Министерство образования Республики Беларусь

Оршанский колледж ВГУ имени П.М. Машерова

#### Отчет №4

По теме «**Задания по интерфейсам**»

по практике по программированию

###### Выполнил учащийся Волога П. В.

3ПОИТ21 2023 г.

Проверил Трибис А. В.

2023 г.

Орша, 2023

**Цель:** .

**Задания.**

1. Создайте интерфейс "Фигура" с методами для вычисления площади и периметра. Затем реализуйте этот интерфейс в классах геометрических фигур (круг, прямоугольник, треугольник).
2. Разработайте приложение для учета продуктов в магазине с использованием интерфейсов. Создайте интерфейс "Товар" с методами для определения стоимости и остатка товара на складе. Реализуйте этот интерфейс в классах различных товаров.
3. Создайте систему учета студентов в университете с помощью интерфейсов. Создайте интерфейс "Студент" с методами для определения среднего балла и получения информации о курсе. Реализуйте этот интерфейс в классах студентов разных курсов.
4. Реализуйте приложение для работы с библиотекой книг с использованием интерфейсов. Создайте интерфейс "Книга" с методами для проверки доступности и выдачи книги. Реализуйте этот интерфейс в классах различных книг.
5. Создайте приложение для рисования на холсте с использованием интерфейсов. Создайте интерфейс "Рисунок" с методами для рисования линий, кругов и прямоугольников. Реализуйте этот интерфейс в классе для работы с холстом.

**Ход выполнения работы.**

**1.** В листинге 1 представлен код по решению задачи создания класса Person.

**Листинг1.** Код по решению задачи создания класса Person.

using System;

public class Person // Определение класса Person

{

public string Name { get; set; } // Определение свойства Name для хранения имени человека

public double Age { get; set; } // Определение свойства Age для хранения возраста человека

public string Address { get; set; } // Определение свойства Address для хранения адреса человека

public Person() // Конструктор класса Person

{

Console.Write("Введите имя: ");

Name = Console.ReadLine(); // Чтение введенного имени из консоли и сохранение его в свойство Name

Console.Write("Введите возраст: ");

if (!double.TryParse(Console.ReadLine(), out double age)) // Преобразование введенного возраста в число

{

Console.WriteLine("Некорректный ввод возраста. Пожалуйста, введите число."); // Вывод сообщения об ошибке, если введенное значение не является числом

throw new ArgumentException("Некорректный ввод возраста"); // Генерация исключения с сообщением об ошибке

}

if (age < 0) // Проверка, что возраст не меньше нуля

{

Console.WriteLine("Такого возраста быть не может. Возраст не может быть меньше 0."); // Вывод сообщения об ошибке, если возраст меньше нуля

throw new ArgumentException("Возраст не может быть меньше 0"); // Генерация исключения с сообщением об ошибке

}

else if (age > 122) // Проверка, что возраст не больше 122 лет

{

Console.WriteLine("Люди столько не живут. Возраст не может превышать 122 года."); // Вывод сообщения об ошибке, если возраст больше 122 лет

throw new ArgumentException("Возраст не может превышать 122 года"); // Генерация исключения с сообщением об ошибке

}

Age = age; // Сохранение введенного возраста в свойство Age

Console.Write("Введите адрес: ");

Address = Console.ReadLine(); // Чтение введенного адреса из консоли и сохранение его в свойство Address

}

public string GetInfo() // Метод для получения информации о человеке

{

return $"Имя: {Name}, Возраст: {Age}, Адрес: {Address}"; // Форматирование и возврат строки с информацией о человеке

}

}

public class vivod\_Person // Определение класса vivod\_Person для вывода информации о человеке

{

static void Main(string[] args) // Главный метод программы

{

vivod\_Person VIvod\_Person = new vivod\_Person(); // Создание нового объекта класса vivod\_Person

VIvod\_Person.Vivod\_Person(); // Вызов метода Vivod\_Person объекта VIvod\_Person для вывода информации о человеке

}

public void Vivod\_Person() // Метод для вывода информации о человеке

{

try // Блок try-catch для обработки исключений, которые могут быть сгенерированы при создании объектов класса Person

{

Person person1 = new Person(); // Создание нового объекта класса Person и запрос информации о человеке

Console.WriteLine(person1.GetInfo()); // Вывод информации о человеке, используя метод GetInfo объекта person1

}

catch (ArgumentException) // Обработка исключений типа ArgumentException, которые могут быть сгенерированы при создании объекта класса Person

{

}

}

}

На рисунке 1 представлен результат работы программы, а также работа всех исключений.

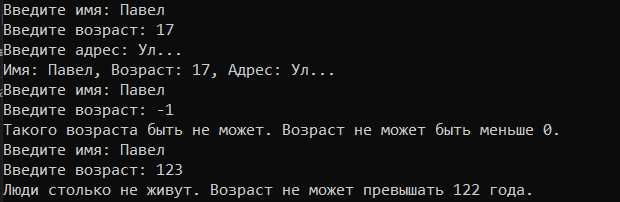


Рисунок 1 – Результат работы программы

**2.** В листинге 2 представлен код по решению задачи наследования и полиморфизма.

**Листинг2.** Код по решению задачи наследования и полиморфизма.

using System;

public abstract class Shape // Абстрактный класс Shape

{

public abstract double Area(); // Абстрактный метод для вычисления площади

public abstract double Perimeter(); // Абстрактный метод для вычисления периметра

}

public class Circle : Shape // Класс Circle, наследующий от Shape

{

public double radius; // Приватное поле для радиуса

public Circle() // Конструктор класса Circle

{

Console.Write("Введите радиус круга: "); // Запрос на ввод радиуса

if (!double.TryParse(Console.ReadLine(), out radius)) // Преобразование введенного значения в double

