**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

Факультет «Информатика и вычислительная техника»

Кафедра ИУ5 «Системы обработки информации и управления»

**Отчет по лабораторной работе № 3**

Задание: разработать программу, реализующую работу с коллекциями

Студент Левин Михаил Александрович

*фамилия, имя, отчество*

Группа ИУ5-32Б

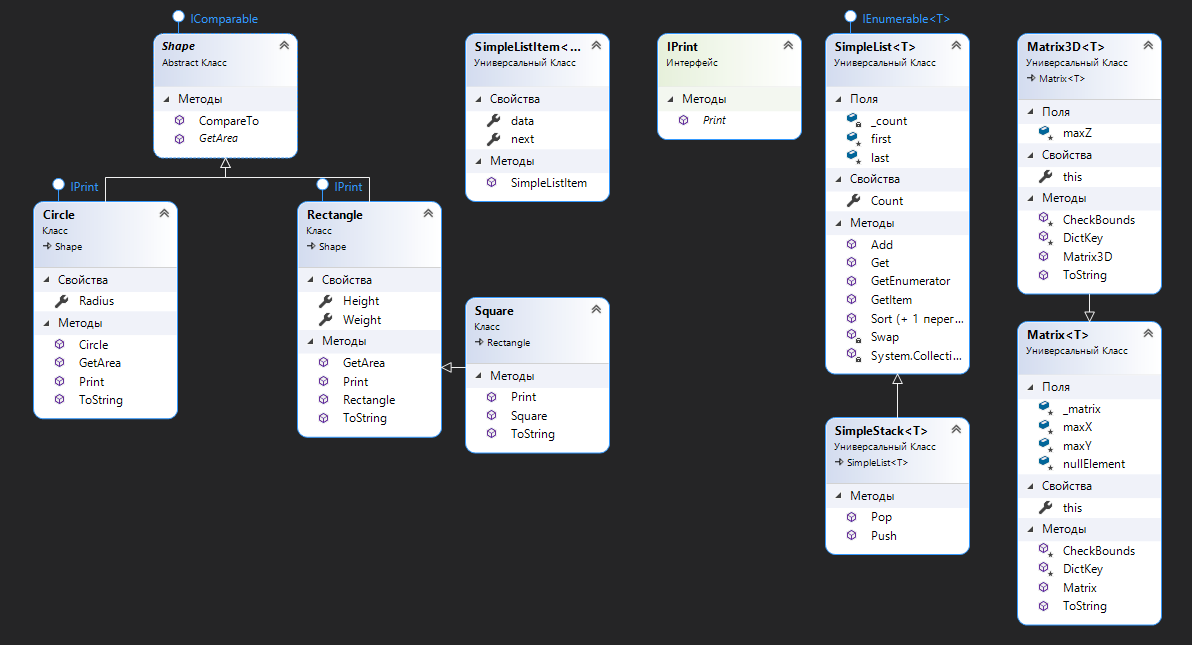
Москва, 2020 г.

# 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.

# Диаграмма классов



# Текст программы

**IPrint.cs**

namespace Lab3

{

interface IPrint

{

void Print();

}

}

**Matrix.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

public class Matrix<T>

{

/// <summary>

/// Словарь для хранения значений

/// </summary>

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

/// <summary>

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

/// </summary>

protected int maxX;

/// <summary>

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

/// </summary>

protected int maxY;

/// <summary>

/// Пустой элемент, который возвращается если элемент с нужными

///координатами не был задан

/// </summary>

protected T nullElement;

/// <summary>

/// Конструктор

/// </summary>

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

{

this.maxX = px;

this.maxY = py;

this.nullElement = nullElementParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

public T this[int x, int y]

{

get

{

CheckBounds(x, y);

string key = DictKey(x, y);

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

{

return this.\_matrix[key];

}

else

{

return this.nullElement;

}

}

set

{

CheckBounds(x, y);

string key = DictKey(x, y);

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

}

}

/// <summary>

/// Проверка границ

/// </summary>

protected void CheckBounds(int x, int y)

{

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");

}

/// <summary>

/// Формирование ключа

/// </summary>

protected string DictKey(int x, int y)

{

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

{

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

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

}

b.Append("]\n");

}

return b.ToString();

}

}

}

**Matrix3D.cs**

using System;

using System.Text;

namespace Lab3

{

class Matrix3D<T>:Matrix<T>

{

/// <summary>

/// Количество элементов по оси Z

/// </summary>

protected int maxZ;

public Matrix3D(int px, int py, int pz, T nullElementParam) : base(px, py, nullElementParam)

{

maxZ = pz;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

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);

}

}

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

{

CheckBounds(x,y);

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

}

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

{

return $"{DictKey(x, y)}\_{z.ToString()}";

}

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 x = 0; x < this.maxX; x++)

{

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

{

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

{

b.Append($"[{x},{y},{z}] = {this[x,y,z]}\n");

}

}

}

return b.ToString();

}

}

}

**Rectangle.cs**

using System;

namespace Lab3

{

/// <summary>

/// Класс прямоугольник

/// </summary>

class Rectangle:Shape,IPrint

{

public double Height { get; set; }

public double Weight { get; set; }

public override double GetArea()

{

return Height \* Weight;

}

public Rectangle(double w, double h)

{

Weight = w;

Height = h;

}

public override string ToString()

{

return $"Прямоугольник. Ширина: {Weight}. Высота: {Height}. Площадь: {GetArea()}";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

**Shape.cs**

using System;

namespace Lab3

{

/// <summary>

/// Абстрактный класс геометрическая фигура

/// </summary>

abstract class Shape:IComparable

{

/// <summary>

/// Метод вычисления площади

/// </summary>

/// <returns></returns>

public abstract double GetArea();

public int CompareTo(object obj)

{

double areaCurrent = GetArea();

double areaObj = (obj as Shape).GetArea();

if (areaCurrent > areaObj) return 1;

if (areaCurrent < areaObj) return -1;

return 0;

}

}

}

**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>

/// <param name="element"></param>

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 обощенного 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>

/// Реализация алгоритма быстрой сортировки

/// </summary>

/// <param name="low"></param>

/// <param name="high"></param>

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;

}

}

}

**SimpleListItem.cs**

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;

}

}

}

**SimpleStack.cs**

using System;

namespace Lab3

{

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

{

public void Push(T element)

{

Add(element); //добавляем новый элемент

}

public T Pop()

{

var current = GetItem(Count - 1); //забираем последний элемент

Count--; //уменьшаем количетво элементов

//если ещё остались элементы

if (Count > 0)

{

//находим предыдущий

var prevision = GetItem(Count - 1);

//предыдущий становится последним (убираем извлеченный элемент)

last = prevision;

}

else //если элементов больше не осталось - обнулям ссылки на начало и конец списка

{

last = null;

first = null;

}

//возвращем извлеченный элемент

return current.data;

}

}

}

**Square.cs**

using System;

namespace Lab3

{

/// <summary>

/// Квадрат

/// </summary>

class Square:Rectangle

{

public Square(double l):base(l,l)

{

}

public override string ToString()

{

return $"Квадрат. Длина стороны: {Weight}. Площадь: {GetArea()}";

}

public new void Print()

{

Console.WriteLine(ToString());

}

}

}

**Program.cs**

using System;

using System.Collections;

using System.Collections.Generic;

namespace Lab3

{

class Program

{

static void Main(string[] args)

{

Console.Write("Введите длину стороны квадрата: ");

double l = double.Parse(Console.ReadLine());

var square = new Square(l);

Console.Write("Введите ширину прямоугольника: ");

double w = double.Parse(Console.ReadLine());

Console.Write("Введите ширину высоту: ");

double h = double.Parse(Console.ReadLine());

var rectangle = new Rectangle(w, h);

Console.Write("Введите радиус круга: ");

double r = double.Parse(Console.ReadLine());

var circle = new Circle(r);

ArrayList figures = new ArrayList {square, rectangle, circle };

Console.WriteLine("\nНеотсортированный ArrayList: ");

foreach (var figure in figures)

{

Console.WriteLine(figure.ToString());

}

figures.Sort();

Console.WriteLine("\nОтсортированный ArrayList: ");

foreach (var figure in figures)

{

Console.WriteLine(figure.ToString());

}

List<Shape> shapes = new List<Shape> {square, rectangle, circle };

Console.WriteLine("\nНеотсортированный List<Shape>");

foreach (var shape in shapes)

{

Console.WriteLine(shape.ToString());

}

shapes.Sort();

Console.WriteLine("\nОтсортированный List<Shape>");

foreach (var shape in shapes)

{

Console.WriteLine(shape.ToString());

}

/\*матрица\*/

Console.WriteLine("\nМатрица 3D:");

Matrix3D<Shape> cube = new Matrix3D<Shape>(3, 3, 3, null);

cube[0, 0, 0] = rectangle;

cube[1, 1, 1] = square;

cube[2, 2, 2] = circle;

Console.WriteLine(cube.ToString());

/\*SimpleStack\*/

Console.WriteLine("\nСтек: ");

var stack = new SimpleStack<Shape>();

stack.Push(square);

stack.Push(rectangle);

stack.Push(circle);

while (stack.Count > 0)

{

Shape shape = stack.Pop();

Console.WriteLine(shape);

