МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8

# по дисциплине «Объектно-ориентированное проектирование и программирование»

на тему: «Интерфейсы и абстрактные классы»

Выполнил студент гр. ИТИ-21:  
Буякевич Д.О.  
 Принял преподаватель:  
 Чернявская А.Э.

Гомель 2022

**Цель работы:** научиться работать с наследованием и абстрактными классами.

**Ход работы:**

На рисунке 1 показан вариант лабораторной работы, а на рисунке 2 требования к выполнению.

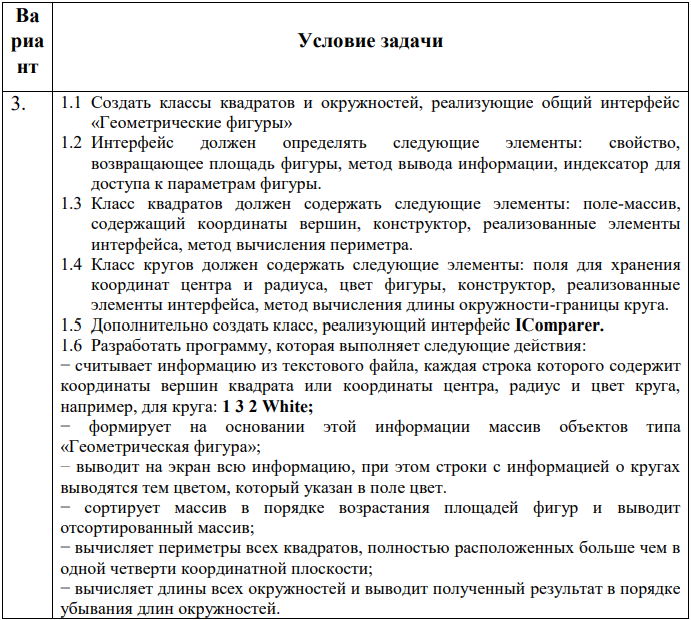


Рисунок 1 – Вариант

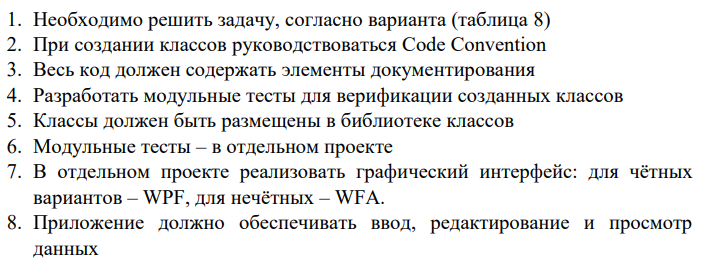


Рисунок 2 – Требования

На рисунках 3-7 показана работа программы.

