Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Московский государственный технический университет имени Н.Э.Баумана

(МГТУ им. Н.Э.Баумана)

**Отчет по лабораторной работе №3**

**по курсу БКИТ**

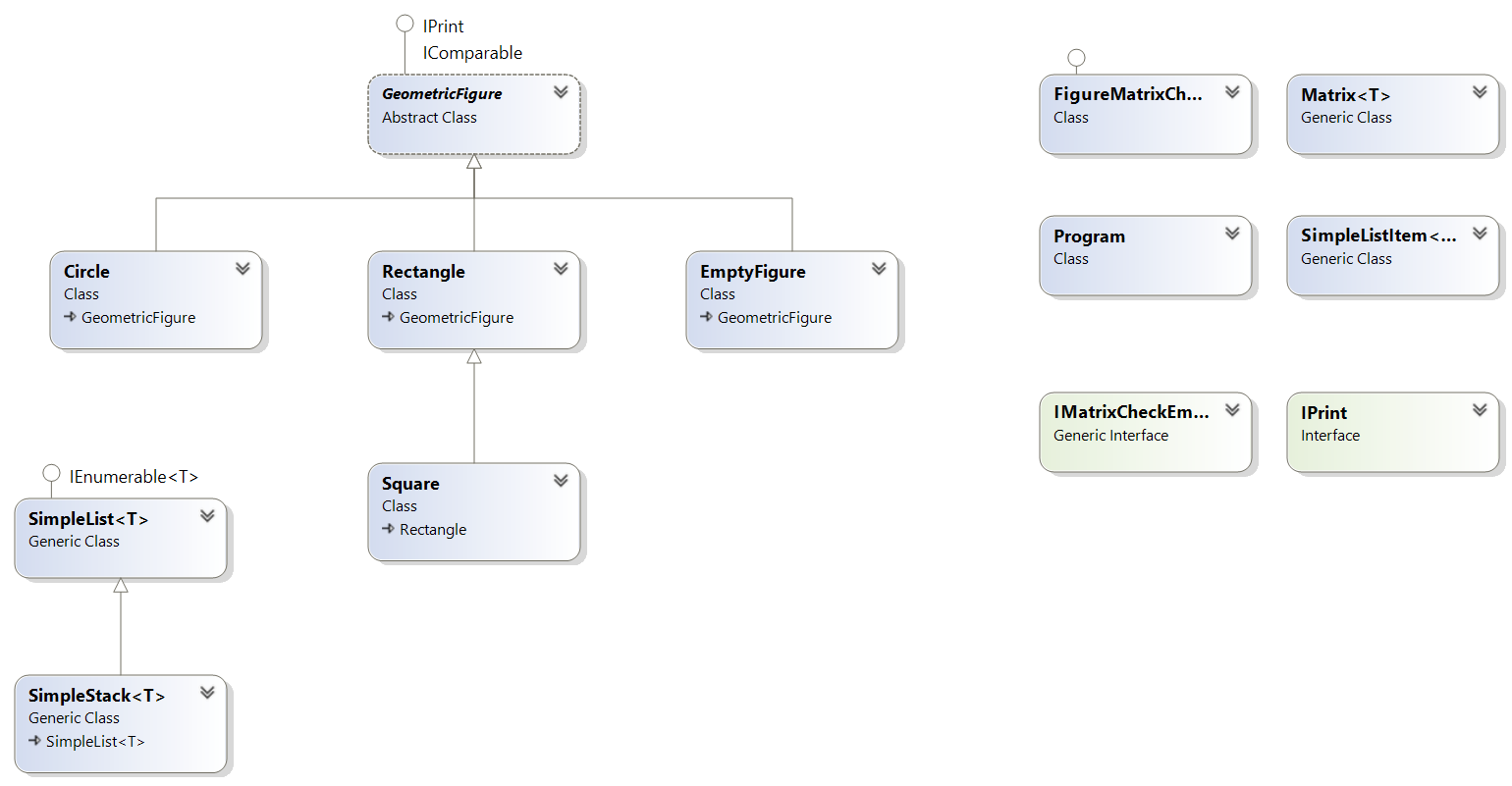
Выполнил: Труфанов Вадим, ИУ 5-33

Преподаватель: Гапанюк Ю.Е.