{

Console.WriteLine("Некорректный ввод радиуса. Пожалуйста, введите число."); // Сообщение об ошибке

throw new ArgumentException("Некорректный ввод радиуса"); // Генерация исключения

}

}

public override double Area() // Переопределение метода Area для круга

{

return Math.PI \* Math.Pow(radius, 2); // Возвращаем площадь круга

}

public override double Perimeter() // Переопределение метода Perimeter для круга

{

return 2 \* Math.PI \* radius; // Возвращаем периметр круга

}

}

public class Rectangle : Shape // Класс Rectangle, наследующий от Shape

{

public double length; // Приватное поле для длины

public double width; // Приватное поле для ширины

public Rectangle() // Конструктор класса Rectangle

{

Console.Write("Введите длину прямоугольника: "); // Запрос на ввод длины

if (!double.TryParse(Console.ReadLine(), out length)) // Преобразовать введенного значения в double

{

Console.WriteLine("Некорректный ввод длины. Пожалуйста, введите число."); // Сообщение об ошибке

throw new ArgumentException("Некорректный ввод длины"); // Генерация исключения

}

Console.Write("Введите ширину прямоугольника: "); // Запрос на ввод ширины

if (!double.TryParse(Console.ReadLine(), out width)) // Попытка преобразовать введенное значение в double

{

Console.WriteLine("Некорректный ввод ширины. Пожалуйста, введите число."); // Сообщение об ошибке

throw new ArgumentException("Некорректный ввод ширины"); // Генерация исключения

}

}

public override double Area() // Переопределение метода Area для прямоугольника

{

return length \* width; // Возвращаем площадь прямоугольника

}

public override double Perimeter() // Переопределение метода Perimeter для прямоугольника

{

return 2 \* (length + width); // Возвращаем периметр прямоугольника

}

}

public class Vivod\_Figur // Класс Vivod\_Figur для вывода информации о фигурах

{

static void Main(string[] args) // Главный метод программы

{

Vivod\_Figur vivod\_Figur = new Vivod\_Figur(); // Создание объекта класса Vivod\_Figur

vivod\_Figur.vivod\_figur(); // Вызов метода vivod\_figur

}

public void vivod\_figur() // Метод vivod\_figur для вывода информации о фигурах

{

try // Блок try для обработки исключений

{

Circle circle = new Circle(); // Создание объекта класса Circle

Console.WriteLine($"Площадь круга: {circle.Area()}, Периметр круга: {circle.Perimeter()}"); // Вывод площади и периметра круга

Rectangle rectangle = new Rectangle(); // Создание объекта класса Rectangle

Console.WriteLine($"Площадь прямоугольника: {rectangle.Area()}, Периметр прямоугольника: {rectangle.Perimeter()}"); // Вывод площади и периметра прямоугольника

}

catch (ArgumentException) // Обработка исключения ArgumentException

{

}

}

}

На рисунке 2 представлен результат работы программы, а также работа всех исключений.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Результат работы программы

**3.** В листинге 3 представлен код по решению задачи создания композиции.

**Листинг3.** Код по решению задачи создания композиции.

using System; // Используем пространство имен System

using System.Text.RegularExpressions; // Используем пространство имен System.Text.RegularExpressions для работы с регулярными выражениями

public class Author // Класс Author

{

public string Name { get; set; } // Свойство Name

public int BirthYear { get; set; } // Свойство BirthYear

public Author() // Конструктор класса Author

{

Console.Write("Введите имя автора: "); // Запрос на ввод имени автора

Name = Console.ReadLine(); // Чтение введенного имени автора

if (!Regex.IsMatch(Name, @"^[a-zA-Zа-яА-Я\s]+$")) // Проверка имени автора на соответствие регулярному выражению

{

Console.WriteLine("Некорректный ввод имени. Имя может содержать только буквы и пробелы."); // Сообщение об ошибке

throw new ArgumentException("Некорректный ввод имени"); // Генерация исключения

}

Console.Write("Введите год рождения автора: "); // Запрос на ввод года рождения автора

if (!int.TryParse(Console.ReadLine(), out int birthYear) || birthYear < 1 || birthYear > 2023) // Проверка года рождения на соответствие условиям

{

Console.WriteLine("Некорректный ввод года рождения. Пожалуйста, введите число от 1 до 2023."); // Сообщение об ошибке

throw new ArgumentException("Некорректный ввод года рождения"); // Генерация исключения

}

BirthYear = birthYear; // Присваивание значения свойству BirthYear

}

public override string ToString() // Переопределение метода ToString для класса Author

{

return $"Имя: {Name}, Год рождения: {BirthYear}"; // Возвращаем строковое представление объекта класса Author

}

}

public class Book // Класс Book

{

public string Title { get; set; } // Свойство Title

public int ReleaseYear { get; set; } // Свойство ReleaseYear

public Author Author { get; set; } // Свойство Author

public Book() // Конструктор класса Book

{

Console.Write("Введите название книги: "); // Запрос на ввод названия книги

Title = Console.ReadLine(); // Чтение введенного названия книги

if (!Regex.IsMatch(Title, @"^[a-zA-Z0-9а-яА-Я\s]+$")) // Проверка названия книги на соответствие регулярному выражению

{

Console.WriteLine("Некорректный ввод названия книги. Название может содержать только буквы, цифры и пробелы."); // Сообщение об ошибке

throw new ArgumentException("Некорректный ввод названия книги"); // Генерация исключения

