**Лабораторна робота № 4**

# Тема: Особливості ООП в С# та обробка виключних ситуацій

## Мета: Одержати практичні навички по створенню й використанню класів у мові C#, навчитися реалізовувати обробку виключних ситуацій

## **Теоретичні відомості**

### **Класи**

*Клас* є типом даних, визначеним користувачем. Він повинен являти собою одну логічну сутність, наприклад, бути моделлю реального об'єкта або процесу. *Елементами* класу є *дані* й *функції,* призначені для їхньої обробки.

*Опис класу* містить ключове слово class, за яким слідує його *ім'я,* а далі у фігурних дужках - *тіло* класу, тобто список його елементів. Крім того, для класу можна задати його базові класи (предки) і ряд необов'язкових атрибутів і специфікаторів, що визначають різні характеристики класу:

[ атрибути ] [ специфікатори ] **class** ім'я\_класу [ : предки ]

{

тіло-класу

}

*Ім'я\_класу* задається програмістом за загальними правилами С#. *Тіло класу* – це список описів його елементів, поміщений у фігурні дужки. Список може бути порожнім, якщо клас не містить жодного елемента.

*Специфікатори* визначають властивості класу, а також доступність класу для інших елементів програми.

У даній лабораторній роботі ми будемо розглядати класи, які описуються в просторі імен безпосередньо. Для таких класів допускаються тільки два специфікатори: public і internal. За замовчуванням, тобто якщо жоден специфікатор доступу не зазначений, мається на увазі специфікатор internal.

Клас є узагальненим поняттям, що визначає характеристики й поводження деякої множини конкретних об'єктів цього класу, що називаються *екземплярами* або *об'єктами класу.*

Об'єкти створюються явним або неявним чином, тобто або програмістом, або системою. Програміст створює екземпляр класу за допомогою операції new, наприклад:

Demo a = new Demo();// створення екземпляра класу Demo

Demo b = new Demo();//створення іншого екземпляра класу Demo

Клас відноситься до посилальних типів даних, пам'ять під які виділяється в купі. Таким чином, змінні a і b зберігають не самі об'єкти, а посилання на об'єкти, тобто їхні адреси. Якщо достатній для зберігання об'єкта обсяг пам'яті виділити не вдалося, операція new генерує виключення OutOfMemoryException.

Механізм виконання присвоювання однаковий для величин будь-якого типу, як посилального, так і значимого, однак результати розрізняються. При присвоюванні значень копіюються значення, а при присвоюванні посилань - посилання, тому після присвоювання одного об'єкта іншому ми отримаємо два посилання, що вказують на одну й ту саму область пам'яті.

Аналогічна ситуація з операцією перевірки на рівність. Величини значимого типу рівні, якщо рівні їхні значення. Величини посилального типу (екземпляри класу в цьому випадку) рівні, якщо вони посилаються на ті самі дані.

Для кожного об'єкта при його створенні в пам'яті виділяється окрема область, у якій зберігаються ці дані. Крім того, у класі можуть бути присутніми *статичні елементи,* які існують у єдиному екземплярі для всіх об'єктів класу. Часто статичні дані називають *даними класу,* а інші - *даними екземпляра.*

Функціональні елементи класу не тиражуються, тобто завжди зберігаються в єдиному екземплярі. Для роботи з даними класу використовуються *методи класу (статичні методи),* для роботи з даними екземпляра – *методи екземпляра,* або просто *методи.*

Поля й методи є основними елементами класу. Крім того, у класі можна задавати інші елементи. Нижче наведені всі можливі елементи класу:

*– Константи* класу зберігають незмінні значення, пов'язані із класом.

*– Поля* містять дані класу.

*– Методи* реалізують обчислення або інші дії, виконувані класом або екземпляром.

*– Властивості* визначають характеристики класу в сукупності зі способами їхнього завдання й одержання, тобто методами запису й читання.

*– Конструктори* реалізують дії по ініціалізації екземплярів або класу в цілому.

*– Деструктори* визначають дії, які необхідно виконати до того, як об'єкт буде знищений.

*– Індексатори* забезпечують можливість доступу до елементів класу за їхнім порядковим номером.

*– Операції* задають дії з об'єктами за допомогою знаків операцій.

*– Події* визначають повідомлення, які може генерувати клас.

*– Типи* – це типи даних, внутрішні стосовно класу.

### **Ключове слово this**

Кожний об'єкт містить свій екземпляр полів класу. Методи перебувають у пам'яті в єдиному екземплярі й використовуються всіма об'єктами спільно, тому необхідно забезпечити роботу методів нестатичних екземплярів з полями саме того об'єкта, для якого вони були викликані. Для цього в будь-який нестатичний метод автоматично передається прихований параметр this, у якому зберігається посилання на екземпляр, що викликав функцію.

У явному вигляді параметр this застосовується для того, щоб повернути з методу посилання на викликавший об'єкт, а також для ідентифікації поля у випадку, якщо його ім'я збігається з ім'ям параметра методу, наприклад:

class Demo

{

double у;

public Demo T() // метод повертає посилання на екземпляр

{

return this;

}

public void Sety( double у)

{

this.у = у; // полю у привласнюється значення параметра у

}

### **Конструктори**

*Конструктор* призначений для ініціалізації об'єкта. Він викликається автоматично при створенні об'єкта класу за допомогою операції new. Ім'я конструктора збігається з ім'ям класу. Нижче перераховані властивості конструкторів:

– Конструктор *не повертає значення,* навіть типу void.

– Клас може мати *декілька конструкторів* з різними параметрами для різних видів ініціалізації.

– Якщо програміст не вказав жодного конструктора або якісь поля не були ініціалізовані, полям значимих типів привласнюється нуль, полям посилальних типів - значення null.

– Конструктор, викликаний без параметрів, називається *конструктором за замовчуванням.*

Можна задавати початкові значення полів класу при описі класу. Це зручно в тому випадку, коли для всіх екземплярів класу початкові значення якогось поля однакові. Якщо ж при створенні об'єктів потрібно привласнювати полю різні значення, це варто робити в конструкторі. У наступному прикладі в клас Demo доданий конструктор, а поля зроблені закритими (непотрібні в цей момент елементи опущені). У програмі створюються два об'єкти з різними значеннями полів.

Лістинг 4.1. Клас із конструктором

using System;

namespace ConsoleApplication1

{

class Demo

{

public Demo(int a, double у) // конструктор з параметрами

{

this.a = a;

this.у = у;

}

publicdouble Gety() // метод одержання поля у

{

return у;

}

int a;

double у;

}

class Classl

{

staticvoid Main()

{

Demo a = new Demo(300, 0.002); // виклик конструктора

Console.WriteLine(a.Gety()); // результат: 0.002

Demo b = new Demo(1, 5.71); // виклик конструктора

Console.WriteLine(b.Gety()); // результат: 5.71

}

}

}

Часто буває зручно задати в класі *декілька конструкторів,* щоб забезпечити можливість ініціалізації об'єктів різними способами.

Всі конструктори повинні мати різні сигнатури.

Якщо один з конструкторів виконує які-небудь дії, а інший повинен робити те ж саме плюс ще що-небудь, зручно *викликати перший конструктор із другого.* Для цього використовується вже згадане ключове слово this в іншому контексті, наприклад:

class Demo

{

public Demo( int a ) // конструктор 1

{

this.a = a;

}

public Demo(int a, double у) : this( a ) // виклик конструктора 1

{

this.у = у;

}

...

}

Конструкція, що перебуває після двокрапки, називається *ініціалізатором,* тобто тим кодом, що виконується до початку виконання тіла конструктора.

Конструктор будь-якого класу, якщо не зазначений ініціалізатор, автоматично викликає конструктор свого предка. Це можна зробити і явно за допомогою ключового слова base, що позначає конструктор базового класу.

