Wydział Sztuki,Techniki i Komunikacji

Kierunek studiów: informatyka

Przedmiot: Programowanie Obiektowe

Dokumentacja

Projektu nr 3

Autor Piotr Skupiński

nr albumu: 59369 grupa 2

Warszawa 06.2022

Zawartość

[Klasa Prostokąt 3](#_Toc104371119)

[Klasa Kwadrat 3](#_Toc104371120)

[Klasa Koło 4](#_Toc104371121)

[Klasa FillElipse 4](#_Toc104371122)

[KlasaFillRectangle 5](#_Toc104371123)

[Klasa DrawPie 5](#_Toc104371124)

[Klasa FillPie 6](#_Toc104371125)

[Klasa DrawArc 6](#_Toc104371126)

[Klasa KrzywaKardynalna 7](#_Toc104371127)

[Klasa FillClosedCurve 9](#_Toc104371128)

[Klasa WielokątForemny 10](#_Toc104371129)

[Klasa WielokątForemnyWypełniony 11](#_Toc104371130)

[Dokumentacja Formularzy 11](#_Toc104371131)

[Formularz FormFiguryGeoInitial 11](#_Toc104371132)

[Formularz Prezentacja Losowa Ze Slajderem 12](#_Toc104371133)

[Formularz Kreślenie Figur Geometrycznych 13](#_Toc104371134)

[Samoocena 13](#_Toc104371135)

[Samoocena Labolatorium 13](#_Toc104371136)

[Samoocena Projekt 14](#_Toc104371137)

## Klasa BryłaAbstrakcyjna

Klasa została uzupełniona o funkcjonalność get; set; dla Wysokości, Kierunku Obrotu i PromieniaBryły za według schematu

protected int WysokośćBryły;

public int Wysokość\_Bryły

{

get { return WysokośćBryły; }

set { WysokośćBryły = value; }

}

A także o nową funkcję abstrakcyjną OdświeżAtrybuty która służy do aktualizacji atrybutów bryły.

public abstract void OdświeżAtrybuty();

## Klasa Walec

Do konstruktora klasy został dodany kierunek obrotu.

Ta zmiana dotyczy wszystkich brył.

Klasa Walec została uzupełniona o kalkulacje Powierzchni i Objętości, tak samo jak pozostałe klasy które miały łatwe do zaimplementowania wzory na te właściwości.

this.ObjętośćBryły = (float)(Math.PI \* PromieńBryły \*

PromieńBryły \* WysokośćBryły);

this.PowierzchniaBryły = (float)(2 \* Math.PI \* PromieńBryły \*

PromieńBryły + 2 \* Math.PI \* PromieńBryły \* WysokośćBryły);

Klasa posiada też pierwszą implementację Metody OdświeżAtrybuty

public override void OdświeżAtrybuty()

{

ObjętośćBryły = (float)(Math.PI \* PromieńBryły \*

PromieńBryły \* WysokośćBryły);

PowierzchniaBryły = (float)(2 \* Math.PI \* PromieńBryły \*

PromieńBryły + 2 \* Math.PI \* PromieńBryły \* WysokośćBryły);

}

Ponieważ metoda ta została zaimplementowana po utworzeniu możliwości zmiany promienia, kierunku obrotu i wysokości do metody wykreśl zostały przesunięte obliczenia:

YsS = YsP - WysokośćBryły;

XsS = XsP;

Oś\_Duża = PromieńBryły \* 2;

Oś\_Mała = PromieńBryły / 2;

## Klasa WalecPochyły

public WalecPochyły(int PromieńBryły, int WysokośćBryły, int StopieńWielokąta,

int XsP, int YsP, float KątNachylenia, bool KierunekObrotu, Color KolorLinii,

DashStyle StylLinii, float GrubośćLinii) : base(PromieńBryły, WysokośćBryły, StopieńWielokąta, XsP, YsP, KierunekObrotu, KolorLinii, StylLinii, GrubośćLinii)

{

RodzajBryły = TypyBryły.BG\_WalecPochyły;

this.KątNachylenia = KątNachylenia;

//wyznaczenie przesunięcia wierzchołka stożka

XsS = (int)(XsP + (WysokośćBryły / Math.Tan(Math.PI \* KątNachylenia / 180)));

}

public override void Wykreśl(Graphics Rysownica)

{

//wyznaczenie pozostałych współrzędnych środka sufitu Walca

YsS = YsP - WysokośćBryły;

XsS = (int)(XsP + (WysokośćBryły / Math.Tan(Math.PI \* KątNachylenia / 180)));

Oś\_Duża = PromieńBryły \* 2;

Oś\_Mała = PromieńBryły / 2;

using (Pen Pióro = new Pen(Kolor\_Linii, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

//wykreślenie podłogi i sufitu Walca

Rysownica.DrawEllipse(Pióro, XsP - Oś\_Duża / 2, YsP - Oś\_Mała / 2, Oś\_Duża,

Oś\_Mała);

Rysownica.DrawEllipse(Pióro, XsS - Oś\_Duża / 2, YsS - Oś\_Mała / 2, Oś\_Duża,

Oś\_Mała);

//wykreślenie prążków na ścianie bocznej

using (Pen PióroPrążków = new Pen(Kolor\_Linii, 0.5f \* Grubość\_Linii))

{

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

{

Rysownica.DrawLine(PióroPrążków,

WielokątPodłogi[i], WielokątSufitu[i]);

}

}

//wykreślenie krawędzi bocznych walca

//lewa

Rysownica.DrawLine(Pióro, XsP - Oś\_Duża / 2, YsP, XsS - Oś\_Duża / 2, YsS);

//prawa

Rysownica.DrawLine(Pióro, XsP + Oś\_Duża / 2, YsP, XsS + Oś\_Duża / 2, YsS);

Widoczny = true;

}

## Klasa Stożek

Klasa podlega tym samym zmianom jak te przedstawione w klasie Walec

## Klasa Stożek Pochyły

Klasa praktycznie bez zmian względem ćwiczeniowej.

## Klasa Stożek Dwustronny

Klasa korzystająca z polimorfizmu w dużym stopniu

public class StożekDwustronny : Stożek

{

int XsD, YsD;

public StożekDwustronny(int PromieńBryły, int WysokośćBryły, int StopieńWielokąta

, int XsP, int YsP, bool KierunekObrotu

, Color KolorLinii, DashStyle StylLinii, float GrubośćLinii)

: base(PromieńBryły, WysokośćBryły, StopieńWielokąta,

XsP, YsP, KierunekObrotu, KolorLinii, StylLinii, GrubośćLinii)

{

RodzajBryły = TypyBryły.BG\_StożekDwustronny;

XsD = XsP;

YsD = YsP + WysokośćBryły;

this.ObjętośćBryły = (float)(Math.PI \* PromieńBryły \*

PromieńBryły \* WysokośćBryły / 3)\*2;

this.PowierzchniaBryły = (float)(Math.PI \* PromieńBryły \*

PromieńBryły + Math.PI \* PromieńBryły \*

Math.Sqrt(WysokośćBryły \* WysokośćBryły + PromieńBryły \* PromieńBryły)\*2);

}//od konstruktora

public override void Wykreśl(Graphics Rysownica)

{

XsS = XsP;

YsS = YsP - WysokośćBryły;

Oś\_Duża = 2 \* PromieńBryły;

Oś\_Mała = PromieńBryły / 2;

XsD = XsP;

YsD = YsP + WysokośćBryły;

using (Pen Pióro = new Pen(Kolor\_Linii, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

//wykreślenie Podłogi

Rysownica.DrawEllipse(Pióro, XsP - Oś\_Duża / 2, YsP - Oś\_Mała / 2, Oś\_Duża, Oś\_Mała);

//wykreślenie prążków

using (Pen PióroPrążków = new Pen(Kolor\_Linii, Grubość\_Linii / 4))

{

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

{

Rysownica.DrawLine(PióroPrążków, WielokątPodłogi[i], new Point(XsS, YsS));

Rysownica.DrawLine(PióroPrążków, WielokątPodłogi[i], new Point(XsD, YsD));

}

}

//wykreślenie krawędzi

Rysownica.DrawLine(Pióro, XsP - Oś\_Duża / 2, YsP, XsS, YsS);

Rysownica.DrawLine(Pióro, XsP + Oś\_Duża / 2, YsP, XsS, YsS);

Rysownica.DrawLine(Pióro, XsP - Oś\_Duża / 2, YsP, XsD, YsD);

Rysownica.DrawLine(Pióro, XsP + Oś\_Duża / 2, YsP, XsD, YsD);

Widoczny = true;

}

}//wykreśl

public override void Wymaż(Control Kontrolka, Graphics Rysownica)

{

if (Widoczny)

{

using (Pen Pióro = new Pen(Kontrolka.BackColor, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

//wykreślenie Podłogi

Rysownica.DrawEllipse(Pióro, XsP - Oś\_Duża / 2, YsP - Oś\_Mała / 2, Oś\_Duża, Oś\_Mała);

//wykreślenie prążków

using (Pen PióroPrążków = new Pen(Kontrolka.BackColor, Grubość\_Linii / 4))

{

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

{

Rysownica.DrawLine(PióroPrążków, WielokątPodłogi[i], new Point(XsS, YsS));

Rysownica.DrawLine(PióroPrążków, WielokątPodłogi[i], new Point(XsD, YsD));

}

}

//wykreślenie krawędzi

Rysownica.DrawLine(Pióro, XsP - Oś\_Duża / 2, YsP, XsS, YsS);

Rysownica.DrawLine(Pióro, XsP + Oś\_Duża / 2, YsP, XsS, YsS);

Rysownica.DrawLine(Pióro, XsP - Oś\_Duża / 2, YsP, XsD, YsD);

Rysownica.DrawLine(Pióro, XsP + Oś\_Duża / 2, YsP, XsD, YsD);

Widoczny = false;

}

}

}

public override void OdświeżAtrybuty()

{

ObjętośćBryły = (float)(Math.PI \* PromieńBryły \*

PromieńBryły \* WysokośćBryły / 3) \* 2;

PowierzchniaBryły = (float)(Math.PI \* PromieńBryły \*

PromieńBryły + Math.PI \* PromieńBryły \*

Math.Sqrt(WysokośćBryły \* WysokośćBryły + PromieńBryły \* PromieńBryły) \* 2);

}

## Klasa Kula

Klasa be zmian innych niż te dotyczące wszystkich brył.

## Klasa Graniastosłup

Klasa be zmian innych niż te dotyczące wszystkich brył.

## Klasa GraniastosłupPochyły

Klasa wprowadzająca podobne zmiany do tych jakie wprowadza Stożek Pochyły względem Stożka, nie posiada niestety wzorów na objętość i powierzchnię.

public class GraniastosłupPochyły : Graniastosłup

{

public GraniastosłupPochyły(int PromieńBryły, int StopieńWielokąta,

int WysokośćBryły, int XsP, int YsP, float KątNachylenia, bool KierunekObrotu,

Color KolorLinii, DashStyle StylLinii, float GrubośćLinii)

: base(PromieńBryły, StopieńWielokąta, WysokośćBryły, XsP, YsP, KierunekObrotu,

KolorLinii, StylLinii, GrubośćLinii)

{

this.KątNachylenia = KątNachylenia;

RodzajBryły = TypyBryły.BG\_GraniastosłupPochyły;

XsS = (int)(XsP + (WysokośćBryły / Math.Tan(Math.PI \* KątNachylenia / 180)));

YsS = YsP - WysokośćBryły;

Oś\_Duża = 2 \* PromieńBryły;

Oś\_Mała = PromieńBryły / 2;

//wyznaczenie punktów do wykreślenia graniastosłupa

WielokątPodłogi = new Point[StopieńWielokąta + 1];

WielokątSufitu = new Point[StopieńWielokąta + 1];

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

{

WielokątPodłogi[i] = new Point();

WielokątSufitu[i] = new Point();

WielokątPodłogi[i].X = (int)(XsP + Oś\_Duża / 2 \*

Math.Cos(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

//Przejście z Rad na Degree!!!

