Міністерство освіти і науки України

Національний технічний університет України „КПІ”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки

інформації та управління

**Протокол**

з основ Web-програмування № 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Виконав**  **студент** |  | *ІП-63 Карпа Маркіян Володимирович* |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |
|  |  |  |  |
| **Номер залікової книжки та курс** |  | 6314, другий курс |  |

Київ 2018

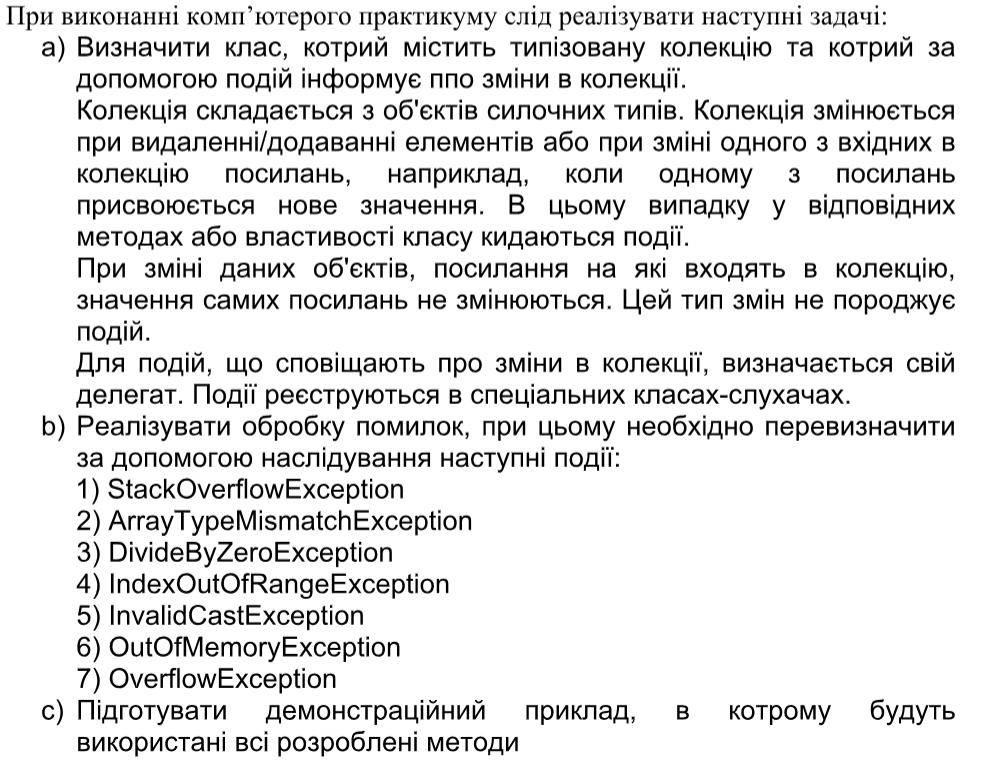
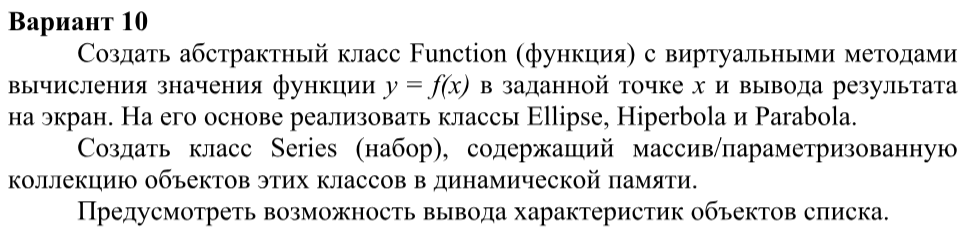
ЗМІСТ

[1 Постановка задачі 3](#_Toc515634943)

