Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

Механіко-технологічний факультет

ЗВІТ

ПРО ВИКОНАННЯ ЛАБОРАТОРНОЇ РОБОТИ №A[12]

з навчальної дисципліни

“Базові методології та технології програмування”

ПРОГРАМНА РЕАЛІЗАЦІЯ АБСТРАКТНИХ ТИПІВ ДАНИХ

ЗАВДАННЯ ВИДАВ

доцент кафедри кібербезпеки та програмного забезпечення

Доренський О. П.

[https://github.com/odorenskyi/](https://github.com/odorenskyi/Dmytro-Parkhomenko-KB18)

ВИКОНАВ

студент академічної групи КБ-24

Заріцкий В. А.

ПЕРЕВІРИВ

викладач кафедри кібербезпеки   
та програмного забезпечення

Коваленко А. С.

Кропивницький – 2025

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.   
**Варіант №**1

**Завдання до лабораторної роботи**

Як складову заголовкового файлу ModulesZaritsky.h розробити клас ClassLab12\_Zaritsky — формальне представлення абстракції сутності предметної області (об’єкта) за варіантом, ― поведінка об’єкта якого реалізовує розв’язування задачі 12.1.

1. 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\_Zaritsky із заголовкового файлу ModulesZaritsky.h та виконує його unit-тестування за тест-сьютом(ами) із \Lab12\TestSuite\, протоколюючи результати тестування в текстовий файл \Lab12\TestSuite\TestResults.txt.

# Аналіз вимог до програмного модуля (класу C++)

Необхідно створити клас ClassLab12\_Zaritsky, який буде абстракцією кубічного кавуна з такими вимогами:

1. Атрибути класу:

* Довжина грані куба (a)
* Колір шкірки кавуна (зовнішній вигляд)
* Колір м'якоті кавуна (внутрішній вигляд)
* Стиглість кавуна (у відсотках)

1. Операції:

* Конструктор(и) для ініціалізації атрибутів
* Надання значень атрибутів (getters)
* Обчислення і повернення об'єму кавуна за формулою
* Зміна значень атрибутів (setters) з валідацією

1. Обмеження і валідація:

* Довжина грані повинна бути додатним числом
* Стиглість має бути в діапазоні від 0% до 100%

# Проєктування архітектури програмного модуля

Клас ClassLab12\_Zaritsky матиме наступну структуру:

Приватні дані-члени:

* edgeLength: double — Довжина грані куба (м)
* string skinColor — Колір шкірки кавуна
* string fleshColor — Колір м'якоті кавуна
* double ripeness — Стиглість кавуна (%)

Публічні функції-члени:

* ClassLab12\_Zaritsky() — конструктор за замовчуванням
* ClassLab12\_Zaritsky(double edge, std::string skin, std::string flesh, double ripe) — конструктор з параметрами
* double getEdgeLength() — повертає довжину грані куба
* string getSkinColor() — повертає колір шкірки
* string getFleshColor() — повертає колір м'якоті
* double getRipeness() — повертає рівень стиглості (у відсотках)
* double getVolume() — повертає/обчислює об'єм куба
* void setEdgeLength(double edge) — встановлює довжину грані куба
* void setFleshColor(std::string flesh) — встановлює колір м’якоті
* void setRipeness(double ripe) — встановлює рівень стиглості

# Реалізація класу ClassLab12\_Zaritsky в заголовковому файлі

#ifndef MODULES\_ZARITSKY\_H

#define MODULES\_ZARITSKY\_H

#include <string>

#include <cmath>

#include <stdexcept>

// Клас, що представляє абстракцію кавуна кубічної форми

class ClassLab12\_Zaritsky {

private:

double edgeLength; // Довжина грані куба (м)

std::string skinColor; // Колір шкірки кавуна

std::string fleshColor; // Колір м'якоті кавуна

double ripeness; // Стиглість кавуна (%)

public:

// Конструктор за замовчуванням

ClassLab12\_Zaritsky();

ClassLab12\_Zaritsky(double edge, std::string skin, std::string flesh, double ripe);

// Отримання довжини грані куба

double getEdgeLength() const;

// Отримання кольору шкірки

std::string getSkinColor() const;

// Отримання кольору м'якоті

std::string getFleshColor() const;

// Отримання рівня стиглості

double getRipeness() const;

// Обчислення об'єму куба

double getVolume() const;

// Встановлення довжини грані куба

void setEdgeLength(double edge);

// Встановлення кольору шкірки

void setSkinColor(std::string skin);

// Встановлення кольору м'якоті

void setFleshColor(std::string flesh);

// Встановлення рівня стиглості

void setRipeness(double ripe);

};

inline ClassLab12\_Zaritsky::ClassLab12\_Zaritsky()

: edgeLength(0.2), skinColor("green"), fleshColor("red"), ripeness(90.0) {

}

inline ClassLab12\_Zaritsky::ClassLab12\_Zaritsky(double edge, std::string skin, std::string flesh, double ripe) {

setEdgeLength(edge);

setSkinColor(skin);

setFleshColor(flesh);

setRipeness(ripe);

}

inline double ClassLab12\_Zaritsky::getEdgeLength() const {

return edgeLength;

}

inline std::string ClassLab12\_Zaritsky::getSkinColor() const {

return skinColor;

}

inline std::string ClassLab12\_Zaritsky::getFleshColor() const {

return fleshColor;

}

inline double ClassLab12\_Zaritsky::getRipeness() const {

return ripeness;

}

inline double ClassLab12\_Zaritsky::getVolume() const {

return pow(edgeLength, 3);

}

inline void ClassLab12\_Zaritsky::setEdgeLength(double edge) {

if (edge <= 0) {

throw std::invalid\_argument("Довжина грані має бути додатним числом");

}

edgeLength = edge;

}

inline void ClassLab12\_Zaritsky::setSkinColor(std::string skin) {

skinColor = skin;

}

inline void ClassLab12\_Zaritsky::setFleshColor(std::string flesh) {

fleshColor = flesh;

}

inline void ClassLab12\_Zaritsky::setRipeness(double ripe) {

if (ripe < 0 || ripe > 100) {

throw std::invalid\_argument("Стиглість має бути в діапазоні від 0 до 100");

}

ripeness = ripe;

}

#endif // MODULES\_ZARITSKY\_H

# Аналіз і постановка задачі завдання 2 (додаток Teacher)

Мета: Розробити консольний додаток Teacher, який виконуватиме автоматизоване тестування класу ClassLab12\_Zaritsky та протоколюватиме результати.

Функціональні вимоги:

* Перевірка розташування файлу проєкту: Додаток має перевіряти, чи файл main.cpp під час компіляції знаходився у теці \Lab12\prj. Якщо умова не виконана, додаток повинен видати 100 звукових сигналів та записати рядок "Встановлені вимоги порядку виконання лабораторної роботи порушено!" у текстовий файл TestResults.txt.
* Unit-тестування класу ClassLab12\_Zaritsky: Якщо файл main.cpp знаходиться у \Lab12\prj, додаток створює об'єкт класу ClassLab12\_Zaritsky із заголовкового файлу ModulesZaritsky.h. Виконує unit-тестування цього об'єкта за тест-сьютами із теки \Lab12\TestSuite\. Протоколює результати тестування у текстовий файл \Lab12\TestSuiteTestResults.txt.

Формат тест-кейсів: Для додатку Teacher я розробив спеціальний формат тест-кейсів, де кожен кейс має наступну структуру:

Test Case ID: TC\_XXX

Action: Опис дії, що тестується

Expected Result: Очікуваний результат

Test Result: PASS/FAIL

Цей формат дозволяє легко парсити текстові файли з тест-кейсами та обробляти їх у коді. Кожен елемент розділяється новим рядком, а різні тест-кейси можуть бути розділені порожнім рядком для кращої читабельності.

