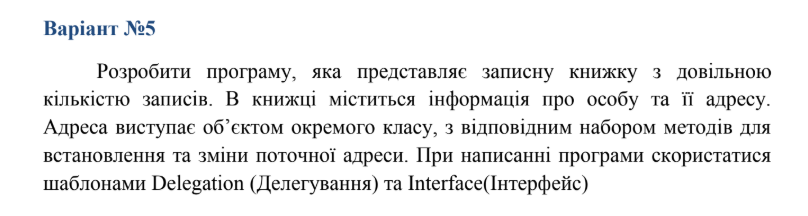
**Міністерство освіти і науки України**Національний лісотехнічний університет України  
Кафедра інформаційних технологій

**ЗВІТ**Лабораторні роботи 1-3 з дисципліни:  
  
«ООАП»  
Варіант: 5

Виконав: студент групи КНз-31   
 Павлишин Артем  
Перевірив: Нечепуренко Андрій

Львів — 2025

Лабораторна робота №1 . Інтерфейс. Абстрактний суперклас. Незмінний. Заступник  
  


КОД:  
  
#include <iostream>

#include <vector>

#include <string>

using namespace std;

class IAddressable {

public:

virtual void setAddress(const string& city, const string& street, int house) = 0;

virtual void showAddress() const = 0;

virtual ~IAddressable() {}

};

class Address {

private:

string city;

string street;

int house;

public:

Address(const string& city = "", const string& street = "", int house = 0)

: city(city), street(street), house(house) {}

void setCity(const string& c) { city = c; }

void setStreet(const string& s) { street = s; }

void setHouse(int h) { house = h; }

string getFullAddress() const {

return city + ", вул. " + street + ", буд. " + to\_string(house);

}

void show() const {

cout << "Місто: " << city << ", Вулиця: " << street << ", Будинок: " << house << endl;

}

};

class Person : public IAddressable {

private:

string name;

string phone;

Address address; // делегування

public:

Person(const string& name, const string& phone, const Address& addr)

: name(name), phone(phone), address(addr) {}

void setAddress(const string& city, const string& street, int house) override {

address.setCity(city);

address.setStreet(street);

address.setHouse(house);

}

void showAddress() const override {

cout << "Адреса: ";

address.show();

}

void showPerson() const {

cout << "Ім'я: " << name << endl;

cout << "Телефон: " << phone << endl;

showAddress();

cout << "-----------------------------" << endl;

}

string getName() const { return name; }

string getPhone() const { return phone; }

string getAddressString() const { return address.getFullAddress(); }

};

class Notebook {

private:

vector<IAddressable\*> records;

public:

void addRecord(IAddressable\* record) {

records.push\_back(record);

}

void showAll() const {

cout << "=== Вміст записної книжки ===" << endl;

for (auto r : records)

r->showAddress();

}

};

int main() {

Address a1("Львів", "Зелена", 12);

Address a2("Київ", "Хрещатик", 5);

Person p1("Андрій", "+380981234567", a1);

Person p2("Марія", "+380671112233", a2);

Notebook nb;

nb.addRecord(&p1);

nb.addRecord(&p2);

p1.showPerson();

p2.showPerson();

cout << "\n=== Після зміни адреси Андрія ===" << endl;

p1.setAddress("Одеса", "Дерибасівська", 8);

