**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И. Раззакова**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

Дисциплина «**Объектно-Ориентированное Программирование**»

Отчет

Лабораторная работа №6

Выполнил: студент группы ПИ-2-21

Тойчубекова Айчурок

Проверил: Мусабаев Э. Б.

**Бишкек 2024**

**Лабораторная работа №6**

**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.*

#include <iostream>

#include <string>

using namespace std;

// Родительский класс Employee

class Employee {

protected:

string name;

string position;

double salary;

public:

Employee() {}

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

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

~Employee() {}

double calculateAnnualSalary() {

return salary \* 12;

}

void display() {

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

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

cout << "Salary: $" << salary << "/month" << endl;

}

};

// Класс-потомок Manager

class Manager : public Employee {

private:

double annual\_bonus;

string company\_car;

int stock\_options;

public:

Manager() {}

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) {}

~Manager() {}

void display() {

Employee::display(); // Вызов метода display() из родительского класса

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

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

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

}

};

int main() {

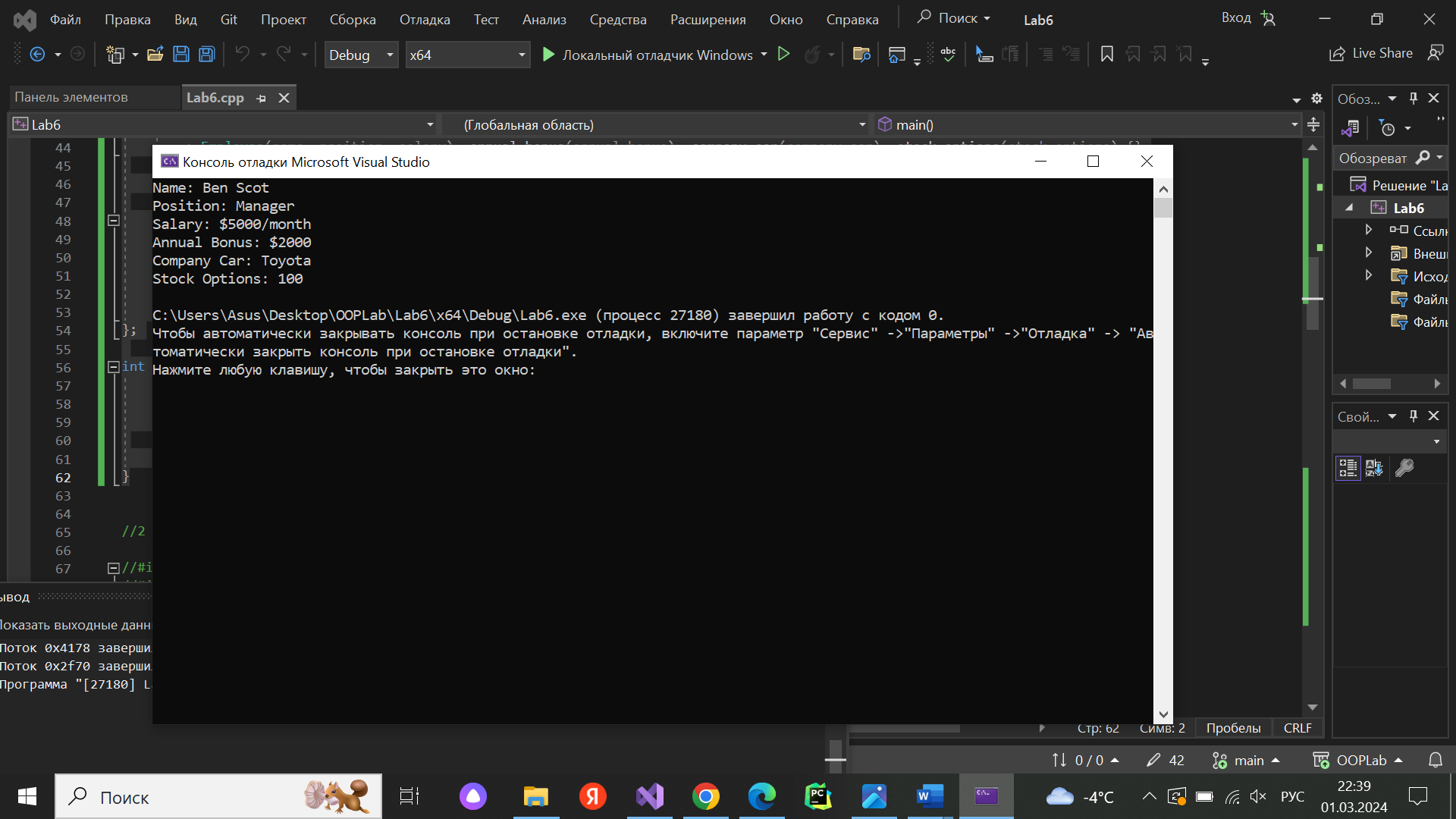
// Пример использования классов

Manager manager("John Doe", "Manager", 5000, 2000, "Toyota", 100);

manager.display();

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 <string>

using namespace std;

// Родительский класс Display

class Display {

protected:

string type;

long colors;

int x\_resolution;

int y\_resolution;

public:

Display(const string& type, long colors, int x\_resolution, int y\_resolution)

: type(type), colors(colors), x\_resolution(x\_resolution), y\_resolution(y\_resolution) {}

void display() {

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

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

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

}

};

// Родительский класс MotherBoard

class MotherBoard {

protected:

int processor;

int speed;

int RAM;

public:

MotherBoard(int processor, int speed, int RAM)

