Министерство образования и науки Кыргызской Республики

Кыргызский государственный технический университет

им. И.Раззакова

Факультет информационных технологий

Кафедра «Программное обеспечение компьютерных систем»

Направление: 710400 «Программная инженерия»

Дисциплина: « Объектно-ориентрованное проектирование / Объектно-ориентрованное программирование»

**ОТЧЕТ**

По лабораторной работе №6

Тема: «Наследование»

Выполнила: студент группы

ПИ(б)-3-21 Табылдиева Шоола

Проверил: Мусабаев Э. Б.

Бишкек – 2024

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.*

#include <iostream>

#include <string>

class Employee {

protected:

std::string name;

std::string position;

double salary;

public:

// Default constructor

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

// Parameterized constructor

Employee(const std::string& empName, const std::string& empPosition, double empSalary)

: name(empName), position(empPosition), salary(empSalary) {}

// Destructor

virtual ~Employee() {}

// Calculate annual salary

double calculateAnnualSalary() const {

return salary \* 12;

}

// Display information about the employee

void displayInfo() const {

std::cout << "Name: " << name << std::endl;

std::cout << "Position: " << position << std::endl;

std::cout << "Monthly Salary: $" << salary << std::endl;

std::cout << "Annual Salary: $" << calculateAnnualSalary() << std::endl;

}

};

class Manager : public Employee {

private:

double annual\_bonus;

std::string company\_car;

int stock\_options;

public:

// Default constructor

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

// Parameterized constructor

Manager(const std::string& empName, const std::string& empPosition, double empSalary,

double bonus, const std::string& car, int options)

: Employee(empName, empPosition, empSalary), annual\_bonus(bonus),

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

// Destructor

~Manager() {}

// Display information about the manager

void displayManagerInfo() const {

displayInfo(); // Inherited from the base class

std::cout << "Annual Bonus: $" << annual\_bonus << std::endl;

std::cout << "Company Car: " << company\_car << std::endl;

std::cout << "Stock Options: " << stock\_options << std::endl;

}

};

int main() {

// Test Employee class

Employee employee("John Doe", "Developer", 5000.0);

employee.displayInfo();

std::cout << "\n";

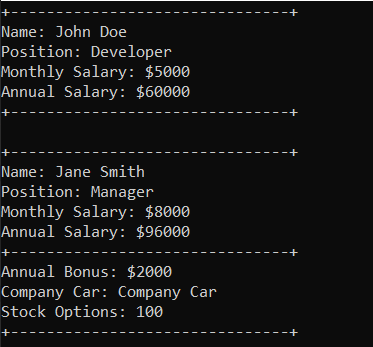
// Test Manager class

Manager manager("Jane Smith", "Manager", 8000.0, 2000.0, "Company Car", 100);

manager.displayManagerInfo();

return 0;

}



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().

#include <iostream>

#include <cstring>

class Display {

private:

char type[32];

long colors;

int x\_resolution;

int y\_resolution;

public:

// Parameterized constructor

Display(const char\* displayType, long displayColors, int resolutionX, int resolutionY)

: colors(displayColors), x\_resolution(resolutionX), y\_resolution(resolutionY) {

strcpy\_s(type, displayType);

}

const long getColors() const {

return colors;

}

// Display information about the Display

void displayInfo() const {

std::cout << "Display Type: " << type << std::endl;

std::cout << "Colors: " << colors << std::endl;

std::cout << "Resolution: " << x\_resolution << "x" << y\_resolution << std::endl;

}

};

class MotherBoard {

protected:

int processor;

int speed;

int RAM;

public:

// Parameterized constructor

MotherBoard(int processorType, int processorSpeed, int memoryRAM)

: processor(processorType), speed(processorSpeed), RAM(memoryRAM) {}

// Display information about the MotherBoard

void displayInfo() const {

std::cout << "Processor Type: " << processor << std::endl;

std::cout << "Processor Speed: " << speed << " GHz" << std::endl;

std::cout << "RAM: " << RAM << " GB" << std::endl;

}

};

class Computer : private Display, public MotherBoard {

private:

char name[64];

int hard\_disk;

public:

// Parameterized constructor

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

int resolutionX, int resolutionY, int processorType, int processorSpeed,

int memoryRAM, int diskSize)

: Display(displayType, displayColors, resolutionX, resolutionY),

MotherBoard(processorType, processorSpeed, memoryRAM), hard\_disk(diskSize) {

strcpy\_s(name, computerName);

}

// Display information about the Computer

void show() const {

std::cout << "COLORS FROM PARENT: " << this->getColors() << std::endl;

std::cout << "Computer Name: " << name << std::endl;

Display::displayInfo(); // Display information from Display

MotherBoard::displayInfo();

std::cout << "Hard Disk Size: " << hard\_disk << " GB" << std::endl;

}

};

int main() {

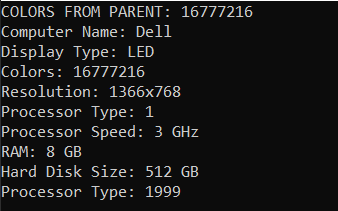
// Test Computer class

Computer myComputer("Dell", "LED", 16777216, 1366, 768, 1, 3, 8, 512);

myComputer.show();

return 0;

}



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.

