### Міністерство освіти і науки України Національний технічний університет України

Київський Політехнічний Інститут імені Ігоря Сікорського Кафедра ОТ

Лабораторна робота № 7

з дисципліни "Основи Web-програмування"

Виконав:

студент 2-го курсу Групи: ІП-64 ФІОТ

Соляніков Павло Залiковка: №6404  
У списку : 22

**Київ-2018**

## Постановка задачі до комп’ютерного практикуму № 7

### При виконанні комп’ютерого практикуму слід реалізувати наступні задачі:

1. Перезавантажити 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 перезавантаженим 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 рівним 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 DeepCop () 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 ().

1. Перезавантажити віртуальний метод string ToString() для формування строки з інформацією про всі елементи списку
2. Підготувати демонстраційний приклад, в котрому будуть використані всі розроблені методи
3. 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 ппо зміни в колекції.

Колекція складається з об'єктів силочних 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.**

# ВЛАСНИЙ ВАРІАНТ ЗАВДАННЯ:

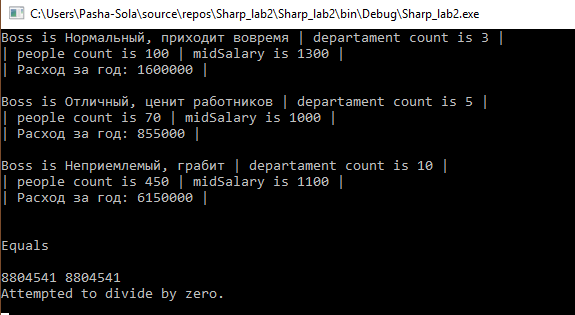
### **Вариант 22**

Создать абстрактный класс Organization, содержащий информацию о руководителе, количестве отделов, персонале и т.д. и выводом общего кол-ва персонала, персонала в каждом из отделов, средней заработной платы. На его основе реализовать классы Plant (завод), Ministry (министерство) и University (ВУЗ). Наряду с общими полями и методами, каждый класс должен содержать и специфичные для него поля.

Создать класс Registry, содержащий массив/параметризованную коллекцию объектов этих классов в динамической памяти.

Предусмотреть возможность вывода характеристик объектов списка.

# СКРІНШОТИ:



Код Програми:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Sharp\_lab2

{

[Serializable]

abstract class Organization

{

string infoAboutTheBoss;

int departamentCount;

int personCount;

int midSalary;

public string infoBoss

{

get { return infoAboutTheBoss; }

}

public int depCount

{

get { return departamentCount; }

}

public int mSalary

{

get { return midSalary; }

}

public int persCount

{

get { return personCount; }

}

public Organization(string s, int a, int b, int salary)

{

this.infoAboutTheBoss = s;

this.departamentCount = a;

this.personCount = b;

this.midSalary = salary;

}

public abstract int countYearExpenses();

public override string ToString()

{

return string.Format("Boss is {0} | departament count is {1} |\n| people count is {2} | midSalary is {3} |\n| Расход за год: {4} |\n", infoAboutTheBoss, departamentCount, persCount, midSalary, countYearExpenses());

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Sharp\_lab2

{

[Serializable]

class Ministry : Organization

{

int ministryCosts;

public Ministry(string s, int a, int b, int salary, int minCosts) : base(s, a, b, salary) { this.ministryCosts = minCosts; }

public override int countYearExpenses()

{

return persCount \* mSalary \* 12 + 10000 + ministryCosts;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

Ministry org = obj as Ministry; // возвращает null если объект нельзя привести к типу Ministry

if (org as Organization == null)

return false;

return org.depCount == this.depCount && org.infoBoss == this.infoBoss && org.mSalary == this.mSalary && org.persCount == this.persCount && org.ministryCosts == this.ministryCosts;

}

public bool Equals(Ministry org) // аргумент типа Ministry

{

if (org == null)

return false;

return org.depCount == this.depCount && org.infoBoss == this.infoBoss && org.mSalary == this.mSalary && org.persCount == this.persCount;