Протокол читання тест-кейсів реалізовано в класі TestSuiteProcessor, який аналізує файл рядок за рядком, виявляє ключові слова і формує структури даних для подальшої обробки і виконання тестів.

# Аналіз вимог до програмного засобу Teacher, проєктування архітектури й детальне проєктування ПЗ

Аналіз вимог: Додаток Teacher повинен виконувати дві основні функції: перевірку структури файлів та автоматизоване тестування. Кожна з цих функцій має свої особливості та вимоги до реалізації.

Перевірка структури: Потребує доступу до файлової системи для визначення шляху до файлу main.cpp.

Тестування: Потребує читання тест-кейсів з файлів, створення об'єкта класу, виклику його методів та порівняння отриманих результатів з очікуваними. Це передбачає парсинг текстових файлів та динамічну роботу з об'єктами тестування.

Проєктування архітектури: Додаток Teacher буде мати модульну архітектуру, яка дозволить розділити логіку на окремі компоненти для кращої керованості та розширюваності.

Основні компоненти:

* Функція main: Точка входу в додаток. Відповідає за початкові перевірки (шлях компіляції) та координацію роботи інших компонентів.
* PathChecker (клас/функція): Відповідає за перевірку коректності розташування файлу main.cpp.
* Logger (клас): Відповідає за запис результатів у текстові файли (TestResults.txt та TestSuiteTestResults.txt) та виведення повідомлень користувачеві.
* TestCaseParser (локальний клас або функція): Відповідає за читання та парсинг тестових кейсів з текстових файлів. Буде реалізований як локальний клас або функція, як зазначено в методичних вказівках.
* UnitTestRunner (клас): Координує виконання тестів. Створює об'єкт класу ClassLab12\_Zaritsky, викликає методи тестування та порівнює результати.
* SoundPlayer (клас/функція): Відповідає за відтворення звукових сигналів.

# Вихідний код ModulesZaritskyi.h:

#ifndef MODULESZARITSKYI\_H\_INCLUDED

#define MODULESZARITSKYI\_H\_INCLUDED

#include <string>

#include <cmath>

#include <stdexcept>

struct Temperature {

double celsius;

double fahrenheit;

};

double s\_calculation(double x, double y, double z);

double calculateElectricityBill(double kWh);

Temperature calculateAverageTemperature(const double temps[12]);

int countBits(int N);

void processTask10\_1(const std::string &inputFileName, const std::string &outputFileName);

void processTask10\_2(const std::string &inputFileName);

void processTask10\_3(const std::string &outputFileName, double x, double y, double z, int b);

// Клас, що представляє абстракцію кавуна кубічної форми

class ClassLab12\_Zaritsky {

private:

double edgeLength; // Довжина грані куба (м)

std::string skinColor; // Колір шкірки кавуна

std::string fleshColor; // Колір м'якоті кавуна

double ripeness; // Стиглість кавуна (%)

public:

// Конструктор за замовчуванням

ClassLab12\_Zaritsky();

ClassLab12\_Zaritsky(double edge, std::string skin, std::string flesh, double ripe);

// Отримання довжини грані куба

double getEdgeLength() const;

// Отримання кольору шкірки

std::string getSkinColor() const;

// Отримання кольору м'якоті

std::string getFleshColor() const;

// Отримання рівня стиглості

double getRipeness() const;

// Обчислення об'єму куба

double getVolume() const;

// Встановлення довжини грані куба

void setEdgeLength(double edge);

// Встановлення кольору шкірки

void setSkinColor(std::string skin);

// Встановлення кольору м'якоті

void setFleshColor(std::string flesh);

// Встановлення рівня стиглості

void setRipeness(double ripe);

};

inline ClassLab12\_Zaritsky::ClassLab12\_Zaritsky()

: edgeLength(0.2), skinColor("green"), fleshColor("red"), ripeness(90.0) {

}

inline ClassLab12\_Zaritsky::ClassLab12\_Zaritsky(double edge, std::string skin, std::string flesh, double ripe) {

if (edge <= 0) {

throw std::invalid\_argument("Довжина грані має бути додатним числом");

}

if (ripe < 0 || ripe > 100) {

throw std::invalid\_argument("Стиглість має бути в діапазоні від 0 до 100");

}

edgeLength = edge;

skinColor = skin;

fleshColor = flesh;

ripeness = ripe;

}

inline double ClassLab12\_Zaritsky::getEdgeLength() const {

return edgeLength;

}

inline std::string ClassLab12\_Zaritsky::getSkinColor() const {

return skinColor;

}

inline std::string ClassLab12\_Zaritsky::getFleshColor() const {

return fleshColor;

}

inline double ClassLab12\_Zaritsky::getRipeness() const {

return ripeness;

}

inline double ClassLab12\_Zaritsky::getVolume() const {

return pow(edgeLength, 3);

}

inline void ClassLab12\_Zaritsky::setEdgeLength(double edge) {

if (edge <= 0) {

throw std::invalid\_argument("Довжина грані має бути додатним числом");

}

edgeLength = edge;

}

inline void ClassLab12\_Zaritsky::setSkinColor(std::string skin) {

skinColor = skin;

}

inline void ClassLab12\_Zaritsky::setFleshColor(std::string flesh) {

fleshColor = flesh;

}

inline void ClassLab12\_Zaritsky::setRipeness(double ripe) {

if (ripe < 0 || ripe > 100) {

throw std::invalid\_argument("Стиглість має бути в діапазоні від 0 до 100");

}

ripeness = ripe;

}

#endif // MODULESZARITSKYI\_H\_INCLUDED

# Вихідний код проєкту Teacher:

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <cstdlib>

#include <filesystem>

#include <sstream>

#include <cmath>

#include <thread>

#include <chrono>

#include "ModulesZaritskyi.h"

// Структура для представлення тест-кейса

struct TestCase {

std::string id;

std::string action;

std::string expectedResult;

std::string testResult;

};

// Локальний клас для читання і обробки тест-кейсів

class TestSuiteProcessor {

private:

std::vector<TestCase> testCases;

std::string testSuitePath;

std::string resultsPath;

public:

TestSuiteProcessor(const std::string& suitePath, const std::string& outputPath)

: testSuitePath(suitePath), resultsPath(outputPath) {}

// Читання тест-кейсів з файлу

bool readTestSuite() {

std::ifstream file(testSuitePath);

if (!file.is\_open()) {

std::cerr << "Не вдалося відкрити файл з тест-сьютом: " << testSuitePath << std::endl;

return false;

}

std::string line;

TestCase currentTest;

int state = 0; // 0 - очікування ID, 1 - очікування Action, 2 - очікування Expected Result, 3 - очікування Test Result

while (std::getline(file, line)) {

if (line.empty() || line[0] == '#') continue; // Пропускаємо порожні рядки та коментарі

if (line.find("Test Case ID:") != std::string::npos) {

if (state > 0) {

testCases.push\_back(currentTest);

currentTest = TestCase();

}

currentTest.id = line.substr(line.find(":") + 1);

// Видаляємо пробіли на початку та в кінці

size\_t start = currentTest.id.find\_first\_not\_of(" \t");

size\_t end = currentTest.id.find\_last\_not\_of(" \t");

if (start != std::string::npos) {

currentTest.id = currentTest.id.substr(start, end - start + 1);

}

state = 1;

} else if (line.find("Action:") != std::string::npos && state == 1) {

currentTest.action = line.substr(line.find(":") + 1);

state = 2;

} else if (line.find("Expected Result:") != std::string::npos && state == 2) {

currentTest.expectedResult = line.substr(line.find(":") + 1);

state = 3;

} else if (line.find("Test Result:") != std::string::npos && state == 3) {

currentTest.testResult = line.substr(line.find(":") + 1);

testCases.push\_back(currentTest);

currentTest = TestCase();

state = 0;

}

}

// Додати останній тест, якщо він є

if (state > 0) {

testCases.push\_back(currentTest);

}

file.close();

return true;

