**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

Швиденко Дмитро Юрійович

**Прийняла:**

доц. Добролюбова М.В.

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

Колекція- це клас, призначений для групування зв’язних об’єктів, керування ними та обробки їх в циклах.

Колекції є важливим елементом програміста, але рішення про їх застосування не завжди є очевидним.

Колекція - це структура даних, призначена для збереження та організації групи об'єктів. Колекції дозволяють зручно керувати даними, включаючи додавання, видалення, пошук та інші операції над об'єктами.

Основні колекції, які не чутливі до регістру (case-insensitive), це:

* List<T>: Список, який дозволяє зберігати об'єкти в порядку їх додавання.
* Dictionary<TKey, TValue>: Словник, де кожен ключ пов'язаний зі значенням.
* HashSet<T>: Множина, яка зберігає унікальні значення.

Якщо виникає проблема з регістром при пошуку в колекціях, ви можете використовувати порівняння, яке не чутливе до регістру. Наприклад, ви можете використовувати методи StringComparer.OrdinalIgnoreCase для рядків або створити власний компаратор для об'єктів, який виконує порівняння без урахування регістру.

1. **Що таке регулярні вирази і для чого вони застосовуються?**

Регулярні вирази - це потужний інструмент для роботи з рядками в текстовому пошуку та обробці. Вони визначають шаблони символів, які використовуються для пошуку та витягування текстової інформації. Регулярні вирази дозволяють вам виконувати наступні завдання:

Пошук: За допомогою регулярних виразів можна шукати в тексті конкретні шаблони, слова, числа тощо.

Валідація: Ви можете перевіряти, чи відповідає заданий текст певному формату, наприклад, чи є електронна адреса правильною, чи номер телефону має правильний формат.

Заміна: Регулярні вирази дозволяють вам замінювати один текст іншим відповідно до заданого шаблону.

Вилучення даних: Ви можете витягувати певну інформацію із тексту, наприклад, вилучати номери телефонів, електронні адреси, URL-адреси тощо.

1. **Що таке XML? Де і для чого використовується XML?**

XML (eXtensible Markup Language) - це розширювана мова розмітки, яка використовується для збереження та обміну структурованими даними в текстовому форматі. XML базується на тезаурусі мови маркування HTML, але не обмежується певними тегами і дозволяє визначати власні теги та структуру даних.

XML використовується для різних цілей, таких як:

Збереження даних: XML дозволяє структуровано зберігати дані в текстовому форматі. Використовується для збереження конфігурацій, параметрів, налаштувань, тощо.

Обмін даними: XML є популярним форматом для обміну даними між різними системами, програмами та платформами. Він дозволяє легко інтерпретувати та обробляти дані, навіть якщо їх створює одна система, анотації для іншої.

Конфігурація: XML використовується для збереження налаштувань та конфігураційних файлів програм.

Веб-сервіси: XML використовується як формат обміну даними для веб-сервісів, таких як SOAP (Simple Object Access Protocol) та REST (Representational State Transfer).

Документація: XML може бути використаний для створення структурованої документації, такої як DocBook або DITA.

Основна перевага XML полягає в тому, що він є читабельним та розширюваним форматом для обробки даних. Він дозволяє створювати власні теги та структури даних, що робить його дуже гнучким і використовується в широкому спектрі додатків і областей, де необхідно працювати зі структурованими даними.

1. **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?**

Атрибут - це метадані, які можна прикріпити до елементів програми, таких як класи, методи, властивості, поля тощо. Атрибути дозволяють робити додаткові декларації або налаштування для цих елементів. Вони розширюють інформацію про програмний код і допомагають використовувати деякі вбудовані функції інструментів або створювати власні.

Для створення атрибуту користувача в C# вам потрібно виконати такі кроки:

Створити власний клас атрибуту, який повинен успадковувати System.Attribute.

Визначити, які параметри атрибуту вам потрібні. Ви можете використовувати публічні властивості або поля для цього класу.

Використовуйте цей атрибут, прикріпляючи його до елементів програми, які ви хочете маркувати атрибутами. Це може бути зроблено за допомогою круглих дужок [].

Використовуйте рефлексію для отримання інформації про атрибути в програмі.

Різниця між позиційними та іменованими параметрами атрибутів полягає в способі передачі параметрів. Позиційні параметри передаються у тому порядку, в якому вони визначені в конструкторі атрибуту, а іменовані параметри передаються з вказівкою імені параметра.

Атрибути використовуються для маркування коду і передачі додаткової інформації для компілятора або інших інструментів. Наприклад, вони можуть використовуватися для позначення коду, який потребує спеціального опрацювання або для генерації документації, перевірки валідації, тощо.

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

Час життя об'єкта (Object Lifetime) - це період, протягом якого об'єкт існує в пам'яті комп'ютера. Об'єкти створюються, використовуються і врешті-решт знищуються (видаляються з пам'яті). Мови програмування, такі як C#, автоматично керують часом життя об'єктів за допомогою збирача сміття (Garbage Collector, GC).

Збирач сміття (GC) - це компонент в середовищі виконання програми, який автоматично видаляє об'єкти, які більше не використовуються, для вивільнення пам'яті і запобігання витокам пам'яті. GC відслідковує посилання на об'єкти і періодично перевіряє, чи мають ці об'єкти активні посилання.

GC використовує концепцію поколінь для керування часом життя об'єктів. Є три основних покоління:

Generation 0 (Gen 0): Це наймолодше покоління, куди розміщуються нові об'єкти. Більшість об'єктів швидко стають сміттям і видаляються.

Generation 1 (Gen 1): Це покоління, куди потрапляють об'єкти, які вижили після зборки сміття в Gen 0.

Generation 2 (Gen 2): Це найстарше покоління, куди потрапляють об'єкти, які вижили кілька зборок сміття.

Збирач сміття зазвичай фокусується на зборці Gen 0 і Gen 1, оскільки більшість об'єктів швидко стають сміттям. Зборка Gen 2 відбувається рідше. Ця стратегія дозволяє зменшити витрати на збірку сміття та підтримувати ефективну роботу програм.

Збирач сміття автоматично визначає, які об'єкти вважати сміттям і видаляє їх, щоб вивільнити пам'ять для подальшого використання. Ця система допомагає програмістам уникнути багатьох проблем, пов'язаних з керуванням пам'яттю вручну.

1. **У чому полягає різниця між синхронним і асинхронним викликом методів?**

Різниця між синхронним і асинхронним викликом методів полягає в тому, як вони керують виконанням коду:

Синхронний метод (Synchronous):

Виконується послідовно, крок за кроком.

Блокує виконання поточного потоку до завершення методу.

Чекає на результат виклику, перед тим як продовжити виконання інших операцій.

Асинхронний метод (Asynchronous):

Виконується паралельно з іншими операціями без блокування поточного потоку.

Дозволяє продовжувати виконання інших операцій під час виконання асинхронного методу.

Результат асинхронного методу може бути отриманий пізніше, коли він завершиться.

Основні переваги асинхронного програмування полягають у здатності програми продовжувати роботу під час виконання довгих або блокуючих операцій. Це дозволяє покращити реагування програми і зменшити блокування потоків.

У мові C# асинхронні методи зазвичай позначаються ключовим словом async, і вони повинні повертати об'єкт типу Task або Task<TResult>. Вони дозволяють програмі розподілити важкі завдання на декілька потоків та виконувати їх паралельно.

Проте варто враховувати, що неправильне використання асинхронності може призвести до проблем з управлінням потоками та ресурсами, тому важливо ретельно розглядати використання асинхронності в програмі.