Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

ОТЧЕТ

О ЛАБОРАТОРНОЙ РАБОТЕ № 4

Виртуальные функции. Абстрактные классы

по дисциплине «Программирование на языках высокого уровня»

Выполнил ст. гр. 450503 Д.С. Абрамчук

Проверил асс. каф. ЭВМ И.Г. Скиба

Минск 2025

# **1 ПОСТАНОВКА ЗАДАЧИ**

Реализовать абстрактный класс Shape, содержащий интерфейс иерархии. Создать производные абстрактные классы TwoDShape и ThreeDShape, от которых унаследовать всевозможные конкретные формы. Реализовать виртуальные функции print (для вывода типа и размера объектов каждого класса), area, draw и volume.

# **2 ДИАГРАММА КЛАССОВ**

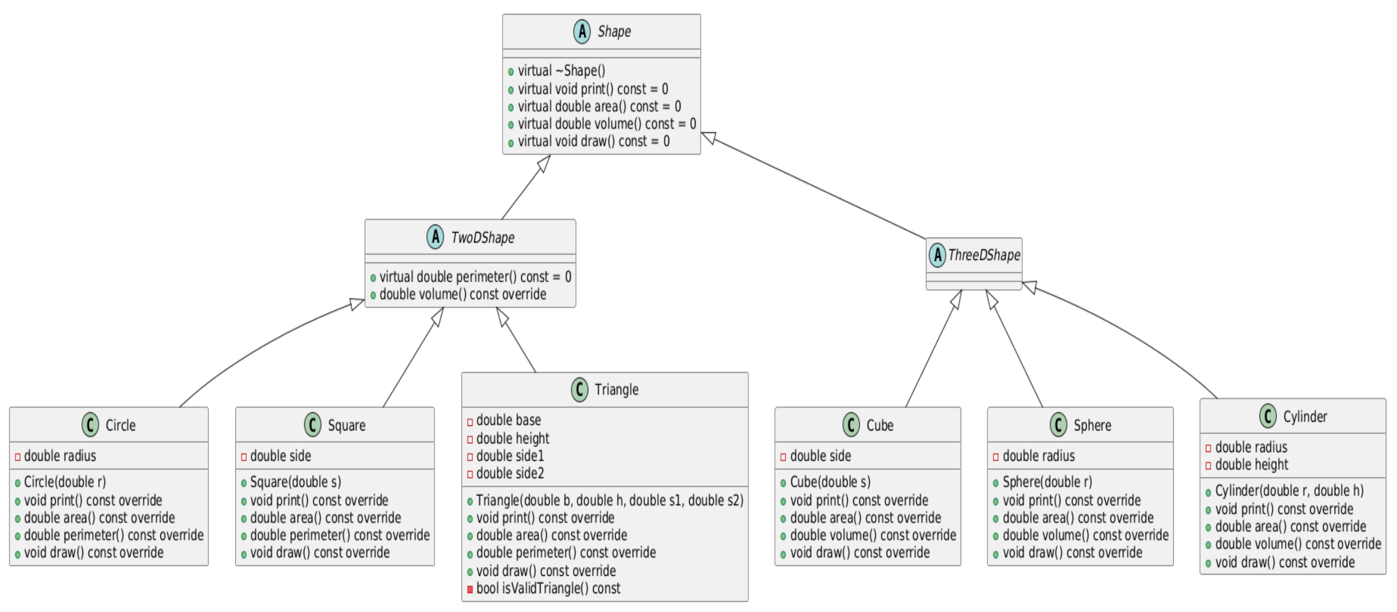


Рисунок 2.1 – диаграмма классов

# **3 ЛИСТИНГ КОДА**

Файл main.cpp

#include <iostream>  
#include "../header /Circle.h"  
#include "../header /function.h"  
  
  
using namespace std;  
  
int main() {  
 auto shapes = new Shape\*[10];  
 int shapeCount = 0;  
 int capacity = 10;  
  
 int choice;  
  
 while (true) {  
 displayMenu();  
 cin >> choice;  
  
 if (cin.fail()) {  
 cout << "Invalid input! Please enter a number." << endl;  
 clearInputBuffer();  
 continue;  
 }  
 clearInputBuffer();  
  
 switch (choice) {  
 case 1:  
 addCircle(shapes, shapeCount, capacity);  
 break;  
 case 2:  
 addSquare(shapes, shapeCount, capacity);  
 break;  
 case 3:  
 addTriangle(shapes, shapeCount, capacity);  
 break;  
 case 4:  
 addSphere(shapes, shapeCount, capacity);  
 break;  
 case 5:  
 addCube(shapes, shapeCount, capacity);  
 break;  
 case 6:  
 addCylinder(shapes, shapeCount, capacity);  
 break;  
 case 7:  
 showAllShapes(span<Shape\*>(shapes, shapeCount));  
 break;  
 case 8:  
 show2DShapes(span<Shape\*>(shapes, shapeCount));  
 break;  
 case 9:  
 show3DShapes(span<Shape\*>(shapes, shapeCount));  
 break;  
 case 10:  
 calculateTotalArea(span<Shape\*>(shapes, shapeCount));  
 break;  
 case 11:  
 deleteAllShapes(shapes, shapeCount, capacity);  
 break;  
 case 0:  
 for (int i = 0; i < shapeCount; ++i) {  
 delete shapes[i];  
 }  
 delete[] shapes;  
 cout << "Program is over" << endl;  
 return 0;  
 default:  
 cout << "Invalid choice! Please try again." << endl;  
 }  
 }  
}

Файл function.cpp

#include <iostream>  
#include "../header /Circle.h"  
#include "../header /Cube.h"  
#include "../header /Cylinder.h"  
#include "../header /Sphere.h"  
#include "../header /Square.h"  
#include "../header /Triangle.h"  
#include <limits>  
#include <stdexcept>  
#include <span>  
  
using namespace std;  
  
void clearInputBuffer() {  
 cin.clear();  
 cin.ignore(numeric\_limits<streamsize>::max(), '\n');  
}  
  
double getPositiveInput(const string &prompt) {  
 double value;  
 while (true) {  
 cout << prompt;  
 cin >> value;  
 if (cin.fail() || value <= 0) {  
 cout << "Error: Please enter a positive number!" << endl;  
 clearInputBuffer();  
 } else {  
 clearInputBuffer();  
 return value;  
 }  
 }  
}  
  
void displayMenu() {  
 cout << "\n MENU " << endl;  
 cout << "1. Add Circle" << endl;  
 cout << "2. Add Square" << endl;  
 cout << "3. Add Triangle" << endl;  
 cout << "4. Add Sphere" << endl;  
 cout << "5. Add Cube" << endl;  
 cout << "6. Add Cylinder" << endl;  
 cout << "7. Show All Shapes" << endl;  
 cout << "8. Show Only 2D Shapes" << endl;  
 cout << "9. Show Only 3D Shapes" << endl;  
 cout << "10. Calculate Total Area" << endl;  
 cout << "11. Delete All Shapes" << endl;  
 cout << "0. Exit" << endl;  
 cout << "Choose number: ";  
}  
  
void resizeShapesArray(Shape \*\*&shapes, int &capacity, int shapeCount) {  
 int newCapacity = capacity \* 2;  
 auto newShapes = new Shape \*[newCapacity];  
  
 for (int i = 0; i < shapeCount; ++i) {  
 newShapes[i] = shapes[i];  
 }  
  
 delete[] shapes;  
 shapes = newShapes;  
 capacity = newCapacity;  
}  
  
void addCircle(Shape \*\*&shapes, int &shapeCount, int &capacity) {  
 if (shapeCount >= capacity) {  
 resizeShapesArray(shapes, capacity, shapeCount);  
 }  
  
 double radius = getPositiveInput("Enter circle radius: ");  
 shapes[shapeCount++] = new Circle(radius);  
 cout << "Circle added!" << endl;  
}  
  