}

// Виконання тестів

bool runTests() {

std::ofstream results(resultsPath);

if (!results.is\_open()) {

std::cerr << "Не вдалося відкрити файл для запису результатів: " << resultsPath << std::endl;

return false;

}

results << "Результати модульного тестування класу ClassLab12\_Zaritsky\n";

results << "=====================================================\n\n";

int passCount = 0;

int failCount = 0;

// Групування тестів за категоріями

results << "1. ТЕСТИ КОНСТРУКТОРІВ\n";

results << "======================\n\n";

runTestGroup(results, "TC\_CONSTRUCTOR\_", passCount, failCount);

results << "\n2. ТЕСТИ ГЕТТЕРІВ\n";

results << "==================\n\n";

runTestGroup(results, "TC\_GET\_", passCount, failCount);

results << "\n3. ТЕСТИ СЕТТЕРІВ\n";

results << "==================\n\n";

runTestGroup(results, "TC\_SET\_", passCount, failCount);

results << "\n4. ІНТЕГРАЦІЙНІ ТЕСТИ\n";

results << "======================\n\n";

runTestGroup(results, "TC\_INTEGRATION\_", passCount, failCount);

results << "\nПідсумок тестування:\n";

results << "====================\n";

results << "Всього тестів: " << testCases.size() << "\n";

results << "Пройдено: " << passCount << "\n";

results << "Провалено: " << failCount << "\n";

results << "Відсоток успішності: " << (testCases.size() > 0 ? (passCount \* 100.0 / testCases.size()) : 0) << "%\n";

results << "Результат: " << (failCount == 0 ? "УСПІХ" : "ПОМИЛКА") << "\n";

results.close();

return failCount == 0;

}

private:

// Виконання групи тестів

void runTestGroup(std::ofstream& results, const std::string& prefix, int& passCount, int& failCount) {

for (const auto& test : testCases) {

if (test.id.find(prefix) != std::string::npos) {

results << "Test Case ID: " << test.id << "\n";

results << "Action: " << test.action << "\n";

results << "Expected Result: " << test.expectedResult << "\n";

bool passed = executeTest(test, results);

if (passed) {

results << "Test Result: PASS\n";

passCount++;

} else {

results << "Test Result: FAIL\n";

failCount++;

}

results << "\n";

}

}

}

// Виконання окремого тесту

bool executeTest(const TestCase& test, std::ofstream& results) {

try {

// ТЕСТИ КОНСТРУКТОРІВ

if (test.id == "TC\_CONSTRUCTOR\_DEFAULT\_001") {

ClassLab12\_Zaritsky watermelon;

return std::abs(watermelon.getEdgeLength() - 0.2) < 0.0001;

}

else if (test.id == "TC\_CONSTRUCTOR\_DEFAULT\_002") {

ClassLab12\_Zaritsky watermelon;

return watermelon.getSkinColor() == "green";

}

else if (test.id == "TC\_CONSTRUCTOR\_DEFAULT\_003") {

ClassLab12\_Zaritsky watermelon;

return watermelon.getFleshColor() == "red";

}

else if (test.id == "TC\_CONSTRUCTOR\_DEFAULT\_004") {

ClassLab12\_Zaritsky watermelon;

return std::abs(watermelon.getRipeness() - 90.0) < 0.0001;

}

else if (test.id == "TC\_CONSTRUCTOR\_PARAMS\_001") {

ClassLab12\_Zaritsky watermelon(0.5, "dark green", "pink", 85.5);

return std::abs(watermelon.getEdgeLength() - 0.5) < 0.0001 &&

watermelon.getSkinColor() == "dark green" &&

watermelon.getFleshColor() == "pink" &&

std::abs(watermelon.getRipeness() - 85.5) < 0.0001;

}

else if (test.id == "TC\_CONSTRUCTOR\_PARAMS\_002") {

ClassLab12\_Zaritsky watermelon(0.01, "", "", 0.0);

return std::abs(watermelon.getEdgeLength() - 0.01) < 0.0001 &&

watermelon.getSkinColor() == "" &&

watermelon.getFleshColor() == "" &&

std::abs(watermelon.getRipeness() - 0.0) < 0.0001;

}

else if (test.id == "TC\_CONSTRUCTOR\_PARAMS\_003") {

ClassLab12\_Zaritsky watermelon(10.0, "very long color name", "another long color", 100.0);

return std::abs(watermelon.getEdgeLength() - 10.0) < 0.0001 &&

watermelon.getSkinColor() == "very long color name" &&

watermelon.getFleshColor() == "another long color" &&

std::abs(watermelon.getRipeness() - 100.0) < 0.0001;

}

else if (test.id == "TC\_CONSTRUCTOR\_PARAMS\_004") {

try {

ClassLab12\_Zaritsky watermelon(-1.0, "green", "red", 50.0);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Довжина грані має бути додатним числом";

}

}

else if (test.id == "TC\_CONSTRUCTOR\_PARAMS\_005") {

try {

ClassLab12\_Zaritsky watermelon(0.0, "green", "red", 50.0);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Довжина грані має бути додатним числом";

}

}

else if (test.id == "TC\_CONSTRUCTOR\_PARAMS\_006") {

try {

ClassLab12\_Zaritsky watermelon(1.0, "green", "red", -10.0);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Стиглість має бути в діапазоні від 0 до 100";

}

}

else if (test.id == "TC\_CONSTRUCTOR\_PARAMS\_007") {

try {

ClassLab12\_Zaritsky watermelon(1.0, "green", "red", 150.0);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Стиглість має бути в діапазоні від 0 до 100";

}

}

// ТЕСТИ ГЕТТЕРІВ

else if (test.id == "TC\_GET\_EDGE\_001") {

ClassLab12\_Zaritsky watermelon;

return std::abs(watermelon.getEdgeLength() - 0.2) < 0.0001;

}

else if (test.id == "TC\_GET\_EDGE\_002") {

ClassLab12\_Zaritsky watermelon(1.5, "green", "red", 50.0);

return std::abs(watermelon.getEdgeLength() - 1.5) < 0.0001;

}

else if (test.id == "TC\_GET\_EDGE\_003") {

ClassLab12\_Zaritsky watermelon(0.001, "green", "red", 50.0);

return std::abs(watermelon.getEdgeLength() - 0.001) < 0.0001;

}

else if (test.id == "TC\_GET\_SKIN\_001") {

ClassLab12\_Zaritsky watermelon;

return watermelon.getSkinColor() == "green";

}

else if (test.id == "TC\_GET\_SKIN\_002") {

ClassLab12\_Zaritsky watermelon(1.0, "yellow", "red", 50.0);

return watermelon.getSkinColor() == "yellow";

}

else if (test.id == "TC\_GET\_SKIN\_003") {

ClassLab12\_Zaritsky watermelon(1.0, "", "red", 50.0);

return watermelon.getSkinColor() == "";

}

else if (test.id == "TC\_GET\_FLESH\_001") {

ClassLab12\_Zaritsky watermelon;

return watermelon.getFleshColor() == "red";

}

else if (test.id == "TC\_GET\_FLESH\_002") {

ClassLab12\_Zaritsky watermelon(1.0, "green", "white", 50.0);

return watermelon.getFleshColor() == "white";

}

else if (test.id == "TC\_GET\_FLESH\_003") {

ClassLab12\_Zaritsky watermelon(1.0, "green", "very bright crimson red", 50.0);

return watermelon.getFleshColor() == "very bright crimson red";

}

else if (test.id == "TC\_GET\_RIPENESS\_001") {

ClassLab12\_Zaritsky watermelon;

return std::abs(watermelon.getRipeness() - 90.0) < 0.0001;

}

else if (test.id == "TC\_GET\_RIPENESS\_002") {

ClassLab12\_Zaritsky watermelon(1.0, "green", "red", 0.0);

return std::abs(watermelon.getRipeness() - 0.0) < 0.0001;

