**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. И. РАЗЗАКОВА**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**ОТЧЕТ**

**по лабораторной работе №6**

**Выполнила:** студент группы ПИ-3-21

Джанышбекова Акмарал

**Проверила:** Мусабаев Э. Б.

**Бишкек 2024**

**Задание №1**

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 программы:**

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

string position;

double salary;

public:

// Конструктор по умолчанию

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

// Конструктор с аргументами

Employee(const string& name, const string& position, double salary)

: name(name), position(position), salary(salary) {}

// Деструктор

~Employee() {

std::cout << "Объект Employee уничтожен для " << name << endl;

}

// Расчет годовой заработной платы

double calculateAnnualSalary() const {

return salary \* 12;

}

// Вывод информации о работнике на консоль

void displayInfo() const {

std::cout << "----------------------------------------\n"

<< "Информация о сотрудниках\n"

<< "----------------------------------------\n"

<< "Имя: " << name << "\nДолжность: " << position << "\nЗаработная плата: $" << salary << "\n";

}

};

class Manager : public Employee {

private:

double annual\_bonus;

string company\_car;

int stock\_options;

public:

// Конструктор по умолчанию

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

// Конструктор с аргументами

Manager(const string& name, const string& position, double salary,

double annual\_bonus, const string& company\_car, int stock\_options)

: Employee(name, position, salary), annual\_bonus(annual\_bonus),

company\_car(company\_car), stock\_options(stock\_options) {}

// Деструктор

~Manager() {

cout << "\nОбъект Manager уничтожен для " << name << endl;

}

// Вывод информации о менеджере на консоль

void displayInfo() const {

Employee::displayInfo(); // Вызываем метод родительского класса

cout << "Годовой бонус: " << annual\_bonus << "\nКомпанийская машина: " << company\_car

<< "\nОпции по акциям: " << stock\_options << endl;

}

double getTotalAnnualEarnings() const {

// Вызываем метод calculateAnnualSalary базового класса и добавляем к нему annual\_bonus

return calculateAnnualSalary() + annual\_bonus;

}

};

int main() {

setlocale(LC\_ALL, "Ru");

Employee emp("Джон Доу", "Инженер-программист", 5000.0);

emp.displayInfo();

cout << "Годовая зарплата: $" << emp.calculateAnnualSalary() << endl;

Manager manager("Алиса Смит", "Руководитель проекта", 7000.0, 2000.0, "Компанийская машина ABC", 500);

manager.displayInfo();

cout << "Годовая зарплата: $" << manager.calculateAnnualSalary() << endl;

cout << "Итоговая годовая зарплата: $" << manager.getTotalAnnualEarnings() << endl;

Manager manager2("Bob Johnson", "Engineering Manager", 8000.0, 2500.0, "Company Car XYZ", 600);

manager2.displayInfo();

std::cout << "Annual Salary: $" << manager2.calculateAnnualSalary() << "\n";

cout << "Итоговая годовая зарплата: $" << manager2.getTotalAnnualEarnings() << endl;

return 0;

}

**Задание №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-потомокКомпьютер **(*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 классовс 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 программы:**

#include <iostream>

#include <cstring>

using namespace std;

// Родительский класс Display

class Display {

protected:

char type[32];

long colors;

int x\_resolution;

int y\_resolution;

public:

// Конструктор с аргументами

Display(const char\* displayType, long displayColors, int xRes, int yRes)

: colors(displayColors), x\_resolution(xRes), y\_resolution(yRes) {

strncpy\_s(type, displayType, \_TRUNCATE);

}

// Размещение информации о классе-родителе на консоли

void displayInfo() const {

std::cout << "----------------------------------------\n"

<< "Информация о дисплее\n"

<< "----------------------------------------\n"

<< "Тип: " << type << "\nЦвета: " << colors

<< "\nРазрешение: " << x\_resolution << "x" << y\_resolution << "\n";

}

};

// Родительский класс MotherBoard

class MotherBoard {

protected:

int processor;

int speed;

int RAM;

public:

// Конструктор с аргументами

MotherBoard(int proc, int spd, int ram)

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

// Размещение информации о классе-родителе на консоли

void displayInfo() const {

std::cout << "----------------------------------------\n"

<< "Информация о материнской плате\n"

<< "----------------------------------------\n"

<< "Процессор: " << processor << "\nСкорость: " << speed

<< " GHz\nОперативная память: " << RAM << " GB\n";

}

};

// Класс-потомок Computer

class Computer : public Display, public MotherBoard {

private:

char name[64];

int hard\_disk;

public:

// Конструктор с аргументами

Computer(const char\* compName, const char\* displayType, long displayColors,

int xRes, int yRes, int proc, int spd, int ram, int disk)

: Display(displayType, displayColors, xRes, yRes),

MotherBoard(proc, spd, ram), hard\_disk(disk) {

strncpy\_s(name, compName, \_TRUNCATE);

