Міністерство освіти і науки України

Національний технічний університет України „КПІ”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки

інформації та управління

**ЗВІТ**

до лабораторної роботи № 2

з дисципліни “Основи Web-програмування”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Виконав**  **студент** |  | *ІП-61 Кушка Михайло Олександрович* |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | *Ліщук К. І.* |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2018

# Зміст

[**1. Постановка Задачі 3**](#_Toc513915848)

[**2. Результат роботи програми 4**](#_Toc513915849)

[**3. Код програми 5**](#_Toc513915850)

# Постановка Задачі

При виконанні комп’ютерого практикуму слід реалізувати наступні задачі:

1. Перевантажити віртуальний метод bool Equals (object obj), таким чином, щоб об‘єкти були рівними, якщо рівні всі дані об‘єктів. Для кожного з класів самостійно визначити, які атрибути використовуються для порівняння;
2. Визначити операції == та != . При цьому врахувати, що визначення операцій повинно бути погоджено з перевантаженим методом Equals, тобто критерії, за якими перевіряється рівність об'єктів в методі Equals, повинні використовуватися і при перевірці рівності об'єктів в операціях == та !=;
3. Перевизначити віртуальний метод int GetHashCode(). Класи базової бібліотеки, що викликають метод GetHashCode() з призначеного користувальницького типу, припускають, що рівням об'єктів відповідають рівні значення хеш-кодів. Тому в разі, коли під рівністю об'єктів розуміється збіг даних (а не посилань), реалізація методу GetHashCode() повинна для об'єктів з однаковими даними повертати рівні значення хеш-кодів.
4. Визначити метод object DeepCopy() для створення повної копії об'єкта. Визначені в деяких класах базової бібліотеки методи Clone() та Copy() створюють обмежену (shallow) копію об'єкта - при копіюванні об'єкта копії створюються тільки для полів структурних типів, для полів, на які посилаються типи, копіюються тільки посилання. В результаті в обмеженій копії об'єкта поля-посилання вказують на ті ж об'єкти, що і в вихідному об'єкті. Метод DeepCopy() повинен створити повні копії всіх об'єктів, посилання на які містять поля типу. Після створення повна копія не залежить від вихідного об'єкта - зміна будь-якого поля або властивості вихідного об'єкта не повинно призводити до зміни копії. При реалізації методу DeepCopy() в класі, який має поле типу System.Collections.ArrayList, слід мати на увазі, що визначені в класі ArrayList конструктор ArrayList (ICollection) і метод Clone() при створенні копії колекції, що складається з елементів, на які посилаються типів, копіюють тільки посилання. Метод DeepCopy() повинен створити як копії елементів колекції ArrayList, так і повні копії об'єктів, на які посилаються елементи колекції. Для типів, що містять колекції, реалізація методу DeepCopy() спрощується, якщо в типах елементів колекцій також визначити метод DeepCopy().
5. Перезавантажити віртуальний метод string ToString() для формування строки з інформацією про всі елементи списку
6. Підготувати демонстраційний приклад, в котрому будуть використані всі розроблені методи
7. Підготувати звіт з результатами виконаної роботи.

При виконанні комп’ютерого практикуму слід реалізувати наступні задачі:

1. Визначити клас, котрий містить типізовану колекцію та котрий за допомогою подій інформує про зміни в колекції.

Колекція складається з об'єктів посилальних типів. Колекція змінюється при видаленні/додаванні елементів або при зміні одного з вхідних в колекцію посилань, наприклад, коли одному з посилань присвоюється нове значення. В цьому випадку у відповідних методах або властивості класу кидаються події.

При зміні даних об'єктів, посилання на які входять в колекцію, значення самих посилань не змінюються. Цей тип змін не породжує подій.

Для подій, що сповіщають про зміни в колекції, визначається свій делегат. Події реєструються в спеціальних класах-слухачах.