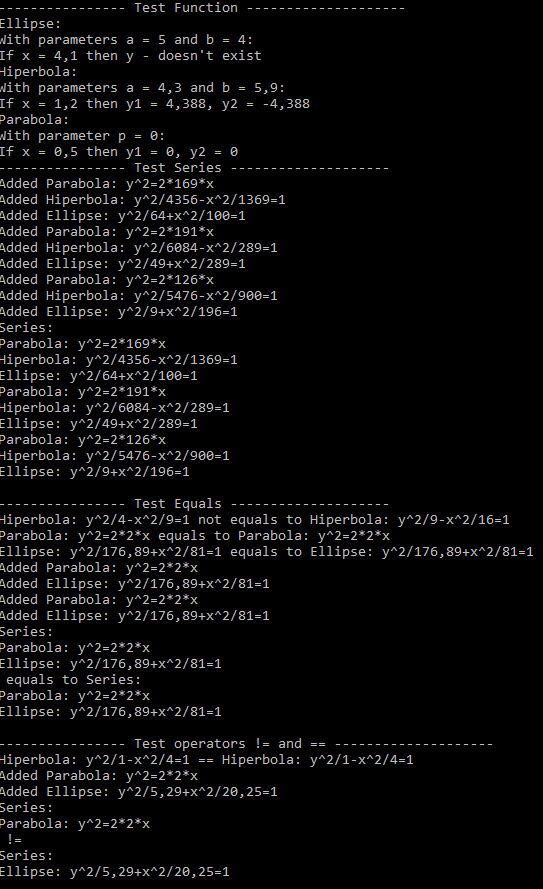
[2 Демонстрація роботи програми 4](#_Toc515634944)

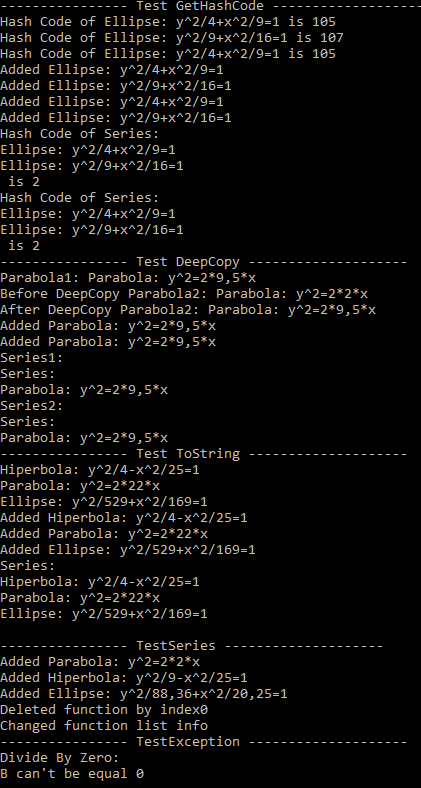
[3 Тексти програмного коду 6](#_Toc515634945)

# Постановка задачі



# Демонстрація роботи програми





# Тексти програмного коду

Файл ArrayTypeMismatchException.cs

using System;

namespace Lab2

{

class ArrayTypeMismatchException : Exception

{

private const string DEFAULT\_MESSAGE = "Array Type Mismatch";

public ArrayTypeMismatchException(string message) : base(DEFAULT\_MESSAGE + ":\n" + message)

{

}

}

}

Файл DivideByZeroException.cs

using System;

namespace Lab2

{

class DivideByZeroException : Exception

{

private const string DEFAULT\_MESSAGE = "Divide By Zero";

public DivideByZeroException(string message) : base(DEFAULT\_MESSAGE + ":\n" + message)

{

}

}

}

Файл Ellipse.cs

using System;

namespace Lab2

{

class Ellipse : Function

{

private double A { get; set; }

private double \_b;

public double B {

get {

return \_b;

}

set {

if (value == 0)

throw new DivideByZeroException("B can't be equal 0");

else

\_b = value;

}

}

public Ellipse(double a, double b)

{

A = a;

B = b;

}

public override double[] Calculate(double x)

{

double ySquared = Math.Pow(A, 2) \* (1 - Math.Pow(x, 2) / Math.Pow(B, 2));

if(ySquared < 0)

{

return null;

}

double y = Math.Sqrt(ySquared);

return new double[] { y, -y };

}

public override void Print(double[] y, double x)

{

Console.WriteLine(String.Format("Ellipse:\nWith parameters a = {0:0.###} and b = {1:0.###}:", A, B));

if(y == null)

Console.WriteLine(String.Format("If x = {0:0.###} then y - doesn't exist", x));

else

Console.WriteLine(String.Format("If x = {0:0.###} then y1 = {1:0.###}, y2 = {2:0.###}", x, y[0], y[1]));

}

// Override Equals

public override bool Equals(object obj)

{

if(obj == null || GetType() != obj.GetType())

{

return false;

}

Ellipse ellipse = (Ellipse)obj;

return A == ellipse.A && B == ellipse.B;

}

// Override GetHashCode

public override int GetHashCode()

{

return (int)(A + B) % 300 + 100;

}

// Override ToString

public override string ToString()

{

return String.Format("Ellipse: y^2/{0}+x^2/{1}=1", Math.Pow(A, 2), Math.Pow(B, 2));

}

// Method returns copy of an object

public override object DeepCopy()

{

return new Ellipse(A, B);

}

}

}

Файл EventArgs.cs

using System;

namespace Lab2

{

class EventArgs

{

public string Text { get; set; }

public EventArgs(string text)

{

Text = text;

}

}

}

Файл Function.cs

using System;

namespace Lab2

{

abstract class Function

{

public abstract double[] Calculate(double x);

public abstract void Print(double[] y, double x);

public static bool operator ==(Function function1, Function function2)

{

return function1.Equals(function2);

}

public static bool operator !=(Function function1, Function function2)

{

return !function1.Equals(function2);

}

public abstract object DeepCopy();

}

}

Файл Hiperbola.cs

using System;

namespace Lab2

{

class Hiperbola : Function

{

private double A { get; set; }

private double \_b;

public double B {

get {

return \_b;

}

set {

if (value == 0)

throw new DivideByZeroException("B can't be equal 0");

else

\_b = value;

}

}

public Hiperbola(double a, double b)

{

A = a;

B = b;

}

public override double[] Calculate(double x)

{

double ySquared = Math.Pow(A, 2) \* (1 + Math.Pow(x, 2) / Math.Pow(B, 2));

if (ySquared < 0)

{

return null;

}

double y = Math.Sqrt(ySquared);

return new double[] { y, -y };

}

public override void Print(double[] y, double x)

{

Console.WriteLine(String.Format("Hiperbola:\nWith parameters a = {0:0.###} and b = {1:0.###}:", A, B));

if (y == null)

Console.WriteLine(String.Format("If x = {0:0.###} then y - doesn't exist", x));

else

Console.WriteLine(String.Format("If x = {0:0.###} then y1 = {1:0.###}, y2 = {2:0.###}", x, y[0], y[1]));

}

// Override Equals

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

{

return false;

}

Hiperbola ellipse = (Hiperbola)obj;

return A == ellipse.A && B == ellipse.B;

}

// Override GetHashCode

public override int GetHashCode()

{

return (int)(A + B) % 300 + 100;

}

// Override ToString

public override string ToString()

{

return String.Format("Hiperbola: y^2/{0}-x^2/{1}=1", Math.Pow(A, 2), Math.Pow(B, 2));

}

// Method returns copy of an object

public override object DeepCopy()

{

return new Hiperbola(A, B);

}

}

}

Файл IndexOutOfRangeException.cs

using System;

namespace Lab2

{

class IndexOutOfRangeException : Exception

{

private const string DEFAULT\_MESSAGE = "Index Out Of Range";

public IndexOutOfRangeException(string message) : base(DEFAULT\_MESSAGE + ":\n" + message)

