Чернівецький національний університет імені Юрія Федьковича

Навчально-науковий інститут фізико-технічних та комп’ютерних наук

Кафедра математичних проблем управління і кібернетики

Звіт

про виконання лабораторної роботи №5

«Спадкування. Ієрархія та композиція»

з дисципліни

«Об’єктно-орієнтоване програмування»

Виконала: студентка 141-A+Б групи

Жданюк А.С.

Перевірив: канд.фіз.-мат. наук,

доцент Лазорик В.В.

Оцінка:

Дата захисту:

Чернівці 2024

Тема:**Спадкування.Ієрархія та композиція**

**Завдання для лабораторної:**

1. Зайти в свій обліковий запис на github.com. Зайти в github classroom.

2. Клонувати репозиторій: <https://classroom.github.com/a/m683j7Cm> на робочий комп’ютер. В репозиторії знаходиться в функції main() викликається демонстраційний приклад виконання подібної лабораторної роботи, який можна використовувати (модифікувати та використовувати) для розв’язання задачі власних задач.

3. Розв’язати задачі згідно варіанту. Визначити ієрархію класів та класи згідно теми задачі. Повну структуру класів і їх взаємозв'язок продумати самостійно.

4. У класах реалізувати конструктори, деструктори, статичні, віртуальні та не віртуальні методи згідно з варіантом завдання. У конструкторах та деструкторах.

5. Передбачити введення початкових даних: з клавіатури, файлу та використовуючи датчик випадкових чисел.

6. Написати тести для кожної задачі.

7. Розробити програму мовою C++ тестування всіх можливосте створених класів із виведення відповідної інформації

8. В процесі написання функцій розв’язання задач лабораторної роботи періодично здійснювати синхронізацію з репозиторієм на github.com, з поясненням виконаної роботи (git add ., git commit -m”коментар”, git push).

9. Оформити звіт про виконання лабораторної роботи.

10. Звіт відправити в для оцінювання в [https://moodle.chnu.edu.ua](https://moodle.chnu.edu.ua/)

**Мої завдання для лабораторної роботи:**

Задача 1.7 Створити клас рідина, що має поля: назва, щільність. Визначити конструктори, деструктор і функцію друку. Створити похідний клас - спиртні напої, що має міцність. Визначити конструктори за замовчуванням і з різним числом параметрів, деструктори, функцію друку. Визначити функції перепризначення

щільності й міцності. Написати програму тестування всіх можливостей цього класу.

Задача 2.7 Створити класи колесо, велосипед і автомобіль. Скласти з них ієрархію або композицію.

Задача 3.7 Створити ієрархію класів людей і викладач. Перевизначити вивід у потік і введення з потоку, конструктор копіювання, оператор присвоювання через відповідні функції базового класу.

**Хід лабораторної роботи**

**Перше завдання**



**Ієрархія в програмуванні -** це коли кожен елемент коду має своє місце та рівень важливості.

#include <iostream>

#include <string>

class Fluid {

protected:

std::string name;

double density;

public:

// Constructors

Fluid() : name(""), density(0.0) {}

Fluid(const std::string& \_name, double \_density) : name(\_name), density(\_density) {}

// Destructor

virtual ~Fluid() {}

// Print function

virtual void print() const {

std::cout << "Fluid: " << name << ", Density: " << density << " g/cm^3" << std::endl;

}

// Setter for density

virtual void setDensity(double \_density) {

density = \_density;

}

};

class AlcoholicBeverage : public Fluid {

private:

double strength;

public:

// Constructors

AlcoholicBeverage() : Fluid(), strength(0.0) {}

AlcoholicBeverage(const std::string& \_name, double \_density, double \_strength)

: Fluid(\_name, \_density), strength(\_strength) {}

// Destructor

~AlcoholicBeverage() override {}

// Print function

void print() const override {

std::cout << "Alcoholic Beverage: " << name << ", Density: " << density << " g/cm^3, Strength: " << strength << "%" << std::endl;

}

// Setter for strength

void setStrength(double \_strength) {

strength = \_strength;

}

// Redefining density setter

void setDensity(double \_density) override {

density = \_density;

}

};

int main() {

Fluid water("Water", 1.0);

AlcoholicBeverage beer("Beer", 1.02, 5.0);

// Printing fluids

water.print();

beer.print();

// Changing properties

water.setDensity(0.997); // Setting density of water at 20°C

beer.setStrength(4.5); // Setting strength of beer

// Printing updated properties

std::cout << "\nUpdated properties:\n";

water.print();

beer.print();

return 0;

}

**Друге завдання**



#include <iostream>

#include <string>

class Wheel {

protected:

int size; // Size of the wheel in inches

public:

// Constructor

Wheel(int \_size) : size(\_size) {}

// Destructor

virtual ~Wheel() {}

// Print function

virtual void print() const {

std::cout << "Wheel size: " << size << " inches" << std::endl;

}

};

class Bicycle {

protected:

Wheel\* frontWheel;

Wheel\* rearWheel;

public:

// Constructor

Bicycle(int frontSize, int rearSize) {

frontWheel = new Wheel(frontSize);

rearWheel = new Wheel(rearSize);

}

// Destructor

~Bicycle() {

delete frontWheel;

delete rearWheel;

}

// Print function

void print() const {

std::cout << "Bicycle details:" << std::endl;

std::cout << "Front ";

frontWheel->print();

std::cout << "Rear ";

rearWheel->print();

}

};

class Car {

protected:

Wheel\* wheels[4]; // Four wheels for a car

public:

// Constructor

Car(int wheelSize) {

for (int i = 0; i < 4; ++i) {

wheels[i] = new Wheel(wheelSize);

}

}

// Destructor

~Car() {

for (int i = 0; i < 4; ++i) {

delete wheels[i];

}

}

// Print function

void print() const {

std::cout << "Car details:" << std::endl;

for (int i = 0; i < 4; ++i) {

std::cout << "Wheel " << i+1 << ": ";

wheels[i]->print();

}

}

};

int main() {

Bicycle bike(26, 26);

Car car(18);

std::cout << "Printing Bicycle details:" << std::endl;

bike.print();

std::cout << "\nPrinting Car details:" << std::endl;

car.print();

return 0;

}

**Третє завдання**



#include <iostream>

#include <string>

class Person {

protected:

std::string name;

int age;

public:

// Constructors

Person() : name(""), age(0) {}

Person(const std::string& \_name, int \_age) : name(\_name), age(\_age) {}

Person(const Person& other) : name(other.name), age(other.age) {}

// Destructor

virtual ~Person() {}

// Assignment operator

Person& operator=(const Person& other) {

if (this != &other) {

name = other.name;

age = other.age;

}

return \*this;

}

// Output stream operator

friend std::ostream& operator<<(std::ostream& os, const Person& person) {

os << "Name: " << person.name << ", Age: " << person.age;

return os;

}

// Input stream operator

friend std::istream& operator>>(std::istream& is, Person& person) {

std::cout << "Enter name: ";

is >> person.name;

std::cout << "Enter age: ";

is >> person.age;

return is;

}

};

class Teacher : public Person {

private:

std::string subject;

public:

// Constructors

Teacher() : Person(), subject("") {}

Teacher(const std::string& \_name, int \_age, const std::string& \_subject)

: Person(\_name, \_age), subject(\_subject) {}

Teacher(const Teacher& other) : Person(other), subject(other.subject) {}

// Destructor

~Teacher() {}

// Assignment operator

Teacher& operator=(const Teacher& other) {

if (this != &other) {

Person::operator=(other);

subject = other.subject;

}

return \*this;

}

// Output stream operator

friend std::ostream& operator<<(std::ostream& os, const Teacher& teacher) {

os << "Teacher: " << static\_cast<const Person&>(teacher) << ", Subject: " << teacher.subject;

return os;

}

// Input stream operator

friend std::istream& operator>>(std::istream& is, Teacher& teacher) {

is >> static\_cast<Person&>(teacher);

std::cout << "Enter subject: ";

is >> teacher.subject;

return is;

}

};

int main() {

// Creating a Person object

Person person;

std::cout << "Enter person's details:" << std::endl;

std::cin >> person;

std::cout << "Person's details: " << person << std::endl;

// Creating a Teacher object

Teacher teacher;

std::cout << "\nEnter teacher's details:" << std::endl;

std::cin >> teacher;

std::cout << "Teacher's details: " << teacher << std::endl;

return 0;

}