Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

кафедра ПМ иК

КУРСОВАЯ РАБОТА
по дисциплине «Объектно-ориентированное программирование»

Тема: «Программа с использованием объектно-ориентированных технологий на тему “Студенты”»

Выполнил: студент группы ИС-242
Журбенко Владислав Евгеньевич
Проверил: ассистент кафедры ПМиК
Сороковых Дарья Анатольевна

Оценка _____

Новосибирск – 2023г.

Содержание

1. **Постановка задачи.**
2. **Теория:**
   a) **Полиморфизм;**
   b) **Абстракция;**
   c) **Инкапсуляция;**
   d) **Наследование.**
3. **Иерархия объектов.**
4. **Описание алгоритма.**
5. **Результат работы.**
6. **Приложение. Листинг.**
7. **GitHub: https://github.com/xCredo/kurs_oop.git**

Постановка задачи.

Написать программу с использованием объектно-ориентированных технологий. Количество созданных классов – не менее трёх.

Какие объекты должна описывать иерархия классов, выбирается по таблице согласно своему варианту (вариант определяется по последней цифре зачетной книжки).

Постановка задачи, Содержимое классов – Ваше творческое решение. В таблице к каждой теме приводится пример возможных полей данных и действий. Таким образом, результатом работы будет: иерархия классов и программа с каким-либо примером работы с объектами этих классов.

Язык и Среда разработки: C++ (Dev-C++, Visual Studio и др.)

Использовать простые алгоритмы, позволяющие понять применяемую технологию.

Теория:

Полиморфизм

Полиморфизм - это принцип объектно-ориентированного программирования, который описывает возможность объектов различных классов использовать одинаковые интерфейсы или методы, но давать различные реализации этих методов.

Полиморфизм позволяет обрабатывать объекты производных классов как объекты базового класса, что делает код более гибким и удобным для использования. Принцип полиморфизма включает в себя переопределение методов (полиморфизм через наследование) и полиморфизм через интерфейсы.

Полиморфизм через наследование позволяет производным классам переопределить методы базового класса с собственной реализацией, при этом код, который использует базовый класс, может вызывать эти методы, не зависимо от фактического типа объекта. Это позволяет программисту работать с объектами различных классов, используя общие интерфейсы или абстрактные классы.

Полиморфизм через интерфейсы предполагает, что разные классы могут реализовать один и тот же интерфейс, но давать различные реализации. Это позволяет использовать объекты разных классов через общий интерфейс, обеспечивая единые способы взаимодействия с ними.

Пример в коде:
Программа демонстрирует полиморфизм через использование виртуальных функций displayInfo() в классах **PaidStudent** и **BudgetStudent**, которые переопределены от абстрактного класса **Student**. Полиморфизм позволяет вызывать соответствующую реализацию метода в зависимости от типа объекта.

Абстракция

Абстракция - это принцип объектно-ориентированного программирования, который предоставляет возможность создания упрощенных, концептуальных моделей объектов и их свойств, игнорируя лишние детали реализации.

В контексте ООП, абстракция позволяет определить общие характеристики и поведение объектов, выделяя их отличительные черты и роли в системе. Она

позволяет разработчику сконцентрироваться на существенных аспектах объектов и игнорировать малозначительные детали.

Абстракция часто реализуется через абстрактные классы или интерфейсы. Абстрактный класс определяет общие свойства и методы для группы связанных классов, но не предоставляет конкретную реализацию для всех методов, оставляя их на уровне производных классов. Интерфейс определяет набор методов, которые должен реализовать класс, но не предписывает, как эти методы должны быть реализованы.

Пример в коде:
Программа использует абстрактный класс **Student**, который определяет общие свойства и методы для всех студентов, а наследуемые классы **PaidStudent** и **BudgetStudent** предоставляют конкретные реализации.

Инкапсуляция

Инкапсуляция - это принцип объектно-ориентированного программирования, который объединяет данные (поля) и методы, оперирующие этими данными, внутри класса, и скрывает их от прямого доступа извне. Основная идея инкапсуляции заключается в том, что объекты должны иметь четко определенный интерфейс, через который можно выполнять операции с объектами, в то время как внутренняя структура и детали реализации остаются скрытыми.

