**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчёт по лабораторной работе №3

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31 |  | преподаватель каф. ИУ5 |
| Олюшкин М. К. |  | Гапанюк Ю. Е. |
| Подпись и дата: |  | Подпись и дата: |

# Постановка задачи:

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:
   * 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.

# Код программы

* Circle.cs

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;

return result;

}

public void Print()

{

Console.WriteLine(this.ToString ());

} } }

* IPrint.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

interface IPrint

{

void Print();

}

}

* Matrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

public class Matrix<T>

{

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

int maxX;

int maxY;

int maxZ;

T nullElement;

// Конструктор

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

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.nullElement = nullElementParam;

}

// Индексатор для доступа к данных

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

{

get

{

CheckBounds(x, y, z);

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

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

{

return this.\_matrix[key];

}

else

{

return this.nullElement;

}

}

set

{

CheckBounds(x, y, z);

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

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

}

}

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

{

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

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

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

}

// 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 b = new StringBuilder();

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

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

{

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

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

{

b.Append("[");

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

{

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

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

if (temp != null)

{

b.Append(temp.ToString());

}

else

{

b.Append("-");

}

}

b.Append("]\n");

}

}

return b.ToString();

}

}

}

* Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace Laba\_2

{

abstract class Figure : IComparable

{

public string Type

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

string \_Type;

public abstract double Area();

public override string ToString()

{

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

}

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(1, 2);

Square square = new Square(3);

Circle circle = new Circle(4);

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();

}

}

}

* Rectangle.cs

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 ());

}

}

}

* SimpleListProjects.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

public class SimpleListItem<T>

public T data { get; set; }

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

public SimpleListItem(T param)

{

this.data = param;

}

}

public class SimpleList<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;

//Пропускаем нужное количество элементов

while (i < number)

{

current = current.next;

i++;

}

return current;

}

// Чтение элемента с заданным номером

public T Get(int number)

{

return GetItem(number).data;

}

}

}

* Square.cs

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 ());

}

}

}

* Stack.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Laba\_2

{

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

{

// Добавление в стэк

public void Push(T element)

{

//Добавление в конец списка уже реализовано

Add(element);

}

// Удаление и чтение из стека

public T Pop()

{

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

T Result = default(T);

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<T> 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;

}

}

}

# Скриншот результата:

