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 №7

Тема: «Виртуальные методы»

Выполнила: студентка группы

ПИ(б)-3-21 Пак Ксения

Проверил: Мусабаев Э. Б.

Бишкек – 2024

Задания

1. Создать базовый класс Array с полями: массив типа unsigned и поле для хранения количества элементов у текущего объекта массива. Максимально возможный размер массива задается статической константой. Реализовать конструктор инициализации, задающий количество элементов и начальное значение (по умолчанию 0). Реализовать в классе Array виртуальную функцию поэлементного сложения массивов. Реализовать два класса, переопределив виртуальную функцию сложения. Вызывающая программа должна продемонстрировать все варианты вызова виртуальных функций.

#include <iostream>

class Array {

protected:

static const int maxSize = 100;

unsigned arr[maxSize];

int size;

public:

Array(int newSize, unsigned initialValue = 0) : size(newSize) {

for (int i = 0; i < size; ++i) {

arr[i] = initialValue;

}

}

unsigned getElement(int index) const {

if (index >= 0 && index < size) {

return arr[index];

}

return 0;

}

virtual void addArray(const Array& other) const {

std::cout << "Base Array Addition:" << std::endl;

for (int i = 0; i < size; ++i) {

std::cout << getElement(i) + other.getElement(i) << " ";

}

std::cout << std::endl;

}

};

class ArrayAddition1 : public Array {

public:

ArrayAddition1(int newSize, unsigned initialValue = 0) : Array(newSize, initialValue) {}

void addArray(const Array& other) const override {

std::cout << "ArrayAddition1:" << std::endl;

for (int i = 0; i < size; ++i) {

std::cout << arr[i] + other.getElement(i) << " ";

}

std::cout << std::endl;

}

};

class ArrayAddition2 : public Array {

public:

ArrayAddition2(int newSize, unsigned initialValue = 0) : Array(newSize, initialValue) {}

void addArray(const Array& other) const override {

std::cout << "ArrayAddition2:" << std::endl;

for (int i = 0; i < size; ++i) {

std::cout << arr[i] + other.getElement(i) << " ";

}

std::cout << std::endl;

}

};

int main() {

Array baseArray(3, 1);

ArrayAddition1 array1(3, 2);

ArrayAddition2 array2(3, 3);

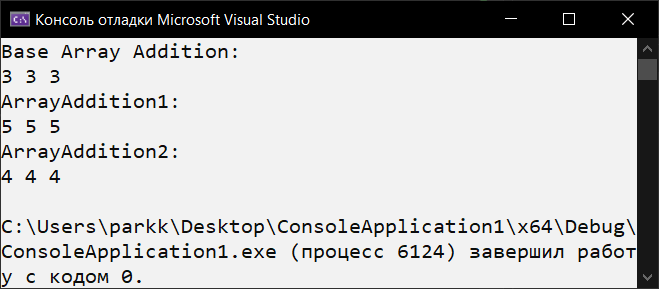
baseArray.addArray(array1);

array1.addArray(array2);

array2.addArray(baseArray);

return 0;

}



1. Создать абстрактный базовый класс Figure с виртуальными методами вычисления площади и периметра. Создать производные класса: Rectangle (прямоугольник), Circle (круг), Trapezium (трапеция) со своими функциями площади и периметра. Самостоятельно определить, какие поля необходимы, какие из них можно задать в базовом классе, а какие в производных. Площадь трапеции: *S = (a + b) x h/2.*

#include <iostream>

class Figure {

public:

virtual double calculateArea() const = 0;

virtual double calculatePerimeter() const = 0;

virtual ~Figure() {}

};

class Rectangle : public Figure {

private:

double length;

double width;

public:

Rectangle(double len, double wid) : length(len), width(wid) {}

double calculateArea() const override {

return length \* width;

}

double calculatePerimeter() const override {

return 2 \* (length + width);

}

};

class Circle : public Figure {

private:

double radius;

public:

Circle(double rad) : radius(rad) {}

double calculateArea() const override {

return 3.14 \* radius \* radius;

}

double calculatePerimeter() const override {

return 2 \* 3.14 \* radius;

}

};

class Trapezium : public Figure {

private:

double a;

double b;

double left;

double right;

double height;

public:

Trapezium(double baseA, double baseB, double leftSide, double rightSide, double h) : a(baseA), b(baseB), height(h), left(leftSide), right(rightSide) {}

double calculateArea() const override {

return (a + b) \* height / 2;

}

double calculatePerimeter() const override {

return a + b + left + right;

}

};

int main() {

//Figure figure;

Rectangle rectangle(5, 3);

Circle circle(4);

Trapezium trapezium(4, 8, 3, 4, 5);