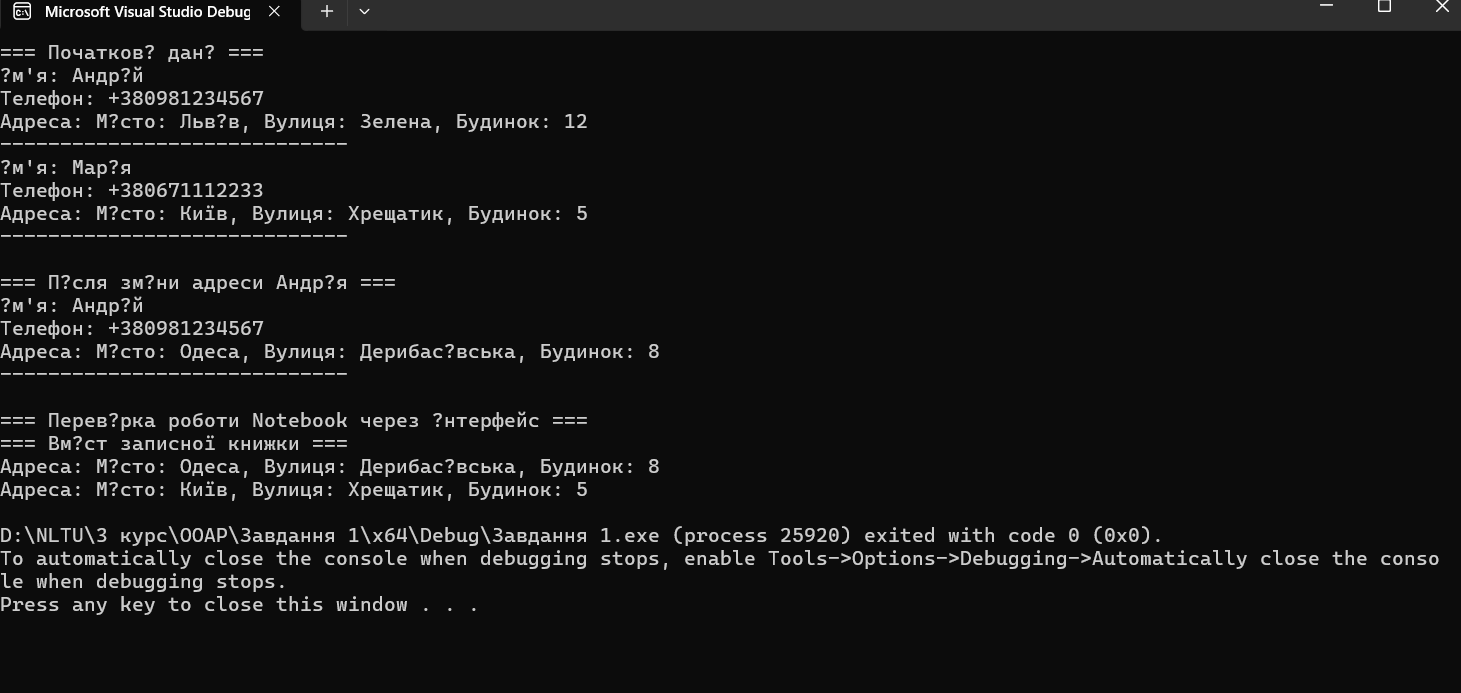
p1.showPerson();

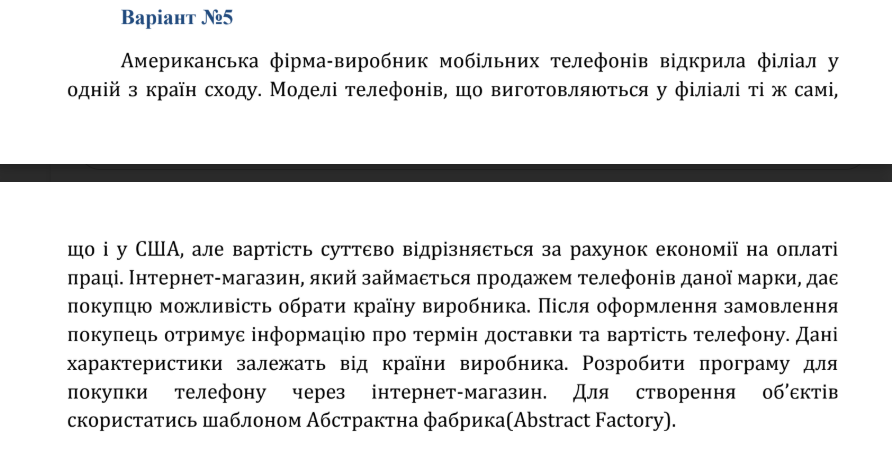
cout << "\n=== Перевірка делегування через записну книжку ===" << endl;

nb.showAll();

return 0;

}

ВИКОНАННЯ:  


Лабораторна №2. Абстрактна фабрика. Будівельник  


КОД:  
#include <iostream>

#include <memory>

#include <string>

#include <iomanip>

#include <limits>

using namespace std;

class IPhone {

public:

virtual ~IPhone() = default;

virtual string modelName() const = 0;

virtual string country() const = 0;

virtual double price() const = 0;

virtual int deliveryDays() const = 0;

virtual void print() const {

cout << "Модель: " << modelName()

<< " | Виробник: " << country()

<< " | Ціна: " << fixed << setprecision(2) << price() << " $"

<< " | Доставка: " << deliveryDays() << " днів" << endl;

}

};

class PhoneBase : public IPhone {

protected:

string \_model;

string \_country;

double \_basePrice;

double \_coef;

int \_delivery;

public:

PhoneBase(string model, string country, double basePrice, double coef, int delivery)

: \_model(model), \_country(country), \_basePrice(basePrice), \_coef(coef), \_delivery(delivery) {

