Міністерство освіти і науки України
Вінницький національний технічний університет

Методичні вказівки до лабораторних робіт з дисципліни «Технології програмування» для студентів зі спеціальності «Кібербезпека»

Вінниця
ВНТУ
2017

Затверджено Методичною радою Вінницького національного технічного університету Міністерства освіти і науки України як «Методичні вказівки. Електронне видання» та рекомендовано до використання в навчальному процесі(протокол № 9 від 17.05.2017 р.)

Рецензенти:
В. В. Карпінець, кандидат технічних наук, доцент
Д. І. Катєльніков, кандидат технічних наук, доцент

Методичні вказівки до лабораторних робіт з дисципліни «Технології програмування» для студентів зі спеціальності «Кібербезпека» / Уклад. Ю.Є. Яремчук, Д.П. Присяжний, І. О. Дьогтєва, О. В. Салієва – Вінниця: ВНТУ, 2017. – 97 с.

У даних методичних вказівках до лабораторних робіт наводяться основні рекомендації до вивчення, підготовки та проведення лабораторних робіт з дисципліни «Технології програмування» та організації самостійної роботи студентів.

ЗМІСТ

ВСТУП

Лабораторні роботи є сполучною ланкою між лекційними заняттями і самостійною роботою студентів. В процесі виконання лабораторних робіт експериментально перевіряються ключові питання курсу «Технології програмування», набуваються практичні навички створення програм.

Метою методичних вказівок є надання допомоги студентам в отриманні практичних навичок роботи з текстами програм на мові C # у середовищі Visual Studio.

C #–об'єктно-орієнтована мова програмування, придатна для вирішення широкого кола завдань.C # відноситься до сім'ї мов з C-подібним синтаксисом, серед них його синтаксис найбільш близький до C ++ і Java. Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML.

У серії лабораторних робіт використовується середовище розробки Visual Studio.NET, яке надає потужні і зручні засоби написання, коригування, компіляції, налагодження і запуску додатків, що використовують .NET-сумісні мови, зокрема, C #.

Під час виконання лабораторних робіт студенти зможуть:

- ✓ розуміти основні елементи .NET Framework і зв'язок C # з елементами платформи;
- ✓ працювати в середовищі розробки Visual Studio;
- ✓ створювати, налагоджувати, компілювати і виконувати програми;
- ✓ створювати і використовувати змінні;
- ✓ використовувати вирази мови;
- ✓ створювати методи;
- ✓ створювати, форматувати і використовувати масиви;
- ✓ знати базові концепції і термінологію об'єктно-орієнтованого програмування;
- ✓ створювати, форматувати і руйнувати об'єкти в програмі;
- ✓ створювати класи та ієрархії класів;
- ✓ визначати операції і події в призначеному для користувача класі;
- ✓ реалізовувати властивості і індексатори;
- ✓ використовувати стандартні і призначені для користувача атрибути.

Студент зобов'язаний до лабораторного заняття прочитати методичні вказівки до лабораторної роботи і спробувати виконати її самостійно. Під час лабораторного заняття студент показує викладачеві результати роботи, проводить консультації з питань, які виникли, та завершує роботу.

Захист роботи полягає в виконанні завдання до лабораторної роботи, відповіді на питання по темі лабораторної роботи і внесення деяких змін в тексти програм, які розроблялись,в присутності викладача.

Результати виконання робіт рекомендується зберігати в особистих папках.

ЛАБОРАТОРНА РОБОТА № 1
МОВА ПРОГРАМУВАННЯ C#. ДАНІ,ВИРАЗИ, ІДЕНТИФІКАТОРИ, ОПЕРАТОРИ

Мета:ознайомитись з інтегрованим середовищем розробки Visual Studio, з видами даних у C# та виразами, навчитись правильно використовувати типи даних при визначенні змінних, розглянути різні оператори та ідентифікатори.

1.1 Теоретичні відомості

1.1.1 Мова програмування C#. Visual Studio

Мова програмування C# була створена в 1999 році компанією Microsoft. Спочатку мова мала назву COOL–скорочення від C-like Object-Oriented Language, але пізніше вона була змінена на C#.

Мова була створена Андерсом Хайлсбургом, щоб допомогти усунути недоліки інших мов програмування, які використовувались на той час. Мова строго типізована, об'єктно-орієнтована, і компонентно-орієнтована, використовується уніфікована система типів. На відміну від C або C ++, C# управляє пам'яттю і здійснює управління ресурсами, що вписується в поняття керованого коду. C# використовує механізм збору сміття, щоб звільнити пам'ять і ресурси, на які більше немає посилань в коді програми, що допомагає запобігти проблемам витоку пам'яті.

✓ Строго-типізована –мова передбачає перевірку типів змінних в коді.

✓ Об'єктно-орієнтована C# надає розробникові використовувати всі принципи об'єктно-орієнтованого програмування, такі як інкапсуляція, успадкування і поліморфізм.

✓ КомпонентноорієнтованаC# дозволяє створювати програмні компоненти для автономних пакетів функціональності.

✓ Уніфікована система типів–всі типи C#, від примітивних до посилань, успадковують від одного типу, відомого як Object.

Visual Studio є найкращим інструментом для більшості програмістів, які пишуть код на C#. Visual Studio представляє собою інтегроване середовище розробки (IDE), що містить безліч функцій, які покликані зробити процес кодування набагато простішим. За допомогою Visual Studio ви можете бачити підсвічування синтаксису кольором і форматування коду, що дозволяє легше побачити ключові слова і структуру коду. Intellisense дозволяє заощадити час на написання коду і пошук сигнатур методів.

Visual Studio включає в себе інтегровані засоби налагодження, які допоможуть усунути помилки у вашому коді.

З Visual Studio ви можете переглядати оголошення і редагувати їх в інших файлах проекту, не втрачаючи при цьому вашої поточної позиції.

Ви можете керувати всім вашим рішенням, яке може включати в себе кілька файлів проекту. Розгортання додатків також легко здійснюється за допомогою Visual Studio, наприклад, розгортання на мобільний пристрій, веб-сервер або хмару.

1.1.2 Типи даних

Усі програми зберігають і обробляють дані в межах пам'яті комп'ютера. C# підтримує два види типів даних, що використовуються для представлення інформації в реальному житті.

Типи **значень**, які так називаються, оскільки вони містять фактичне значення даних, які зберігають.

Наприклад, ви могли б використати тип **int**, який зберігає значення 3. Буквальне значення 3 зберігається в змінній, що ви оголошуєте, щоб зберегти значення.

За винятком DateTime і рядків, в таблиці A.1 (додаток A), перераховані типи даних, що є скороченнями для структур в .NET, які представляють типи даних в Microsoft .NET Framework. Так використання **int**аналогічне використанню System.Int32.

Типи **посилань**, які також відомі як об'єкти. Ці типи створюються з класів, які ми розглянемо пізніше. Тип посилань зберігає посилання на розміщення об'кта в пам'яті комп'ютера. Якщо ви знайомі з C/C ++, то ви можете сприймати посилання на ділянку в пам'яті як показчик в C/C++.

1.1.3Вирази

У C# вираз вважається командою. За допомогою виразів можна виконати певні дії в коді, такі як виклик методу або виконання розрахунків. Вирази також використовуються для оголошення змінних і присвоєння значення.

Вирази формуються з лексем. Цими токенами можуть бути ключові слова, ідентифікатори (змінні), оператори, і крапка з комою (;). Всі вирази в C# повинні закінчуватися крапкою з комою.

1.1.4 Ідентифікатори

В C# **ідентифікатор**–це назва, яку ви надаєте елементу у вашій програмі. До переліку елементів належать:

Namespaces (простори імен)–.NETFramework використовує простори імен як спосіб відокремлення файлів класів в суміжні відділи або категорії. Це також допомагає уникнути конфліктів імен в додатках, які можуть містити класи з таким же ім'ям.

Classes (класи) –класи це основа для типу посилань. Вони визначають яку структуру буде приймати об'єкт при створенні екземпляра класу.

Methods (методи) –окремі функціональні елементи в програмі.

Variables (змінні) – ці ідентифікатори, які ви створюєте для зберігання значень або посилань на об'єкти у вашому коді. Змінні по суті– це ім'я комірки пам'яті.

При створенні змінної в C# ви повинні дати їй тип даних. Ви можете присвоїти значення змінної під час її створення або пізніше в вашому програмному коді. C # не дозволить використовувати змінну без початкового значення, це запобігає використанню у вашому додатку небажаних даних.Наступний приклад коду демонструє оголошення змінної і присвоєння їй значення.

```
intmyVar = 0;
```

C# має деякі обмеження щодо ідентифікаторів, на які Вам варто звернути увагу.

По-перше, ідентифікатори чутливі до регістру, тому що C# чутлива до регістру мова. Це означає, що ідентифікатори, такі якmyVar, _myVar, andmyvar, розглядаються як різніідентифікатори.

Ідентифікатори можуть містити тільки літери (в будь-якому регістрі), цифри і символ підкреслення. Ви можете почати ідентифікатор тільки з букви або символу підкреслення. Ви не можете задати ідентифікатор, що починається з цифри.myVari_myVar– значення, які дозволено використовувати, а 2Vars–ні.

C#має набір зарезервованих ключових слів, які використовуються мовою програмуванння. Ви не можете використовувати ці ключові слова в якості ідентифікатора в коді. Ви можете скористатися чутливістю до регістру в C # і використовувати словоDoubleв якості ідентифікатора, щоб відрізнити його відзарезервованого ключового словаdouble,але це не рекомендований підхід.

У таблиці А.2 (додаток А) наведено зарезервовані ключові словаC#.

1.1.5 Оператори

Оператор–це токен, який застосовується до операцій для одного або декількох операндів у виразі. Приклади:

3 + 4– вираз, де до абсолютного значення 4 додається абсолютне значення 3.

counter++– вираз, який буде призводити до того, що змінна (лічильник) збільшується на одиницю.

Не всі оператори підходять для всіх типів даних в C#. Як приклад, в попередньому списку оператор + використовується для підсумовування двох чисел. Ви можете використовувати один і той же оператор, щоб об'єднати два рядки в один, наприклад:

“Тарас” + “Бульба” та в результаті отримаємо новий рядокТарасБульба.

Однак ви не можете використовувати інкремент (++) для рядків.Іншими словами, наступний приклад викличе помилку вC#.

“Тарас”++

У таблиці А.3 (додаток А) наведено перелік операторів C# за типом.

1.1.6Перетворення типів

C# підтримує два типи, що властиві для конверсії (перетворення) типів даних,**явний і неявний**. C# буде використовувати неявне перетворення там, де це буде можливим, головним чином у тому випадку, якщо перетворення не призведе до втрати даних або коли перетворення можливо здійснити з сумісним типом даних. Нижче наведено приклад неявного перетворення даних.

intmyInt = 2147483647;

longmyLong= myInt;

Типlongмає розмір 64 біт в пам'яті в той час як типintвикористовує 32 біти. Саме томуlongможе бути легко помістити значення, що міститься в типі int. Однак перехід відlongуже доintможе спричинити втрату даних і вам потрібно використовувати явне перетворення у цьому випадку.

Явне перетворення може бути виконане одним з двох способів, що продемонстовані нижче.

```
doublemyDouble = 1234.6;
intmyInt;// Перетворення double до int шляхом додавання модифікатора типу у дужках перед типом, який потрібно конвертуватимyInt = (int)myDouble;
```

Наступний варіант використовує спеціальний метод .NETFramework.

```
doublemyDouble = 1234.6;
intmyInt;
// Перетворення double до int шляхом використання класу Convert та методу ToInt32().
// Перетворення значення value до 32 бітного цілочисельного типу.
myInt = Convert.ToInt32(myDouble);
```

Ви знайдете безліч інших методів класу Convert, що здійснює перетворення до різних типів даних, такі якToBoolean(), ToByte(), ToChar() та ін.

МетодConvert.ToInt32()також може бути використаний для перетворення рядка до числового типу даних. Наприклад, ви можете мати програму з графічним інтерфейсом, в якому вхідні дані подаються в текстовому форматі. Ці строкові значення передаються до вашої програми. Застосування описаного вище способу може допомогти запобігти виникненню помилок в коді при спробі використання невірного типу даних.

C# також надає інший механізм для перетворення типів. Використання методівTryParse()таParse()також може допомогти впоратись з цим завданням. Ці методи були додані до типів в C# так як і класConvert. Приклад допоможе продемонструвати використання цього механізму.

```
// Приклад із застосуванням TryParse()
bool result = Int32.TryParse(value, out number);
// Приклад із застосуванням Parse()
int number = Int32.Parse(value);
```

У прикладіTryParse(), метод повертаєіндикатор того, чи вдало була проведена конвертація. В прикладі зParse(), якщо конвертацію не валось здійснити–буде отримана помилка.

1.2 Завдання до лабораторної роботи (Додаток А)

1.3 Контрольні запитання

1. Які є основні (базові) типи даних в мові C#?
2. Які особливості використання цілочисельних типів даних?
3. Які особливості типів даних з плаваючою комою (дійсні типи)?
4. Які є базові символьні типи даних в мові C#?
5. Які види або категорії операторів існують в мові C#?
6. Які оператори належать до категорії арифметичних операторів?
7. Приклади використання операторів (+),(–), множення (*) і ділення (/).
8. Які особливості використання оператору обчислення остачі (%)?
9. Приклади застосування операторів інкременту (++) та декременту (—).
10. Які особливості застосування операторів відношення в C#?
11. Які особливості застосування логічних операторів в мові C#?
12. Як працюють складені оператори присвоювання?
13. Які особливості застосування порозрядних операторів у мові C#?
14. Які особливості застосування операторів зсуву в C#?
15. Що собою представляють спеціальні оператори для особливих випадків?

ЛАБОРАТОРНА РОБОТА № 2
УМОВНІ ОПЕРАТОРИ ТА ЦИКЛИ

Мета роботи: навчитися застосовувати, відповідно до поставлених задач, умовні оператори та цикли.

2.1 Теоретичні відомості

2.1.1 Операторif

В C# ви можете описати логіку в коді програми, що дозволяє виконувати різні частини коду в залежності від стану даних в програмі. Ви можете, наприклад, запитати користувачів, чи хочуть вони зберегти зміни в файлі, який відкритий в програмі. Завдяки умовним операторам ви можете передбачити виконання коду на основі відповіді, наданої користувачем. В C# використовуються умовні оператори для досягнення цієї функціональності.

Основний умовний оператор в C# - це if. Альтернативою є switch.

В C# if пов'язаний з булевою логікою. Якщо значення true, то блок коду, що пов'язаний з цим значенням виконується. Якщо значення false, тоді рядок коду, що стоїть після логічного значення не виконується, у випадку блоку коду у фігурних дужках, не виконується цілий блок коду.

Наступний приклад демонструє використання if для того, щоб зрозуміти чи отримали ми у відповідь слово 'Привіт'.

Оператор if

```
string message = "Привіт";
if (message == "Привіт")
// рядоккоду, щобудевиконуватись, якщозначенняповідомлення (змінної message) "Привіт"
```

або

```
string message= "Привіт";
if (message == "Привіт")
{
    // тут потрібно розміщувати декілька рядків коду,
    //що будуть виконуватись, якщо відповідь"Привіт"
}
```

В C#iftакож може використовуватись зelse. Код післяelseвиконується, коли умовав операторіifє хибним (false).

Наступний приклад демонструє як використовувати ifelse , коли умова хибна (false).

Оператори if else

```
String message;
if (message == "Доброгодня")
{
    // Блоккоду, щобудевиконуватись, якщоувідповідьотримано "Доброгодня".
}
else
{
    // Блок коду, що буде виконуватись, якщо отримано будь-яку іншу відповідь.
}
```

ifоператор також може використовуватись разом selseif. Умовні вирази перевіряються послідовно в операторах if в порядку, в якому вони зустрічаються в коді. Якщо хоча б один If повертаєtrue, то виконується лише той блок коду, з яким він пов'язаний, після цього ми виходимо за межі конструкції умовних операторів.

Наступний приклад демонструє як використовувати if з elseif.

if else if

```
stringmessage;
if (message == "Доброгодня")
{
    //Блок коду, що буде виконуватись, якщо у відповідь отримано "Доброго дня".
}
elseif (message == "Допобачення")
{
    //Блок коду, що буде виконуватись, якщо у відповідь отримано "До побачення".
}
else
{
    //Блок коду, що буде виконуватись, якщо отримано будь-яку іншу відповідь, що не належить до відповідей, що містяться вище.
}
```

Ви можете додати настільки багатоelseifблоків наскільки вам потрібно для того, щоб правильно описати логіку програми. У випадку великої кількості варіантів ви может спробувати використати операторswitch.

2.1.2Операторswitch

Якщо у вас занадто багатоelseifвиразів, ваш код може стати не зрозумілим або занадто складим для розуміння. У цьому випадку краще використовувати**switch**. Операторswitchможе замінитиelseifвирази.

Наступний приклад демонструє як використовувати операторswitch

```
string message = "Привіт";
switch (message)
{
    case "Привіт":
    //Блоккоду, щобудевиконуватись, якщоувідповідьотримано"Привіт".
    break;
    case "Доброго дня":
    //Блок коду, що буде виконуватись, якщо у відповідь отримано"Доброго дня".
    break;
    case "До побачення":
    //Блок коду, що буде виконуватись, якщо у відповідь отримано"До побачення".
    break;
    default:
    //Блоккоду, щобудевиконуватись, якщоотримановідповідь, щоненалежитьдожодноїзвищезгаданих
    break;
}
```

Ви могли помітити блок, що позначений ключовим словомdefault:. Цей блок виконується, коли жоден з попередніх блоків не було виконано.

В кожномусаземіститься ключове словоbreak.Він використовується для того, щоб після проходження блоку коду, що задовільняє умові, жоден інший блок коду з конструкціїswitchне виконувався. Якщо ви не використаєте ключове словоbreak, код не буде компілюватись. Якщо вам потрібно обробити декілька варіантів тим же сегментом коду, ви можете використовувати конструкцію, що демонструється нижче.

```
stringmessage;
switch (message)
{
    case "Допобачення":
    //Блок коду, що буде виконуватись, якщо у відповідь отримано "До побачення".
    break;
    case "Доброго дня":
    case "Привіт":
    //Блок коду, що буде виконуватись, якщо у відповідь отримано "Доброго дня"
    // чи "Привіт";
    break;
    default:
    //Блок, що виконується, коли жодна з вищеперечислених умов не зустрілась.
    break;
}
```

Якщо ви знайомі з C, то ви можете помітити, що ви можете використовувати рядок в якості змінної для оператора switch в C#, тобто змінна не повинна бути лише цілочисельною чи переліченням. В C# оператор switch здійснює підтримку наступних типів:sbyte, byte, short, ushort, int, uint, long, ulong, char, string, enumerations.

C# передбачає ряд стандартних конструкцій, відомих як цикли, які можна використовувати для реалізації ітераційної логіки. Якщо ви вивчали інші мови програмування, то ви зможете впізнати ті ж for цикли, while цикли та dowhile цикли. C # підтримує усі ці три види циклів.

2.1.3Цикл for

У **циклifor**блок коду виконується кілька разівпоки вказана умова ([condition]) не стає хибною. Виможетевизначитициклнаступнимчином.

```
for ([initializers]; [condition]; [iterator])
{
    //код, щоповторюється
}
```

Ініціалізація[initializers]використоуєтьсядлязаданняпочатковогозначенняліч ильникациклу. Під час кожної ітерації перевіряється вказана умова, наприклад, чи лічильник належить проміжку значень, при яких виконується цикл.І якщо це так, то виконується тіло циклу. В кінці кожної ітерації циклу, лічильник[iterator]змінюється.

Наступний приклад показує як використовуватиforцикл для виконання коду 10 разів.

```
for (int i = 0 ; i < 10; i++)
{
    // код, щовиконується
}
```

Уцьомуприкладі,і = 0;цеініціалізація,і< 10;цеумова: колиумоваблоккодувиконуєтьсятаі++ – ітератор, щозбільшуєзначеннялічильниканаодиницю.

2.1.4 Циклwhile

Циклwhileдозволяєвамвиконуватиблоккодупокизначеннязаданоїумовивірне (true).Наприклад, виможетевикористовуватициклwhileдлятого, щобоброблятиданівведенікористувачемпокикористувачнеповідомитьнам, щоуньоговженемаєданихдлявведення.

Наступний приклад демонструє нам як використовувати циклwhile

```
stringresponse = "";
while (response != "Кінець")
{
    // Код, що відповідає за обробку даних.
    response = Console.ReadLine();
}
```

2.1.5Циклdo

Циклdoдужесхожийдowhileциклузавиключеннямтого, щоциклdовиконуєкод, щознаходитьсявтіліциклупринаймніраз. В цикліwhile, якщо умова хибна відразу, то тіло циклу ніколи не буде виконано.

Ви можете використати циклdoякщо код буде , наприклад, виконуватись у відповідь на введення даних користувачем.

```
do
{
    // Обробкаданих.
    response = Console.ReadLine();
} while (response != "Кінець");
```

2.2 Завдання до лабораторної роботи(Додаток Б)

2.3 Контрольні запитання:

1. В яких випадках доцільно застосовувати оператор умовного переходу?
2. Який вигляд має повна форма оператора умовного переходу if?
3. Який вигляд має скорочена форма оператора умовного переходу?
4. Приклади використання оператора if, що має повну форму представлення.
5. Приклади використання оператора if, що має скорочену форму представлення.
6. Приклади застосування вкладених операторів if.
7. Який вигляд та принцип роботи конструкції if-else-if?
8. Який загальний вигляд має оператор варіанту switch?
9. Яке призначення блоку default в операторі switch?
10. Приклади використання оператора switch без та з використання блоку default.
11. Що таке цикл і для чого вони потрібні?
12. Які бувають цикли?
13. Характеристика циклу for.
14. Характеристика циклу while.
15. Характеристика циклу do.

ЛАБОРАТОРНА РОБОТА № 3
СКЛАДНІ СТРУКТУРИ ДАНИХ C#

Мета роботи: навчитися створювати та працювати з масивами, структурами, з'ясувати поняття перерахування (переліченння), їх створення і використання.

3.1 Теоретичні відомості

3.1.1 Масиви

Платформа .NET включає різні вбудовані типи даних, такі якint, decimal, string, та boolean.Ці типи можуть бути ідентифіковані як прості типи даних, тому що вони складаються зєдиного, простого значення, такого як число, текстове значення, істина або хиба. Можна стверджувати, щоString – ценепростий тип даних і в деякій мірі це правда. C/C++ та інші подібні мови вважають, щоString – це символьний масив. Основна відмінність C# – те, що ми можемо використовувати значення типуStringв якості простої структури даних, не хвилюючись про доступ до нього як до символьного масиву.

Складні типи даних підходять для випадків, коли вам потрібно зберегти кілька елементів як єдине ціле. Розглянемо дні тижня, місяці року, колоду карт. Кожен з цих прикладів вимагає декількох значень, щоб відобразити поняття. Ми сприймаємо тиждень, як єдине ціле, що складається з семи днів. Коли ми говоримо приблизно тиждень, ми враховуємо те, з чого складається тиждень.

Масив являє собою набір об'єктів, які згруповані разом і управляються як єдине ціле. Ви можете думати про масив як про послідовність елементів однакового типу. Ви можете створювати прості масиви, які мають один вимір (список), два виміри (таблиці), три виміри (куб) і так далі. Масиви в C# мають такі особливості:

- ✓ Кожен елемент в масиві містить значення.
- ✓ Масиви індексуються з нуля, тобто перший елемент в масиві – елемент на позиції 0.
- ✓ Розмір масиву –загальна кількість елементів, які він містить.
- ✓ Масиви можуть бути одновимірні, багатовимірними, або зубчасті.
- ✓ Розряд масиву –число розмірностей в масиві.

Масиви певного типу можуть містити елементи тільки цього типу. Якщо вам необхідно здійснювати маніпуляції набором на відміну від роботи з об'єктом або типом значення, рекомендується використовувати один з типів колекцій, які визначені в просторі імен System.Collections.

3.1.2 Створення і використання одновимірних масивів

Коли ви оголошуєте масив, ви вказуєте тип даних, який він містить та ім'я масиву. Оголошення масиву насправді не виділяє для нього пам'ять. CLR фізично створює масив, коли ви використовуєте ключове слово new і також саме тоді ви повинні й задати розмір масиву.

Щоб оголосити **одновимірний масив**, ви вказуєте тип елементів в масиві і використовуєте дужки [], щоб вказати, що змінна –масив. Пізніше,

ви визначаєте розмір масиву, коли виділяєте пам'ять для масиву за допомогою ключового слова new. Розмір масиву –ціле число.

Наступний приклад коду показує, як створити одновимірний масив цілих чисел, починаюючи з нульового елементу і завершуючи дев'ятим.

```
int[] arrayName = newint[10];
```

В наступному прикладі, ми оголошуємо масив і присвоюємо йому значення.

Компілятор знає наскільки великим робити масив, щоб він містив значення, що задаються у фігурних дужках.

```
int[] arrayName = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

3.1.3Доступ до даних в масиві

Ви можете отримати доступ до даних в масиві декількома способами, наприклад, шляхом визначення індексу конкретного елемента, або ітерацією через весь масив, повертаючи кожен елемент послідовно.

У наступному прикладі коду використовується індекс для доступу до елементу з індексом два.

```
//Доступ до елементів масиву за індексом
int[] oldNumbers = { 1, 2, 3, 4, 5 };
//number буде містити значення 3
intnumber = oldNumbers[2];
```

Примітка: Масиви починаються з нульового індекса, тому перший елемент в масиві будь-якої розмірності з індексом нуль. Останній елемент має індекс N-1, де N є розмірність. При спробі доступу до елементу поза цим діапазоном CLR генерує виняток(помилку) IndexOutOfRangeException.

Ви можете перебрати масив, використовуючи для цього цикл. Ви можете використовувати властивість Length масиву, щоб визначити, коли зупинити цикл.

У наступному прикладі коду показано, як використовувати цикл для перебору масиву.

```
//Перебір всіх елементів масиву
int[] oldNumbers = { 1, 2, 3, 4, 5 };
for (int i = 0; i < oldNumbers.Length; i++)
{
    int number = oldNumbers[i];
}
```

3.1.4 Багатовимірнімасиви

Масиви можуть мати розмірність більше, ніж 1. Число розмірностей відповідає числу індексів, які використовуються, щоб ідентифікувати окремий елемент в масиві. Ви можете визначити до 32 розмірностей, але вам рідко будуть потрібні більше, ніж 3. Ви оголошуєте **багатовимірну змінну типу масив**, так як ви оголошуєте одновимірний масив, але ви розділяєте розмірності комами.

Наступний приклад коду показує, як створити масив цілих чисел з двома размірностями.

```
// Створюємо масив, що довжиною 10 елементів (рядок) і ширина - 10 елементів (стовпчик)
int[ , ] arrayName = newint[10,10];
```

Для того, щоб отримати доступ до елементів в багатовимірному масиві, ви повинні вказувати усі індекси, як в прикладі коду нижче.

```
// Досуп до першого елемента першого рядка першого стовпчика
intvalue = arrayName[0,0]
// Досуп до першого елемента першого рядка другого стовпчика
intvalue2 = arrayName[0, 1];
// Досуп до першого елемента дрягого рядка першого стовпчика
intvalue2 = arrayName[1, 0];
```

3.1.5Зубчасті масиви

Зубчастий масив –це масив масивів, і розмір кожного масиву може бути різний. Зубчасті масиви можуть бути використані для моделювання розріджених структур даних, де, можливо, не завжди потрібно виділяти пам'ять для кожного елемента, якщо вона не буде використовуватися. Наступний приклад коду показує, як оголосити і задати зубчастий масив. Зверніть увагу на те, що Ви повинні визначити розмір першого масиву, але Ви не повинні визначати розмір масивів, які містяться в цьому масиві. Ви виділяєте пам'ять кожному масиву в зубчастому масиві окремо, за допомогою ключового слова new.

```
int[][] jaggedArray = new int[10][];
jaggedArray[0] = newType[5]; // можназадаватибудь-якийрозмір.
jaggedArray[1] = new Type[7];
...
jaggedArray[9] = newType[21];
```

3.1.6Перерахування(перелічення)

Перерахування –є структурою, яка дозволяє створити змінну з фіксованим набором можливих значень. Найбільш поширеним прикладом є використання перерахування для визначення дня тижня. Є тільки сім можливих значень для днів тижня, і ви можете бути впевнені, що ці значення не зміняться.

Найкращою практикою є задання перерахування безпосередньо поблизу простору імен так, щоб всі класи в цьому просторі імен мали б доступ до нього, якщо це необхідно. Можна також помістити ваші змінні з перерахування всередині класів або структур.

За замовчуванням значення перерахувань починаються з 0 і кожний наступний член збільшує значення індекса на 1.

3.1.7Створення і використання перерахувань

Щоб створити перерахування, ви оголошуєте його в коді за допомогою наступного синтаксису, який демонструє створення перерахування під назвоюDay,який містить дні тижня:

```
enum Day { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };
```

За замовчуванням значення перерахувань починаються з 0 і кожний наступний член збільшується на в 1. В результаті, попереднє перерахування 'День' буде містити значення:

Sunday = 0
Monday = 1
Tuesday = 2

Ви можете змінити значення за замовчуванням, вказавши початкове значення для вашого перерахування, як показано в наступному прикладі

enum Day { Sunday = 1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

У цьому прикладі, неділі присвоюється значення 1 замість 0.

Зараз Monday = 2, Tuesday = 3, і т.д.

Ключове словоenumвикористовується для вказівки "типу"якого буде змінnaDay. Перерахування підтримуютьвбудовані типи даних і можна використовувати один з наступних:byte, sbyte, short, ushort, int, uint, long, ulong.

Для того, щоб змінити тип даних за замовчуванням для вашого перерахування, ви можете вказати тип даних зі списку вище, так як:

enum Day : short { Sunday = 1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

Базовий тип визначає, скільки пам'яті буде виділено для кожного лічильника в перерахуванні. Під час компіляції, ваше перерахування буде конвертоване в числовий літерал в коді. Якщо ви використовуєте VisualStudio, функція Intellisense цілком здатна розпізнати ваше перерахування і буде автоматично відображати строкові значення в IDE, коли ви вводите ім'я enum.

Важливо відзначити, що вам необхідно буде використовувати явне приведення, якщо ви хочете перетворити з значення типу enum до іншого типу. Розглянемо наступний приклад, в якому оператор присвоює значення перелічення Sun до типу ште, щоб перетворити перерахування в ціле значення.

int x = (int)Day.Sun;

Використання перерахувань

Day favoriteDay = Day.Friday;

Використання перерахувань має ряд переваг в порівнянні з використанням тексту або числових типів:

✓ Покращена керованість. Обмежуючи змінну фіксованим набором допустимих значень, ви менше схильні використовувати неприпустимі аргументи і робити помилки.

✓ У VisualStudio, функція IntelliSense підказує вам можливі значення при використанні enum.

✓ Краще читається код. Синтаксис перерахування робить ваш код зрозумілішим.

Кожен член перерахування має ім'я і значення. Ім'я є рядком, щовизначається в фігурних дужках, наприклад, неділя або понеділок. За замовчуванням, значення є цілим числом. Якщо ви не вкажете значення для кожного члена, їмпризначаються значення у порядку зростання починаючи з 0. Наприклад, Day.Sunday дорівнює 0 і Day.Monday дорівнює 1.

У наступному прикладі показано, як можна використовувати імена і значення для присвоєння:

// Задання змінної перелічення за іменем.
DayfavoriteDay = Day.Friday;

```
// Задання змінної перелічення за значенням.
DayfavoriteDay = (Day)4;
```

3.2 Завдання до лабораторної роботи(Додаток В)

3.3. Контрольні питання:

1. Приклад оголошення змінної, що є структурою.
2. Що таке масив
3. Які бувають масиви?
4. Опис синтаксису масивів.
5. Охарактеризуйте масиви як об'єкти.
6. Охарактеризуйте одномірні масиви.
7. Охарактеризуйте багатовимірні масиви.
8. Охарактеризуйте масиви масивів.
9. Використання оператора foreach з масивами.
10. Передача масивів в якості аргументів.
11. Передача масивів за допомогою параметрів ref і out.
12. Неявно типізовані масиви.
13. Приклад оголошення змінної що є перечисленням (зчисленням).

ЛАБОРАТОРНА РОБОТА № 4
МЕТОДИ

Мета: навчитись працювати з методами, з'ясувати їх використання.

4.1 Теоретичні відомості

4.1.1 Методи

Можливість визначати та використовувати **методи** – це фундаментальний компонент ООП, тому що методи дозволяють інкапсулювати операції, що зберігають дані всередині типу.

Зазвичай кожний додаток, що Ви розробили за допомогоюMicrosoft .NETFrameworktaC#, матиме багато методів, у кожного з яких є своя окрема мета. Деякі методи необхідні для функціонування додатка. Наприклад усі додатки, що розроблені наC# для персональних комп'ютерів, повинні мати метод з назвою**Main**. Цей метод слугує точкою входу в додаток. Коли користувач запускає такий додаток,CLRвиконає метод**Main**в першу чергу.

Методи можуть бути розроблені для внутрішнього використання у класі та бути недоступними для інших класів. Існують також методи з ідентифікатором доступу*public* – цедозволить іншим класам звернутися до цього методу.

Взагалі .NETFrameworkскладається з класів, що мають методи, які Ви можете викликати з Вашого додатку для взаємодії з користувачем чи з комп'ютером.

Для визначення методу нам потрібно описати метод та створити його тіло.Опис дозволяє нам визначити ідентифікатор доступу, тип даних, що метод буде повертати, назву методу та параметри. **Тіло методу** – це реалізація того функціоналу, що повинен мати метод.

Кожну частину опису можна пояснити так:

- **Ідентифікатор доступу** – відповідає за доступність нашого методу (хто може викликати цей метод)
  - ✓ *private* – дозволяє викликати метод тільки всередині класу чи структури;
  - ✓ *protected* – дозволяє викликати цей метод з класу, де метод визначено та з похідних від нього;
  - ✓ *public* – дозволяє викликати цей метод з від усюди;
  - ✓ *internal* – дозволяє викликати цей метод всередині зборки;
  - ✓ *static* – визначає, що метод статичний член класу, а не член якогось об'єкта;
- **Тип даних**, що повернеться – визначає, який тип даних буде повернуто. Якщо метод нічого не повертає, потрібно вказати*void*.

- **Назва метода** – усі методи повинні мати назви, інакше ми не зможемо викликати їх.
- **Параметри** – це список параметрів, що метод приймає у якості вхідних даних. Якщо параметрів декілька, їх треба розділити комою.

Приклад простого методу:

```
public Boolean StartService(string serviceName)
{
    //реалізація методи, що починає якийсь сервіс
}
```

У цьому прикладі*public*– ідентифікатор доступу,*Boolean*– тип даних, що буде повернуто,***StartService***– назва методу,*stringserviceName*– параметр. Зверніть увагу, що параметр складається з типу даних та назви параметра.DeclaringMethods

Ви викликаєте метод з вашого додатка для того, щоб виконати код, який є реалізацією цього методу. Вам не потрібно розуміти, як працює цей код. Ви навіть на матимете можливість переглянути код, якщо цей метод у збірці, до якої у Вас немає доступу, наприклад, бібліотеки .NET Framework.

Для виклику методу Ви повинні вказати його назву та, якщо потрібно, додати параметри, які підтримуються цим методом.

Наступний приклад покаже як викликати метод *StartService*, передати декілька параметрів типу*int* та *bool*, що задовольняють вимогам цього методу.

```
var uptime = 2000;
var shutdownAutomatically = true;
StartService(uptime, shutdownAutomatically);
//метод StartService
void StartService(int uptime, bool shutdownAutomatically)
{
    //реалізація методу
}
```

Якщо метод повертає якесь значення, Ви повинні вказати як оброблювати це значення, зазвичай, присвоюючи у Вашому коді результат виконання цього методу до якоїсь змінної такого ж типу.

Наступний приклад покаже як отримати результат роботи методу ***GetServiceName***присвоюючи його до змінної *serviceName*.

```
string serviceName = GetServiceName();
//метод GetServiceName
string GetServiceName()
{
    return “FourthCoffee.SalesService”;
}
```

Цей приклад показав як повернути одне значення, але можуть бути такі ситуації, коли потрібно повернути декілька значень. Існує три підходи, які Ви можете використати, щоб досягти цього:

- Повернути масив чи колекцію
- Використовувати ключове слово ***ref***

- Використовувати ключове слово ***out***

У наступному прикладі ми викликаємо метод *ReturnMultiOut*. Параметри передаються з ключовим словом *out*.Це вказує на те, що змінні передається по “посиланню” і повинна бути визначена всередині методу. Зверніть увагу, що нам не потрібно призначати результат виконання цього методу до якоїсь змінної як у попередньому прикладі.

```
int first;
sting sValue;
ReturnMultiOut(out first, out sValue);
Console.WriteLine($“{first.ToString()}, {sValue}”);
static void ReturnMultiOut(out int i, out string s)
{
    i = 25;
    s = “usingout”;
}
```

У наступному прикладі використовується ключове слово *ref*для того, щоб повернути декілька значень з методу. Зазвичай, ключове слово refвимагає, щоб змінна була попередньо ініціалізована або вже використана.

```
sValue = string.Empty;
ReturnMultiRef(ref first, ref sValue);
Console.WriteLine($“{first.ToString()}, {sValue}”);

static void ReturnMultiRef(ref int I, ref string s)
{
    i = 50;
    s = “usingref”;
}
```

Коли Ви визначаєте метод, ви можете закласти в нього різні варіанти змінних за різних обставин. Для цього потрібно перевантажити метод, щоб створити декілька методів які реалізують той самий функціонал, але за умови, що вони приймають різні параметри в залежності від контексту, де ці методи викликаються.

Перевантажені методи мають однакову назву для того, щоб акцентувати своє призначення, але мають різний набір параметрів. Саме це і робить метод перевантаженим.

Ви не зможете визначити перевантажений метод який відрізняється від оригінального лише типом даних, що буде повернуто. Також не потрібно перевантажувати метод тільки для того, щоб змінити послідовність параметрів.

Наступний приклад покаже три версії методу **StopService**, усі з різними наборами параметрів.

```
voidStopSerive()
{
    //ця версія не приймає жодних параметрів
}
void StopService(string serviceName)
{
//ця версія приймає параметр типу string
}
```

```
void StopService(int serviceID)
{
    //ця версія приймає параметр типу int
}
```

Коли Ви викликаєте StopSerive, Ви будете мати вибір з цих перевантажених методів. Просто вкажіть ті аргументи, що задовольняють потрібне перевантаження, і компілятор вирішить який саме метод потрібно буде викликати залежно від того, які параметри ви передали.

Одна з ключових особливостей C# це можливість працювати з додатками та компонентами, що розроблені з використанням іншим технології. Один з таких принципів, що використовується yWindows – Component Object Model, або просто COM. COMне підтримує перевантаження методів, але використовує методи, які можуть приймати необов'язкові параметри. Для простішої взаємодії з COM та компонентами, C# підтримує необов'язкові параметри.

Необов'язкові параметри також корисні в інших ситуаціях. Вони забезпечують компактні та прості рішення коли неможливо використовувати перевантаження. Коли Ви визначаєте метод, що підтримує необов'язкові параметри дуже важливо зазначити спочатку обов'язкові параметри і тільки далі зазначити необов'язкові параметри.

Наступний приклад показує як створити метод, який приймає один обов'язковий параметр (forceStop) та два необов'язкових (serviceName, serviceID). Зверніть увагу, що механізм визначення необов'язкових параметрів вимагає значення за замовчуванням.

```
void StopService(bool forceStop, string serviceName = null, int serviceID = 1)
{
    //код
}
```

Ви можете викликати метод з необов'язковими параметрами так само як і інші методи. Потрібно вказати назву методу та забезпечити усі необхідні параметри. Різниця полягає в тому, що ви можете не назначувати необов'язкові параметри. У цьому випадку компілятор використає значення за замовчуванням.

Зазвичай, коли Ви Викликаєте метод, послідовність параметрів буде відповідати послідовності параметрів, саме так, як зазначено у описі методу. Якщо параметри несумісні або використовуються не в тій послідовності, Ви отримаєте помилку компіляції.

У VisualC# Ви можете зазначати параметри за їх ім'ям і тому стає неважливим у якій послідовності Ви вказуєте їх. Для того, щоб використовувати іменовані параметри, Ви повинні вказати назву параметра та значення, розділивши їх двокрапкою.

Наступний приклад показує як викликати метод StopServiceвикористовуючи іменований параметрserviceID.

```
StopService(true, serviceID: 1);
```

Ви можете комбінувати позиційні та іменовані параметри за умови, що позиційні параметри завжди зазначені перед іменованими.

4.1.2Обробка помилок

Вийняток – це ознака помилки або виняткової ситуації. Метод може створити виняток якщо буде знайдено щось несподіване, наприклад, додаток намагається відкрити файл, якого не існує в системі.

Коли метод створює вийняток, то код, що викликав цей метод, повинен бути готовим до обробки цього винятку(містити блок try/catch, про який буде сказано пізніше). Якщо цей код не готовий до оброки (наприклад, його немає), то код, що іде після виконання методу, що кинув виняток, буде скасований та виняток буде поширюватися вгору до тих пір, поки не знайдеться код, який зможете обробити цю помилку. Якщо не було знайдено такого коду, то виникне повідомлення для користувача та весь додаток закриється.

Ключова програмна конструкція, що дозволяє Вам реалізувати структурну обробку винятків у Вашому додатку – це блок try/catch. Вам потрібно огорнути код, що може створити виняток у блок trута додати один чи декілька блоків catchдля обробки винятків. Рекомендується спочатку обробляти більш специфічні помилки і тільки потім узагальнені. Наприклад, якщо Ви працюєте з файловою системою, краще спочатку обробити виняток FileNotFoundException (у першому блоці catch), а далі більш узагальнений виняток Exception, який спробує обробити будь-які винятки.

Наступний приклад показує як працювати з конструкцією try/catch.

```
try
{
    //Код, який може створити виняток
}
catch(FileNotFoundExceptionex)
{
    //Обробка, якщо не було знайдено файл
}
catch (Exceptionex)
{
    //Обробка, якщо виникло щось несподіване
}
```

Інколи методи повинні складатися з коду, що повинен виконуватися завжди, навіть якщо є виняток. Наприклад, метод повинен переконатися, що файл буде закрито перед тим, як вивільняться ресурси. Блок finallyдозволяє це зробити.

Цей блок потрібно визначати після усіх блоків catch у конструкції try/catch. Код у цьому блоці буде виконано незалежно від того, чи було все добре, чи були винятки, навіть якщо оброблені. Якщо буде створено виняток, то його обробка буде зроблена до того, як виконається код з блоку finally.

Ви також можете додати блок finally без визначення блоків catch. У цьому випадку, усі виключення залишаться необробленими, але код у блоці finallyвсе одно буде виконано.

Наступний приклад покаже як використовувати конструкцію try/catch/finally.

```
try
{
}
catch (NullReferenceException ex)
{
}
catch (Exception ex)
{
}
finally
{
    //цей код виконається завжди
}
```

Ви можете створити екземпляр класу винятків у Вашому коді і далі поширити його. Коли ви поширюєте виняток, виконання коду припиняється та CLRпередає контроль до першого доступного обробника таких винятків.

Для того, щоб поширити виняток використовуйте ключове слово throwта далі вкажіть об'єкт винятку.

Наступний код показує як створити виняток NullReferenceException та поширити його.

```
var ex = new NullReferenceException("The 'Name' parameter is null");
throw ex;
```

Поширена стратегія – це намагатися спіймати та обробити будь-який виняток. Якщо обробник не може вирішити проблему, він може поширити виняток далі. Наприклад:

```
try {}
catch (NullReferenceException ex) {}
catch (Exception ex)
{
    //Намагаємося обробити, але не можемо. Поширюємо далі.
throw;
}
```

4.2 Завдання до лабораторної роботи(Додаток Г)

4.3 Контрольні запитання:

1. Поняття методу.
2. Різниця між простими і статичними.
3. Оголошення методів.
4. Параметри ref і out.
5. Перевантаження методів.
6. Змінне число параметрів методу.
7. Віртуальні методи.
8. підміна методів.
9. Метод Main ().
10. Використання методів в програмі.

ЛАБОРАТОРНА РОБОТА №5
СТРУКТУРИ, РОЗШИРЕННЯ СТРУКТУР

Мета: опис поняття та використання структур, робота зі створення та розширення структур.

5.1 Теоретичні відомості

5.1.1 Структури

В C# **структура** – це програмна конструкція, яку можна використовувати для визначення типу користувачем. Це по суті структура даних, що представляє собою пов'язані фрагменти інформації в якості окремої сутності. Наприклад:

✓ структура, що називається Point, може складатися з полів для заданнях-координати та y-координати.

✓ структура, що називається Circle, може складатися з полів для представлення координати x, координати y та радіуса.

✓ структура з ім'ям Color може складатися з полів для задання червоної, зеленої та синьої компонент.

Більшість вбудованих типів в C#, таких як int чи bool, визначаються структурами. Ви можете використовувати структури, щоб створювати свої власні типи, які ведуть себе як вбудовані типи.

5.1.2 Створенняstruct

Для оголошення структур використовується ключове слово **struct**так, як показано на наступному прикладі:

```
//Задання Struct
public struct Coffee
{
   public int Strength;
   public string Bean;
   public string CountryOfOrigin;
   // Іншіметоди, поля, властивостітаподії
}
```

Ключовому слову**struct**у наведеному вище прикладіпередує модифікатор доступу public, що задає область видимості, тобто визначає, де ви можете використовувати цей тип. Ви можете використовувати наступні модифікатори доступу при оголошенні struct:

✓ public - код доступний у будь-якій збірці.

✓ internal - доступний для використання в тій же збірці, але не доступний для коду з іншої збірки. Це значення за замовчуванням, якщо ви не вказуєте модифікатор доступу.

✓ private - доступний тільки для коду всередині структури, яка містить його. Ви можете використовувати лише модифікатор доступу privite з вкладеними структурами.

Структури можуть містити безліч членів, включаючи конструктори, поля, константи, властивості, індексатори, методи, оператори, події і навіть вкладені типи. Майте на увазі, що структури призначені для того, щоб бути легкими, тому, якщо ви виявили багато методів, конструкторів і подій, ви повинні розглянути питання використання замість структури класу.

5.1.3 Використання структури

Для створення структури, ви використовуєте ключове слово new, як продемонстровано в прикладі нижче:

```
Coffee coffee1 = new Coffee();
coffee1.Strength = 3;
coffee1.Bean = "Arabica";
coffee1.CountryOfOrigin = "Kenya";
```

5.1.4Ініціалізація структури

Ви могли помітити, що синтаксис інстанціалізації структури, наприклад,newCoffee();схожий на синтаксис для виклику методу. Це відбувається тому, що коли виреалізуєте структуру, ви насправді викликаєте особливий тип методу, що називається конструктором. Конструктор являє собою метод в структурі, яка має ту ж назву, що і структура.

Коли ви реалізуєте структуру без аргументів, так якnewCoffee();ви викликаєте конструктор за замовчуванням, який створюється за допомогою компілятора.Якщо ви хочете мати можливість задати значення полів за замовчуванням, коли ви реалізуєте структуру, ви можете додати конструктор, що приймають параметри для вашої структури.

У наступному прикладі показано, як створити конструктор для структури.

Додавання конструктора

```
publicstructCoffee
{
  // Конструкторзпараметрами.
   public Coffee(int strength, string bean, string countryOfOrigin)
   {
      this.Strength = strength;
      this.Bean = bean;
     this.CountryOfOrigin = countryOfOrigin;
   }
  // Вирази визначають поля структури та задають значення за замовчуанням.
  public int Strength;
   public string Bean;
  publicstringCountryOfOrigin;
    // Інші методи, поля, властивості та події.
}
```

У наступному прикладі показано, як використовувати цей конструктор для створення екземпляра класу Coffee.

Виклик конструктора

```
// Виклик конструктора з параметрами.
Coffee coffee1 = new Coffee(4, "Arabica", "Colombia");
```

Ви можете додати кілька конструкторів до вашої структури. Кожен конструктор може приймати різні комбінації параметрів. Тим не менш, ви не можете додати конструктор за замовчуванням(без параметрів) для структури, так як він створюється компілятором.

5.1.5Розширення структур

Для того, щоб вийти за рамки простої структури, ви можете розширити її, додавши властивості та індексатори(indexers). у структурах.

Створення властивостей

В C# **властивість**(property) – це програмна конструкція, яка дозволяє з клієнтського коду отримувати або встановлювати значення приватних полів всередині структури або класу. Для споживачів вашої структури або класу властивість задається як поле. У вашій структурі або класі властивість реалізується з використанням аксессорів(accessors), які представляють собою особливий тип методу. Властивість може включати в себе один або обидва з наступних дій:

- Аксессор доступу, щоб забезпечити доступ для читання поля.
- Аксессор запису, щоб забезпечити доступ для запису значення поля.

У наступному прикладі показано як реалізувати властивістьStrengthв структурі:

```
public struct Coffee
{
   private int strength;
   public int Strength
  {
      get { return strength; }
     set { strength = value; }
   }
}
```

getаксессорвикористовує ключове словоreturn, щоб повернути значення приватного поля.

setаксессорвикористовує спеціальну локальну змінnyvalue, що містить значення, що передається при "присвоєнні" цій властивості значення.

Наступний приклад показує, як використовувати властивість структури:

```
//Використаннявластивості
Coffeecoffee1 = newCoffee();
coffee1.Strength = 3;
intcoffeeStrength = coffee1.Strength;
```

Код, що використовує властивість структури, виглядає як ніби робота відбувається з відкритим полем. Проте використання властивостей має декілька переваг над використанням просто відкритий полів структури(класу):

- Ви можете використовувати властивості, щоб контролювати доступ до полів. Якщо зазначити лишеаксессорget,можливості запису не буде(лише для зчитування). І навпаки, якщо зазначити лишеаксессорset,можливості зчитуванняне буде(лише можливість запису)

```
// Цю властивіть можна буде тільки зчитати
```

```
// код вигдяу coffee1.Strength = 3; призведе до
// помилки
public int Strength
{
   get { return strength; }
}
// в цю властивість можна лише записати значення
public string Bean
{
   set { bean = value; }
}
```

- Ви можете змінити реалізацію властивостей, не зачіпаючи код клієнта.

Наприклад, ви можете додати логіку перевірки, або викликати метод замість читання значення поля.

