МІНІСТЕРСТВО НАУКИ І ОСВІТИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ. ІГОРЯ СІКОРСЬКОГО»

КАФЕДРА АПЕПС

Звіти до лабораторних робіт №2-4

Об’єктно-орієнтоване програмування-2

Варіант 9

Виконав:

студент 2 курсу,

групи ТВ-51, ТЕФ

Іванів А.П.

Київ — 2017

**Код програми:**

**Book.cs**  
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab234

{

class Book

{

private string authorSurname = "Rowling";

private string authorName = "Joanne";

private string name = "Harry Potter";

private DateTime date = new DateTime(1997, 6, 26);

private int price = 500;

private int quantity = 10;

public Book()

:this ("","","", DateTime.MinValue, 0,0)

{

Console.WriteLine("Book constructor, 0 parametrs");

}

public Book(string AuthorSurname)

:this (AuthorSurname, "", "", DateTime.MinValue, 0, 0)

{

Console.WriteLine("Book constructor, 1 parametrs");

}

public Book(string AuthorSurname, string AuthorName, string Name)

:this (AuthorSurname, AuthorName, Name, DateTime.MinValue, 0, 0)

{

Console.WriteLine("Book constructor, 3 parametrs");

}

public Book(string AuthorSurname, string AuthorName, string Name, DateTime Date, int Price, int Quantity)

{

\_authorsurname = AuthorSurname;

\_authorname = AuthorName;

\_name = Name;

\_date = Date;

if (Price < 0)

Price = 0;

\_price = Price;

if (Quantity < 0)

Quantity = 0;

\_quantity = Quantity;

Console.WriteLine("Book constructor, 6 parametrs");

}

public virtual void Display()

{

Console.WriteLine("|------------------|------------------|------------------|------------------|------------------|-------------------|");

Console.WriteLine("| Author Surname | Author Name | Name | Date | Price | Quantity |");

Console.WriteLine("|------------------|------------------|------------------|------------------|------------------|-------------------|");

Console.WriteLine("|{0,-18}|{1, -18}|{2,-18}|{3, -18}|{4, -18}|{5,-18} |", this.\_authorsurname, this.\_authorname, this.\_name, this.\_date, this.\_price, this.\_quantity);

Console.WriteLine("|------------------|------------------|------------------|------------------|------------------|-------------------|");

}

public static Book operator +(Book b1, int price)

{

b1.\_price = b1.\_price + price;

return b1;

}

public static Book operator -(Book b1, Book b2)

{

Book result = new Book();

if (b1.\_price < b2.\_price) result.\_price = b2.\_price - b1.\_price;

else result.\_price = b1.\_price - b2.\_price;

return result;

}

public static bool operator !=(Book b1, Book b2)

{

if (b1.\_authorsurname != b2.\_authorsurname) return true;

if (b1.\_name != b2.\_name) return true;

return false;

}

public static bool operator ==(Book b1, Book b2)

{

if (b1.\_price == b2.\_price)

if (b1.\_quantity == b2.\_quantity)

return true;

return false;

}

public static bool operator true(Book b)

{

if (b.\_price >= 0 && b.\_quantity >= 0) return true;

return false;

}

public static bool operator false(Book b)

{

if (b.\_price < 0 || b.\_quantity < 0) return true;

return false;

}

public static implicit operator int(Book b)

{

return b.\_quantity;

}

public string \_authorsurname { get; set; }

public string \_authorname

{

get { return authorName; }

set { authorName = value; }

}

public string \_name

{

get { return name; }

set { name = value; }

}

public DateTime \_date

{

get { return date; }

set { date = value; }

}

public int \_price

{

get { return price; }

set { price = price < 0 ? 0 : value; }

}

public int \_quantity

{

get { return quantity; }

set { quantity = value; }

}

}

}

**AvaragePrice.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab234

{

class AvaragePrice : Book

{

private static float value = 0;

private static int count = 0;

public static void ChangeValue(int price, int quantity)

{

value = value \* ((float)count / (float)(count + quantity));

if (value == 0)

{

value = price;

}

else if (price > value)

{

value += ((float)price \* ((float)quantity / (float)(count + quantity)));

}

else if (price < value)

{

value -= ((float)price \* ((float)quantity / (float)(count + quantity)));

}

count += quantity;

}

static AvaragePrice()

{

value = 0;

Console.WriteLine("Avarage Price static constructor");

Console.WriteLine("-------------------------------------------");

}

public AvaragePrice()

:base ()

{

Console.WriteLine("Avarage Price constructor, 0 parametrs");

}

public AvaragePrice(string AuthorSurname, string AuthorName, string Name, DateTime Date, int Price, int Quantity)