WielokątPodłogi[i].Y = (int)(YsP + Oś\_Mała / 2 \*

Math.Sin(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

//sufit

WielokątSufitu[i].X = (int)(XsS + Oś\_Duża / 2 \*

Math.Cos(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

WielokątSufitu[i].Y = (int)(YsS + Oś\_Mała / 2 \*

Math.Sin(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

}

}

public override void Wykreśl(Graphics Rysownica)

{

//wyznaczenie środka sufitu

XsS = (int)(XsP + (WysokośćBryły / Math.Tan(Math.PI \* KątNachylenia / 180)));

YsS = YsP - WysokośćBryły;

//Wyznaczenie osi elipsy w którą wpisane będzie podłoga i sufit

Oś\_Duża = 2 \* PromieńBryły;

Oś\_Mała = PromieńBryły / 2;

using (Pen Pióro = new Pen(Kolor\_Linii, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

Rysownica.DrawLines(Pióro, WielokątPodłogi);

Rysownica.DrawLines(Pióro, WielokątSufitu);

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

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], WielokątSufitu[i]);

Widoczny = true;

}

}

## Klasa PolihedronPłaski

Klasa która zamienia prostokątne ściany graniastosłupa na trójkątne

public class PolihedronPłaski : Graniastosłup

{

public PolihedronPłaski(int PromieńBryły, int StopieńWielokąta,

int WysokośćBryły, int XsP, int YsP, bool KierunekObrotu,

Color KolorLinii, DashStyle StylLinii, float GrubośćLinii)

: base(PromieńBryły, StopieńWielokąta, WysokośćBryły, XsP, YsP, KierunekObrotu,

KolorLinii, StylLinii, GrubośćLinii)

{

RodzajBryły = TypyBryły.BG\_PoilhedronPłaski;

XsS = XsP;

YsS = YsP - WysokośćBryły;

Oś\_Duża = 2 \* PromieńBryły;

Oś\_Mała = PromieńBryły / 2;

//wyznaczenie punktów do wykreślenia graniastosłupa

WielokątPodłogi = new Point[StopieńWielokąta + 1];

WielokątSufitu = new Point[StopieńWielokąta + 1];

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

{

WielokątPodłogi[i] = new Point();

WielokątSufitu[i] = new Point();

WielokątPodłogi[i].X = (int)(XsP + Oś\_Duża / 2 \*

Math.Cos(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

WielokątPodłogi[i].Y = (int)(YsP + Oś\_Mała / 2 \*

Math.Sin(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

//sufit

WielokątSufitu[i].X = (int)(XsS + Oś\_Duża / 2 \*

Math.Cos(Math.PI \* (KątPołożeniaWierzchołka + KątMiędzyWierzchołkami / 2 + i \* KątMiędzyWierzchołkami) / 180f));

WielokątSufitu[i].Y = (int)(YsS + Oś\_Mała / 2 \*

Math.Sin(Math.PI \* (KątPołożeniaWierzchołka + KątMiędzyWierzchołkami / 2 + i \* KątMiędzyWierzchołkami) / 180f));

}

}

public override void Wykreśl(Graphics Rysownica)

{

XsS = XsP;

YsS = YsP - WysokośćBryły;

Oś\_Duża = 2 \* PromieńBryły;

Oś\_Mała = PromieńBryły / 2;

using (Pen Pióro = new Pen(Kolor\_Linii, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

Rysownica.DrawLines(Pióro, WielokątPodłogi);

Rysownica.DrawLines(Pióro, WielokątSufitu);

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

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], WielokątSufitu[i]);

}

for (int i = 1; i <= StopieńWielokąta; i++)

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], WielokątSufitu[i-1]);

}

Rysownica.DrawLine(Pióro, WielokątPodłogi[0], WielokątSufitu[StopieńWielokąta]);

Widoczny = true;