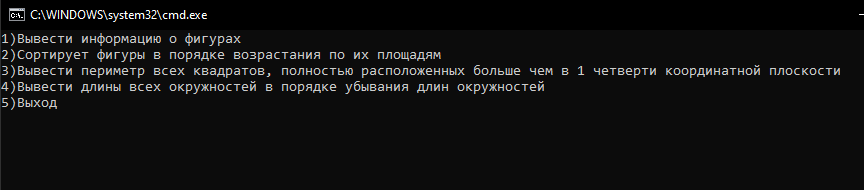


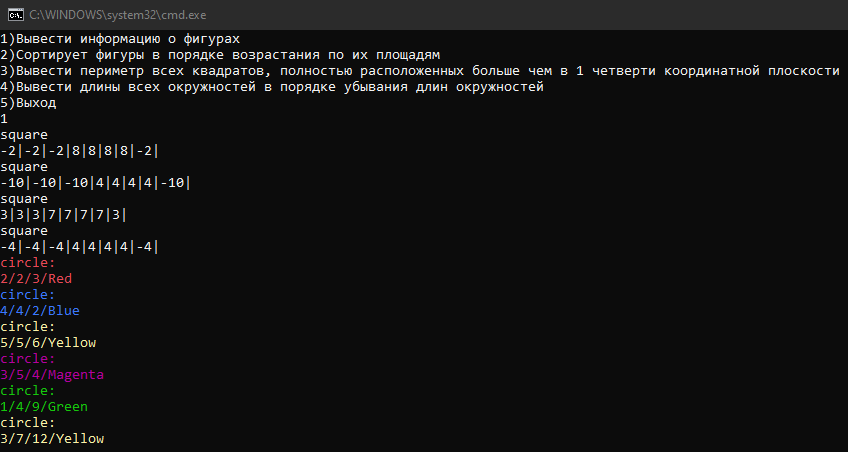
Рисунок 3 – Меню 

Рисунок 4 – Пункт 1

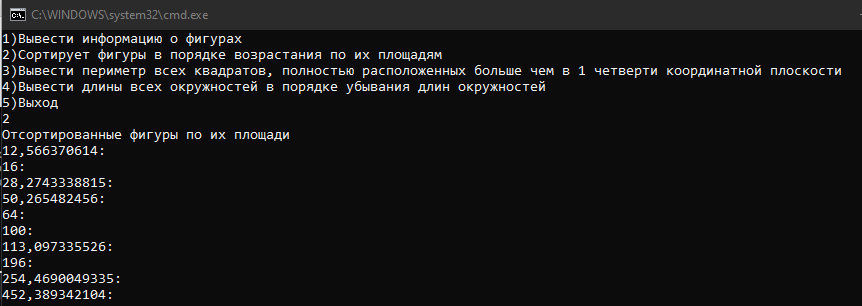


Рисунок 5 – Пункт 2

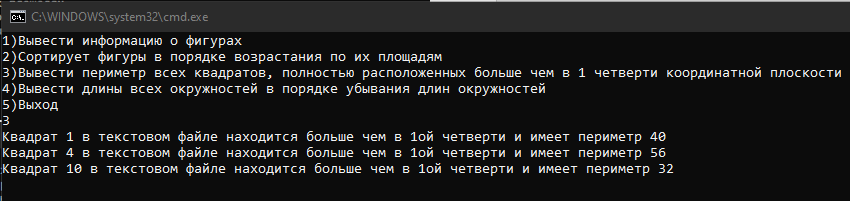


Рисунок 6 – Пункт 3

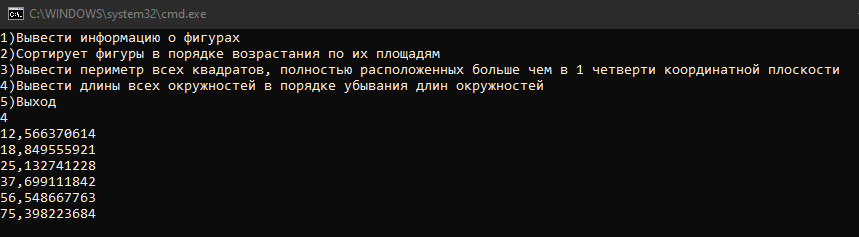


Рисунок 7 – Пункт 4

**Вывод:** в процессе работы познакомились с абстрактными классами и интерфейсами, реализовывали интерфейсы в классах.

**ПРИЛОЖЕНИЕ А**

**ЛИСТИНГ ПРОГРАММЫ**

**Program.cs**

using System;

namespace Lab\_08

{

internal static class Program

{

static void Main(string[] args)

{

GeometricShape geometricShape = new GeometricShape();

bool isWork = true;

int punctOfMenu = 0;

while (isWork)

{

GeometricShape.CountOfCircles();

geometricShape.FindAllShapesInTextFile();

geometricShape.CloseInputSphereCoordinates();

Console.WriteLine("1)Вывести информацию о фигурах");

Console.WriteLine("2)Сортирует фигуры в порядке возрастания по их площадям");

Console.WriteLine("3)Вывести периметр всех квадратов, полностью расположенных больше чем в 1 четверти координатной плоскости");

Console.WriteLine("4)Вывести длины всех окружностей в порядке убывания длин окружностей");

Console.WriteLine("5)Выход");

punctOfMenu = Convert.ToInt32(Console.ReadLine());

switch (punctOfMenu)

{

case 1:

geometricShape.OutPutSquare();

geometricShape.OutPutCircle();

break;

case 2:

geometricShape.GetArea();

geometricShape.SortByArea();

geometricShape.OutPutSortedAreas();

//geometricShape.OutPutSquare();

//geometricShape.OutPutCircle();

break;

case 3:

geometricShape.GetSquarePerimetrerMoreThanOneQuater();

break;

case 4:

geometricShape.GetCircleLength();

break;

case 5:

isWork = false;

break;

default:

Console.Clear();

continue;

}

Console.ReadKey();

Console.Clear();

}

}

}

}

**Square.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_08

{

internal class Square : GeometricShape,IGeometricShapes

{

public int[] coordinates = new int[8];

public double Area { get; set; }

public Square(int leftBotX, int leftBotY, int leftTopX, int leftTopY, int rightTopX, int rightTopY, int rightBotX, int rightBotY)

{

coordinates[0] = leftBotX;

coordinates[1] = leftBotY;

coordinates[2] = leftTopX;

coordinates[3] = leftTopY;

coordinates[4] = rightTopX;

coordinates[5] = rightTopY;

coordinates[6] = rightBotX;

coordinates[7] = rightBotY;

}

public double GetArea()

{

Area = (coordinates[3] - coordinates[1]) \* (coordinates[3] - coordinates[1]);

return Area;

}

public double Perimetr()

{

return 1;

}

public void ShowInformation()

{

Console.WriteLine("Координата квадрата: " + coordinates[0]);

Console.WriteLine("Координата квадрата: " + coordinates[1]);

Console.WriteLine("Координата квадрата: " + coordinates[2]);

Console.WriteLine("Координата квадрата: " + coordinates[3]);

Console.WriteLine("Координата квадрата: " + coordinates[4]);

Console.WriteLine("Координата квадрата: " + coordinates[5]);

Console.WriteLine("Координата квадрата: " + coordinates[6]);

Console.WriteLine("Координата квадрата: " + coordinates[7]);

}

public int this[int index]

{

get => coordinates[index];

set => coordinates[index] = value;

}

}

}

**Circle.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_08

{

internal class Circle : GeometricShape,IGeometricShapes

{

private int[] coordinates = new int[2];

private int \_radius;

private string \_circleColor;

public double Area { get; set; }

public Circle(int x,int y,int radius,string color)

{

coordinates[0] = x;

coordinates[1] = y;

\_radius = radius;

\_circleColor = color;

}

public void ShowInformation()

{

Console.WriteLine($"Координаты центра круга: {{{coordinates[0]},{coordinates[1]}}}");

Console.WriteLine($"Радиус: {\_radius}");

Console.WriteLine($"Цвет окружности: {\_circleColor}");

}

public double CircleLength()

{

const double pi = 3.1415926535;

double circleLength = 2 \* pi \* \_radius;

return circleLength;

}

public int this[int index]

{

get => coordinates[index];

set => coordinates[index] = value;

}

}

}

**IGeometricShapes.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_08

{

internal interface IGeometricShapes

{

double Area { get; set; }

void ShowInformation();

int this[int index] { get;set; }

}

}

**GeometricShape.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Collections;

namespace Lab\_08

{

public class GeometricShape

{

public double Area { get; set; }

static private int \_squareCounter;

static private int \_circleCounter;

static readonly string[] textFile = File.ReadAllLines($"Shapes/shapes.txt");

private readonly int[,] squareCoordinates = new int[textFile.Length, 8];

private readonly int[,] circleCoordinates = new int[textFile.Length, 3];

private readonly string[] circleColor = new string[textFile.Length];

private readonly double[] ShapesArea = new double[CountOfSquares() + CountOfCircles()];

private readonly double[] SquarePerimetr = new double[CountOfSquares()];

private readonly double[] CirclesLength = new double[CountOfCircles()];

// Временные массивы для выборки информации из текстового файла

string[] stringWithSquareCoordinates;

string[] stringWithCircleCoordinates;

string[] stringWithCircleColor;

public GeometricShape[] Shapes = new GeometricShape[textFile.Length];

ConsoleColor[] consoleColor = new ConsoleColor[5] { ConsoleColor.Red, ConsoleColor.Green, ConsoleColor.Yellow, ConsoleColor.Blue, ConsoleColor.Magenta };

// находит все фигуры в файле и создает новые объекты

public void FindAllShapesInTextFile()

{

for (int i = 0; i < textFile.Length; i++)

{

if (textFile[i].StartsWith("Square:"))

{

Shapes[i] = new Square(squareCoordinates[i, 0], squareCoordinates[i, 1], squareCoordinates[i, 2], squareCoordinates[i, 3], squareCoordinates[i, 4], squareCoordinates[i, 5], squareCoordinates[i, 6], squareCoordinates[i, 7]);

}

else if (textFile[i].StartsWith("Circle:"))

{

Shapes[i] = new Circle(circleCoordinates[i, 0], circleCoordinates[i, 1], circleCoordinates[i, 2], circleColor[i]);

}

}

}

public static int CountOfCircles()

{

for (int i = 0; i < textFile.Length; i++)

{

if (textFile[i].StartsWith("Circle:"))

{

\_circleCounter++;

}

}

return \_circleCounter;

}

public static int CountOfSquares()

{

for (int i = 0; i < textFile.Length; i++)

{

if (textFile[i].StartsWith("Square:"))

{

\_squareCounter++;

}

}

return \_squareCounter;

}

public void CloseInputSphereCoordinates()

{

for (int i = 0; i < textFile.Length; i++)

{

if (textFile[i].StartsWith("Square:"))

{

stringWithSquareCoordinates = textFile[i].Substring(8).Split(',');

for (int j = 0; j < squareCoordinates.GetLength(1); j++)

{

squareCoordinates[i, j] = Int32.Parse(stringWithSquareCoordinates[j]);

}

}

if (textFile[i].StartsWith("Circle:"))

{

stringWithCircleColor = textFile[i].Substring(8).Split(',');

circleColor[i] = stringWithCircleColor[3];

for (int k = 0; k < consoleColor.Length; k++)