}

else if (test.id == "TC\_GET\_RIPENESS\_003") {

ClassLab12\_Zaritsky watermelon(1.0, "green", "red", 100.0);

return std::abs(watermelon.getRipeness() - 100.0) < 0.0001;

}

else if (test.id == "TC\_GET\_RIPENESS\_004") {

ClassLab12\_Zaritsky watermelon(1.0, "green", "red", 55.75);

return std::abs(watermelon.getRipeness() - 55.75) < 0.0001;

}

// ТЕСТИ getVolume()

else if (test.id == "TC\_GET\_VOLUME\_001") {

ClassLab12\_Zaritsky watermelon(1.0, "green", "red", 50.0);

return std::abs(watermelon.getVolume() - 1.0) < 0.0001;

}

else if (test.id == "TC\_GET\_VOLUME\_002") {

ClassLab12\_Zaritsky watermelon(2.0, "green", "red", 50.0);

return std::abs(watermelon.getVolume() - 8.0) < 0.0001;

}

else if (test.id == "TC\_GET\_VOLUME\_003") {

ClassLab12\_Zaritsky watermelon(0.5, "green", "red", 50.0);

return std::abs(watermelon.getVolume() - 0.125) < 0.0001;

}

else if (test.id == "TC\_GET\_VOLUME\_004") {

ClassLab12\_Zaritsky watermelon;

return std::abs(watermelon.getVolume() - 0.008) < 0.0001;

}

else if (test.id == "TC\_GET\_VOLUME\_005") {

ClassLab12\_Zaritsky watermelon(3.0, "green", "red", 50.0);

return std::abs(watermelon.getVolume() - 27.0) < 0.0001;

}

else if (test.id == "TC\_GET\_VOLUME\_006") {

ClassLab12\_Zaritsky watermelon(0.1, "green", "red", 50.0);

return std::abs(watermelon.getVolume() - 0.001) < 0.0001;

}

// ТЕСТИ СЕТТЕРІВ

else if (test.id == "TC\_SET\_EDGE\_001") {

ClassLab12\_Zaritsky watermelon;

watermelon.setEdgeLength(1.5);

return std::abs(watermelon.getEdgeLength() - 1.5) < 0.0001;

}

else if (test.id == "TC\_SET\_EDGE\_002") {

ClassLab12\_Zaritsky watermelon;

watermelon.setEdgeLength(0.01);

return std::abs(watermelon.getEdgeLength() - 0.01) < 0.0001;

}

else if (test.id == "TC\_SET\_EDGE\_003") {

ClassLab12\_Zaritsky watermelon;

watermelon.setEdgeLength(10.0);

return std::abs(watermelon.getEdgeLength() - 10.0) < 0.0001;

}

else if (test.id == "TC\_SET\_EDGE\_FAIL\_001") {

ClassLab12\_Zaritsky watermelon;

try {

watermelon.setEdgeLength(-1.0);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Довжина грані має бути додатним числом";

}

}

else if (test.id == "TC\_SET\_EDGE\_FAIL\_002") {

ClassLab12\_Zaritsky watermelon;

try {

watermelon.setEdgeLength(0.0);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Довжина грані має бути додатним числом";

}

}

else if (test.id == "TC\_SET\_EDGE\_FAIL\_003") {

ClassLab12\_Zaritsky watermelon;

try {

watermelon.setEdgeLength(-100.5);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Довжина грані має бути додатним числом";

}

}

else if (test.id == "TC\_SET\_SKIN\_001") {

ClassLab12\_Zaritsky watermelon;

watermelon.setSkinColor("yellow");

return watermelon.getSkinColor() == "yellow";

}

else if (test.id == "TC\_SET\_SKIN\_002") {

ClassLab12\_Zaritsky watermelon;

watermelon.setSkinColor("");

return watermelon.getSkinColor() == "";

}

else if (test.id == "TC\_SET\_SKIN\_003") {

ClassLab12\_Zaritsky watermelon;

watermelon.setSkinColor("striped green and yellow");

return watermelon.getSkinColor() == "striped green and yellow";

}

else if (test.id == "TC\_SET\_SKIN\_004") {

ClassLab12\_Zaritsky watermelon;

watermelon.setSkinColor("green");

watermelon.setSkinColor("yellow");

return watermelon.getSkinColor() == "yellow";

}

else if (test.id == "TC\_SET\_FLESH\_001") {

ClassLab12\_Zaritsky watermelon;

watermelon.setFleshColor("white");

return watermelon.getFleshColor() == "white";

}

else if (test.id == "TC\_SET\_FLESH\_002") {

ClassLab12\_Zaritsky watermelon;

watermelon.setFleshColor("pink");

return watermelon.getFleshColor() == "pink";

}

else if (test.id == "TC\_SET\_FLESH\_003") {

ClassLab12\_Zaritsky watermelon;

watermelon.setFleshColor("");

return watermelon.getFleshColor() == "";

}

else if (test.id == "TC\_SET\_FLESH\_004") {

ClassLab12\_Zaritsky watermelon;

watermelon.setFleshColor("deep crimson red with black seeds");

return watermelon.getFleshColor() == "deep crimson red with black seeds";

}

else if (test.id == "TC\_SET\_RIPENESS\_001") {

ClassLab12\_Zaritsky watermelon;

watermelon.setRipeness(50.0);

return std::abs(watermelon.getRipeness() - 50.0) < 0.0001;

}

else if (test.id == "TC\_SET\_RIPENESS\_002") {

ClassLab12\_Zaritsky watermelon;

watermelon.setRipeness(0.0);

return std::abs(watermelon.getRipeness() - 0.0) < 0.0001;

}

else if (test.id == "TC\_SET\_RIPENESS\_003") {

ClassLab12\_Zaritsky watermelon;

watermelon.setRipeness(100.0);

return std::abs(watermelon.getRipeness() - 100.0) < 0.0001;

}

else if (test.id == "TC\_SET\_RIPENESS\_004") {

ClassLab12\_Zaritsky watermelon;

watermelon.setRipeness(75.25);

return std::abs(watermelon.getRipeness() - 75.25) < 0.0001;

}

else if (test.id == "TC\_SET\_RIPENESS\_FAIL\_001") {

ClassLab12\_Zaritsky watermelon;

try {

watermelon.setRipeness(-1.0);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Стиглість має бути в діапазоні від 0 до 100";

}

}

else if (test.id == "TC\_SET\_RIPENESS\_FAIL\_002") {

ClassLab12\_Zaritsky watermelon;

try {

watermelon.setRipeness(101.0);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Стиглість має бути в діапазоні від 0 до 100";

}

}

else if (test.id == "TC\_SET\_RIPENESS\_FAIL\_003") {

ClassLab12\_Zaritsky watermelon;

try {

watermelon.setRipeness(-50.5);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Стиглість має бути в діапазоні від 0 до 100";

}

}

else if (test.id == "TC\_SET\_RIPENESS\_FAIL\_004") {

ClassLab12\_Zaritsky watermelon;

try {

watermelon.setRipeness(200.0);

return false;

} catch (const std::invalid\_argument& e) {

return std::string(e.what()) == "Стиглість має бути в діапазоні від 0 до 100";

}

}

// ІНТЕГРАЦІЙНІ ТЕСТИ

else if (test.id == "TC\_INTEGRATION\_001") {

ClassLab12\_Zaritsky watermelon;

watermelon.setEdgeLength(2.5);

watermelon.setSkinColor("dark green");

watermelon.setFleshColor("bright red");

watermelon.setRipeness(85.0);

return std::abs(watermelon.getEdgeLength() - 2.5) < 0.0001 &&

watermelon.getSkinColor() == "dark green" &&

watermelon.getFleshColor() == "bright red" &&

std::abs(watermelon.getRipeness() - 85.0) < 0.0001;

