**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Белова Е.О. |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

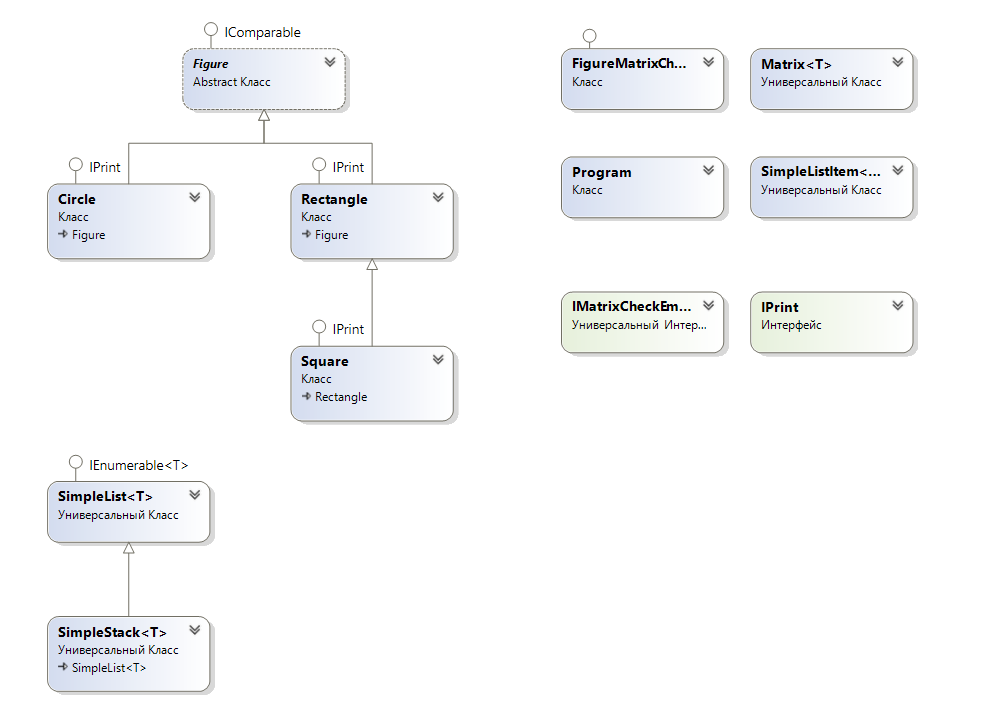
г. Москва, 2018 г.

**Описание задания:**

Разработать программу, реализующую работу с коллекциями.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создать объекты классов «Прямоугольник», «Квадрат», «Круг».
3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры.
4. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
5. Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
6. Модифицировать класс разреженной матрицы (проект SparseMatrix) для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.
7. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (разобранного в пособии). Необходимо добавить в класс методы:
   1. public void Push(T element) – добавление в стек;
   2. public T Pop() – чтение с удалением из стека.
8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

**Диаграмма классов:**



**Текст программы:**

**Program.cs:**

using DescriptionFigures;

using System;

using System.Collections;

using System.Collections.Generic;

namespace Figures

{

class Program

{

static void Main(string[] args)

{

//2

Rectangle rect = new Rectangle(5, 4);

Square square = new Square(5);

Circle circle = new Circle(5);

//4

ArrayList al = new ArrayList

{

rect,

circle,

square

};

al.Sort();

foreach (object i in al)

Console.WriteLine(i.ToString());

//5

List<Figure> bl = new List<Figure>

{

rect,

circle,

square

};

bl.Sort();

foreach (Figure i in bl)

Console.WriteLine(i.ToString());

//6

Console.WriteLine("\nМатрица:");

Matrix<Figure> matrix = new Matrix<Figure>(3, 3, 4, new FigureMatrixCheckEmpty());

matrix[0, 0, 0] = rect;

matrix[1, 1, 1] = square;

matrix[2, 2, 2] = circle;

Console.WriteLine(matrix.ToString());

Console.WriteLine();

//8

SimpleStack<Figure> stack = new SimpleStack<Figure>();

stack.Push(rect);

stack.Push(square);

stack.Push(circle);

while (stack.Count > 0)

{

Figure f = stack.Pop();

Console.WriteLine(f);

}

}

}

}

**Circle.cs:**

using System;

namespace DescriptionFigures

{

class Circle : Figure, IPrint

{

public double Radius { get; set; }

public Circle(double pr)

{

this.Radius = pr;

this.Type = "Круг";

}

public override double Area()

{

double Result = Math.PI \* this.Radius \* this.Radius;

return Result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

**Figure.cs:**

using System;

namespace DescriptionFigures

{

abstract class Figure : IComparable

{

public string Type { get; set; }

public abstract double Area();

// 3

public int CompareTo(object obj)

{

Figure p = (Figure)obj;

if (this.Area() < p.Area())

return -1;

else if (this.Area() == p.Area())

return 0;

else

return 1;

}

public override string ToString()

{

return this.Type + " площадью " + this.Area().ToString();

}

}

}

**FigureMatrixCheckEmpty.cs:**

using DescriptionFigures;

namespace Figures

{

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

/// <summary>

/// В качестве пустого элемента возвращается null

/// </summary>

public Figure getEmptyElement()

{

return null;

}

/// <summary>

/// Проверка что переданный параметр равен null

/// </summary>

public bool checkEmptyElement(Figure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

}

**IMatrixCheckEmpty.cs:**

namespace Figures

{

/// <summary>

/// Проверка пустого элемента матрицы

/// </summary>

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

}

**IPrint.cs:**

namespace DescriptionFigures

{

interface IPrint

{

void Print();