}

}

public override void Wymaż(Control Kontrolka, Graphics Rysownica)

{

using (Pen Pióro = new Pen(Kontrolka.BackColor, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

Rysownica.DrawLines(Pióro, WielokątPodłogi);

Rysownica.DrawLines(Pióro, WielokątSufitu);

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

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], WielokątSufitu[i]);

}

for (int i = 1; i <= StopieńWielokąta; i++)

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], WielokątSufitu[i - 1]);

}

Rysownica.DrawLine(Pióro, WielokątPodłogi[0], WielokątSufitu[StopieńWielokąta]);

Widoczny = false;

}

}

public override void Obróć\_i\_Wykreśl(Control Kontrolka, Graphics Rysownica, float KątObrotu)

{

if (Widoczny)

{

Wymaż(Kontrolka, Rysownica);

if (KierunekObrotu)

KątPołożeniaWierzchołka -= KątObrotu;

else

KątPołożeniaWierzchołka += KątObrotu;

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

{

WielokątPodłogi[i].X = (int)(XsP + Oś\_Duża / 2 \* Math.Cos(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

WielokątPodłogi[i].Y = (int)(YsP + Oś\_Mała / 2 \* Math.Sin(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

WielokątSufitu[i].X = (int)(XsS + Oś\_Duża / 2 \* Math.Cos(Math.PI \* (KątPołożeniaWierzchołka + KątMiędzyWierzchołkami / 2 + i \* KątMiędzyWierzchołkami) / 180f));

WielokątSufitu[i].Y = (int)(YsS + Oś\_Mała / 2 \* Math.Sin(Math.PI \* (KątPołożeniaWierzchołka + KątMiędzyWierzchołkami / 2 + i \* KątMiędzyWierzchołkami) / 180f));

}

Wykreśl(Rysownica);

}

}

## Klasa Polihedron

Klasa która zmienia płaski sufit i podstawę na ostrosłupy.

public class Polihedron: PolihedronPłaski

{

int XsD, YsD;

int XsT, YsT;

public Polihedron(int PromieńBryły, int StopieńWielokąta,

int WysokośćBryły, int XsP, int YsP, bool KierunekObrotu,

Color KolorLinii, DashStyle StylLinii, float GrubośćLinii)

: base(PromieńBryły, StopieńWielokąta, WysokośćBryły, XsP, YsP, KierunekObrotu,

KolorLinii, StylLinii, GrubośćLinii)

{

XsT = XsP;

YsT = YsP - WysokośćBryły \* 2;

RodzajBryły = TypyBryły.BG\_Polihedron;

XsS = XsP;

YsS = YsP - WysokośćBryły;

XsD = XsP;

YsD = YsP + WysokośćBryły;

Oś\_Duża = 2 \* PromieńBryły;

Oś\_Mała = PromieńBryły / 2;

//wyznaczenie punktów do wykreślenia graniastosłupa

WielokątPodłogi = new Point[StopieńWielokąta + 1];

WielokątSufitu = new Point[StopieńWielokąta + 1];

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

{

WielokątPodłogi[i] = new Point();

WielokątSufitu[i] = new Point();

WielokątPodłogi[i].X = (int)(XsP + Oś\_Duża / 2 \*

Math.Cos(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

//Przejście z Rad na Degree!!!

WielokątPodłogi[i].Y = (int)(YsP + Oś\_Mała / 2 \*

Math.Sin(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

//sufit

WielokątSufitu[i].X = (int)(XsS + Oś\_Duża / 2 \*

Math.Cos(Math.PI \* (KątPołożeniaWierzchołka+ KątMiędzyWierzchołkami/2 + i \* KątMiędzyWierzchołkami) / 180f));

WielokątSufitu[i].Y = (int)(YsS + Oś\_Mała / 2 \*

Math.Sin(Math.PI \* (KątPołożeniaWierzchołka+ KątMiędzyWierzchołkami/2 + i \* KątMiędzyWierzchołkami) / 180f));

}

}

public override void Wykreśl(Graphics Rysownica)

{

XsT = XsP;