}

string modelName() const override { return \_model; }

string country() const override { return \_country; }

double price() const override { return \_basePrice \* \_coef; }

int deliveryDays() const override { return \_delivery; }

};

class PhoneA : public PhoneBase {

public:

PhoneA(string country, double coef, int delivery)

: PhoneBase("IPhone 17", country, 799.0, coef, delivery) {

}

};

class PhoneB : public PhoneBase {

public:

PhoneB(string country, double coef, int delivery)

: PhoneBase("IPhone 17 Air", country, 999.0, coef, delivery) {

}

};

class PhoneC : public PhoneBase {

public:

PhoneC(string country, double coef, int delivery)

: PhoneBase("IPhone 17 Pro", country, 1099.0, coef, delivery) {

}

};

class PhoneD : public PhoneBase {

public:

PhoneD(string country, double coef, int delivery)

: PhoneBase("IPhone 17 Pro Max", country, 1199.0, coef, delivery) {

}

};

class IPhoneFactory {

public:

virtual ~IPhoneFactory() = default;

virtual unique\_ptr<IPhone> createA() const = 0;

virtual unique\_ptr<IPhone> createB() const = 0;

virtual unique\_ptr<IPhone> createC() const = 0;

virtual unique\_ptr<IPhone> createD() const = 0;

virtual string countryName() const = 0;

virtual int deliveryDays() const = 0;

virtual double coef() const = 0;

};

class USFactory : public IPhoneFactory {

public:

string countryName() const override { return "USA"; }

int deliveryDays() const override { return 3; }

double coef() const override { return 1.20; } // +20%

unique\_ptr<IPhone> createA() const override { return make\_unique<PhoneA>(countryName(), coef(), deliveryDays()); }

unique\_ptr<IPhone> createB() const override { return make\_unique<PhoneB>(countryName(), coef(), deliveryDays()); }

unique\_ptr<IPhone> createC() const override { return make\_unique<PhoneC>(countryName(), coef(), deliveryDays()); }

unique\_ptr<IPhone> createD() const override { return make\_unique<PhoneD>(countryName(), coef(), deliveryDays()); }

};

class IsraelFactory : public IPhoneFactory {

public:

string countryName() const override { return "Israel"; }

int deliveryDays() const override { return 10; }

double coef() const override { return 0.85; } // -15%

unique\_ptr<IPhone> createA() const override { return make\_unique<PhoneA>(countryName(), coef(), deliveryDays()); }

unique\_ptr<IPhone> createB() const override { return make\_unique<PhoneB>(countryName(), coef(), deliveryDays()); }

unique\_ptr<IPhone> createC() const override { return make\_unique<PhoneC>(countryName(), coef(), deliveryDays()); }

unique\_ptr<IPhone> createD() const override { return make\_unique<PhoneD>(countryName(), coef(), deliveryDays()); }

};

class Store {

const IPhoneFactory& factory;

public:

explicit Store(const IPhoneFactory& f) : factory(f) {}

unique\_ptr<IPhone> order(const int id) const {

switch (id) {

case 1: return factory.createA();

case 2: return factory.createB();

case 3: return factory.createC();

case 4: return factory.createD();

default:

throw runtime\_error("Невідомий вибір моделі");

}

}

};

int main() {

setlocale(LC\_ALL, "Ukrainian");

cout << "=== Інтернет-магазин телефонів - Arterium ===\n";

cout << "Оберіть країну виробника:\n";

cout << "1 - USA (Ціна більша, швидка доставка)\n";

cout << "2 - Israel (Ціна нижче, повільна доставка)\n> ";

int choice;

cin >> choice;

unique\_ptr<IPhoneFactory> factory;

if (choice == 1) factory = make\_unique<USFactory>();

else if (choice == 2) factory = make\_unique<IsraelFactory>();

else {

cout << "Невірний вибір країни\n";

return 0;

}

cout << "\nОберіть модель:\n";

cout << "1 - IPhone 17\n";

cout << "2 - IPhone 17 Air\n";

cout << "3 - IPhone 17 Pro\n";

cout << "4 - IPhone 17 Pro Max\n> ";

int modelId;

cin >> modelId;

Store shop(\*factory);

try {

auto phone = shop.order(modelId);

cout << "\nВаш вибір:\n";

phone->print();