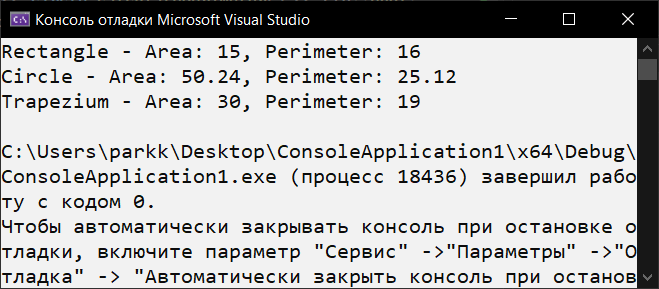
std::cout << "Rectangle - Area: " << rectangle.calculateArea() << ", Perimeter: " << rectangle.calculatePerimeter() << std::endl;

std::cout << "Circle - Area: " << circle.calculateArea() << ", Perimeter: " << circle.calculatePerimeter() << std::endl;

std::cout << "Trapezium - Area: " << trapezium.calculateArea() << ", Perimeter: " << trapezium.calculatePerimeter() << std::endl;

return 0;

}



1. Создать абстрактный базовый класс Currency (валюта) для работы с денежными суммами. Определить виртуальные функции перевода в рубли и вывода на экран. Реализовать производные классы Dollar (доллар) и Euro (евро) со своими функциями перевода и вывода на экран.

#include <iostream>

class Currency {

public:

virtual void toRubles(double amount) const = 0;

};

class Dollar : public Currency {

public:

void toRubles(double amount) const override {

double rubles = amount \* 90.69;

std::cout << "Amount " << amount << " in Rubles : " << rubles << std::endl;

}

};

class Euro : public Currency {

public:

void toRubles(double amount) const override {

double rubles = amount \* 99.24;

std::cout << "Amount " << amount << " in Rubles: " << rubles << std::endl;

}

};

int main() {

Dollar dollar;

Euro euro;

double amount;

std::cout << "Enter amount in Dollars: ";

std::cin >> amount;

dollar.toRubles(amount);

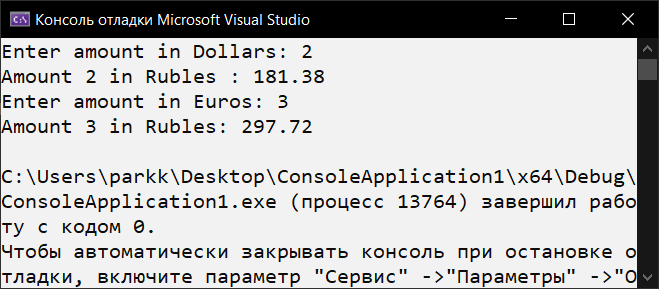
std::cout << "Enter amount in Euros: ";

std::cin >> amount;

euro.toRubles(amount);

return 0;

}



Ответы на вопросы

1. Что такое виртуальная функция (метод) при наследовании?

**Виртуальный метод** – это метод, используется в базовом классе, который дает возможность переопределения методов в производном классе при наследовании. Виртуальные методы описываются с помощью ключевого слова virtual в базовом классе. При наследовании виртуальный метод может быть переопределен и перегружен. При перегрузке меняется прототип метода.

1. Чем отличается переопределение метода от его перегрузки при наследовании?

Переопределение – это процесс предоставления новой реализации метода в производном классе, который уже определен в базовом классе.

class Shape {

public:

virtual void draw() const {

// Реализация в базовом классе

}

};

class Circle : public Shape {

public:

void draw() const override {

// Новая реализация в производном классе

}

};

Перегрузка – это процесс создания нескольких методов с одним и тем же именем в одном классе, но с разными параметрами (типами или количеством).

class Calculator {

public:

int add(int a, int b) {

// Реализация для двух целых чисел

}

double add(double a, double b) {

// Реализация для двух вещественных чисел

}

};

1. Что такое чисто виртуальная функция? Для чего она используется?

**Чисто виртуальной функцией** называется такая виртуальная функция, которая не имеет определения в базовом классе. Чтобы определить чисто виртуальную функцию, после прототипа функции необходимо указать оператор присваивания (знак равенства) и ноль. Для чисто виртуальной функции блок с кодом функции (тело) не указывается. В производном классе чисто виртуальная функция должна быть переопределена, поскольку из базового класса эта функция вызвана быть не может, там для неё нет программного кода.

Цель подобных функций - просто определить функционал без реализации, а реализацию определят производные классы.

1. Синтаксис виртуальной функции?

virtual тип\_результата имя-функции (аргументы)=0;

1. Как называется класс, содержащий хотя бы одну чисто виртуальную функцию?

Класс, содержащий хотя бы одну чисто виртуальную функцию, называется абстрактным.

1. Приведите пример иерархической структуры при наследовании?

Базовый (абстрактный) класс определяет структуру всех производных классов (каждый класс должен иметь поле, определяющее линейные размеры фигуры, и метод для вычисления фигуры). Конкретная реализация способа вычисления площади фигуры определяется в каждом производном классе по-своему.

1. Как использовать виртуальные методы в многоуровневом наследовании?

Виртуальные методы в многоуровневом наследовании в C++ используются аналогично одноуровневому наследованию. Виртуальные методы предоставляют механизм полиморфизма, что позволяет вызывать методы производного класса через указатель или ссылку на базовый класс.