void addSquare(Shape \*\*&shapes, int &shapeCount, int &capacity) {  
 if (shapeCount >= capacity) {  
 resizeShapesArray(shapes, capacity, shapeCount);  
 }  
  
 double side = getPositiveInput("Enter square side length: ");  
 shapes[shapeCount++] = new Square(side);  
 cout << "Square added!" << endl;  
}  
  
void addTriangle(Shape \*\*&shapes, int &shapeCount, int &capacity) {  
 if (shapeCount >= capacity) {  
 resizeShapesArray(shapes, capacity, shapeCount);  
 }  
  
 double base = getPositiveInput("Enter base: ");  
 double height = getPositiveInput("Enter height: ");  
 double side1 = getPositiveInput("Enter first side: ");  
 double side2 = getPositiveInput("Enter second side: ");  
  
 try {  
 shapes[shapeCount++] = new Triangle(base, height, side1, side2);  
 cout << "Triangle added!" << endl;  
 } catch (const invalid\_argument &e) {  
 cout << "Error creating triangle: " << e.what() << endl;  
 cout << "Please ensure:" << endl;  
 cout << "1. All sides are positive" << endl;  
 cout << "2. Sum of any two sides > third side" << endl;  
 cout << "3. Height matches the given sides" << endl;  
 shapeCount--;  
 }  
}  
  
void addSphere(Shape \*\*&shapes, int &shapeCount, int &capacity) {  
 if (shapeCount >= capacity) {  
 resizeShapesArray(shapes, capacity, shapeCount);  
 }  
  
 double radius = getPositiveInput("Enter sphere radius: ");  
 shapes[shapeCount++] = new Sphere(radius);  
 cout << "Sphere added!" << endl;  
}  
  
void addCube(Shape \*\*&shapes, int &shapeCount, int &capacity) {  
 if (shapeCount >= capacity) {  
 resizeShapesArray(shapes, capacity, shapeCount);  
 }  
  
 double side = getPositiveInput("Enter cube side length: ");  
 shapes[shapeCount++] = new Cube(side);  
 cout << "Cube added!" << endl;  
}  
  
void addCylinder(Shape \*\*&shapes, int &shapeCount, int &capacity) {  
 if (shapeCount >= capacity) {  
 resizeShapesArray(shapes, capacity, shapeCount);  
 }  
  
 double radius = getPositiveInput("Enter cylinder radius: ");  
 double height = getPositiveInput("Enter cylinder height: ");  
 shapes[shapeCount++] = new Cylinder(radius, height);  
 cout << "Cylinder added!" << endl;  
}  
  
void showAllShapes(span<Shape \*> shapes) {  
 if (shapes.empty()) {  
 cout << "No shapes created!" << endl;  
 return;  
 }  
  
 cout << "\n\t SHAPES (" << shapes.size() << ")" << endl;  
 for (size\_t i = 0; i < shapes.size(); ++i) {  
 cout << "\n Shape " << i + 1 << " ---" << endl;  
 shapes[i]->print();  
 cout << "Area: " << shapes[i]->area() << endl;  
 cout << "Volume: " << shapes[i]->volume() << endl;  
 cout << "Visualization:" << endl;  
 shapes[i]->draw();  
 }  
}  
  
void show2DShapes(span<Shape \*> shapes) {  
 cout << "\n\t2D SHAPES" << endl;  
 bool found = false;  
  
 for (size\_t i = 0; i < shapes.size(); ++i) {  
 auto shape2D = dynamic\_cast<const TwoDShape \*>(shapes[i]);  
 if (shape2D) {  
 found = true;  
 cout << "\n Shape " << i + 1 << " " << endl;  
 shapes[i]->print();  
 cout << "Area: " << shapes[i]->area() << endl;  
 cout << "Perimeter: " << shape2D->perimeter() << endl;  
 cout << "Visualization:" << endl;  
 shapes[i]->draw();  
 }  
 }  
  
 if (!found) {  
 cout << "No 2D shapes found!" << endl;  
 }  
}  
  
void show3DShapes(span<Shape \*> shapes) {  
 cout << "\n\t3D SHAPES" << endl;  
 bool found = false;  
  
 for (size\_t i = 0; i < shapes.size(); ++i) {  
 auto shape3D = dynamic\_cast<const ThreeDShape \*>(shapes[i]);  
 if (shape3D) {  
 found = true;  
 cout << "\n Shape " << i + 1 << " " << endl;  
 shapes[i]->print();  
 cout << "Surface Area: " << shapes[i]->area() << endl;  
 cout << "Volume: " << shapes[i]->volume() << endl;  
 cout << "Visualization:" << endl;  
 shapes[i]->draw();  
 }  
 }  
  
 if (!found) {  
 cout << "No 3D shapes!" << endl;  
 }  
}  
  
void calculateTotalArea(span<Shape \*> shapes) {  
 if (shapes.empty()) {  
 cout << "No shapes to calculate!" << endl;  
 return;  
 }  
  
 double totalArea = 0;  
 for (auto shape: shapes) {  
 totalArea += shape->area();  
 }  
 cout << "Total area of all shapes: " << totalArea << endl;  
}  
  
void deleteAllShapes(Shape \*\*&shapes, int &shapeCount, int &capacity) {  
 for (int i = 0; i < shapeCount; ++i) {  
 delete shapes[i];  
 }  
 delete[] shapes;  
  
 capacity = 10;  
 shapes = new Shape \*[capacity];  
 shapeCount = 0;  
 cout << "All shapes deleted!" << endl;  
}

Файл function.h

#ifndef LAB\_\_\_4\_\_FUNCTION\_H  
#define LAB\_\_\_4\_\_FUNCTION\_H  
  
#include "../header /Shape .h"  
#include <string>  
#include <span>  
  
void clearInputBuffer();  
  
double getPositiveInput(const std::string &prompt);  
  
void displayMenu();  
  
void resizeShapesArray(Shape \*\*&shapes, int &capacity, int shapeCount);  
  
void addCircle(Shape \*\*&shapes, int &shapeCount, int &capacity);  
  
void addSquare(Shape \*\*&shapes, int &shapeCount, int &capacity);  
  
void addTriangle(Shape \*\*&shapes, int &shapeCount, int &capacity);  
  
void addSphere(Shape \*\*&shapes, int &shapeCount, int &capacity);  
  
void addCube(Shape \*\*&shapes, int &shapeCount, int &capacity);  
  
void addCylinder(Shape \*\*&shapes, int &shapeCount, int &capacity);  
  
void showAllShapes(std::span<Shape \*> shapes);  
  
void show2DShapes(std::span<Shape \*> shapes);  
  
void show3DShapes(std::span<Shape \*> shapes);  
  
void calculateTotalArea(std::span<Shape \*> shapes);  
  
void deleteAllShapes(Shape \*\*&shapes, int &shapeCount, int &capacity);  
  
#endif

Файл Circle.cpp

#include "../header /Circle.h"  
#include "../header /Const.h"  
#include <iostream>  
  
Circle::Circle(double r) : radius(r) {}  
  
void Circle::print() const {  
 std::cout << "Circle: radius = " << radius << std::endl;  
}  
  
double Circle::area() const {  
 return constants::pi \* radius \* radius;  
}  
  
double Circle::perimeter() const {  
 return 2 \* constants::pi \* radius;  
}  
  
void Circle::draw() const {  
 std::cout << " \*\*\* " << std::endl;  
 std::cout << " \* \* " << std::endl;  
 std::cout << "\* \*" << std::endl;  
 std::cout << " \* \* " << std::endl;  
 std::cout << " \*\*\* " << std::endl;  
}

Файл Circle.h

#ifndef LAB\_\_\_4\_\_CIRCLE\_H  
#define LAB\_\_\_4\_\_CIRCLE\_H  
  
#include <iostream>  
#include "Const.h"  
#include "TwoDShape.h"  
  
class Circle : public TwoDShape {  
private:  
 double radius;  
public:  
 explicit Circle(double r);  
  
 void print() const override;  
  
 double area() const override;  
  
 double perimeter() const override;  
  
 void draw() const override;  
};  
  