Конструктор базового класу викликається явно в тих випадках, коли йому потрібно передати параметри.

### **Властивості**

*Властивості* служать для організації доступу до полів класу. Як правило, властивість пов'язана із закритим полем класу й визначає методи його одержання й встановлення. Синтаксис властивості:

[ атрибути ] [ специфікатори ] тип ім'я\_властивості

{

[ [ атрибути ] [ специфікатори ] get код\_доступу ]

[ [ атрибути ] [ специфікатори ] set код\_доступу ]

}

Значення специфікаторів для властивостей і методів аналогічні. Найчастіше властивості оголошуються як відкриті (зі специфікатором public), оскільки вони входять в інтерфейс об'єкта.

*Код доступу* являє собою блоки операторів, які виконуються при одержанні (get) або встановленні (set) властивості. Може бути відсутньою або частина get, або set, але не обидві одночасно.

Якщо відсутня частина set, властивість доступна тільки для читання (read-only), якщо відсутня частина get, властивість доступна тільки для запису (write-only).

Специфікатори доступу для окремої частини повинні задавати або такий же, або більш обмежений доступ, ніж специфікатор доступу для властивості в цілому. Наприклад, якщо властивість описана як publiс, його частини можуть мати будь-який специфікатор доступу, а якщо властивість має доступ protected internal, його частини можуть оголошуватися як internal, protected або private.

Приклад опису властивостей:

public class Button: Control

{

private string caption; // закрите поле, з яким зв'язана властивість

public string Caption { // властивість

get { // спосіб одержання властивості

return caption;

}

set { // спосіб встановлення властивості

if (caption != value) {

caption = value;

}

}

}

...

}

Метод запису звичайно містить дії по перевірці допустимості встановлюваного значення, метод читання може містити, наприклад, підтримку лічильника звертань до поля.

У програмі властивість виглядає як поле класу, наприклад:

Button ok = new Button();

ok.Caption = "OK"; // викликається метод встановлення властивості

string s = ok.Caption; // викликається метод одержання властивості

При звертанні до властивості автоматично викликаються зазначені в ньому методи читання й встановлення.

Синтаксично читання й запис властивості виглядають майже як методи. Метод get повинен містити оператор return, що повертає вираз, для типу якого повинне існувати неявне перетворення до типу властивості. У методі set використовується параметр зі стандартним ім'ям value, що містить встановлюване значення.

Властивість може не зв'язуватися з полем. Фактично, вона описує один або два методи, які здійснюють деякі дії над даними того ж типу, що й властивість. На відміну від відкритих полів, *властивості забезпечують розділення між внутрішнім станом об'єкта і його інтерфейсом* і, таким чином, спрощують внесення змін у клас.

### **Приклад створення класу**

Лістинг 4.2. Клас Man

using System;

namespace ConsoleApplication1

{

class Man

{

public Man()

{

this.name = "John Doe";

this.height = 180;

this.age = 20;

}

public Man( string name ) : this()

{

this.name = name;

}

public Man( int height, int age, string name )

{

this.name = name;

this.height = height;

this.age = age;

}

publicstring GetName()

{

return name;

}

publicint GetHeight()

{

return height;

}

publicvoid SetHeight(int height)

{

this.height = height;

}

publicint Age

{

get

{

return age;

}

set

{

age = value;

}

}

publicvoid Passport()

{

Console.WriteLine("Ім'я: {0} \t Зріст = {1} Вік= {2} ", name, height, age );

}

// закриті поля

string name;

int height, age;

}

class Classl

{

staticvoid Main()

{

Man X = new Man();

X.Passport();

Man Ivan = new Man( "Іван" );

Ivan.Passport();

Man Masha = new Man( 201, 18, "Маша" );

Masha.Passport();

Ivan.Age = 21;

Ivan.Passport();

}

}

}

Результат роботи програми:

Ім'я: John Doe Зріст = 180 Вік = 20

Ім'я: Іван Зріст = 180 Вік = 20

Ім'я: Маша Зріст = 201 Вік = 18

Ім'я: Іван Зріст = 180 Вік = 21

У класі три закриті поля (name, height і age), чотири методи (GetName, GetHeight, SetHeight і Passport), одна властивість (Age) і три конструктори, що дозволяють задати при створенні об'єкта жодного, один або три параметри.

**Обробка виключних ситуацій**

**Виключна ситуація** (або виключення) – це ситуація, в результаті якої генерується помилка, а робота програми переривається.

Виняткові ситуації можуть виникати у процесі роботи будь-якої програми. Наприклад, користувач може розділити в калькуляторі 5 на 0 або відкрити за допомогою WinAMP-у текстовий файл. Якщо не враховувати подібні ситуації при розробці, то у кінцевого користувача будуть виникати зайві труднощі при користуванні з вашим програмним забезпеченням. Виникнення помилок неминуче. Запобігти небажаним наслідкам помилок можна тільки за допомогою правильного проектування інтерфейсу користувача.

У більшості мов програмування є можливість обробки виняткових ситуацій. І в кожній мові ця техніка своя. В .**NET** є теж своя техніка обробки виняткових ситуацій - **SEH** (*Structured Exception Handling*). Зручність даної техніки в тому, що вона єдина для всіх **.NET-мов**.

Ще однією особливістю виняткових ситуацій в .**NET** є те, що будь-яке виключення – це об'єкт, похідний від класу **System.Exception**. У даного класу є декілька дуже корисних властивостей, які допоможуть одержати докладну інформацію про виниклу ситуацію:

**Message** - властивість, що повертає опис виниклої помилки.

**Source** - властивість, що повертає ім'я об'єкта або додатка, в якому виникла помилка.

**StackTrace** - властивість, що повертає послідовність викликів, які призвели до помилки.

Для того, щоб виявляти виключення в C# застосовується наступна конструкція:

try

{

//Тут міститься код, що може викликати помилку (а може й не викликати)

}

catch ()

{

//Тут міститься код, що буде обробляти помилку

}

finally

{

//Тут міститься код, що буде виконаний незалежно

//від того, відбулася помилка чи ні

}

Дана конструкція складається із трьох блоків: **try**, **catch** і **finally**. Останній блок (*finally*) є необов'язковим. Зазвичай його використовують для коректного завершення програми. Наприклад, якщо програма працює з базою даних, і в процесі роботи відбулася помилка, то перш ніж вийти необхідно відключитися від бази даних, щоб не зіпсувати дані. Також є безліч інших випадків, коли потрібно використовувати блок **finally**.

**Приклад 4.1.** Обробка виключення, яке виникає при діленні на нуль в програмі, що виконує ділення **X** на **Y**.

static void Main(string[] args)

{

int x = 15;

int y = 5;

Console.WriteLine(x / y);

}

У результаті роботи програми на екран буде виведене число 3. А тепер привласнимо **Y** значення нуль. Якщо скомпілювати дану програму й запустити (не в **Visual Studio**, а самостійно), то на екран буде виведене повідомлення про помилку:

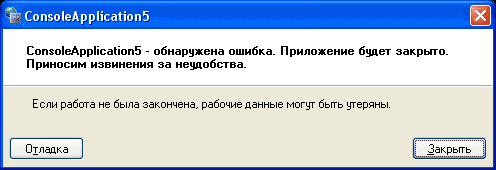


Рисунок 4.1 – Повідомлення про помилку

Щоб користувач вашої програми не побачив подібне, мало змістовне для нього, повідомлення, потрібно обробити цю виняткову ситуацію. У просторі імен **System** визначена множина готових виключень на різні помилки. Їх потрібно знати й користуватися ними. Можна також створювати й свої типи виключень, коли не одне з наявних не підходить для поставленої задачі. Ділення на 0 поширена помилка, тому для неї є своє виключення, що має назву **DivideByZeroException**. Назви інших виключень можна подивитися в довідці **Visual Studio**. Скористаємося цим виключенням й конструкцією **try/catch**, щоб обробити появу даної помилки.