}

// Метод для отображения информации о компьютере на экране

void show() const {

std::cout << "----------------------------------------\n"

<< "Информация о компьютере\n"

<< "----------------------------------------\n"

<< "Имя компьютера: " << name << "\nРазмер жесткого диска: " << hard\_disk << " GB\n";

Display::displayInfo();

MotherBoard::displayInfo();

}

};

int main() {

setlocale(LC\_ALL, "RU");

// Пример использования класса Computer

Computer myComputer("MyPC", "LCD", 16777216, 1920, 1080, 4, 3, 16, 500);

myComputer.show();

return 0;

}

**Задание №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 программы:**

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

string name;

string position;

double salary;

public:

// Конструктор по умолчанию

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

// Конструктор с аргументами

Employee(const string& name, const string& position, double salary)

: name(name), position(position), salary(salary) {}

// Деструктор

~Employee() {

std::cout << "Объект Employee уничтожен для " << name << endl;

}

// Расчет годовой заработной платы

double calculateAnnualSalary() const {

return salary \* 12;

}

// Вывод информации о работнике на консоль

void displayInfo() const {

std::cout << "----------------------------------------\n"

<< "Информация о сотрудниках\n"

<< "----------------------------------------\n"

<< "Имя: " << name << "\nДолжность: " << position << "\nЗаработная плата: $" << salary << "\n";

}

};

class Manager : private Employee { //// private Employee

private:

double annual\_bonus;

string company\_car;

int stock\_options;

public:

// Конструктор по умолчанию

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

// Конструктор с аргументами

Manager(const string& name, const string& position, double salary,

double annual\_bonus, const string& company\_car, int stock\_options)

: Employee(name, position, salary), annual\_bonus(annual\_bonus),

company\_car(company\_car), stock\_options(stock\_options) {}

// Деструктор

~Manager() {

cout << "\nОбъект Manager уничтожен для " << name << endl;

}

//// Восстановление доступа к методам базового класса

using Employee::calculateAnnualSalary;

using Employee::displayInfo;

// Вывод информации о менеджере на консоль

void displayInfo() const {

Employee::displayInfo(); // Вызываем метод родительского класса

cout << "Годовой бонус: " << annual\_bonus << "\nКомпанийская машина: " << company\_car

<< "\nОпции по акциям: " << stock\_options << endl;

}

double getTotalAnnualEarnings() const {

// Вызываем метод calculateAnnualSalary базового класса и добавляем к нему annual\_bonus

return calculateAnnualSalary() + annual\_bonus;

}

};

int main() {

setlocale(LC\_ALL, "Ru");

Employee emp("Джон Доу", "Инженер-программист", 5000.0);

emp.displayInfo();

cout << "Годовая зарплата: $" << emp.calculateAnnualSalary() << endl;

Manager manager("Алиса Смит", "Руководитель проекта", 7000.0, 2000.0, "Компанийская машина ABC", 500);

manager.displayInfo();

cout << "Годовая зарплата: $" << manager.calculateAnnualSalary() << endl;

cout << "Итоговая годовая зарплата: $" << manager.getTotalAnnualEarnings() << endl;

}

**ВОПРОСЫ К ЗАЩИТЕ ЛАБОРАТОРНОЙ РАБОТЫ №6**

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

Наследование в (ООП) - это принцип, который позволяет создавать новый класс на основе уже существующего класса, называемого базовым (родительским) классом. Новый класс, созданный на основе базового класса, называется производным (потомком) классом. Производный класс наследует свойства и методы базового класса и может расширять или изменять их.

Пример наследования в C++:

#include <iostream>

#include <string>

using namespace std;

// Базовый (родительский) класс

class Animal {

protected:

string name;

public:

Animal(const string& animalName) : name(animalName) {}

void eat() const {

cout << name << " ест." << endl;

}

void sleep() const {

cout << name << " спит." << endl;

}

};

// Производный (потомковый) класс

class Cat : public Animal {

private:

string breed;

public:

Cat(const string& catName, const string& catBreed)

: Animal(catName), breed(catBreed) {}

void meow() const {

cout << name << " говорит: Мяу!" << endl;

}

void displayInfo() const {

cout << "Имя: " << name << "\nПорода: " << breed << endl;

}

};

int main() {

setlocale(LC\_ALL, "RU");

// Пример использования наследования

cout << "Информация о кошке по имени Вискарь: " << endl;

Cat myCat("Вискарь", "Персидская");

myCat.displayInfo();

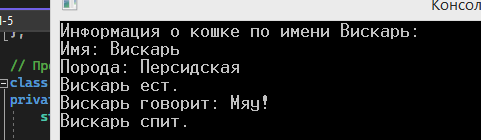
myCat.eat();

myCat.meow();

myCat.sleep();

return 0;

}



В этом примере класс Cat наследует от класса Animal. Класс Cat получает доступ к методам eat и sleep из класса Animal и добавляет свои собственные методы, такие как meow.