Task 3

#include <iostream>

#include <string>

class Employee {

private:

std::string name;

std::string position;

double salary;

public:

// Default constructor

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

// Parameterized constructor

Employee(const std::string& empName, const std::string& empPosition, double empSalary)

: name(empName), position(empPosition), salary(empSalary) {}

// Destructor

virtual ~Employee() {}

// Calculate annual salary

double calculateAnnualSalary() const {

return salary \* 12;

}

// Display information about the employee

void displayInfo() const {

std::cout << "Name: " << name << std::endl;

std::cout << "Position: " << position << std::endl;

std::cout << "Monthly Salary: $" << salary << std::endl;

std::cout << "Annual Salary: $" << calculateAnnualSalary() << std::endl;

}

const std::string& getName() const {

return name;

}

};

class Manager : private Employee {

private:

double annual\_bonus;

std::string company\_car;

int stock\_options;

public:

// Default constructor

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

// Parameterized constructor

Manager(const std::string& empName, const std::string& empPosition, double empSalary,

double bonus, const std::string& car, int options)

: Employee(empName, empPosition, empSalary), annual\_bonus(bonus),

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

// Destructor

~Manager() {}

// Display information about the manager

void displayManagerInfo() const {

displayInfo(); // Inherited from the base class

std::cout << "Name FROM MANAGER CLASS: " << this->getName() << std::endl;

std::cout << "Annual Bonus: $" << annual\_bonus << std::endl;

std::cout << "Company Car: " << company\_car << std::endl;

std::cout << "Stock Options: " << stock\_options << std::endl;

}

};

int main() {

// Test Employee class

Employee employee("John Doe", "Developer", 5000.0);

employee.displayInfo();

std::cout << "\n";

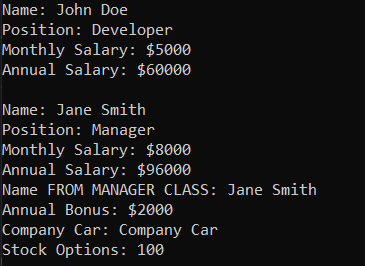
// Test Manager class

Manager manager("Jane Smith", "Manager", 8000.0, 2000.0, "Company Car", 100);

manager.displayManagerInfo();

return 0;

}



Ответы на вопросы

1. Что такое в ООП? Приведите примеры.

**Наследование —** механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса.

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

Выигрыш от него состоит в том, что наследование позволяет использовать существующий код несколько раз. Имея написанный и отлаженный базовый класс, мы можем его больше не модифицировать, при этом механизм наследования позволит нам приспособить его для работы в различных ситуациях. Используя уже написанный код, мы экономим время и деньги, а также увеличиваем надежность программы.

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?

В зависимости от типа наследования (public, private, protected) и доступности членов (public, private, protected).

1. Чем отличается идентификатор доступа от механизма доступа?

Идентификаторы доступа (public, private, protected) указывают на видимость членов класса, а механизм наследования определяет, как эти члены будут наследоваться в производных классах.

1. Назовите основные правила наследования (используйте таблицу наследования).
2. private-члены базового класса не наследуются;
3. при public-наследовании уровень доступа члена класса не меняется;
4. при private-наследовании наследуемые члены становятся private-членом производного класса;
5. при protected-наследовании наследуемые члены становятся protected -членами производного класса.
6. Синтаксис объявления производного класса, как наследника базового класса?

Class *производный\_класс*: тип\_наследования *базовый\_класс* {

программный код производного класса

};

1. Что такое простое, множественное и многократное наследование?

**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?

Члены класса, объявленные с идентификатором protected, доступны внутри самого класса и в производных классах. Это позволяет использовать protected для создания членов, которые могут быть использованы только внутри класса и его производных.

При использовании protected в базовом классе, все эти protected-члены наследуются производными классами. Это означает, что производный класс имеет доступ к protected-членам базового класса, как если бы они были членами самого производного класса.

protected позволяет обеспечивать уровень защиты для данных, который выше, чем у public, но ниже, чем у private. Это полезно, когда требуется, чтобы производный класс имел доступ к определенным данным базового класса, но не предоставлял полный открытый доступ извне.

1. Какие существуют виды конструкторов при наследовании?

Конструкторы при простом и множественном наследовании.

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?

При простом наследовании конструктор производного класса вызывает конструктор базового класса.

Конструктор базового класса вызывается автоматически до выполнения кода конструктора производного класса.

Инициализация членов базового класса происходит в списке инициализации конструктора производного класса.

1. Каковы особенности конструктора при многократном наследовании?

При многократном наследовании порядок вызова конструкторов базовых классов определяется порядком указания их имён в списке наследования.

Каждый конструктор базового класса вызывается в соответствии с указанным порядком.

Инициализация членов базовых классов происходит в списках инициализации конструктора производного класса в том порядке, в котором указаны базовые классы.

1. Какова последовательность вызова конструкторов и деструкторов при наследовании?

При создании объектов производного класса сначала вызывается конструктор базового класса, а уже затем конструктор класса производного. Правило вызова деструкторов состоит в том, что деструкторы вызываются в обратном порядке, если сравнить с последовательностью вызова конструкторов. Первым вызывается деструктор производного клааса, а после этого деструкторы базовых классов (очередность вызова деструкторов – справа налево в списке наследования базовых классов).