}

Console.Write("Введите год выпуска книги: "); // Запрос на ввод года выпуска книги

if (!int.TryParse(Console.ReadLine(), out int releaseYear) || releaseYear < 1 || releaseYear > 2023) // Проверка года выпуска на соответствие условиям

{

Console.WriteLine("Некорректный ввод года выпуска. Пожалуйста, введите число от 1 до 2023."); // Сообщение об ошибке

throw new ArgumentException("Некорректный ввод года выпуска"); // Генерация исключения

}

ReleaseYear = releaseYear; // Присваивание значения свойству ReleaseYear

Console.WriteLine("Введите информацию об авторе:");

Author = new Author(); // Создание объекта класса Author и присваивание его свойству Author

}

public override string ToString() // Переопределение метода ToString для класса Book

{

return $"Название: {Title}, Год выпуска: {ReleaseYear}, Автор: {Author}"; // Возвращаем строковое представление объекта класса Book

}

}

public class Vivod\_Author\_Book // Класс Vivod\_Author\_Book

{

static void Main(string[] args) // Главный метод программы

{

Vivod\_Author\_Book vivod\_author\_book = new Vivod\_Author\_Book(); // Создание объекта класса Vivod\_Author\_Book

vivod\_author\_book.Vivod\_author\_book(); // Вызов метода Vivod\_author\_book

}

public void Vivod\_author\_book() // Метод Vivod\_author\_book

{

try // Блок try для обработки исключений

{

Book book = new Book(); // Создание объекта класса Book

Console.WriteLine(book); // Вывод информации о книге

}

catch (ArgumentException) // Обработка исключения ArgumentException

{

}

}

}

На рисунке 3 представлен результат работы программы, а также работа всех исключений.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 – Результат работы программы

**4.** В листинге 4 представлен код по решению задачи создания интерфейса и абстрактного класса.

**Листинг4.** Код по решению задачи создания интерфейса и абстрактного класса.

using System;

public interface IDrawable // Интерфейс IDrawable, который определяет метод Draw

{

void Draw(); // Метод Draw, который должен быть реализован в классах, реализующих этот интерфейс

}

public class Circle : IDrawable // Класс Circle, реализующий интерфейс IDrawable

{

public double Radius { get; set; } // Свойство Radius для хранения радиуса круга

public Circle(double radius) // Конструктор класса Circle, принимающий радиус круга в качестве параметра

{

Radius = radius; // Присваивание значения параметра свойству Radius

}

public void Draw() // Реализация метода Draw для класса Circle

{

Console.WriteLine($"Рисуем круг с радиусом {Radius}"); // Вывод информации о круге

}

}

public class Rectangle : IDrawable // Класс Rectangle, реализующий интерфейс IDrawable

{

public double Width { get; set; } // Свойство Width для хранения ширины прямоугольника

public double Height { get; set; } // Свойство Height для хранения высоты прямоугольника

public Rectangle(double width, double height) // Конструктор класса Rectangle, принимающий ширину и высоту прямоугольника в качестве параметров

{

Width = width; // Присваивание значения параметра свойству Width

Height = height; // Присваивание значения параметра свойству Height

}

public void Draw() // Реализация метода Draw для класса Rectangle

{

Console.WriteLine($"Рисуем прямоугольник шириной {Width} и высотой {Height}"); // Вывод информации о прямоугольнике

}

}

public class Triangle : IDrawable // Класс Triangle, реализующий интерфейс IDrawable

{

public double Base { get; set; } // Свойство Base для хранения основания треугольника

public double Height { get; set; } // Свойство Height для хранения высоты треугольника

public Triangle(double baseLength, double height) // Конструктор класса Triangle, принимающий основание и высоту треугольника в качестве параметров

{

Base = baseLength; // Присваивание значения параметра свойству Base

Height = height; // Присваивание значения параметра свойству Height

}

public void Draw() // Реализация метода Draw для класса Triangle

{

Console.WriteLine($"Рисуем треугольник с основанием {Base} и высотой {Height}"); // Вывод информации о треугольнике

}

}

public class Interface // Класс Interface

{

static void Main(string[] args) // Главный метод программы

{

Interface iNterface = new Interface();

iNterface.vivod\_Interface();

}

public void vivod\_Interface()

{

Console.WriteLine("Введите радиус круга:");

double circleRadius = ReadDouble();

Console.WriteLine("Введите ширину прямоугольника:");

double rectangleWidth = ReadDouble();

Console.WriteLine("Введите высоту прямоугольника:");

double rectangleHeight = ReadDouble();

Console.WriteLine("Введите основание треугольника:");

double triangleBase = ReadDouble();

Console.WriteLine("Введите высоту треугольника:");

double triangleHeight = ReadDouble();

IDrawable[] drawables = new IDrawable[]

{

new Circle(circleRadius), // Создаем объект класса Circle с заданным радиусом и добавляем его в массив drawables

new Rectangle(rectangleWidth, rectangleHeight), // Создаем объект класса Rectangle с заданными шириной и высотой и добавляем его в массив drawables

new Triangle(triangleBase, triangleHeight) // Создаем объект класса Triangle с заданным основанием и высотой и добавляем его в массив drawables

};

foreach (var drawable in drawables) // Проходим по каждому элементу массива drawables

{

drawable.Draw(); // Вызываем метод Draw для текущего элемента

}

Console.ReadKey(); // Ждем нажатия клавиши пользователем, чтобы консольное окно не закрылось сразу после выполнения программы

}

private double ReadDouble() // Метод для безопасного чтения числа типа double из консоли

{

while (true) // Бесконечный цикл, который будет продолжаться, пока пользователь не введет корректное число

{

string input = Console.ReadLine(); // Читаем строку из консоли

if (double.TryParse(input, out double result)) // Пытаемся преобразовать строку в число типа double

{

return result; // Если преобразование успешно, возвращаем полученное число

}

else

{

Console.WriteLine("Некорректный ввод. Пожалуйста, введите число."); // Если преобразование не удалось, выводим сообщение об ошибке и повторяем цикл

}

}

}

}

На рисунке 4 представлен результат работы программы.