}

else if (test.id == "TC\_INTEGRATION\_002") {

ClassLab12\_Zaritsky watermelon;

watermelon.setEdgeLength(2.0);

double volume1 = watermelon.getVolume();

watermelon.setEdgeLength(3.0);

double volume2 = watermelon.getVolume();

return std::abs(volume1 - 8.0) < 0.0001 && std::abs(volume2 - 27.0) < 0.0001;

}

else if (test.id == "TC\_INTEGRATION\_003") {

ClassLab12\_Zaritsky watermelon;

watermelon.setSkinColor("green");

watermelon.setSkinColor("yellow");

watermelon.setSkinColor("red");

return watermelon.getSkinColor() == "red";

}

else {

results << "Невідомий ідентифікатор тест-кейса: " << test.id << "\n";

return false;

}

}

catch (const std::exception& e) {

results << "Виняток під час виконання тесту: " << e.what() << "\n";

return false;

}

catch (...) {

results << "Невідомий виняток під час виконання тесту\n";

return false;

}

}

};

int main() {

system("chcp 65001 && cls");

// Перевірка коректності розташування файлів

std::filesystem::path currentPath = std::filesystem::current\_path();

std::string folderName = currentPath.filename().string();

if (folderName != "prj") {

// Вимога порушена, файл проєкту не в \Lab12\prj

std::ofstream errorFile("TestResults.txt");

if (errorFile.is\_open()) {

errorFile << "Встановлені вимоги порядку виконання лабораторної роботи порушено!" << std::endl;

errorFile.close();

}

// Вивести 100 звукових сигналів

for (int i = 0; i < 100; ++i) {

std::cout << '\a'; // Звуковий сигнал

std::this\_thread::sleep\_for(std::chrono::milliseconds(50)); // Затримка між сигналами

}

return 1;

}

// Шляхи до файлів

std::string testSuitePath = "../TestSuite/TestSuite.txt";

std::string resultsPath = "../TestSuite/TestResults.txt";

// Створення процесора тест-сьютів

TestSuiteProcessor processor(testSuitePath, resultsPath);

// Читання тест-сьютів

if (!processor.readTestSuite()) {

std::cerr << "Помилка читання тест-сьютів!" << std::endl;

return 1;

}

// Запуск тестів

bool success = processor.runTests();

std::cout << "Тестування " << (success ? "успішно завершено" : "завершено з помилками") << std::endl;

std::cout << "Результати записані у файл: " << resultsPath << std::endl;

return success ? 0 : 1;

}

# Вміст файлу TestResults.txt:

Результати модульного тестування класу ClassLab12\_Zaritsky

=====================================================

1. ТЕСТИ КОНСТРУКТОРІВ

======================

Test Case ID: TC\_CONSTRUCTOR\_DEFAULT\_001

Action: Створення об'єкта за замовчуванням та перевірка ініціалізації edgeLength

Expected Result: edgeLength = 0.2

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_DEFAULT\_002

Action: Створення об'єкта за замовчуванням та перевірка ініціалізації skinColor

Expected Result: skinColor = "green"

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_DEFAULT\_003

Action: Створення об'єкта за замовчуванням та перевірка ініціалізації fleshColor

Expected Result: fleshColor = "red"

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_DEFAULT\_004

Action: Створення об'єкта за замовчуванням та перевірка ініціалізації ripeness

Expected Result: ripeness = 90.0

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_PARAMS\_001

Action: Створення об'єкта з параметрами (0.5, "dark green", "pink", 85.5) та перевірка всіх атрибутів

Expected Result: edgeLength=0.5, skinColor="dark green", fleshColor="pink", ripeness=85.5

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_PARAMS\_002

Action: Створення об'єкта з мінімальними валідними параметрами (0.01, "", "", 0.0)

Expected Result: edgeLength=0.01, skinColor="", fleshColor="", ripeness=0.0

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_PARAMS\_003

Action: Створення об'єкта з максимальними валідними параметрами (10.0, "very long color name", "another long color", 100.0)

Expected Result: edgeLength=10.0, skinColor="very long color name", fleshColor="another long color", ripeness=100.0

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_PARAMS\_004

Action: Спроба створення об'єкта з негативною довжиною грані (-1.0, "green", "red", 50.0)

Expected Result: Виняток std::invalid\_argument з повідомленням "Довжина грані має бути додатним числом"

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_PARAMS\_005

Action: Спроба створення об'єкта з нульовою довжиною грані (0.0, "green", "red", 50.0)

Expected Result: Виняток std::invalid\_argument з повідомленням "Довжина грані має бути додатним числом"

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_PARAMS\_006

Action: Спроба створення об'єкта з негативною стиглістю (1.0, "green", "red", -10.0)

Expected Result: Виняток std::invalid\_argument з повідомленням "Стиглість має бути в діапазоні від 0 до 100"

Test Result: PASS

Test Case ID: TC\_CONSTRUCTOR\_PARAMS\_007

Action: Спроба створення об'єкта зі стиглістю більше 100 (1.0, "green", "red", 150.0)

Expected Result: Виняток std::invalid\_argument з повідомленням "Стиглість має бути в діапазоні від 0 до 100"

Test Result: PASS

2. ТЕСТИ ГЕТТЕРІВ

==================

Test Case ID: TC\_GET\_EDGE\_001

Action: Створення об'єкта за замовчуванням та виклик getEdgeLength()

Expected Result: Повернення значення 0.2

Test Result: PASS

Test Case ID: TC\_GET\_EDGE\_002

Action: Створення об'єкта з edgeLength=1.5 та виклик getEdgeLength()

Expected Result: Повернення значення 1.5

Test Result: PASS

Test Case ID: TC\_GET\_EDGE\_003

Action: Створення об'єкта з дуже малим значенням edgeLength=0.001 та виклик getEdgeLength()

Expected Result: Повернення значення 0.001

Test Result: PASS

Test Case ID: TC\_GET\_SKIN\_001

Action: Створення об'єкта за замовчуванням та виклик getSkinColor()

Expected Result: Повернення рядка "green"

Test Result: PASS

Test Case ID: TC\_GET\_SKIN\_002

Action: Створення об'єкта з skinColor="yellow" та виклик getSkinColor()

Expected Result: Повернення рядка "yellow"

Test Result: PASS

Test Case ID: TC\_GET\_SKIN\_003

Action: Створення об'єкта з пустим skinColor="" та виклик getSkinColor()

Expected Result: Повернення порожнього рядка ""

Test Result: PASS

Test Case ID: TC\_GET\_FLESH\_001

Action: Створення об'єкта за замовчуванням та виклик getFleshColor()

Expected Result: Повернення рядка "red"

Test Result: PASS

Test Case ID: TC\_GET\_FLESH\_002

Action: Створення об'єкта з fleshColor="white" та виклик getFleshColor()

Expected Result: Повернення рядка "white"

Test Result: PASS

Test Case ID: TC\_GET\_FLESH\_003

Action: Створення об'єкта з довгим fleshColor="very bright crimson red" та виклик getFleshColor()

Expected Result: Повернення рядка "very bright crimson red"

Test Result: PASS

Test Case ID: TC\_GET\_RIPENESS\_001

Action: Створення об'єкта за замовчуванням та виклик getRipeness()

Expected Result: Повернення значення 90.0

Test Result: PASS

Test Case ID: TC\_GET\_RIPENESS\_002

Action: Створення об'єкта з ripeness=0.0 та виклик getRipeness()

Expected Result: Повернення значення 0.0

Test Result: PASS

Test Case ID: TC\_GET\_RIPENESS\_003

Action: Створення об'єкта з ripeness=100.0 та виклик getRipeness()

Expected Result: Повернення значення 100.0

Test Result: PASS

Test Case ID: TC\_GET\_RIPENESS\_004

Action: Створення об'єкта з ripeness=55.75 та виклик getRipeness()