}

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

{

if ((Object)first == null || (Object)second == null)//проверить на null

return false;

return first.Equals(second);

}

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

{

if ((Object)first == null || (Object)second == null)//проверить на null

return false;

return first.Equals(second);

}

public override int GetHashCode()

{

int hash = 0;

int bird = 1;

foreach (char c in this.infoBoss)

{

hash += System.Convert.ToInt32(c) \* bird;

bird += 1;

}

hash += this.depCount \* 100 + this.persCount \* 200 + this.mSalary \* 300 + this.ministryCosts \* 400;

return hash;

}

public Ministry DeepCopy()

{

Ministry obj = new Ministry(this.infoBoss, this.depCount, this.persCount, this.mSalary, this.ministryCosts);

return obj;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Sharp\_lab2

{

[Serializable]

class Plant : Organization

{

int plantCosts;

public Plant(string s, int a, int b, int salary, int pCosts) : base(s, a, b, salary) { this.plantCosts = pCosts; }

public override int countYearExpenses()

{

return persCount \* mSalary \* 12 + 5000 + plantCosts;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

Plant org = obj as Plant; // возвращает null если объект нельзя привести к типу Plant

if (org as Organization == null)

return false;

return org.depCount == this.depCount && org.infoBoss == this.infoBoss && org.mSalary == this.mSalary && org.persCount == this.persCount && org.plantCosts == this.plantCosts;

}

public bool Equals(Plant org) // аргумент типа Plant

{

if (org == null)

return false;

return org.depCount == this.depCount && org.infoBoss == this.infoBoss && org.mSalary == this.mSalary && org.persCount == this.persCount && org.plantCosts == this.plantCosts;

}

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

{

if ((Object)first == null || (Object)second == null)//проверить на null

return false;

return first.Equals(second);

}

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

{

if ((Object)first == null || (Object)second == null)//проверить на null

return false;

return first.Equals(second);

}

public override int GetHashCode()

{

int hash = 0;

int bird = 1;

foreach (char c in this.infoBoss)

{

hash += System.Convert.ToInt32(c) \* bird;

bird += 1;

}

hash += this.depCount \* 100 + this.persCount \* 200 + this.mSalary \* 300 + this.plantCosts \* 400;

return hash;

}

public Plant DeepCopy()

{

Plant obj = new Plant(this.infoBoss, this.depCount, this.persCount, this.mSalary, this.plantCosts);

return obj;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Sharp\_lab2

{

[Serializable]

class University : Organization

{

int universityCosts;

public University(string s, int a, int b, int salary, int univCosts) : base(s, a, b, salary) { this.universityCosts = univCosts; }

public override int countYearExpenses()

{

return persCount \* mSalary \* 12 + 20000 + universityCosts;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

University org = obj as University; // возвращает null если объект нельзя привести к типу University

if (org as University == null)

return false;

return org.depCount == this.depCount && org.infoBoss == this.infoBoss && org.mSalary == this.mSalary && org.persCount == this.persCount && org.universityCosts == this.universityCosts;

}

public bool Equals(University org) // аргумент типа University

{

if (org == null)

return false;

return org.depCount == this.depCount && org.infoBoss == this.infoBoss && org.mSalary == this.mSalary && org.persCount == this.persCount;

}

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

{

if ((Object)first == null || (Object)second == null)//проверить на null

return false;

return first.Equals(second);

}

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

{

if ((Object)first == null || (Object)second == null)//проверить на null

return false;

return first.Equals(second);

}

public override int GetHashCode()

{

int hash = 0;

int bird = 1;

foreach (char c in this.infoBoss)

{

hash += System.Convert.ToInt32(c) \* bird;

bird += 1;

}

hash += this.depCount \* 100 + this.persCount \* 200 + this.mSalary \* 300 + this.universityCosts \* 400;

return hash;

}

public University DeepCopy()

{

University obj = new University(this.infoBoss, this.depCount, this.persCount, this.mSalary, this.universityCosts);

return obj;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Runtime.Serialization.Formatters.Binary;

namespace Sharp\_lab2

{

[Serializable]

class Registry

{

// Organization[] arrayObj = { unic, plant, minist };