static void Main(string[] args)

{

int x = 15;

int y = 0;

try

{

Console.WriteLine(x / y);

}

catch (DivideByZeroException e)

{

Console.WriteLine("Message: " + e.Message);

Console.WriteLine("Source: " + e.Source);

Console.WriteLine("StackTrace: " + e.StackTrace);

}

Console.ReadLine();

}

Тепер "небезпечний" код розміщено у блоці **try**, після чого в блоці catch помилка буде "піймана". Оскільки відомо про те, яка помилка виникне, було явно вказано її тип (*DivideByZeroException*). Після чого були виведені дані про цю помилку на консоль. От що в нас вийшло в процесі роботи:

Результат работы

Рисунок 4.2 – Результат роботи програми з прикладу 8.1 після обробки виключення

При цьому програма не "вилітає", а продовжує працювати. Інформація, отримана із властивостей **Message**, **Source** і **StackTrace** потрібна тільки при налагодженні програми. Зрозуміло, що цю інформацію знати користувачеві зовсім не обов'язково. Для нього можна вивести щось на зразок наступної фрази: "На нуль ділити не можна!".

У наведеному прикладі оброблено всього лише одну помилку (ділення на нуль). Звичайно ж, можна обробляти й декілька помилок відразу, для цього потрібно декілька разів писати блок **catch** з різними типами помилок:

try

{

//Тут міститься код, що може викликати помилку (а може й не викликати)

}

catch ()

{

//Тут міститься код, що буде обробляти помилку 1

}

catch ()

{

//Тут міститься код, що буде обробляти помилку 2

}

finally

{

//Тут міститься код, що буде виконаний незалежно

//від того, відбулася чи помилка ні

}

Цілком може виникнути ситуація, коли невідомо яка помилка виникне та якого типу вона буде. Тоді можна явно не писати тип помилки. Але при цьому не можна буде довідатися значення властивостей **Message**, **Source** і **StackTrace**:

try

{

Console.WriteLine(x / y);

}

catch

{

Console.WriteLine("Здається щось трапилося!");

}

Розглянемо як самостійно створювати свої типи помилок.

**Приклад 4.2.** Для даного прикладу створимо клас **Teacher** та метод **GiveMoney()** (видати зарплату). Даний метод одержує єдиний параметр - розмір зарплати в гривнях. Але що якщо значення цього параметра буде дорівнювати негативному числу? Це і є виняткова ситуація. (Насправді для даного типу помилки передбачене виключення **IndexOutOfRangeException**, але припустимо, що ми цього не знали). Нам необхідно перевіряти значення вхідного параметра і якщо воно менше нуля викликати виключення. Цей підхід є найбільш вірним. Є й інші (невірні) рішення: ви можете взагалі не нараховувати зарплату якщо параметр негативний і не сповіщати про це користувача (недоліки такого методу, думаю, очевидні), ви можете виводити повідомлення про помилку прямо в методі **GiveMoney()** (Наприклад, за допомогою **Console.WriteLine()**), але від цього ваш клас втрачає гнучкість, тому що прив’язується до роботи з консоллю. Розглянемо написання "правильного" методу **GiveMoney()**:

public class Teacher : Worker

{

public override void GiveMoney(int money)

{

if (money < 0)

{

throw new NoMoneyException();

}

...

