**Министерство науки и высшего образования Российской Федерации** **Федеральное государственное бюджетное образовательное учреждение высшего образования** **«Московский государственный технический университет** **имени Н. Э. Баумана** **(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Радиотехнический»**

**Кафедра «Системы обработки информации и управления»**

Отчет по лабораторной работе №3

по курсу БКИТ

Выполнила:

Студентка Группы РТ5-31Б   
Веденеева Екатерина Ильинична

подпись, дата

Проверил:

подпись, дата

2020г.

1. **Задание**

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.
9. **Код**

**Файл Program.cs**

using System;

using System.Linq;

using System.Text;

using System.Collections;

using System.Collections.Generic;

namespace Lab3

{

interface IPrint

{

void Print();

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Веденеева Екатерина РТ5-31Б Лабораторная 2");

Rectangle r = new Rectangle(2, 4);

Square s = new Square(4);

Circle c = new Circle(1);

/\*

r.Print();

s.Print();

c.Print();

\*/

var L = new List<Figure> { r, s, c };

foreach (Figure n in L)

Console.WriteLine(n.ToString());

Console.WriteLine( " ");

L.Sort();

//Console.WriteLine("Сортировка необобщенного списка");

foreach (Figure n in L)

Console.WriteLine(n.ToString());

Console.WriteLine("\nМатрица");

Matrix<Figure> matrix = new Matrix<Figure>(3, 3, 3,new FigureMatrixCheckEmpty());

matrix[0, 0,0] = r;

matrix[1, 1,0] = s;

matrix[2, 2,0] = c;

Console.WriteLine(matrix.ToString());

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

//добавление данных в стек

stack.Push(r);

stack.Push(s);

stack.Push(c);

//чтение данных из стека

while (stack.Count > 0)

{

Figure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadLine();

}

}

}

**Файл SimpleListItem.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

public class SimpleListItem<T> where T: IComparable<T>

{

public T data { get; set; }

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

public SimpleListItem(T param)

{

this.data = param;

}

}

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

{

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

{

//Присоединение элемента к цепочке

this.last.next = newItem;

//Присоединенный элемент считается последним

this.last = newItem;

}

}

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;

}

public T Get(int number)

{

return GetItem(number).data;

}

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;

}

}

System.Collections.IEnumerator

System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public void Sort()

{

Sort(0, this.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;

}

}

}

**Файл Circle.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

/// <summary>

/// Класс "Круг", наследник Г.Ф.

/// </summary>

class Circle : Figure, IPrint

{

public Circle(int r)

{

Radius = r;

this.Type = "Круг";

}

private int \_radius;

public int Radius

{

get { return \_radius; }

set { \_radius = value; }

}

public override double Calculate\_area()

{

return 3.14 \* Radius \* Radius;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

**Файл SimpleStack.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

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

{

public void Push(T element)

{

Add(element);

}

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;

this.first = null;

this.last = null;

}

else

{

//Поиск предпоследнего элемента

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

//Чтение значения из последнего элемента

Result = newLast.next.data;

this.last = newLast;

newLast.next = null;

}

this.Count--;

return Result;

}

}

}

**Файл Square.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

/// <summary>

/// Класс "Квадрат", наследник "Прямоугольника"

/// </summary>

class Square : Rectangle, IPrint

{

public Square(int l) : base(l, 0)

{

Height = l;

Width = l;

this.Type = "Квадрат";

}

new public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

**Файл Rectangle.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

/// <summary>

/// Класс "Прямоугольник", наследник Г.ф.

/// </summary>

class Rectangle : Figure, IPrint

{

public Rectangle(int h, int w)

{

Height = h;

Width = w;

this.Type = "Прямоугольник";

}

private int \_height;

public int Height

{

get { return \_height; }

set { \_height = value; }

}

private int \_width;

public int Width

{

get { return \_width; }

set { \_width = value; }

}

public override double Calculate\_area()

{

return Height \* Width;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

**Файл figure.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

abstract class Figure : IComparable<Figure>

{

public int CompareTo(Figure o)

{

Figure f = o as Figure;

//Если площадь текущего элемента больше площади сравнваемого элемента, текущий должен стояь перед

if (this.Calculate\_area() > f.Calculate\_area())

return -1;

if (this.Calculate\_area() < f.Calculate\_area())

return 1;

if (this.Calculate\_area() == f.Calculate\_area())

return 0;

else

throw new Exception("Невозможно сравнить два объекта");

}

/// <summary>

/// Абстрактный клас "Геометрическая фгура"

/// </summary>

/// <summary>

/// Тип фигуры

/// </summary>

public string Type

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

private string \_Type;

public virtual double Calculate\_area()

{

return 0;

}

/// <summary>

/// Приведение к строке, переопределение метода Object

/// </summary>

public override string ToString()

{

return this.Type + " площадью " +

this.Calculate\_area().ToString();

}

}

}

**Файл Matrix.cs**

using System.Collections.Generic;

using System.Linq;

using System.Text;

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;

}

}

}

public interface IMatrixCheckEmpty<T>

{

/// <summary>

/// Возвращает пустой элемент

/// </summary>

T getEmptyElement();

/// <summary>

/// Проверка что элемент является пустым

/// </summary>

bool checkEmptyElement(T element);

}

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;

/// <summary>

/// Количество элементов по вертикали .....(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()

{

StringBuilder b = new StringBuilder();

Console.WriteLine("XY");

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

{

int k = 0;

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");

}

Console.WriteLine("XZ");

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

{

int k = 0;

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, k, j]))

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}

b.Append("]\n");

}

Console.WriteLine("ZY");

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

{

int k = 0;

b.Append("[");

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

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t");

}

//Если текущий элемент не пустой

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

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}

b.Append("]\n");

}

return b.ToString();

}

}

**3. Результат выполнения программы**

