Міністерство освіти та науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

ЗВІТ

до лабораторної роботи №1

з дисципліни «Основи Web-програмування»

Виконав: студент 2 курсу групи ІП-64 , ФІОТ

Царук Володимир

Номер залікової: 6429 Номер в списку: 27

Київ – 2017

**Постановка задачі до комп’ютерного практикуму № 7**

При виконанні комп’ютерого практикуму слід реалізувати наступні задачі:

1. Перезавантажити віртуальний метод bool Equals (object obj), таким чином, щоб об‘єкти були рівними, якщо рівні всі дані об‘єктів. Для кожного з класів самостійно визначити, які атрибути використовуються для порівняння;
2. Визначити операції == та != . При цьому врахувати, що визначення операцій повинно бути погоджено з перезавантаженим методом Equals, тобто критерії, за якими перевіряється рівність об'єктів в методі Equals, повинні використовуватися і при перевірці рівності об'єктів в операціях == та !=;
3. Перевизначити віртуальний метод int GetHashCode(). Класи базової бібліотеки, що викликають метод GetHashCode() з призначеного користувальницького типу, припускають, що рівним об'єктів відповідають рівні значення хеш-кодів. Тому в разі, коли під рівністю об'єктів розуміється збіг даних (а не посилань), реалізація методу GetHashCode() повинна для об'єктів з однаковими даними повертати рівні значення хеш-кодів.
4. Визначити метод object DeepCopy() для створення повної копії об'єкта. Визначені в деяких класах базової бібліотеки методи Clone() та Copy() створюють обмежену (shallow) копію об'єкта - при копіюванні об'єкта копії створюються тільки для полів структурних типів, для полів, на які посилаються типи, копіюються тільки посилання. В результаті в обмеженій копії об'єкта поля-посилання вказують на ті ж об'єкти, що і в вихідному об'єкті. Метод DeepCopy() повинен створити повні копії всіх об'єктів, посилання на які містять поля типу. Після створення повна копія не залежить від вихідного об'єкта - зміна будь-якого поля або властивості вихідного об'єкта не повинно призводити до зміни копії. При реалізації методу DeepCopy() в класі, який має поле типу System.Collections.ArrayList, слід мати на увазі, що визначені в класі ArrayList конструктор ArrayList (ICollection) і метод Clone() при створенні копії колекції, що складається з елементів, на які посилаються типів, копіюють тільки посилання. Метод DeepCop () повинен створити як копії елементів колекції ArrayList, так і повні копії об'єктів, на які посилаються елементи колекції. Для типів, що містять колекції, реалізація методу DeepCopy() спрощується, якщо в типах елементів колекцій також визначити метод DeepCopy ().
5. Перезавантажити віртуальний метод string ToString() для формування строки з інформацією про всі елементи списку
6. Підготувати демонстраційний приклад, в котрому будуть використані всі розроблені методи
7. Підготувати звіт з результатами виконаної роботи.

При виконанні комп’ютерого практикуму слід реалізувати наступні задачі:

1. Визначити клас, котрий містить типізовану колекцію та котрий за допомогою подій інформує ппо зміни в колекції.

Колекція складається з об'єктів силочних типів. Колекція змінюється при видаленні/додаванні елементів або при зміні одного з вхідних в колекцію посилань, наприклад, коли одному з посилань присвоюється нове значення. В цьому випадку у відповідних методах або властивості класу кидаються події.

При зміні даних об'єктів, посилання на які входять в колекцію, значення самих посилань не змінюються. Цей тип змін не породжує подій.

Для подій, що сповіщають про зміни в колекції, визначається свій делегат. Події реєструються в спеціальних класах-слухачах.

1. Реалізувати обробку помилок, при цьому необхідно перевизначити за допомогою наслідування наступні події:

1) StackOverflowException

2) ArrayTypeMismatchException

3) DivideByZeroException

4) IndexOutOfRangeException

5) InvalidCastException

6) OutOfMemoryException

7) OverflowException

1. Підготувати демонстраційний приклад, в котрому будуть використані всі розроблені методи
2. Підготувати звіт з результатами виконаної роботи.