Принцип инкапсуляции помогает обеспечить контролируемый доступ к данным и методам класса, предотвращая модификацию или неправильное использование данных извне. Для этого используются модификаторы доступа, такие как публичный (public), приватный (private), защищенный (protected), которые определяют, какие члены класса могут быть доступны извне, какие могут быть доступны только внутри класса или его подклассов.

Важными понятиями, связанными с инкапсуляцией, являются сокрытие информации и согласованность интерфейса.

Пример в коде:

В классах **Student**, **PaidStudent** и **BudgetStudent** данные о студентах (имя, возраст, группа и т.д.) скрыты от внешнего доступа, а методы-геттеры предоставляют контролируемый доступ к ним.

Наследование

Наследование - это принцип объектно-ориентированного программирования, который позволяет создавать новые классы на основе существующих классов. В рамках наследования, существующий класс, называемый родительским или базовым классом, передает свои свойства (поля и методы) новому классу, который называется дочерним или производным классом.

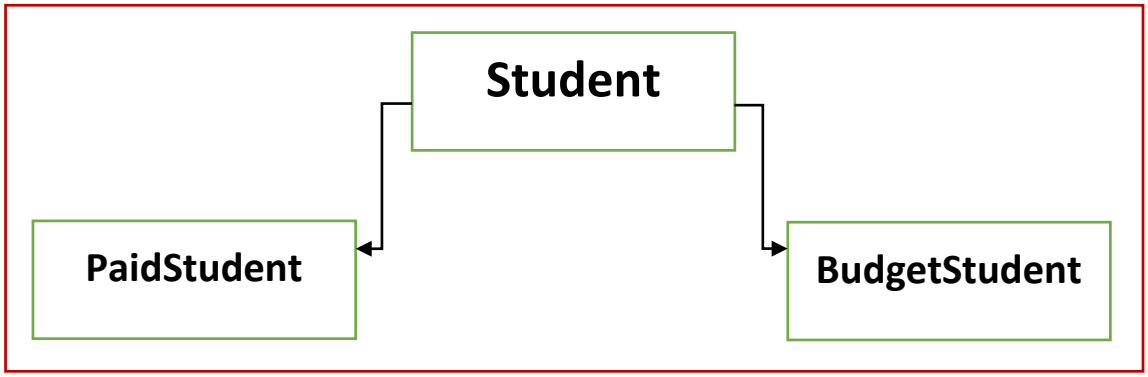
Дочерний класс наследует все свойства родительского класса и может расширять их, добавлять новые свойства и методы, или переопределять унаследованные методы для своих нужд. Это позволяет создавать иерархию классов, группируя их по общим характеристикам и поведению.

Принцип наследования позволяет повторно использовать код, минимизировать дублирование и упростить проектирование иерархии классов. Он также способствует созданию более абстрактной и гибкой архитектуры программы, позволяя работать с объектами различных классов, производных от одного базового класса, с помощью общего интерфейса.

Пример в коде:

Классы **PaidStudent** и **BudgetStudent** наследуются от базового класса **Student**. Они наследуют его свойства и методы, а также добавляют свои собственные, соответствующие различным типам студентов.

Иерархия объектов



Абстрактный класс Student является базовым для иерархии.
Класс PaidStudent и BudgetStudent наследуются от класса Student.
У каждого класса есть свои уникальные поля и методы:

Класс Student:

Метод Student (конструктор): принимает параметры name (имя студента), age (возраст студента) и group (группа, в которой учится студент). Создает объект класса Student и инициализирует соответствующие поля.

Метод getName() const: возвращает имя студента.

Метод getAge() const: возвращает возраст студента.

Метод getGroup() const: возвращает группу студента.

Класс PaidStudent, унаследованный от класса Student:

Метод PaidStudent (конструктор): принимает параметры name, age, group (унаследованные от класса Student) и major (основная специализация студента). Создает объект класса PaidStudent и инициализирует соответствующие поля.

Метод getMajor() const: возвращает основную специализацию студента.

Класс BudgetStudent, унаследованный от класса Student:

Метод BudgetStudent (конструктор): принимает параметры name, age, group (унаследованные от класса Student) и previousEducation (предыдущее образование

студента). Создает объект класса BudgetStudent и инициализирует соответствующие поля.