#endif

Файл Const.h

#ifndef LAB\_\_\_4\_\_CONST\_H  
#define LAB\_\_\_4\_\_CONST\_H  
namespace constants {  
 inline constexpr double pi = 3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679;  
}  
#endif

Файл Cube.cpp

#include "../header /Cube.h"  
#include <iostream>  
  
Cube::Cube(double s) : side(s) {}  
  
void Cube::print() const {  
 std::cout << "Cube: side = " << side << std::endl;  
}  
  
double Cube::area() const {  
 return 6 \* side \* side;  
}  
  
double Cube::volume() const {  
 return side \* side \* side;  
}  
  
void Cube::draw() const {  
 std::cout << std::endl;  
 std::cout << " +---------+" << std::endl;  
 std::cout << " /| /|" << std::endl;  
 std::cout << " / | / |" << std::endl;  
 std::cout << " +---------+ |" << std::endl;  
 std::cout << " | | | |" << std::endl;  
 std::cout << " | +------|--+" << std::endl;  
 std::cout << " | / | /" << std::endl;  
 std::cout << " +---------+" << std::endl;  
 std::cout << " Side: " << side << std::endl;  
}

Файл Cube.h

#ifndef LAB\_\_\_4\_\_CUBE\_H  
#define LAB\_\_\_4\_\_CUBE\_H  
  
#include "ThreeDShape.h"  
  
class Cube : public ThreeDShape {  
private:  
 double side;  
  
public:  
 explicit Cube(double s);  
  
 void print() const override;  
  
 double area() const override;  
  
 double volume() const override;  
  
 void draw() const override;  
};  
  
#endif

Файл Cylinder.cpp

#include "../header /Cylinder.h"  
#include "../header /Const.h"  
#include <iostream>  
  
Cylinder::Cylinder(double r, double h) : radius(r), height(h) {}  
  
void Cylinder::print() const {  
 std::cout << "Cylinder: radius = " << radius << ", height = " << height << std::endl;  
}  
  
double Cylinder::area() const {  
 return 2 \* constants::pi \* radius \* (radius + height);  
}  
  
double Cylinder::volume() const {  
 return constants::pi \* radius \* radius \* height;  
}  
  
void Cylinder::draw() const {  
 std::cout << " \_\_\_\_\_ " << std::endl;  
 std::cout << " / \\ " << std::endl;  
 std::cout << " / \\ " << std::endl;  
 std::cout << "| |" << std::endl;  
 std::cout << "| |" << std::endl;  
 std::cout << " \\ / " << std::endl;  
 std::cout << " \\\_\_\_\_\_/ " << std::endl;  
}

Файл Cylinder.h

#ifndef LAB\_\_\_4\_\_CYLINDER\_H  
#define LAB\_\_\_4\_\_CYLINDER\_H  
  
#include "ThreeDShape.h"  
  
class Cylinder : public ThreeDShape {  
private:  
 double radius;  
 double height;  
  
public:  
 Cylinder(double r, double h);  
  
 void print() const override;  
  
 double area() const override;  
  
 double volume() const override;  
  
 void draw() const override;  
};  
  
#endif

Файл Shape.h

#ifndef LAB\_\_\_4\_\_SHAPE\_H  
#define LAB\_\_\_4\_\_SHAPE\_H  
  
#include <iostream>  
  
class Shape {  
public:  
 virtual ~Shape() = default;  
  
 virtual void print() const = 0;  
  
 virtual double area() const = 0;  
  
 virtual double volume() const = 0;  
  
 virtual void draw() const = 0;  
};  
  
#endif

Файл Sphere.cpp

#include "../header /Sphere.h"  
#include "../header /Const.h"  
#include <iostream>  
  
Sphere::Sphere(double r) : radius(r) {}  
  
void Sphere::print() const {  
 std::cout << "Sphere: radius = " << radius << std::endl;  
}  
  
double Sphere::area() const {  
 return 4 \* constants::pi \* radius \* radius;  
}  
  
double Sphere::volume() const {  
 return (4.0/3.0) \* constants::pi \* radius \* radius \* radius;  
}  
  
void Sphere::draw() const {  
 std::cout << " \*\*\*\*\* " << std::endl;  
 std::cout << " \* \* " << std::endl;  
 std::cout << " \* \* " << std::endl;  
 std::cout << " \* \* " << std::endl;  
 std::cout << " \* \* " << std::endl;  
 std::cout << " \*\*\*\*\* " << std::endl;  
}

Файл Sphere.h

#ifndef LAB\_\_\_4\_\_SPHERE\_H  
#define LAB\_\_\_4\_\_SPHERE\_H  
  
#include "ThreeDShape.h"  
  
class Sphere : public ThreeDShape {  
private:  
 double radius;  
  
public:  
 explicit Sphere(double r);  
  
 void print() const override;  
  
 double area() const override;  
  
 double volume() const override;  
  
 void draw() const override;  
};  
  
#endif

Файл Square.cpp

#include "../header /Square.h"  
#include <iostream>  
  
Square::Square(double s) : side(s) {}  
  
void Square::print() const {  
 std::cout << "Square: side = " << side << std::endl;  
}  
  
double Square::area() const {  
 return side \* side;  
}  
  
double Square::perimeter() const {  
 return 4 \* side;  
}  
  
void Square::draw() const {  
 std::cout << "┌──────┐" << std::endl;  
 std::cout << "│ │" << std::endl;  
 std::cout << "│ │" << std::endl;  
 std::cout << "└──────┘" << std::endl;  
}

Файл Square.h

#ifndef LAB\_\_\_4\_\_SQUARE\_H  
#define LAB\_\_\_4\_\_SQUARE\_H  
  
#include "TwoDShape.h"  
  
class Square : public TwoDShape {  
private:  
 double side;  
  
public:  
 explicit Square(double s);  
  
 void print() const override;  
  
 double area() const override;  
  
 double perimeter() const override;  
  
 void draw() const override;  
};  
  
#endif

Файл Triangle.cpp

#include "../header /Triangle.h"  
#include <iostream>  
#include <cmath>  
#include <stdexcept>  
  
Triangle::Triangle(double b, double h, double s1, double s2)  
 : base(b), height(h), side1(s1), side2(s2) {  
  
 if (!isValidTriangle()) {  
 throw std::invalid\_argument("Invalid triangle: height doesn't match the sides");  
 }  
}  
  
bool Triangle::isValidTriangle() const {  
 if (base <= 0 || height <= 0 || side1 <= 0 || side2 <= 0) {  
 return false;  
 }  
  
 if (side1 + side2 <= base || side1 + base <= side2 || side2 + base <= side1) {  
 return false;  
 }  
  
 double s = (side1 + side2 + base) / 2;  
 double areaHeron = sqrt(s \* (s - side1) \* (s - side2) \* (s - base));  
  
 double areaBaseHeight = (base \* height) / 2;  
  
 const double epsilon = 1e-6;  
 return std::abs(areaHeron - areaBaseHeight) < epsilon;  
}  
  
void Triangle::print() const {  
 std::cout << "Triangle: base = " << base << ", height = " << height  
 << ", sides = " << side1 << ", " << side2 << std::endl;  
}  
  
double Triangle::area() const {  
 return (base \* height) / 2;  
}  
  
double Triangle::perimeter() const {  
 return base + side1 + side2;  
}  
  
void Triangle::draw() const {  
 std::cout << " /\\ " << std::endl;  
 std::cout << " / \\ " << std::endl;  
 std::cout << " / \\ " << std::endl;  
 std::cout << " /\_\_\_\_\_\_\\ " << std::endl;  
}