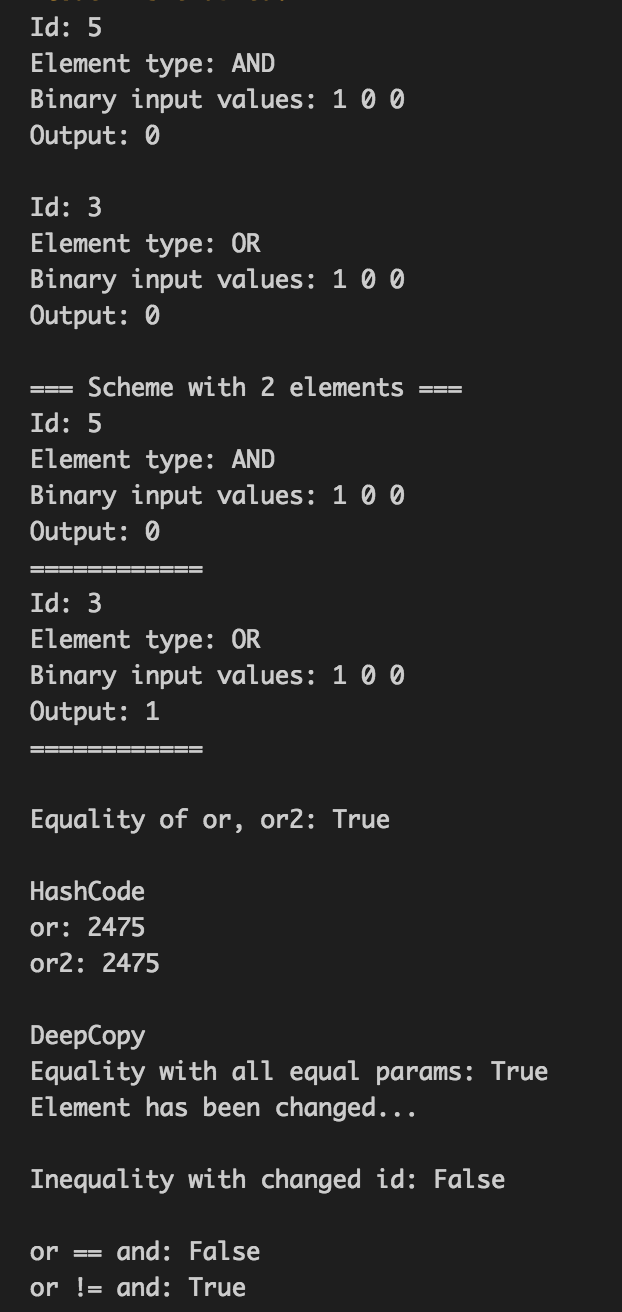
1. Реалізувати обробку помилок, при цьому необхідно перевизначити за допомогою наслідування наступні події:
   * 1. StackOverflowException
     2. ArrayTypeMismatchException
     3. DivideByZeroException
     4. IndexOutOfRangeException
     5. InvalidCastException
     6. OutOfMemoryException
     7. OverflowException
2. Підготувати демонстраційний приклад, в котрому будуть використані всі розроблені методи
3. Підготувати звіт з результатами виконаної роботи.