YsT = YsP - WysokośćBryły \* 2;

XsS = XsP;

YsS = YsP - WysokośćBryły;

XsD = XsP;

YsD = YsP + WysokośćBryły;

Oś\_Duża = 2 \* PromieńBryły;

Oś\_Mała = PromieńBryły / 2;

using (Pen Pióro = new Pen(Kolor\_Linii, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

Rysownica.DrawLines(Pióro, WielokątPodłogi);

Rysownica.DrawLines(Pióro, WielokątSufitu);

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

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], WielokątSufitu[i]);

Rysownica.DrawLine(Pióro, WielokątSufitu[i], new Point(XsT, YsT));

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], new Point(XsD, YsD));

}

for (int i = 1; i <= StopieńWielokąta; i++)

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], WielokątSufitu[i - 1]);

}

Rysownica.DrawLine(Pióro, WielokątPodłogi[0], WielokątSufitu[StopieńWielokąta]);

Widoczny = true;

}

}

public override void Wymaż(Control Kontrolka, Graphics Rysownica)

{

using (Pen Pióro = new Pen(Kontrolka.BackColor, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

Rysownica.DrawLines(Pióro, WielokątPodłogi);

Rysownica.DrawLines(Pióro, WielokątSufitu);

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

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], WielokątSufitu[i]);

Rysownica.DrawLine(Pióro, WielokątSufitu[i], new Point(XsT, YsT));

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], new Point(XsD, YsD));

}

for (int i = 1; i <= StopieńWielokąta; i++)

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], WielokątSufitu[i - 1]);

}

Rysownica.DrawLine(Pióro, WielokątPodłogi[0], WielokątSufitu[StopieńWielokąta]);

Widoczny = false;

}

}

## Klasa Ostrosłup

Klasa dodaję metodę która powinna być zaimplementowana dużo wcześniej, służy ona do wyznaczania wierzchołków wielokąta wpisanego w okrąg.

public virtual Point[] KalkulujWierzchołkiPodłogi()

{

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

{

WielokątPodłogi[i].X = (int)(XsP + Oś\_Duża / 2 \* Math.Cos

(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

WielokątPodłogi[i].Y = (int)(YsP + Oś\_Mała / 2 \* Math.Sin

(Math.PI \* (KątPołożeniaWierzchołka + i \* KątMiędzyWierzchołkami) / 180f));

}

return WielokątPodłogi;

}

## Klasa Ostrosłup Pochyły

Klasa odtwórcza względem stożka pochyłego. public class OstrosłupPochyły:Ostrosłup

{

public OstrosłupPochyły(int PromieńBryły, int StopieńWielokąta,

int WysokośćBryły, int XsP, int YsP,float KątNachylenia, bool KierunekObrotu,

Color KolorLinii, DashStyle StylLinii, float GrubośćLinii)

: base(PromieńBryły, StopieńWielokąta,WysokośćBryły,XsP,YsP, KierunekObrotu,

KolorLinii, StylLinii, GrubośćLinii)

{

RodzajBryły = TypyBryły.BG\_OstrosłupPochyły;

this.KątNachylenia = KątNachylenia;

XsS = (int)(XsP + (WysokośćBryły / Math.Tan(Math.PI \* KątNachylenia / 180)));

YsS = YsP - WysokośćBryły;

}//konstruktor

public override void Wykreśl(Graphics Rysownica)

{

XsS = (int)(XsP + (WysokośćBryły / Math.Tan(Math.PI \* KątNachylenia / 180)));

RodzajBryły = TypyBryły.BG\_GraniastosłupPochyły;

YsS = YsP - WysokośćBryły;

Oś\_Duża = PromieńBryły \* 2;

Oś\_Mała = PromieńBryły / 2;

using (Pen Pióro = new Pen(Kolor\_Linii, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

Rysownica.DrawLines(Pióro, WielokątPodłogi);

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

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], new Point(XsS, YsS));

Widoczny = true;

}

}

}

## Klasa Ostrosłup Dwustronny

Klasa odtwórcza względem Stożka Dwustronnego.