:base (AuthorSurname,AuthorName,Name,Date,Price,Quantity)

{

ChangeValue(base.\_price, base.\_quantity);

Console.WriteLine("Avarage Price constructor, 6 parametrs");

Console.WriteLine("-------------------------------------------");

}

public float AvarPrice

{

get { return value; }

}

public int Count

{

set { count = value; }

}

public virtual void Display()

{

Console.WriteLine("|------------------|------------------|------------------|------------------|------------------|------------------|-------------------|");

Console.WriteLine("| Author Surname | Author Name | Name | Date | Price | Quantity | Avarage Price |");

Console.WriteLine("|------------------|------------------|------------------|------------------|------------------|------------------|-------------------|");

Console.WriteLine("|{0,-18}|{1, -18}|{2,-18}| {3:d} |{4, -18}|{5, -18}|{6,-18} |", this.\_authorsurname, this.\_authorname, this.\_name, this.\_date, this.\_price, this.\_quantity, AvaragePrice.value);

Console.WriteLine("|------------------|------------------|------------------|------------------|------------------|------------------|-------------------|");

}

}

}

**BooksArray.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab234

{

class BooksArray

{

private static AvaragePrice[] booksArray;

private static uint ArraySize;

public uint Size

{

get { return ArraySize; }

}

public BooksArray(uint sz)

{

ArraySize = sz;

booksArray = new AvaragePrice[ArraySize];

}

public AvaragePrice this[uint index]

{

get

{

if (index >= 0 && index < ArraySize) return booksArray[index];

else return null;

}

set

{

if (index >= 0 && index < ArraySize) booksArray[index] = value;

}

}

}

}

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab234

{

class Program

{

static void Main(string[] args)

{

Console.SetWindowPosition(0, 0);

Console.SetWindowSize(150, 50);

Console.SetBufferSize(1800, 250);

AvaragePrice b1 = new AvaragePrice("Franko", "Ivan", "Moisei", DateTime.Parse("25/06/1905"), 100, 7);

AvaragePrice b2 = new AvaragePrice("Shevchenko", "Taras", "Kavkaz", DateTime.Parse("18/09/1845"), 5000, 2);

AvaragePrice b3 = new AvaragePrice

{

\_authorsurname = "Gogol'",

\_authorname = "Mukola",

\_name = "Vii",

\_date = DateTime.Parse("13/08/1835"),

\_price = 777,

\_quantity = 200

};

Book a = b1 + 10;

Book c = b1 - b3;

b1.Display();

b2.Display();

Console.WriteLine("Zagalna vartist knug b1 i b3 dorivnye {0, -10}", a.\_price);

Console.WriteLine("Riznutsa v zini mizh knugamu b1 i b3 {0, -10}", c.\_price);

if (b1 != b3) Console.WriteLine("True: books are different");

else Console.WriteLine("False: books are same");

if (b1 == b3) Console.WriteLine("True: price and count of some books are equial");

else Console.WriteLine("False: price and quantity of some books are different");

if (a) Console.WriteLine("Information of quantity and price of book A are true");

int b = b1;

Console.WriteLine("Quantiy of b1 books are {0,-4}\n", b);

uint N = 5;

BooksArray mass = new BooksArray(N);

mass[0] = new AvaragePrice("Ukrainka", "Lesiya", "Lisova pisnya", DateTime.Parse("04/05/1812"), 150, 400);

mass[1] = new AvaragePrice("Franko", "Ivan", "Moisei", DateTime.Parse("25/06/1905"), 200, 10);

mass[2] = new AvaragePrice("Besson", "Lyk", "Miniputes", DateTime.Parse("01/01/2004"), 300, 1000);

mass[3] = new AvaragePrice("Shevchenko", "Taras", "Kavkaz", DateTime.Parse("18/09/1845"), 10000, 1);

mass[4] = new AvaragePrice("Gogol", "Mukola", "Vii", DateTime.Parse("13/08/1835"), 700, 130);

for (uint i = 0; i < N; i++)

{

mass[i].Display();

}

Console.Read();

}

}

}

Оскільки ми використовуємо зчеплення конструкторів, то у нас спрацює той конструктор, з якою кількістю параметрів в мейні ми оголосили наш клас.

При ініціалізації даних, ми можемо спочатку їх внести при оголошенні змінних в заданому класу. Далі ми можемо переоголосити змінні в конструкторі ініціалізації. І ще також одна з найуживаніших форм ініціалізації – це при створенні об’єкта класу. Ось приклад:

AvaragePrice b1 = new AvaragePrice("Franko", "Ivan", "Moisei", DateTime.Parse("25/06/1905"), 100, 7);

Приклад використання перевантаженої операції приведення типів:

public static implicit operator int(Book b)

{

return b.\_quantity;

}

І ось використання його в мейні:

int b = b1;

Console.WriteLine("Quantiy of b1 books are {0,-4}\n", b);

Індексатор я використовую для заповнення масиву даних і потім виведення їх на екран:

mass[0] = new AvaragePrice("Ukrainka", "Lesiya", "Lisova pisnya", DateTime.Parse("04/05/1812"), 150, 400);

mass[1] = new AvaragePrice("Franko", "Ivan", "Moisei", DateTime.Parse("25/06/1905"), 200, 10);

mass[2] = new AvaragePrice("Besson", "Lyk", "Miniputes", DateTime.Parse("01/01/2004"), 300, 1000);

mass[3] = new AvaragePrice("Shevchenko", "Taras", "Kavkaz", DateTime.Parse("18/09/1845"), 10000, 1);

mass[4] = new AvaragePrice("Gogol", "Mukola", "Vii", DateTime.Parse("13/08/1835"), 700, 130);

for (uint i = 0; i < N; i++)

{

mass[i].Display();

}

**Теоретичні питання:**

1. **Коли спрацьовує статичний конструктор?**

Статичний конструктор використовується для ініціалізації будь-яких статичних даних. Він викликається автоматично перед створенням першого екземпляра або посилання на будь-які статичні члени.

1. **Як реагує компілятор на мові С# на визначення будь-якого конструктора ініціалізіції? В яких випадках доцільно використоввати шаблон проектування зчеплення конструкторів?**

Кожен клас C # забезпечується конструктором за замовчуванням, який при необхідності може бути перевизначений. За визначенням такої конструктор ніколи не сприймає аргументів. Після розміщення нового об'єкта в пам'яті конструктор за замовчуванням гарантує установку всіх полів до відповідних стандартні значення. Якщо ви не задоволені такими присвоювання за замовчуванням, можете перевизначити конструктор за замовчуванням відповідно до своїх потреб. Конструктор також може приймати один або кілька параметрів. Для цього достатньо оголосити їх в дужках після імені конструктора. В залежності від вхідних параметрів, компілятор використовує один з описаних конструкторів.

Шаблон проектування зчеплення конструкторів корисний, коли є клас, який визначає кілька конструкторів.

1. **Чи можна використовуючи поля позначені як public, забезпечити інкапсуляцію даних? Які методи слід використовувати для доступу до внутрішніх даних обєкту, щоб це відповідало ідеї захисту даних?**

Метою інкапсуляції є забезпечення узгодженості внутрішнього стану об'єкта. У C # для інкапсуляції використовуються публічні властивості і методи об'єкта. Змінні, за рідкісним винятком, не повинні бути публічно доступними. Проілюструвати инкапсуляцию можна на простому прикладі. Припустимо, нам необхідно зберігати дійсне значення і його строкове представлення (наприклад, для того, щоб не робити кожен раз конвертацію в разі частого використання). Приклад реалізації без інкапсуляції такий:

class NoEncapsulation

{

public double ValueDouble ;

public string ValueString ;

}

При цьому ми можемо окремо змінювати як саме значення Value , так і його строкове представлення, і в певний момент може виникнути їх невідповідність (наприклад, в результаті виключення). Приклад реалізації з використанням інкапсуляції:

class EncapsulationExample

{

private double valueDouble ;

private string valueString ;

public double ValueDouble

{

get { return valueDouble ; }

Set

{

valueDouble = value ;

valueString = value . ToString ();

}

}

public string ValueString

{

get { return valueString ; }

Set

{

double tmp\_value = Convert . ToDouble ( value ); // тут може виникнути виключення

valueDouble = tmp\_value ;

valueString = value ;

}

}

}

Тут доступ до змінних valueDouble і valueString можливий тільки через властивості ValueDouble і ValueString . Якщо ми спробуємо привласнити властивості ValueString некоректну рядок і виникне виключення в момент конвертації, то внутрішні змінні залишаться в колишньому, узгодженому стані, оскільки виняток викликає вихід з процедури.

**4) У чому полягає загальний формат перевантаження для унарних операторів на мові C#? Наведіть приклад. Які особливості визанчення перевантажених унарних операторів? Наведіть приклад.**

Можна перевантажувати : +, -,!, ++, -, true, false

Загальна структура перегрузки унарних операторів має наступний вигляд:

public static [повертаємий\_тип] operator [оператор]([тип\_операнда] [операнд])

{

//функціонал оператора

}

Приклад:

Public static Money operator ++ (Money a) // перевантаження «++»

{

a.Amount ++;

return a;

}

Public static Money operator - (Money a) // перевантаження «-»

{

a.Amount--;

return a;

}