Expected Result: Повернення значення 55.75

Test Result: PASS

Test Case ID: TC\_GET\_VOLUME\_001

Action: Створення об'єкта з edgeLength=1.0 та виклик getVolume()

Expected Result: Повернення значення 1.0 (1? = 1)

Test Result: PASS

Test Case ID: TC\_GET\_VOLUME\_002

Action: Створення об'єкта з edgeLength=2.0 та виклик getVolume()

Expected Result: Повернення значення 8.0 (2? = 8)

Test Result: PASS

Test Case ID: TC\_GET\_VOLUME\_003

Action: Створення об'єкта з edgeLength=0.5 та виклик getVolume()

Expected Result: Повернення значення 0.125 (0.5? = 0.125)

Test Result: PASS

Test Case ID: TC\_GET\_VOLUME\_004

Action: Створення об'єкта за замовчуванням (edgeLength=0.2) та виклик getVolume()

Expected Result: Повернення значення 0.008 (0.2? = 0.008)

Test Result: PASS

Test Case ID: TC\_GET\_VOLUME\_005

Action: Створення об'єкта з edgeLength=3.0 та виклик getVolume()

Expected Result: Повернення значення 27.0 (3? = 27)

Test Result: PASS

Test Case ID: TC\_GET\_VOLUME\_006

Action: Створення об'єкта з edgeLength=0.1 та виклик getVolume()

Expected Result: Повернення значення 0.001 (0.1? = 0.001)

Test Result: PASS

3. ТЕСТИ СЕТТЕРІВ

==================

Test Case ID: TC\_SET\_EDGE\_001

Action: Створення об'єкта та виклик setEdgeLength(1.5)

Expected Result: getEdgeLength() повертає 1.5

Test Result: PASS

Test Case ID: TC\_SET\_EDGE\_002

Action: Створення об'єкта та виклик setEdgeLength(0.01)

Expected Result: getEdgeLength() повертає 0.01

Test Result: PASS

Test Case ID: TC\_SET\_EDGE\_003

Action: Створення об'єкта та виклик setEdgeLength(10.0)

Expected Result: getEdgeLength() повертає 10.0

Test Result: PASS

Test Case ID: TC\_SET\_EDGE\_FAIL\_001

Action: Створення об'єкта та виклик setEdgeLength(-1.0)

Expected Result: Виняток std::invalid\_argument з повідомленням "Довжина грані має бути додатним числом"

Test Result: PASS

Test Case ID: TC\_SET\_EDGE\_FAIL\_002

Action: Створення об'єкта та виклик setEdgeLength(0.0)

Expected Result: Виняток std::invalid\_argument з повідомленням "Довжина грані має бути додатним числом"

Test Result: PASS

Test Case ID: TC\_SET\_EDGE\_FAIL\_003

Action: Створення об'єкта та виклик setEdgeLength(-100.5)

Expected Result: Виняток std::invalid\_argument з повідомленням "Довжина грані має бути додатним числом"

Test Result: PASS

Test Case ID: TC\_SET\_SKIN\_001

Action: Створення об'єкта та виклик setSkinColor("yellow")

Expected Result: getSkinColor() повертає "yellow"

Test Result: PASS

Test Case ID: TC\_SET\_SKIN\_002

Action: Створення об'єкта та виклик setSkinColor("")

Expected Result: getSkinColor() повертає ""

Test Result: PASS

Test Case ID: TC\_SET\_SKIN\_003

Action: Створення об'єкта та виклик setSkinColor("striped green and yellow")

Expected Result: getSkinColor() повертає "striped green and yellow"

Test Result: PASS

Test Case ID: TC\_SET\_SKIN\_004

Action: Створення об'єкта та послідовні виклики setSkinColor("green"), setSkinColor("yellow")

Expected Result: getSkinColor() повертає "yellow"

Test Result: PASS

Test Case ID: TC\_SET\_FLESH\_001

Action: Створення об'єкта та виклик setFleshColor("white")

Expected Result: getFleshColor() повертає "white"

Test Result: PASS

Test Case ID: TC\_SET\_FLESH\_002

Action: Створення об'єкта та виклик setFleshColor("pink")

Expected Result: getFleshColor() повертає "pink"

Test Result: PASS

Test Case ID: TC\_SET\_FLESH\_003

Action: Створення об'єкта та виклик setFleshColor("")

Expected Result: getFleshColor() повертає ""

Test Result: PASS

Test Case ID: TC\_SET\_FLESH\_004

Action: Створення об'єкта та виклик setFleshColor("deep crimson red with black seeds")

Expected Result: getFleshColor() повертає "deep crimson red with black seeds"

Test Result: PASS

Test Case ID: TC\_SET\_RIPENESS\_001

Action: Створення об'єкта та виклик setRipeness(50.0)

Expected Result: getRipeness() повертає 50.0

Test Result: PASS

Test Case ID: TC\_SET\_RIPENESS\_002

Action: Створення об'єкта та виклик setRipeness(0.0)

Expected Result: getRipeness() повертає 0.0

Test Result: PASS

Test Case ID: TC\_SET\_RIPENESS\_003

Action: Створення об'єкта та виклик setRipeness(100.0)

Expected Result: getRipeness() повертає 100.0

Test Result: PASS

Test Case ID: TC\_SET\_RIPENESS\_004

Action: Створення об'єкта та виклик setRipeness(75.25)

Expected Result: getRipeness() повертає 75.25

Test Result: PASS

Test Case ID: TC\_SET\_RIPENESS\_FAIL\_001

Action: Створення об'єкта та виклик setRipeness(-1.0)

Expected Result: Виняток std::invalid\_argument з повідомленням "Стиглість має бути в діапазоні від 0 до 100"

Test Result: PASS

Test Case ID: TC\_SET\_RIPENESS\_FAIL\_002

Action: Створення об'єкта та виклик setRipeness(101.0)

Expected Result: Виняток std::invalid\_argument з повідомленням "Стиглість має бути в діапазоні від 0 до 100"

Test Result: PASS

Test Case ID: TC\_SET\_RIPENESS\_FAIL\_003

Action: Створення об'єкта та виклик setRipeness(-50.5)

Expected Result: Виняток std::invalid\_argument з повідомленням "Стиглість має бути в діапазоні від 0 до 100"

Test Result: PASS

Test Case ID: TC\_SET\_RIPENESS\_FAIL\_004

Action: Створення об'єкта та виклик setRipeness(200.0)

Expected Result: Виняток std::invalid\_argument з повідомленням "Стиглість має бути в діапазоні від 0 до 100"

Test Result: PASS

4. ІНТЕГРАЦІЙНІ ТЕСТИ

======================

Test Case ID: TC\_INTEGRATION\_001

Action: Створення об'єкта, встановлення всіх атрибутів через сеттери, перевірка через геттери

Expected Result: Всі значення встановлені коректно

Test Result: PASS

Test Case ID: TC\_INTEGRATION\_002

Action: Створення об'єкта, зміна edgeLength та перевірка зміни getVolume()

Expected Result: Об'єм змінюється відповідно до нового значення довжини грані

Test Result: PASS

Test Case ID: TC\_INTEGRATION\_003

Action: Послідовна зміна одного атрибута кілька разів та перевірка збереження останнього значення

Expected Result: Зберігається тільки останнє встановлене значення

Test Result: PASS

Підсумок тестування:

====================

Всього тестів: 55

Пройдено: 55

Провалено: 0

Відсоток успішності: 100%

Результат: УСПІХ

# Аргументи досягнення мети:

1. **Набуття навичок концептуалізації предметної області** - я навчився виділяти ключові сутності та їх взаємозв'язки з реального світу для подальшого програмування.
2. **Розвиток аналітичного мислення** - процес об'єктного аналізу вимагає системного підходу до декомпозиції складних задач на простіші компоненти.
3. **Формування вмінь абстрагування** - здатність виділяти суттєві характеристики об'єктів, ігноруючи несуттєві деталі.
4. **Розуміння принципів інкапсуляції** - вивчення способів приховування внутрішньої реалізації та надання контрольованого доступу до даних.
5. **Освоєння методології проєктування програмних систем** - застосування структурованого підходу до створення програмного забезпечення.
6. **Розвиток навичок моделювання** - створення формальних моделей реальних об'єктів у вигляді класів С++.
7. **Вивчення принципів визначення інтерфейсів** - розуміння того, як правильно описувати зовнішню поведінку об'єктів.
8. **Формування розуміння зв’язків між сутностями** - аналіз взаємодії різних компонентів системи.
9. **Набуття досвіду декомпозиції складних систем** - поділ великих задач на керовані частини.
10. **Розвиток навичок документування проєктних рішень** - фіксація артефактів аналізу та проєктування.
11. **Вивчення стандартів ISO/IEC 12207** - ознайомлення з міжнародними стандартами розробки ПЗ.
12. **Формування системного мислення** - розуміння програми як цілісної системи взаємопов'язаних компонентів.
13. **Розвиток навичок вимогового аналізу** - вміння збирати, аналізувати та специфікувати вимоги до програмного модуля.
14. **Освоєння методів архітектурного проєктування** - планування загальної структури програмного продукту.
15. **Вивчення принципів детального проєктування** - конкретизація архітектурних рішень до рівня реалізації.
16. **Формування навичок створення специфікацій** - документування технічних вимог та рішень.
17. **Розуміння життєвого циклу розробки ПЗ** - усвідомлення етапів від аналізу до тестування.
18. **Набуття досвіду роботи з проєктною документацією** - створення та ведення технічної документації.
19. **Розвиток навичок критичного мислення** - оцінка різних варіантів проєктних рішень.
20. **Формування розуміння якості програмного коду** - усвідомлення важливості читабельності, підтримуваності та розширюваності.
21. **Освоєння синтаксису класів С++** - вивчення правил оголошення та визначення класів.
22. **Розуміння концепції конструкторів** - навчання ініціалізації об'єктів при їх створенні.
23. **Вивчення рівнів доступу (public/private)** - контроль видимості членів класу.
24. **Освоєння техніки перевантаження функцій** - створення функцій з однаковими іменами, але різними параметрами.
25. **Вивчення параметрів за замовчуванням** - спрощення використання функцій з опціональними параметрами.
26. **Розуміння оператора розширення області видимості** - правильне кваліфікування членів класу.
27. **Освоєння принципів організації коду** - розділення оголошень та визначень між заголовковими та вихідними файлами.
28. **Вивчення техніки створення заголовкових файлів** - структурування інтерфейсів класів.
29. **Розуміння різниці між** struct **та** class - вибір відповідного типу для конкретних задач.
30. **Освоєння роботи з об'єктами класів** - створення екземплярів та виклик їх методів.
31. **Вивчення принципів інкапсуляції даних** - приховування внутрішньої реалізації.
32. **Розуміння концепції методів класу** - функції, що визначають поведінку об'єктів.
33. **Освоєння техніки доступу до членів об'єкта** - використання операторів . та ->.
34. **Вивчення правил іменування у С++** - дотримання конвенцій найменування.
35. **Розуміння області видимості змінних** - контроль доступу до даних на різних рівнях.
36. **Освоєння принципів модульності** - створення самодостатніх програмних компонентів.
37. **Вивчення техніки компіляції проєктів** - розуміння процесу збірки програм.
38. **Розуміння залежностей між файлами** - управління включеннями заголовкових файлів.
39. **Освоєння роботи з препроцесором** - використання директив препроцесора.
40. **Вивчення принципів створення АТД** - реалізація абстрактних типів даних засобами С++.
41. **Розуміння важливості модульного тестування** - перевірка коректності роботи окремих компонентів.
42. **Освоєння методології unit-тестування** - систематичний підхід до тестування програмних модулів.
43. **Вивчення структури тест-кейсів** - формалізація тестових сценаріїв.
44. **Розуміння концепції тест-сьютів** - організація наборів тестів.
45. **Освоєння техніки створення тестових даних** - підготовка вхідних параметрів для тестування.
46. **Вивчення принципів верифікації результатів** - порівняння отриманих та очікуваних результатів.
47. **Розуміння важливості покриття коду тестами** - забезпечення перевірки всіх шляхів виконання.
48. **Освоєння автоматизації тестування** - створення програмних засобів для тестування.
49. **Вивчення техніки протоколювання результатів** - фіксація результатів тестування у файлах.
50. **Розуміння ролі регресійного тестування** - перевірка збереження функціональності після змін.
51. **Освоєння принципів тестування граничних значень** - перевірка поведінки на межах діапазонів.
52. **Вивчення техніки тестування помилкових ситуацій** - перевірка обробки некоректних даних.
53. **Розуміння важливості системного тестування** - перевірка роботи програми в цілому.
54. **Освоєння методів валідації програмного коду** - перевірка відповідності вимогам.
55. **Вивчення принципів забезпечення якості ПЗ** - комплексний підхід до контролю якості.
56. **Розуміння ролі тестової документації** - важливість документування тестових процедур.
57. **Освоєння техніки аналізу результатів тестування** - інтерпретація отриманих даних.
58. **Вивчення методів виявлення та локалізації помилок** - ефективний пошук дефектів.
59. **Розуміння принципів тестування інтерфейсів** - перевірка взаємодії між компонентами.
60. **Освоєння культури тестування** - формування відповідального ставлення до якості коду.
61. **Освоєння IDE Code::Blocks** - ефективне використання інтегрованого середовища розробки.
62. **Вивчення кросплатформової розробки** - створення ПЗ для різних операційних систем.
63. **Розуміння процесу компіляції** - перетворення вихідного коду у виконуваний файл.
64. **Освоєння роботи з проєктами** - організація файлів та ресурсів розробки.
65. **Вивчення системи контролю версій Git** - управління змінами у коді.
66. **Розуміння структури репозиторію** - організація файлів проєкту.
67. **Освоєння роботи з GitHub** - використання хмарного сховища коду.
68. **Вивчення принципів організації файлової структури** - логічне розміщення компонентів проєкту.
69. **Розуміння важливості резервного копіювання** - збереження результатів роботи.
70. **Освоєння роботи з текстовими редакторами** - ефективне редагування коду та документації.
71. **Вивчення налаштування середовища розробки** - оптимізація робочого процесу.
72. **Розуміння процесу налагодження** - виявлення та усунення помилок у коді.
73. **Освоєння роботи з консольними додатками** - створення програм командного рядка.
74. **Вивчення принципів розгортання ПЗ** - підготовка програм до використання.
75. **Розуміння важливості документації коду** - створення коментарів та технічних описів.
76. **Освоєння техніки профілювання продуктивності** - оптимізація швидкодії програм.
77. **Вивчення принципів командної розробки** - співпраця над спільними проєктами.
78. **Розуміння стандартів кодування** - дотримання узгоджених правил оформлення.
79. **Освоєння автоматизації збірки** - використання скриптів для компіляції.
80. **Вивчення інтеграційних процесів** - об'єднання різних компонентів системи.
81. **Формування технічної культури програміста** - дотримання професійних стандартів.
82. **Розвиток навичок самостійної роботи** - здатність вирішувати задачі без постійного керівництва.
83. **Вивчення методів оцінки складності алгоритмів** - розуміння ефективності рішень.
84. **Освоєння принципів рефакторингу коду** - покращення структури без зміни функціональності.
85. **Формування розуміння архітектурних патернів** - використання перевірених рішень.
86. **Розвиток навичок технічного письма** - створення якісної документації.
87. **Вивчення методів презентації результатів** - ефективне представлення роботи.
88. **Освоєння принципів код-рев'ю** - взаємна перевірка якості коду.
89. **Формування розуміння етики програмування** - відповідальне використання технологій.
90. **Розвиток навичок критичного аналізу** - об'єктивна оцінка власних рішень.
91. **Вивчення методів оптимізації пам'яті** - ефективне використання ресурсів.
92. **Освоєння принципів портабельності коду** - створення переносимих рішень.
93. **Формування розуміння безпеки ПЗ** - врахування аспектів інформаційної безпеки.
94. **Розвиток навичок планування проєктів** - організація робочого процесу.
95. **Вивчення методів управління складністю** - контроль зростання складності системи.
96. **Освоєння принципів повторного використання коду** - створення багаторазових компонентів.
97. **Формування розуміння взаємодії з користувачем** - проєктування зручних інтерфейсів.
98. **Розвиток навичок технічного менеджменту** - управління технічними проєктами.
99. **Вивчення тенденцій розвитку технологій** - орієнтування у сучасних підходах до програмування.
100. **Формування фундаменту для подальшого навчання** - створення базових знань для вивчення більш складних концепцій об'єктно-орієнтованого програмування, патернів проєктування та архітектури програмних систем.