**Варіант індивідуального завдання:**

Создать абстрактный класс Point (точка). На его основе создать классы ColoredPoint и Line. На основе класса Line создать класс ColoredLine и класс PolyLine (многоугольник).

Все классы должны иметь виртуальные методы установки и получения значений всех координат, а также изменения цвета и получения текущего цвета.

Создать класс Picture, содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти.

Предусмотреть возможность вывода характеристик объектов списка.

**Код програми:**

ColoredLine.cs

using System.Windows.Media;

namespace lab2

{

class ColoredLine: Line

{

private Color color;

public Color Color { get => color; set => color = value; }

public ColoredLine(int x1,int y1, int x2, int y2, Color color):base(x1,y1,x2,y2)

{

this.Color = color;

}

public override bool Equals(object obj)

{

if (!(obj is ColoredLine)) return false;

ColoredLine tmp = (ColoredLine)obj;

if (this.X1 == tmp.X1 && this.Y1 == tmp.Y1 && this.X2 == tmp.X2 &&

this.Y2 == tmp.Y2 && this.Color == tmp.Color)

return true;

else return false;

}

public static bool operator ==(ColoredLine p1, ColoredLine p2)

{

if (p1.Equals(p2)) return true;

else return false;

}

public static bool operator !=(ColoredLine p1, ColoredLine p2)

{

if (p1.Equals(p2)) return false;

else return true;

}

public override int GetHashCode()

{

return (this.X1 + this.Y1 + this.X2 + this.Y2)%90 +

(this.Color.A + this.Color.B + this.Color.G + this.Color.R)%150+350;

}

public override Point DeepCopy()

{

return new ColoredLine(this.X1, this.Y1, this.X2, this.Y2, this.Color);

}

public override string ToString()

{

string tmp = "ColoredLine : A(" + this.X1 + ";" + this.Y1 + "), B(" +

this.X2 + ";" + this.Y2 + ") color= " + this.Color.ToString();

return tmp;

}

}

}

ColoredPoint.cs

using System.Windows.Media;

namespace lab2

{

class ColoredPoint: Point

{

private int x;

private int y;

private Color color;

public int X { get => x; set => x = value; }

public int Y { get => y; set => y = value; }

public Color Color { get => color; set => color = value; }

public ColoredPoint(int x, int y, Color color)

{

this.x = x;

this.y = y;

this.Color = color;

}

public override bool Equals(object obj)

{

if (!(obj is ColoredPoint)) return false;

ColoredPoint tmp = (ColoredPoint)obj;

if (this.X == tmp.X && this.Y == tmp.Y && this.Color == tmp.Color)

return true;

else return false;

}

public static bool operator==(ColoredPoint p1,ColoredPoint p2)

{

if (p1.Equals(p2)) return true;

else return false;

}

public static bool operator !=(ColoredPoint p1, ColoredPoint p2)

{

if (p1.Equals(p2)) return false;

else return true;

}

public override int GetHashCode()

{

return (this.X+this.Y)%100 +

(this.Color.A+this.Color.B+this.Color.G+this.Color.R)%90;

}

public override Point DeepCopy()

{

return new ColoredPoint(this.X, this.Y, this.Color);

}

public override string ToString()

{

string tmp = "ColoredPoint : (" + this.X + ";" + this.Y +

") color = " + this.Color.ToString();

return tmp;

}

}

}

ExeptionPrg.cs

using System;

using System.Collections.Generic;

using System.Windows.Media;

namespace lab2

{

static class ExeptionPrg

{

static public void test()

{

Picture picture = new Picture(new ColoredLine(0, 0, 1, 1, Color.FromArgb(0, 199, 21, 133)),

new ColoredPoint(2, 2, Color.FromArgb(0, 199, 21, 133)), new Line(4, 4, 5, 5),

new PolyLine(new Line(0, 0, 1, 1), new Line(1, 1, 1, 0), new Line(1, 0, 0, 0)));

ColoredLine coloredLine = (ColoredLine)picture.get(0);