Метод getPreviousEducation() const: возвращает предыдущее образование студента.

Наиболее важные методы классов **Student**, **PaidStudent** и **BudgetStudent** - это конструкторы, которые инициализируют объекты классов с заданными значениями атрибутов. Классы также имеют методы доступа (геттеры), которые позволяют получить значения атрибутов объектов классов.

Описание Алгоритма

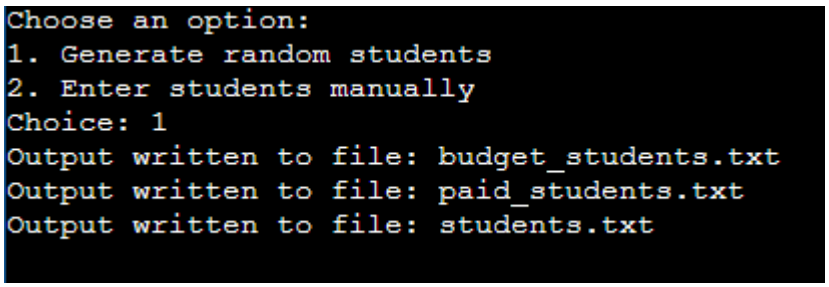
1. Программа спрашивает пользователя о выборе опции: генерация случайных студентов или ввод студентов вручную.

2. В зависимости от выбора, программа генерирует случайных студентов или просит пользователя ввести суммарное число студентов.

3. Для каждого студента создаются объекты класса **PaidStudent** или **BudgetStudent**, используя случайно выбранные данные.

4. Информация о каждом студенте записывается в соответствующие файлы: "budget_students.txt", "paid_students.txt" и "students.txt", где хранится общий список всех студентов без разделения на Paid & Budget.

5. Функция writeOutputToFile() используется для записи информации в файлы.



Результат работы

main.cpp | budget_students.txt | paid_students.txt | students.txt

```
Students Details:


-------------------------

1
Name: Елизабет
Age: 21
Group: Группа ИС-243
Previous Education: College
-------------------------
2
Name: Анна
Age: 18
Group: Группа ИС-242
Major: Engineering
-------------------------
3
Name: Виталя
Age: 21
Group: Группа ИС-242
Major: Engineering
-------------------------
4
```

```
Paid Student Details:


-------------------------

2.
Name: Анна
Age: 18
Group: Группа ИС-242
Major: Engineering
-------------------------

3.
Name: Виталя
Age: 21
Group: Группа ИС-242
Major: Engineering
-------------------------

7.
Name: Виталя
Age: 18
Group: Группа ИС-242
Major: Engineering
-------------------------
```

```
Budget Student Details:


--------------------------

1.
Name: Елизабет
Age: 21
Group: Группа ИС-243
Previous Education: College
--------------------------

4.
Name: Анна
Age: 21
Group: Группа ИС-242
Previous Education: College
--------------------------

5.
Name: Изобелла
Age: 19
Group: Группа ИС-242
Previous Education: High School
--------------------------
```

Приложение. Листинг.

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <ctime>

const std::string names[] = {"Владислав", "Елизабет", "Михаил", "Ольга", "Виталя",
"Анна", "Георгий", "Софья", "Борис", "Изобелла"};
const std::string groups[] = {"Группа ИС-241", "Группа ИС-242", "Группа ИС-243"};
const int ages[] = {18, 19, 20, 21};

int getRandomNumber(int min, int max) {
    static const double fraction = 1.0 / (RAND_MAX + 1.0);
    return min + static_cast<int>((max - min + 1) * (std::rand() * fraction));
}

void writeOutputToFile(const std::string& filename, const std::string& content) {
    std::ofstream outputFile(filename);
    if (outputFile.is_open()) {
        outputFile << content;
        outputFile.close();
        std::cout << "Output written to file: " << filename << "\n";
    }

    else {
        std::cerr << "Unable to open file: " << filename << "\n";
    }
}

class Student {
public:
    Student(const std::string& name, int age, const std::string& group)
        : name(name), age(age), group(group) {}

    std::string getName() const {
        return name;
    }

    int getAge() const {
        return age;
    }

    std::string getGroup() const {
        return group;
    }

private:
    std::string name;
    int age;
    std::string group;
};

class PaidStudent : public Student {
```

```
public:
    PaidStudent(const std::string& name, int age, const std::string& group, const
std::string& major)
        : Student(name, age, group), major(major) {}

    std::string getMajor() const {
        return major;
    }

private:
    std::string major;
};

class BudgetStudent : public Student {
public:
    BudgetStudent(const std::string& name, int age, const std::string& group, const
std::string& previousEducation)
        : Student(name, age, group), previousEducation(previousEducation) {}

    std::string getPreviousEducation() const {
        return previousEducation;
    }

private:
    std::string previousEducation;
};

int main() {
    std::srand(static_cast<unsigned int>(std::time(nullptr)));

    int choice;
    std::cout << "Choose an option:\n";
    std::cout << "1. Generate random students\n";
    std::cout << "2. Enter students manually\n";
    std::cout << "Choice: ";
    std::cin >> choice;

    if (choice == 1 || choice == 2) {
        int numStudents;
        if (choice == 1) {
            numStudents = getRandomNumber(1, 30);
        }

        else {
            std::cout << "Enter the number of students: ";
            std::cin >> numStudents;
        }

        std::string paidOutputContent = "Paid Student Details:\n";
        paidOutputContent += "\n\n-------------------------\n\n";

        std::string budgetOutputContent = "Budget Student Details:\n";
        budgetOutputContent += "\n\n-------------------------\n\n";

        std::string studentsOutputContent = "Students Details:\n";
```

```
        studentsOutputContent += "\n\n-------------------------\n\n";

        for (int i = 0; i < numStudents; ++i) {
            std::string name = names[getRandomNumber(0, 9)];
            int age = ages[getRandomNumber(0, 3)];
            std::string group = groups[getRandomNumber(0, 2)];
            std::string previousEducation = (age >= 20) ? "College" : "High School";

            if (getRandomNumber(0, 1) == 0) {
                std::string major = "Engineering";
                PaidStudent paidStudent(name, age, group, major);
                paidOutputContent += std::to_string(i+1) + ".\n";
                paidOutputContent += "Name: " + paidStudent.getName() + "\n";
                paidOutputContent += "Age: " + std::to_string(paidStudent.getAge())
+ "\n";
                paidOutputContent += "Group: " + paidStudent.getGroup() + "\n";
                paidOutputContent += "Major: " + major + "\n";
                paidOutputContent += "-------------------------\n\n";
                studentsOutputContent += std::to_string(i+1) + "\n";
                studentsOutputContent += "Name: " + paidStudent.getName() + "\n";
                studentsOutputContent += "Age: " +
std::to_string(paidStudent.getAge()) + "\n";
                studentsOutputContent += "Group: " + paidStudent.getGroup() + "\n";
                studentsOutputContent += "Major: " + major + "\n";
                studentsOutputContent += "-------------------------\n";
            }

            else {
                BudgetStudent budgetStudent(name, age, group, previousEducation);
                budgetOutputContent += std::to_string(i+1) + ".\n";
                budgetOutputContent += "Name: " + budgetStudent.getName() + "\n";
                budgetOutputContent += "Age: " +
std::to_string(budgetStudent.getAge()) + "\n";
                budgetOutputContent += "Group: " + budgetStudent.getGroup() + "\n";
                budgetOutputContent += "Previous Education: " + previousEducation +
"\n";
                budgetOutputContent += "-------------------------\n\n";
                studentsOutputContent += std::to_string(i+1) + "\n";
                studentsOutputContent += "Name: " + budgetStudent.getName() + "\n";
                studentsOutputContent += "Age: " +
std::to_string(budgetStudent.getAge()) + "\n";
                studentsOutputContent += "Group: " + budgetStudent.getGroup() +
"\n";
                studentsOutputContent += "Previous Education: " + previousEducation
+ "\n";
                studentsOutputContent += "-------------------------\n";
            }
        }

        if (choice == 1 || choice == 2) {
            writeOutputToFile("budget_students.txt", budgetOutputContent);
            writeOutputToFile("paid_students.txt", paidOutputContent);
            writeOutputToFile("students.txt", studentsOutputContent);

        }
```

```
    }

    else {
        std::cout << "Invalid choice. Exiting...\n";
        return 1;
    }

    return 0;
}
```