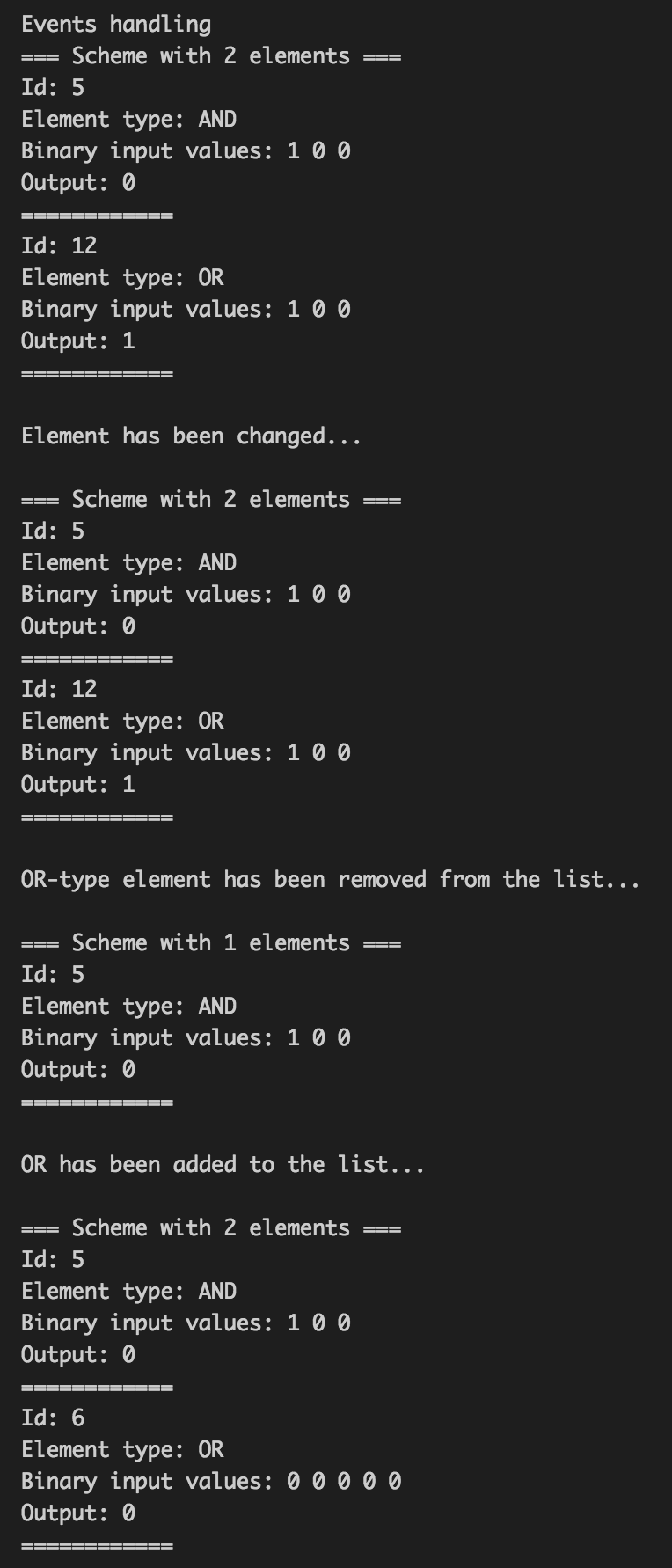
Описать абстрактный класс Element (элемент логической схемы), задав в нем числовой идентификатор, количество входов, идентификаторы присоединенных к нему элементов (до 10) и двоичные значения на входах и выходе. На его основе реализовать классы AND и OR — двоичные вентили, которые могут иметь различное количество входов и один выход и реализуют логическое умножение и сложение соответственно.

Создать класс Sсheme (схема), содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти.

Предусмотреть возможности вывода характеристик объектов списка и вычисление значений, формируемых на выходах схемы по заданным значениям входов.

# Результат роботи програми





# Код програми

using System;

namespace lab2

{

class Program

{

static void Main(string[] args)

{

short[] bytes = {1, 0, 0};

AND and = new AND(bytes, 5);

OR or = new OR(bytes, 3);

OR or2 = new OR(bytes, 3);

ChangeHandler changeHandler = new ChangeHandler();

Console.WriteLine(and);

Console.WriteLine(or);

Element[] arr = {and, or};

Scheme scheme = new Scheme(arr);

scheme.calcAllOutputs();

Console.WriteLine(scheme);

// Subscribe for events

and.onChange += changeHandler.Changed;

or.onChange += changeHandler.Changed;

or2.onChange += changeHandler.Changed;

scheme.onRemove += changeHandler.Removed;

scheme.onAdd += changeHandler.Added;

// Equals

Console.WriteLine("Equality of or, or2: " + or.Equals(or2));

// HashCode

Console.WriteLine("\nHashCode");

Console.WriteLine("or: " + or.GetHashCode());

Console.WriteLine("or2: " + or2.GetHashCode());

// DeepCopy

object or\_copy = or.DeepCopy();

Console.WriteLine("\nDeepCopy");

Console.WriteLine("Equality with all equal params: " + (or == or2));

or.setId(12);

Console.WriteLine("Inequality with changed id: " + (or == or2));

// ==, !=

Console.WriteLine("\nor == and: " + (or == and));

Console.WriteLine("or != and: " + (or != and));

// Events handling

Console.WriteLine("\nEvents handling");

Console.WriteLine(scheme);

// Check changing the element

or.setId(12);

Console.WriteLine(scheme);

// Check removing the element

scheme.RemoveElementAt(1);

Console.WriteLine(scheme);

// Check adding the elemnt

scheme.AddElement(new OR(new short[]{0, 0, 0, 0, 0}, 6));

Console.WriteLine(scheme);

// Exceptions

// throw new DivideByZeroException();

// throw new StackOverflowException();

// throw new ArrayTypeMismatchException();

// throw new IndexOutOfRangeException();

// throw new InvalidCastException();

// throw new OutOfMemoryException();

// throw new OverflowException();

}

}

/\*

Class-template for any type elements.