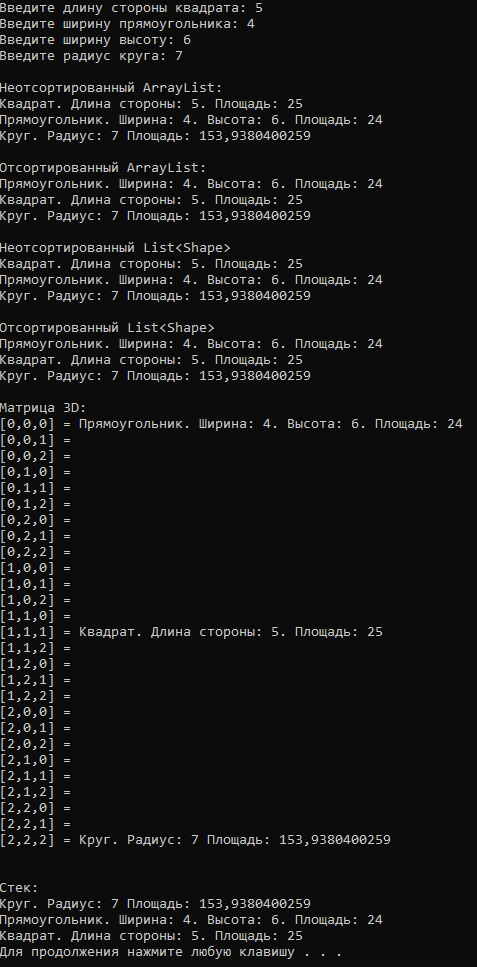
}

}

}

}

# Экранные формы



# Контрольные вопросы:

1. В чем сходство и различие между обобщенными и необобщенными коллекциями?

При работе с необобщёнными коллекциями происходит трата ресурсов на упаковку и распаковку объектов, так как объекты хранятся в коллекции не типизированными. Обобщенные коллекции содержат объекты типа Т.

1. Как осуществляется работа с кортежем?

Кортежи предоставляют краткий синтаксис для группирования нескольких элементов данных в упрощенную структуру данных. Чаще всего кортежи используются как возвращаемый методом тип. То есть вместо определения параметров метода out можно сгруппировать результаты метода в возвращаемый тип кортежа. Имена полей кортежей указываются явным образом либо в выражении инициализации кортежа, либо в определении типа кортежа.

1. Как осуществляется сортировка коллекций?

Для сортировки коллекции используется вызов метода Sort. Класс коллекции должен реализовывать интерфейс IComparable, который требует реализации метода CompareTo, в теле которого и происходит сравнение двух элементов по определенным пользователем правилам.

1. Как можно реализовать класс разреженной матрицы на основе класса словаря?

Для реализации хранения элементов разреженной матрицы на основе класса словаря (Dictionary) в качестве ключа словаря используется адрес хранениях элементов по координатам, представленный в виде строки, а в качестве значения словаря сам элемент типа Т. Это позволяет наращивать количества пространств в дочерних классах не меняя реализацию хранения значений. При изменении количества пространств будет изменять только ключевая строка, хранящая «адрес» элемента.

1. Как можно реализовать классы списка и стека без использования стандартных коллекций?

Пользовательскую реализацию классов стек и список можно организовать на базе отдельно созданного пользовательского класса, который выступать в качестве «ячейки» списка (стека). «Ячейка» должна содержать в себе поля для хранения значения, и ссылку на следующий элемент (в случае односвязного списка). Сам же список (стек) представляет собой отдельный пользовательский класс, который уже агрегирует «ячейки» хранящие начальный элементы списка и конечный, а также хранить количество ячеек списка (стека). Далее уже в методах по изменению списка (стека) осуществляется переход от одной ячейки к другой по полю next.

1. Как осуществляется работа с обобщенным списком?

Обобщенный список представлен классом List<T>. Представляет строго типизированный список объектов, доступных по индексу. Поддерживает методы для поиска по списку, выполнения сортировки и других операций со списками.

1. Как осуществляется работа с необобщенным списком?

Необобщенный список представлен классом ArrayList. Реализует интерфейс IList с помощью массива с динамическим изменением размера по требованию.

1. Как осуществляется работа с обобщенным стеком?

Обобщённый стек представлен классом Stack<T>. Представляет коллекцию переменного размера экземпляров одинакового заданного типа, обслуживаемую по принципу "последним пришел - первым вышел" (LIFO). Метод Push<T> добавляет новый элемент в стек, метод Pop извлекает элемент из стека и возвращает его.

1. Как осуществляется работа с обобщенной очередью?

Обобщенная очередь представлена классом Queue<T>. Представляет коллекцию объектов, основанную на принципе «первым поступил — первым обслужен». Метод Enqueue<T> добавляет элемент в конец очереди, Dequeue() – удаляет элемент из начала очереди и возвращает его.

1. Как осуществляется работа с обобщенным словарем?

Обобщённый словарь представлен классом Dictionary<TKey,TValue>. Добавление записей в словарь осуществляется через вызов метода Add(TKey, TValue), извлечение данные возможно, например, через индексатор по ключевому значению.