**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «**Информатика и системы управления**»

Кафедра ИУ5. Группа 31

Отчет по лабораторной работе № 3

по предмету: «Базовые компоненты интернет-технологий»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студентка группы ИУ5-31 |  |
| Лебедева Дарья |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

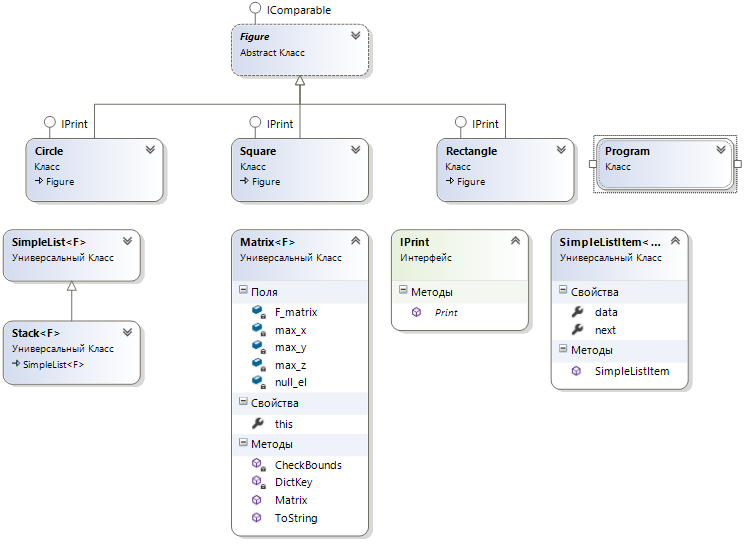
г. Москва, 2017 г.

**Лабораторная работа №3**

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**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace Laba\_2

{

// Абстрактный класс «Геометрическая фигура»

abstract class Figure : IComparable

{

public string Type

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

string \_Type;

// Вычисление площади фигуры

public abstract double Area();

public override string ToString()

{

return this.\_Type + " Площадь =" + this.Area().ToString();

}

// Сравнение элементов (для сортировки списка)

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;

}

}

class Program

{

static void Main(string[] args)

{

Rectangle rect = new Rectangle(3, 4);

Square square = new Square(5);

Circle circle = new Circle(10);

rect.Print();

square.Print();

circle.Print();

///////////////////////////////////////////////////////////

Console.WriteLine("\nArrayList");

ArrayList al = new ArrayList();

al.Add(circle);

al.Add(rect);

al.Add(square);

al.Sort();

foreach (var x in al) Console.WriteLine(x);

//////////////////////////////////////////////////////////

Console.WriteLine("\nList<Figure>");

List<Figure> fl = new List<Figure>();

fl.Add(circle);

fl.Add(rect);

fl.Add(square);

fl.Sort();

foreach (var x in fl) Console.WriteLine(x);

//////////////////////////////////////////////////////////

Console.WriteLine("\nМатрица");

Matrix<double> cube = new Matrix<double>(3, 3, 3, 0);

cube[0, 0, 0] = rect.Area();

cube[1, 1, 1] = square.Area();

cube[2, 2, 2] = circle.Area();

Console.WriteLine(cube.ToString());

//////////////////////////////////////////////////////////

Console.WriteLine("\nСтек");

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

stack.Push(rect);

stack.Push(square);

stack.Push(circle);

while (stack.Count > 0)

{

Figure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadLine();

}

}}

* using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

class Square : Figure, IPrint

{

double height\_2;

public Square(double hg)

{

this.height\_2 = hg;

this.Type = "Квадрат";

}

public override double Area()

{

double result = this.height\_2 \* this.height\_2;

return result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

//////////////////////////////////////////////////////

* using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

class Rectangle : Figure, IPrint

{

double height;

double width;

public Rectangle(double hg, double wd)

{

this.height = hg;

this.width = wd;

this.Type = "Прямоугольник";

}

public override double Area()

{

double result = this.height \* this.width;

return result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

/////////////////////////////////////////////////

* using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

class Circle : Figure, IPrint

{

double radius;

public Circle(double rd)

{

this.radius = rd;

this.Type = "Круг";

}

public override double Area()

{

double result = Math.PI \* this.radius \* this.radius;

//double result = this.radius \* this.radius;

return result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

////////////////////////////////////////

* using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

interface IPrint

{

void Print();