public class OstrosłupDwustronny:Ostrosłup

{

int XsD, YsD;

public OstrosłupDwustronny(int PromieńBryły, int StopieńWielokąta,

int WysokośćBryły, int XsP, int YsP, bool KierunekObrotu,

Color KolorLinii, DashStyle StylLinii, float GrubośćLinii)

: base(PromieńBryły, StopieńWielokąta, WysokośćBryły, XsP, YsP, KierunekObrotu,

KolorLinii, StylLinii, GrubośćLinii)

{

RodzajBryły = TypyBryły.BG\_OstrosłupDwustronny;

XsD = XsP;

YsD = YsP + WysokośćBryły;

YsS = YsP - WysokośćBryły;

double A = PromieńBryły \* (Math.Cos(Math.PI / 2 \* StopieńWielokąta));

double BocznaWysokość = Math.Sqrt((PromieńBryły \* (Math.Cos(Math.PI / 2 \* StopieńWielokąta))) \*

(PromieńBryły \* (Math.Cos(Math.PI / 2 \* StopieńWielokąta))) + WysokośćBryły \* WysokośćBryły);

double Bok = (Math.Sin(Math.PI / 2 \* StopieńWielokąta)) \* 2 \* PromieńBryły;

double Obwód = Bok \* StopieńWielokąta;

this.ObjętośćBryły = (float)(PromieńBryły \* StopieńWielokąta

\* PromieńBryły \* (Math.Sin(Math.PI \* 2 / StopieńWielokąta)) \* WysokośćBryły / 3)\*2;

this.PowierzchniaBryły = (float)((Bok \* A / 2) + (Obwód \* BocznaWysokość) / 2)\*2;

}//konstruktor

public override void Wykreśl(Graphics Rysownica)

{

YsD = YsP + WysokośćBryły;

YsS = YsP - WysokośćBryły;

XsS = XsP;

YsS = YsP - WysokośćBryły;

Oś\_Duża = PromieńBryły \* 2;

Oś\_Mała = PromieńBryły / 2;

using (Pen Pióro = new Pen(Kolor\_Linii, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

Rysownica.DrawLines(Pióro, WielokątPodłogi);

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

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], new Point(XsS, YsS));

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], new Point(XsD, YsD));

}

Widoczny = true;

}

}

public override void Wymaż(Control Kontrolka, Graphics Rysownica)

{

using (Pen Pióro = new Pen(Kontrolka.BackColor, Grubość\_Linii))

{

Pióro.DashStyle = Styl\_Linii;

Rysownica.DrawLines(Pióro, WielokątPodłogi);

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

{

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], new Point(XsS, YsS));

Rysownica.DrawLine(Pióro, WielokątPodłogi[i], new Point(XsD, YsD));

}

Widoczny = false;

}

}

public override void OdświeżAtrybuty()

{

double A = PromieńBryły \* (Math.Cos(Math.PI / 2 \* StopieńWielokąta));

double BocznaWysokość = Math.Sqrt((PromieńBryły \* (Math.Cos(Math.PI / 2 \* StopieńWielokąta))) \*

(PromieńBryły \* (Math.Cos(Math.PI / 2 \* StopieńWielokąta))) + WysokośćBryły \* WysokośćBryły);

double Bok = (Math.Sin(Math.PI / 2 \* StopieńWielokąta)) \* 2 \* PromieńBryły;

double Obwód = Bok \* StopieńWielokąta;

this.ObjętośćBryły = (float)(PromieńBryły \* StopieńWielokąta

\* PromieńBryły \* (Math.Sin(Math.PI \* 2 / StopieńWielokąta)) \* WysokośćBryły / 3) \* 2;

this.PowierzchniaBryły = (float)((Bok \* A / 2) + (Obwód \* BocznaWysokość) / 2) \* 2;

