**Міністерство освіти та науки України**

**Державний вищий навчальний заклад “Київський національний економічний університет ім. Вадима Гетьмана”**

**Кафедра інформаційних систем в економіці**

**Дисципліна “Адміністрування та програмування баз даних в штучному інтелекті”**

**ЗВІТ**

**з лабораторної роботи №3**

**«С# Professional»**

**Підготував:**

студент 5 курсу гр. ІШІ-501

спеціальності «8і09»

Донєв Данило Романович

**Прийняла:**

доц. Добролюбова М.В.

Київ-2023

***ВІДПОВІДІ НА ПИТАННЯ***

# Що таке колекція? Перелічіть основні колекції, які не чутливі до регістру. Як можна виправити дану проблему?

Колекція – це клас, призначений для групування зв’язаних об’єктів, керування ними та обробки їх в циклах.Колекції є важливим інструментом програміста, але рішення про їх застосування не завжди є очевидним.

Колекції слід застосовувати, якщо:

* окремі елементи застосовуються для однакових цілей і є однаково важливими;
* на момент компіляції число елементів не відомо або не зафіксовано;
* необхідна підтримка операції перебору всіх елементів;
* необхідна підтримка впорядковування елементів;
* необхідно використовувати елементи з бібліотеки, від якої очікується наявність типу колекції.
* В інфраструктурі .Net більшість об'єктів-колекцій реалізує інтерфейс IEnumerable, а об'єкти-ітератори називаються «перелічувачами» (enumerators) і реалізують інтерфейс IEnumerator.

У .NET є спеціальні версії колекцій, які не чутливі до регістру.

* StringDictionary: це колекція пар ключ-значення, де ключі, фактично, не чутливі до регістру.
* NameValueCollection: ця колекція також не чутлива до регістру.
* Hashtable та Dictionary<TKey, TValue> може бути конфігуровані так, щоб бути нечутливими до регістру, передаванням в конструктор провайдера порівняння строк, що не чутливий до регістру.

Зокрема, можна виористати StringComparer.OrdinalIgnoreCase або StringComparer.CurrentCultureIgnoreCase як параметр для Dictionary<TKey, TValue>:

var dict = new Dictionary<string, int>(StringComparer.OrdinalIgnoreCase);

За замовчуванням, колекції StringDictionary і NameValueCollection в .NET не враховують регістр ключів. Щоб змінити цю поведінку, доведеться створити власну колекцію, яка враховує регістр.

# Що таке регулярні вирази і для чого вони застосовуються?

Регулярні вирази – це незамінний інструмент для багатьох додатків, в яких ведеться робота з рядками або аналіз об'ємних блоків тексту. Порівнюючи з текстом регулярні вирази, що складаються з чисел, букв в певному регістрі або шістнадцяткових рядків, можна приймати рішення, що впливають на роботу програми.

Регулярні вирази – це формальна мова пошуку та здійснення маніпуляцій з підрядками в тексті, що заснована на використанні метасимволів.

Основа обробки тексту за допомогою регулярних виразів – це підсистема обробки регулярних виразів, представлена в платформі .NET Framework об'єктом System.Text.RegularExpnessiоns.Regex. Тобто клас Regex надає можливості по роботі з регулярними виразами. Мінімальний набір відомостей, який потрібно надати підсистемі обробки регулярних виразів для обробки тексту, зводиться до: створення шаблону регулярного виразу і застосування його до аналізуємого тексту.

# Що таке XML? Де і для чого використовується XML?

XML (Extensible Markup Language) – розширювана мова розмітки

XML – текстовий формат, призначений для зберігання структурованих даних (замість існуючих файлів баз даних), для обміну інформацією між програмами, а також для створення на його основі більш спеціалізованих мов розмітки (наприклад, XHTML).

XML – це описана в текстовому форматі ієрархічна структура, призначена для зберігання будь-яких даних. Візуально структура може бути представлена як дерево елементів. Елементи XML описуються тегами.