# Відповіді на контрольні запитання:

## 1. Результати виконання концептуалізації предметної області, об'єктного аналізу та визначення інтерфейсів

**Результатами концептуалізації предметної області є:**

* Ідентифіковані ключові сутності та об'єкти предметної області
* Визначені атрибути (властивості) кожної сутності
* Встановлені зв'язки та відношення між сутностями
* Створена концептуальна модель предметної області

**Результатами об'єктного аналізу є:**

* Виділені класи об'єктів з їх відповідальностями
* Визначені методи (операції) для кожного класу
* Встановлена ієрархія класів та їх взаємодія
* Створені діаграми класів та їх взаємозв'язків
* Специфіковані вимоги до поведінки об'єктів

**Результатами визначення інтерфейсів є:**

* Формальні специфікації публічних методів класів
* Визначені сигнатури функцій (назва, параметри, тип повернення)
* Встановлені контракти взаємодії між об'єктами
* Документація поведінки кожного методу

## 2. Зв'язок між процесами концептуалізації, об'єктного аналізу та визначення інтерфейсів

Ці процеси утворюють **послідовний ланцюжок уточнення та деталізації:**

1. **Концептуалізація** → створює загальне розуміння предметної області
2. **Об'єктний аналіз** → деталізує концепції до рівня програмних об'єктів
3. **Визначення інтерфейсів** → формалізує способи взаємодії з об'єктами

**Характер зв'язку:**

* **Послідовність:** кожен наступний процес базується на результатах попереднього
* **Ітеративність:** може виникати необхідність повернення до попередніх етапів для уточнення
* **Взаємозалежність:** зміни на одному рівні впливають на інші рівні
* **Уточнення:** кожен етап додає більше деталей та формальності

## 3. Критерії вибору між struct та class у С++

Використовувати **struct**, коли:

* **Прості дані:** структура містить переважно публічні дані без складної логіки
* **Пасивні сутності:** об'єкт служить контейнером для групування пов'язаних даних
* **Мінімальна поведінка:** відсутні або мінімальні методи обробки даних
* **Сумісність з С:** потрібна сумісність з кодом на мові С
* **Простота використання:** всі члени за замовчуванням публічні

Використовувати **class**, коли:

* **Інкапсуляція даних:** потрібно приховати внутрішню реалізацію
* **Активні об'єкти:** сутність має складну поведінку та методи
* **Контроль доступу:** необхідні різні рівні доступу (private, protected, public)
* **Абстракція:** реалізується повноцінний абстрактний тип даних
* **ООП-підхід:** застосовуються принципи об'єктно-орієнтованого програмування

## 4. Інтерфейс класу в програмуванні

**Інтерфейс класу** — це множина публічних методів та властивостей, через які зовнішній код може взаємодіяти з об'єктами цього класу.

**Складові інтерфейсу:**

* **Публічні методи** — функції, доступні для виклику ззовні
* **Публічні атрибути** — дані, доступні для читання/запису (рідко використовується)
* **Конструктори** — методи створення та ініціалізації об'єктів
* **Деструктор** — метод знищення об'єкта

**Характеристики якісного інтерфейсу:**

* **Повнота:** забезпечує всі необхідні операції
* **Мінімальність:** не містить зайвих методів
* **Консистентність:** узгоджене іменування та поведінка
* **Зрозумілість:** інтуїтивно зрозумілі назви та логіка

## 5. Чому конструктор не може бути з закритим доступом

**Технічні причини:**

* Конструктор з private доступом **не може бути викликаний** ззовні класу
* Це **унеможливлює створення об'єктів** класу звичайним способом
* Порушується **основна мета конструктора** — ініціалізація нових об'єктів

**Виключення (спеціальні випадки):**

* **Singleton pattern:** приватний конструктор використовується для обмеження створення єдиного екземпляра
* **Factory methods:** об'єкти створюються через статичні методи класу
* **Статичні класи:** класи, що містять тільки статичні методи

**Загальне правило:** для звичайних класів конструктор повинен бути public або protected (для успадкування).

## 6. Порівняльний аналіз перевантажених функцій та функцій з параметрами за замовчуванням

Перевантажені функції

**Переваги:**

* **Гнучкість:** кожна версія може мати унікальну логіку
* **Оптимізація:** можна оптимізувати кожну версію окремо
* **Ясність:** явно видно всі варіанти використання
* **Контроль типів:** строга типізація параметрів

**Недоліки:**

* **Дублювання коду:** можливе повторення логіки
* **Складність підтримки:** більше функцій для модифікації
* **Збільшення розміру:** більше скомпільованого коду

Функції з параметрами за замовчуванням

**Переваги:**

* **Компактність:** одна функція замість декількох
* **Простота підтримки:** зміни в одному місці
* **Меншій розмір коду:** менше дублювання
* **Зручність використання:** можна викликати з різною кількістю параметрів

**Недоліки:**

* **Обмежена гнучкість:** однакова логіка для всіх варіантів
* **Можливі помилки:** неочікувані значення за замовчуванням
* **Обмеження порядку:** параметри за замовчуванням тільки в кінці

## 7. Оператори доступу до членів об'єктів класу

Оператор крапка (.)

MyClass obj;

obj.publicMethod(); // Виклик методу

obj.publicVariable = 5; // Доступ до змінної

**Використання:** для доступу до членів об'єкта **за значенням**

Оператор стрілка (->)

MyClass\* ptr = new MyClass();

ptr->publicMethod(); // Виклик методу

ptr->publicVariable = 5; // Доступ до змінної

**Використання:** для доступу до членів об'єкта **за вказівником**

Оператор розіменування з крапкою

MyClass\* ptr = new MyClass();

(\*ptr).publicMethod(); // Еквівалент ptr->publicMethod()

**Використання:** альтернативний спосіб доступу через вказівник

## 8. Реалізація принципу інкапсуляції через клас С++

**Клас як ADT реалізує інкапсуляцію через:**

Рівні доступу:

* private**:** приховує внутрішню реалізацію та дані
* public**:** надає контрольований інтерфейс для взаємодії
* protected**:** забезпечує доступ для класів-нащадків

Механізми інкапсуляції:

**1. Приховування даних:**

class BankAccount {

private:

double balance; // Приховані дані

public:

void deposit(double amount); // Контрольований доступ

double getBalance() const; // Читання через метод

};

**2. Контроль доступу:**

* Дані доступні тільки через публічні методи
* Можливість валідації вхідних параметрів
* Захист від некоректних змін стану

**3. Абстракція реалізації:**

* Користувач працює з інтерфейсом, не знаючи деталей
* Можливість зміни внутрішньої реалізації без впливу на клієнтський код
* Спрощення використання складних структур даних

**Переваги такого підходу:**

* **Безпека:** захист від некоректного використання
* **Гнучкість:** можливість зміни реалізації
* **Простота:** зручний інтерфейс для користувача
* **Надійність:** контроль цілісності даних