}

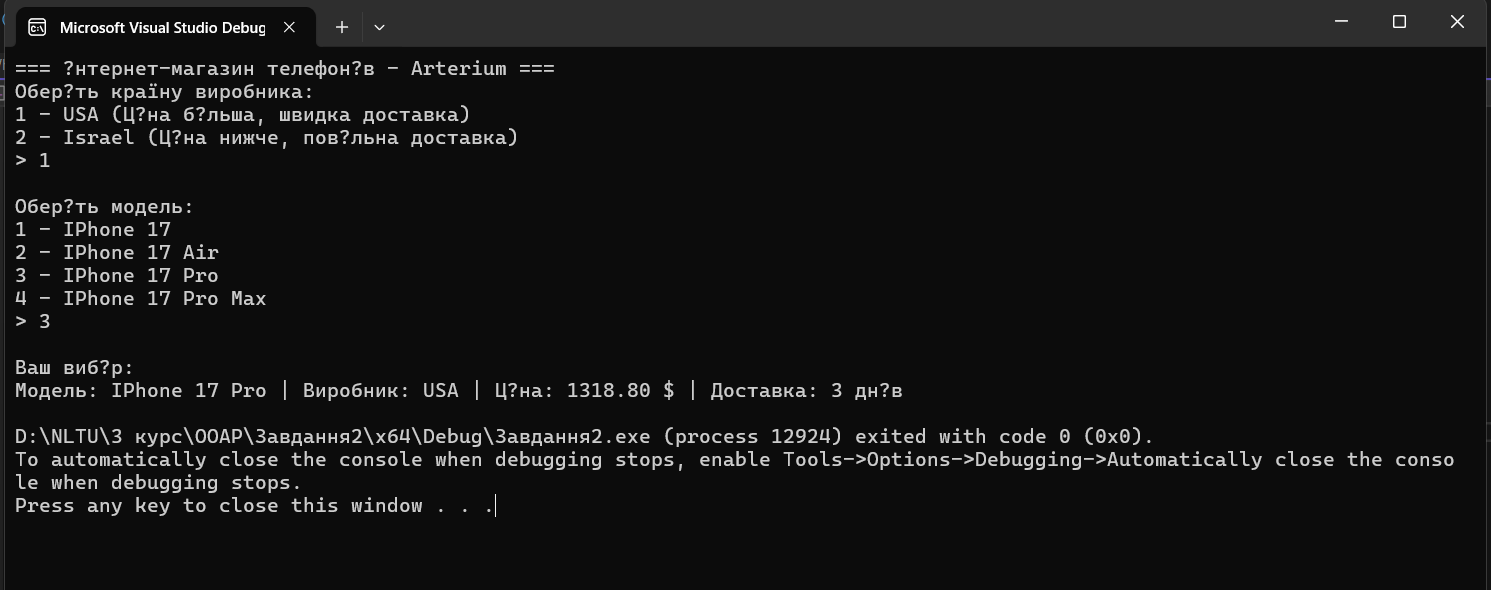
catch (const exception& e) {

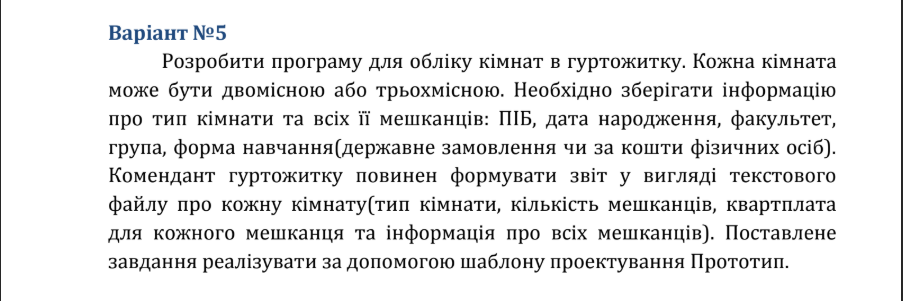
cout << "Помилка: " << e.what() << endl;

}

return 0;

}

ВИКОНАННЯ:  


Лабораторна №3. Фабричний метод. Прототип. Одинак.  


КОД:  
#include <iostream>

#include <vector>

#include <string>

#include <memory>

#include <fstream>

using namespace std;

// ===============================================

// ПРОТОТИП: ОБЛІК КІМНАТ У ГУРТОЖИТКУ

// ===============================================

// ===== КЛАС Resident (мешканець) =====

class Resident {

private:

string name;

string birthDate;

string faculty;

string group;

string form; // форма навчання (державна / контракт)

public:

Resident(string n = "", string b = "", string f = "", string g = "", string fm = "")

: name(n), birthDate(b), faculty(f), group(g), form(fm) {

}

void show() const {

cout << " ПІБ: " << name << endl

<< " Дата народження: " << birthDate << endl

<< " Факультет: " << faculty << endl

<< " Група: " << group << endl

<< " Форма навчання: " << form << endl;

