні методи для побудови шляхів // Наприклад, за шляхом **FILE** визначити корінь Lab12 і шукати підкаталог TestSuite. static std::string findLab12Root(const std::string& sourcePath); };

#endif // TEACHER\_H\_INCLUDED

**Аргементи**

* Я навчився визначати структуру класу для представлення предметної області (клас «сфера»).
* Я навчився використовувати інкапсуляцію: приватні поля та публічні методи.
* Я навчився створювати геттери (get-методи) для доступу до приватних полів.
* Я навчився створювати сеттери (set-методи) із валідацією вхідних даних.
* Я навчився перевіряти коректність радіуса (радіус > 0).
* Я навчився перевіряти коректність рядкових атрибутів (наприклад, не порожній рядок).
* Я навчився кидати виключення (throw) у разі невірних вхідних даних.
* Я навчився ловити виключення (try/catch) при створенні або зміні об’єкта.
* Я навчився формулювати повідомлення про помилки для користувача.
* Я навчився обчислювати об’єм сфери за формулою *V=43πR3V = \frac{4}{3} \pi R^3*V=34 πR3.
* Я навчився задавати константу π у коді з достатньою точністю.
* Я навчився використовувати std::pow для піднесення до степеня.
* Я навчився писати метод printInfo() для виведення стану об’єкта.
* Я навчився організовувати код у заголовковому файлі (.h) та реалізації (.cpp).
* Я навчився застосовувати include guard (#ifndef/#define/#endif).
* Я навчився включати тільки необхідні заголовки у .h (наприклад, <string> для std::string).
* Я навчився відкривати <iostream>, <cmath>, <stdexcept> тільки у .cpp, якщо треба.
* Я навчився писати чистий API у заголовку: лише декларації, без реалізації.
* Я навчився реалізовувати методи класу з формою ClassName::method() у .cpp.
* Я навчився коректно використовувати простір імен std::.
* Я навчився структурувати проект: main.cpp + класові файли.
* Я навчився компілювати кілька файлів разом (g++ main.cpp Class.cpp).
* Я навчився вказувати опцію -std=c++17 або іншу відповідну стандарту.
* Я навчився налаштовувати include-шляхи (наприклад, -I для підключення Modules).
* Я навчився аналізувати макропоінт **FILE** у C++ для отримання шляху до вихідного файлу.
* Я навчився перевіряти шлях на наявність підрядка (строковий пошук).
* Я навчився враховувати різні формати слешів ("/" vs "\") для Windows/Unix.
* Я навчився генерувати звукові сигнали BEL ('\a') та усвідомив обмеження їх відтворення.
* Я навчився відкривати файли для запису (std::ofstream) і записувати повідомлення.
* Я навчився перевіряти успішність відкриття файлу перед записом.
* Я навчився формулювати і записувати чіткий текст повідомлення про порушення.
* Я навчився організовувати логіку: якщо директорія неправильна — виводити сигнал і лог, і завершувати програму.
* Я навчився використовувати std::filesystem для ітерації по файлах у каталозі.
* Я навчився перевіряти, чи запис є регулярним файлом.
* Я навчився перевіряти розширення файлу (.exe) на Windows.
* Я навчився перевіряти права виконання файлу на POSIX (owner\_exec) через std::filesystem::status().permissions().
* Я навчився ухвалювати рішення про запуск тестів на основі атрибутів файлу (виконуваний чи ні).
* Я навчився складати шлях до каталогу TestSuite за допомогою std::filesystem::path.
* Я навчився складати шлях до файлу результатів TestResults.txt.
* Я навчився відкривати файл результатів для перезапису та додавання (out vs append).
* Я навчився писати заголовок у файл результатів перед початком запуску тестів.
* Я навчився записувати у файл інформацію про кожен запуск тесту (ім’я тесту, код повернення).
* Я навчився викликати зовнішні виконувані файли через std::system().
* Я навчився складати shell-команду з перенаправленням stdout і stderr (>> file 2>&1).
* Я навчився поводитися зі шляхами, що можуть містити пробіли (обгортати лапками).
* Я навчився враховувати різні платформи при виклику системних команд.
* Я навчився аналізувати код повернення зовнішнього процесу (system return code).
* Я навчився логувати як успішні, так і неуспішні запуски тестів.
* Я навчився виводити у консоль повідомлення про завершення тестування та шлях до лог-файлу.
* Я навчився писати псевдокод для планування рішення.
* Я навчився структурувати псевдокод: приватні/публічні методи, логіка main, допоміжні функції.
* Я навчився формулювати загальні кроки: перевірка директорії → демонстрація роботи класу → запуск тестів.