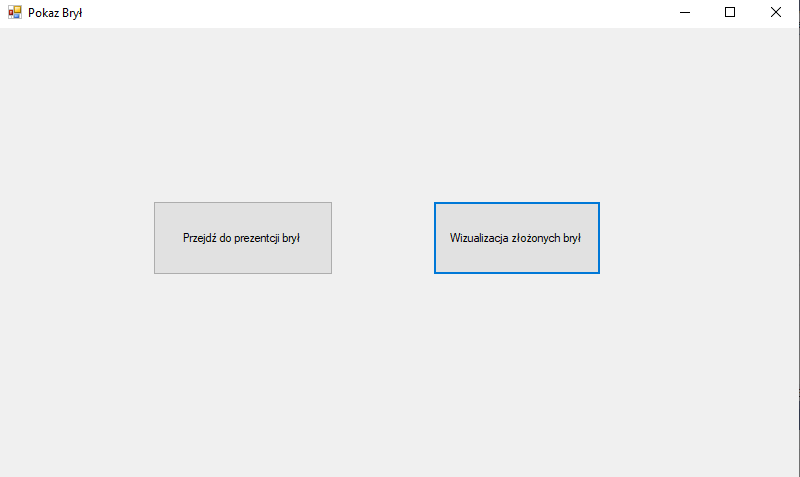
}

# Dokumentacja Użytkownika

## Formularz PokazInitialForm

Formularz po witający pozwala na poruszanie się między dwoma funkcjonalnymi formularzami.

Został on zaimplementowany właściwie niepotrzebnie ponieważ formularz dla brył regularnych ma wszystkie funkcjonalności stworzone dla formularza brył złożonych.



## 

## Formularz Pokaz Brył Regularnych

### Kreślenie brył

Graphical user interface, application, Word

Description automatically generated



Aby uruchomić pokaz brył należy wybrać atrybuty bryły do wykreślenia Menu formularza pozwala na manipulacje Kolorami, Stylem linii którą są kreślone figury i jej Grubością. Kontrolki z grupy Tworzenie Bryły z lewej strony ekranu pozwalają na Wizualizację Atrybutów Graficznych, Wybór typu kreślonej bryły, i jej parametrów jak Promień. Po ustaleniu upragnionych atrybutów bryły, aby wykreślić bryłę należy kliknąć w powierzchnię graficzną, a następnie kliknąć przycisk Wykreśl bryłę.



### Pokaz brył

Text, whiteboard

Description automatically generated

Aby uruchomić pokaz brył należy kliknąć w przycisk Włącz Pokaz, przycisk ten uruchamia pokaz brył Text, whiteboard

Description automatically generated



W trakcie pokazu zablokowana jest możliwość tworzenia nowych brył, ale można dokonać zmian w stworzonych już bryłach za pomocą menu z prawej strony. Prezentowaną bryłę można się przełączać za pomocą przycisków Poprzednia i Następna, lub poprzez zaznaczenie kontrolki Automatyczny w Grupie Pokaz Brył Geometrycznych przez co figury będą automatycznie przełączane. Grupa Kontrolek Kryterium Pokazu Brył pozwala na uporządkowanie brył według wybranej wartości(nie wszystkie figury mają prawidłowo zaimplementowane obliczenia objętości i pola).

### Zapisywanie i poruszanie się między formularzami

Graphical user interface, text, application

Description automatically generatedProgram posiada częściowo zaimplementowaną możliwość zapisu obrazu z powierzchni graficznej(tworzy plik BMP z artefaktami). Funkcjonalność powrót pozwala na przejście do początkowego formularza. Natomiast funkcjonalność wyjście zamyka program.

# Samoocena

## Samoocena Laboratorium

Samoocena: 5

Wszystkie Figury i kontrolki zostały zaimplementowane i działają prawidłowo.

## Samoocena Projekt

Samoocena: 5

Projekt posiada funkcjonalności, które były wskazane jako opcjonalne. A wszystkie zadane właściwości są funkcjonalne. W projekcie jest wiele miejsc na wprowadzenie usprawnień ale dotyczą one głównie jego rozwiniętych funkcjonalności np. Klasa bryły mogłaby mieć znacznie mniej linijek kodu gdyby wprowadzić kilka nowych metod.