Console.WriteLine(coloredLine);

ColoredPoint coloredPoint = (ColoredPoint)picture.get(1);

Console.WriteLine(coloredPoint);

Line line = (Line)picture.get(2);

Console.WriteLine(line);

PolyLine polyLine = (PolyLine)picture.get(3);

Picture picture2 = new Picture(coloredPoint, coloredLine, line, polyLine);

List<Line> tmplist1 = new List<Line>();

tmplist1.Add(new Line(0, 0, 0, 1));

tmplist1.Add(new Line(0, 1, 1, 1));

tmplist1.Add(new Line(1, 1, 1, 0));

tmplist1.Add(new Line(1, 0, 0, 0));

PolyLine polyLine1 = new PolyLine(tmplist1);

Console.WriteLine(polyLine);

List<Point> tmpList = new List<Point>();

tmpList.Add(coloredPoint);

tmpList.Add(coloredLine);

tmpList.Add(line);

tmpList.Add(polyLine);

picture2 = new Picture(tmpList);

Console.WriteLine(picture2);

if (picture.Equals(picture2))

Console.WriteLine("picture1 = picture2");

if (coloredPoint == (new ColoredPoint(2, 2, Color.FromArgb(0, 199, 21, 133))))

Console.WriteLine("coloredPoint = (new ColoredPoint(2, 2, Color.FromArgb(0, 199, 21, 133)))");

ColoredPoint coloredPoint2 = new ColoredPoint(2, 1, Color.FromArgb(0, 199, 21, 133));

if (!(coloredLine != picture.get(0)))

Console.WriteLine("coloredLine = picture.get(0)");

if (!(coloredPoint != picture.get(1)))

Console.WriteLine("coloredPoint = picture.get(1)");

if (!(line != picture.get(2)))

Console.WriteLine("line рiвний =.get(2)");

if (!(polyLine != picture.get(3)))

Console.WriteLine("polyLine = picture.get(3)");

if (!(picture2 != picture))

Console.WriteLine("picture2 = picture");

Console.WriteLine("GetHashCode:");

Console.WriteLine(coloredLine.GetHashCode());

Console.WriteLine(coloredPoint.GetHashCode());

Console.WriteLine(line.GetHashCode());

Console.WriteLine(polyLine.GetHashCode());

Console.WriteLine(picture2.GetHashCode());

ColoredPoint mp1 = (ColoredPoint)coloredPoint.DeepCopy();

ColoredLine mcl = (ColoredLine)coloredLine.DeepCopy();

Line ml = (Line)line.DeepCopy();

PolyLine mpol = (PolyLine)polyLine.DeepCopy();

Picture mp = picture.DeepCopy();

Console.WriteLine("Пiсля створення deepcopies зрiвнюємо з оригiналом ");

Console.WriteLine(mp1 == coloredPoint);

Console.WriteLine(mcl == coloredLine);

Console.WriteLine(ml == line);

Console.WriteLine(mpol == polyLine);

Console.WriteLine(mp == picture);

picture.Deleted += Show\_Message;

picture.Added += Show\_Message;

polyLine.Added += Show\_Message;

polyLine.Deleted += Show\_Message;

polyLine.Changed += Show\_Message;

picture.Changed += Show\_Message;

polyLine.deleteItem(0);

picture.deleteItem(0);

picture.addFigure(polyLine);

polyLine.Lines = new List<Line>();

picture.Figures = new List<Point>();

try

{

polyLine = new PolyLine(new Line(1, 2, 3, 4), new Line(0, 0, 0, 0));

}

catch (ArgumentException e)

{

Console.WriteLine(e.Message);

}

try

{

polyLine = new PolyLine(new Line(1, 2, 3, 4), new Line(0, 0, 0, 0));

polyLine.addFigure(new Line(0, 0, 2, 2));

}

catch (ArgumentException e)

{

Console.WriteLine(e.Message);

}

try

{

Picture p = new Picture();

p.deleteItem(10);

}

catch (IndexOutOfRangeException e)

{

Console.WriteLine(e.Message);

}

try

{

PolyLine p = new PolyLine();

p.deleteItem(10);

