МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

**Кафедра ''САПР''**

Отчет о выполненной лабораторной работе № 3  
«Разработка программы, реализующей работу с коллекциями»  
по дисциплине:   
“Объектно-ориентированное программирование”

Выполнили ст. гр. 20ВА1:  
Комаров М.В.

Приняли:  
Гудков А.А.  
Подмарькова Е.М.

Пенза 2023

**Цель работы:** разработать программу, реализующую работу с коллекциями.

**Листинг**:

Console.WriteLine("Rectangle created: ");

Rectangle r = new Rectangle(10, 15);

r.GetArea();

r.Print();

Console.WriteLine();

Console.WriteLine("Square created: ");

Square s = new Square(16);

s.GetArea();

s.Print();

Console.WriteLine();

Console.WriteLine("Circle created: ");

Circle c = new Circle(20);

c.GetArea();

c.Print();

Console.WriteLine();

Console.WriteLine("ArrayList created: ");

List(r, s, c);

Console.WriteLine();

Console.WriteLine("List<Figure> created: ");

List2(r, s, c);

Console.WriteLine();

Console.WriteLine("Matrix created: ");

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

matrix[0, 0, 0] = new Rectangle(2, 3);

matrix[0, 0, 1] = new Square(8);

matrix[0, 1, 0] = new Circle(45);

matrix[2, 2, 0] = new Circle(23);

matrix[1, 0, 1] = new Square(22);

matrix[1, 1, 1] = new Rectangle(33, 1);

Console.Write(matrix.ToString());

Console.WriteLine(" ");

SimpleList<Figure> list1 = new SimpleList<Figure>

{

s,

r,

c

};

foreach (var x in list1) Console.WriteLine(x);

list1.Sort();

Console.WriteLine("\nСортировка списка");

foreach (var x in list1) Console.WriteLine(x);

Console.WriteLine(" ");

void List(Rectangle p, Square k, Circle kr)

{

ArrayList list = new ArrayList() { p.GetArea(), k.GetArea(), kr.GetArea() };

list.Sort();

for (int j = 0; j < 3; j++)

{

Console.WriteLine(list[j]);

}

}

void List2(Rectangle p, Square k, Circle kr)

{

List<double> list = new List<double>() { p.GetArea(), k.GetArea(), kr.GetArea() };

list.Sort();

for (int j = 0; j < 3; j++)

{

Console.WriteLine((double)list[j]);

}

}

interface IPrint

{

void Print();

}

public class Matrix<T>

{

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

int maxX;

int maxY;

int maxZ;

T nullElement;

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

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.nullElement = nullElementParam;

}

void ChekBounds(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 ");

}

public override string ToString()

{

StringBuilder sb = new StringBuilder();

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

{

sb.Append("[\n");

for (int x = 0; x < maxX; x++)

{

sb.Append("\t[");

for (int z = 0; z < maxY; z++)

{

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

if (z > 0)

sb.Append("\t");

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

sb.Append(this[x, y, z].ToString());

else

sb.Append("none");

}

sb.Append("]\n");

}

sb.Append("]\n");

}

return sb.ToString();

}

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

{

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

}

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

{

get

{

ChekBounds(x, y, z);

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

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

return this.\_matrix[key];

else

return this.nullElement;

}

set

{

ChekBounds(x, y, z);

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

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

}

}

}

public class SimpleListItem<T>

{

public T data

{

get;

set;

}

public SimpleListItem<T> next

{

get;

set;

}

public SimpleListItem(T param)

{

data = param;

}

}

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

{

protected SimpleListItem<T> first = null;

protected SimpleListItem<T> last = null;

protected int \_count;

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

public void Add(T element)

{

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

Count++;

if (last == null)

{

first = newItem;

last = newItem;

}

else

{

last.next = newItem;

last = newItem;

}

}

public SimpleListItem<T> GetItem(int number)

{

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

{

throw new Exception("Выход за границу {number}");

}

SimpleListItem<T> current = first;

int i = 0;

while (i < number)

{

current = current.next;

i++;

}

return current;

}

public T Get(int number)

{

return GetItem(number).data;

}

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = first;

while (current != null)

{

yield return current.data;

current = current.next;

}

}

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

{

return GetEnumerator();

}

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;

}

protected 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);

}

}

public void Sort()

{

Sort(0, this.Count - 1);

}

}

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

{

public SimpleStack() : base() { }

public void Add(T element) { }

public void GetItem(int number) { }

public void Get(int number) { }

public void Sort() { }

public void Push(T element)

{

SimpleListItem<T> item = new SimpleListItem<T>(element);

Count++;

if (first == null)

{

first = item;

last = item;

}

else

{

item.next = first;

first = item;

}

}

public T Pop()