{

}

}

}

Файл InvalidCastException.cs

using System;

namespace Lab2

{

class InvalidCastException : Exception

{

private const string DEFAULT\_MESSAGE = "Invalid Cast";

public InvalidCastException(string message) : base(DEFAULT\_MESSAGE + ":\n" + message)

{

}

}

}

Файл OutOfMemoryException.cs

using System;

namespace Lab2

{

class OutOfMemoryException : Exception

{

private const string DEFAULT\_MESSAGE = "Out Of Memory";

public OutOfMemoryException(string message) : base(DEFAULT\_MESSAGE + ":\n" + message)

{

}

}

}

Файл OverflowException.cs

using System;

namespace Lab2

{

class OverflowException : Exception

{

private const string DEFAULT\_MESSAGE = "Overflow";

public OverflowException(string message) : base(DEFAULT\_MESSAGE + ":\n" + message)

{

}

}

}

Файл Parabola.cs

using System;

namespace Lab2

{

class Parabola : Function

{

private double P { get; set; }

public Parabola(double p)

{

P = p;

}

public override double[] Calculate(double x)

{

double ySquared = 2 \* P \* x;

if (ySquared < 0)

{

return null;

}

double y = Math.Sqrt(ySquared);

return new double[] { y, -y };

}

public override void Print(double[] y, double x)

{

Console.WriteLine(String.Format("Parabola:\nWith parameter p = {0:0.###}:", P));

if (y == null)

Console.WriteLine(String.Format("If x = {0:0.###} then y - doesn't exist", x));

else

Console.WriteLine(String.Format("If x = {0:0.###} then y1 = {1:0.###}, y2 = {2:0.###}", x, y[0], y[1]));

}

// Override Equals

public override bool Equals(object obj)

{

if (obj == null || GetType() != obj.GetType())

{

return false;

}

Parabola ellipse = (Parabola)obj;

return P == ellipse.P;

}

// Override GetHashCode

public override int GetHashCode()

{

return (int)(P) % 300 + 100;

}

// Override ToString

public override string ToString()

{

return String.Format("Parabola: y^2=2\*{0}\*x", P);

}

// Method returns copy of an object

public override object DeepCopy()

{

return new Parabola(P);

}

}

}

Файл Series.cs

using System;

using System.Collections.Generic;

namespace Lab2

{

class Series

{

private List<Function> \_functionList;

public List<Function> FunctionList {

get {

return \_functionList;

}

set {

\_functionList.Clear();

Changed(this, new EventArgs("Changed function list info"));

for (int i = 0; i < value.Count; i++)

{

\_functionList.Add(value[i]);

}

}

}

// Events

private delegate void SeriesEventHandler(object sender, EventArgs e);

private event SeriesEventHandler Added;

private event SeriesEventHandler Deleted;

private event SeriesEventHandler Changed;

// Constructors

public Series()

{

\_functionList = new List<Function>();

InitializeEvents();

}

public Series(params Function[] functions)

{

\_functionList = new List<Function>();

InitializeEvents();

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

{

Add(functions[i]);

}

}

public Series(List<Function> functionList)

{

\_functionList = functionList;

InitializeEvents();

}

// Event Methods

private void InitializeEvents()

{

Added += OnEventTriggered;

Deleted += OnEventTriggered;

Changed += OnEventTriggered;

}

public void OnEventTriggered(object sender, EventArgs e)

{

Console.WriteLine(e.Text);

}

// Basic Methods to work with collection

public void Add(Function function)

{

\_functionList.Add(function);

Added?.Invoke(this, new EventArgs("Added " + function.ToString()));

}

public void Remove(object obj)

{

\_functionList.Remove((Function)obj);

Deleted(this, new EventArgs("Deleted " + ((Function)obj).ToString()));

}

public void RemoveAt(int index)

{

\_functionList.RemoveAt(index);

Deleted(this, new EventArgs("Deleted function by index" + index.ToString()));

}

// Override ToString

public override string ToString()

{

string result = "Series:\n";

foreach(Function function in \_functionList)

{

result += function.ToString() + '\n';

}

return result;

}

// Override Equals

public override bool Equals(object obj)

{

List<Function> anotherFunctionList = ((Series)obj).FunctionList;

if(anotherFunctionList.Count != \_functionList.Count)

{

return false;

}

for (int i = 0; i < \_functionList.Count; i++)

{

if (!\_functionList[i].Equals(anotherFunctionList[i]))

{

return false;

}

}

return true;

}

// Override GetHashCode

public override int GetHashCode()

{

int hashCode = 0;

foreach(var element in \_functionList)

{

hashCode += element.GetHashCode();

}

hashCode /= 100;

return hashCode;

}

// Method returns copy of an object

public object DeepCopy()

{

Series newSeries = new Series();

foreach (var element in \_functionList)

{

newSeries.Add(element);

}

return newSeries;

}

// Override operators == and !=

public static bool operator ==(Series series1, Series series2)

{

return series1.Equals(series2);

}

public static bool operator !=(Series series1, Series series2)

{

return !series1.Equals(series2);

}

}

}

Файл Source.cs

using System;

namespace Lab2

{

class Source

{

static public void Main(String[] args)

{

Test test = new Test();

test.ShowTests();

Console.ReadLine();

}

}

}

Файл StackOverflowException.cs

using System;

namespace Lab2

{

class StackOverflowException : Exception

{

private const string DEFAULT\_MESSAGE = "Stack Overflow";

public StackOverflowException(string message) : base(DEFAULT\_MESSAGE + ":\n" + message)

{

}

}

}

Файл TestFunctionClasses.cs

using System;

using System.Collections.Generic;

namespace Lab2

{

class Test

{

// Variant tasks

// How Function and it subclasses work

private void TestFunctionClasses()

{

Console.WriteLine("---------------- Test Function --------------------");

try

{

Function ellipse = new Ellipse(5, 4);

Function hiperbola = new Hiperbola(4.3, 5.9);

Function parabola = new Parabola(0);

ellipse.Print(ellipse.Calculate(4.1), 4.1);

hiperbola.Print(hiperbola.Calculate(1.2f), 1.2f);

parabola.Print(parabola.Calculate(0.5f), 0.5f);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

return;

}

}

// How Series class works

private void TestSeriesClass()

{

Console.WriteLine("---------------- Test Series --------------------");

Random random = new Random();

Series series = new Series();

for (int i = 1; i <= 3; i++)

{

Function ellipse = new Ellipse(random.Next(1, 10), random.Next(10, 20));

Function hiperbola = new Hiperbola(random.Next(1, 100), random.Next(5, 50));

Function parabola = new Parabola(random.Next(100, 200));

series.Add(parabola);

series.Add(hiperbola);

series.Add(ellipse);

}

Console.WriteLine(series);

}

// Basic tasks 1

// How Equals works

private void TestEquals()

{

Console.WriteLine("---------------- Test Equals --------------------");

void CheckIfEquals<T>(T t1, T t2)

{

string equalsMessage = "";

if (t1.Equals(t2))

equalsMessage = "equals";

else

equalsMessage = "not equals";

Console.WriteLine(String.Format("{0} {1} to {2}", t1.ToString(), equalsMessage, t2.ToString()));

}

Function hiperbola1 = new Hiperbola(2, 3);

Function hiperbola2 = new Hiperbola(3, 4);

CheckIfEquals(hiperbola1, hiperbola2);

Function parabola1 = new Parabola(2);

Function parabola2 = new Parabola(2);

CheckIfEquals(parabola1, parabola2);

Function ellipse1 = new Ellipse(13.3, 9);

Function ellipse2 = new Ellipse(13.3, 9);

CheckIfEquals(ellipse1, ellipse2);

Series series1 = new Series();

series1.Add(parabola1);

series1.Add(ellipse1);

Series series2 = new Series();

series2.Add(parabola2);

series2.Add(ellipse2);

CheckIfEquals(series1, series2);

}

// How operators != and == work

private void TestOperations()

{

Console.WriteLine("---------------- Test operators != and == --------------------");

Function hiperbola1 = new Hiperbola(1, 2);

Function hiperbola2 = new Hiperbola(1, 2);

if(hiperbola1 == hiperbola2)

Console.WriteLine(String.Format("{0} == {1}", hiperbola1, hiperbola2));

else

Console.WriteLine(String.Format("{0} != {1}", hiperbola1, hiperbola2));

Series series1 = new Series();

series1.Add(new Parabola(2));

Series series2 = new Series();

series2.Add(new Ellipse(2.3, 4.5));

if(series1 != series2)

{

Console.WriteLine(String.Format("{0} !=\n{1}", series1, series2));

}

}

// How GetHashCode work

private void TestGetHashCode()

{

Console.WriteLine("---------------- Test GetHashCode --------------------");

void ShowHashCode<T>(T t)

{

Console.WriteLine("Hash Code of {0} is {1}", t, t.GetHashCode());

}

Function ellipse1 = new Ellipse(2, 3);

Function ellipse2 = new Ellipse(3, 4);

Function ellipse3 = new Ellipse(2, 3);

ShowHashCode(ellipse1);

ShowHashCode(ellipse2);

ShowHashCode(ellipse3);

Series series1 = new Series();

Series series2 = new Series();

series1.Add(ellipse1);

series1.Add(ellipse2);

series2.Add(ellipse1);

series2.Add(ellipse2);

ShowHashCode(series1);

ShowHashCode(series2);

}

// How DeepCopy works

private void TestDeepCopy()

{

Console.WriteLine("---------------- Test DeepCopy --------------------");

Function parabola1 = new Parabola(9.5f);

Function parabola2 = new Parabola(2);

Console.WriteLine(String.Format("Parabola1: {0}", parabola1));

Console.WriteLine(String.Format("Before DeepCopy Parabola2: {0}", parabola2));

parabola2 = (Parabola)parabola1.DeepCopy();

Console.WriteLine(String.Format("After DeepCopy Parabola2: {0}", parabola2));

Series series1 = new Series();

series1.Add(parabola1);

Series series2 = (Series)series1.DeepCopy();

Console.Write(String.Format("Series1:\n{0}", series1));

Console.Write(String.Format("Series2:\n{0}", series2));

}

// How ToString workds

private void TestToString()

{

Console.WriteLine("---------------- Test ToString --------------------");

Function hiperbola = new Hiperbola(2, 5);

Function parabola = new Parabola(22);

Function ellipse = new Ellipse(23, 13);

Console.WriteLine(hiperbola);

Console.WriteLine(parabola);

Console.WriteLine(ellipse);

Series series = new Series(hiperbola, parabola, ellipse);

Console.WriteLine(series);

}

// Basic tasks 2

private void TestSeries()

{

Console.WriteLine("---------------- TestSeries --------------------");

Series series = new Series();

series.Add(new Parabola(2));

series.Add(new Hiperbola(3, 5));

series.Add(new Ellipse(9.4, 4.5));

series.RemoveAt(0);

series.FunctionList = new List<Function> { new Parabola(3), new Parabola(2.2) };

}

private void TestException()

{

Console.WriteLine("---------------- TestException --------------------");

try

{

Function hiperbola = new Hiperbola(2, 0);

}catch(Exception e)

{

Console.WriteLine(e.Message);

}

}

//Main method which shows all tests

public void ShowTests()

{

TestFunctionClasses();

TestSeriesClass();

TestEquals();

TestOperations();

TestGetHashCode();

TestDeepCopy();

TestToString();

TestSeries();

TestException();

}

}

}