**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет технологий»

Отчет по лабораторной работе №3

«Коллекции»

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

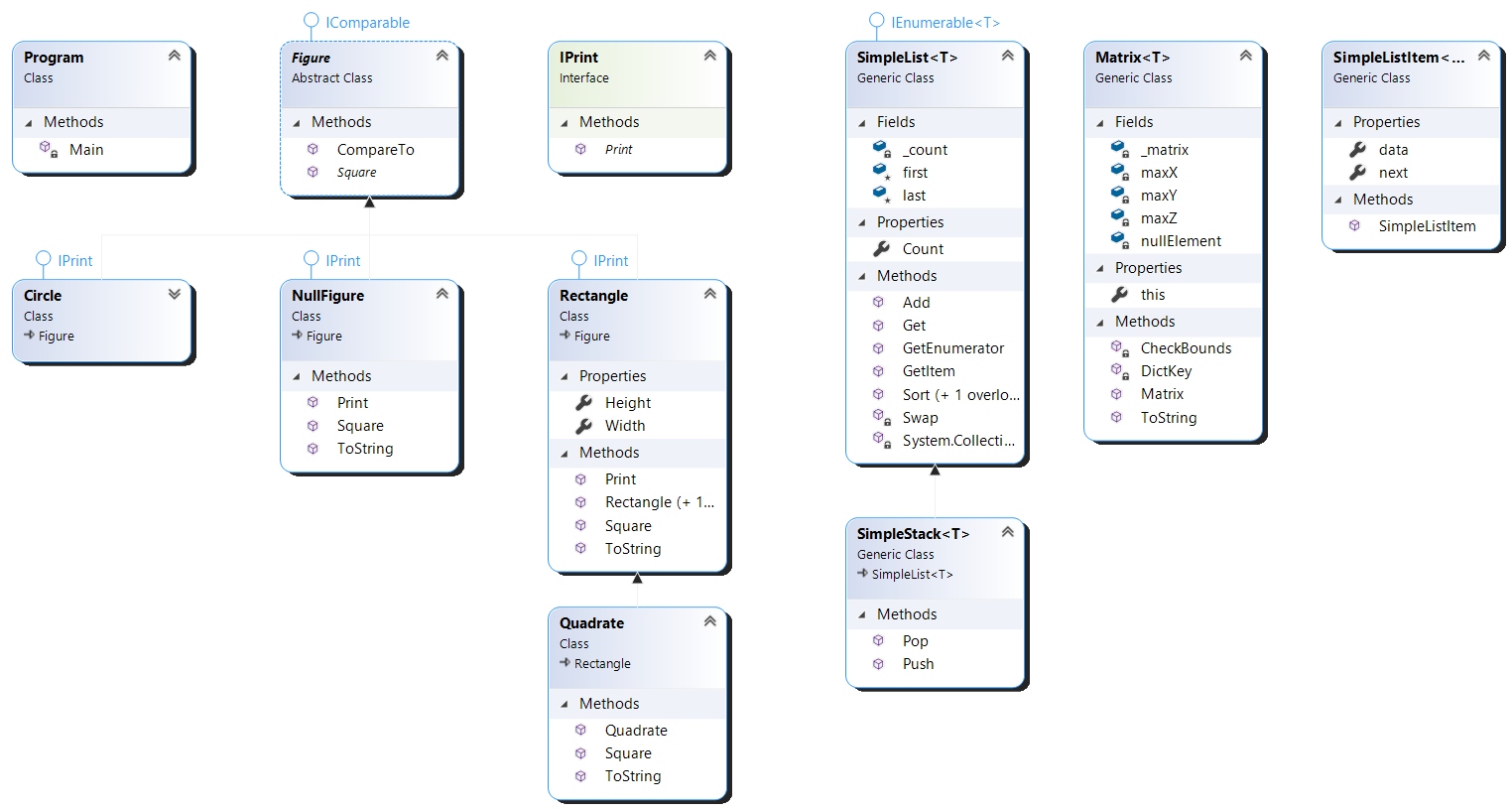
Москва, 2020 г.

**Условие 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 (проект SimpleListProject). Необходимо добавить в класс методы:
   * public void Push(T element) – добавление в стек;
   * 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 System;

using System.Collections;

using System.Collections.Generic;

namespace BKIT\_LAB3

{

class Program

{

static void Main(string[] args)

{

Rectangle rect = new Rectangle(5, 7);

Quadrate quad = new Quadrate(8);

Circle circ = new Circle(3);

Console.WriteLine("Before sorting:\n\t" + rect + "\n\t" + quad + "\n\t" + circ);

ArrayList figuresArrayList = new ArrayList();

figuresArrayList.Add(rect);

figuresArrayList.Add(quad);

figuresArrayList.Add(circ);

figuresArrayList.Sort();

Console.WriteLine("\nArrayList sorting: ");

foreach (object obj in figuresArrayList)

Console.WriteLine("\t" + obj);