: processor(processor), speed(speed), RAM(RAM) {}

void display() {

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

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

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

}

};

// Класс-потомок Computer

class Computer : public Display, public MotherBoard {

private:

string name;

int hard\_disk;

public:

Computer(const string& type, long colors, int x\_resolution, int y\_resolution,

int processor, int speed, int RAM, const string& name, int hard\_disk)

: Display(type, colors, x\_resolution, y\_resolution), MotherBoard(processor, speed, RAM), name(name), hard\_disk(hard\_disk) {}

void Show() {

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

Display::display(); // Вызов метода display() из класса Display

MotherBoard::display(); // Вызов метода display() из класса MotherBoard

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

}

};

int main() {

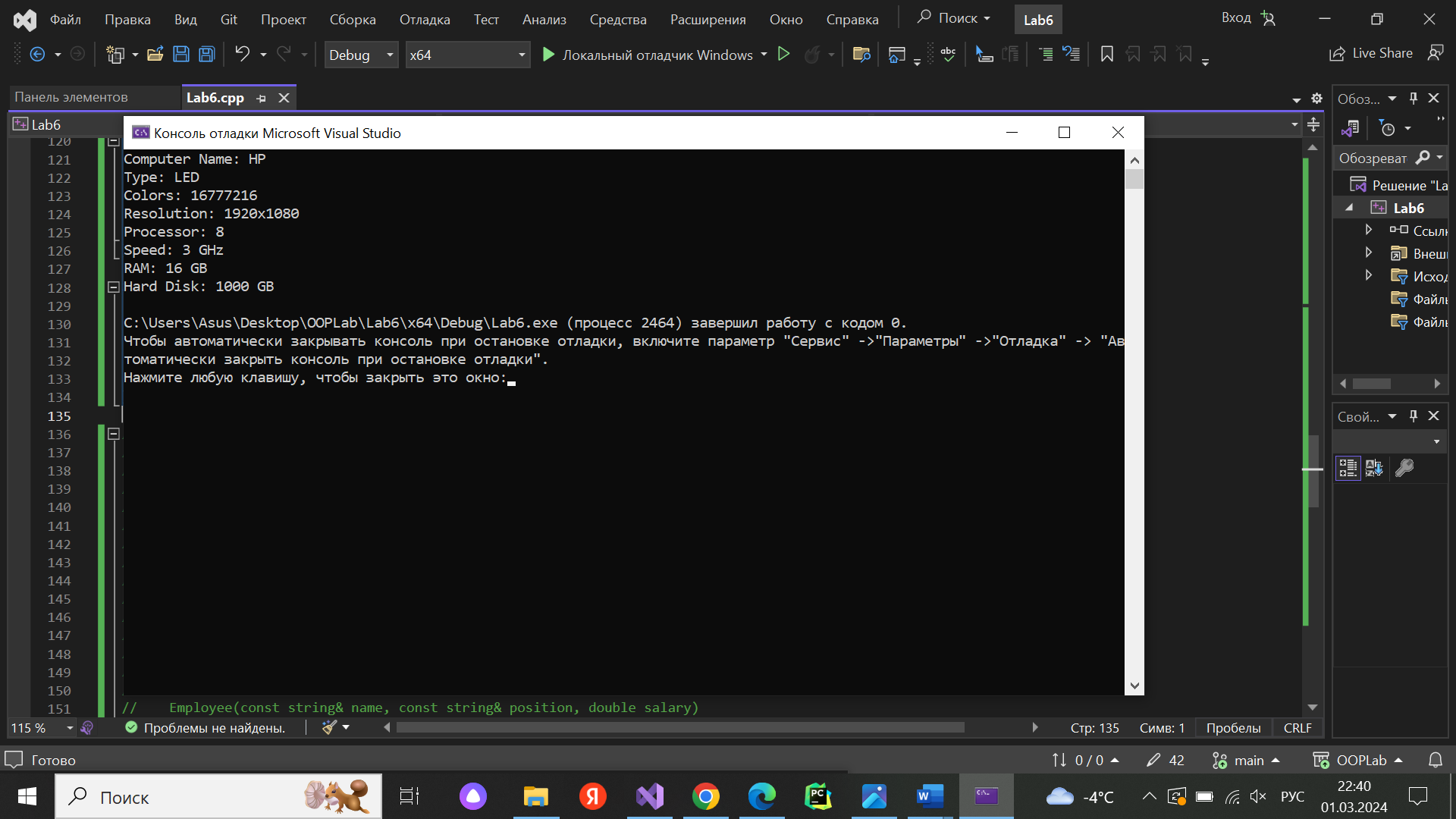
// Пример использования класса Computer

Computer computer("LED", 16777216, 1920, 1080, 8, 3, 16, "HP", 1000);

computer.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.

#include <iostream>

#include <string>

using namespace std;

// Родительский класс Employee

class Employee {

private:

string name;

string position;

double salary;

public:

Employee() {}

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

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

~Employee() {}

double calculateAnnualSalary() {

return salary \* 12;

}

void display() {

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

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

cout << "Salary: $" << salary << "/month" << endl;

}

// Доступ к закрытым членам базового класса из производного класса

friend class Manager;

};

// Класс-потомок Manager

class Manager : public Employee {

private:

double annual\_bonus;

string company\_car;

int stock\_options;

public:

Manager() {}

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) {}

~Manager() {}

void display() {

Employee::display(); // Вызов метода display() из родительского класса

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

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

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

}

};

int main() {

// Пример использования классов

Manager manager("John Doe", "Manager", 5000, 2000, "Toyota", 100);

manager.display();

return 0;

}