XML використовується в багатьох областях та ситуаціях, ось лише кілька прикладів:

Конфігурація програмного забезпечення: багато програмних систем використовують XML для зберігання конфігураційних даних. Наприклад, .NET використовує XML-файли для веб-конфігурації.

Обмін даними в інтернеті: XML часто використовується для обміну даними між системами через HTTP.

Формати документів: формати документів, такі як Open XML (використовується в Microsoft Office) та деякі формати e-book використовують XML для зберігання даних.

Серіалізація даних: XML може використовуватися для серіалізації об'єктів в пам'яті в стабільний формат, який можна зберегти або передати.

Основний принцип XML полягає в тому, що він дозволяє вам визначати власні теги та створювати документи, які позначають особливості ваших даних. За цим простим принципом криється неймовірно потужний та гнучкий інструмент для позначення інформації.

# Що таке атрибут? Як створити атрибут користувача? Перерахуйте всі необхідні кроки і обмеження. У чому відмінність між позиційними та іменованими параметрами атрибутів? До чого застосовуються атрибути?

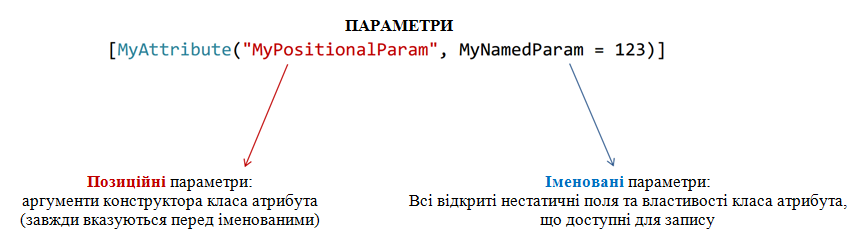
Атрибути в C# - це класи, які дозволяють надати додаткову метаінформацію про елементи програми, такі як типи, поля, методи, параметри тощо. Інформація про атрибути зберігається в метаданих збірки, і її можна отримати використовуючи рефлексію.

**Правила створення атрибута користувача:**

1. Ім’я атрибута повинно містити суфікс Attribute (дане правило є рекомендацією, але його краще притримуватись).

2. Клас-атрибут повинен наслідуватися від системного класу System.Attribute.

3. Клас-атрибут повинен бути декорований атрибутом [AttributeUsageAttribute].



**Позиційні** параметри - це параметри, що передаються через конструктор атрибуту, і вони є обов'язковими під час створення атрибуту.

**Іменовані** параметри - це властивості або поля, які можна налаштувати після створення атрибуту. Вони не обов'язкові і можуть бути пропущені

Атрибути використовуються для різних цілей: вони можуть використовуватися для визначення метаданих, контролю виконання коду, як маркери в коді для рефлексії або інструментів аналізу коду тощо.

# Що таке час життя об'єкта, керована купа, збирач сміття? Що таке покоління GC і які покоління ви знаєте?

Час життя об'єкта - це період від моменту створення об'єкта в пам'яті до моменту його знищення збиральником сміття (Garbage Collector, GC). Коли об'єкт більше не має жодних посилань на нього і не може бути досягнутий з коду, він стає "мертвим" об'єктом, і збиральник сміття може його знищити.

Керована купа - це область в пам'яті, головним чином використовується .NET для зберігання і управління об'єктами. Коли ви створюєте новий об'єкт (наприклад, викликом new в C#), пам'ять для об'єкта виділяється у керованій купі.

Збиральник сміття (Garbage Collector, GC) - це автоматичний управляючий механізм у .NET, який виділяє пам'ять для нових об'єктів та звільняє пам'ять від об'єктів, що більше не використовується (тобто в "мертвих" об'єктів).

Покоління збірника сміття - це механізм, за яким збиральник сміття (GC) групує об'єкти в керованій купі. В .NET є три покоління: 0, 1 та 2.

Покоління 0 - місце, куди належать всі нові об'єкти. Коли покоління 0 заповнюється, GC проводить збір сміття в поколінні 0.