```
public int Strength
{
   get { return strength; }
   set
   {
      if(value < 1)
      { strength = 1; }
      else if(value > 5)
      { strength = 5; }
      else { strength = value; }
    }
}
```

Якщовихочетестворитивластивість, якапростоотримуєівстановлюєзначенняприватногополябезбудь-якоїдодаткової логіки, виможетевикористовуватискороченийсинтаксис.

Щоб створити властивість, яка зчитує і записує в приватне поле, ви можете використовувати наступнийзапис:

```
public int Strength { get; set; }
```

Щоб створити властивість, яка зчитує з приватної поля, ви можете використовувати наступний запис:

```
publicintStrength { get; }
```

Щоб створити властивість, яке записує в приватне поле, ви можете використовувати наступний запис:

```
publicintStrength { set; }
```

Для кожногоскороченого випадкукомпілятор неявно створить приватне поле для властивості.

Створення індексаторів

У деяких сценаріях Ви могли б хотіти використовувати структуру або клас як контейнер для масиву значень. Наприклад, Ви могли б створити структуру, щоб представляти напої, що доступні в кафе. Структура могла б використовувати масив рядків, щоб зберегти список напоїв.

Наступний приклад показує структуру, яка включає в собі масив:

```
public struct Menu
{
```

```
    public string[] beverages;
    public Menu(string bev1, string bev2)
    {
       beverages = new string[] { "Americano", "Café au Lait", "Café Macchiato",
"Cappuccino", "Espresso", bev1, bev2 };
    }
}
```

У цьому випадку, щоб отримати, наприклад, перший напій зі списку потрібно було б написати так:

```
Menu myMenu = new Menu(/*параметри*/);
stringfirstDrink = myMenu.beverages[0];
```

Було б зручніше, якби можна було б записати так:stringfirstDrink = myMenu.beverages[0].Ви можете зробити це, якщо написати **індексатор**. Щоб оголосити індексатор, Ви використовуєте ключове словоthis.

Наступний приклад показує, як визначити індексатор для структури:

```
public struct Menu
{
   private string[] beverages;
   // індексатор   publicstringthis[intindex]
   {
     get { returnthis.beverages[index]; }
     set { this.beverages[index] = value; }
   }
   // Дозволяє зрозуміти кількість елементів
  publicintLength
   {
     get { returnbeverages.Length; }
   }
}
```

Ви використовуєте наступний синтаксис, щоб отримати напої зі списку:

```
Menu myMenu = new Menu();
string firstDrink = myMenu[0];
intnumberOfChoices = myMenu.Length;
```

Ви можете створити індексатор тільки для зчитування або тільки для запису (так само як і для властивості)

5.2 Завдання до лабораторної роботи(Додаток Д)

5.3 Контрольні запитання:

1. C # структури. Приклади.
2. C # масив структур.
3. Конструктор структури C #.
4. C # структури даних.
5. Відмінність структури від класу C #.

ЛАБОРАТОРНА РОБОТА №6
КЛАСИ

Мета: навчитися створювати та працювати з класами.

1.1 Теоретичні відомості

6.1.1 Створення класів та об'єктів

Класидозволяютьстворювативласніцілісністабагаторазовіувикористанні типи.Інтерфейси дають змогу визначатипринципипобудови,якіреалізовуютькласидлясумісностізкласами користувача.

УС#виможетевизначитивласнийтипствоpившиклас.Якпрограмнаконстр укція,класив C# єосновноючастиноюоб'єктно-орієнтованогопрограмування.Створюючиклас,вивизначаєтеструктурудлятип у.Класвизначаєхарактеристикитаповедінкукожноїсутностів логічнийтарозширювальнийспосіб.Вireалізуєтеповедінкувизначаючиметоди,поля,властивостітаподіїусвоємукласі.

ПрипустимовиствориликлacDrinksMachine.

Вивикористалиключовесловоclassдляоголошенняклacу,якпоказановприкладі:

```
publicclassDrinksMachine
{
  //методи,поля,властивостітаподії
}
```

Ключовесловоclassйдепіслябудь-якогомодифікаторудоступу.В даномуприкладі використовуєтьсямодифікаторудоступуpublic.

Додаванняполейдокласів

Длявизначенняповедінкинашогоавтоматузпродажунапоїв Вам потрібно дадатиполятавластивості, наприклад,такіякмодель автомату,вік, ...Вистворитеметоди автомату,такіяквиготовленняеспрессочикапучіно. Вкінцівивизначитеподіїяківідображатимуть щось,щовимагаєвашоїуваги(наприклад, що кава закінчилася).

Усвомукласівиможетедодаватиметоди,поля,властивостітаподіїщовизначаютьповедінкутахарактеристикивашоготипутакякпоказановприкладі:

```
publicclassDrinksMachine
{
 // Приватнеполе
 private string _location;

 // Конструктор
 public DrinksMachine(string model, string make)
{
Model = model;
Make = make;
 }
```

```
  // Властивість
publicstringLocation
  {
get { return _location; }
set
    {
if (value != null)
        _location = value;
    }
  }

  // Властивості
publicstringMake { get;  }
publicstringModel { get;  }
  // Оголошення методів
publicvoidMakeCappuccino()
  {
    // Код методу
  }

publicvoidMakeEspresso()
  {
    // Код методу
  }

  // Оголошення події
publiceventOutOfBeansHandlerOutOfBeans;
}
```

6.1.2 Частковікласи

C#надаєможливістьреалізаціїчастковихкласів.**Частковікласи**надаютьможливістьрозділятиреалізаціюкласувдекількохфайлах,якіприкомпіляції"поєднаються"водинзагальнийфайл.

Частковікласикориснівтакихвипадках:

✓ Роботанадвеликимпроектом.Розділенняклассуміждекількомафайламинадаєможливістьдекількомпрограмістампрацюватинадоднимкласомодночасно.

✓ Підчасроботиз автоматично згенерованимкодом. VisualStudioвикористовуєданийпідхідпідчасроботизWindowsForms, Webсервісамитаіншими. Microsoftрекомендуєнезмінюватиавтозгенерованийкоддляданихкомпонентів,бовониможутьзмінитисяпідчаскомпіляціхчизмінипроекту.Якрішення,виможетестворитиінший класякчастковийзтакимжеіменемтареалізуватисвоїзмінитам.

Прикладчастковокласу:

```
public partial class DrinksMachine
{
publicvoidMakeCappuccino()
  {
    // Логіка методу тут
  }
}
```

```
publicpartialclassDrinksMachine
{
publicvoidMakeEspresso()
  {
    // Логіка методу тут
  }
}
```

Додаток:витакожможетерозділятиструктуритаінтерфейсиміждекількомафайлами.

6.1.3Ініціалізаціякласу

Клас–цетількиописсструктуридлятипу.Длявикористанняповедінкитахарактеристик,яківивизначиливкласі,потрібностворитиоб’єктданогокласу.

Длястворенняоб’єктукласу,використовуйтеключовесловo*new*такяквприкладі:

```
// Створення об'єкта
DrinksMachinedm = newDrinksMachine();
```

При**ініціалізації**класувтакийспосібвідбуваютьсянаступніречі:

- Вистворюєтеновийоб’єкттипуDrinksMachinevпам’яті.
- Вистворюєтевказівник(змінну)зіменемdmякийпосилаєтьсянановийоб’єктDrinsMachine.

Приствореннізмінноїзамістьявноговизначеннятипу DrinksMachinevиможетенадатиможливістькомпіляторувизначититиппідчаскомпіляції.Данийспосібназиваєтьсяавтоматичнимвизначеннямтипу.Длявикористанняспособувизначеннятипу,вистворюєтеоб’єктзадопомогоюзключовимсловом*var*такяквприкладі:

```
// Створення об'єкт
vardm = newDrinksMachine();
```

Вданомувипадкукомпіляторнезнаєточноготипузмінноїdm.Колизміннаініціалізуєтьсяякпосиланнянаоб’єктDrinksMachinekомпіляторприсвоюєїйтипDrinksMachine.Використаннявизначеннятипувтакийспосібневикликаєзмінвроботіпрограми,цевикористовуєтьсядляуникненняповтореньіменкласів.

Впевнихвипадкахвизначеннятипупокращуєчитабельністькоду, а в деяких – ні.Загальнеправило–використовувати автоматичневизначеннятипу,колитипзмінноїабсолютнозрозумілий.

Післяініціалізаціїоб’єктувиможетевикористатибудь-якуйогочастину) – метод,поле,властивістьчиподію,яківивизначитивкласі,такякпоказановприкладі:

```
var dm = new DrinksMachine("Beancrusher 3000", "Fourth Coffee");

Console.WriteLine(dm.Model);

dm.Location = "Somewhere near Kyiv";

dm.MakeEspresso();
```

6.1.4 Інкапсуляціяв C#

Інкапсуляція, що частовважаєтьсяголовнимстовпомвоб’єктно-орієнтованомупрограмуванні,використовуєтьсядлявизначеннядоступудочленакласучиструктури.

C#використовуєінкапсуляціюдляконтролюдоступудополівтавластивостейкласів.Дехторозглядаєконтрольдоступудочленівкласуякчастинуінкапсуляції,
адехторозглядаєїїяквкладенняконтролюповедінкичленівкласувреалізаціїкласу.Другепоясненняможебутичастковорозширенев
C#зарахунокможливостівикористаннячастковихкласів.

Private, Public, ProtectedmaInternal

Поданатаблиця Е.1 (додаток Е)відображаємодифікаторидоступуякіможутьбутизастосованідочленівкласутаяквониконтролюютьдоступіншогокодупрограми.

Властивості

Як
своєріднучастинуінкапсуляціїможнарозглядативластивостіувашомукласі.Властивостінадаютьможливістьконтролюдоступукористувачадозміннихувашомукласі.Властивостімістятьзакритуреалізаціютаперевіркуправильностівиконуваногокоду.Дляприкладувампотрібноперевірятидатународження,щовонавправильномуформатічизнаходитьсявкоректномупроміжку.

НижчеподаноприкладреалізаціївластивостейдлякласуDrinksMachine:

```
public class DrinksMachine
{
  private int age;
  private string make;

public int Age
      {
        get { return age; }
        set { age = value; }
      }

public string Make
      {
        get { return make; }
        set { make = value; }
      }

public string Model { get; set; }

public DrinksMachine(int age)
      {
        this.Age = age;
      }

public DrinksMachine(string make, string model)
      {
        this.Make = make;
```

```
    this.Model = model;
  }

  public DrinksMachine(int age, string make, string model)
  {
    this.Age = age;
    this.Make = make;
    this.Model = model;
  }
}
```

Так само як і в структурах можливо написати властивості лише для зчитування, лише для запису та автоматично згенерованівластивості.

6.1.5 Використанняконструкторів

Коливипоглянетенарозділствореннякласу,випомітитещоініціалізаціявідбуваласязадопомогоюцьогокоду:

```
DrinksMachine dm = new DrinksMachine();
```

Синтаксиссхожийнавикликметоду.Цетому,щоколивиініціалізуєтеклас,вивикликаєтеспеціальнийметод,якийназиваєтьсяконструктором.**Конструктор**–методкласу,якиймаєтесамеім’я,щоісамклас.Конструкторнеповертаєзначення(навітьnevoid).

Конструкторчастовикористовуєтьсядляприсвоєнняспеціальнихчистандартнихзначеньчленамновогооб’єкту,приклад:

```
public class DrinksMachine
{
  public int Age { get; set; }

publicDrinksMachine()
  {
Age = 0;
  }
}
```

Конструкторбезпараметрів називаєтьсяконструкторомзазамовчуванням. Цейконструкторвикликаєтьсяколивідбуваєтьсяініціалізаціякласубезаргументів.Якщовинереалізуєтеконструкторувашомукласі,C#компіляторавтоматичностворитьпорожнійpublicконструктордлявашогоскомпільованогокласу.

Вбагатьохвипадкахбуваєкориснодлякористувачавашогокласуматиможливістьвстановлюватиспеціальнізначеннячленівкласу,коливініціалізується.Дляприкладу,хтосьхочестворитиновийоб’єктклasуDrinksMachineзадавши одночасно виробника та модель автомата.Вашкласможемістититидекількаконструкторівзрізнимикомбінаціямипараметрівдляініціалізації.Данийспосібназиваєтьсяперевантаженням.

Приклад декількох конструкторів класу:

```
publicclassDrinksMachine
{
public int Age { get; set; }
```

```
  public string Make { get; set; }
  public string Model { get; set; }

  public DrinksMachine(int age)
  {
    this.Age = age;
  }

  public DrinksMachine(string make, string model)
  {
    this.Make = make;
    this.Model = model;
  }

  public DrinksMachine(int age, string make, string model)
  {
    this.Age = age;
    this.Make = make;
    this.Model = model;
  }
}
```

Користувачівашогокласуможутьвикористатибудь-якийконструктордлястворенняоб’єктувашогокласу,базуючисьнаданихякідоступніїмвцейчас.Дляприкладу:

```
var dm1 = new DrinksMachine(2);
var dm2 = new DrinksMachine("Fourth Coffee", "BeanCrusher 3000");
var dm3 = new DrinksMachine(3, "Fourth Coffee", "BeanToaster Turbo");
```

6.1.6 Створеннястатичнихкласівтавластивостей

Вдеякихвипадкахвампотрібностворитиклас,якийміститимедеякийкориснийфункціонал, анедлявідображенняякоїсьсутності.Дляприкладувихочетестворитинабірметодівдляконвертаціїваг в англійській(британська імперська) системі мірдоваги в метричній системітанавпаки.Немаєсенсустворюватиоб’єктпевногокласудляданихметодів,адженепотрібнозберігатичиотримуватипевніоб’єктноописуючідані.Тобтоприсутністьоб’єктунеселогічногосенсу.

Втакихвипадкахвиможетестворити**статичнийклас**.Статичнийклас–цекласякийнеможебутиініціалізований.Дляствореннястатичногокласутребавикористатиключовесловоstatic.Будь-якічленикласутакожповиннівикористовуватиключовесловоstatic.Приклад:

```
public static class Conversions
{
  public static double PoundsToKilos(double pounds)
  {
    // фунти в кілограми
    double kilos = pounds * 0.4536;
    return kilos;
  }

  public static double KilosToPounds(double kilos)
```

```
  {
    // кілограми в фунти
    double pounds = kilos * 2.205;
return pounds;
  }
}
```

Длявикликуметодустатичногокласу,вивикликаєтеметодвідіменікласузамістьіменіоб’єктукласу.Приклад:

```
double weightInKilos = 80;
double weightInPounds = Conversions.KilosToPounds(weightInKilos);
```

Статичнічленикласу

Нестатичнікласиможутьмістити**статичнічлени**.Цекорисноколипевнаповедінкачивластивістьзалежитьвідоб’єкту, апевнавідсамогокласу. Методи,поля,властивостітаподіїможутьбутиоголошенístатичними.Статичнівластивостічастовикористовуютьсядляповерненнязначеньвластивихвсімоб’єктамкласучивідслідковуванняскількиоб’єктівкласубулостворено.Статичніметодичастовикористовуютьсявметодах,якіпов’язанізкласом,наприклад, функціяпорівняння.

Дляоголошенняstatичногочленакласувикористовуєтьсяключовесловоstatic:

```
public class DrinksMachine
{
  public int Age { get; set; }
  public string Make { get; set; }
  public string Model { get; set; }

  public static int CountDrinksMachines()
  {
    // логіка методу тут
}
}
```

Незалежновідкількостіоб’єктіввашогокласу,існуєтількиодинекземплярстатичногочлена.Непотрібноініціалізовуватиоб'єктдлявикористаннястатичногометоду.Можнаотриматидоступвикористовуючиім’якласузамістьіменізмінної. Приклад:

```
intdrinksMachineCount= DrinksMachine.CountDrinksMachines();
```

6.1.7 Анонімні класи

Як вже, мабуть, ви здогадались **анонімні класи** – це класи, що не мають явно вказаної назви. Анонімні класи дають змогу створити об’єкт без потреби спочатку визначити тип. Назву цього класу буде автоматично згенеровано компілятором.

Ось як можна створити об’єкт анонімного класу:

```
var anAnonymousObject = new { Name = "Tom", Age = 65 };
```

Цей клас буде мати два публічних поля Name (зі значенням “Tom”) та Age(зі значенням 65).

Після створення ви можете працювати з цим об’єктом так само, як з об’єктом не анонімного класу:

```
Console.WriteLine("Name: {0} Age: {1}", anAnonymousObject.Name, anAnonymousObject.Age;
```

Після створення анонімного класу ви можете використовувати його, щоб створювати інші об’єкти цього класу:

```
var secondAnonymousObject = new { Name = "Hal", Age = 46 };
```

(Компілятор C# дивиться на назви, тип, кількість та порядок полів в оголошенні анонімного класу, щоб зрозуміти чи ці класи є однаковими.)

Зауваження: Існує декілька обмежень при роботі з анонімними класами:

- Анонімні класи можуть мати лише публічні поля.
- Всі поля повинні бути ініціалізованими(присвоєне значення).
- Поля не можуть бути статичними.
- Ви не можете використовувати оголошення методів всередині анонімних класів.

6.1.8Успадкування(наслідування). Класи, що не наслідуються (Sealed класи)

Успадкування(наслідування) є одним з основних концептів в об’єктно-орієнтованому програмуванні. Ви можете використовувати успадкування, щоб перевикористовувати код, що має спільні риси та має залежність один з одним. Наприклад менеджер, програміст, директор є працівниками компанії, а працівник в свою чергу є людиною.

Уявіть собі ситуацію, що вам необхідно написати програму, що симулює роботу всіх працівників. У кожного з них є спільні поля(ім’я, вік, стать, …) але в той же самий час в них є різні інші поля, які можуть бути лише у деяких типів працівників(у менеджера є список програмістів, чого нема у самих програмістів).

Наслідування дозволяє створити базовий клас, що містить основні параметри і кожен клас, що буде наслідувати базовий, буде мати(успадкує) ці параметри і, можливо, якісь інші. Клас, що наслідує базовий, зазвичай називають дочірнім(підкласом), а сам базовий – батьківським(надкласом

Наш приклад можна зобразити за допомогою наступної діаграми на рис.Е.1 (Додаток Е).

Застосування наслідування(Inheritance)

У C# не підтримується напряму множинне наслідування(успадкування). Множинне наслідування – це концепт,врезультатіякогокількабазовихкласівможутьбутиуспадкованіодни мпідкласом.

Щоб зробитиуспадкуваннявід базового класу в C#видодаєтепісляім'я похідногокласудвокрапкутаім'ябазовогокласу. Наступний приклад демонструє клас Manager, який наслідує клас Employee.

```
classManager : Employee
{
   private char payRateIndicator;
   private Employee[] emps;
}
```

Для простого визначення класу в C# використовується ключове словоclass,після якого ми пишемо ім'я класуManager, двокрапку і після цього ім'я базового класуEmployee. Дивлячись на код вище ми не можемо сказати, що класManagerуспадкував відEmployee, тому нам потрібно буде подивитись в цей клас(Employee) теж, щоб зрозуміти, які властивості для нас є доступні. КласEmployee:

```
class Employee
{
   private string empNumber;
   private string firstName;
   private string lastName;
   private string address;
   public string EmpNumber
   {
      get
      {
         return empNumber;
      }
      set
      {
         empNumber = value;
      }
   }
   public string FirstName
   {
      get
      {
         return firstName;
      }
      set
      {
         firstName = value;
      }
   }
   public string LastName
   {
      get
      {
         return lastName;
      }
      set
      {
         lastName = value;
      }
   }
   public string Address
   {
      get
      {
         return address;
      }
      set
      {
         address = value;
      }
   }
}
```

Intellisense може дати вам можливість візуального представлення успадкованих полів. Наприклад, коли ми задаємоекземпляртипуManagerв нашому коді і після цього ставимо крапку, ми побачимо список властивостей класуManager,а також властивості з базового класуEmployee., що демонструється на рис. Е.2 (Додаток Е).

Ми вже обговорили успадкування для класів і показали як базовий клас може бути успадкований підкласом і ми обговорили абстрактні класи. Обидва концепти фокусуються на можливості класу бути успадкованим і надавати атрибути та поведінку іншим класам для того, щоб ми могли не писати новий, а використовувати уже написаний код. Але що трапиться, якщо ви вирішите, що ви не хочете, щоб ваш клас не був успадкований? Ми це можемо зробити дуже просто за допомогою створення sealed класу.Ви можете використати ключове словосealedy вашому класі для того, щоб заборонити можливість успадковування для вашого класу. Якщо клас спробує зробити наслідування від sealed класу, то компілятор покаже помилку.

Оскільки ми необговорювалицю тему під час розгляду структур, важливо зазначити, що структури є **sealed**по замовченню.

6.1.9Абстрактні класи

Пригадаємопопередню тему – наслідування. Ми помітили, що класEmployeeє базовим класом дляManagertaProgrammer. Ми можемо продовжувати розширювати класEmployee,створюючистільки підкласів скільки вимагається для різних типів працівників у нашому додатку. Водночас зрозуміло, що немає сенсу мати можливість створювати об'єкт класу Employee напряму, адже абсурдно мати працівника без обов'язків.

Для того, щоб додати цю поведінку до нашого коду, нам потрібно задати класEmployeeяк абстрактний. **Абстрактний клас** частково відноситься до теми інтерфейсів, яку митакожрозглядаємо. Для абстрактного класу неможливо створити екземпляр, тобто ми не зможемо створити нового об'єкта Employee в коді, використовуючи цей запис:

EmployeenewEmployee = newEmployee();

Коли ви створюєте абстрактний клас, ви частково можете задати поведінку об'єкта в класі, або не задавати її взагалі. Абстрактний клас вимагає, щоб у підкласі ви здійснили імплементацію частини функціональності або описали всю функціональність. Якщо ми розширимо наш попередній приклад з класами Employee і Managerclasses, використовуючи абстрактні класи, ми зможемо продемонструвати цей концепт краще. Помітьте, що клас Employee зараз містить певні методи, для яких необхідно задати поведінку.

```
abstract class Employee
  {
    private string empNumber;
    private string firstName;
    private string lastName;
    private string address;

    .....

    public virtual void Login()
    {
    }

    public virtual void LogOff()
    {
    }

    public abstract void EatLunch();
  }
```

В цьому прикладі ..... означають, що певна частина коду була випущена задля скорочення.

Також помітьте, що ми додали ключове слово abstract до нашого класу:abstractclassEmployee. Саме ця дія робить наш клас абстрактним і встановлює певні вимоги. Як тільки ви створили абстрактний клас, ви можете вирішити і задати методи, які необхідно буде задавати у підкласах і які методи можна буде задавати чи переписати в підкласах.

Будь-який метод, який ви задекларували в абстрактному класі і що буде містити певну імплементацію в абстрактному класі, може бути переписаний в підкласі, але для цього до методу вам потрібно буде додати ключове слово virtual.

Помітьте, що у попередньому прикладі кодуLogin()таLogOff()обидва зазначені зі словом virtual. Це означає, що ви можете задати імплементацію в абстрактному класі, але у під класі ви можете переписати реалізацію цього методу або, якщо вона вас влаштовує в абстрактному класі, залишити поточну реалізацію.

МетодEatLunch()заданий з ключовим словом abstract, як клас. Є певні обмеження щодо таких методів:

✓ Абстрактний метод не може існувати не в абстрактному класі.

✓ Абстрактний метод не може мати жодної імплементації, включаючи пусті фігурні дужки.

✓ Сигнатура абстрактного методу повинна закінчуватись крапкою з комою.

✓ Абстрактний метод повинен бути реалізований в будь-якому підкласі, в інакшому разі буде згенеровано попередження компілятора вC#.

6.2 Завданнядолабораторної роботи(Додаток Е)

6.3 Контрольні запитання:

1. Що таке клас та об’єкт класу?
2. Яка відмінність між описом класу та об’єктом класу?
3. Що оголошується в класі?
4. Що називається полями класу?
5. Які елементи мови C# можуть відноситись до функцій-членів класу?
6. Яка загальна форма опису класу?
7. Які є типи (модифікатори) доступу до членів класу?
8. Які можливості доступу дає використання ідентифікатору protected internal?
9. Чи може бути відсутній тип (модифікатор) доступу при описі членів класу?
10. Приклади опису найпростіших класів, що містять тільки дані.
11. Як створити об’єкт (екземпляр класу)?
12. До яких типів даних належать класи: до типів значення чи до посилальних типів?
13. Що таке конструктори? Які відмінності між методами класу і конструкторами?
14. Яка загальна форма конструктора?
15. Скільки конструкторів може мати клас?
16. Що таке конструктор за замовчуванням?
17. Приклад ініціалізації класу з допомогою конструкторів.
18. Що означає “збір сміття” в C#?
19. Для чого призначені деструктори?
20. Які особливості використання ключового слова this в класі?

ЛАБОРАТОРНАЯ РАБОТА № 7
ІНТЕРФЕЙСИ. КОЛЕКЦІЇ

Мета: навчитися створювати та реалізовувати інтерфейси

7.1 Теоретичні відомості

7.1.1 Створення інтерфейсів

Інтерфейси схожі на класи, але без реалізації. Вони визначають множину характеристик і поведінки, задаючи сигнатури для методів, властивостей, подій та перелічення без їх імплементації. Якщо клас імплементує інтерфейс, тоді він має задати реалізацію для кожного члена інтерфейсу. Імплементуючи інтерфейс, клас має гарантовано повністю надати функціональність задану в інтерфейсі.

Помітьте важливу різницю у використанні Інтерфейсів. Клас**імплементує**інтерфейс на відміну від**успадковує**базовий клас.

Ви можете уявити інтерфейс як контракт. Імплементуючи певний інтерфейс, клас гарантує, що у ньому буде функціональність задана в контракті.

Синтаксис задання інтерфейсу схожий на синтаксис визначення класу. Ви використовуєте ключове слово interface для визначення інтерфейсу як це показано в прикладі нижче:

```
// Визначення інтерфейсу
public interface IBeverage
{
  // Methods, properties, events, and indexers go here.
}
```

Зауваження: У конвенції написання коду вказано, що імена інтерфейсів мають починатись з "I" хоча це і не обов'язково.

Визначення інтерфейсів можуть містити модифікатор доступу, що схоже на задання класів. Ви можете використовувати модифікатори доступу під час декларації вашого інтерфейсу, подані у таблиці Є.1 (Додаток Є):

Додання членів інтерфейсу

Інтерфейс визначає сигнатуру членів, але не задає імплементацію. Інтерфейс може включати методи, властивості, події та індексатори:

- Для визначення методу ви задаєте ім'я методу, тип що повертається і параметри:

```
int GetServingTemperature(bool includesMilk);
```

- Для визначення властивості, ви зазначаєте ім'я властивості, тип властивості, get та set:

```
bool IsFairTrade { get; set; }
```

- Для визначення події, ви використовуєте ключове слово event, що супроводжується делегатом, що здійснює обробку події та ім'ям події:

event EventHandler OnSoldOut;

- Для визначення індексатора, ви зазначаєте тип, що повертається та get і set:

string this[int index] { get; set; }

Члени інтерфейсу не включають модифікатори доступу. Усі члени є public. Інтерфейси не можуть включати членів, що відносяться лише до внутрішньої функціональності класу, такі як поля, константи, оператори або конструктори.

Давайте поглянемо на конкретний приклад. Припустимо, що ви хочете розробити програму лояльності для роботи програми компанії, що займається кавою. Ви можете створити інтерфейс, що називається ILoyaltyCardHolder та визначає:

✓ Властивість лише для читання з назвою TotalPoints.

✓ Метод з назвою AddPoints, що працює з десятковими значеннями.

✓ Метод, що називається ResetPoints.

Наступний приклад демонструє інтерфейс, що визначає одну властивість, що лише зчитується, та два методи:

```
// визначення інтерфейсу
public interface ILoyaltyCardHolder
{
  int TotalPoints { get; }
  int AddPoints(decimal transactionValue);
  void ResetPoints();
}
```

Помітьте, щометодивінтерфейсіневключаютьтілометоду. Так само, у властивостях в інтерфейсі визначається чи можуть вони зчитуватись і чи можуть вони задаватись (get; set;), але не зазначено жодної реалізації. Інтерфейс просто заявляє, що будь-який клас, що його реалізує, повинен включати в себе і забезпечити реалізацію для трьох членів. Розробник класу може вибрати як реалізуються методи. Наприклад, імплементація методу AddPoints буде приймати десяткові значення аргументів (сума готівки при здійсненні транзакції) і повертатиме ціле значення (число бонусів, що додається). Розробник класу може реалізувати ці методи багатьма способами. Наприклад, реалізація методу AddPoints може:

✓ Обраховувати кількість бонусів, здійснюючи множення суми транзакції на фіксоване число.

✓ Отримати число бонусів, викликавши інший сервіс.

✓ Отримати число бонусів, додавши до обчислення додаткові параметри, такі як геолокація та ін.

Наступний приклад демонструє клас, що реалізовує інтерфейс ILoyaltyCardHolder:

```
// Реалізація інтерфейсу
public class Customer : ILoyaltyCardHolder
{
  private int totalPoints;
  public int TotalPoints
  {
    get { return totalPoints; }
  }
  public int AddPoints(decimal transactionValue)
  {
    int points = Decimal.ToInt32(transactionValue);
    totalPoints += points;
```

```
  }
  public void ResetPoints()
  {
    totalPoints = 0;
  }
  // Інші члени класу Customer.
}
```

Здійснюючи імплементацію інтерфейсу ILoyaltyCardHolder, клас вказує користувачу, що у ньому здійснена реалізація операції AddPoints. Однією з ключових переваг інтерфейсу є те, що вони можуть поділити код на модулі. Ви можете змінити шлях реалізації інтерфейсу в класі в будь-який час без втручання в користувацькі класи, що оперують визначення інтерфейсів, а не їх реалізаціями.

Явна та неявна реалізації

Коли ви створюєте клас, що імплементує інтерфейс, ви обираєте чи явно чи не явно його задати. Для задання інтерфейсу неявно, ви реалізуєте кожен член інтерфейсу з сигнатурою, що відповідає визначенню в інтерфейсі. Для реалізації інтерфейсу в явному вигляді, ви можете вказати кожне ім'я члена так, щоб було ясно, що елемент належить до певного інтерфейсу.

Наступний приклад показує явну реалізацію інтерфейсу IBeverage:

```
// РеалізаціяінтерфейсуявноpublicclassCoffee : IBeverage
{
  private int servingTempWithoutMilk { get; set; }
  private int servingTempWithMilk { get; set; }
  public int IBeverage.GetServingTemperature(bool includesMilk)
  {
    if(includesMilk)
    {
       return servingTempWithMilk;
    }
    else
    {
      return servingTempWithoutMilk;
    }
  }
  public bool IBeverage.IsFairTrade { get; set; }
  // Інші члени, що не належать до інтерфейсу.
}
```

У більшості випадків реалізовувати інтерфейс явно чи не явно є лише естетичним вибором. Деякі розробники полюбляють явну реалізацію інтерфейсу, оскільки це робить код більш зрозумілим. Єдиним сценарієм при якому ви мусите використовувати явну реалізацію інтерфейсу–це якщо ви використовуєте два інтерфейси, що використовують ті ж самі ім'я. Наприклад, якщо ви використовуєте інтерфейс з назвоюIBeverageтаIInventoryItem, і обидва вони декларують властивість Boolean з назвоюIsAvailable, вам потрібно буде задати хоча б одну з цих властивостей явноIsAvailable. У цьому сценарії компілятор не зможе вирішити цю проблему без явної реалізації.

7.1.2 Поліморфізм інтерфейсів

Поліморфізм–це концепт, у якому стверджується, коли ви можете представляти собою екземпляр класу як екземпляр будь-якого інтерфейсу, який реалізує клас. Поліморфізм інтерфейсів може допомогти збільшити гнучкість і модульність коду. Припустимо у вас є декілька класів, що реалізовують інтерфейс IBeverage, такі як Coffee, Tea, Juice і т.д. Ви можете писати код, що працює із будь-яким з цих класів, як з екземпляром IBeverage без знання будь-яких деталей реалізації класу. Наприклад, ви можете побудувати колекцію екзекмплярів IBeverage без потреби отримання інформації про кожен клас що імплементує IBeverage.

Наприклад, якщо клас Coffee реалізує IBeverage інтерфейс, ви можете представити новий об'єкт Coffee як екземпляр Coffee або екземпляр IBeverage:

```
// представлення об'єкту за типом інтерфейсу
Coffeecoffee1 = newCoffee();
IBeveragecoffee2 = newCoffee();
```

Ви можете використовувати неявне приведення для перетворення до типу інтерфейсу, тому що ви знаєте, що клас повинен включати в себе всі елементи інтерфейсу.

```
// Приведення до типу інтерфейсу
IBeveragebeverage = coffee1;
```

Ви повинні використовувати явне приведення для перетворення з типу інтерфейсу для похідного типу класу, оскільки клас може включати в себе елементи, які не визначені в інтерфейсі.

```
// Приведення об'єкту типу інтерфейсу до похідного класу
Coffee coffee3 = beverage as Coffee;
// або
Coffee coffee4 = (Coffee)beverage;
```

Реалізаціябагатьохінтерфейсів

Убагатьохвипадках, вибудетехотітистворитикласи, щореалізовуютьбільше, ніждинінтерфейс.

Наприклад, ви можете хотіти:

✓ Реалізувати IDisposable інтерфейс для того, щоб .NETruntime розпорядився вашим класом вірно.

✓ Реалізувати IComparable інтерфейс, щоб дозволити колекціям класів сортувати екземпляри класів.

✓ Реалізувати ваш власний інтерфейс для визначення функціональності вашого класу.

Для реалізації декількох інтерфейсів, ви повинні додати список інтерфейсів розділений комою під час оголошення вашого класу. Ваш клас має реалізувати кожен член кожного інтерфейсу, що ви додали до декларації вашого класу.

Наступний приклад демонструє, як створити клас, що реалізовує декілька інтерфейсів:

```
// Оголошення класу, що використовує декілька інтерфейсів
public class Coffee: IBeverage, IInventoryItem
{
}
```

7.1.3 Колекції

Коливистворюєтебагатоелементіводногоітогожтипу, незважаючинатечицецілічислачицерядкичицеоб'єктикласу, наприклад**Coffee**, вамзнадобитьсяшляхдляобробкицихелементівнібицемасив. Ви, можливо захочете рахувати кількість елементів в масиві, додавати елементи або видаляти елементи з масиву або переглядати один за одним ці елементи. Для цих цілей ви можете використовувати колекції.

Колекції–це важливий інструмент для обробки багатьох елементів. Вони також потрібні і для розробки графічних додатків. Такі елементи як випадаючий список чи меню–це як правило прив'язка до даних в колекції.

Обираємо колекції

Всі колекції мають спільні властивості. Для обробки колекції елементів ви повинні мати можливість:

- ✓ Додавати елементи до колекції.
- ✓ Видаляти елементи з колекції.
- ✓ Отримувати конкретні елементи з колекції.
- ✓ Підрахувати кількість елементів в колекції.
- ✓ Перебрати елементи в колекції один за одним.

Кожен клас колекції в C # надає методи і властивості, які допомагають здійснити ці основні операції. Але крім цих операцій ви захочете керуватиколекціями по-різному в залежності від конкретних вимог вашої програми. Колекції поділяються на категорії:

- **Клас List**–клас списку як одновимірний масив, який динамічно розширюється в міру додавання елементів. Наприклад, використовувати клас список можна для підтримки списку доступних напоїв у вашому кафе.
- **Клас Dictionary**(Словник) зберігає колекцію пар ключ/значення. Кожен елемент в колекції складається з двох об'єктів–ключа і значення. Значення є об'єкт, який ви хочете зберігати та переглядати, а ключ є об'єктом, який ви використовуєте для індексування та пошуку значення.У більшості класів словник, ключ повинен бути унікальним, в той час як повторювані значення цілком прийнятні. Наприклад, ви могли б використовувати клас словників для зберігання списку рецептів кави. Ключ буде містити унікальне ім'я кави, а значення буде містити інгредієнти та інструкції для приготування кави.
- **Класи Queue**(Черга) являє собою колекції об'єктів, що умовно називається "перший прийшов–перший пішов", як і відбувається зазвичай (якщо відкинути випадки "я тільки запитати") у звичайних чергах. Ми отримуємо елементи з колекції в тому ж порядку, в якому вони були додані. Наприклад, ви можете використовувати клас чергу для обробки замовлень в кафе, щоб гарантувати, що клієнти отримують свої напої в порядку черги.

- **Класи Stack** (Стек) представляє собою колекцію об'єктів, що умовно називається "останній прийшов–перший пішов". Елемент, який ви додали в колекцію останнім є першим елементом, який ви будете переглядати. Наприклад, ви можете використовувати клас стек, щоб визначити 10 останніх відвідувачів вашого кафе.

Коли ви вибираєте вбудований клас колекцій для певного сценарію, задайте собі наступні запитання:

✓ Вам потрібен список, словник, стек або черга?

✓ Чи буде вам потрібно впорядкувати колекцію?

✓ Наскільки велику колекцію ви очікуєте отримати?

✓ Якщо ви використовуєте клас словник, чи ви повинні будете отримувати елементи тільки за індексом, чи також за ключем?

✓ Чи складається ваша колекція виключно з рядків?

Якщо ви зможете відповісти на всі ці питання, ви будете мати можливість вибрати у C# клас колекції, який найкращим чином відповідає вашим потребам.

7.1.4 Стандартні класи колекцій

Простір імен **System.Collections** надає ряд колекцій загального призначення, який включає в себе списки, словники, черги і стеки. У таблиці Є.2 (Додаток Є) наведено найбільш важливі класи колекцій в просторі імен System.Collections.

7.1.5 Спеціалізовані класи колекцій

Простір імен **System.Collections.Specialized** надає класи колекцій, що задовольняють особливі вимоги, такі як спеціалізовані колекції словники і строго типізовані колекції рядків. Таблиця Є.3 (Додаток Є) демонструє найбільш важливі класи колекцій в просторі імен System.Collections.Specialized.

7.1.6 Використання колекцій

Найбільш часто використовувана колекція–це клас ArrayList. ArrayList зберігає елементи як лінійну колекцію елементів. Ви можете додати об'єкти будь-якого типу до колекції ArrayList, але ArrayList представляє кожен елемент в колекції як екземпляр System.Object. Коли ви додасте елемент до колекції ArrayList, ArrayList неявно приведе до типу чи конвертує ваші елементи до типу Object. Коли ви проходите по елементах колекції, ви маєте явно перетворювати об'єкти назад в оригінальний тип.

Наступний приклад демонструє як додавати і проходитись по колекції ArrayList:

Додавання і проходження по елементах ArrayList

```
// СтворенняArrayListколекції.
ArrayListbeverages = newArrayList();

// Створенняелементів, якімихочемододатидоколекції.
Coffee coffee1 = new Coffee(4, "Arabica", "Columbia");
Coffee coffee2 = new Coffee(3, "Arabica", "Vietnam");
Coffee coffee3 = new Coffee(4, "Robusta", "Indonesia");
```

```
// Додавання елементів до колекції.
// Елементи неявно приводяться до типу Object, коли ви їх додаєте.
beverages.Add(coffee1);
beverages.Add(coffee2);
beverages.Add(coffee3);

// Проходження по елементах у колекції.
// Елементи повинні бути явно перетворені назад до початкового типу.
Coffee firstCoffee = (Coffee)beverages[0];
Coffee secondCoffee = (Coffee)beverages[1];
```

Коли ви працюєте з колекціями, одним з найбільш звичних задач є ітерування по колекції. В основному це означає, що ви переглядаєте кожен елемент колекції по черзі, як правило, щоб оцінити кожен елемент по відношенню до деяких критеріїв або для знаходження певних значень для кожного елементу. Для того, щоб перебрати колекцію, ви можете використовувати цикл**foreach**. Цикл**foreach**відображає кожен елемент з колекції по черзі, використовуючи ім'я змінної, що ви задали під час декларації циклу.

Наступний приклад демонструє, як проходити по колекції ArrayList:

```
// Проходженняпосписку
foreach(Coffee coffee in beverages)
{
  Console.WriteLine("Bean type: {0}", coffee.Bean);
  Console.WriteLine("Country of origin: {0}", coffee.CountryOfOrigin);
  Console.WriteLine("Strength (1-5): {0}", coffee.Strength);
}
```

Використання колекції словник

Клас словник зберігає колекцію пар ключ/значення. Найбільш поширений клас словників–**Hashtable**. Коли ви додаєте елемент до колекції Hashtable, ви маєте визначити ключ та значення. Ключ та значення можуть бути екземплярами будь-якого класу, але Hashtable неявно переведе ключ і значення до типу об'єкт. Коли ви проходитесь по значеннях колекції, ви маєте явно перевести об'єкт в його оригінальний тип.

Наступний приклад демонструє як проходити по елементах з колекції Hashtable. У цьому випадку ключ і значення це рядки:

Додавання і проходження по елементах Hashtable

```
// СтворенняновоїHashtableколекції.
Hashtableingredients = newHashtable();

// Додавання пар ключ/значення до колекції.
ingredients.Add("Café au Lait", "Coffee, Milk");
ingredients.Add("Café Mocha", "Coffee, Milk, Chocolate");
ingredients.Add("Cappuccino", "Coffee, Milk, Foam");
ingredients.Add("Irish Coffee", "Coffee, Whiskey, Cream, Sugar");
ingredients.Add("Macchiato", "Coffee, Milk, Foam");

// Перевірка чи такий ключ існує.
```

```
if(ingredients.ContainsKey("Café Mocha"))
{
  // Отримання значення, що асоціюється зі значенням ключа.
  Console.WriteLine("The ingredients of a Café Mocha are: {0}", ingredients["Café Mocha"]);
}
```

Класи словників, такі як Hashtable, взагалі-то містять дві перечислюванні колекції–ключ і значення. Ви можете проходитись по кожній з цих колекцій. У більшості випадків однак ви будете здійснювати ітерацію по колекції ключів, наприклад для отримання значення, що асоціюється з кожним із ключів.

У наступному прикладі демонструється як проходити по ключах в колекції Hashtable і отримувати значення, що асоціюється з кожним ключем:

```
// Ітеруванняпоколекції
foreach(string key in ingredients.Keys)
{
  // Для кожного ключа по черзі знаходиться значення, що з ним асоціюється.
  Console.WriteLine("The ingredients of a {0} are {1}", key, ingredients[key]);
}
```

Запити по колекції за допомогою предикатів лямбда-виразів

Деякі колекції в .NETFramework не підтримують позначень масиву для доступу до елементу в колекції. Ці колекції надають метод Find для визначення місця розташування елементів в колекції. Метод Find вимагає задання предиката, який буде використаний як критерій пошуку. В цьому випадку предикат стає методом, який буде перевіряти кожен елемент в колекції, повертаючи логічне значення Booleanна основі результатів співпадінь. Пошук закінчується, як тільки елемент знайдений.

Предикати, як правило, записуються у вигляді лямбда-виразу. За аналогією з методами, з якими ви уже знайомі, лямбда-вираз містить список параметрів і тіло методу, але він не містить ім'я методу і не містить тип значення, що повертається. Тип, що повертається, отримується з контексту, в якому використовується **лямбда-вирази**.

Наступний приклад лямбда-виразу для запиту по колекції об'єктів Employee(колекція також продемонстрована).

