Министерство науки и высшего образования РФ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет

им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

**СТРУКТУРЫ ХРАНЕНИЯ ГЕОМЕТРИЧЕСКИХ ФИГУР**

Выполнила: Пикова Анна Дмитриевна, студентка группы 381903-3

Проверил: программист 1 категории МОСТ Лебедев И. Г.

Нижний Новгород

2020

**Содержание**

[Введение 3](#_Toc40486339)

[1. Постановка задачи 4](#_Toc40486340)

[2. Руководство пользователя 5](#_Toc40486341)

[3. Руководство программиста 6](#_Toc40486342)

[3.1. Структура программы 6](#_Toc40486343)

[3.2. Описание алгоритма 6](#_Toc40486344)

[3.2.1. Формула длины отрезка 6](#_Toc40486345)

[3.2.2. Формулы нахождения площадей фигур 6](#_Toc40486346)

[3.2.3. Формула нахождения объёма цилиндра 6](#_Toc40486347)

[3.3. Описание структур данных и функций 6](#_Toc40486348)

[3.3.1. Класс TBaseFigure 6](#_Toc40486349)

[3.3.2. Класс TPoint 7](#_Toc40486350)

[3.3.3. Класс TLine 8](#_Toc40486351)

[3.3.4. Класс TPlaneFigure 9](#_Toc40486352)

[3.3.5. Класс TCircle 10](#_Toc40486353)

[3.3.6. Класс TSquare 11](#_Toc40486354)

[3.3.7. Класс TVolumeFigure 12](#_Toc40486355)

[3.3.8. Класс TCylinder 12](#_Toc40486356)

[3.3.9. Класс TContainer 13](#_Toc40486357)

[3.3.10. Основная программа 15](#_Toc40486358)

[Заключение 16](#_Toc40486359)

[Литература 17](#_Toc40486360)

[Приложение 18](#_Toc40486361)

# Введение

Иногда возникает необходимость определить класс, представляющий абстрактные понятия, для которых нельзя создать объекты. Такие классы приобретают смысл только как базовые классы в некоторых производных классах. Причиной является то, что невозможно дать осмысленное определений виртуальных функций класса.

Примером такого класса является класс геометрических фигур. В реальности есть конкретные фигуры: треугольник, квадрат, круг, прямоугольник и др. Однако абстрактной фигуры самой по себе не существует. В то же время может потребоваться определить для всех фигур какой-то общий класс, который будет содержать общую функциональность. И для описания подобных сущностей используются абстрактные классы.

Таким образом, для этой цели и была создана программа, с помощью которой можно продемонстрировать работу иерархии классов геометрических фигур.

# Постановка задачи

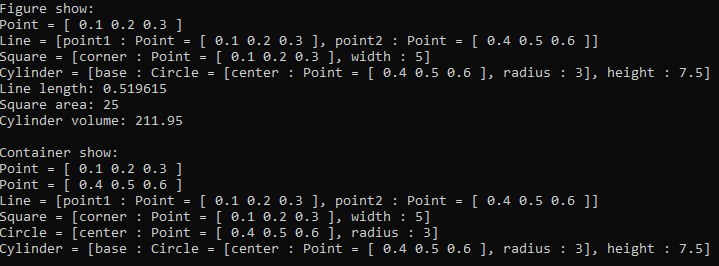
**Задача:** написать структуру данных для работы с геометрическими объектами в n-мерном пространстве с использованием абстракции, инкапсуляции, наследования, полиморфизма, перегрузки операций и шаблонов, продемонстрировать работу на примере.

**Входные данные:** параметры геометрических фигур.

**Выходные данные:** параметры геометрических фигур.

# Руководство пользователя

1. С использованием прилагаемой библиотеки классов создать объекты фигур, заполнить параметры с помощью конструкторов, при необходимости добавить в контейнер и вывести или использовать вычислительные методы.



* Пример использования программы.

# Руководство программиста

## Структура программы

* **TBaseFigure.h** – содержит объявление класса TBaseFigure (базовая фигура);
* **TPoint.h** – содержит объявление класса TPoint (точка);
* **TLine.h** – содержит объявление класса TLine (отрезок);
* **TPlaneFigure.h** – содержит объявление класса TPnaneFigure (плоская фигура);
* **TCircle.h** – содержит объявление класса TCircle (круг);
* **TSquare.h** – содержит объявление класса TSquare (квадрат);
* **TVolumeFigure.h** – содержит объявление класса TVolumeFigure (объёмная фигура);
* **TCylinder.h** – содержит объявление класса TCylinder (цилиндр);
* **TContainer.h** – содержит объявление класса TContainer («контейнер»);
* **Lab7\_GeometricShapes.cpp** – содержит пример использования.

## Описание алгоритма

Опишем основные математические формулы, используемые в программе.

### Формула длины отрезка

* – эта формула длины отрезка позволяет рассчитать расстояние между двумя произвольными точками на плоскости, при условии, что известны координаты этих точек.

### Формулы нахождения площадей фигур

* – площадь квадрата, где a – сторона квадрата.
* – площадь круга, где r – радиус окружности.

### Формула нахождения объёма цилиндра

* , где Sосн – площадь основания, h – высота цилиндра.

## Описание структур данных и функций

### Класс TBaseFigure

Объявление базового класса фигуры выглядит следующим образом:

template<typename T>

class TBaseFigure

{

protected:

int dim;

public:

TBaseFigure(int dim\_);

virtual void show() = 0;

};

#### Поля класса

int dim;

**Назначение:** хранит размерность.

#### Методы класса

TBaseFigure(int dim\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** размерность.

**Выходные данные:** отсутствуют.

virtual void show() = 0;

**Назначение:** вывод данных о фигуре.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

### Класс TPoint

Объявление класса «Точка» выглядит следующим образом:

template<typename T>

class TPoint : public TBaseFigure<T>

{

protected:

using TBaseFigure<T>::dim;

T\* coords;

public:

TPoint(int dim\_);

TPoint(int dim\_, T\* coords\_);

TPoint(const TPoint& other);

~TPoint();

T& operator[](int index);

T operator[](int index) const;

void show();

int getDim() { return dim; }

};

#### Поля класса

T\* coords;

**Назначение:** хранит массив координат.

#### Методы класса

TPoint(int dim\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** размерность.

**Выходные данные:** отсутствуют.

TPoint(int dim\_, T\* coords\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** размерность, массив координат.

**Выходные данные:** отсутствуют.

TPoint(const TPoint& other);

**Назначение:** конструктор копирования.

**Входные параметры:** other – копируемая точка.

**Выходные данные:** отсутствуют.

~TPoint();

**Назначение:** деструктор.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

T& operator[](int index);

T operator[](int index) const;

**Назначение:** перегрузка оператора индексации [].

**Входные параметры:** значение индекса.

**Выходные данные:** указатель на начало строки.

void show();

**Назначение:** вывод данных о точке.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

int getDim() { return dim; }

**Назначение:** возвращает размерность.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

### Класс TLine

Объявление класса «Отрезок» выглядит следующим образом:

template<typename T>

class TLine : public TBaseFigure<T>

{

protected:

using TBaseFigure<T>::dim;

public:

TPoint<T> point1, point2;

TLine(TPoint<T> point1\_, TPoint<T> point2\_);

void show();

T length();

int getDim() { return dim; }

};

#### Методы класса

TLine(TPoint<T> point1\_, TPoint<T> point2\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** две точки.

**Выходные данные:** отсутствуют.

void show();

**Назначение:** вывод данных об отрезке.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

T length();

**Назначение:** определяет длину отрезка.

**Входные параметры:** отсутствуют.

**Выходные данные:** длина отрезка.

int getDim() { return dim; }

**Назначение:** возвращает размерность.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

### Класс TPlaneFigure

Объявление класса плоских фигур выглядит следующим образом:

template<typename T>

class TPlaneFigure : public TBaseFigure<T>

{

protected:

using TBaseFigure<T>::dim;

public:

TPlaneFigure(int dim\_);

virtual T area() = 0;

};

#### Методы класса

TPlaneFigure(int dim\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** размерность.

**Выходные данные:** отсутствуют.

virtual T area() = 0;

**Назначение:** вычисление площади.

**Входные параметры:** отсутствуют.

**Выходные данные:** площадь.

### Класс TCircle

Объявление класса «Круг» выглядит следующим образом:

template<typename T>

class TCircle : public TPlaneFigure<T>

{

protected:

using TPlaneFigure<T>::dim;

public:

TPoint<T> center;

T radius;

TCircle(TPoint<T> center\_, T radius\_);

void show();

T area();

int getDim() { return dim; }

};

#### Поля класса

TPoint<T> center;

**Назначение:** точка центра.

T radius;

**Назначение:** радиус.

#### Методы класса

TCircle(TPoint<T> center\_, T radius\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** точка центра, радиус.

**Выходные данные:** отсутствуют.

void show();

**Назначение:** вывод данных о круге.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

T area();

**Назначение:** определяет площадь круга.

**Входные параметры:** отсутствуют.

**Выходные данные:** площадь круга.

int getDim() { return dim; }

**Назначение:** возвращает размерность.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

### Класс TSquare

Объявление класса «Квадрат» выглядит следующим образом:

template<typename T>

class TSquare : public TPlaneFigure<T>

{

protected:

using TPlaneFigure<T>::dim;

public:

TPoint<T> corner;

T width;

TSquare(TPoint<T> corner\_, T width\_);

void show();

T area();

int getDim() { return dim; }

};

#### Поля класса

TPoint<T> corner;

**Назначение:** точка угла.

T width;

**Назначение:** длина стороны.

#### Методы класса

TSquare(TPoint<T> corner\_, T width\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** точка угла, длина стороны.

**Выходные данные:** отсутствуют.

void show();

**Назначение:** вывод данных о квадрате.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

T area();

**Назначение:** определяет площадь квадрата.

**Входные параметры:** отсутствуют.

**Выходные данные:** площадь квадрата.

int getDim() { return dim; }

**Назначение:** возвращает размерность.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

### Класс TVolumeFigure

Объявление класса объёмных фигур выглядит следующим образом:

template<typename T>

class TVolumeFigure : public TBaseFigure<T>

{

protected:

using TBaseFigure<T>::dim;

public:

TVolumeFigure(int dim\_);

virtual T volume() = 0;

};

#### Методы класса

TVolumeFigure(int dim\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** размерность.

**Выходные данные:** отсутствуют.

virtual T volume() = 0;

**Назначение:** вычисление объёма.

**Входные параметры:** отсутствуют.

**Выходные данные:** объём.

### Класс TCylinder

Объявление класса «Цилиндр» выглядит следующим образом:

template<typename T>

class TCylinder : public TVolumeFigure<T>

{

protected:

using TVolumeFigure<T>::dim;

public:

TCircle<T> base;

T height;

TCylinder(TCircle<T> base\_, T height\_);

void show();

T volume();

int getDim() { return dim; }

};

#### Поля класса

TCircle<T> base;

**Назначение:** основание.

T height;

**Назначение:** высота.

#### Методы класса

TCylinder(TCircle<T> base\_, T height\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** основание, высота.

**Выходные данные:** отсутствуют.

void show();

**Назначение:** вывод данных о цилиндре.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

T volume();

**Назначение:** определяет объём цилиндра

**Входные параметры:** отсутствуют.

**Выходные данные:** объём цилиндра.

int getDim() { return dim; }

**Назначение:** возвращает размерность.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

### Класс TContainer

Объявление класса «Контейнер» выглядит следующим образом:

template<typename T>

class TContainer

{

TBaseFigure<T>\*\* elements;

int maxSize;

int currentSize;

public:

TContainer(int maxSize\_);

TContainer(const TContainer& other);

~TContainer();

int getCurrentSize();

int getMaxSize();

template<typename TT> void add(const TT& elem);

void remove(int index);

void show();

TBaseFigure<T>& operator[](int index);

const TBaseFigure<T>& operator[](int index) const;

};

#### Поля класса

TBaseFigure<T>\*\* elements;

**Назначение:** массив указателей на объекты фигур.

int maxSize;

**Назначение:** максимальный размер.

int currentSize;

**Назначение:** текущий размер.

#### Методы класса

TContainer(int maxSize\_);

**Назначение:** конструктор инициализации.

**Входные параметры:** максимальный размер.

**Выходные данные:** отсутствуют.

TContainer(const TContainer& other);

**Назначение:** конструктор копирования.

**Входные параметры:** other – копируемый контейнер.

**Выходные данные:** отсутствуют.

~TContainer();

**Назначение:** деструктор.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

int getCurrentSize();

**Назначение:** возвращает текущий размер.

**Входные параметры:** отсутствуют.

**Выходные данные:** текущий размер.

int getMaxSize();

**Назначение:** возвращает максимальный размер.

**Входные параметры:** отсутствуют.

**Выходные данные:** максимальный размер.

template<typename TT> void add(const TT& elem);

**Назначение:** добавляет объект в контейнер.

**Входные параметры:** объект, который нужно добавить.

**Выходные данные:** отсутствуют.

void remove(int index);

**Назначение:** удаляет существующий объект.

**Входные параметры:** индекс объекта.

**Выходные данные:** отсутствуют.

void show();

**Назначение:** выводит данные обо всех фигурах, добавленных в контейнер.

**Входные параметры:** отсутствуют.

**Выходные данные:** отсутствуют.

TBaseFigure<T>& operator[](int index);

const TBaseFigure<T>& operator[](int index) const;

**Назначение:** перегрузка оператора индексации [].

**Входные параметры:** значение индекса.

**Выходные данные:** указатель на начало строки.

### Основная программа

#### Основная функция

int main();

**Назначение:** основная функция (точка входа).

# Заключение

В ходе выполнения лабораторной работы была разработана иерархия классов «Геометрические фигуры».

# Литература

1. Бьерн Страуструп. Язык программирования C++, 2-е изд: Пер. с англ. — М.: Издательский дом «Бином», 2017. — 1136 с.
2. Герберт Шилдт. C++ для начинающих. Шаг за шагом, 2-е изд: Пер. с англ – М.: Издательский дом «ЭКОМ Паблишерз», 2013. – 640 с.
3. Липпман Стенли Б., Лажойе Жози, Му Барбара Э. Язык программирования C++, базовый курс, 5-е изд: Пер. с англ. – М.: Издательский дом «Диалектика», 2018. – 1118 с.

# Приложение

**Приложение 1.** TPlaneFigure.h

#pragma once

#include "TBaseFigure.h"

template<typename T>

class TPlaneFigure : public TBaseFigure<T>

{

protected:

using TBaseFigure<T>::dim;

public:

TPlaneFigure(int dim\_);

virtual T area() = 0;

};

template<typename T>

TPlaneFigure<T>::TPlaneFigure(int dim\_)

: TBaseFigure<T>(dim\_)

{

}

**Приложение 2.** TPoint.h

#pragma once

#include "TBaseFigure.h"

template<typename T>

class TPoint : public TBaseFigure<T>

{

protected:

using TBaseFigure<T>::dim;

T\* coords;

public:

TPoint(int dim\_);

TPoint(int dim\_, T\* coords\_);

TPoint(const TPoint& other);

~TPoint();

T& operator[](int index);

T operator[](int index) const;

void show();

int getDim() { return dim; }

};

template<typename T>

TPoint<T>::TPoint(int dim\_)

: TBaseFigure<T>(dim\_)

{

coords = new T[dim\_];

for (int i = 0; i < dim; i++)

coords[i] = 0;

}

template<typename T>

TPoint<T>::TPoint(int dim\_, T\* coords\_)

: TBaseFigure<T>(dim\_)

{

coords = new T[dim\_];

for (int i = 0; i < dim; i++)

coords[i] = coords\_[i];

}

template<typename T>

TPoint<T>::TPoint(const TPoint& other)

: TPoint<T>(other.dim, other.coords)

{

}

template<typename T>

TPoint<T>::~TPoint()

{

delete[] coords;

}

template<typename T>

T& TPoint<T>::operator[](int index)

{

return coords[index];

}

template<typename T>

T TPoint<T>::operator[](int index) const

{

return coords[index];

}

template<typename T>

void TPoint<T>::show()

{

std::cout << "Point = [ ";

for (int i = 0; i < dim; i++)

std::cout << coords[i] << ' ';

std::cout << ']';

}

**Приложение 3.** TLine.h

#pragma once

#include "TPoint.h"

#include <cmath>

template<typename T>

class TLine : public TBaseFigure<T>

{

protected:

using TBaseFigure<T>::dim;

public:

TPoint<T> point1, point2;

TLine(TPoint<T> point1\_, TPoint<T> point2\_);

void show();

T length();

int getDim() { return dim; }

};

template<typename T>

TLine<T>::TLine(TPoint<T> point1\_, TPoint<T> point2\_)

: TBaseFigure<T>(point1\_.getDim()), point1(point1\_), point2(point2\_)

{

}

template<typename T>

void TLine<T>::show()

{

std::cout << "Line = [point1 : ", point1.show();

std::cout << ", point2 : ", point2.show();

std::cout << "]";

}

template<typename T>

T TLine<T>::length()

{

T sum = 0;

for (int i = 0; i < dim; i++)

sum += (point1[i] - point2[i]) \* (point1[i] - point2[i]);

return sqrt(sum);

}

**Приложение 4.** TPlaneFigure.h

#pragma once

#include "TBaseFigure.h"

template<typename T>

class TPlaneFigure : public TBaseFigure<T>

{

protected:

using TBaseFigure<T>::dim;

public:

TPlaneFigure(int dim\_);

virtual T area() = 0;

};

template<typename T>

TPlaneFigure<T>::TPlaneFigure(int dim\_)

: TBaseFigure<T>(dim\_)

{

}

**Приложение 5.** TCircle.h

#pragma once

#include "TPoint.h"

#include "TPlaneFigure.h"

#include <cmath>

template<typename T>

class TCircle : public TPlaneFigure<T>

{

protected:

using TPlaneFigure<T>::dim;

public:

TPoint<T> center;

T radius;

TCircle(TPoint<T> center\_, T radius\_);

void show();

T area();

int getDim() { return dim; }

};

template<typename T>

TCircle<T>::TCircle(TPoint<T> center\_, T radius\_)

: TPlaneFigure<T>(center\_.getDim()), center(center\_)

{

if (radius\_ <= 0)

throw "radius must be positive";

radius = radius\_;

}

template<typename T>

void TCircle<T>::show()

{

std::cout << "Circle = [center : ", center.show();

std::cout << ", radius : " << radius << "]";

}

template<typename T>

T TCircle<T>::area()

{

return 3.14 \* radius \* radius;

}

**Приложение 6.** TSquare.h

#pragma once

#include "TPoint.h"

#include "TPlaneFigure.h"

template<typename T>

class TSquare : public TPlaneFigure<T>

{

protected:

using TPlaneFigure<T>::dim;

public:

TPoint<T> corner;

T width;

TSquare(TPoint<T> corner\_, T width\_);

void show();

T area();

int getDim() { return dim; }

};

template<typename T>

TSquare<T>::TSquare(TPoint<T> corner\_, T width\_)

: TPlaneFigure<T>(corner\_.getDim()), corner(corner\_)

{

width = width\_;

}

template<typename T>

void TSquare<T>::show()

{

std::cout << "Square = [corner : ", corner.show();

std::cout << ", width : " << width << "]";

}

template<typename T>

T TSquare<T>::area()

{

return width \* width;

}

**Приложение 7.** TVolumeFigure.h

#pragma once

#include "TBaseFigure.h"

template<typename T>

class TVolumeFigure : public TBaseFigure<T>

{

protected:

using TBaseFigure<T>::dim;

public:

TVolumeFigure(int dim\_);

virtual T volume() = 0;

};

template<typename T>

TVolumeFigure<T>::TVolumeFigure(int dim\_)

: TBaseFigure<T>(dim\_)

{

}

**Приложение 8.** TCylinder.h

#pragma once

#include "TVolumeFigure.h"

#include "TCircle.h"

template<typename T>

class TCylinder : public TVolumeFigure<T>

{

protected:

using TVolumeFigure<T>::dim;

public:

TCircle<T> base;

T height;

TCylinder(TCircle<T> base\_, T height\_);

void show();

T volume();

int getDim() { return dim; }

};

template<typename T>

TCylinder<T>::TCylinder(TCircle<T> base\_, T height\_)

: TVolumeFigure<T>(base\_.getDim()), base(base\_)

{

height = height\_;

}

template<typename T>

void TCylinder<T>::show()

{

std::cout << "Cylinder = [base : ", base.show();

std::cout << ", height : " << height << "]";

}

template<typename T>

T TCylinder<T>::volume()

{

return base.area() \* height;

}

**Приложение 9.** TContainer.h

#pragma once

#include <iostream>

#include "TBaseFigure.h"

template<typename T>

class TContainer

{

TBaseFigure<T>\*\* elements;

int maxSize;

int currentSize;

public:

TContainer(int maxSize\_);

TContainer(const TContainer& other);

~TContainer();

int getCurrentSize();

int getMaxSize();

template<typename TT> void add(const TT& elem);

void remove(int index);

void show();

TBaseFigure<T>& operator[](int index);

const TBaseFigure<T>& operator[](int index) const;

};

template<typename T>

TContainer<T>::TContainer(int maxSize\_)

{

if (maxSize\_ <= 0)

throw "size must be positive";

maxSize = maxSize\_;

currentSize = 0;

elements = new TBaseFigure<T>\*[maxSize];

}

template<typename T>

TContainer<T>::TContainer(const TContainer& other)

{

maxSize = other.maxSize;

currentSize = other.currentSize;

elements = new TBaseFigure<T>\*[maxSize];

for (int i = 0; i < currentSize; i++)

elements[i] = new TBaseFigure<T>(\*other.elements[i]);

}

template<typename T>

TContainer<T>::~TContainer()

{

for (int i = 0; i < currentSize; i++)

delete elements[i];

delete[] elements;

}

template<typename T>

int TContainer<T>::getCurrentSize()

{

return currentSize;

}

template<typename T>

inline int TContainer<T>::getMaxSize()

{

return maxSize;

}

template<typename T> template<typename TT>

void TContainer<T>::add(const TT& elem)

{

if (currentSize == maxSize)

throw "container is full";

elements[currentSize] = new TT(elem);

currentSize += 1;

}

template<typename T>

void TContainer<T>::remove(int index)

{

if ((index < 0) || (index >= currentSize))

throw "element not exists";

delete elements[index];

elements[index] = elements[currentSize - 1];

currentSize -= 1;

}

template<typename T>

void TContainer<T>::show()

{

for (int i = 0; i < currentSize; i++)

{

elements[i]->show();

std::cout << '\n';

}

}

template<typename T>

TBaseFigure<T>& TContainer<T>::operator[](int index)

{

return \*elements[index];

}

template<typename T>

const TBaseFigure<T>& TContainer<T>::operator[](int index) const

{

return \*elements[index];

}

**Приложение 10.** Lab7\_GeometricShapes.cpp

#include <iostream>

#include "TContainer.h"

#include "TPoint.h"

#include "TLine.h"

#include "TCircle.h"

#include "TSquare.h"

#include "TCylinder.h"

int main()

{

TContainer<double> container(6);

double point1coords[] = { 0.1, 0.2, 0.3 };

double point2coords[] = { 0.4, 0.5, 0.6 };

TPoint<double> point1(3, point1coords);

TPoint<double> point2(3, point2coords);

TLine<double> line(point1, point2);

TSquare<double> square(point1, 5.0);

TCircle<double> circle(point2, 3.0);

TCylinder<double> cylinder(circle, 7.5);

container.add(point1);

container.add(point2);

container.add(line);

container.add(square);

container.add(circle);

container.add(cylinder);

std::cout << "Figure show:\n";

point1.show();

std::cout << '\n';

line.show();

std::cout << '\n';

square.show();

std::cout << '\n';

cylinder.show();

std::cout << '\n';

std::cout << "Line length: " << line.length() << '\n';

std::cout << "Square area: " << square.area() << '\n';

std::cout << "Cylinder volume: " << cylinder.volume() << '\n';

std::cout << "\nContainer show:\n";

container.show();

}