Покоління 1 - використовується як буфер між короткочасними об'єктами (в поколінні 0) та довгочасними об'єктами (в поколінні 2).

Покоління 2 - місце для довгочасних об'єктів. Робота збірника сміття з поколінням 2 відбувається рідше, ніж з поколіннями 0 та 1.

Механізм поколінь є надзвичайно ефективним, оскільки зазвичай об'єкти короткого терміну створюються та знищуються частіше, ніж об'єкти, які живуть протягом більш тривалого часу. Таким чином, типові заходи збирання сміття включають перевірку лише об'єктів в поколінні 0, а більш тривалі (зазвичай більш ресурсомісткі) заходи, що включають об'єкти в поколінні 2, запускаються менш часто

# У чому полягає різниця між синхронним і асинхронним викликом методів?

В синхронному виклику метода кожна операція виконується одна за одною. Це означає, що ви повинні дочекатися завершення однієї операції, перш ніж виконувати іншу. Якщо якась операція блокується (наприклад, очікує на завантаження даних з файлу або через мережу), виконання вашого коду продовжується тільки після того, як операція була завершена.

var result = LongRunningMethod(); *// Execution pauses until LongRunningMethod has completed*Console.WriteLine(result);

Асинхронний виклик методу, в свою чергу, дозволяє вашому коду продовжуватися без очікування завершення операцій. Замість цього, коли викликається асинхронний метод, він повертає задачу (об'єкт Task), яка представляє майбутні результати. Ви можете очікувати цю задачу в асинхронному методі з допомогою ключового слова await, яке автоматично "випустить" потік, на якому працює ваш код, дозволяючи йому виконувати іншу роботу.

var task = LongRunningMethodAsync(); *// Execution continues without waiting for LongRunningMethodAsync to complete  
// Do some other work...*var result = await task; *// Wait for the task to complete, but without blocking the executing thread*Console.WriteLine(result);

Асинхронна модель дозволяє писати код, який може "робити багато речей одночасно", не збільшуючи кількість фізично використовуваних потоків. Це особливо корисно в сценаріях введення/виведення, де велика кількість операцій може заблокувати виконання, чекаючи завершення вводу/виводу.

***Завдання 1***

Створіть власну збірку користувача, яка буде використовуватися для роботи з конвертером температури.

Створіть програму, в якій надайте користувачеві доступ до збірки з Завдання 1. Реалізуйте використання методу конвертації значення температури із шкали Цельсія в шкалу Фаренгейта. Виконуючи завдання використовуйте тільки рефлексію.

ПОСИЛАННЯ НА GITHUB

<https://github.com/doniev-danylo/kneu_cs/tree/main/lab3>

КОД ПРОГРАМИ

using System.Reflection;  
  
namespace ReflectionApp  
{  
 class Lab3Task1  
 {  
 static string PathResolver(string filePath)  
 {  
 var projectDirectory = Directory.GetParent(Environment.CurrentDirectory).Parent?.Parent?.FullName;  
 var filePathFromRoot = Path.Combine(projectDirectory, filePath);  
 Console.WriteLine(filePath);  
  
 if (File.Exists(filePathFromRoot))  
 {  
 return filePathFromRoot;  
 }  
  
 return filePath;  
 }  
  
 static void Main(string[] args)  
 {  
 Assembly assembly =  
 Assembly.LoadFrom(PathResolver("TemperatureConverter.dll"));  
 Type type = assembly.GetType("TemperatureConverter.TemperatureConverter");  
 MethodInfo method = type.GetMethod("ConvertCelsiusToFahrenheit");  
  
 double temperatureInCelsius = 20.0;  
 object result = method.Invoke(null, new object[] { temperatureInCelsius });  
  
 Console.WriteLine($"Температура у Цельсіях: {temperatureInCelsius}");  
 Console.WriteLine($"Температура у Фаренгейтах: {result}");  
 }  
 }  
}

РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ

