**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

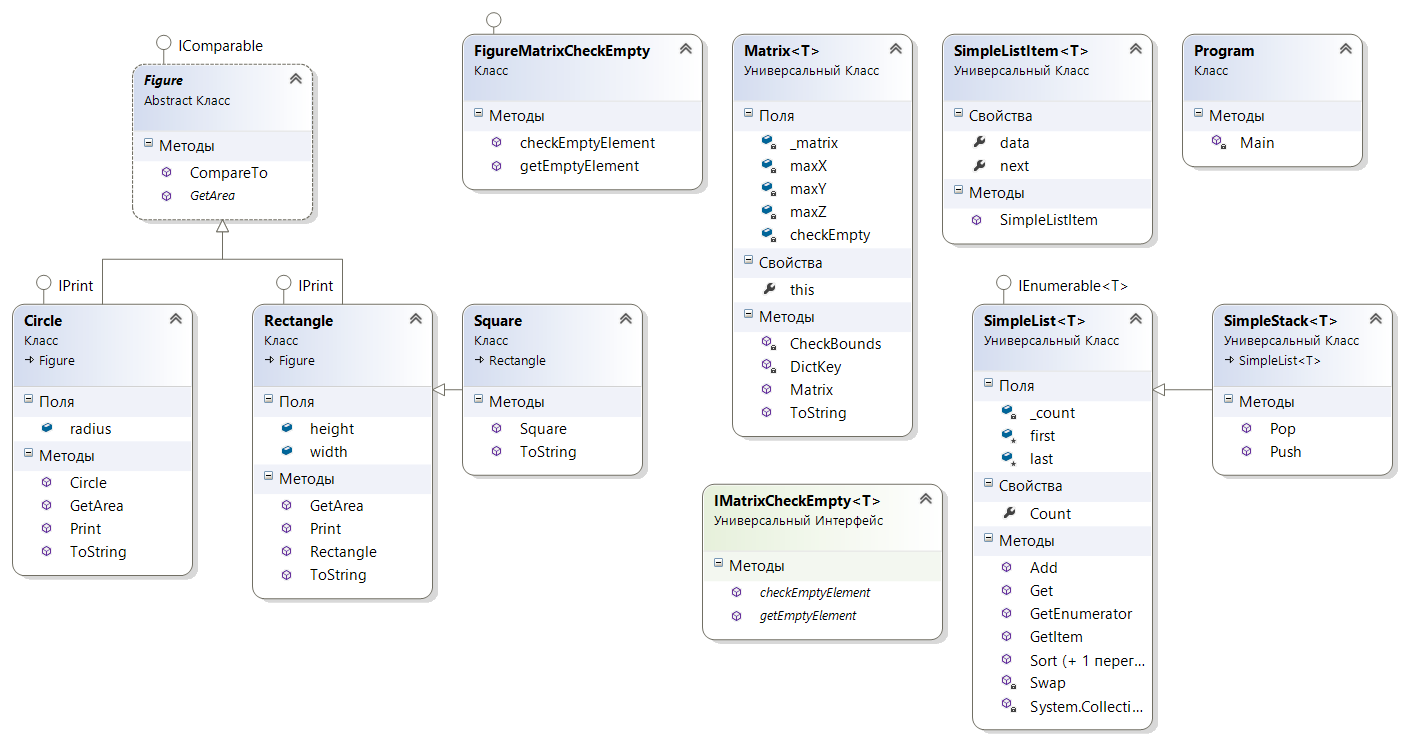
|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Храмцов Денис |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

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:
   * 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**

***Файл IPrint.cs***

using System;

namespace Lab2

{

interface IPrint

{

void Print();

}

}

***Файл IMatrixCheckEmpty.cs***

using System;

namespace Lab3

{

/// <summary>

/// Проверка пустого элемента матрицы

/// </summary>

public interface IMatrixCheckEmpty<T>

{

/// <summary>

/// Возвращает пустой элемент

/// </summary>

T getEmptyElement();

/// <summary>

/// Проверка что элемент является пустым

/// </summary>

bool checkEmptyElement(T element);

}

}

***Файл FigureMatrixCheckEmpty.cs***

using System;

namespace Lab3

{

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; }

}

}

***Файл MatrixCheckEmpty.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab3

{

public class Matrix<T>

{

/// <summary>

/// Словарь для хранения значений

/// </summary>

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

/// <summary>

/// Количество элементов по длине (максимальное количество столбцов)

/// </summary>

int maxX;

/// <summary>

/// Количество элементов по ширине (максимальное количество строк)

/// </summary>

int maxY;

/// <summary>

/// Количество элементов по высоте (максимальное количество строк)

/// </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()

{

//Класс StringBuilder используется для построения длинных строк

//Это увеличивает производительность по сравнению с созданием и склеиванием

//большого количества обычных строк

StringBuilder b = new StringBuilder();

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

{

b.Append("[");

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

{

b.Append("[");

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

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t");

}

//Если текущий элемент не пустой

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

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}

b.Append("]");

}

b.Append("]\n");

}

return b.ToString();

}

}

}

***Файл Figure.cs***

using System;

namespace Lab3

{

abstract class Figure:IComparable

{

// Возвращает:

// Значение, указывающее, каков относительный порядок сравниваемых объектов.

// Возвращаемые значения представляют следующие результаты сравнения. Значение

// Значение Меньше нуля Данный экземпляр в порядке сортировки следует перед

// obj. Zero Данный экземпляр имеет ту же позицию в порядке сортировки, что

// и объект obj. Больше нуля. Данный экземпляр в порядке сортировки следует

// после obj.

//

/// <summary>

/// Возвращает площадь фигуры

/// </summary>

/// <returns>Площадь</returns>

abstract public double GetArea();

public int CompareTo(object o)

{

Figure obj = (Figure)o;

if (this.GetArea() > obj.GetArea()) return -1;

else if (this.GetArea() < obj.GetArea()) return 1;

else return 0;

}

}

}

***Файл Rectangle.cs***

using System;

namespace Lab3

{

class Rectangle : Figure, IPrint

{

public double width, height;

public Rectangle(double a, double b) { this.width = a; this.height = b; }

override public double GetArea()

{

return width \* height;

}

public override string ToString()

{

return "Прямоугольник. Длина:" + height + " Ширина:" + width + " Площадь:" + GetArea();

}

public void Print() { Console.WriteLine(ToString()); }

}

}

***Файл Square.cs***

using System;

namespace Lab3

{

class Square : Rectangle

{

public Square(double a) : base(a, a) { }

public override string ToString()

{

return "Квадрат. Длина:" + width + " Площадь:" + GetArea();

}

}

}

***Файл Circle.cs***

using System;

namespace Lab3

{

class Circle : Figure, IPrint

{

public double radius;

public Circle(double r) { this.radius = r; }

override public double GetArea()

{

return 2 \* Math.PI \* Math.Pow(radius, 2);

}

public override string ToString()

{

return "Круг. Радиус:" + radius + " Площадь:" + GetArea();

}

public void Print() { Console.WriteLine(ToString()); }

}

}

***Файл SimpleList.cs***

using System;

using System.Collections.Generic;

namespace Lab3

{

/// <summary>

/// Список

/// </summary>

public class SimpleList<T> : IEnumerable<T>

where T : IComparable

{

/// <summary>

/// Первый элемент списка

/// </summary>

protected SimpleListItem<T> first = null;

/// <summary>

/// Последний элемент списка

/// </summary>

protected SimpleListItem<T> last = null;

/// <summary>

/// Количество элементов

/// </summary>

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

/// <summary>

/// Добавление элемента

/// </summary>

public void Add(T element)

{

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

}

}

/// <summary>

/// Чтение контейнера с заданным номером

/// </summary>

public SimpleListItem<T> GetItem(int number)

{

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

{

//Можно создать собственный класс исключения

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

}

SimpleListItem<T> 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;

}

/// <summary>

/// Чтение элемента с заданным номером

/// </summary>

public T Get(int number)

{

return GetItem(number).data;

}

/// <summary>

/// Для перебора коллекции

/// </summary>

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

//Перебор элементов

while (current != null)

{

//Возврат текущего значения

yield return current.data;

//Переход к следующему элементу

current = current.next;

}

}

//Реализация обобщенного IEnumerator<T> требует реализации необобщенного интерфейса

//Данный метод добавляется автоматически при реализации интерфейса

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

/// <summary>

/// Cортировка

/// </summary>

public void Sort()

{

Sort(0, this.Count - 1);

}

/// <summary>

/// Алгоритм быстрой сортировки

/// </summary>

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);

}

/// <summary>

/// Вспомогательный метод для обмена элементов при сортировке

/// </summary>

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;

}

}

}

***Файл SimpleListItem.cs***

using System;

namespace Lab3

{

/// <summary>

/// Элемент списка

/// </summary>

public class SimpleListItem<T>

{

/// <summary>

/// Данные

/// </summary>

public T data { get; set; }

/// <summary>

/// Следующий элемент

/// </summary>

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

///конструктор

public SimpleListItem(T param)

{

this.data = param;

}

}

}

***Файл SimpleStack.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

public class SimpleStack<T>: SimpleList<T> where T:IComparable

{

public void Push(T obj)

{

Add(obj);

}

public T Pop()

{

T Result = default(T);

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

if (this.Count == 1)

{

return this.first.data;

this.first = null;

this.last = null;

this.Count--;

}

else

{

Result = this.last.data;

GetItem(this.Count - 2).next = null;

this.last = GetItem(this.Count - 2);

this.Count--;

return Result;

}

}

}

}

***Файл Program.cs***

using System;

namespace Lab2

{

class Program

{

static void Main(string[] args)

{

Square s = new Square(10);

s.Print();

Rectangle r = new Rectangle(1, 3);

r.Print();

Circle gf = new Circle(10);

Console.WriteLine(gf.ToString());

gf.Print();

Console.ReadKey();

}

}

}

**Пример выполнения программы**