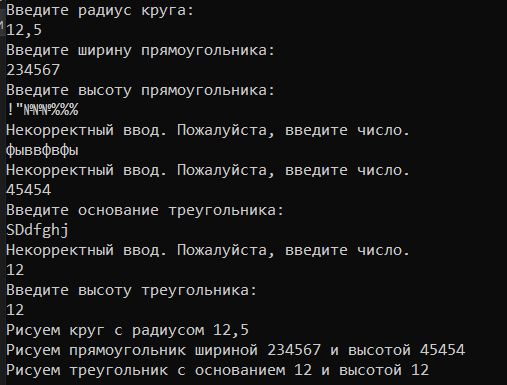


Рисунок 4 – Результат работы программы

**5.** В листинге 5 представлен код по решению задачи создания события.

**Листинг5.** Код по решению задачи создания события.

using System;

public class TemperatureSensor // Класс датчика температуры

{

private double \_temperature; // Закрытая переменная для хранения значения температуры

public delegate void TemperatureChangedHandler(double newTemperature); // Делегат, который будет использоваться для события изменения температуры

public event TemperatureChangedHandler TemperatureChanged; // Событие, которое происходит при изменении температуры

public double Temperature // Свойство для доступа к переменной \_temperature

{

get { return \_temperature; } // Возвращает значение \_temperature

set // Устанавливает значение \_temperature

{

\_temperature = value; // Присваивает новое значение \_temperature

TemperatureChanged?.Invoke(\_temperature); // Вызывает событие TemperatureChanged, если оно не равно null

}

}

}

public class Thermostat // Класс термостата

{

private bool \_heatingOn; // Закрытая переменная для хранения состояния отопления

public void OnTemperatureChanged(double newTemperature) // Метод, который вызывается при изменении температуры

{

if (newTemperature < 17) // Если температура меньше 17 градусов

{

Console.WriteLine("Температура снизилась. Включаем отопление."); // Выводим сообщение о включении отопления

\_heatingOn = true;

}

else if (newTemperature >= 17.0) // Если температура больше или равна 17 градусам

{

Console.WriteLine("Температура повысилась. Выключаем отопление."); // Выводим сообщение о выключении отопления

\_heatingOn = false;

}

}

}

public class vivod\_Temperature // Класс для вывода температуры

{

static void Main(string[] args) // Главный метод программы

{

vivod\_Temperature vivod\_tEmperature = new vivod\_Temperature(); // Создаем экземпляр класса vivod\_Temperature

vivod\_tEmperature.vivod\_temperature(); // Вызываем метод vivod\_temperature()

}

public void vivod\_temperature() // Метод для вывода температуры

{

var sensor = new TemperatureSensor(); // Создаем экземпляр класса TemperatureSensor

var thermostat = new Thermostat(); // Создаем экземпляр класса Thermostat

sensor.TemperatureChanged += thermostat.OnTemperatureChanged; // Подписываемся на событие изменения температуры

while (true)

{

Console.WriteLine("Введите новую температуру:");

string input = Console.ReadLine(); // Считываем ввод пользователя

double temperature;

if (Double.TryParse(input, out temperature)) // Пытаемся преобразовать ввод пользователя в число типа double

{

sensor.Temperature = temperature; // Если преобразование успешно, устанавливаем новое значение температуры

Console.ReadLine();

Console.Clear(); // Очищаем консоль

}

else

{

Console.WriteLine("Неправильный ввод. Пожалуйста, введите число."); // Если преобразование не удалось, выводим сообщение об ошибке

Console.ReadLine();

Console.Clear(); // Очищаем консоль

}

}

}

}

На рисунке 5 представлен результат работы программы, а также работа всех исключений.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 – Результат работы программы

**6.** В листинге 6 представлен код по решению задачи создания класса Student со своими свойствам.

**Листинг6.** Код по решению задачи создания класса Student со своими свойствам.

using System;

using System.Text.RegularExpressions;

using System.Numerics;

public class Student // Определение класса "Студент".

{

private string firstName; // Приватное поле для хранения имени студента.

private string lastName; // Приватное поле для хранения фамилии студента.

private BigInteger age; // Приватное поле для хранения возраста студента.

private double averageScore; // Приватное поле для хранения среднего балла студента.

public string FirstName // Свойство для доступа к имени студента.

{

get { return firstName; } // Геттер для получения имени студента.

set // Сеттер для установки имени студента.

{

if (!Regex.IsMatch(value, @"^[a-zA-Zа-яА-Я]+$")) // Проверка введенного имени на соответствие условию (только буквы).

throw new Exception("Имя студента должно содержать только буквы."); // Генерация исключения, если имя не соответствует условию.

firstName = value; // Установка значения имени студента.

}

}

public string LastName // Свойство для доступа к фамилии студента.

{

get { return lastName; } // Геттер для получения фамилии студента.

set // Сеттер для установки фамилии студента.

{

if (!Regex.IsMatch(value, @"^[a-zA-Zа-яА-Я]+$")) // Проверка введенной фамилии на соответствие условию (только буквы).

throw new Exception("Фамилия студента должна содержать только буквы."); // Генерация исключения, если фамилия не соответствует условию.

lastName = value; // Установка значения фамилии студента.

}

}

public BigInteger Age // Свойство для доступа к возрасту студента.

{

get { return age; } // Геттер для получения возраста студента.

set // Сеттер для установки возраста студента.

{

if (value < 0 || value > 122) // Проверка введенного возраста на соответствие условию (от 0 до 122).

throw new Exception("Возраст студента должен быть в диапазоне от 0 до 122 лет."); // Генерация исключения, если возраст не соответствует условию.

age = value; // Установка значения возраста студента.

}

}

public double AverageScore // Свойство для доступа к среднему баллу студента.

{

get { return averageScore; } // Геттер для получения среднего балла студента.

set // Сеттер для установки среднего балла студента.

{

if (value < 0.0 || value > 10.0) // Проверка введенного среднего балла на соответствие условию (от 0.0 до 10.0).

throw new Exception("Средний балл должен быть в диапазоне от 0.0 до 10.0."); // Генерация исключения, если средний балл не соответствует условию.

averageScore = value; // Установка значения среднего балла студента.

}

}

public Student(string firstName, string lastName, BigInteger age, double averageScore) // Конструктор класса "Студент".

{

FirstName = firstName; // Установка значения имени студента.

LastName = lastName; // Установка значения фамилии студента.

Age = age; // Установка значения возраста студента.

AverageScore = averageScore; // Установка значения среднего балла студента.

}

public void PrintStudentInfo() // Метод для вывода информации о студенте.

{

Console.WriteLine($"Имя: {FirstName}, Фамилия: {LastName}, Возраст: {Age}, Средний балл: {AverageScore}"); // Вывод информации о студенте.

}

}

public class Vivod\_Studenta // Класс для вывода информации о студенте.

{

static void Main(string[] args) // Главный метод программы.

{

Vivod\_Studenta vivod\_studenta = new Vivod\_Studenta(); // Создание экземпляра класса Vivod\_Studenta.

vivod\_studenta.Vivod\_studenta(); // Вызов метода Vivod\_studenta().

}

public void Vivod\_studenta() // Метод для ввода данных о студенте и вывода информации о нем.

{

try // Блок try-catch для обработки исключений.

{

Console.Write("Введите имя студента: ");

string firstName = Console.ReadLine(); // Чтение ввода пользователя.

Console.Write("Введите фамилию студента: ");

string lastName = Console.ReadLine(); // Чтение ввода пользователя.

Console.Write("Введите возраст студента: ");

BigInteger age = BigInteger.Parse(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в BigInteger.

Console.Write("Введите средний балл студента: ");

double averageScore = Convert.ToDouble(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в double.

Student student = new Student(firstName, lastName, age, averageScore); // Создание экземпляра класса "Студент" с введенными данными.

student.PrintStudentInfo(); // Вызов метода PrintStudentInfo() для вывода информации о студенте.

Console.ReadKey(); // Ожидание нажатия клавиши пользователем перед завершением программы.

}

catch (FormatException) // Обработка исключений формата ввода.

{

Console.WriteLine("Произошла ошибка: Возраст студента должен быть в диапазоне от 0 до 122 лет."); // Вывод сообщения об ошибке.

}

catch (Exception ex) // Обработка всех остальных исключений.

{

Console.WriteLine($"Произошла ошибка: {ex.Message}"); // Вывод сообщения об ошибке.

}

Console.ReadKey(); // Ожидание нажатия клавиши пользователем перед завершением программы.

}

}

На рисунке 6 представлен результат работы программы, а также работа всех исключений.

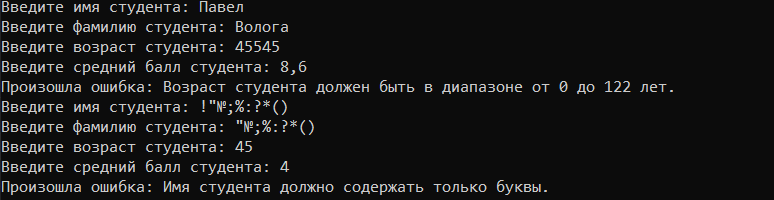


Рисунок 6 – Результат работы программы

**7.** В листинге 7 представлен код по решению задачи создания структуры Train.

**Листинг7.** Код по решению задачи создания структуры Train.

using System;

using System.Linq;

public struct Train // Определение структуры "Поезд".

{

public string Destination; // Публичное поле для хранения пункта назначения поезда.

public int Number; // Публичное поле для хранения номера поезда.

public DateTime DepartureTime; // Публичное поле для хранения времени отправления поезда.

public Train(string destination, int number, DateTime departureTime) // Конструктор структуры "Поезд".

{

Destination = destination; // Установка значения пункта назначения поезда.

Number = number; // Установка значения номера поезда.

DepartureTime = departureTime; // Установка значения времени отправления поезда.

}

public void PrintTrainInfo() // Метод для вывода информации о поезде.

{

Console.WriteLine($"Пункт назначения: {Destination}, Номер поезда: {Number}, Время отправления: {DepartureTime}"); // Вывод информации о поезде.

}

}

public class Trains // Определение класса "Поезда".

{

Train[] trains = new Train[5]; // Массив из пяти элементов структуры "Поезд".

public void vivod\_trains() // Метод для ввода данных о поездах и вывода информации о них.

{

trains = trains.OrderBy(t => t.Number).ToArray(); // Сортировка массива поездов по номеру поезда.

for (int i = 0; i < 5; i++) // Цикл для ввода данных о пяти поездах.

{

try // Блок try-catch для обработки исключений.

{

Console.WriteLine("Введите пункт назначения, номер поезда и время отправления (через пробел):"); // Вывод сообщения.

string[] input = Console.ReadLine().Split(' '); // Чтение ввода пользователя и разделение его на части.

trains[i] = new Train(input[0], int.Parse(input[1]), DateTime.Parse(input[2])); // Создание нового поезда с введенными данными и добавление его в массив.

}

catch (Exception) // Обработка всех возможных исключений.

{

Console.WriteLine("Неправильный ввод. Пожалуйста, попробуйте еще раз."); // Вывод сообщения об ошибке.

i--; // Уменьшение счетчика цикла на один, чтобы повторить ввод данных для текущего поезда.

}

}

try // Блок try-catch для обработки исключений.

{

Console.WriteLine("Введите номер поезда:"); // Вывод сообщения.

int number = int.Parse(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в число.

Train train = trains.FirstOrDefault(t => t.Number == number); // Поиск поезда с введенным номером в массиве.

if (train.Number != 0) // Если такой поезд найден...

train.PrintTrainInfo(); // ...вывод информации о нем...

else

Console.WriteLine("Поезд с таким номером не найден."); // ...иначе вывод сообщения об ошибке.

}

catch (Exception) // Обработка всех возможных исключений.

{

Console.WriteLine("Неправильный ввод. Пожалуйста, введите номер поезда."); // Вывод сообщения об ошибке.

}

trains = trains.OrderBy(t => t.Destination).ThenBy(t => t.DepartureTime).ToArray(); // Сортировка массива поездов по пункту назначения и времени отправления.

}

static void Main(string[] args) // Главный метод программы.

{

Trains trainProgram = new Trains(); // Создание экземпляра класса "Поезда".

trainProgram.vivod\_trains(); // Вызов метода для ввода данных о поездах и вывода информации о них.

Console.ReadKey(); // Ожидание нажатия клавиши пользователем перед завершением программы.

}

}

На рисунке 7 представлен результат работы программы, а также работа всех исключений.

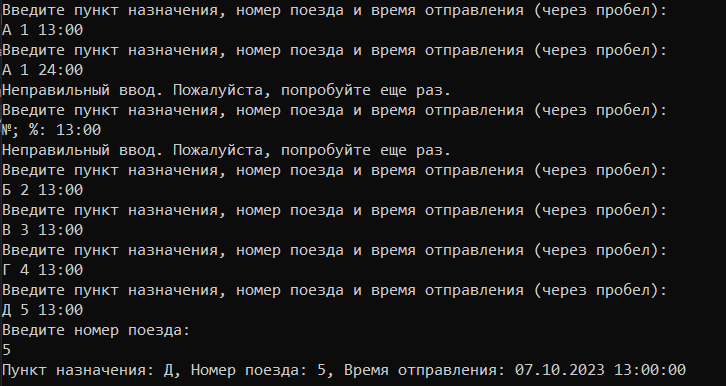


Рисунок 7 – Результат работы программы

**8.** В листинге 8 представлен код по решению разработки иерархии классов для геометрических фигур.

**Листинг8.** Код по решению разработки иерархии классов для геометрических фигур.

using System;

using System.Text.RegularExpressions;

public class Figuri // Определение класса "Фигуры".

{

static void Main(string[] args) // Главный метод программы.

{

Figuri figuri = new Figuri(); // Создание экземпляра класса "Фигуры".

figuri.vivod\_Figuri(); // Вызов метода для ввода данных о фигурах и вывода информации о них.

}

public void vivod\_Figuri() // Метод для ввода данных о фигурах и вывода информации о них.

{

try // Блок try-catch для обработки исключений.

{

Console.WriteLine("Введите радиус круга:"); // Вывод сообщения.

double circleRadius = Convert.ToDouble(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в число.

if (circleRadius < 0) throw new Exception("Радиус не может быть отрицательным числом."); // Проверка введенного радиуса на соответствие условию (неотрицательное число).

Circle circle = new Circle(circleRadius); // Создание нового круга с введенным радиусом.

Console.WriteLine($"Площадь круга: {circle.Area()}, Периметр: {circle.Perimeter()}"); // Вывод площади и периметра круга.

Console.WriteLine("Введите ширину и высоту прямоугольника:"); // Вывод сообщения.

double rectangleWidth = Convert.ToDouble(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в число.

double rectangleHeight = Convert.ToDouble(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в число.

if (rectangleWidth < 0 || rectangleHeight < 0) throw new Exception("Ширина и высота не могут быть отрицательными числами."); // Проверка введенной ширины и высоты на соответствие условию (неотрицательные числа).

Rectangle rectangle = new Rectangle(rectangleWidth, rectangleHeight); // Создание нового прямоугольника с введенной шириной и высотой.

Console.WriteLine($"Площадь прямоугольника: {rectangle.Area()}, Периметр: {rectangle.Perimeter()}"); // Вывод площади и периметра прямоугольника.

Console.WriteLine("Введите основание, высоту и две стороны треугольника:"); // Вывод сообщения.

double triangleBase = Convert.ToDouble(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в число.

double triangleHeight = Convert.ToDouble(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в число.

double triangleSideA = Convert.ToDouble(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в число.

double triangleSideB = Convert.ToDouble(Console.ReadLine()); // Чтение ввода пользователя и преобразование его в число.

if (triangleBase < 0 || triangleHeight < 0 || triangleSideA < 0 || triangleSideB < 0) throw new Exception("Основание, высота и стороны не могут быть отрицательными числами."); // Проверка введенного основания, высоты и сторон на соответствие условию (неотрицательные числа).

Triangle triangle = new Triangle(triangleBase, triangleHeight, triangleSideA, triangleSideB); // Создание нового треугольника с введенным основанием, высотой и сторонами.