Москва, 2018

1. **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.
9. ****Диаграмма классов**
10. **Текст программы**

*Класс Program*

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

class Program

{

static void Main(string[] args)

{

Rectangle rect = new Rectangle(3, 4);

Circle circ = new Circle(4);

Square square = new Square(5);

Circle circ2 = new Circle(6);

Rectangle rect2 = new Rectangle(5, 6);

//ARRAYLIST

Console.WriteLine("\nArrayList");

ArrayList al = new ArrayList();

al.Add(circ);

al.Add(rect);

al.Add(square);

al.Add(rect2);

al.Add(circ2);

Console.WriteLine("\nArrayList - до сортировки");

foreach (var figure in al)

Console.WriteLine(figure);

Console.WriteLine("\nArrayList - сортировка");

al.Sort();

foreach (var figure in al)

Console.WriteLine(figure);

//LIST

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

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

fl.Add(circ);

fl.Add(rect);

fl.Add(square);

fl.Add(rect2);

fl.Add(circ2);

Console.WriteLine("\nПеред сортировкой:");

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

fl.Sort();

Console.WriteLine("\nПосле сортировки:");

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

//MATRIX

Console.WriteLine("\nМатрица");

Matrix<GeometricFigure> matrix = new Matrix<GeometricFigure>(5, 5, 5, new FigureMatrixCheckEmpty());

matrix[3,0, 0] = rect;

matrix[1, 1,4] = square;

matrix[2, 2,2] = circ;

matrix[3, 2, 1] = circ2;

matrix[0, 4, 3] = rect2;

Console.WriteLine(matrix.ToString());

//Stack

Console.WriteLine("\nСтек");

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

stack.Push(rect);

stack.Push(circ2);

stack.Push(square);

stack.Push(rect2);

stack.Push(circ);

while (stack.Count > 0)

{

GeometricFigure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadLine();

}

}

}

*Класс Circle*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

class Circle : GeometricFigure

{

private double rad;

public double Rad { get => rad; set => rad = value; }

public Circle(double radius) {

Rad = radius;

}

public override string ToString()

{

return "Circle, radius = " + Rad + ", area = " + calcArea();

}

public override double calcArea()

{

return Math.PI \* Rad \* Rad;

}

}

}

*Класс EmptyFigure*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

class EmptyFigure : GeometricFigure

{

public override double calcArea()

{

return 0;

}

}

}

*Класс FigureMatrixCheckEmpty*

using System;

namespace Lab3

{

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<GeometricFigure>

{

/// <summary>

/// В качестве пустого элемента возвращается null

/// </summary>

public GeometricFigure getEmptyElement()

{

return null;

}

/// <summary>

/// Проверка что переданный параметр равен null

/// </summary>

public bool checkEmptyElement(GeometricFigure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

}

*Класс GeometricFigure*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

public abstract class GeometricFigure: IPrint, IComparable

{

public abstract double calcArea();

public void Print()

{

Console.WriteLine(this.ToString());

}

public int CompareTo(object obj)

{

GeometricFigure figure = (GeometricFigure) obj;

//Сравнение

if (this.calcArea() < figure.calcArea()) return -1;

else if (this.calcArea() > figure.calcArea()) return 1;

else return 0;

}

}

}

*Интерфейс IMatrixCheckEmpty*

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);

}

}

*Интерфейс IPrint*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

public interface IPrint

{

void Print();

}

}

*Класс Matrix*

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 горизонтали (максимальное количество столбцов)

/// </summary>

int maxX;

/// <summary>

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

/// </summary>

int maxY;

int maxZ;

/// <summary>

/// Реализация интерфейса для проверки пустого элемента

/// </summary>

IMatrixCheckEmpty<T> сheckEmpty;

/// <summary>

/// Конструктор

/// </summary>

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

{

this.maxZ = pz;

this.maxX = px;

this.maxY = py;

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 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++)

{

//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("]\n");

}

}

return b.ToString();

}

}

}

*Класс Recatangle*

using System;

namespace Lab3

{

public class Rectangle : GeometricFigure

{

private double a, b;

public Rectangle(double a, double b) {

A = a;

B = b;

}

public double A { get => a; set => a = value; }

public double B { get => b; set => b = value; }

public override double calcArea()

{

return A \* B;

}

public override string ToString()

{

return "Rectangle, a = " + A + ",b = " + B+",area = " + calcArea();

}

}

}

*Класс SimpleList*

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;

}

}

}

*Класс SimpleListItem*

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;

}

}

}

*Класс SimpleStack*

using System;

namespace Lab3

{

/// <summary>

/// Класс стек

/// </summary>

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

{

/// <summary>

/// Добавление в стек

/// </summary>

public void Push(T element)

{

//Добавление в конец списка уже реализовано

Add(element);

}

/// <summary>

/// Удаление и чтение из стека

/// </summary>

public T Pop()

{

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

T Result = default(T);

//Если стек пуст, возвращается значение по умолчанию для типа

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

//Если элемент единственный

if (this.Count == 1)

{

//то из него читаются данные

Result = this.first.data;

//обнуляются указатели начала и конца списка

this.first = null;

this.last = null;

}

//В списке более одного элемента

else

{

//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--;

//Возврат результата

return Result;

}

}

}

*Класс Square*

using System;

namespace Lab3

{

public class Square : Rectangle

{

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

public override string ToString()

{

return "Square, " + "a = "+ A+",area = "+calcArea();

}

}

}

1. **Экранные формы с примерами выполнения программы**

