**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ им. И. РАЗЗАКОВА**

**Кафедра «Программное обеспечение компьютерных систем»**

**Отчет**

**ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ С++**

**Лабораторная работа №6**

**Выполнила: Эмилбекова Элиза ПИ-2-21**

**Проверил: Мусабаев Э.Б.**

**Бишкек 2024**

**Тема: Наследование**

**Задания:**

1. Проблему экономии времени, доходов, а также увеличения надежности создаваемой объектно-ориентированной программы можно решить с помощью одного из главных принципов ООП - наследования, используя для создания класса-потомка готовый протестированный и отлаженный код класса-родителя. Для решения этой проблемы используйте преимущества **простого наследования**. Для этого создайте родительский класс работник (**Employee)**, протестируйте и отладьте его. В описании этого класса есть поля**:** имя работника - занимаемая должность - position[64], заработная плата за месяц- salary;

методы: конструктор по умолчанию, конструктор с аргументами, деструктор, расчет заработной платы за год, размещение информации о классе-родителе на консоли.

На основе отлаженного класса-родителя **Employee** создайте с помощью **механизма наследования** **public** класс-потомок менеджер (**Manager)**.

Класс - потомок Manager состоит из**:**

полей**:** ежегодный бонус **(annual\_bonus)**, машина компании (**company\_car[64]**)**,** опционы на акции (**stock\_options**) типа (**int**);

методов: конструктор по умолчанию, конструктор с аргументами, деструктор, размещение информации о классе-потомке на консоли.

*Определитесь с идентификаторами доступа к членам класса, не нарушая основного принципа ООП – инкапсуляции в наследовании.*

**Код:**

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

string position;

double salary;

public:

Employee() : name(""), position(""), salary(0.0) {}

Employee(const string& n, const string& pos, double sal)

: name(n), position(pos), salary(sal) {}

~Employee() {

cout << "Деструктор Employee вызван для " << name << endl;

}

double calculateAnnualSalary() const {

return salary \* 12;

}

void displayInfo() const {

cout << "Имя: " << name << ", Должность: " << position << ", Заработная плата: " << salary << endl;

}

};

class Manager : public Employee {

private:

double annual\_bonus;

string company\_car;

int stock\_options;

public:

Manager()

: Employee(), annual\_bonus(0.0), company\_car(""), stock\_options(0) {}

Manager(const string& n, const std::string& pos, double sal, double bonus, const string& car, int options)

: Employee(n, pos, sal), annual\_bonus(bonus), company\_car(car), stock\_options(options) {}

~Manager() {

cout << "Деструктор Manager вызван для " << name << endl;

}

void displayInfo() const {

Employee::displayInfo(); // Вызываем метод displayInfo из класса-родителя

cout << "Ежегодный бонус: " << annual\_bonus << ", Автомобиль компании: " << company\_car << ", Опции на акции: " << stock\_options << endl;

}

};

int main() {

setlocale(LC\_ALL, "Russian");

string empName, empPosition;

double empSalary;

cout << "Введите данные для работника:" << endl;

cout << "Имя: ";

getline(cin, empName);

cout << "Должность: ";

getline(std::cin, empPosition);

cout << "Заработная плата: ";

cin >> empSalary;

Employee emp1(empName, empPosition, empSalary);

emp1.displayInfo();

cout << "Ежегодная Заработная плата: " << emp1.calculateAnnualSalary() << endl;

string mgrName, mgrPosition, mgrCar;

double mgrSalary, mgrBonus;

int mgrOptions;

cin.ignore();

cout << "Введите данные для менеджера:" << endl;

cout << "Имя: ";

getline(cin, mgrName);

cout << "Должность: ";

getline(cin, mgrPosition);

cout << "Заработная плата: ";

cin >> mgrSalary;

cin.ignore();

cout << "Ежегодный бонус: ";

cin >> mgrBonus;

cin.ignore();

cout << "Автомобиль компании: ";

getline(std::cin, mgrCar);

cout << "Опции на акции: ";

cin >> mgrOptions;

Manager manager1(mgrName, mgrPosition, mgrSalary, mgrBonus, mgrCar, mgrOptions);

manager1.displayInfo();

cout << "Ежегодная Заработная плата: " << manager1.calculateAnnualSalary() << endl;

return 0;

}

Класс Manager является дочерним классом для Employee. Он наследует все поля и методы родительского класса и добавляет к ним свои уникальные характеристики - ежегодный бонус, машину компании и опции на акции.

**2. Используйте преимущества множественного наследования, а именно: разработайте, протестируйте и отладьте родительские классы Дисплей (*Display*) и Материнская плата (M*otherBoard),* и на их основе создайте класс-потомок Компьютер (*Сomputer*):**

**В описании *класса-родителя* *Display* есть**

**поля: тип монитора char type[32], количество цветов long colors, разрешение по оси х int x\_resolution, разрешение по оси у int y\_resolution.**

**методы: конструктор с аргументами; размещение информации о классе-родителе на консоли.**

**В описании *класса-родителя MotherBoard имеет***

**поля: тип процессора int processor, скорость процессора int speed, объем оперативной памяти int RAM;**

**методы: конструктор с аргументами; размещение информации о классе-родителе на консоли.**

**- *Класс-потомок Сomputer* создать на основе родительских классов с помощью механизма наследования public и отобразить его на экране, добавив:**

**поля: марка компьютера (name [64]), объем жесткого диска (hard\_disk).**

**методы: конструктор с аргументами, метод Show().**

**Код:**

#include <iostream>

#include <cstring>

using namespace std;

class Display {

protected:

char type[32];

long colors;

int x\_resolution;

int y\_resolution;

public:

Display(const char\* t, long c, int x\_res, int y\_res) : colors(c), x\_resolution(x\_res), y\_resolution(y\_res) {

strncpy\_s(type, t, sizeof(type) - 1);

type[sizeof(type) - 1] = '\0';

}

void displayInfo() const {

cout << "Тип монитора: " << type << ", Количество цветов: " << colors

<< ", Разрешение: " << x\_resolution << "x" << y\_resolution << endl;

}

};

class MotherBoard {

protected:

int processor;

int speed;

int RAM;

public:

MotherBoard(int proc, int spd, int ram) : processor(proc), speed(spd), RAM(ram) {}

void displayInfo() const {

cout << "Тип процессора: " << processor << ", Скорость процессора: " << speed

<< ", Объем оперативной памяти: " << RAM << " GB" << endl;

}

};

class Computer : public Display, public MotherBoard {

private:

char name[64];

int hard\_disk;

public:

Computer(const char\* n, int hd, const char\* t, long c, int x\_res, int y\_res, int proc, int spd, int ram)

: Display(t, c, x\_res, y\_res), MotherBoard(proc, spd, ram), hard\_disk(hd) {

strncpy\_s(name, n, sizeof(name) - 1);

name[sizeof(name) - 1] = '\0';

}

void displayInfo() const {

cout << "Марка компьютера: " << name << ", Объем жесткого диска: " << hard\_disk << " GB" << endl;

Display::displayInfo(); // Метод из родительского класса Display

MotherBoard::displayInfo(); // Метод из родительского класса MotherBoard

}

};

int main() {

setlocale(LC\_ALL, "Russian");

Computer myComputer("MyPC", 512, "LED", 16'777'216, 1920, 1080, 3, 3600, 32);

myComputer.displayInfo();

return 0;

}

Суть программы заключается в демонстрации принципов множественного наследования и создания объекта компьютера на основе двух родительских классов: `Display` и `MotherBoard`.

1. Класс Display:

- Представляет монитор с характеристиками: тип монитора, количество цветов, разрешение по осям X и Y.

- Реализует метод `displayInfo()`, который выводит информацию о мониторе на консоль.

2.Класс MotherBoard:

- Представляет материнскую плату компьютера с характеристиками: тип процессора, скорость процессора, объем оперативной памяти.

- Реализует метод `displayInfo()`, который выводит информацию о материнской плате на консоль.

3. Класс Computer:

- Является классом-потомком, наследующим от `Display` и `MotherBoard`.

- Добавляет свои уникальные характеристики: марка компьютера и объем жесткого диска.

- Реализует метод `Show()`, который выводит на консоль информацию о компьютере, включая информацию о мониторе и материнской плате.

Таким образом, программа демонстрирует, как можно организовать иерархию классов с использованием множественного наследования, а также как создать объект класса-потомка и использовать его методы для вывода информации на консоль.

**3. В задании № 1 изменить механизм наследования так, чтобы закрыть доступ к членам базового класса из производного класса. А затем восстановить этот доступ всеми известным Вам способами.**

**Код:**

#include <iostream>

#include <cstring>

using namespace std;

// Родительский класс Employee

class Employee {

private:

string name;

string position;

double salary;

public:

Employee() : name(""), position(""), salary(0.0) {}

Employee(const string& n, const string& pos, double sal)

: name(n), position(pos), salary(sal) {}

~Employee() {

cout << "Деструктор Employee вызван для " << name << endl;

}

double calculateAnnualSalary() const {

return salary \* 12;

}

void displayInfo() const {

cout << "Имя: " << name << ", Должность: " << position << ", Заработная плата: " << salary << endl;

}

// Объявление дружественного класса Manager

friend class Manager;

};

// Производный класс Manager

class Manager : public Employee {

private:

double annual\_bonus;

string company\_car;

int stock\_options;

public:

Manager() : Employee(), annual\_bonus(0.0), company\_car(""), stock\_options(0) {}

Manager(const string& n, const string& pos, double sal, double bonus, const string& car, int options)

: Employee(n, pos, sal), annual\_bonus(bonus), company\_car(car), stock\_options(options) {}

~Manager() {

cout << "Деструктор Manager вызван для " << name << endl;

}

// Метод для доступа к членам базового класса

void displayInfoManager() const {

// Мы можем обращаться к private-членам базового класса, так как Manager объявлен как friend

displayInfo();

cout << "Ежегодный бонус: " << annual\_bonus << ", Автомобиль компании: " << company\_car << ", Опции на акции: " << stock\_options << endl;

}

// Восстановление доступа к методу calculateAnnualSalary из базового класса

double calculateAnnualSalary() const {

// Мы можем вызвать метод базового класса из производного, так как Manager объявлен как friend

return Employee::calculateAnnualSalary();

}

};

int main() {

setlocale(LC\_ALL, "Russian");

Manager manager1("John Doe", "Manager", 5000, 1000, "Company Car", 500);

manager1.displayInfoManager(); // Мы используем public-метод производного класса для доступа к private-членам базового класса

cout << "Ежегодная Заработная плата: " << manager1.calculateAnnualSalary() << endl; // Восстанавливаем доступ и используем метод базового класса

return 0;

}

В задаче требовалось изменить механизм наследования в классах Employee и Manager так, чтобы закрыть доступ к членам базового класса из производного класса. Затем необходимо было восстановить этот доступ, используя дружественные функции или классы.

**Решение:**

**Закрытие доступа:**

В классе Manager при наследовании от Employee использован модификатор доступа private, чтобы закрыть доступ к членам базового класса.

**Восстановление доступа:**

В классе Manager объявлен дружественным класс Employee, что позволяет Manager иметь доступ к private-членам Employee.

Добавлены методы displayInfoManager и calculateAnnualSalary в класс Manager, которые обращаются к private-членам базового класса через public-методы.

1. **такое в ООП? Приведите примеры.**

ООП (Объектно-Ориентированное Программирование): Это методология программирования, основанная на использовании объектов, которые объединяют данные и методы для их обработки в единую сущность. Примеры:

Класс и Объект: Класс "Автомобиль" с атрибутами (цвет, модель) и методами (запуск, остановка).

Инкапсуляция: Сокрытие деталей реализации, например, скрытие внутренних данных класса от внешнего мира.

Наследование: Создание нового класса на основе существующего, чтобы повторно использовать код и расширить его функциональность.

Полиморфизм: Возможность использования одного интерфейса для различных типов данных.

1. **В чем заключается выигрыш для программиста при использовании им наследования?**

Выигрыш при использовании наследования:

Повторное использование кода: Наследование позволяет создавать новые классы на основе существующих, повторно использовать код базового класса.

Расширение функциональности: Программист может добавлять новые методы и поля в производный класс, расширяя функциональность базового класса.

Управление кодом: Код становится более организованным и структурированным благодаря иерархии классов.

1. **Какие члены базового класса не наследуются?**

Private-члены: Private-члены базового класса не наследуются производным классом и остаются недоступными извне.

От чего зависит доступ к членам базового класса из произвольного класса?

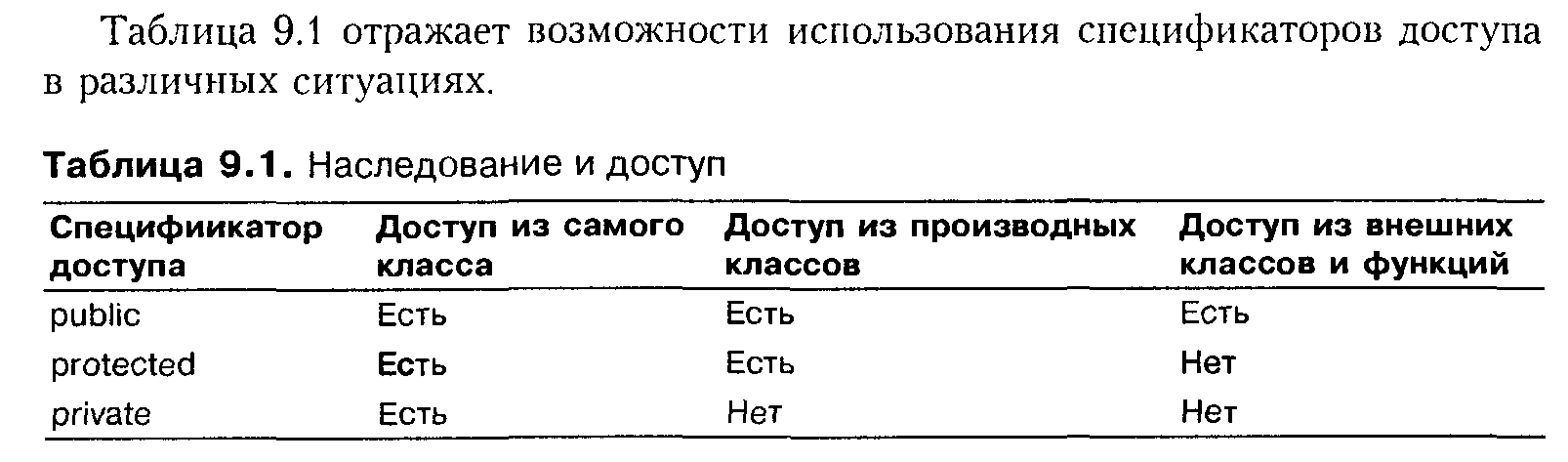
Модификатор доступа: Доступ зависит от модификатора доступа, который определен для членов базового класса. Public-члены доступны из производного класса, private-члены доступны только внутри базового класса, а protected-члены доступны внутри базового класса и его производных.

1. **Чем отличается идентификатор доступа от механизма доступа?**

Идентификатор доступа: Это ключевое слово (public, private, protected), которое определяет уровень доступа к членам класса.

Механизм доступа: Это общее понятие, описывающее, каким образом классы и их члены взаимодействуют друг с другом, включая наследование и модификаторы доступа.

1. **Назовите основные правила наследования (используйте таблицу наследования).**



1. **Синтаксис объявления производного класса, как наследника базового класса?**

cppCopy code

class DerivedClass : [public/protected/private] BaseClass { // Тело производного класса };

1. **Что такое простое, множественное и многократное наследование?**

Простое наследование: Класс наследует только от одного базового класса.

Множественное наследование: Класс наследует от нескольких базовых классов.

Многократное наследование: Один и тот же класс может выступать в роли базового класса для нескольких производных классов.

1. **Особенности использования идентификатора доступа protected при наследовании классов?**

Protected: Члены класса с модификатором доступа protected доступны внутри самого класса, а также внутри производных классов. Они не доступны извне.

1. **Какие существуют виды конструкторов при наследовании?**

Конструктор базового класса: Вызывается при создании объекта базового класса.

Конструктор производного класса: Вызывается при создании объекта производного класса.

1. **Каков синтаксис объявления конструктора производного класса?**

**имя\_конструктора производного класса (аргументы конструктора базового класса, аргументы конструктора производного класса): имя\_ конструктора базового класса (аргументы базового класс), {имя\_атрибута1 производного класса = значение; …}**

1. **Каковы особенности конструктора при простом наследовании?**

Конструктор производного класса автоматически вызывает конструктор базового класса.

1. **Каковы особенности конструктора при многократном наследовании?**

В случае многократного наследования, конструкторы базовых классов вызываются в порядке, определенном порядком их указания в списке наследования.

1. **Какова последовательность вызова конструкторов и деструкторов при наследовании?**

При создании объекта производного класса вызываются конструкторы базовых классов в порядке, определенном списком наследования. При уничтожении объекта вызываются деструкторы в обратном порядке.