}

catch (IndexOutOfRangeException e)

{

Console.WriteLine(e.Message);

}

try

{

Picture p = new Picture();

Point t = p.get(10);

}

catch (IndexOutOfRangeException e)

{

Console.WriteLine(e.Message);

}

Console.ReadLine();

}

private static void Show\_Message(string message)

{

Console.WriteLine(message);

}

}

}

Line.cs

namespace lab2

{

class Line: Point

{

private int x1,x2;

private int y1,y2;

public Line()

{

this.x1 = this.x2 = this.y2 = this.y1 = 0;

}

public Line(int x1, int y1, int x2, int y2)

{

this.x1 = x1;

this.x2 = x2;

this.y1 = y1;

this.y2 = y2;

}

public int X1 { get => x1; set => x1 = value; }

public int X2 { get => x2; set => x2 = value; }

public int Y1 { get => y1; set => y1 = value; }

public int Y2 { get => y2; set => y2 = value; }

public override bool Equals(object obj)

{

if (!(obj is Line)) return false;

Line tmp = (Line)obj;

if (this.X1 == tmp.X1 && this.Y1 == tmp.Y1 && this.X2 == tmp.X2 &&

this.Y2==tmp.Y2)

return true;

else return false;

}

public static bool operator ==(Line p1, Line p2)

{

if (p1.Equals(p2)) return true;

else return false;

}

public static bool operator !=(Line p1, Line p2)

{

if (p1.Equals(p2)) return false;

else return true;

}

public override int GetHashCode()

{

return (this.X1 + this.Y1 + this.X2 + this.Y2) % 150 + 250 ;

}

public override Point DeepCopy()

{

return new Line(this.X1, this.Y1, this.X2, this.Y2);

}

public override string ToString()

{

string tmp = "Line : A(" + this.X1 + ";" + this.Y1 + "), B(" +

this.X2+";"+this.Y2+")";

return tmp;

}

}

}

Picture.cs

using System.Collections.Generic;

namespace lab2

{

class Picture

{

private List<Point> figures = new List<Point>();

internal List<Point> Figures { get => figures; set {

if (value.Count == figures.Count)

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

{

if (!figures[i].Equals(value[i]))

Changed($"Picture змiнений");

}

else Changed($"Picture змiнений");

figures = value;

} }

public delegate void PictureStateHandler(string message);

public event PictureStateHandler Deleted;

public event PictureStateHandler Changed;

public event PictureStateHandler Added;

public Picture(params Point[] figures)

{

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

{

this.Figures.Add(figures[i].DeepCopy());

}

}

public Picture(List<Point> figures)

{

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

{

this.Figures.Add(figures[i].DeepCopy());

}

}

public Point get(int index)

{

if (index < 0 || index > this.Figures.Count)

throw new System.IndexOutOfRangeException("Iндекс повинен бути в межах [0;" + Figures.Count + "]");

return Figures[index].DeepCopy();

}

public void deleteItem(int index)

{

if (index < 0 || index > Figures.Count)

throw new System.IndexOutOfRangeException("Iндекс повинен бути в межах [0;" + Figures.Count+"]");

Figures.RemoveAt(index);

if(Deleted!=null)

Deleted($"Видалили {Figures[index]}");

}

public void addFigure(Point point)

{

if (Added != null)

Added($"Додали " + point);

Figures.Add(point);

}

public override bool Equals(object obj)

{

if (!(obj is Picture)) return false;

Picture tmp = (Picture)obj;

if (tmp.Figures.Count == this.Figures.Count)

{

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

{

if (this.Figures.IndexOf(tmp.Figures[i]) == -1)

return false;

}

return true;

}

return false;

}

public static bool operator ==(Picture p1, Picture p2)

{

if (p1.Equals(p2)) return true;

else return false;

}

public static bool operator !=(Picture p1, Picture p2)

{

if (p1.Equals(p2)) return false;

else return true;

}

public override int GetHashCode()

{

int hash = 0;

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

{

hash += Figures[i].GetHashCode();

}

hash %= 9500;

hash += 2500;

return hash;

}

public Picture DeepCopy()

{

List<Point> tmp = new List<Point>();

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

{

tmp.Add(this.Figures[i].DeepCopy());

}

return new Picture(tmp);

}

public override string ToString()

{

string tmp = "Picture:";

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

{

tmp += "\n"+Figures[i].ToString();

}

return tmp;

}

}

}

Point.cs

namespace lab2

{

abstract class Point

{

abstract public Point DeepCopy();

}

}

PolyLine.cs

using System;

using System.Collections.Generic;

namespace lab2

{

class PolyLine : Line

{

private List<Line> lines = new List<Line>();

internal List<Line> Lines { get {return lines; } set

{

if(value.Count == lines.Count)

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

{

if(!lines[i].Equals(value[i]))

Changed($"Polyline змiнений");

}

else Changed($"Polyline змiнений");

lines = value;

}

}

public delegate void PolyLineStateHandler(string message);

public event PolyLineStateHandler Deleted;

public event PolyLineStateHandler Changed;

public event PolyLineStateHandler Added;

public PolyLine(params Line[] lines)

{

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

{

if (lines[i].X2 != lines[(i + 1) % lines.Length].X1

|| lines[i].Y2 != lines[(i + 1) % lines.Length].Y1)

throw new ArgumentException("Лiнiї повиннi бути з'єднанi. X2, Y2 з першої лiнiї = з X1, Y1 з другої лiнiї");

}

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

{

this.Lines.Add((Line)lines[i].DeepCopy());

}

}

public PolyLine(List<Line> lines)

{

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

{

if (lines[i].X2 != lines[(i + 1) % lines.Count].X1

|| lines[i].Y2 != lines[(i + 1) % lines.Count].Y1)

throw new ArgumentException("Лiнiї повиннi бути з'єднанi. X2, Y2 з першої лiнiї = з X1, Y1 з другої лiнiї");

}

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

this.Lines.Add((Line)lines[i].DeepCopy());

}

public void deleteItem(int index)

{

if (index < 0 || index > lines.Count)

throw new System.IndexOutOfRangeException("Iндекс повинен бути в межах [0;" + lines.Count + "]");

Lines.RemoveAt(index);

if (Deleted != null)

Deleted($"Видалили {Lines[index]}");

}

public void addFigure(Line line)

{

if (lines.Count < 0) lines.Add(line);

if (line.X1 != lines[(this.lines.Count) - 1].X2 || line.Y1 != this.lines[(lines.Count) - 1].Y2)

throw new ArgumentException("Лiнiї повиннi бути з'єднанi.X2, Y2 з першої = рівнi з X1, Y1 з другої лiнiї");

if (Added != null)

Added($"Додали " + line);

lines.Add(line);

}

public override bool Equals(object obj)

{

if (!(obj is PolyLine)) return false;

PolyLine tmp = (PolyLine)obj;

if(tmp.Lines.Count==this.Lines.Count)

{

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

{

if (this.Lines.IndexOf(tmp.Lines[i])==-1)

return false;

}

return true;

}

return false;

}

public static bool operator ==(PolyLine p1, PolyLine p2)

{

if (p1.Equals(p2)) return true;

else return false;

}

public static bool operator !=(PolyLine p1, PolyLine p2)

{

if (p1.Equals(p2)) return false;

else return true;

}

public override int GetHashCode()

{

int hash = 0;

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

{

hash += Lines[i].GetHashCode();

}

hash %= 900;

hash += 600;

return hash;

}

public override Point DeepCopy()

{

List<Line> tmp = new List<Line>();

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

tmp.Add((Line)Lines[i].DeepCopy());

return new PolyLine(tmp);

}

public override string ToString()

{

string tmp = "PolyLine:\n";

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

{

tmp += Lines[i].ToString()+"\n";

}

return tmp;

}

}

}

Program.cs

using System;

using System.Collections.Generic;

using System.Windows.Media;

namespace lab2

{

class Program

{

static void Main(string[] args)

{

ExeptionPrg.test();

}

}

}

Iлюстрації