}

string toString() const {

return "ПІБ: " + name + "\nДата народження: " + birthDate +

"\nФакультет: " + faculty + "\nГрупа: " + group +

"\nФорма навчання: " + form + "\n";

}

};

// ===== АБСТРАКТНИЙ ПРОТОТИП Room =====

class Room {

protected:

string type; // тип кімнати

vector<Resident> residents;

public:

virtual ~Room() {}

virtual unique\_ptr<Room> clone() const = 0; // метод клонування (Prototype)

virtual void addResident(const Resident& r) { residents.push\_back(r); }

virtual void showInfo() const {

cout << "Тип кімнати: " << type << endl;

cout << "Кількість мешканців: " << residents.size() << endl;

for (size\_t i = 0; i < residents.size(); ++i) {

cout << "\n-- Мешканець " << i + 1 << " --" << endl;

residents[i].show();

}

}

virtual string toString() const {

string info = "Тип кімнати: " + type + "\n";

info += "Кількість мешканців: " + to\_string(residents.size()) + "\n";

for (size\_t i = 0; i < residents.size(); ++i) {

info += "\n-- Мешканець " + to\_string(i + 1) + " --\n";

info += residents[i].toString();

}

info += "===============================\n";

return info;

}

};

// ===== Конкретний прототип: двомісна кімната =====

class DoubleRoom : public Room {

public:

DoubleRoom() { type = "Двомісна"; }

DoubleRoom(const DoubleRoom& other) { \*this = other; } // копіювання

unique\_ptr<Room> clone() const override {

return make\_unique<DoubleRoom>(\*this);

}

};

// ===== Конкретний прототип: тримісна кімната =====

class TripleRoom : public Room {

public:

TripleRoom() { type = "Тримісна"; }

TripleRoom(const TripleRoom& other) { \*this = other; }

unique\_ptr<Room> clone() const override {

return make\_unique<TripleRoom>(\*this);

}

};

// ===== Клас Hostel (контейнер для кімнат) =====

class Hostel {

private:

vector<unique\_ptr<Room>> rooms;

public:

void addRoom(unique\_ptr<Room> room) {

rooms.push\_back(move(room));

}

void showAll() const {

cout << "=== ІНФОРМАЦІЯ ПРО ВСІ КІМНАТИ ===" << endl;

for (const auto& r : rooms) {

r->showInfo();

cout << "-------------------------------\n";

}

}

void generateReport(const string& filename) const {

ofstream out(filename);

if (!out) {

cerr << "Помилка відкриття файлу!" << endl;

return;

}

for (const auto& r : rooms)

out << r->toString();

out.close();

cout << "Звіт збережено у файлі: " << filename << endl;

}

};

// ===== MAIN =====

int main() {

setlocale(LC\_ALL, "UKRAINIAN");

// Створюємо прототипи кімнат

DoubleRoom doubleProto;

TripleRoom tripleProto;

// Створюємо об'єкт гуртожитку

Hostel hostel;

// --- Кімната 1 (на основі двомісного прототипу) ---

auto room1 = doubleProto.clone();

room1->addResident(Resident("Іваненко Іван", "2003-02-11", "ІТ", "КНз-21", "державна"));

room1->addResident(Resident("Петренко Марія", "2004-04-21", "ІТ", "КНз-21", "контракт"));

hostel.addRoom(move(room1));

// --- Кімната 2 (на основі тримісного прототипу) ---

auto room2 = tripleProto.clone();

room2->addResident(Resident("Сидоренко Олег", "2002-12-10", "Лісогосп.", "ЛГ-11", "державна"));

room2->addResident(Resident("Білик Анна", "2003-07-05", "Лісогосп.", "ЛГ-11", "контракт"));

room2->addResident(Resident("Демчук Павло", "2002-10-09", "Лісогосп.", "ЛГ-11", "державна"));

hostel.addRoom(move(room2));

// --- Кімната 3 (ще один клон двомісної) ---

auto room3 = doubleProto.clone();

room3->addResident(Resident("Коваль Юлія", "2003-03-03", "Економіка", "ЕК-12", "контракт"));

room3->addResident(Resident("Гринюк Назар", "2003-06-20", "Економіка", "ЕК-12", "контракт"));

hostel.addRoom(move(room3));

// Виведення інформації у консоль

hostel.showAll();

// Формування текстового звіту

hostel.generateReport("HostelReport.txt");

return 0;

}

ВИКОНАННЯ:  