Organization[] ObjArray;

public Registry()

{

Organization unic = new University("Нормальный, приходит вовремя", 3, 100, 1300, 20000);

Organization plant = new Plant("Отличный, ценит работников", 5, 70, 1000, 10000);

Organization minist = new Ministry("Неприемлемый, грабит", 10, 450, 1100, 200000);

ObjArray = new Organization[] { unic, plant, minist };

//printInfo(unic);

//printInfo(plant);

//printInfo(minist);

}

public Registry DeepCopy()

{

using (var ms = new System.IO.MemoryStream())

{

var formatter = new BinaryFormatter();

formatter.Serialize(ms, this);

ms.Position = 0;

return (Registry)formatter.Deserialize(ms);

}

}

public void printInfo(Organization org)

{

Console.WriteLine("Info about our boss: " + org.infoBoss + " \nCount of departaments is "

+ org.depCount + "\n Count of people is " + org.persCount + "\n Middle Salary is : " + org.mSalary + "\n Summary count Year Expenses is : " + org.countYearExpenses() + "\n");

}

public override string ToString()

{

string res = "";

for (int i = 0; i < ObjArray.Length; i++)

res = string.Concat(res, string.Concat(ObjArray[i] + "\n"));

return res;

}

}

class CastException : InvalidCastException

{

public CastException()

: base() { }

public CastException(string message)

: base(message) { }

public CastException(string format, params object[] args)

: base(string.Format(format, args)) { }

public CastException(string message, Exception innerException)

: base(message, innerException) { }

public CastException(string format, Exception innerException, params object[] args)

: base(string.Format(format, args), innerException) { }

}

class ArrTypeMismatchException : ArrayTypeMismatchException

{

public ArrTypeMismatchException()

: base() { }

public ArrTypeMismatchException(string message)

: base(message) { }

public ArrTypeMismatchException(string format, params object[] args)

: base(string.Format(format, args)) { }

public ArrTypeMismatchException(string message, Exception innerException)

: base(message, innerException) { }

public ArrTypeMismatchException(string format, Exception innerException, params object[] args)

: base(string.Format(format, args), innerException) { }

}

class DivideByZero : DivideByZeroException

{

public DivideByZero()

: base() { }

public DivideByZero(string message)

: base(message) { }

public DivideByZero(string format, params object[] args)

: base(string.Format(format, args)) { }

public DivideByZero(string message, Exception innerException)

: base(message, innerException) { }

public DivideByZero(string format, Exception innerException, params object[] args)

: base(string.Format(format, args), innerException) { }

}

class MyException3 : OutOfMemoryException

{

public MyException3()

: base() { }

public MyException3(string message)

: base(message) { }

public MyException3(string format, params object[] args)

: base(string.Format(format, args)) { }

public MyException3(string message, Exception innerException)

: base(message, innerException) { }

public MyException3(string format, Exception innerException, params object[] args)

: base(string.Format(format, args), innerException) { }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Sharp\_lab2

{

class Program

{

static void Main(string[] args)

{

Registry regist = new Registry();

// Console.WriteLine(regist);

Registry regist1 = regist.DeepCopy();

Console.WriteLine(regist1);// ToString

Organization unic = new Ministry("Нормальный, приходит вовремя", 3, 100, 1300, 20000);

Organization unic1 = new Ministry("Нормальный, приходит вовремя", 3, 100, 1300, 20000);

if (unic.Equals(unic1))

{

Console.WriteLine("Equals");

}

else

{

Console.WriteLine(" Not Equals");

}

Console.WriteLine("\n" + unic.GetHashCode() + " " + unic1.GetHashCode());

int a = 4;

int b = 2;

int c = 0;

try

{

if (c == 0) { throw new DivideByZero(); }

a = b / c;

}

catch (DivideByZero e)

{

Console.WriteLine(e.Message);

}

Console.ReadLine();

}

}

}