```
List<Employee> employees= new List<Employee>()
{
new Employee() { empID = 001, Name = "Tom", Department= "Sales"},
new Employee() { empID = 024, Name = "Joan", Department= "HR"},
new Employee() { empID = 023, Name = "Fred", Department= "Accounting" },
new Employee() { empID = 040, Name = "Mike", Department= "Sales" },
};

// Знайти елемент списку, де employeeid дорівнює 023
Employee match = employees.Find((Employee p) => { return p.empID == 023; });
Console.WriteLine("empID: {0}\nName: {1}\nDepartment: {2}", match.empID, match.Name, match.Department);
```

Результат згенерований кодом:

```
empID: 023
Name: Fred
```

Department: Accounting

Лямбда-вираз у коді вище наступний(Employee p) => { return p.empID == 023; }.

7.1.7 Узагальнення (Generics)

Утеміпроколекції, випобачили, щопривикористанняколекціїArrayListдозволенозберігатирізнітипиданих. Це відрізняється від масиву, де типи даних в масиві повинні бути одного і того ж типу. Але у ArrayList також існує проблема. Ми обговорювали раніше, що усе, що ви зберігаєте в ArrayList автоматично переводиться в тип Object, оскільки він є кореневим в .NET. Пригадаємо, що поліморфізм дозволяє базовому класу представляти підклас. .NET використовує Object як базовий клас для усіх інших типів, що створюються в C#.

Той факт, що ArrayList зберігає усі елементи як Object також означає, що під час проходження вам потрібно приводити усі об'єкти назад в той тип, у якому вони були спочатку. Це може бути проблемою або навіть може викликати помилку. Для вирішення цієї помилки ви можете використовувати **Узагальнення(Generics)**.

Узагальнення дозволяють вам створювати та використовувати строго типізовані колекції і при цьому не потрібно здійснювати приведення типів та здійснювати пакування та розпакування (box, unbox) типів значень.

Створення та використання класу Узагальнення

Класи Узагальнення працюють з параметром типу T, в класі чи при заданні інтерфейсу. Вам не потрібно задавати тип T до тих пір поки ви не створите екземпляр класу. Для створення класу Узагальнення вам потрібно:

✓ Додати параметр типу T в трикутних дужках після ім'я класу.

✓ Використовуйте параметр типу T замість назви типу членів вашого класу.

Наступний приклад демонструє як створювати клас узагальнення:

```
// Створення класу Узагальнення
publicclassCustomList<T>
{
  public T this[int index] { get; set; }
  public void Add(T item)
  {
    // Логікаметоду.
  }
  public void Remove(T item)
  {
    // Логіка методу.
  }
}
```

КоливистворюєтеекземплярвашогокласуУзагальнення, визадаєтетип, щовихочетевикористовуватиякпараметртипу. Наприклад, якщо ви хочете використовувати вами заданий список для зберігання об'єктів типу Coffee, ви будете задавати Coffee як параметр типу.

Наступний приклад показує як створити екземпляр класу узагальнення:

```
//створення екземпляру класу узагальнення
CustomList<Coffee>clc = newCustomList<Coffee>;
Coffee coffee1 = new Coffee();
Coffee coffee2 = new Coffee();
clc.Add(coffee1);
clc.Add(coffee2);
CoffeefirstCoffee = clc[0];
```

Коливистворюєтеекземпляркласу, токоженоб'єкт T будезаміненийнаоб'єкттипу, щовизадали. Наприклад, Якщовизадаєте CustomList класзпараметрамитипу Coffee:

- ✓ Метод Add буде приймати лише аргументи типу Coffee.
- ✓ Метод Remove буде приймати лише аргументи типу Coffee.
- ✓ Індексатор повертатиме значення типу Coffee.

Переваги Generics

Використання класів узагальнення, особливо для колекцій, має три великі переваги порівняно з не узагальненими підходами: безпека типів, немає приведення типів, немає пакування та розпакування (boxing, unboxing).

7.1.8 Безпека типів

Розглянемо приклад, де ви використовуєте ArrayList для зберігання колекції об'єктів типу Coffee. Ви можете додати об'єкти будь-якого типу до ArrayList. Уявимо, що розробник додав об'єкт типу Tea до колекції. Код працюватиме без посилок, однак виникне помилка під час виконання при виклику методу Sort, оскільки колекції не можуть порівнювати об'єкти різних типів(у загальному випадку). Більш того, коли ви переглядаєте об'єкт колекції, ви маєте приводити об'єкт до коректного типу. Якщо ви намагатиметесь привести об'єкт до не правильного типу, то виникне помилка під час виконання.

Наступний приклад демонструє обмеження, що стосуються обмежень безпеки типів при використанні підходу ArrayList:

```
// Обмеження безпеки типів при використанні не узагальнених колекцій
var coffee1 = new Coffee();
var coffee2 = new Coffee();
var tea1 = new Tea();
var arrayList1 = new ArrayList();
arrayList1.Add(coffee1);
arrayList1.Add(coffee2);
arrayList1.Add(tea1);
// Метод Sort видаєпомилкучасувиконання, томущоколекціянеєоднорідною.
arrayList1.Sort();
//                                    Приведеннятипіввидаєпомилкучасувиконання,
```

бовинеможетепривестиекземпляртипу Tea до Coffee.

Coffeecoffee3 = (Coffee)arrayList1[2];

АльтернативоюArrayListможебутиузагальненняList<T>длязбереженняколекціїоб'єктівCoffee. Коли ви створюєте список, ви надаєте елемент типу Coffee. У цьому випадку ваш список гарантовано буде однорідним, тому що вам не буде дозволено під час запуску коду додати об'єкт іншого типу. Метод Sort буде працювати, оскільки колекція буде однорідна. В решті решт, індексатор поверне об'єкт типу Coffee, а не System.Object, отже немає ризиків отримання помилки приведення типів.

Наступний приклад показує альтернативне використання замість ArrayList узагальнення List<T>:

```
// Використання узагальнень для безпечного використання типів
var coffee1 = new Coffee();
var coffee2 = new Coffee();
var tea1 = new Tea();
var genericList1 = new List<Coffee>();
genericList1.Add(coffee1);
genericList1.Add(coffee2);
// Цей рядок викличе помилку під час побудови програми, оскільки аргумент не типу Coffee.
genericList1.Add(tea1);
// Метод Sort буде працювати, оскільки гарантовано, що колекція буде однорідна.
genericList1.Sort();
// Індексатор повертає об'єкт типу Coffee, тому немає потреби робити перетворення значення, що повертається.
Coffeecoffee3 = genericList[1];
```

Приведення типів

Приведення типів–це досить дороге задоволення в плані використання обчислювальних ресурсів. Коли ви додаєте елемент в ArrayList, ваші елементи неявно приводяться до типу System.Object. Коли ви проходитесь по елементах з ArrayList, ви маєте явно привести їх назад до оригінального типу. Використовуючи узагальнення для додавання і проходження по елементах без приведення типів, ви можете покращити продуктивність вашого додатку.

Пакування та розпакування

Якщо ви хочете зберігати значення(наприклад int,float,double,...) в ArrayList, ви повинні здійснити **пакування (boxing)** елементів при додаванні до колекції і **розпакування (unboxing)**, коли ви їх хочете переглянути. Boxing та unboxing вимагає значних обчислювальних ресурсів і може значно уповільнити програму, особливо якщо ви проходитесь по великій колекції. На противагу, ви можете додати значення до узагальнення без пакування та розпакування.

Наступний приклад показує різницю між узагальненим і не узагальнемим підходом щодо використання колекцій:

```
// Boxing та Unboxing: Узагальнення на противагу не узагальненому підходу
int number1 = 1;
var arrayList1 = new ArrayList();
// Пакування Int32 як System.Object.
```

```
arrayList1.Add(number1);
// Розпакування Int32.
int number2 = (int)arrayList1[0];
var genericList1 = new List<Int32>();
// Додавання Int32 безпакування.
genericList1.Add(number1);
//Отримання Int32 безрозпакування.
int number3 = genericList1[0];
```

Обмеженнядляузагальнень

У деяких випадках, ви можете обмежити типи, що розробник може використовувати, коли він створює екземпляр узагальненого класу. Природа цих обмежень залежить від логіки, що ви плануєте реалізувати. Наприклад, якщо колекція класів використовує властивість з назвою AverageRating для сортування елементів в колекції, вам потрібне буде обмеження на тип параметру для класу, що включає властивість AverageRating. Уявимо, що AverageRating властивість визначенаIBeverage в інтерфейсі. Для реалізації ви будете обмежувати тип параметра для класів, що реалізують інтерфейс IBeverage, використовуючи ключове слово where.

Наступний приклад демонструє як задати обмеження на параметр типу для класів, які реалізують певний інтерфейс:

```
// Обмеження параметра типу
public class CustomList<T> where T : IBeverage
{
}
```

Ви можете застосувати шість типів обмежень для параметру типу, який наведений у таблиці Є. (Додаток Є).

Ви можете застосувати наступні шість типів обмежень для параметру типу:

```
// Застосування багатьох обмежень
public class CustomList<T> where T : IBeverage, IComparable<T>, new()
{
}
```

Використання узагальнення для колекцій

Одним з найбільш загальних і важливих застосувань узагальнень є створення колекцій. Узагальнення для колекції поділяються на дві широкі категорії: узагальнення списку і узагальнення словників. Узагальнення списку зберігає колекцію об'єктів типу T.

Клас List<T>

Клас List<T> надає строго типізовані альтернативи для класу ArrayList. Як клас ArrayList, клас List<T> включає методи для:

- ✓ Додавання елементу.
- ✓ Видалення елементу.
- ✓ Додавання елементу за певним індексом.
- ✓ Сортування елементів в колекції з використанням порівняння за замовчуванням або створення власного способу порівняння.
- ✓ Зміна порядку частини чи усієї колекції.

Наступний приклад демонструє як використовувати клас List<T>.

```
// ВикористанняклассуList<T>
string s1 = "Latte";
string s2 = "Espresso";
string s3 = "Americano";
string s4 = "Cappuccino";
strings5 = "Mocha";
// Додавання елементів до строго типізованої колекції.
var coffeeBeverages = new List<String>();
coffeeBeverages.Add(s1);
coffeeBeverages.Add(s2);
coffeeBeverages.Add(s3);
coffeeBeverages.Add(s4);
coffeeBeverages.Add(s5);
// Сортування елементів з використанням порівняння за замовчуванням.
// Для об'єкта типу String, порівняння за замовчуванням в алфавітному порядку.
coffeeBeverages.Sort();
// Колекція у консольному вікні.
foreach(String coffeeBeverage in coffeeBeverages)
{
  Console.WriteLine(coffeeBeverage);
}
```

Інші узагальнення для списків

Простір імен System.Collections.Generic також включає різні узагальнення колекцій, що надають більш спеціалізовану функціональність:

- Клас LinkedList<T> надає узагальнення, в якому кожен елемент пов'язаний з попереднім і наступним елементом колекції. Кожен елемент в колекції представлений об'єктом LinkedListNode<T>, що містить значення типу T, посилання на екземпляр батька LinkedList<T> , посилання на попередній елемент в колекції, і посилання на наступний елемент в колекції.
- Клас Queue<T> представляє строго типізовану колекцію типу "перший зайшов - перший вийшов".
- Клас Stack<T> представляє строго типізовану колекцію типу "останній зайшов - перший пішов".

Використання узагальнення словників

Клас словників зберігає колекцію пар ключ/значення. Значення - це об'єкт, що ви хочете зберігати, а ключ - це об'єкт, що ви використовуєте як індекс для отримання значення. Наприклад, ви можее використовувати клас словник для зберігання рецептів кави, де ключ - це назва кави і значення - це спосіб приготування. У випадку узагальнення словників і ключ і значення є строго типізованими.

Клас Dictionary<TKey, TValue>

Dictionary<TKey, TValue> - строго типізований клас словників загального призначення. Ви можете додати однакові значення, але ключі мають бути унікальні. Клас видасть помилку ArgumentException, якщо ви будете намагатись додати ключ, що уже існує в словнику.

Наступний клас демонструє як використовувати словник <TKey, TValue>:

```
// Використання словника <TKey, TValue>
// Створення нового словника рядків з ключами, що теж містять рядок.
```

```
var coffeeCodes = new Dictionary<String, String>();
// Додаванняелементіввсловник.
coffeeCodes.Add("CAL", "Café Au Lait");
coffeeCodes.Add("CSM", "Cinammon Spice Mocha");
coffeeCodes.Add("ER", "Espresso Romano");
coffeeCodes.Add("RM", "Raspberry Mocha");
coffeeCodes.Add("IC", "Iced Coffee");
// Наступнийрядоквикличепомилку ArgumentException, оскількиключужеіснує.
// coffeeCodes.Add("IC", "Instant Coffee");
// Для отримання значення, що асоціюється з ключем, ви можете використовувати індексатор.
// Виникне KeyNotFoundException помилка, якщо такий ключ не існує.
Console.WriteLine("The value associated with the key \"CAL\" is {0}", coffeeCodes["CAL"]);
// Як альтернатива ви можете використовувати метод TryGetValue.
// Буде повернено true, якщо ключ існує і false, якщо ключ не існує.
string csmValue = "";
if(coffeeCodes.TryGetValue("CSM", out csmValue))
{
  Console.WriteLine("The value associated with the key \"CSM\" is {0}", csmValue);
}
else

{
  Console.WriteLine("Thekey \"CSM\" wasnotfound");
}
// Ви також можете використовувати індексатор для зміни значення, що асоціюється з ключем.
coffeeCodes["IC"] = "InstantCoffee";
```

Іншікласисловників

SortedList<TKey, TValue>iSortedDictionary<TKey, TValue>класинадаютьузагальненнясловників, уякихелементисортованізаключем. Різницяміжцимикласамивреалізації:

- КласузагальненняSortedListвикористовуєменшепам'яті, ніжSortedDictionary.
- КласSortedDictionaryушвидшийібільшефективнийудодаванніівидалені несортованихданих.

Використанняколекційінтерфейсів

ПростіріменSystem.Collections.Generiснадаєрядзагальнихколекцій, щобзадовольнитипотребирізнихсценаріївикористання. Проте будуть обставини, при яких ви захочете створити свої власні класи узагальнень для того, щоб забезпечити більш спеціалізовану функціональність. Наприклад, ви зможете зберігати дані у вигляді дерева.

Що потрібно почати робити, якщо ви хочете створити окремий клас колекції? Всі колекції мають деякі спільні риси. Наприклад, ви захочете отримати можливість перераховувати елементи колекції за допомогою циклу foreach, і ви будете потребувати методи для додавання елементів, видалення елементів і очищення списку.

Інтерфейси IEnumerable та IEnumerable<T>

Якщо ви хочете мати змогу використовувати foreach для перерахування елементів у вашій власній колекції узагальнення, ви маєте реалізувати інтерфейс IEnumerable<T>. Інтерфейс IEnumerable<T> визначає єдиний метод з назвою GetEnumerator(). Цей метод має повертати об'єкт типу IEnumerator<T>. foreach опирається на об'єкти перерахування і здійснює перерахування по колекції.

Інтерфейс IEnumerable<T> наслідується від інтерфейсу IEnumerable, що також визначає єдиний метод GetEnumerator(). Коли інтерфейс успадковується від іншого інтерфейсу, він надає усі елементи батьківського інтерфейсу. Іншими словами, якщо ви реалізуєте IEnumerable<T>, вам також потрібно реалізувати IEnumerable.

Інтерфейс ICollection<T>

ICollection<T> інтерфейс визначає базову функціональність, що притаманна усім колекціям узагальнень. Інтерфейс успадковує від IEnumerable<T>, що означає, якщо ви хочете реалізувати ICollection<T>, ви повинні також реалізувати члени, що визначені IEnumerable<T> і IEnumerable.

Методи інтерфейсаICollection<T>наведені в таблиці Є.5 (Додаток Є).

Властивості інтерфейса ICollection<T> наведені в таблиці Є.6 (Додаток Є).

Інтерфейс IList<T>

Інтерфейс IList<T> визначає ключову функціональність для списків узагальнень. Ви повинні реалізувати цей інтерфейс, якщо ви визначаєте лінійну колекцію одиничних об'єктів. На додаток до членів, що успадковуються від ICollection<T>, IList<T> інтерфейс визаначає методи і властивості, що дозволяє використовувати індексатори для роботи з елементами в колекції. Наприклад, ящо ви створите список з назвою myList, ви можете використовувати myList[0] для досутуп до перших елементів в колекції.

Методи інтерфейсаIList<T>наведені в таблиці Є.7 (Додаток Є).

Властивості інтерфейса IList<T>наведені в таблиці Є.8 (Додаток Є).

Інтерфейс IDictionary<TKey, TValue>

Інтерфейс IDictionary<TKey, TValue> визначає ключову функціональність для узагальнених словників. Ви повинні реалізувати інтерфейс, якщо ви визначаєте колекцію пар значення/ ключ. На додаток до елементів, що наслідуються від ICollection<T>, IDictionary<T> інтерфейси визначають методи і властивості, що є специфічними для роботи з парами ключ-значення.

Методи інтерфейсаIDictionary<TKey, TValue>наведені в таблиці Є.9 (Додаток Є).

Властивості інтерфейса IDictionary<TKey, TValue>наведені в таблиці Є.10 (Додаток Є).

Створення перелічуваних колекцій

Для перелічення по колекції, ви зазвичай використовуєте цикл foreach. Цикл foreach проходиться по кожному елементу по черзі в порядку, що підходить до колекції. foreach маскує певні ускладнення перерахувань. Для роботи foreach клас колекцій узагальнення має реалізувати інтерфейс IEnumerable<T>. Цей інтерфейс містить метод GetEnumerator, що повертає тип IEnumerator<T>.

Інтерфейс IEnumerator<T>

Інтерфейс IEnumerator<T> визначає функціональність, що усі перерахування мають реалізуватись.

Методи інтерфейсаIEnumerator<T>наведені в таблиці Є.11 (Додаток Є).

Властивості інтерфейса IEnumerator<T>наведені в таблиці Є.12 (Додаток Є).

Завжди при перерахуванні є покажчик на елемент колекції. Стартова точка - це вказівник перед першим елементом. Коли ви викликаєте метод MoveNext, покажчик переміщується на наступний елемент в колекції. Метод MoveNext повертає істину (true), якщо можна було перевести покажчик на одну позицію вперед, або хиба (false), якщо було досягнуто кінця колекції. В кожний момент часу під час перерахування властивість Current повертає елемент, на який покажчик вказує.

Коли ви створили перерахування, вам потрібно визначити:

✓ Що є першим елементом перерахування в колекції.

✓ В якому порядку покажчик повинен рухатись по перерахуванню.

Інтерфейс IEnumerable<T>

Інтерфейс IEnumerable<T> визначає єдиний метод з назвою GetEnumerator. Він повертає екземпляр IEnumerator<T>.

МетодGetEnumeratorповертаєпокажчикдляперерахуваннядлявашогокласуколекції. Цей покажчик буде використовуватись для foreach циклу, якщо ви не зазначите альтернативу. Однак, ви можете створити додаткові методи для визначення альтернативного покажчика перерахування.

Наступний приклад демонструє колекцію, що реалізує покажчик перерахування за замовчуванням. А також альтернативний покажчик, що здійснює перерахунок колекції у зворотньому порядку:

```
// Визначення альтернативного методу перерахунку
classCustomCollection<T> : IEnumerable<T>
{
publicIEnumerator<T>Backwards()
  {
    // Цей метод повертає альтернативний перерахунок
    // Реалізаціянедемонструється
  }
#regionIEnumerable<T>Members
  publicIEnumerator<T>GetEnumerator()
  {
    // Цей метод поветрає перерахування за замовчуванням
    // Реалізаціянедемонструється
  }
  #endregion
  #region IEnumerable Members
```

```
    IEnumeratorIEnumerable.GetEnumerator()
    {
      // Цей метод вимагається, оскільки IEnumerable<T> успадковується від IEnumerable
       thrownewNotImplementedException();
    }
    #endregion
}
```

Наступнийприкладдемонструєякпокажчикзазамовчуваннямабоальтернативнийпокажчикздійснюєітераціюпоколекції:

```
// Проходженняпоколекції
CustomCollection<Int32> numbers = new CustomCollection<Int32>();
// Додавання елементів до колекцій.
// Використовується перерахування за замовчуванням по колекції:
foreach (int number in numbers)
{
  // …
}
// Використання альтернативного перерахування для ітерації по колекції:
foreach(intnumberinnumbers.Backwards())
{
  // …
}
```

Реалізація перерахування

Ви можете створити перерахування, створивши власний клас, що реалізує інтерфейс IEnumerator<T>. Однак, якщо ваш клас використовує за основу тип перерахування для зберігання даних, ви можете використовувати ітератор для реалізації інтерфейсу IEnumerable<T> без надання реалізації IEnumerator<T>. Найкращий спосіб зрозуміти ітератори - це написати простий приклад.

Наступний приклад показує як ви можете використовувати ітератори для реалізації перерахування:

```
// Використання перерахувань за допомогою ітераторів
usingSystem;
using System.Collections;
using System.Collections.Generic;
class BasicCollection<T> : IEnumerable<T>
{
   private List<T> data = new List<T>();
   public void FillList(params T [] items)
   {
      foreach (var datum in items)
        data.Add(datum);
   }
   IEnumerator<T> IEnumerable<T>.GetEnumerator()
   {
      foreach (var datum in data)
      {
         yield return datum;
      }
```

```
    }
    IEnumerator IEnumerable.GetEnumerator()
    {
       throw new NotImplementedException();
    }
}
```

НаступнийприкладдемонструєекземплярколекціїузагальненняList<T>длязберіганн яданих. ЕкземплярList<T>міститьметодFillList. КолиметодGetEnumeratorвикликається, циклforeachпроходитьсяпоколекції. Уциклiforeach, yieldповертаєкоженелементколекції. Цейyieldповертаєвираз, щовизначаєітератор—посуті, yieldпризупиняєвиконаннядоповерненняпоточногоелементаітількитодіпереходитьдонас тупногоелементавпослідовності. Такимчином, хочаметодGetEnumerator, неповертаєтипIEnumerator, компіляторможепобудуватиперелічення, виходячивідлогікиітерації, щовипередбачили.

7.2 Завдання до лабораторної роботи (Додаток Є)

7.3 Контрольні запитання:

1. Для чого використовуються інтерфейси?
2. Яка відмінність між інтерфейсами та абстрактними класами?
3. Скільки класів можуть мати реалізацію методів інтерфейсу?
4. Скільки інтерфейсів може бути реалізовано в одному класі?
5. Який загальний вигляд опису інтерфейсу?
6. Які елементи мови програмування можна вказувати в інтерфейсах?
7. Як виглядає загальна форма реалізації інтерфейсу в класі?
8. Яка загальна форма класу, що реалізує декілька інтерфейсів?
9. Яким чином в інтерфейсі описується властивість?
10. Які елементи програмування мови C# не можна описувати в інтерфейсах?
11. Що таке явна реалізація члену інтерфейсу?
12. В яких випадках краще використовувати інтерфейс, а в яких абстрактний клас?
13. Яка головна перевага колекцій?
14. Які є види колекцій?
15. Якими типами даних оперують неузагальнені колекції?
16. Якими типами даних оперують спеціальні колекції?
17. В якому просторі імен оголошуються спеціальні колекції?
18. Які стандартні структури даних реалізують узагальнені колекції?
19. Які особливості використання паралельних (багатопотокових) колекцій?
20. В якому просторі імен визначені паралельні (багатопотокові) колекції?

ЛАБОРАТОРНА РОБОТА № 8
ПОДІЇ. ДЕЛЕГАТИ

Мета роботи: навчитися викликати події та ознайомитись з делегатами.

8.1 Теоретичні відомості

8.1.1 Події та делегати

Події (events)це механізм, що дозволяє об'єктам повідомляти іншим об'єктам, коли щось відбувається. Наприклад,на веб сторінці генерується подія, коли користувач взаємодіє з елементом управління, натискаючи на кнопку. Ви можете створити код, що підписується на ці події і відповідає певними діями на них.

Без подій, вашій програмі знадобиться постійно переглядати значення в елементів управління і шукати зміни стану, що потребують дій у відповідь.

Це був би дуже не оптимальний спосіб розробки додатку. В цьому уроці ми вивчимо як створювати події і як на них підписуватись.

Делегати – це спеціальний тип, що визначає сигнатуру методів, іншими словами – тип даних, що повертається та параметри методу. Виходячи з назви, делегат поводить себе як представник для методу з відповідною сигнатурою.

8.1.2 Створення подій та делегатів

Коли ви створюєте подію в структурі чи класі, вам потрібен буде спосіб, що надасть можливість іншому коду підписатись на подію. УС#ви можете це зробити створивши делегат. Повторимо, делегат – це спеціальний тип, що визначає сигнатуру методу, іншими словами тип і параметри методу. З імені зрозуміло, що делегат себе поводить як представник методу з відповідною сигнатурою.

Коли ви визначили подію, ви робите асоціацію вашого делегата з подією. Для того, щоб підписатись на подію з клієнтської частини коду, вам потрібно:

✓Створити метод з сигнатурою, що відповідає делегату події. Цей метод відомий як обробник події.

✓Підписатись на подію, додавши ім’я обробника методу до публікації події, іншими словами, об'єкту, що буде формувуати подію.

✓Коли подія створена, делегат запускає всі методи обробки події, що підписані на цю подію.

Припустимо, що ви створили структуру з ім’ям Coffee. Однією з причин для створення такої структури є потреба у відслідковуванні рівню запасу кожного екземпляра кави (Coffee). Коли запас кави опускається нижче певної визначеної межі, ви хочете створити подію, що попередить вас у вашій системі замовлень про те, що у вас закінчуються кавові зерна.

Першим чином, у такому випадку вам потрібно визначити делегат. Для визначення делегата, ви використовуєте ключове слово – delegate.Делагат включає два параметри:

- ✓ Перший параметр – це параметр, що створює подію – у цьому випадку екземпляр Coffee.
- ✓ Другий параметр – цеаргументи події – іншими словами, будь-яка інша інформація, що ви хочете надати споживачу вашого програмного продукту. Це має бути екземпляр класу EventArgs,або екземпляр класу, що походить від EventArgs.

Далі вам потрібно оголосити подію.Для задання події, вам потрібно використати ключове словоevent.Перед заданням ім'я вашої події ви задаєте ім'я делегата, який ви хочете асоціювати з вашою подією.

Наступний приклад демонструє як задавати делегати і події:

```
//Задання Delegate і Event
publicstructCoffee
{
  publicEventArgse;
  publicdelegatevoidOutOfBeansHandler(Coffeecoffee, EventArgsargs);
  publiceventOutOfBeansHandlerOutOfBeans;
}
```

У цьому прикладі ви визначаєте подію з назвою OutOfBeans. Ви асоціюєте її з ім'ям делегатаOutOfBeansHandler. Делегат OutOfBeansHandler приймає два параметри, екземпляр Coffee, який є об'єктом, що створює подію, та екземплярEventArgs, що може бути використаний, щоб отримати більше інформацію про подію.

8.1.3 Виникнення та підписка подій

Після того, як ви визначили подію і делегат, ви можете писати код, що спричинює цю подію за певних обставин. Коли ви спричиняєте подію, делегат, що асоціюється з цією подією, викличе усі методи обробки події, які були підписані на цю подію.

Для того, щоб виникла подія, вам потрібно зробити наступні кроки:

1.Перевірити чи подія дорівнює null. Подія буде дорівнювати null, якщо ніякий код на цю подію не підписаний.

2.Викличіть подію і надайте аргументи делегату.

Наприклад, структура Coffeeвключає метод з назвоюMakeCoffee. Кожен раз, коли ви викликаєте методMakeCoffee, метод зменшує рівень запасу екземпляру Coffee.Якщо рівень запасу опускається певної межі, тодіMakeCoffeeметод спричинить виникнення подіїOutOfBeans.

Наступний приклад демонструє як спричиняти подію:

```
// Спричинення події
public struct Coffee
{
  // Задання події та делегата.
  publicEventArgse = null;
  public delegate void OutOfBeansHandler(Coffee coffee, EventArgs args);
  public event OutOfBeansHandler OutOfBeans;
  int currentStockLevel;
  int minimumStockLevel;
  public void MakeCoffee()
  {
```

```
        // Зменшення рівня запасу.
        currentStockLevel--;
        // Якщо рівень запасу опускається нижче мінімума, ми спричиняємо подію.
        if (currentStockLevel < minimumStockLevel)
        {
          // Перевірка події на null.
          if (OutOfBeans != null)
          {
            // Спричинення події.
            OutOfBeans(this, e);
          }
        }
      }
    }
```

Для того, щоб спричинити подію використовується схожий синтаксис як і для виклику метода. Ви надаєте аргументи, які співпадають з параметрами, що вимагаються делегатом. Перший аргумент – це об'єкт, що спричиняє подію. Помітьте, що ключове словоthisвикористовується для зазначення поточного екземпляру Coffee.Другий параметр це екземплярEventArgs, що може бутиnull, якщо вам не потрібно надавати жодної іншої інформації тим, хто на нього підписний.

Якщо ви хочете обробляти подію, вам потрібно зробити наступне:

✓ Створити метод з сигнатурою, що відповідає делегату події.

✓ Використати додатковий оператор(+=)для додання методу обробки події до вашої події.

Припустимо, що ви створили екземпляр структуриCoffeeз назвоюcoffee1. У вашому класі Inventoryви хочете підписатись на OutOfBeans, що може бути спричинена coffee1.

Зауваження:У попередніх розділах було показано як структураCoffee, подія OutOfBeansі делегатOutOfBeansHandlerзадаються.

Наступний приклад демонструє як підписатись на подію:

```
// Підписуємось на подію
public class Inventory
{
  public void HandleOutOfBeans(Coffee sender, EventArgs args)
  {
    stringcoffeeBean = sender.Bean;
    // Зміна порядку кави в зернах.
  }
  public void SubscribeToEvent()
  {
    coffee1.OutOfBeans += HandleOutOfBeans;
  }
}
```

В цьому прикладі, сигнатура HandleOutOfBeansметоду відповідає делегату для подіїOutOfBeans.Коли ви викликаєте методSubscribeToEvent,методHandleOutOfBeansдодається до списку підписників подіїOutOfBeansоб'єкту coffee1.

Для того, щоб відписатись від події, ви використовуєте оператор віднімання (-=)для видалення вашого методу обробки події від події.

Наступний приклад демонструє як відписатись від події:

```
//Відпистись від події
publicvoidUnsubscribeFromEvent()
{
  coffee1.OutOfBeans -= HandleOutOfBeans;
}
```

Життєвий цикл об'єкта

Життєвий цикл об'єкта має декілька стадій, що починаються зі створення об'єкта, а закінчуються його зруйнуванням. Для створення об'єкта у вашому додатку, ви використовуєте ключове словоnew. Коли загальномовне виконуюче середовище (CLR) виконує код для створення нового об'єкта, проходять наступні кроки:

1. Воно виділяє блок пам'яті, який достатньо великий для утримання об'єкта.
2. Воно ініціалізовує блок пам'яті новим об'єктом.

CLR займається виділенням пам'яті для всіх керованих об'єктів. Проте при використанні некерованих об'єктів, вам може бути потрібно написати код для виділення пам'яті для некерованих об'єктів, що ви створили. Некеровані об'єкти - це ті об'єкти, що не є .NET компонентами такими як MicrosoftWord об'єкт, підключення до бази даних, або файл ресурсів.

Коли ви завершили роботу з об'єктом, ви можете позбутись його для звільнення ресурсів таких як підключення до бази даних чи дескриптори файлів, що він споживає. При утилізації об'єкта, CLR використовується механізм, що надивається збирач сміття (garbagecollector - GC) для виконання наступних кроків:

1. GC вивільняє ресурси.
2. Пам'ять, що була виділена на об'єкт буде утилізована.

Збирач сміття (GC) - це окремий процес, що запускається у власному потоці щоразу, коли керований код додатка працює.

✓ GCдозволяє розробляти додатки без необхідності турбуватися про виділення пам'яті.

✓ GC ефективно виділяє об'єкти в керованій кучі.

✓ Об'єкти, які більше не використовуються, очищаються з пам'яті, і пам'ять стає доступна для майбутніх алокацій об'єктів.

Коли .NET програма виконана, GC ініціалізується CLR. GC виділяє сегмент пам'яті, який буде використовуватися для зберігання і управління об'єктами для кожної .NET програми, що запущена. Ця область пам'яті називається керована купа (managed heap) , яка відрізняється від нативної купи, використовуваної в контексті операційної системи.

Керована купа існує для кожного керованого процесу, який запущений, і кожен потік в процесі виділяє пам'ять для об'єктів в цьому процесі в тій же купі. Це означає, що кожен процес має свій власний віртуальний простір пам'яті. Це означає, що кожен процес має власну віртуальну пам'ять.

Зауваження: Об'єм сегмента, що алокується GC залежить від імплементації і може бути змінений у будь-який час під час оновлення. Коли ви пишете вашу програму, ви ніколи не повинні робити припущення про сегмент виділеної пам'яті, що буде використовуватисьGC.

GC виникає, якщо виконуються наступні умови:

- Не вистачає фізичної пам'яті.
- Пам'ять, яка на разі виділена під об'єкти, перевищує допустимий поріг. Поріг може змінюватись в процесі виконання програми.
- МетодGC.Collectвикликаний. Зазвичай, вамнепотрібновикликатицейметод, оскількиGCпрацюєпостійно. Навіть якщо ви викличете цей метод, це не гарантує того, що цей метод почне свою роботу тоді, коли ви його викличите.

8.1.4 Реалізація шаблону видалення

Шаблон видалення – це шаблон, що призначений для вивільнення ресурсів, що використовували об'єкти. .NETFramework надаєIDisposableінтерфейс в просторі імен System, що дозволяє реалізувати шаблон видалення у вашому додатку.

Інтерфейс IDisposable визначає єдиний метод без параметрів з назвою Dispose. Ви повинні використовувати метод Dispose для вивільнення усіх некерованих ресурсів, що використовують ваш об'єкт. Якщо клас наслідує інший клас, який вже перевизначив метод Dispose, то викликаючи батьківський метод Dispose, можна бути впевненим, що будь-які ресурси виділенні батьківською реалізацією класа теж будуть звільнені.

Виклик методу Dispose не руйнує об'єкт. Об'єкт залишається в пам'яті до того часу, поки останнє посилання на об'єкт не буде видалено і GC не вивільнить ресурси.

Багато класів в .NETFramework, що охоплюють некеровані ресурси(такі як StreamWriter) клас реалізовують IDisposable інтерфейс. Клас StreamWriter реалізує об'єкт TextWriter для запису текстової інформації в потік. Ви також повинні реалізувати інтерфейс IDisposable, коли ви створюєте свої власні класи, що посилаються на не керований тип.

Релізація інтерфейсу IDisposable

Для реалізації інтерфейсу IDisposable у вашому додатку, потрібно виконати наступні кроки:

1. Переконатись, що простір імен System в переліку, додавши наступний рядок на початку вашого файлу з кодом.

```
using System;
```

2. Здійснити реалізацію IDisposable інтерфейу при визначенні класу.

```
...
public class ManagedWord : IDisposable
{
  public void Dispose()
  {
    throw new NotImplementedException();
  }
}
```

3. Додати privateполе до класу(наприкладbool _isDisposed), яке ви зможете використовувати для отримання інформації про статус видалення об'єкту і перевірки того, чи метод Dispose уже був викликаний і ресурси вивільнились.

```
public class ManagedWord : IDisposable
{
  bool _isDisposed;
```

```
  ...
}
```

4. Додати до коду перевірки до ваших загальнодоступних методів перевірку чи об'єкт метод Dispose ще не був викликаний.

```
public void OpenWordDocument(string filePath)
{
  if (this._isDisposed)
    throw new ObjectDisposedException("ManagedWord");
    ...
}
```

5. Додати перевантаження метода Dispose, що приймає Boolean парамент. Перевантажений метод Dispose має розпоряджатись як керованими так і не керованими ресурсами, якщо він був викликаний безпосередньо, в цьому випадку ви передаєте логічний параметр із значенням істини. Якщо ви передаєте істинний параметр зі значенням хиби метод, Dispose повинен тільки намагатися звільнити некеровані ресурси. Ви можете зробити це, якщо об'єкт вже утилізований або буде утилізоваий за допомогою GC.

```
public class ManagedWord : IDisposable
{
  ...
   protected virtual void Dispose(bool isDisposing)
   {
      if (this._isDisposed)
         return;
      if (isDisposing)
      {
        // Вивільнення лише керованих ресурсів.
        ...
      }
      // Завжди вивільняє некеровані ресурси.
      ...
      // Вказує, що об'єкт був видалений.
      this._isDisposed = true;
   }
}
```

6. Додайте код до метода без параметрів Dispose, щоб викликати перевантажений метод Dispose, а після цього викликати метод GC.SuppressFinalize. Метод GC.SuppressFinalize вказує GC, що не потрібно викликати деструктор.

```
public void Dispose()
{
  Dispose(true);
  GC.SuppressFinalize(this);
}
```

Після того, як ви реалізували інтерфейс IDisposable при визначенні вашого класу, ви можете викликати метод Dispose у вашого об'єкта для вивільнення будь-яких ресурсів, що об'єкт може споживати. Ви можете викликати метод Dispose з деструктора, що ви визначили в класі.

Реалізація деструктора

Деструктор – це метод, який буде викликаний, коли GC буде "збирати" цей об'єкт. Для визначення деструктора, вам потрібно додати тільду (~) перед іменем класу. Потім потрібно додати логіку деструктора в дужках.

Наступний приклад демонструє синтаксис додання деструктора.

```
// Визначення деструктора
classManagedWord
{
   ...
   // Деструктор
   ~ManagedWord
   {
      // Логіка деструктора.
   }
}
```

Коли ви визначите деструктор, компілятор автоматично перевизначає метод Finalize класа об'єкта. Тим не менш, ви не можете явно перевизначити метод Finalize; ви повинні оголосити деструктор і дати можливість компілятору виконати перетворення.

Якщо ви хочете гарантувати, щоб метод Dispose постійно викликався, ви можете включити його як частину процесу фіналізації, яку виконує GC. Для цього ви можете додати виклик методу Dispose у деструкторі класу.

Наступний приклад демонструє, як викликати метод Dispose з деструктора.

```
// Виклик метода Dispose з деструктора
classManagedWord
{
   ...
   // Деструктор
   ~ManagedWord
   {
      Dispose(false);
   }
}
```

Управління життєвим циклом об'єкта

Використання типів напряму, що реалізують інтерфейс IDisposable, не є ефективним способом з точки зору управління ресурсами, адже ви повинні також пам'ятати про виклик методу Dispose у вашому коді, коли ви завершили роботу з вашим об'єктом.

Наступний приклад коду показує як викликати метод Dispose для об'єкту, що реалізує інтерфейс IDisposable.

```
// Виклик метода Dispose
var word = new ManagedWord();
 // Код, що використовує об'єкт ManagedWord.
word.Dispose();
```

Викликати метод Dispose явно після того, як код використав об'єкт можна, але якщо код видасть помилку перед викликом метода Dispose, метод Dispose ніколи не буде викликаний. Більш надійним підходом є виклик метода Dispose у блоці finally у try/catch/finally або в try/finally. Будь-який код, що знаходиться в блоці finally завжди виконується, не зважаючи на те, які помилки можуть виникнути. Саме тому цей підхід може завжди гарантувати те, що ваш код викличе метод Dispose.

Наступний приклад коду демонструє як ви можете викликати метод Dispose у блоці finally.

```
// Виклик методу Dispose у блоці finally
varword = default(ManagedWord);
try
{
  word = newManagedWord();
  // Код працює з об'єктом ManagedWord.
}
catch

{
   // Код, що здійснює обробку помилок.
}
finally

{
  if(word!=null)
    word.Dispose();
}
```

Зауваження: Коли ви явно викликаєте метод Dispose, це хороша практика перевірити чи об'єкт не є null перед цим, тому що ви не можете гарантувати стан об'єкта.

Альтернативно(і майже завджи краще), ви можете використовувати usingдля неявного виклику методу Dispose.

Наступний приклад коду демонструє використання using

```
using (var word = default(ManagedWord))
{
  // Код використовує об'єкт ManagedWord.
}
```

Якщо ваш об'єкт не реалізовує з якихось причин інтерфейс IDisposable, try/finally блок – це єдиний безпечний спосіб виконати код для вивільнення ресурсів.

8.2 Завдання до лабораторної роботи(Додаток Ж)

8.3 Контрольні запитання

1. Що таке події?
2. Виклик події.
3. Що таке делегати?
4. Делегати з іменованими методами
5. Делегати з анонімними методами

ПЕРЕЛІК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. ISO 9000 (1-4). Інформаційні технології. Стандарти з керування якістю та забезпечення якості.
2. Андон П.І., Коваль Г.І., Коротун Т.М., Лаврищева Е.М., Суслов В.Ю. Основы качества программных систем.–К.: Академперіодика, 2007.– 860с.
3. Андон Ф.И., Лаврищева Е.М. Методы инженерии компьютерных программных систем.–Наук.думка, 1998.–228с.
4. Бабенко Л.П., Лавріщева К,М, Основи програмної інженерії.–К.: Знання, 2001. – 269с.
5. Бей И.. Взаимодействие разноязыковых модулей. Руководство программистов.– Москва*Санкт- Петербург*Киев, 2005.–869с.
6. Гантер Р. Методы управления проектированием программного обеспечения. – М.: Мир, 1981. – 392 с.
7. Голуб Б.М. C#. Концепція та синтаксис. Навч. посібник / Б.М. Голуб, – Львів: Видавничий центр ЛНУ імені Івана Франка, 2006. – 136 с.
8. Гуриков С. Р. Введение в программирование на языке Visual C#; Форум, Инфра-М, 2013. – 448 с.
9. Иванова Г.С. Объектно-ориентированное программирование: Учеб. для вузов / Г.С Иванова, Т.Н. Ничушкина, Е.К. Пугачев / Под ред. Г.С. Ивановой. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001. – 320 с.
10. Культин Н. Б.C/C++ в задачах и примерах. –СПб.: БХВ-Петербург, 2005.–288 с : ил.
11. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование.–К.: Наукова думка.–1991.–213с.
12. Лаврищева К.М. Методы программирования. Теория, инженерия, практика. – К.; Наукова Думка, 2006.–451с.
13. Лавріщева К.М. Програмна інженерія. –Підручник.–К.: Академперіодика, 2008.– 415с.
14. Мартин Р. С., Мартин М. Принципы, паттерны и методики гибкой разработки на языке C#; Символ-Плюс, 2011. – 768 с.
15. Павловская Т.А. C#. Программирование на языке высокого уровня. Учебник для вузов / Т.А. Павловская – СПб.: Питер, 2007. – 432 с.
16. Пугачев С., Шериев А., Кичинский К. Разработка приложений для Windows 8 на языке C#; БХВ-Петербург, 2013. – 416 с.
17. C++. Объектно-ориентированное программирование. Задачи и упражнения. / В.В. Лаптев, А.В. Морозов, А.В. Бокова. – СПб.: Питер, 2007. – 288 с.
18. Соммервиль І. Інженерия программного обеспечения. – М.; “Вильямс”, 2002. – 624 с./ home page (http://www.software-engin.com)
19. Стадарт ISO/IEC 11404: 1996, 2007. Загальны типи даних МП –Information technology – General- • Purpose Datatypes (GPD).
20. Стандарт ISO/IEC 9126 , ДСТУ Програмна інженерія. Якість продукту (часть1-6).
21. Стиллмен Э., Грин Дж. Изучаем C#; Питер - Москва, 2013. – 688 с.
22. Троелсен Э. Язык программирования C# 2010 и платформа .NET 4.0, 5-е изд. / Э Троелсен. – М.: ООО “И.Д. Вильямс”, 2011. – 1392 с.
23. Фленов Михаил Библия C#; БХВ-Петербург – Москва, 2009. – 560 с.
24. Фленов Михаил Библия C#; БХВ-Петербург, 2011. – 560 с.
25. Хейлсберг А., Торгерсен М., Вилтамут С., Голд П. Язык программирования C#; Питер – Москва, 2012. – 784 с.
26. Шилдт Г. C# 4.0: полное руководство / Г. Шилдт. – М.: ООО “И.Д. Вильямс”, 2011. – 1056 с.

Додаток А
Матеріали до лабораторноїроботи № 1

Таблиця А.1– Типи значень

Тип	**Опис**	**Розмір (в байтах)**	**.NET Type**	**Діапазон**
int	Ціле число	4	System.Int32	-2,147,483,648 to 2,147,483,647
long	Ціле число (Більший діапазон)	8	System.Int64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	Числа з плаваючою точкою	4	System.Single	+/-3.4 x 10^38
double	Числа з плаваючою точкою подвійної точності (більша точність)	8	System.Double	+/-1.7 x 10^308
decimal	Десяткове	16	System.Decimal	28 значущих цифр
char	Один символ	2	System.Char	N/A
bool	Логічний	1	System.Boolean	Trueorfalse
DateTime	Час	8	System.DateTime	0:00:00 on 01/01/0001 to 23:59:59 on 12/31/9999
string	Послідовність символів	2 на символ	System.String	N/A

Таблиця А.2–Зарезервованіключові словаC#

abstract	as	base	boolbreak
byte	case	catchchar	checked
class	const continue	decimal	default
delegate do	double	else	enum event
explicit	extern	false finally	fixed
float	for foreach	goto	if
implicit in	in (generic modifier)	int	interface internal
is	lock	long namespace	new
null	object operator	out	out (generic modifier)
override params	private	protected	public readonly
ref	return	sbyte sealed	short
sizeof	stackalloc static	string	struct
switch this	throw	true	try typeof
uint	ulong	unchecked unsafe	ushort
using	virtual void	volatile	while

Таблиця А.3–ПерелікоператорівC# за типом

Типи	Оператор
Арифтемичні	+, -, *, /, %
Інкремент і декремент	++, --
Порівняння	==, !=, <, >, <=, >=, is
Конкатенаціярядків	+
Логічні/бітовіоперації	&, |, ^, !, ~, &&, ||
Індексація (перелікпочинається з 0)	[ ]
Перетвореннятипів	( ), as
Присвоєння	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, ??
Побітовийзсув	<<, >>
Інформаціяпротип	sizeof, type
Конкатенації і видаленняделегатів	+, -
Перевіркапомилкипереповнення	checked, unchecked
Розіменуваннятаадреси	*, ->, [ ], &
Умовний (тернарнийоператор)	?:

Перелік задач з відповідних тематик до лабораторної роботи № 1:

- *оголошення змінних:*

1. Оголосіть змінні, необхідні для обчислення площі прямокутника.

2. Оголосіть змінні, необхідні для перерахунку ваги з фунтів в кілограми.

3. Визначте вихідні дані і оголосіть змінні, необхідні для обчислення доходу по вкладу.

4. Оголосіть змінні, необхідні для обчислення площі кола.

5. Оголосіть змінні, необхідні для обчислення площі поверхні циліндра.

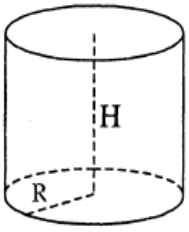
- *інструкція присвоєння:*

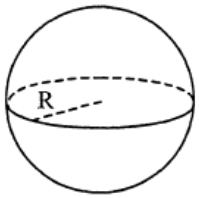
1. Запишіть у вигляді інструкції присвоєння формулу обчислення значення функції $y = -2{,}1x^2 - 0{,}24x + 1{,}6$.

2. Запишіть у вигляді інструкції присвоєння формулу обчислення площі трикутника: $S = \frac{1}{2} \cdot a \cdot h$, де *a*- довжина основи;*h* - висота трикутника.

3. Запишіть у вигляді інструкції присвоєння формулу обчислення площі трапеції: $S = \frac{a+b}{2} \cdot h$, де *a* і *b* - довжини основ;*h* - висота трапеції.

4. Запишіть у вигляді інструкції присвоєння формули обчислення об'єму циліндра: $V = \pi \cdot R^2 \cdot H$.





5. Оголосіть необхідні змінні і запишіть у вигляді інструкції присвоєння формули обчислення об'єму кулі: $V = \frac{3}{4} \cdot \pi \cdot R^3$.

- *виведення і введення інформації:*

1. Написати інструкцію виведення значень цілих змінних a, b і c. Значення кожної змінної повинно бути виведено вокремому рядку.

2. Написати інструкції виведення значень дробових змінних x_1 і x_2. На екрані перед значенням змінної повиненбути виведений пояснювальний текст, який представляє собою ім'язмінної, за яким слідує знак "дорівнює".

3. Оголосіть необхідні змінні і напишіть фрагмент програми обчислення об'єму паралелепіпеда, що забезпечує введеннявихідних даних.

4. Оголосіть необхідні змінні і напишіть фрагмент програми обчислення площі поверхні кулі: $S = 4 \cdot \pi \cdot R^2$, що забезпечує введеннявихідних даних.

5. Оголосіть необхідні змінні і напишіть інструкції введення вихідних даних для програми обчислення вартості покупки, що складається з декількох зошитів і олівців.Передбачається, що користувач буде вводити

дані про кожну складову покупки в окремому рядку: спочатку ціну,потім кількість.

- *програми з лінійною структурою:*

1. Написати програму обчислення об'єму паралелепіпеда. Нижче наведено рекомендований вигляд екрану під час виконанняпрограми.

Обчислення обсягу паралелепіпеда.
Введіть вихідні дані:
Довжина (см) ->9
Ширина (см) ->7.5
Висота (см) ->5
Об'єм паралелепіпеда: 337.50*куб.см.*

2. Написати програму обчислення площі паралелограма. Нижче наведено рекомендований вигляд екрану під час виконання.

Обчислення площі паралелограма.
Введіть вихідні дані:
Сторона (см) ->9
Сторона (см) ->7.5
Градусна міра кута (градуси) ->30
Площа паралелограма: 67.875*кв.см.*

3. Написати програму обчислення опору електричного ланцюга, що складається з двох паралельно з'єднаних опорів.Нижче наведено рекомендованийвиглядвид екрану під часвиконання програми.

Обчислення опору електричного кола, при паралельному з'єднанні елементів.
Введіть вихідні дані:
Величина першого опору (Ом) ->15
Величина другого опору (Ом) ->20
Опір ланцюга: 8.57 *Ом*

4. Написати профамму обчислення відстані між населеними пунктами, зображеними на карті. Нижче наведено рекомендованийвигляд екрану під час виконання програми.

Обчислення відстані між населеними пунктами.
Введіть вихідні дані:
Масштаб карти (кількість кілометрів в одному сантиметрі) ->120
Відстань між точками, які позначають населені пункти (См) ->3.5
Відстань між населеними пунктами 420 *км.*

5. Написати програму обчислення величини доходу по вкладу. Процентна ставка (% річних) і час зберігання (днів) задаються під час роботи програми. Нижче наведено рекомендованийвигляд екрану під час виконання програми.

Обчислення доходу за вкладом.
Введіть вихідні дані:
Величина вкладу (грн.) ->2500
Термін вкладу (днів) ->30
Процентна ставка (річних) ->20
Дохід: 41.10*грн.*
Сума після закінчення терміну вкладу: 2541.10*грн.*

Додаток Б
Матеріали до лабораторноїроботи № 2

Пелелік задач з відповідних тематикдо лабораторної роботи № 2

- *інструкціїif, switch:*

1. Написати програму, яка обчислює частку двох чисел. Програма повинна перевіряти правильність введених користувачемданих і, якщо вони невірні (дільник дорівнює нулю), видавати повідомлення про помилку. Нижче наведено рекомендований вигляд екрану під час виконання програми.

Обчислення частки.
Введіть ділене і дільник, ->12 0
Ви помилилися. Дільник не повинен дорівнювати нулю.

2. Написати програму обчислення опору електричного струму, що складається з двох опорів. Опори можуть бути з'єднані послідовно або паралельно. Нижче наведено рекомендований вигляд екрану під час виконання програми.

Обчислення опору електричного кола.
Введіть вихідні дані:
Величина першого опору (Ом) ->15
Величина другого опору (Ом) ->27.3
*Тип з'єднання (*1 - *послідовне,* 2 - *паралельне)* ->2
Опір ланцюга: 9.68 *Ом*

3. Написати програму, яка переводить час з хвилин і секунд в секунди. Програма повинна перевіряти правильністьведених користувачем даних і в разі, якщо дані невірні,виводити відповідне повідомлення. Рекомендований вигляд екрану під час виконання програми наведено нижче.

Введіть час (хвилини.секунди) ->2.90
Помилка! Кількість секунд не може бути більше 60

4. Написати програму, яка перевіряє, чи є рік високосним. Нижче наведено рекомендований вигляд екрану під час роботи програми.

Введіть рік, наприклад 2000, ->2001
2001 рік - не високосний

5. Написати програму розв’язання квадратного рівняння. програма повинна перевіряти правильність вихідних даних і ввипадку, якщо коефіцієнт при другому ступені невідомого дорівнюєнулю, виводити відповідне повідомлення. Нижче наведено рекомендований вигляд екрану під час виконання програми.

Розв’язання квадратного рівняння
Введіть значення коефіцієнтів ->12 27 -10
Корені рівняння:
x_1 = -25.551
x_2 = -28.449

6. Написати програму обчислення вартості покупки з урахуванням знижки. Знижка в 10% надається, якщо сума покупкибільше 1000 грн.

Нижче наведено рекомендований вигляд екрану підчас виконання програми.

Обчислення вартості покупки з урахуванням знижки
Введіть суму покупки (грн.) ->1200
Вам надається знижка 10%
Сума покупки з урахуванням знижки: 1080.00*грн.*

7. Написати профамму, яка обчислює оптимальну вагу для користувача, порівнює його з реальним і видає рекомендацію пронеобхідності поправитися або схуднути. Оптимальна вага обчислюється за формулою: Зріст (см) - 100. Рекомендований вигляд екранупід час виконання програми наведено нижче.

Введіть зріст (см) і вагу (кг), ->170 68
*Вам треба набрати вагу*2.00 *кг.*

8. Напишіть програму, яка запитує у користувача номер місяця і потім виводить відповідну назву часуроку. У разі, якщо користувач введе неприпустимечисло, програма повинна вивести повідомлення "Помилка введенняданих ". Нижче наведено рекомендований вигляд екрану під часроботи програми.

Введіть номер місяця
->11
зима

9. Написати програму, яка запитує у користувача номер дня тижня і виводить одне з повідомлень: "Робочий день", "Суббота" або "Неділя".

10. Напишіть програму перевірки знання історії архітектури. Програма повинна вивести питання і три варіанти відповіді.Користувач повинен вибрати правильну відповідь і ввести йогономер. Нижче наведено рекомендований вигляд екрану під час виконання програми.

Архітектор Ісаакіївського собору:
1. ДоменікоТрезини
2. Огюст Монферран
3. Карл Россі
Введіть номер правильної ->3
Ви помилилися.
Архітектор Ісаакіївського собору - Огюст Монферран.

- *цикли for,do while, while:*

1. Написати програму, яка обчислює суму перших *n*членів ряду: 1, 3, 5, 7 ... Кількість членів ряду, які будуть шумуватись,задається під час роботи програми. Нижче наведено рекомендований вигляд екрану під час роботи програми.

Обчислення часткової суми ряду: 1,3,5,7 ...
Введіть кількість членів ряду для сумування ->15
Сума перших 15 *членів ряду дорівнює* 330

2. Написатипрограму, яка виводитьтаблицюзначеньфункції $y = |x-2| + |x+1|$. Діапазонзміни аргументу від -4 до 4,крокзбільшення аргументу 0,5.

3. Напишітьпрограму, яка виводить на екран квадратПіфагора - таблицюмноження. Рекомендований вигляд екрану:

	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90

4. Написатипрограму, яка перетворюєвведенекористувачемдесяткове число в двійкове. Рекомендований вигляд екрану під час виконання програми наведено нижче.

Перетвореннядесяткового числа в двійкове
*Введітьціле число від*0*до* 255 ->49
Десятковому числу 49*відповідаєдвійкове*00110001

5. Написати програму, яка обчислює факторіал введеного з клавіатури числа. Рекомендований вигляд екрану під час виконання програми наведено нижче.

Обчислення факторіала.
Введіть число, факторіал якого треба вирахувати->7
Факторіал 7 *дорівнює* 5040

6. Напишіть програму, яка перевіряє, чи є введене користувачем ціле число простим. Рекомендований вигляд екрану під час виконання програми наведено нижче.

Введіть ціле число і натисніть ->45
45 - *не просте число.*

7. Написати програму наближеного обчислення інтеграла методом трапецій. Після кожного циклу обчислень програма повинна виводити обчислене значення, кількість і величинуінтервалів.

8. Напишіть програму, яка виводить на екран таблицю значень функції $y = -2{,}1x^2 - 0{,}24x + 1{,}6$ в діапазоні від -4 до 4. Крок зміниаргументу 0,5.

9. Напишіть програму, яка обчислює число "π" із заданоюкористувачем точністю. Для обчислення значеннячисла "Пі" скористайтеся тим, що значення часткової сумиряду $1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-...$ при підсумовуванні досить великоїкількості членів наближається до значення $\frac{\pi}{4}$. Рекомендований вигляд екрану:.

Задайте точність обчислення ПІ ->0.00001
*Значення числа ПІ з відповідною точністюдорівнює*3.143589
Підсумовано 502 члена ряду.

10. Написати програму, яка обчислює найбільший спільнийдільник двох цілих чисел.

ДодатокВ
Матеріали до лабораторноїроботи № 3

Пелелік задач до лабораторної роботи № 3:

1. Написати програму, яка виводить мінімальний елементведеного з клавіатури масиву цілих чисел. Нижче наведенорекомендований вигляд екрану під час роботи програми.

Пошук мінімального елемента масиву.
Введіть елементи масиву (5 цілих чисел)->23 0 45 -5 12
Мінімальний елемент масиву: -5

2. Написати програму, яка обчислює середнє арифметичнененульових елементів введеного з клавіатури масивуцілих чисел. Нижче наведенорекомендований вигляд екрану під час роботи програми.

Введіть елементи масиву (10 цілих чисел) ->23 0 45 -5 12 0 -2 30 0 64
Сума елементів масиву: 184
Кількість ненульових елементів: 7
Середнє арифметичне ненульових елементів: 23.86

3. Написати програму, яка перевіряє, чи становлятьелементи введеного з клавіатури масиву зростаючу послідовність.

4. Написати програму, яка обчислює, скільки разів введенез клавіатури число зустрічається в масиві.

5. Написати програму, яка об'єднує два упорядкованихпо зростанню масиву в один, також упорядкований масив. Нижче наведенорекомендований вигляд екрану під час роботи програми.

Об'єднання двох упорядкованих за зростанням масивів.
Введіть елементи першого масиву(5 цілих чисел) ->1 3 5 7 9
Введіть елементи другого масиву(5 цілих чисел) ->2 4 6 8 10
*Масив – результат*1 2 3 4 5 6 7 8 9 10

6. Написатипрограму, яка вводить по рядках з клавіатуридвовимірниймасив і обчислює суму йогоелементів по стовпцях.

7. Написатипрограму, яка вводить по рядках з клавіатуридвовимірниймасив і обчислює суму йогоелементівпо рядках.

8. Написатипрограму, яка вводить по рядках з клавіатуридвовимірниймасив і обчислюєсереднєарифметичнейогоелементів.

9. Написатипрограму, яка обчислюєвизначникквадратноїматриці другого порядку. Нижче наведенорекомендований вигляд екрану під час роботи програми.

Введітьматрицю другого порядку.
->5 -7
->1. 3
*визначникматриціdорівнює*22.00

10. Написатипрограму, яка визначає номер рядка квадратноїматриці, сума елементівякої максимальна.

Додаток Г
Матеріали до лабораторноїроботи № 4

Задачідо лабораторної роботи № 4

1. Написати функцію Procent, яка повертає відсотоквід отриманого в якості аргументу числа.

2. Написати функцію "Факторіал" та програму, яка використовує дану функцію для виведення таблиці факторіалів.

3. Написати функцію Dohod, яка обчислює дохід за вкладом. Вихідними даними для функції є: величина вкладу, процентна ставка (річних) і термін вкладу (кількість днів).

4. Написати функцію glasn, яка повертає 1, якщо символ, отриманий функцією як аргумент, є голосною буквою алфавіту, і нуль - в іншому випадку.

5. Написати функцію sogl, яка повертає 1, якщо символ, отриманий функцією як аргумент, є приголосною буквою алфавіту, і 0 - в іншому випадку.

6. Написати функцію frame, яка виводить на екран рамку. Як параметри функції повинні передаватися координати лівого верхнього кута і розмір рамки.

7. Написати функцію, яка обчислює значення a^b. Числа a й b можуть бути будь-якими дробовими позитивними числами.

8. Написати функцію, що забезпечує розв’язання квадратного рівняння. Параметрами функції повинні бути коефіцієнти і корені рівняння. Значення, які обраховує функція, маютьпередаватися в програму, яка викликається з інформацією про наявність коренів рівняння: 2 - два різних корені, 1 - корені однакові, рівняння не має розв’язків. Крім того, функція повинна перевіряти коректність вихідних даних. якщо вихідні дані невірні, то функція повинна повертати - 1.

9. Написати функцію, яка виводить рядок, що складається з однакових символів. Довжина рядка і символ є параметрами процедури.

10. Написати функцію, яка обчислює об’єм і площу поверхні паралелепіпеда.

Додаток Д
Матеріали до лабораторноїроботи № 5

Завдання до лабораторної роботи № 5

Варіант 1

1.Описати структуру з ім'ям STUDENT, що містить наступні поля:

• NAME - прізвище та ініціали;
• GROUP - номер групи;
• SES - успішність (масив з п'яти елементів).

2. Написати програму, яка виконує наступні дії:

• введення з клавіатури даних в масив STUD1, що складається з десяти структур типу STUDENT; записи повинні бути впорядковані за зростанням вмісту поля GROUP;
• виведення на екран прізвищ і номерів груп для всіх студентів, включених в масив, якщо середній бал студента більше 4,0;
• якщо таких немає, вивести відповідне повідомлення.

Варіант 2

1. Описати структуру з ім'ям STUDENT, що містить наступні поля:

• NAME - прізвище та ініціали;
• GROUP- номер групи;
• SES- успішність (масив з п'яти елементів).

2. Написати програму, яка виконує наступні дії:

• введення з клавіатури даних в масив STUD1, що складається з десяти структур типу STUDENT; записи повинні бути впорядковані за зростанням середнього бала;
• виведення на екран прізвищ і номерів груп для всіх студентів, які мають оцінки 4 і 5;
• якщо таких немає, вивести відповідне повідомлення.

Варіант 3

1. Описати структуру з ім'ям STUDENT, що містить наступні поля:

• NAME - прізвище та ініціали;
• GROUP- номер групи;
• SES- успішність (масив з п'яти елементів).

2. Написати програму, яка виконує наступні дії:

• введення з клавіатури даних в масив STUD1, що складається з десяти структур типу STUDENT; записи повинні бути впорядковані за алфавітом;
• виведення на екран прізвищ і номерів груп для всіх студентів, які мають хоча б одну оцінку 2;
• якщо таких студентів немає, вивести відповідне повідомлення.

Варіант 4

1. Описати структуру з ім'ям AEROFLOT, що містить наступні поля:

• NAZN - назва пункту призначення рейсу;
• NUMR - номер рейсу;
• TIP - тип літака.

2. Написати програму, яка виконує наступні дії:

• введення з клавіатури даних в масив AIRPORT, що складається з семи елементів типу AEROFLOT; записи повинні бути впорядковані за зростанням номера рейсу;

• виведення на екран номерів рейсів і типів літаків, що вилітають в пункт призначення, назва якого співпало з назвою, введеним з клавіатури;
• якщо таких рейсів немає, вивести відповідне повідомлення.

Варіант 5

1. Описати структуру з ім'ям AEROFLOT, що містить наступні поля:
• NAZN - назва пункту призначення рейсу;
• NUMR - номер рейсу;
• TIP - тип літака.
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив AIPORT, що складається з семи елементів типу AEROFLOT; записи повинні бути розміщені в алфавітному порядку за назвами пунктів призначення;
• виведення на екран пунктів призначення і номерів рейсів, що обслуговуються літаком, тип якого введено з клавіатури;
• якщо таких рейсів немає, вивести відповідне повідомлення.

Варіант 6

1. Описати структуру з ім'ям WORKER, що містить наступні поля:
• NAME - прізвище та ініціали працівника;
• POS - назва займаної посади;
• YEAR - рік прийому на роботу.
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив TABL, що складається з десяти структур типу WORKER; записи повинні бути розміщені в алфавітному порядку.
• виведення на екран прізвищ працівників, чий стаж роботи в організації перевищує значення, введене з клавіатури;
• якщо таких працівників немає, вивести відповідне повідомлення.

Варіант 7

1. Описати структуру з ім'ям TRAIN, що містить наступні поля:
• NAZN - назва пункту призначення;
• NUMR - номер потяга;
• TIME - час відправлення.
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив RASP, що складається з восьми елементів типу TRAIN; записи повинні бути розміщені в алфавітному порядку за назвами пунктів призначення;
• виведення на екран інформації про потяги, що відправляються після введеного з клавіатури часу;
• якщо таких потягів немає, вивести відповідне повідомлення.

Варіант 8

1. Описати структуру з ім'ям TRAIN, що містить наступні поля:
• NAZN - назва пункту призначення;
• NUMR - номер потяга;
• TIME - час відправлення.
2. Написати програму, яка виконує наступні дії;

• введення з клавіатури даних в масив RASP, що складається з шести елементів типу TRAIN; записи повинні бути впорядковані за часом відправлення потяга;
• виведення на екран інформації про потяги, що прямують в пункт, назву якого введено з клавіатури;
• якщо таких потягів немає, вивести відповідне повідомлення.

Варіант 9

1. Описати структуру з ім'ям TRAIN, що містить наступні поля:
• NAZN - назва пункту призначення;
• NUMR - номер потяга;
• TIME - час відправлення.
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив RASP, що складається з восьми елементів типу TRAIN; записи повинні бути впорядковані за номерами потягів;
• виведення на екран інформації про потяг, номер якого введено з клавіатури;
• якщо таких потягів немає, вивести відповідне повідомлення.

Варіант 10

1. Описати структуру з ім'ям MARSH, що містить такі, поля:
• BEGST - назва початкового пункту маршруту;
• TERM - назва кінцевого пункту маршруту;
• NUMER - номер маршруту.
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив TRAFIC, що складається з восьми елементів типу MARSH; записи повинні бути впорядковані за номерами маршрутів;
• виведення на екран інформації про маршрут, номер якого введено з клавіатури;
• якщо таких маршрутів немає, вивести відповідне повідомлення.

Варіант 11

1. Описати структуру з ім'ям MARSH, що містить наступні поля:
• BEGST - назва початкового пункту маршруту;
• TERM - назва кінцевого пункту маршруту;
• NUMER - номер маршруту.
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив TRAFIC, що складається з восьми елементів типу MARSH; записи повинні бути впорядковані за номерами маршрутів;
• виведення на екран інформації про маршрути, які починаються або завершуються в пункті, назва якого введено з клавіатури;
• якщо таких маршрутів немає, вивести відповідне повідомлення.

Варіант 12

1. Описати структуру з ім'ям NOTE, що містить наступні поля:
• NAME - прізвище, ім'я;
• TELE - номер телефону;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив BLOCKNOTE, що складається з восьми елементів типу NOTE; записи повинні бути впорядковані по датах днів народження;

• виведення на екран інформації про людину, номер телефону якого введено з клавіатури;
• якщо такого немає, вивести відповідне повідомлення.

Варіант 13

1. Описати структуру з ім'ям NOTE, що містить наступні поля:
• NAME - прізвище, ім'я;
• TELE - номер телефону;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив BLOCKNOTE, що складається з восьми елементів типу NOTE; записи повинні бути розміщені за алфавітом;
• виведення на екран інформації про людей, чиї дні народження припадають на місяць, значення якого введено з клавіатури;
• якщо таких немає, вивести відповідне повідомлення.

Варіант 14

1. Описати структуру з ім'ям NOTE, що містить наступні поля:
• NAME - прізвище, ім'я;
• TELE - номер телефону;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив BLOCKNOTE, що складається з восьми елементів типу NOTE; записи повинні бути впорядковані за трьома першими цифрами номера телефону;
• виведення на екран інформації про людину, чиє прізвище введене з клавіатури;
• якщо такого немає, вивести відповідне повідомлення.

Варіант 15

1. Описати структуру з ім'ям ZNAK, що містить наступні поля:
• NAME - прізвище, ім'я;
• ZODIAC - знак Зодіаку;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив BOOK, що складається з восьми елементів типу ZNAK; записи повинні бути впорядковані по датах днів народження;
• виведення на екран інформації про людину, чиє прізвище введене з клавіатури;
• якщо такого немає, вивести відповідне повідомлення.

Варіант 16

1. Описати структуру з ім'ям ZNAK, що містить наступні поля:
• NAME - прізвище, ім'я;
• ZODIAC - знак Зодіаку;
• BDAY - день народження (масив з трьох чисел).
2. Написати програму, яка виконує наступні дії:
• введення з клавіатури даних в масив BOOK, що складається з восьми елементів типу ZNAK; записи повинні бути впорядковані по датах днів народження;
• виведення на екран інформації про людей, які народилися під знаком, назву якого введено з клавіатури;
• якщо таких немає, вивести відповідне повідомлення.

Варіант 17

1. Описати структуру з ім'ям ZNAK, що містить наступні поля:

• NAME - прізвище, ім'я;
• ZODIAC - знак Зодіаку;
• BDAY - день народження (масив з трьох чисел).

2. Написати програму, яка виконує наступні дії:

• введення з клавіатури даних в масив BOOK, що складається з восьми елементів типу ZNAK; записи повинні бути впорядковані по знакам Зодіаку;
• виведення на екран інформації про людей, які народилися в місяць, значення якого введено з клавіатури;
• якщо таких немає, вивести відповідне повідомлення.

Варіант 18

1. Описати структуру з ім'ям PRICE, що містить наступні поля:

• TOVAR - назва товару;
• MAG - назва магазину, в якому продається товар;
• STOIM - вартість товару в грн.

2. Написати програму, яка виконує наступні дії:

• введення з клавіатури даних в масив SPISOK, що складається з восьми елементів типу PRICE; записи повинні бути розміщені в алфавітному порядку за назвами товарів;
• виведення на екран інформації про товар, назву якого введено з клавіатури;
• якщо таких товарів немає, вивести відповідне повідомлення.

Варіант 19

1. Описати структуру з ім'ям PRICE, що містить наступні поля:

• TOVAR - назва товару;
• MAG - назва магазину, в якому продається товар;
• STOIM - вартість товару в грн.

2. Написати програму, яка виконує наступні дії:

• введення з клавіатури даних в масив SPISOK, що складається з восьми елементів типу PRICE; записи повинні бути розміщені в алфавітному порядку за назвами магазинів;
• виведення на екран інформації про товари, що продаються в магазині, назва якого введено з клавіатури;
• якщо такого магазину немає, вивести відповідне повідомлення.

Варіант 20

1. Описати структуру з ім'ям ORDER, що містить наступні поля:

• PLAT - розрахунковий рахунок платника;
• POL - розрахунковий рахунок одержувача;
• SUMMA - перераховується сума в грн.

2. Написати програму, яка виконує наступні дії:

• введення з клавіатури даних в масив SPISOK, що складається з восьми елементів типу ORDER; записи повинні бути розміщені в алфавітному порядку за розрахунковими рахунками платників;
• виведення на екран інформації про суму, знятої з розрахункового рахунку платника, введеного з клавіатури;
• якщо такого розрахункового рахунку немає, вивести відповіднеповідомлення.

Додаток Е
Матеріали до лабораторноїроботи № 6

Таблиця Е.1 – Модифікатори доступу

Модифікаторид оступу	Опис
public	Надаєдоступдочастиникласубудьякійчастинікодуякаміститьоб’єкт
internal	Надаєдоступдочастинивтомусамомублоціkоду,але ненадаєдоступупозаним.
private	Надаєдоступтількидлякодувсерединіkласу.Встановлюєтьсязазамовчуванням,якщоневизначеноіншого модифікатора.
protected	Надаєдоступтількивсерединіkласутакласамнаслідникам.

Рисунок Е.1

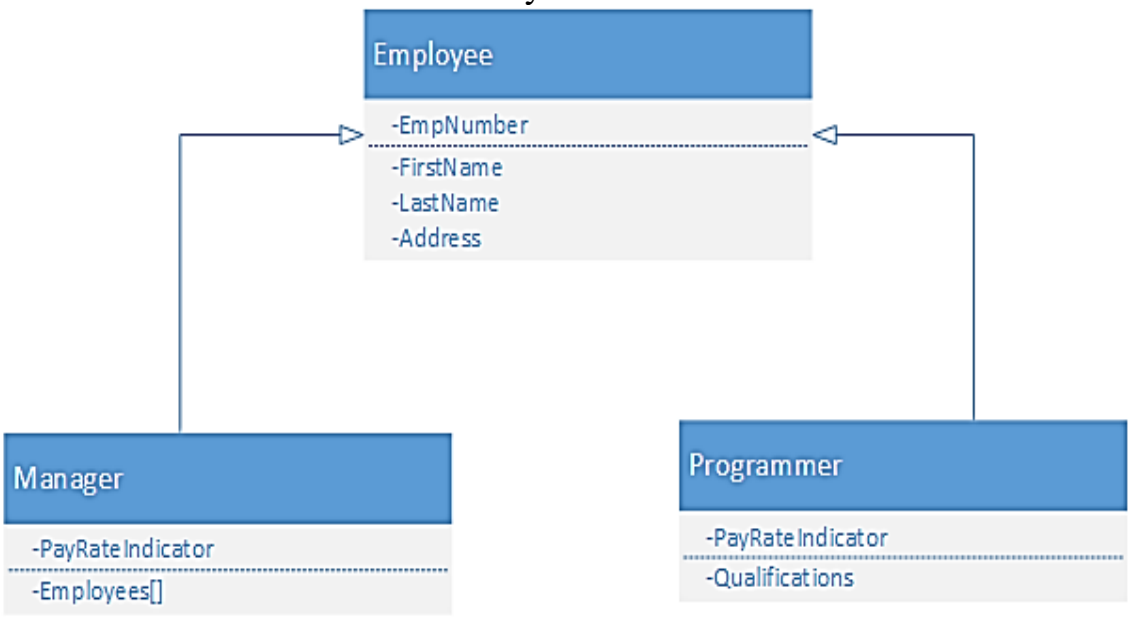
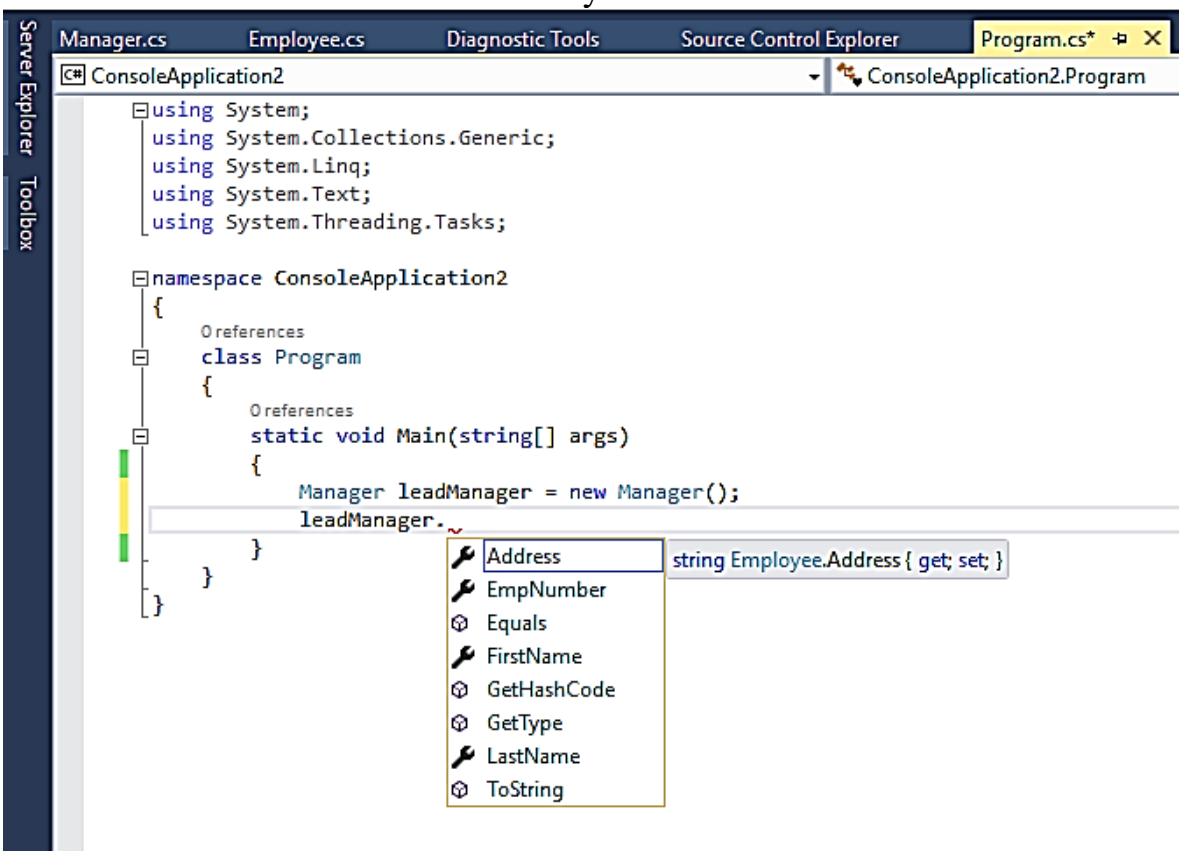

Employee
-EmpNumber
-FirstName
-LastName
-Address
Manager
-PayRateIndicator
-Employees[]
Programmer
-PayRateIndicator
-Qualifications

Рисунок Е.2


Server Explorer
Toolbox
Manager.cs
Employee.cs
Diagnostic Tools
Source Control Explorer
Program.cs*
ConsoleApplication2
ConsoleApplication2.Program
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApplication2
{
0 references
class Program
{
0 references
static void Main(string[] args)
{
Manager leadManager = new Manager();
leadManager.
}
}
}
Address
EmpNumber
Equals
FirstName
GetHashCode
GetType
LastName
ToString
string Employee.Address { get; set; }

Завдання до лабораторної роботи № 6

1.*Загальне завдання.* Визначити користувальницький клас в відповідно до варіанта завдання. Додати в клас наступні конструктори: без параметрів, з параметрами, копіювання.Визначити в класі деструктор, компоненти-функції для перегляду і установки полів даних. Написати демонстраційну програму,в якій створюються і руйнуються об'єкти користувацького класу і кожен виклик конструктора і деструктора супроводжується видачею відповідного повідомлення (який об'єкт який конструктор або деструктор викликав). При роботі з класами користуватисяClassWizard.

Індивідуальнезавдання(відповіднодоваріанта):

Варіант	Завдання
1	Створити клас employee. Клас повинен включати поле типуint для зберігання номера співробітника і поле типу float для зберігання величини його окладу. Методи класу повиннідозволяти користувачеві вводити і відображати данікласу. Написати функцію main (), яка вимагатиме у користувача ввести дані для трьох співробітників і виведеотриману інформацію на екран.
2	Створити клас Int, що імітує стандартний тип int. Єдине поле цього класу повинно мати тип int.Створити методи, які будуть встановлювати значенняполя рівним нулю, ініціалізувати його цілим значенням, виводити значення поля на екран і додавати двазначення типу int. Написати програму, в якій будуть створені три об'єкти класу Int, два з яких - ініціалізовані. Додати два ініціалізованих об'єкти, привласнити результат третього, а потім відобразити результат на екрані.
3	Створити клас типу двохзв'язний список. Поля-дані:покажчик на область пам'яті типу void, покажчики на наступний і попередній. Функції-члени додають елемент до списку, видаляють елемент зі списку, виводять елементи списку від початку і від кінця, шукають заданий елемент в списку.
4	Створити клас типу дата з полями: день (1-31), місяць (1-12), рік (ціле число). Клас має конструктор. Функції-члени установки дня, місяця і року, функції-члени отримання дня, місяця і року, а також дві функції-члени друку: друк за шаблоном «5 січня 2017 року"і "05.01.2017". Функції-члени установки полів класу повинні перевіряти коректність параметрів, які задаються.
5	Створити клас типу прямокутник. Поля - висота і ширина. Функції-члени обчислюють площу, периметр, встановлюють поля і повертають значення. Функції-члени установки полів класу повинні перевіряти коректність параметрів, які задаються. Функція друку.
6	Створити клас типу коло. Поля-дані: радіус, координати центру. Функції-члени обчислюють площу, довжину кола, встановлюють поля і повертають значення.Функції-члени установки полів класу повинні перевіряти коректність параметрів, які задаються (не рівні нулю і не від’ємні). Функція друку.
7	Створити клас множини Set. Функції-члени реалізують додавання і видалення елемента, перетин і різницю множин
8	Створити клас типу квадрат. Поля-дані: сторона. Функції-члени обчислюють площу, периметр, встановлюютьполя і повертають значення.

	Функції-члени установки полів класу повинні перевіряти коректність параметрів, які задаються. Функція друку
9	Створити клас типу час з полями: години (0-23), хвилини (0-59), секунди (0-59). Клас має конструктор. Функції-члени установки часу, отримання години, хвилини і секунди, а також дві функції-члени друку: друк за шаблоном «16 годин 18 хвилин 3 секунди» і «4 p.m. 18 хвилин 3 секунди». Функції-члени установки полів класу повинні перевіряти коректність параметрів, які задаються.
10	Створити клас одновимірний масив цілих чисел (вектор) з полями - кількість фактичних елементів, масив (динамічний). Функції-члени: звернення до окремого елементу масиву, виведення масиву на екран, поелементного додавання і віднімання зі скаляром, виведення елемента за заданим індексом.

2. *Загальне завдання*.Визначте ієрархію класів (відповідно до варіанта- виділити базовий і похідні). Реалізувати класи (самостійно задати члени-дані та методи класу). Написати демонстраційну програму, в якій створюються об'єкти різнихкласів. Визначення класів (* .h), їх реалізацію (* .cpp), демонстраційну програму (* .cpp) помістити в окремі модулі.Нижче наведено перелік класів.

Індивідуальнезавдання(відповіднодоваріанта):

Варіант	Завдання
1	Класи - людина (ім'я, дата народження), абітурієнт (кількість балів), студент (курс, група, факультет), викладач (посада, кафедра)
2	Класи-рослини (назва, вид), дерево (вік), квітка (довжина стебла), троянда (колір)
3	Класи - кадри (ім'я), рітник (спеціальність, цех), інженер (кваліфікація, підрозділ), адміністрація (посада)
4	Класи - ВНТУ (адреса), факультет (назва), група (номер, староста, курс), підгрупа (номер, кількість студентів)
5	Класи - друковане видання (видавництво, рік, назва), журнал (номер, місяць), книга (тематика, автор, кількість сторінок), підручник (призначення)
6	Класи -ссавці (рік), парнокопитні (середовище проживання), птиці (хижаки), тварина (вид, рід, вага)
7	Класи - місце (площа, назва), область (кількість населених пунктів, керівництво), місто (область, кількість жителів, мер), село (район)
8	Класи - товар (назва), радіотовари (призначення), продукт (відділ), молочний продукт (різновид, дата виготовлення), транзистор (тип, номер)
9	Класи - автомобіль (марка, номер), (номер, кількість пасажирів), транспортний засіб (середня швидкість, вид палива, рік випуску)
10	Класи - республіка (вид, уряд), монархія (вид,ім'я монарха), королівство (король), держава (назва, грошова одиниця, символіка)

Додаток Є
Матеріали до лабораторної роботи № 7

Таблиця Є.1 – Модифікатори доступу для інтерфейсів

Модифікатори доступу	Визначення
public	Інтерфейсдоступний у будь-якійзбірці коду.
internal	Інтерфейсдоступний в межах однієїзбірки, де заданий. Це є значення за замовчуванням, якщови незазначитеіншиймодифікатор доступу.

Таблиця Є.2 – Класи колекцій в просторі імен System.Collections

Клас	Визначення
ArrayList	ArrayList - це список загальногопризначення, щозберігаєлінійноколекціюоб'єктів. ArrayListвключаєметоди і властивості, які дозволяютьдодати, видалити, перерахуватиелементиколекціїабопосортуватиїх. Звернітьувагу, щодалі ми будеморозглядатикласList, якиймайже у будь-якійситуаціїкращевикористовуватизамістьArrayList, аджеListвикористовуємеханізмузагальнення(generics), якийдопомагаєуникатипомилокприведеннятипів.
BitArray	BitArray - цекласСписок, щоявляєсобоюколекціюбітівякзначень Boolean. BitArrayунайчастішевсьоговикористовуєтьсяприздійсненніоперацій з бітами і булевійарифметицітавключаєметодидляздійсненнястандартнихбулевихоперацій, такихяк AND, NOT та XOR.
Hashtable	Hashtable - цеклас Словник загальногопризначення, щозберігаєколекцію пар ключ/значення. Hashtablевключаєметоди і властивості, щонадаютьможливістьпереглядатиелементи за ключем, додаватиелементи, видалятиелементи та перевірятипевнезначенняіндексачизначення.
Queue	Queue - цеклас типу "перший прийшов - перший пішов". Queueевключаєметоди для доданняоб'єктів в кінецьчерги(Enqueue) та перегляд об'єктів з початку черги (Dequeue).
SortedList	SortedList -цеклас, щозберігаєколекціюключ/значеннявідсортованузаключем. НадодачудофункціональностікласуHashtable, SortedListдозволяєпроходитисьпоелементахпоключуабопоіндексу.
Stack	КласStack - цеклас типу "перший прийшов - останнійпішов". Stackвключаємотоди для перегляду отанньогоелемента без йоговидалення (Peek), додаванняелемента до вершини стеку (Push), і видаленняостаннього (найвищого) елемента стеку та поверненнявказівника на новийелемент, щознаходиться на вершині стеку (Pop).

Таблиця Є.3 – Класи колекцій в просторі імен System.Collections.Specialized

Клас	Визначення
ListDictionary	ListDictionary - цеклас словник, щооптимізований для невеликих колекцій. Як правило, якщо ваша колекціямістить 10 елементівчименшевикористовуйтеListDictionary. Якщовашаколекціябільша, товикористовуйтеHashtable.
HybridDictionary	HybridDictionary - цеклас словник, щови можете використовувати, коли ви не можете передбачитирозмірколекції. HybridDictionaryвикористовуєреалізаціюListDictionary, якщоколекція не велика і перемикається на Hashtable, якщоколекціяпочинаєзростати.
OrderedDictionary	OrderedDictionary- цепроіндексованийсловник, щодозволяєперебиратиелементизаіндексомчизаключем. Помітьте, щонавідмінувідкласуSortedList, елементи в OrderedDictionaryунесортованізаключем.
NameValueCollection	NameValueCollection - цеіндексований словник, у якому і ключ і значенняце рядки. NameValueCollectionвидастьпомилку, якщови будете намагатисьприсвоїти для ключа чи для значення не рядок. Ви можете пробігтись по елементах за ключемчи за індексом.
StringCollection	StringCollection - це список, де коженелементколекції - це рядок. Використовуйтецейклас, якщовихочетезберігатипросту, лінійноколекціюрядків.
StringDictionary	StringDictionary - це словник, у якому і ключ і значення рядки. На відмінувідNameValueCollectionкласу, ви не можете переглядатиелементи з StringDictionary за індексом.
BitVector32	BitVector32 - це структура, щопредставляє 32-бітні значення як масивбітів так і значенняцілого числа. НавідмінувідBitArray, щоможерозростатись до нескінченності, структура BitVector32 фіксованогорозміру - 32 біти. Як результат, BitVector32 більшефективний, ніжBitArray для не великихзначень. Ви можете розділитиекземплярBitVector32 на секції та ефективнозберігатибагатозначень.

Таблиця Є.4

Обмеження	Визначення
where T : <name of interface>	Тип аргумента має бути чимаєімплементуватизазначенийінтерфейс
where T : <name of base class>	Тип аргументу має бути чипоходитивідзазначеногокласу
where T : U	Тип аргументу має бути чипоходитивіднаданого типу аргументу U
where T : new()	Тип аргументу маєматипублічний за замовчуванням конструктор
where T : struct	Тип аргументу має бути значенням
where T : class	Тип аргументу має бути посиланням

Таблиця Є.5– Методи **і**нтерфейсаICollection<T>

Назва	Визначення
Add	Додамоелемент типу T до колекції.
Clear	Видаленнявсіхелементів з колекції.
Contains	Визначення того, чиколекціяміститьпевнізначення.
CopyTo	Копіюванняелементівколекції в масив.
Remove	Видаленняпевнихоб'єктів з колекції.

Таблиця Є.6– Властивості інтерфейсаICollection<T>

Назва	Визначення
Count	Надаєкількістьелементіввколекції.
IsReadOnly	Визначаєчиколекціятільки для зчитування.

Таблиця Є.7– Методи **і**нтерфейсаIList<T>

Назва	Визначення
Insert	Вставка елементіввколекцію з визначеніндексом.
RemoveAt	Видаленняелементів з колекції з визначеніндексом.

Таблиця Є.8– Властивості інтерфейсаIList<T>

Назва	Визначення
IndexOf	Визначаєпозиціюпевногоелемента в колекції.

Таблиця Є.9– Методи інтерфейсаIDictionary<TKey, TValue>

Назва	Визначення
Add	Додаванняелементів з зазначенимключем та значенням до колекції.
ContainsKey	Визначеннячиколекціявключає пари ключ-значення з певнимключем.
GetEnumerator	Повертаєперерахуванняоб'єктівKeyValuePair<TKey, TValue>.
Remove	Видаленняелементів за визначенимключем з колекції.
TryGetValue	Спробавстановитизначеннявихідного параметра до значення, щоасоціюється з відповіднимключем. Якщо ключ існує, метод повертаєtrue. Якщо ключ не існує, метод повертаєfalse і вихідний параметр не змінюється.

Таблиця Є.10– Властивості інтерфейсаIDictionary<TKey, TValue>

Назва	Визначення
Item	Отримуєчивстановлюєелемент в колекціювідповіднодоключа. Цявластивістьдозволяєвикористовуватипозначенняіндексатора, наприкладmyDictionary[myKey] = myValue.
Keys	ПовертаєключіколекціїякекземплярICollection<T>.
Values	ПовертаєзначенняколекціїякекземплярICollection<T>.

Таблиця Є.11– Методи **і**нтерфейсаIEnumerator<T>

Назва	Визначення
MoveNext	Перейти під час перерахування до наступногоелементу в колекції.
Reset	Встановитиперерахування до стартовоїпозиції, щознаходиться перед першим елементомколекції.

Таблиця Є.12– Властивості інтерфейсаIEnumerator<T>

Назва	Визначення
Current	Надаєелемент, на якому зараз знаходитьсявказівникпід час перерахування.

Завдання до лабораторної роботи № 7

1. Ввести рядки з файлу, записати їх в стек. Вивести рядки в файл в зворотному порядку.

2. Ввести число, занести його цифри в стек. Вивести число, у якого цифри йдуть в зворотному порядку.

3. Скласти два многочлена заданого степеня, якщо коефіцієнти многочленів зберігаються в об'єкті HashMap.

4. Створити стек з елементів каталогу.

5. Не використовуючи допоміжних об'єктів, переставити від’ємні елементи даного списку в кінець, а додатні - в початок цього списку.

6. Організувати обчислення у вигляді стека.

7. Виконати попарне підсумовування довільного скінченного ряду чисел наступним чином: на першому етапі підсумовуються попарно числа, які стоять поруч, на другому етапі підсумовуються результати першого етапу і т.д. до тих пір, поки не залишиться одне число.

8. Задати два стеки, змінити інформацію місцями.

9. Визначити клас Stack. Оголосити об'єкт класу. Ввести послідовність символів і вивести її в зворотному порядку.

10. Помножити два многочлена заданої степеня, якщо коефіцієнти многочленів зберігаються в списках.

11. Визначити клас Set на основі множини цілих чисел, n = розмір. Створити методи для визначення перетину і об'єднання множин.

12. Програма отримує N параметрів виклику (аргументи командного рядка). Ці параметри - елементи вектора. Будується масив типу double, а на базі цього масиву - об'єкт класу DoubleVector. Далі програма виводить в консоль значення елементів вектора у вигляді: Вектор: 2.3 5.0 7.3.

13. Списки (стеки) I (1..N) і U (1..N) містять результати N вимірювань струму і напруги на невідомому опорі R. Знайти наближене число R методом найменших квадратів

Додаток Ж
Матеріали до лабораторної роботи № 8

Завданнядолабораторної роботи № 8

1. Напишіть програму, в якій потрібно реалізувати ситуацію: при виклику примірника делегата кожен раз генерується чергове число в послідовності Фібоначчі.

2. Напишіть програму, в якій відображається вікно з двома кнопками. При натисканні на одній кнопці вікно закривається, а при натисканні на інший кнопці, розміри вікна збільшуються на 10%.

3. Напишіть програму, в якій оголошується делегат для методів з двома аргументами (символ і текст) і цілочисельним результатом. У головному класі необхідно описати два статичних методи. Один статичний метод результатом повертає кількість входжень символу (перший аргумент) в текстовий рядок (другий аргумент). Інший метод результатом повертає індекс першого входження символу (перший аргумент) в текстовий рядок (другий аргумент), або значення -1, якщо символ в текстовому рядку не зустрічається. У головному методі створити екземпляр делегата і за допомогою цього екземпляра викликати кожен з статичних методів.

4. Напишіть програму, в якій оголошується делегат для методів з символьним аргументом, які не повертають результат. Опишіть клас, в якому повинно бути символьне поле і метод, що дозволяє привласнити значення символьному полю об'єкта. У методу один символьний аргумент і метод не повертає результат. Створіть масив об'єктів даного класу. Створіть екземпляр делегата. У список викликів цього делегата необхідно додати посилання на метод (привласнює значення символьному полю) кожного об'єкта з масиву. Перевірити результат виклику такого екземпляра делегата.

5. Напишіть програму, в якій є статичний метод. Першим аргументом статичному методу передається цілочисельний масив. Другим аргументом статичному методу передається посилання на інший метод. У метода-аргумента повинен бути цілочисельний аргумент, який повинен повертати цілочисельний результат. Результатом статичний метод повертає цілочисельний масив. Елементи цього масиву обчислюються як результат виклику методу-аргументу, якщо йому передавати значення елементів з масиву-аргументу. Запропонуйте механізм перевірки функціональності даного статичного методу.