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

Использование наследования позволяет программисту использовать уже написанный и отлаженный код, экономит время и деньги, а также повышает надежность программ, поскольку код можно использовать в различных сценариях без изменения его основной логики.

1. *Какие члены базового класса не наследуются?*

**private**-члены базового класса не наследуются

1. *От чего зависит доступ к членам базового класса из произвольного класса?*

Доступ к членам базового класса из производного класса зависит от модификатора доступа, который применяется к членам базового класса. Если используется модификатор public или protected, то эти члены будут доступны в производном классе соответственно как public или protected. Если используется private, то они будут недоступны в производном классе.

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

**Идентификатор доступа** (public, protected, private) определяет видимость членов класса для внешних объектов. **Механизм доступа** (public, protected, private) в контексте наследования классов определяет, как наследуются эти члены в производном классе. Таким образом, идентификатор доступа указывает, кто видит члены класса, а механизм доступа определяет, как наследуются эти члены в производном классе.

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

| Модификатор доступа в базовом классе | Модификатор доступа в производном классе |

|-----------------------------------------------------|-----------------------------------------------------------|

| public | public |

| protected | protected |

| private | Недоступен в производном классе |

|-----------------------------------------------------|-----------------------------------------------------------|

* private-члены базового класса не наследуются;
* при public-наследовании уровень доступа члена класса не меняется;
* при private-наследовании наследуемые члены становятся private-членом производного класса;
* при protected-наследовании наследуемые члены становятся protected -членами производного класса.

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

class ПроизводныйКласс : [модификатор\_доступа] БазовыйКласс{

// тело производного класса

};

где [модификатор\_доступа] может быть public, protected или private.

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

Наследование бывает простое, множественное и многократное.

**Простое наследование** - это когда у производного класса имеется один базовый класс.

**Множественное наследование** - когда у производного класса имеется несколько базовых классов.

**Многократное наследование** – это когда производный класс может иметь и базовый и производный классы

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

Идентификатор доступа protected при наследовании классов позволяет членам базового класса быть доступными внутри производного класса, но они остаются недоступными для внешних объектов. Таким образом, члены с модификатором protected видны внутри производного класса и в его производных классах, но не видны за пределами их иерархии наследования.

1. *Какие существуют виды конструкторов при наследовании?*

* Конструктор базового класса
* Конструктор производного класса
* Деструктор базового класса
* Деструктор производного класса.

#include <iostream>

// Базовый класс

class BaseClass {

public:

// Конструктор базового класса

BaseClass(int baseValue) : baseField(baseValue) {

std::cout << "BaseClass constructor called" << std::endl;

}

// Деструктор базового класса

~BaseClass() {

std::cout << "BaseClass destructor called" << std::endl;

}

private:

int baseField;

};

// Производный класс

class DerivedClass : public BaseClass {

public:

// Конструктор производного класса

DerivedClass(int baseValue, int derivedValue) : BaseClass(baseValue), derivedField(derivedValue) {

std::cout << "DerivedClass constructor called" << std::endl;

}

// Деструктор производного класса

~DerivedClass() {

std::cout << "DerivedClass destructor called" << std::endl;

}

private:

int derivedField;

};

int main() {

// Создание объекта производного класса

DerivedClass obj(10, 20);

// Объект уничтожится при завершении блока main, вызывая деструкторы

return 0;

}

1. *Каков синтаксис объявления конструктора производного класса?*

class Derived : public Base {

public:

Derived() : Base() {

// Тело конструктора производного класса

}

};

В этом примере Derived - производный класс, который наследует от Base. В конструкторе Derived вызывается конструктор базового класса Base() с использованием инициализации базового класса (: Base()). Таким образом, сначала выполняется конструктор базового класса, а затем конструктор производного класса.

1. *Каковы особенности конструктора при простом наследовании?*

При простом наследовании конструктор производного класса автоматически вызывает конструктор базового класса, и члены базового класса инициализируются перед кодом тела производного класса.

1. *Каковы особенности конструктора при многократном наследовании?*

**При многократном наследовании конструкторы каждого базового класса вызываются в том порядке, в котором они перечислены в списке наследования производного класса.** Каждый конструктор инициализирует свою часть объекта, и этот процесс продолжается для всех базовых классов. Важно учесть, что если несколько базовых классов имеют общий базовый класс, конструктор общего базового класса вызывается только один раз.

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

Конструктор базового класса: Вызывается первым, при создании объекта производного класса.

Конструктор производного класса: Вызывается после конструктора базового класса.

Деструктор производного класса: Вызывается первым при уничтожении объекта.

Деструктор базового класса: Вызывается после деструктора производного класса.