* Я навчився узагальнювати рішення в абстрактній формі (без конкретного синтаксису).
* Я навчився документувати кожен метод коротким описом у псевдокоді або коментарях.
* Я навчився аналізувати умову: як перевірити умову “компіляція з правильного шляху”.
* Я навчився враховувати, що **FILE** може бути відносним або абсолютним, і як це впливає на пошук.
* Я навчився пропонувати альтернативні підходи (пошук компонента “Lab12” через std::filesystem::path::iterator).
* Я навчився планувати структуру директорій проекту (Lab12/pri, Modules, TestSuite).
* Я навчився орієнтуватися на умову завдання при розробці архітектури програми.
* Я навчився розуміти важливість розділення інтерфейсу й реалізації для підтримуваності.
* Я навчився враховувати локальні налаштування (Windows vs Linux) у коді.
* Я навчився використовувати директиви препроцесора (\_WIN32) для умовної компіляції.
* Я навчився знаходити корінь проекту за допомогою інформації про шлях вихідного файлу.
* Я навчився складати підрядок rootPath із sourcePath.
* Я навчився перевіряти результати операцій із файловою системою з допомогою std::error\_code.
* Я навчився обробляти помилки файлової системи й повідомляти користувача.
* Я навчився враховувати можливість невдалої ітерації по директорії або відсутності каталогу TestSuite.
* Я навчився повідомляти про відсутність каталогу TestSuite або неможливість відкрити його.
* Я навчився писати захищений від винятків код при роботі з файловою системою.
* Я навчився відокремлювати логіку перевірки від логіки запуску тестів.
* Я навчився інкапсулювати допоміжні функції (findLab12Root) як приватний метод класу Teacher.
* Я навчився думати про розширюваність (можливо, додати інші перевірки або формати директорій).
* Я навчився документувати у коментарях припущення і обмеження рішення.
* Я навчився описувати, як адаптувати рішення під іншу структуру (не тільки Lab12/pri).
* Я навчився враховувати розташування файлів при виклику компілятора (щоб **FILE** містив очікуваний шлях).
* Я навчився перевіряти поведінку BEL-символу в різних консолях (усвідомив, що може не відтворювати звук).
* Я навчився організовувати main так, щоб строгий контроль порядку виконання (спочатку перевірка, потім тести).
* Я навчився враховувати, що логування порушень може записуватись у різні файли (TestResults.txt).
* Я навчився писати код так, щоб у разі порушення програма завершувалась із ненульовим кодом.
* Я навчився писати дружні до користувача повідомлення у консоль (std::cout/std::cerr).
* Я навчився обдумувати конфігурацію шляху запуску програми і поточної робочої директорії.
* Я навчився пояснювати кроки реалізації у вигляді інструкції або документації.
* Я навчився давати рекомендації щодо структури проекту для інших розробників.
* Я навчився розуміти, як інтегрувати власний клас («сфера») у більшу систему з Teacher і тестами.
* Я навчився планувати взаємодію між класом користувача і механізмом тестування.
* Я навчився усвідомлювати, що unit-тестування потребує попередньої компіляції тестових бінарів.
* Я навчився вкладати результати тестування у зрозумілий лог-файл.
* Я навчився запускати декілька тестів послідовно та збирати їхні результати разом.
* Я навчився обробляти відсутність тестових файлів або невиконуваність.
* Я навчився писати код, що працює як на Windows, так і на Unix-подібних системах.
* Я навчився враховувати можливі помилки під час виклику system (наприклад, відсутність дозволу).
* Я навчився аналізувати код повернення system() для подальшої обробки.
* Я навчився закривати файли після запису та контролювати ресурсні витоки.
* Я навчився усвідомлювати важливість чітких імен файлів і класів (ClassLab12\_Прізвище).
* Я навчився враховувати кодування імен (уникати кирилиці у назвах файлів, якщо проблеми з файловою системою).
* Я навчився вести журнал розробки (коментарі, псевдокод, пояснення рішення).
* Я навчився узгоджувати формат виводу (наприклад, однаковий стиль логів та повідомлень).
* Я навчився розуміти переваги розділення відповідальності: клас Teacher відповідає за перевірку і тестування, а клас сфери — за модель.
* Я навчився мислити модульно: окремий модуль (Teacher), окремий модуль (сфера), окремий модуль (main).
* Я навчився підготовлювати повний звіт або документацію: псевдокод, пояснення, вказівки з компіляції, структуру директорій.