Файл Triangle.h

#ifndef LAB\_\_\_4\_\_TRIANGLE\_H  
#define LAB\_\_\_4\_\_TRIANGLE\_H  
  
#include "TwoDShape.h"  
  
class Triangle : public TwoDShape {  
private:  
 double base;  
 double height;  
 double side1;  
 double side2;  
  
public:  
 Triangle(double b, double h, double s1, double s2);  
  
 void print() const override;  
  
 double area() const override;  
  
 double perimeter() const override;  
  
 void draw() const override;  
  
private:  
 bool isValidTriangle() const;  
};  
  
#endif

Файл TwoDShape.h

#ifndef LAB\_\_\_4\_\_TWODSHAPE\_H  
#define LAB\_\_\_4\_\_TWODSHAPE\_H  
  
#include "Shape .h"  
  
class TwoDShape : public Shape {  
public:  
 ~TwoDShape() override = default;  
  
 virtual double perimeter() const = 0;  
  
 double volume() const override { return 0; }  
};  
  
#endif

Файл ThreeDShape.h

#ifndef LAB\_\_\_4\_\_TWODSHAPE\_H  
#define LAB\_\_\_4\_\_TWODSHAPE\_H  
  
#include "Shape .h"  
  
class TwoDShape : public Shape {  
public:  
 ~TwoDShape() override = default;  
  
 virtual double perimeter() const = 0;  
  
 double volume() const override { return 0; }  
};  
  
#endif

**4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

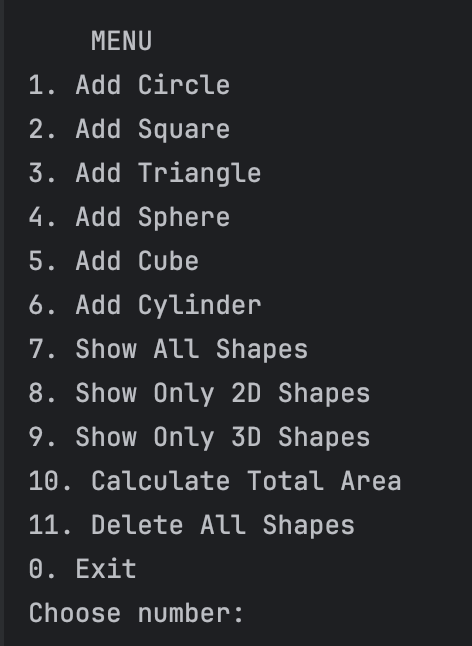


Рисунок 4.1 – меню, с которым взаимодействует пользователь

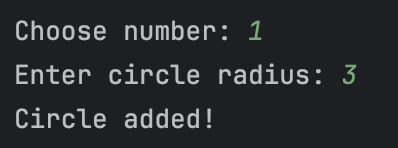


Рисунок 4.2 – добавление круга

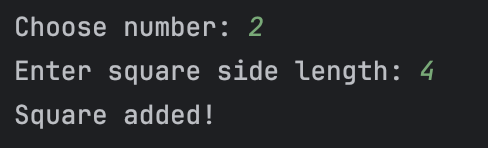


Рисунок 4.3 – добавление квадрата

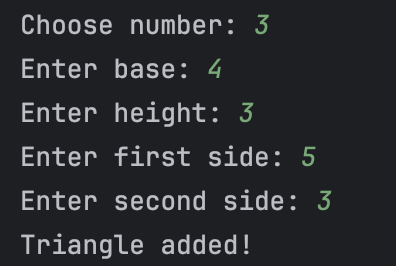


Рисунок 4.4 – добавление треугольника

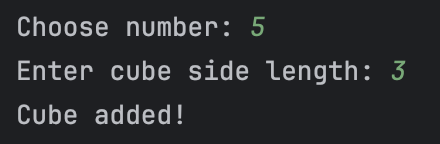
****

Рисунок 4.5 – добавление куба

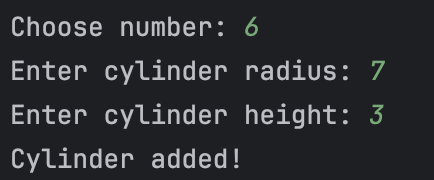
****

Рисунок 4.6 – добавление цилиндра

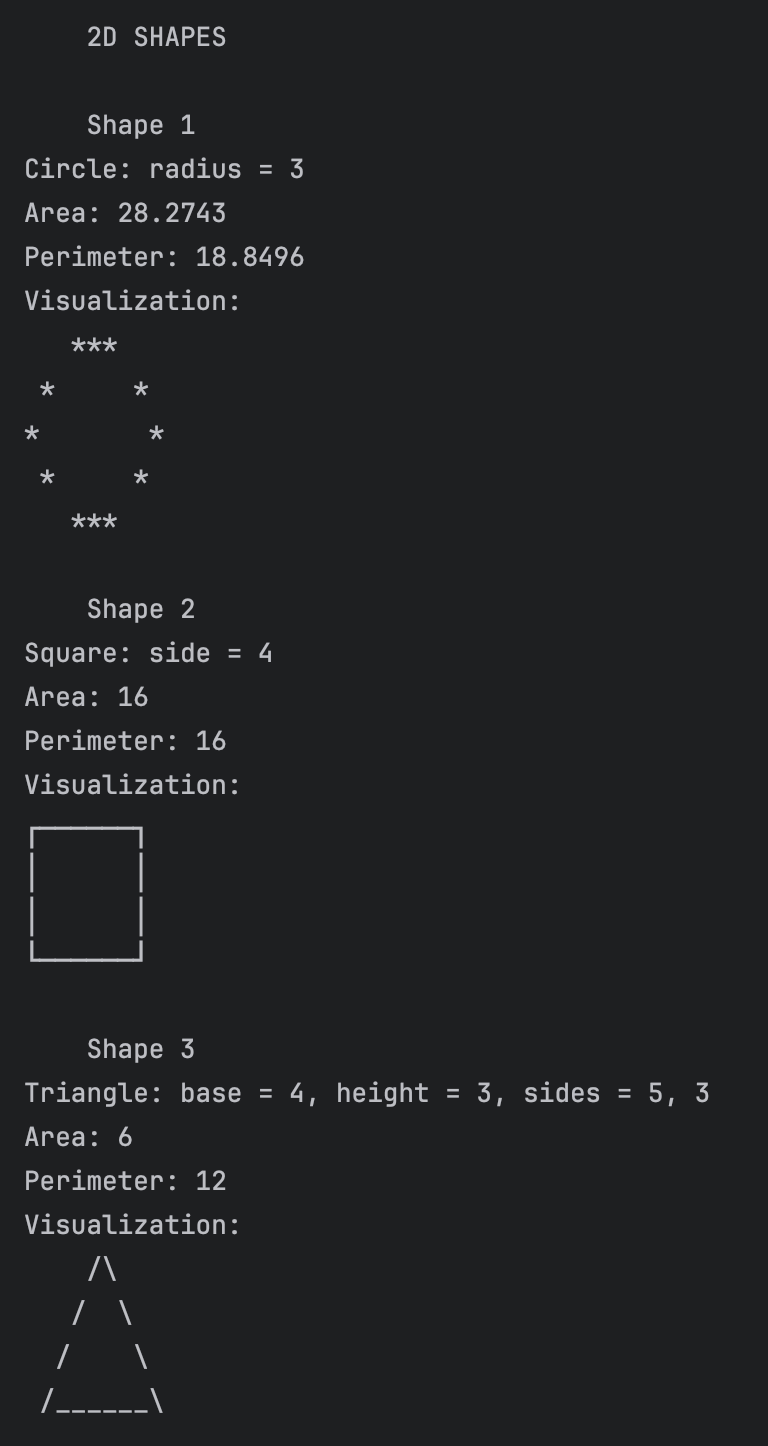
****

Рисунок 4.7 – вывод 2Д фигур



Рисунок 4.8 – вывод 3Д фигур