}

}

/////////////////////////////////////////

* using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

public class Matrix<F>

{//Словарь для хранения значений

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

//Количество элементов по горизонтали (макс количество столбцов)

int max\_x;

//Количество элементов по вертикали (макс количество строк)

int max\_y;

//Добавленное измерение

int max\_z;

//Пустой элемент, который возвращается если элемент с нужными координатами не был задан

F null\_el;

//Конструктор

public Matrix(int fx, int fy, int fz, F null\_elParam)

{

this.max\_x = fx;

this.max\_y = fy;

this.max\_z = fz;

this.null\_el = null\_elParam;

}

//Индекстатор для доступа к данным

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

{

get

{

CheckBounds(x, y, z);

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

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

{

return this.F\_matrix[key];

}

else

{

return this.null\_el;

}

}

set

{

CheckBounds(x, y, z);

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

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

}

}

//Проверка границ

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

{if (x < 0 || x >= this.max\_x) throw new Exception("Значение x =" + x + "выходит за пределы");

if (y < 0 || y >= this.max\_y) throw new Exception("Значение y =" + y + "выходит за пределы");

if (z < 0 || z >= this.max\_z) throw new Exception("Значение z =" + z + "выходит за пределы");

}

//Формирование ключа

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

{

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

}

//Приведение к строке

public override string ToString()

{

//Класс StringBuilder используется для построения длинных строк

//Это увеличивает производительность по сравнению с созданием и склеиванием

//большого количества обычных строк

StringBuilder d = new StringBuilder();

d.Append("Вывод плоскостей XY при фиксированных значениях Z\n");

for (int k = 0; k < this.max\_z; k++)

{

d.Append("z=" + k + "\n");

for (int j = 0; j < this.max\_y; j++)

{

d.Append("[");

for (int i = 0; i < this.max\_x; i++)

{

if (i > 0) d.Append("\t");

F temp = this[i, j, k];

if (temp != null)

{

d.Append(temp.ToString());

}

else

{

d.Append("-");

}

}

d.Append("]\n");

}

}

return d.ToString();

}

}

}

////////////////////////////////////////////////////////////////

* using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

// Элемент списка

public class SimpleListItem<F>

{

// Данные

public F data { get; set; }

// Следующий элемент

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

//конструктор

public SimpleListItem(F parametr)

{

this.data = parametr;

}

}

// Список

public class SimpleList<F>

where F : IComparable

{

// Первый элемент списка

protected SimpleListItem<F> first = null;

// Последний элемент списка

protected SimpleListItem<F> last = null;

// Количество элементов

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

// Добавление элемента

public void Add(F element)

{

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

this.Count++;

//Добавление первого элемента

if (last == null)

{

this.first = newItem;

this.last = newItem;

}

//Добавление следующих элементов

else

{

//Присоединение элемента к цепочке

this.last.next = newItem;

//Просоединенный элемент считается последним

this.last = newItem;

}

}

// Чтение контейнера с заданным номером

public SimpleListItem<F> GetItem(int number)

{

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

{

//Можно создать собственный класс исключения

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

}

SimpleListItem<F> current = this.first;

int i = 0;

//Пропускаем нужное количество элементов

while (i < number)

{

//Переход к следующему элементу

current = current.next;

//Увеличение счетчика

i++;

}

return current;

}

// Чтение элемента с заданным номером

public F Get(int number)

{

return GetItem(number).data;

}

}

}

///////////////////////////////////////////////////////

* using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

class Stack<F> : SimpleList<F> where F : IComparable

{

// Добавление в стэк

public void Push(F element)

{

//Добавление в конец списка уже реализовано

Add(element);

}

// Удаление и чтение из стека

public F Pop()

{

//default(T) - значение для типа T по умолчанию

F Result = default(F);

if (this.Count == 0) return Result;

//Если элемент единственный

if (this.Count == 1)

{

//то из него читаются данные

Result = this.first.data;

this.first = null;

this.last = null;

}

else

{

//В списке более одного элемента

//Поиск предпоследнего элемента

SimpleListItem<F> newLast = this.GetItem(this.Count - 2);

//Чтение значения из последнего элемента

Result = newLast.next.data;

//предпоследний элемент считается последним

this.last = newLast;

//последний элемент удаляется из списка

newLast.next = null;

}

//Уменьшение количества элементов в списке

this.Count--;

return Result;

}

}

}

**Экранные формы с примерами выполнения программы**