Console.WriteLine($"Площадь треугольника: {triangle.Area()}, Периметр: {triangle.Perimeter()}"); // Вывод площади и периметра треугольника.

Console.ReadKey(); // Ожидание нажатия клавиши пользователем перед завершением программы.

}

catch (FormatException) // Обработка исключений формата ввода.

{

Console.WriteLine("Ошибка: ввод должен быть числом."); // Вывод сообщения об ошибке.

Console.ReadKey(); // Ожидание нажатия клавиши пользователем перед завершением программы.

}

catch (Exception ex) // Обработка всех остальных исключений.

{

Console.WriteLine($"Ошибка: {ex.Message}"); // Вывод сообщения об ошибке.

Console.ReadKey(); // Ожидание нажатия клавиши пользователем перед завершением программы.

}

}

}

public abstract class Shape // Определение абстрактного класса "Фигура".

{

public abstract double Area(); // Абстрактный метод для вычисления площади фигуры.

public abstract double Perimeter(); // Абстрактный метод для вычисления периметра фигуры.

}

public class Circle : Shape // Определение класса "Круг", наследующего от класса "Фигура".

{

public double Radius { get; set; } // Свойство для доступа к радиусу круга.

public Circle(double radius) // Конструктор класса "Круг".

{

Radius = radius; // Установка значения радиуса круга.

}

public override double Area() // Метод для вычисления площади круга.

{

return Math.PI \* Math.Pow(Radius, 2); // Возврат значения площади круга.

}

public override double Perimeter() // Метод для вычисления периметра круга.

{

return 2 \* Math.PI \* Radius; // Возврат значения периметра круга.

}

}

public class Rectangle : Shape // Определение класса "Прямоугольник", наследующего от класса "Фигура".

{

public double Width { get; set; } // Свойство для доступа к ширине прямоугольника.

public double Height { get; set; } // Свойство для доступа к высоте прямоугольника.

public Rectangle(double width, double height) // Конструктор класса "Прямоугольник".

{

Width = width; // Установка значения ширины прямоугольника.

Height = height; // Установка значения высоты прямоугольника.

}

public override double Area() // Метод для вычисления площади прямоугольника.

{

return Width \* Height; // Возврат значения площади прямоугольника.

}

public override double Perimeter() // Метод для вычисления периметра прямоугольника.

{

return 2 \* (Width + Height); // Возврат значения периметра прямоугольника.

}

}

public class Triangle : Shape // Определение класса "Треугольник", наследующего от класса "Фигура".

{

public double Base { get; set; } // Свойство для доступа к основанию треугольника.

public double Height { get; set; } // Свойство для доступа к высоте треугольника.

public double SideA { get; set; } // Свойство для доступа к первой стороне треугольника.

public double SideB { get; set; } // Свойство для доступа ко второй стороне треугольника.

public Triangle(double baseLength, double height, double sideA, double sideB) // Конструктор класса "Треугольник".

{

Base = baseLength; // Установка значения основания треугольника.

Height = height; // Установка значения высоты треугольника.

SideA = sideA; // Установка значения первой стороны треугольника.

SideB = sideB; // Установка значения второй стороны треугольника.

}

public override double Area() // Метод для вычисления площади треугольника.

{

return 0.5 \* Base \* Height; // Возврат значения площади треугольника.

}

public override double Perimeter() // Метод для вычисления периметра треугольника.

{

return Base + SideA + SideB; // Возврат значения периметра треугольника.

}

}

На рисунке 8 представлен результат работы программы, а также работа всех исключений.

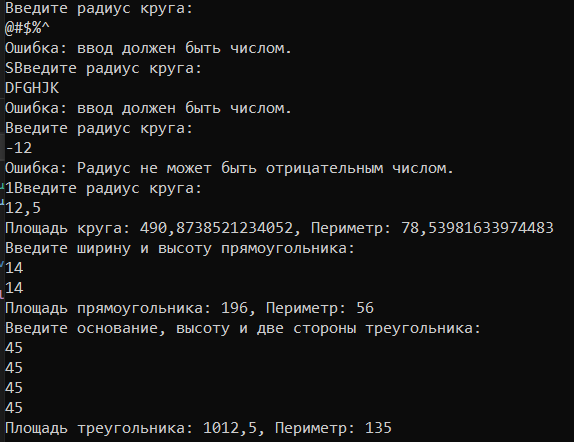


Рисунок 8 – Результат работы программы

Далее было создано отдельно консольное приложение, в котором был следующий листинг9.

**Листинг9.**Код конечного приложения.

using System;

public class Vologa\_Modul\_2

{

static void Main(string[] args)

{

vivod\_Person VIvod\_Person = new vivod\_Person(); // Создание нового объекта класса vivod\_Person

VIvod\_Person.Vivod\_Person(); // Вызов метода Vivod\_Person объекта VIvod\_Person для вывода информации о человеке

Vivod\_Figur vivod\_Figur = new Vivod\_Figur(); // Создание объекта класса Vivod\_Figur

vivod\_Figur.vivod\_figur(); // Вызов метода vivod\_figur

Vivod\_Author\_Book vivod\_author\_book = new Vivod\_Author\_Book(); // Создание объекта класса Vivod\_Author\_Book

vivod\_author\_book.Vivod\_author\_book(); // Вызов метода Vivod\_author\_book

Interface iNterface = new Interface();// Создание объекта класса Interface

iNterface.vivod\_Interface();// Вызов метода vivod\_Interface

vivod\_Temperature vivod\_tEmperature = new vivod\_Temperature(); // Создаем экземпляр класса vivod\_Temperature

vivod\_tEmperature.vivod\_temperature(); // Вызываем метод vivod\_temperature()

Vivod\_Studenta vivod\_studenta = new Vivod\_Studenta(); // Создание экземпляра класса Vivod\_Studenta.

vivod\_studenta.Vivod\_studenta(); // Вызов метода Vivod\_studenta().