\*/

abstract class Element

{

protected int id; // Element id

protected short[] binaryInputValues; // Binary values on the inputs

protected short binaryOutputValue; // Binary output value

public delegate void MethodContainer(); // Delegate for the event

public event MethodContainer onChange; // Event onChange with delegate type MethodContainer

abstract public void calcBinaryOutputValue(); // Calculate the result of the element

/\*

Init Element class with input values and element's id.

\*/

protected Element(short[] \_binaryInputValues, int \_id)

{

if (\_binaryInputValues.Length < 1)

throw new Exception("Number of inputs must be at least one.");

binaryInputValues = \_binaryInputValues;

id = \_id;

}

/\*

Set id of the element.

\*/

public void setId(int newId)

{

id = newId;

onChange();

}

/\*

Get value of the output of the element.

\*/

public short getBinaryOutputValue()

{

return binaryOutputValue;

}

/\*

Override default ToString() method to show all classes nesessary params.

\*/

public override string ToString()

{

string result = "Id: " + id

+ "\nElement type: " + this.GetType().Name

+ "\nBinary input values: ";

foreach(short bin in binaryInputValues)

result += bin + " ";

result += "\nOutput: " + binaryOutputValue + "\n";

return result;

}

/\*

Check 2 Element objects for equality of id and binary input values.

\*/

public override bool Equals(object obj)

{

if (id == ((Element)obj).id && binaryInputValues == ((Element)obj).binaryInputValues)

return true;

return false;

}

/\*

Now elements of the same class with equals id's and binary input values has equal has code.

\*/

public override int GetHashCode()

{

int result = id;

foreach(short bin in binaryInputValues)

{

if (bin == 1)

result \*= 3;

else

result \*= 5;

}

if (this.GetType().Name == "AND")

result \*= 7;

if (this.GetType().Name == "OR")

result \*= 11;

return result;

}

/\*

Copy of all fields of the element and make copy independent of it's parent.

\*/

public object DeepCopy()

{

if (this.GetType().Name == "AND")

return new AND(binaryInputValues, id);

else

return new OR(binaryInputValues, id);

}

/\*

Check equality of the elements.

\*/

public static bool operator== (Element first, Element second)

{

if (first.Equals(second))

return true;

return false;

}

/\*

Check inequality of the elements.

\*/

public static bool operator!= (Element first, Element second)

{

if (first.Equals(second))

return false;

return true;

}

}

/\*

AND element in the scheme.

\*/

class AND : Element

{

public AND(short[] \_binaryInputValues, int \_id) : base(\_binaryInputValues, \_id) {}

public override void calcBinaryOutputValue()

{

binaryOutputValue = binaryInputValues[0];

foreach (short bin in binaryInputValues)

binaryOutputValue &= bin;

}

}

/\*

OR element in the scheme.

\*/

class OR : Element

{

public OR(short[] \_binaryInputValues, int \_id) : base(\_binaryInputValues, \_id) {}

public override void calcBinaryOutputValue()

{

binaryOutputValue = binaryInputValues[0];

foreach (short bin in binaryInputValues)

binaryOutputValue |= bin;

}

}

/\*

Simple scheme as a collection of AND and OR elements.

\*/

class Scheme

{

private static Element[] elements; // Elements, which scheme contains

public delegate void RemoveContainer(string elementName); // Delegate for the remove event

public delegate void AddContainer(string elementName); // Delegate for the add event

public event RemoveContainer onRemove; // Event onRemove with delegate type RemoveContainer

public event AddContainer onAdd; // Event onAdd with delegate type AddContainer

/\*

Init the list of Element's objects.

\*/

public Scheme(Element[] \_elements)

{

if(\_elements.Length < 1)

throw new Exception("Numbers of elements should be at least one.");

elements = \_elements;

}

/\*

Calculate the output for every element in the scheme.

\*/

public void calcAllOutputs()

{

foreach(Element element in elements)

{

element.calcBinaryOutputValue();

}

}

/\*

Display all necessary params of the class.

\*/

public override string ToString()

{

string result = "=== Scheme with " + elements.Length + " elements ===\n";

foreach(Element element in elements)

{

result += element + "============\n";

}

return result;

}

/\*

Remove Element from the list on the <index> position.

\*/

public void RemoveElementAt(int index)

{

if (index < 0 || index > elements.Length-1)

throw new Exception("Array's index is out of range.");

string elementName = elements[elements.Length-1].GetType().Name;

Element[] dest = new Element[elements.Length-1];

if( index > 0 )

Array.Copy(elements, 0, dest, 0, index);

if( index < elements.Length - 1 )

Array.Copy(elements, index + 1, dest, index, elements.Length - index - 1);

elements = dest;

// Make onRemove event

onRemove(elementName);

}

/\*

Add new Element to the end of the list.

\*/

public void AddElement(Element newElement)

{

Element[] dest = new Element[elements.Length+1];

Array.Copy(elements, dest, elements.Length);

dest[elements.Length] = newElement;

elements = dest;

// Make onAdd event

onAdd(newElement.GetType().Name);

}

}

/\*

Handler for change events.

\*/

class ChangeHandler

{

/\*

Element has been changed handler.

\*/

public void Changed()

{

Console.WriteLine("Element has been changed...\n");

}

/\*

Element has been removed from the list handler.

\*/

public void Removed(string elementName)

{

Console.WriteLine(elementName + "-type element has been removed from the list...\n");

}

/\*

Element has been added to the list handler.

\*/

public void Added(string elementName)

{

Console.WriteLine(elementName + " has been added to the list...\n");

}

}

/\*

Custom divide by zero exception.

\*/

public class StackOverflowException : Exception

{

public StackOverflowException ()

: base("You stack is so overflow that I can't work anymore...")

{

}

public StackOverflowException(string message)

: base(message)

{

}

public StackOverflowException(string message, Exception inner)

: base(message, inner)

{

}

protected StackOverflowException(System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context) { }

}

/\*

Custom divide by zero exception.

\*/

public class DivideByZeroException : Exception

{

public DivideByZeroException()

: base("You can't divide by zero. Check your code and try again.")

{

}

public DivideByZeroException(string message)

: base(message)

{

}

public DivideByZeroException(string message, Exception inner)

: base(message, inner)

{

}

protected DivideByZeroException(System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context) { }

}

/\*

Custom array type mismatch exception.

\*/

public class ArrayTypeMismatchException : Exception

{

public ArrayTypeMismatchException()

: base("The array has another type, so you can't put in this element. Please double-check all.")

{

}

public ArrayTypeMismatchException(string message)

: base(message)

{

}

public ArrayTypeMismatchException(string message, Exception inner)

: base(message, inner)

{

}

protected ArrayTypeMismatchException(System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context) { }

}

/\*

Custom index out of range exception.

\*/

public class IndexOutOfRangeException : Exception

{

public IndexOutOfRangeException()

: base("Oops, your index is out of range. Change it!")

{

}

public IndexOutOfRangeException(string message)

: base(message)

{

}

public IndexOutOfRangeException(string message, Exception inner)

: base(message, inner)

{

}

protected IndexOutOfRangeException(System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context) { }

}

/\*

Custom invalid cast exception.

\*/

public class InvalidCastException : Exception

{

public InvalidCastException()

: base("Your casting is so strange that I can't do it, seriously.")

{

}

public InvalidCastException(string message)

: base(message)

{

}

public InvalidCastException(string message, Exception inner)

: base(message, inner)

{

}

protected InvalidCastException(System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context) { }

}

/\*

Custom out of memory exception.

\*/

public class OutOfMemoryException : Exception

{

public OutOfMemoryException()

: base("Think you has not enough memory for your code.")

{

}

public OutOfMemoryException(string message)

: base(message)

{

}

public OutOfMemoryException(string message, Exception inner)

: base(message, inner)

{

}

protected OutOfMemoryException(System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context) { }

}

/\*

Custom overflow exception.

\*/

public class OverflowException : Exception

{

public OverflowException()

: base("Your operations are so strange... BOM! Overflow!")

{

}

public OverflowException(string message)

: base(message)

{

}

public OverflowException(string message, Exception inner)

: base(message, inner)

{

}

protected OverflowException(System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context) { }

}

}