List<Figure> figuresList = new List<Figure> { rect, quad, circ };

figuresList.Sort();

Console.WriteLine("\nList sorting: ");

foreach (Figure figure in figuresList)

Console.WriteLine("\t" + figure);

Console.Write("\nMatrix: ");

Matrix<Figure> figuresMatrix = new Matrix<Figure>(4, 3, 2, new NullFigure());

figuresMatrix[0, 1, 1] = rect;

figuresMatrix[2, 2, 0] = quad;

figuresMatrix[1, 2, 1] = circ;

Console.Write("\n" + figuresMatrix);

Console.WriteLine("\nElements in stack: ");

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

figuresStack.Push(rect);

figuresStack.Push(circ);

figuresStack.Push(quad);

while (figuresStack.Count != 0)

Console.WriteLine("\t" + figuresStack.Pop());

Console.ReadKey();

}

}

}

**Figures.cs:**

using System;

namespace BKIT\_LAB3

{

interface IPrint

{

void Print();

}

public abstract class Figure : IComparable

{

public abstract double Square();

public int CompareTo(object obj)

{

Figure comparedFigure = obj as Figure;

if (comparedFigure != null)

return Square().CompareTo(comparedFigure.Square());

throw new Exception("Impossible to compare these objects!");

}

}

class Rectangle : Figure, IPrint

{

public double Width { get; set; }

public double Height { get; set; }

public Rectangle(double \_width, double \_height)

{

Width = \_width;

Height = \_height;

}

protected Rectangle(double \_width)

{

Width = \_width;

}

public override double Square()

{

return Width \* Height;

}

public override string ToString()

{

return "Rectangle {width: " + Width + ", height: " + Height + ", square: " + Square() + "}";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

class Quadrate : Rectangle

{

public Quadrate(double \_width) : base(\_width) { }

public override string ToString()

{

return "Quadrate {width: " + Width + ", square: " + Square() + "}";

}

public override double Square()

{

return Width \* Width;

}

}

class Circle : Figure, IPrint

{

public double Radius { get; set; }

public Circle(double \_radius)

{

Radius = \_radius;

}

public override double Square()

{

return Math.PI \* Radius \* Radius;

}

public override string ToString()

{

return "Circle {radius: " + Radius + ", square: " + Square() + "}";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

class NullFigure : Figure, IPrint

{

public override double Square()

{

return 0;

}

public override string ToString()

{

return "None";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

**Matrix.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace BKIT\_LAB3

{

public class Matrix<T>

{

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

int maxX;

int maxY;

int maxZ;

T nullElement;

public Matrix(int px, int py, int pz, T nullElementParam)

{

maxX = px;

maxY = py;

maxZ = pz;

nullElement = nullElementParam;

}

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

{

get

{

CheckBounds(x, y, z);

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

return (\_matrix.ContainsKey(key)) ? \_matrix[key] : nullElement;

}

set

{

CheckBounds(x, y, z);

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

\_matrix.Add(key, value);

}

}

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

{

if (x < 0 || x >= maxX) throw new Exception("x=" + x + " выходит за границы");

if (y < 0 || y >= maxY) throw new Exception("y=" + y + " выходит за границы");

if (z < 0 || z >= maxZ) throw new Exception("z=" + z + " выходит за границы");

}

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

{

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

}

public override string ToString()

{

StringBuilder b = new StringBuilder();

for (int z = 0; z != maxZ; ++z)

{

for (int j = 0; j != maxY; ++j)

{

b.Append("[ ");

for (int i = 0; i != maxX; ++i)

{

if (i > 0) b.Append(" | ");

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

}

b.Append(" ]\n");

}

b.Append("\n");

}

return b.ToString();

}

}

}

**SimpleList.cs:**

using System;

using System.Collections.Generic;

namespace SimpleListProject

{

public class SimpleListItem<T>

{

public T data { get; set; }

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

public SimpleListItem(T param)

{

data = param;

}

}

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);

Count++;

if (last == null)

{

first = newItem;

last = newItem;

}

else

{

last.next = newItem;

last = newItem;

}

}

public SimpleListItem<T> GetItem(int number)

{

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

{

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

}

SimpleListItem<T> current = 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 = 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, 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;

}

}

}

**SimpleStack.cs:**

using System;

using SimpleListProject;

namespace BKIT\_LAB3

{

class SimpleStack<T> : SimpleList<T>

where T : IComparable

{

public void Push(T element)

{

Add(element);

}

public T Pop()

{

if (Count != 0)

{

T topElement = Get(Count - 1);

if (Count != 1)

{

last = GetItem(Count - 2);

last.next = null;

}

else

{

first = null;

last = null;

}

--Count;

return topElement;

}

throw new Exception("Impossible to get the element from empty stack");

}

}

}

**Примеры выполнения программы:**

|  |  |
| --- | --- |
| *Пример* | *Вывод программы* |
| 1 |  |