{

if (circleColor[i].ToString() == consoleColor[k].ToString())

{

stringWithCircleCoordinates = textFile[i].Substring(8).Split(',');

for (int j = 0; j < circleCoordinates.GetLength(1); j++)

{

circleCoordinates[i, j] = Int32.Parse(stringWithCircleCoordinates[j]);

}

}

}

circleColor[i] = stringWithCircleColor[3];

}

}

}

// вывод квадратов из текстового файла

public string OutPutSquare()

{

for (int i = 0; i < textFile.Length; i++)

{

if (textFile[i].StartsWith("Square:"))

{

Console.WriteLine("square");

for (int j = 0; j < squareCoordinates.GetLength(1); j++)

{

Console.Write(squareCoordinates[i, j] + "|");

}

Console.WriteLine();

}

}

return " ";

}

// вывод кругов из текстового файла

public string OutPutCircle()

{

for (int i = 0; i < textFile.Length; i++)

{

if (textFile[i].StartsWith("Circle:"))

{

for (int k = 0; k < consoleColor.Length; k++)

{

if (circleColor[i].ToString() == consoleColor[k].ToString())

{

Console.ForegroundColor = consoleColor[k];

Console.WriteLine("circle:");

for (int j = 0; j < circleCoordinates.GetLength(1); j++)

{

Console.Write(circleCoordinates[i, j] + "/");

}

}

}

Console.Write(circleColor[i]);

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine();

}

}

return "";

}

// Получить площади фигур

public void GetArea()

{

const double pi = 3.1415926535;

for (int i = 0; i < Shapes.Length; i++)

{

if (Shapes[i] is Circle)

{

ShapesArea[i] = pi \* circleCoordinates[i, 2] \* circleCoordinates[i, 2];

//Console.WriteLine(Area);

}

if (Shapes[i] is Square)

{

ShapesArea[i] = (squareCoordinates[i, 3] - squareCoordinates[i, 1]) \* (squareCoordinates[i, 3] - squareCoordinates[i, 1]);

//Console.WriteLine(Area);

}

}

}

// Отсортировать фигуры по площади

public void SortByArea()

{

Array.Sort(ShapesArea);

}

public void OutPutSortedAreas()

{

Console.WriteLine("Отсортированные фигуры по их площади");

for (int i = 0; i < ShapesArea.Length; i++)

{

Console.WriteLine(ShapesArea[i] + ": ");

}

}

// Получить периметр квадратов которые лежат больше чем в одной четверти

public void GetSquarePerimetrerMoreThanOneQuater()

{

double perimetr;

int k = 0;

for (int i = 0; i < Shapes.Length; i++)

{

if (Shapes[i] is Square)

{

if (((squareCoordinates[i, 4] > 0 && squareCoordinates[i, 5] > 0) && ((squareCoordinates[i, 0] < 0) && (squareCoordinates[i, 1] > 0))) || (((squareCoordinates[i, 4] > 0) && (squareCoordinates[i, 5] > 0)) && (squareCoordinates[i, 0] > 0) && (squareCoordinates[i, 1] < 0)) || ((squareCoordinates[i, 4] > 0 && squareCoordinates[i, 5] < 0) && ((squareCoordinates[i, 0] < 0) && (squareCoordinates[i, 0] < 0))) || ((squareCoordinates[i, 4] < 0 && squareCoordinates[i, 5] > 0) && ((squareCoordinates[i, 0] < 0) && (squareCoordinates[i, 0] < 0))) || ((squareCoordinates[i, 4] > 0 && squareCoordinates[i, 5] > 0) && ((squareCoordinates[i, 0] < 0) && (squareCoordinates[i, 0] < 0))))

{

perimetr = Math.Abs((squareCoordinates[i, 4] - squareCoordinates[i, 2]) \* 4);

SquarePerimetr[k] = perimetr;

Console.WriteLine($"Квадрат {i + 1} в текстовом файле находится больше чем в 1ой четверти и имеет периметр {perimetr}");

k++;

}

}

}

}

public void GetCircleLength()

{

const double pi = 3.1415926535;

int k = 0;

for (int i = 0; i < Shapes.Length; i++)

{

if (Shapes[i] is Circle)

{

CirclesLength[k] = 2 \* pi \* circleCoordinates[i, 2];

k++;

}

}

Array.Sort(CirclesLength);

for (int i = 0; i < CirclesLength.Length; i++)

{

if (CirclesLength[i] != 0)

{

Console.WriteLine(CirclesLength[i]);

}

}

}

}

}

**Comperer.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_08

{

internal class Comperer : IComparer<IGeometricShapes>

{

public int Compare(IGeometricShapes shape1,IGeometricShapes shape2)

{

if (shape1.Area == shape2.Area)

{

return 1;

}

else

{

return -1;

}

}

}

}