**Лабораторная работа по теме**

«Наследование, перегрузка операторов, обработка исключений»

1) Исследуйте исходный код программы:

1.1 объясните, что такое интерфейс, что он может содержать и чем он отличается от абстрактного класса (case 1-3);

1.2 объясните, как класс Parent реализует интересы InterfaceA, InterfaceB. Обязательно ли перед каждым методов имя интерфейса в классе Parent (case 4)? Что будет, если какой-то методой не реализован?

1.3 поясните тип исключения «NotImplementedException» (case 5);

1.4 зачем нужны абстрактные методы? Что произойдет, если в классе потомка не переопределить абстрактный метод? (case 6);

1.5 зачем нужны виртуальные методы? (case 7);

1.6 возможно ли создать чисто виртуальный метод как в языке C++?

1.7 зачем нужен механизм переопределения метода при наследовании?

1.8 объясните отличия в работе перегруженного метода WriteToFile. Что делает ключевое слово using? (case 8);

1.9 что такое лямбда-выражение? Объясните принцип их работы;

1.10 поясните назначение ключевых слов override и new при их использовании в наследовании;

1.11 объясните механизм обработки исключений;

1.12 что делает ключевое слово base? (case 9);

1.13 на примере case 11-12 объясните отличия в работе методов ToString класса Child (последовательно раскоментируйте и исследуйте каждый из методов);

1.14 зачем нужна перегрузка операторов? В чем отличия от перегрузки операторов в C++ (case 13);

1.15 что такое паттерны проектирования? Зачем нужен паттерн проектирования Singleton? Исследуйте две предложенные реализации этого паттерна, объясните особенности каждой из реализации (case 14).

2) Определить иерархию классов в соответствии с вариантом:

2.1 реализуйте один или несколько интерфейсов, или/и абстрактный класс;

2.2 создайте один из классов бесплодным;

2.3 создайте и переопределите метод записи полей класса в файл (WriteToFile);

2.4 переопределите метод ToString (реализовать на основе лямбда-выражения);

2.5 переопределите методы Equals и GetHashCode;

2.6 при записи в файл выполните обработку исключений (try-catch-finally);

2.7 перегрузите операции (+, <, *);

2.8 продемонстрируйте работу операторов is и as;

2.9 примените паттерн Singleton к одному из классов.

3) Ответьте на дополнительные вопросы:

3.1 кому доступны переменные с модификатором protected на уровне класса?

3.2 наследуются ли переменные с модификатором private?

3.3 поддерживает ли C# множественное наследование?

3.4 можно ли разрешить наследование класса, но запретить перекрытие

3.5 почему нельзя указать модификатор видимости для методов интерфейса?

3.6 назовите отличия между интерфейсом и абстрактным классом.

3.7 для чего используются слова checked и unchecked?

3.8 какой синтаксис нужно использовать в C# для отлова любого возможного исключения?

3.9 будет ли выполнен блок finally, если не было сгенерировано исключение?

3.10 можно ли выполнить несколько блоков catch для одного оператора try?

3.11 назовите стандартные классы исключений и область их использования в .NET.

3.12 за что отвечают методы Dispose(), Finalize()?

3.13 для чего в .NET используется конструкция using(…){…}? Зачем нужен интерфейс IDisposable?

3.14 в чем отличие структуры и класса?

## Варианты:

| Вариант | Задание |
|---|---|
| 1 | Классы – фигура (абстрактный), прямоугольник, элемент управления (интерфейс (ICommand с методами click, move) кнопка (sealed); |
| 2 | Классы – фигура (абстрактный), круг, элемент управления (интерфейс ICommand с методами checked, unchecked), RadioButton (sealed); |
| 3 | Классы – фигура (абстрактный), прямоугольник, элемент управления (интерфейс ICommand с методами show, input), TextBox (sealed); |
| 4 | Классы – бытовая техника (абстрактный), нагревательные приборы (интерфейс IElectrical), Чайник, Утюг, Телевизор (sealed) |
| 5 | Классы – боец (абстрактный класс), всадник, целитель (интерфейс IHealer), друид (sealed); |
| 6 | Классы – транспортное средство (абстрактный), машина, разумное существо (интерфейс ISentient), человек, трансформер (sealed); |