Trains trainProgram = new Trains(); // Создание экземпляра класса "Поезда".

trainProgram.vivod\_trains(); // Вызов метода для ввода данных о поездах и вывода информации о них.

Figuri figuri = new Figuri(); // Создание экземпляра класса "Фигуры".

figuri.vivod\_Figuri(); // Вызов метода для ввода данных о фигурах и вывода информации о них.

}

}

Данное приложение работает следующим образом (рисунок 9).

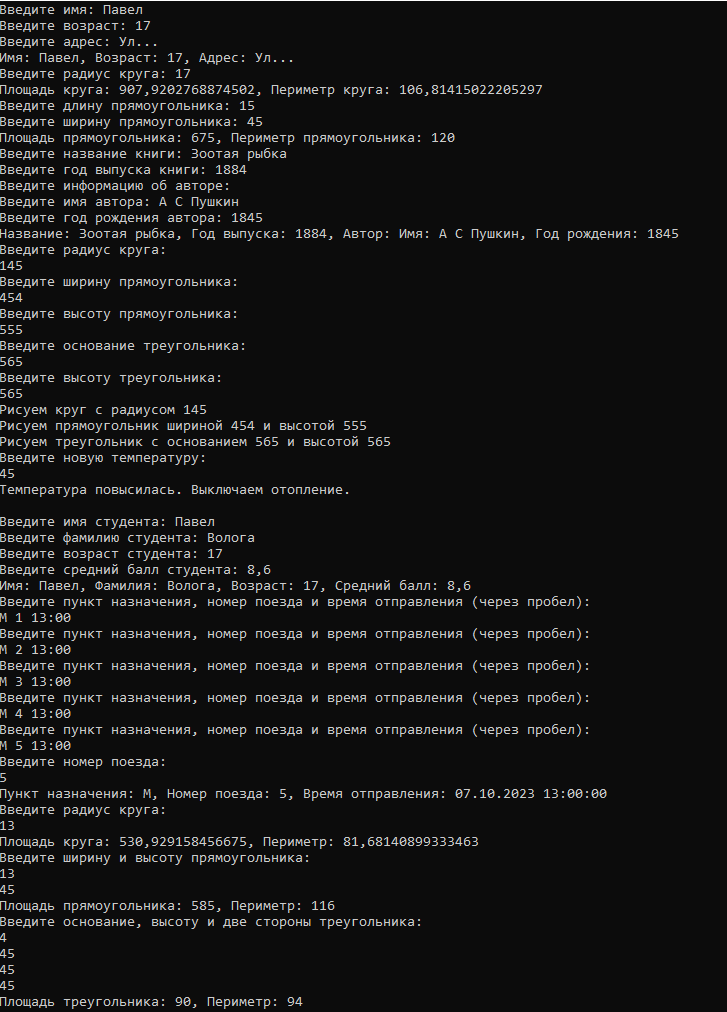


Рисунок 9 ­– Результат работы программы

Работает она таким образом, потому что к конечному проекту подключены все остальные проекты. А здесь они только вызываются.

Так же был создан репозиторий на GitHub. Ссылка –> https://github.com/PaHenChIk/Vologa\_Modul2.

На рисунке 10 представлены файле выложенные на GitHub.

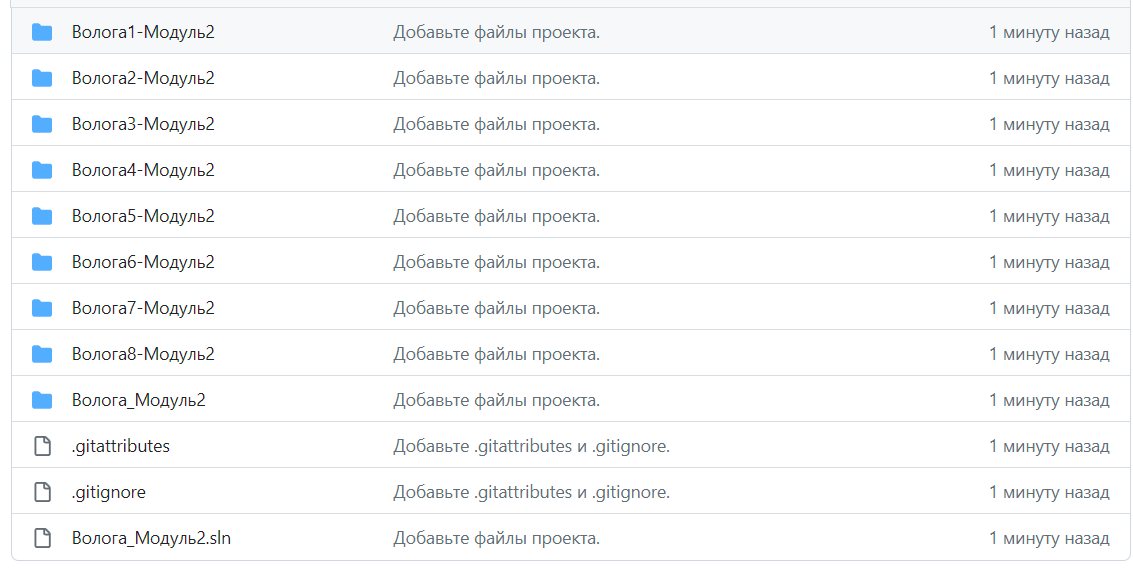


Рисунок 10 – Файлы выложенные GitHub

**Вывод: .**