}

}

**Matrix.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace Figures

{

public class Matrix<T>

{

/// <summary>

/// Словарь для хранения значений

/// </summary>

Dictionary<string, T> \_matrix = new Dictionary<string, T>();

/// <summary>

/// Количество элементов по OX

/// </summary>

int maxX;

/// <summary>

/// Количество элементов по OY

/// </summary>

int maxY;

/// <summary>

/// Количество элементов по OZ

/// </summary>

int maxZ;

/// <summary>

/// Реализация интерфейса для проверки пустого элемента

/// </summary>

IMatrixCheckEmpty<T> сheckEmpty;

/// <summary>

/// Конструктор

/// </summary>

public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> сheckEmptyParam)

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.сheckEmpty = сheckEmptyParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

public T this[int x, int y, int z]

{

set

{

CheckBounds(x, y, z);

string key = DictKey(x, y, z);

this.\_matrix.Add(key, value);

}

get

{

CheckBounds(x, y, z);

string key = DictKey(x, y, z);

if (this.\_matrix.ContainsKey(key))

{

return this.\_matrix[key];

}

else

{

return this.сheckEmpty.getEmptyElement();

}

}

}

/// <summary>

/// Проверка границ

/// </summary>

void CheckBounds(int x, int y, int z)

{

if (x < 0 || x >= this.maxX)

{

throw new ArgumentOutOfRangeException("x", "x=" + x + " выходит за границы");

}

if (y < 0 || y >= this.maxY)

{

throw new ArgumentOutOfRangeException("y", "y=" + y + " выходит за границы");

}

if (z < 0 || z >= this.maxZ)

{

throw new ArgumentOutOfRangeException("z", "z=" + z + " выходит за границы");

}

}

/// <summary>

/// Формирование ключа

/// </summary>

string DictKey(int x, int y, int z)

{

return x.ToString() + "\_" + y.ToString() + "\_" + z.ToString();

}

/// <summary>

/// Приведение к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

StringBuilder b = new StringBuilder();

for (int l = 0; l < this.maxZ; l++)

{

b.Append("|");

for (int j = 0; j < this.maxY; j++)

{

b.Append("[");

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

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t");

}

//Если текущий элемент не пустой

if (!this.сheckEmpty.checkEmptyElement(this[i, j, l]))

{

//Добавить приведенный к строке текущий элемент

b.Append(this[i, j, l].ToString());

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}

b.Append("]");

}

b.Append("|\n");

}

return b.ToString();

}

}

}

**Rectangle.cs:**

using System;

namespace DescriptionFigures

{

class Rectangle : Figure, IPrint

{

public double Height { get; set; }

public double Width { get; set; }

public Rectangle(double ph, double pw)

{

this.Height = ph;

this.Width = pw;

this.Type = "Прямоугольник";

}

public override double Area()

{

double Result = this.Width \* this.Height;

return Result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

**SimpleList.cs:**

using System;

using System.Collections.Generic;

namespace Figures

{

public class SimpleList<T> : IEnumerable<T>

where T : IComparable

{

protected SimpleListItem<T> first = null;

protected SimpleListItem<T> last = null;

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

public void Add(T element)

{

SimpleListItem<T> newItem = new SimpleListItem<T>(element);

this.Count++;

if (last == null)

{

this.first = newItem;

this.last = newItem;

}

else

{

this.last.next = newItem;

this.last = newItem;

}

}

public SimpleListItem<T> GetItem(int number)

{

if ((number < 0) || (number >= this.Count))

{

throw new Exception("Выход за границу индекса");

}

SimpleListItem<T> current = this.first;

int i = 0;

while (i < number)

{

current = current.next;

i++;

}

return current;

}

public T Get(int number)

{

return GetItem(number).data;

}

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

while (current != null)

{

yield return current.data;

current = current.next;

}

}

System.Collections.IEnumerator

System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public void Sort()

{

Sort(0, this.Count - 1);

}

private void Sort(int low, int high)

{

int i = low;

int j = high;

T x = Get((low + high) / 2);

do

{

while (Get(i).CompareTo(x) < 0)

++i;

while (Get(j).CompareTo(x) > 0)

--j;

if (i <= j)

{

Swap(i, j);

i++;

j--;

}

} while (i <= j);

if (low < j)

Sort(low, j);

if (i < high)

Sort(i, high);

}

private void Swap(int i, int j)

{

SimpleListItem<T> ci = GetItem(i);

SimpleListItem<T> cj = GetItem(j);

T temp = ci.data;

ci.data = cj.data;

cj.data = temp;

}

}

}

SimpleListItem.cs:

namespace Figures

{

public class SimpleListItem<T>

{

public T data { get; set; }

public SimpleListItem<T> next { get; set; }

public SimpleListItem(T param)

{

this.data = param;

}

}

}

**SimpleStack.cs:**

using System;

namespace Figures

{

public class SimpleStack<T> : SimpleList<T>

where T : IComparable

{

public void Push(T item)

{

this.Add(item);

}

public T Pop()

{

var data = default(T);

if (Count == 0)

{

return data;

}

else if (Count == 1)

{

data = last.data;

first = last = null;

}

else

{

var newLast = this.GetItem(Count - 2);

data = last.data;

newLast.next = null;

last = newLast;

}

Count--;

return data;

}

}

}

**Square.cs:**

namespace DescriptionFigures

{

class Square : Rectangle, IPrint

{

public Square(double size) : base(size, size)

{

this.Type = "Квадрат";

}

}

}

**Примеры:**