| | |
|---|---|
| 7 | Классы – боец (абстрактный класс), охотник, целитель (интерфейс IHealer), шаман (sealed); |
| 8 | Классы – товар (абстрактный), печатающее устройство (интерфейс IPrinter), сканер (интерфейс IScanner), MFP (МФУ) (sealed) |
| 9 | Классы – техника (абстрактный), самолет, невидимка (интерфейс IInvisible), танк (sealed); |
| 10 | Классы – контейнер (абстрактный класс), итератор (интерфейс ITerator), Array, List, Vector (вектор наследовать от Array и сделать закрытым) |
| 11 | Классы – ПО (абстрактный класс), текстовый процессор, Word (sealed), вирус (интерфейс IVirus), Conficker; |
| 12 | Классы – язык программирования (абстрактный класс), C, C++, ByteCode (интерфейс IByteCode), JAVA, Scala |
| 13 | Классы – фигура (абстрактный класс), прямоугольник, управляемый (интерфейс ICommand с методами move, close, open), Window, MessageBox (sealed); |
| 14 | Классы – фигура (абстрактный класс), прямоугольник, элемент управления (интерфейс ICommand с методами close, resize), Bitmap, Picturebox (sealed); |
| 15 | Классы – фигура (абстрактный класс), круг, изменение размера (интерфейс IResize с методами smaller, bigger), лупа (sealed); |

Исходный код

```
using System;

namespace Lab3
{
    class Program
    {
        static void Main()
        {
            //case 11
            Child child = new Child();
            Console.WriteLine(child.ToString());

            //case 2
            var childObj = (object)child;
            Console.WriteLine(childObj);

            Console.Write("Введите имя файла: ");
            child.FileName = Console.ReadLine();
            child.WriteToFile();
        }
    }

    //case 1
    interface INterfaceA
    {
        void Test();
        int GetInt();
        string FileName { get; set; }
    }
```

```
    //case 2
    interface INterfaceB
    {
        double Test();
        double GetDouble();
    }

    //case 3
    internal abstract class Parent : INterfaceA, INterfaceB
    {
        #region Properties

        public string FileName { get; set; }
        protected Random Rand { get; private set; }

        #endregion

        #region Constructors

        protected Parent()
        {
            FileName = "unknown.txt";
            Rand = new Random();
        }

        #endregion

        #region Methods

        //case 4
        public virtual double GetDouble()
        {
            return Rand.NextDouble();
        }

        //case 4
        void INterfaceA.Test()
        {
            Console.WriteLine("Class Parent: Реализация метода Test интерфеса A");
        }

        //case 4
        double INterfaceB.Test()
        {
            Console.WriteLine("Class Parent: Реализация метода Test интерфеса B");
            //case 5
            throw new NotImplementedException();
        }

        //case 6
        public abstract int GetInt();
        //case 6
        public abstract void MethodAbstract();

        //case 7
        public virtual void MethodVirtual(string str)
        {
            Console.WriteLine("ParentClass: virtual Method");
        }

        //case 8
        public virtual void WriteToFile()
        {
            System.IO.StreamWriter file = null;
            try
```

```
            {
                Console.WriteLine("Write To File: ");
                file = new System.IO.StreamWriter(FileName);
                file.WriteLine("Hello World");
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                Console.WriteLine("Finally: отрабатывает в любом случае");

                if (file != null)
                {
                    file.Close();
                }
            }
        }

        #endregion
    }

    internal class Child : Parent
    {
        #region Properties

        public Int64 Number { get; set; }

        #endregion

        #region Constructors

        //case 9
        public Child() : base()
        {
            Number = GetInt();
        }

        #endregion

        #region Methods

        public override int GetInt()
        {
            return Rand.Next(-10, Int16.MaxValue);
        }

        public void MethodVirtual() { }

        //public new Func<string> ToString = () => { return
DateTime.Now.ToShortTimeString(); };

        //String Interpolation (C# 6)
        //public override string ToString() => $"{this.FileName} + {this.GetInt()}";

        public override string ToString()
        {
            return string.Format("Filename: {0}, Int: {1}", this.FileName, this.GetInt());
        }

        public override void MethodAbstract()
        {
            //case 5
            throw new NotImplementedException();
```

```
        }

        //case 8
        public override void WriteToFile()
        {
            using (System.IO.StreamWriter file = new System.IO.StreamWriter(FileName,
true))
            {
                file.WriteLine("Child: Hello World");
            }
        }

        //case 13
        public static bool operator >(Child first, Child second)
        {
            return first.Number > second.Number;
        }

        //case 13
        public static bool operator <(Child first, Child second) => first.Number <
second.Number;

        #endregion
    }

    //case 14
    public sealed class Singleton
    {
        private static readonly Singleton instance = new Singleton();

        static Singleton() { }

        private Singleton() { }

        public static Singleton Instance
        {
            get
            {
                return instance;
            }
        }
    }

    //case 15
    public sealed class SingletonLazy
    {
        private static readonly Lazy<SingletonLazy> Lazy = new Lazy<SingletonLazy>(() =>
new SingletonLazy());

        public static SingletonLazy Instance => Lazy.Value;

        static SingletonLazy() { }

        private SingletonLazy() { }
    }
}
```