{

T tmp = first.data;

if (first.next != null)

first = first.next;

else

first = null;

return tmp;

}

}

//Абстрактный класс фигура с интерфейсом ICombapable

abstract class Figure : IComparable

{

virtual public double GetArea()

{

return 0;

}

public int CompareTo(object obj)

{

if (obj == null) return 1;

Figure otherFigura = obj as Figure;

if (otherFigura != null)

return this.GetArea().CompareTo(otherFigura.GetArea());

else

throw new ArgumentException("Object is not a Temperature");

}

}

//Прямоугольник

class Rectangle : Figure, IPrint

{

double area;

public double Width { get; set; }

public double Height { get; set; }

public Rectangle(double Width, double Height)

{

this.Width = Width;

this.Height = Height;

area = Width \* Height;

}

public override string ToString() { return $"Rectangle | {area}"; }

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

public override double GetArea()

{

area = Width \* Height;

return area;

}

}

//Квадрат

class Square : Rectangle, IPrint

{

double area, lenght;

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

{

lenght = length;

area = lenght \* lenght;

}

public override string ToString() { return $"Squre | {area}"; }

public override double GetArea()

{

area = lenght \* lenght;

return area;

}

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

}

//Круг

class Circle : Figure, IPrint

{

double area;

public double radius { get; set; }

public Circle(double radius)

{

this.radius = radius;

area = radius \* radius \* 3.1415;

}

public override string ToString() { return $"Circle | {area}"; }

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

public override double GetArea()

{

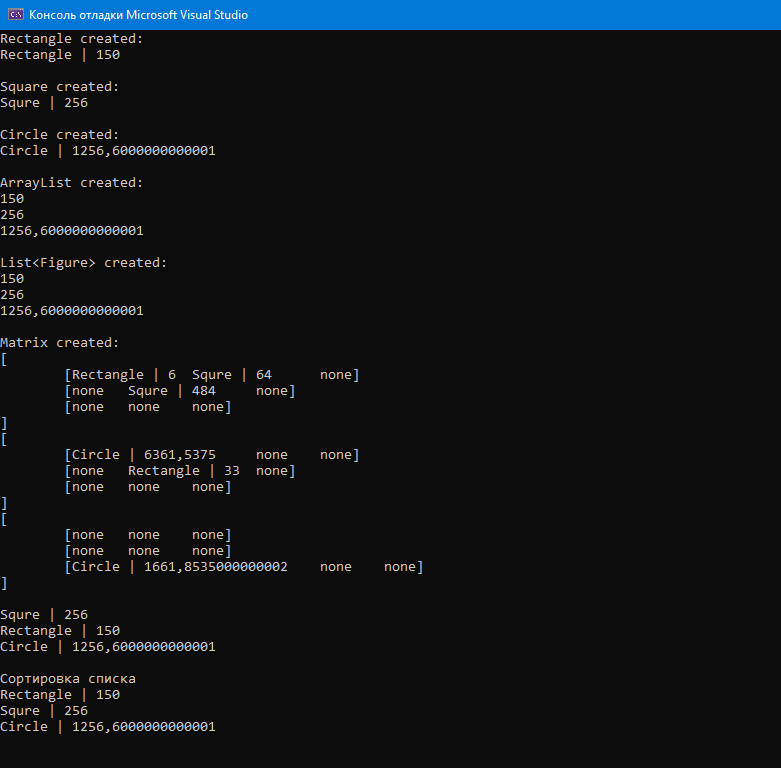
area = radius \* radius \* 3.1415;

return area;

}

}

**Результат программы:**

****

**1. Какие типы коллекций в языке С# Вы знаете?**

*Одно и двунаправленный список, массивы (динамические), список по индексу, стек, очередь.*

**2. Чем отличаются обобщенные и необобщенные коллекции?**

*Обобщённые коллекции оперируют над обобщённым типом данных (<T>). Вместо <T> коллекция может работать с любым типом данных.*

**3. В чем особенность использования коллекционного класса ArrayList?**

*Коллекция класса ArrayList использует динамический массив с обращением по индексу. Он может хранить в себе различные объекты, потому что является обобщённым.*

**4. В чем особенность использования коллекционного класса List?**

*List<T> представляет собой строго типизированный список объектов, доступных по индексу. Он поддерживает методы для поиска по списку, выполнения сортировки и других операций со списками.*

**5. В чем особенность записи и чтения элементов при использовании**

**коллекций ArrayList и List?**

*В коллекции List мы должны указывать тип, который мы используем для чтения и записи (в нашем случае double для площади фигуры).*

**6. Что такое индексаторы и как они используются?**

*Индексаторы позволяют обращаться к элементам внутри класса по индексам, индексов может быть несколько при условии, что они имеют разные типы. К индексаторам можно применять логику* ***get*** *и* ***set****.*