}

}

Перевіряємо значення зарплати і якщо воно менше нуля викликаємо за допомогою команди **throw** помилку типу **NoMoneyException**. Зрозуміло, що такого типу помилки не існує, ми її придумали самі, тому потрібно такий тип помилки зробити. Робиться це за допомогою класу, що успадковується від **System.Exception**. Для прикладу достатньо перевизначити метод **Message**, щоб він виводив потрібний нам тип помилки:

public class NoMoneyException : System.Exception

{

public NoMoneyException(){}

public override string Message

{

get

{

return "Невірне значення заробітної плати";

}

}

}

Помітьте, що у властивості **Message** є тільки **get** метод, оскільки ця властивість тільки для читання. Тепер у нас є свій тип помилки **NoMoneyException**. Настав час перевірити, як він працює:

static void Main(string[] args)

{

Teacher x = new Teacher();

try

{

x.GiveMoney(-100);

}

catch (NoMoneyException e)

{

Console.WriteLine(e.Message);

}

Console.ReadLine();

}

Запустивши додаток, ви побачите на консолі очікуваний напис "Невірне значення заробітної плати". Наше виключення працює.

**Завдання до лабораторної роботи:**

Описати клас, заданий за варіантом, та реалізувати роботу з ним. Кожний розроблювальний клас повинен, як правило, містити наступні елементи: приховані поля, конструктори з параметрами й без параметрів, методи, властивості. Методи й властивості повинні забезпечувати несуперечливий, повний, мінімальний і зручний інтерфейс класу. У програмі повинна виконуватися перевірка всіх розроблених елементів класу.

**Варіант 1**

Описати клас, який реалізує десятковий лічильник, що може збільшувати або зменшувати своє значення на одиницю в заданому діапазоні. Передбачити ініціалізацію лічильника значеннями за замовчуванням і довільними значеннями. Лічильник має два методи: збільшення й зменшення, - і властивість, що дозволяє одержати його поточний стан. При виході за межі діапазону викидаються виключення.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 2**

Описати клас, що реалізує шістандцятковий лічильник, що може збільшувати або зменшувати своє значення на одиницю в заданому діапазоні. Передбачити ініціалізацію лічильника значеннями за замовчуванням і довільними значеннями. Лічильник має два методи: збільшення й зменшення, - і властивість, що дозволяє одержати його поточний стан. При виході за межі діапазону викидаються виключення.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 3**

Описати клас, що представляє трикутник. Передбачити методи для створення об'єктів, переміщення на площини, зміни розмірів і обертання на заданий кут. Описати властивості для одержання стану об'єкта. При неможливості побудови трикутника викидається виключення.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 4**

Побудувати опис класу, що містить інформацію про поштову адресу організації. Передбачити можливість роздільної зміни складових частин адреси й перевірки допустимості значень, що вводяться. У випадку неприпустимих значень полів викидаються виключення.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 5**

Скласти опис класу для представлення комплексних чисел. Забезпечити виконання операцій додавання, віднімання й множення комплексних чисел.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 6**

Скласти опис класу для вектора, заданого координатами його кінців у тривимірному просторі. Забезпечити операції додавання й віднімання векторів з одержанням нового вектора (суми або різниці), обчислення скалярного добутку двох векторів, довжини вектора, косинуса кута між векторами.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 7**

Скласти опис класу прямокутників зі сторонами, паралельними осям координат. Передбачити можливість переміщення прямокутників на площини, зміну розмірів, побудова найменшого прямокутника, що містить два заданих прямокутники, і прямокутника, що є спільною частиною (перетином) двох прямокутників.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 8**

Скласти опис класу для представлення дати. Передбачити можливості встановлення дати й зміни її окремих полів (рік, місяць, день) з перевіркою допустимості значень, що вводяться. У випадку неприпустимих значень полів викидаються виключення. Створити методи зміни дати на задану кількість днів, місяців і років.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 9**

Скласти опис класу для представлення часу. Передбачити можливості встановлення часу й зміни його окремих полів (година, хвилина, секунда) з перевіркою допустимості значень, що вводяться. У випадку неприпустимих значень полів викидаються виключення. Створити методи зміни часу на задану кількість годин, хвилин і секунд.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 10**

Скласти опис класу багаточлена виду ах2 +bх + с. Передбачити методи, що реалізують:

– обчислення значення багаточлена для заданого аргументу;

– операцію додавання, віднімання й множення багаточленів з одержанням нового об'єкта-багаточлена;

– виведення на екран опису багаточлена.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 11**

Описати клас, що представляє трикутник. Передбачити методи для створення об'єктів, обчислення площі, периметра й точки перетину медіан. Описати властивості для одержання стану об'єкта. При неможливості побудови трикутника викидається виключення.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 12**

Описати клас, що представляє коло. Передбачити методи для створення об'єктів, обчислення площі кола, довжини окружності й перевірки влучення заданої точки усередину кола. Описати властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 13**

Описати клас для роботи з рядком, що дозволяє зберігати тільки двійкове число й виконувати з ним арифметичні операції. Передбачити ініціалізацію з перевіркою допустимості значень. У випадку неприпустимих значень викидаються виключення.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 14**

Описати клас дробних (раціональних) чисел, що є відношенням двох цілих чисел. Передбачити методи додавання, віднімання, множення й ділення дробів.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 15**

Описати клас «файл», що містить відомості про ім'я, дату створення й довжині файлу. Передбачити ініціалізацію з перевіркою допустимості значень полів. У випадку неприпустимих значень полів викидаються виключення. Описати метод додавання інформації в кінець файлу й властивості для одержання стану файлу.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 16**

Описати клас «кімната», що містить відомості про метраж, висоту стель і кількість вікон. Передбачити ініціалізацію з перевіркою допустимості значень полів. У випадку неприпустимих значень полів викидаються виключення. Описати методи обчислення площі й обсягу кімнати й властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 17**

Описати клас, що представляє нелінійне рівняння виду:

ах - cos(x) = 0.

Описати метод, що обчислює рішення цього рівняння на заданому інтервалі методом ділення навпіл і викидаючий виключення у випадку відсутності кореня. Описати властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 18**

Описати клас, що представляє квадратне рівняння виду:

ах2 +bх + с = 0.

Описати метод, що обчислює рішення цього рівняння й викидає виключення у випадку відсутності коренів. Описати властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класу.

**Варіант 19**

Описати клас «процесор», що містить відомості про марку, тактову частоту, обсяг кешу й вартість. Передбачити ініціалізацію з перевіркою допустимості значень полів. У випадку неприпустимих значень полів викидаються виключення. Описати властивості для одержання стану об'єкта.

Описати клас «материнська плата», що включає клас «процесор» і обсяг встановленої оперативної пам'яті. Передбачити ініціалізацію з перевіркою допустимості значень поля обсягу пам'яті. У випадку неприпустимих значень поля викидається виключення. Описати властивості для одержання стану об'єкта.

Написати програму, що демонструє всі розроблені елементи класів.

**Варіант 20**

Описати клас «кольорова точка». Для точки задаються координати й колір. Колір описується за допомогою трьох складових (червоний, зелений, синій). Передбачити різні методи ініціалізації об'єкта з перевіркою допустимості значень. Припустимим діапазоном для кожної складової є [0, 255]. У випадку неприпустимих значень полів викидаються виключення. Описати властивості для одержання стану об'єкта й метод зміни кольору.

Написати програму, що демонструє всі розроблені елементи класу.

## ***Контрольні питання:***

1. Як визначити клас?
2. Як створити екземпляр класу?
3. Які елементи можуть входити до складу класу?
4. Як звернутися до елемента класу?
5. Для чого використовується ключове слово this?
6. Для чого застосовуються конструктори?
7. Як визначається, який з конструкторів буде викликаний при створенні об'єкта?
8. Що таке статичні методи класу?
9. Що таке виключна ситуація?
10. Які засоби обробки виключних ситуацій є в мові програмування С#?
11. Які оператори застосовуються для обробки виключних ситуацій?
12. За допомогою яких властивостей можна отримати інформацію про виключну ситуацію?
13. Як можна самостійно створювати типи помилок?