**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ “КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря**

**Сікорського”**

**Факультет інформатики та обчислювальної техніки Кафедра інформаційних систем та технологій (ІСТ)**

Звіт про виконання лабораторної роботи №1

З дисципліни

“Сучасні технології розробки WEB-застосувань на платформі .NET (3 частина)” тема: “Узагальнені типи (Generic) з підтримкою подій.

Колекції”

Перевірив: Виконав:

студент 4 курсу, гр. ІС-03

пос. Вовк Є. А. Козак Іван

2023 р.

**Мета:** Навчитися проектувати та реалізовувати узагальнені типи, а також типи з підтримкою подій.

# Завдання

1. Розробити клас власної узагальненої колекції, використовуючи стандартні інтерфейси колекцій із бібліотек System.Collections та System.Collections.Generic. Стандартні колекції при розробці власної не застосовувати. Для колекції передбачити методи внесення даних будь-якого типу, видалення, пошуку та ін. (відповідно до типу колекції).
2. Додати до класу власної узагальненої колекції підтримку подій та обробку виключних ситуацій.
3. Опис класу колекції та всіх необхідних для роботи з колекцією типів зберегти у динамічній бібліотеці.
4. Створити консольний додаток, в якому продемонструвати використання розробленої власної колекції, підписку на події колекції.

**Варіант:**  8, реалізація кільцевого списку.

# Контрольні запитання:

1. **Дайте визначення колекції. Наведіть типи колекцій**

Колекція представляє собою структуру даних, яка призначена для організації та управління об'єктами. Вона забезпечує зручні операції додавання, вилучення та редагування даних в групах елементів. Типи колекцій можуть відрізнятися в залежності від мови програмування, але основні види включають::

* + Масиви (Arrays): Це фіксована послідовність елементів одного типу даних, доступ до яких здійснюється за допомогою індексів.
  + Списки (Lists): Це динамічна послідовність елементів, яка може змінюватися в розмірі, а елементи можуть бути різних типів.
  + Множини (Sets): Колекція унікальних елементів без визначеного порядку, що виключає наявність дублікатів.
  + Словники (Dictionaries або Maps): Це колекція пар ключ- значення, яка дозволяє швидкий доступ до значень за

допомогою ключа.

* + Черги (Queues): Це послідовність елементів, додавання яких відбувається з одного кінця (кінець черги), а вилучення - з іншого (початок черги).
  + Стеки (Stacks): Схожі на черги, але елементи вилучаються з того ж кінця, що і додавалися (вершина стеку).

# Наведіть основні інтерфейси, які успадковуються колекціями, та їх призначення.

Основні інтерфейси для колекцій знаходяться в просторі імен System.Collections та System.Collections.Generic. Основні інтерфейси включають:

* + IEnumerable (Інтерфейс для перебору):

Призначення: Цей інтерфейс визначає метод GetEnumerator(), який повертає об'єкт, що реалізує інтерфейс IEnumerator. Він дозволяє перебирати елементи колекції.

* + ICollection (Інтерфейс для колекцій):

Призначення: Включає в себе методи для визначення розміру колекції, копіювання елементів у масив, перевірки наявності елемента та інші операції.

* + IList (Інтерфейс для списків):

Призначення: Розширює ICollection та включає методи для вставки, вилучення та отримання елемента за індексом.

* + IEnumerable<T> (Інтерфейс для перебору з типізованими даними): Призначення: Типізована версія IEnumerable для роботи з конкретним типом даних.
  + ICollection<T> (Інтерфейс для колекцій з типізованими даними): Призначення: Типізована версія ICollection для роботи з конкретним типом даних.
  + IList<T> (Інтерфейс для списків з типізованими даними):

Призначення: Типізована версія IList для роботи з конкретним типом даних.

* + IDictionary<TKey, TValue> (Інтерфейс для словників):

Призначення: Визначає операції для роботи зі словниками (колекцією пар ключ-значення).

# Поясніть призначення паттерн «Ітератор» та його реалізацію в

**.Net**

Паттерн "Ітератор" є одним з паттернів проектування, який використовується для представлення об'єктів, які мають послідовність елементів, і надає уніфікований спосіб отримання доступу до цих елементів без розкриття внутрішньої структури цих об'єктів. Основна ідея полягає в тому, щоб визначити інтерфейс для послідовного перегляду елементів об'єкту, а потім забезпечити конкретні реалізації цього інтерфейсу для конкретних класів.

У .NET цей паттерн реалізований за допомогою інтерфейсів IEnumerable та IEnumerator.

* + IEnumerable: Цей інтерфейс визначає метод GetEnumerator(), який повертає об'єкт, що реалізує інтерфейс IEnumerator. Він вказує, що об'єкт може бути перебираним.
  + IEnumerator: Цей інтерфейс визначає методи та властивості, які дозволяють ітератору пересуватися по колекції та повертати поточний елемент.

# Дані якого формату зберігаються у хеш-таблицях, словниках? Які переваги їх використання?

Дані в хеш-таблицях та словниках зазвичай зберігаються у вигляді ключ-значення, де ключ є унікальним ідентифікатором, а значення пов'язані з цим ключем. Формат даних може бути різним, але ключі повинні бути хешованими, тобто перетвореними в хеш-значення за допомогою хеш-функції.

Хеш-таблиці представляють собою структуру даних, де ключі перетворюються у хеш-коди за допомогою хеш-функції, а ці хеш-коди використовуються для швидкого пошуку та доступу до значень. Реалізації хеш-таблиць можуть включати розріджені масиви, списки чи інші структури для ефективного управління ситуаціями колізій, коли два різних ключі отримують однаковий хеш-код.

Словники встановлюють відповідність між ключами і значеннями, можуть використовувати хеш-таблиці чи інші структури для оптимізованого пошуку.

Переваги використання хеш-таблиць та словників включають:

* + Швидкий доступ: Доступ до значень за ключем в хеш-таблиці зазвичай відбувається швидко, оскільки внутрішня структура дозволяє ефективно знаходити місце зберігання значення за його хешем.
  + Унікальні ключі: Ключі у хеш-таблиці повинні бути унікальними, що дозволяє ефективно ідентифікувати та знаходити конкретні значення.
  + Ефективність пошуку та вставки: Хеш-таблиці дозволяють швидко вставляти та видаляти пари ключ-значення, а також здійснювати пошук за конкретним ключем за оптимальним часом.
  + Гнучкість: Хеш-таблиці та словники можуть зберігати дані різних типів і структур, надаючи гнучкість у використанні.
  + Використання пам’яті: Зазвичай, хеш-таблиці дозволяють економити пам'ять, оскільки вони можуть динамічно змінювати свій розмір.

# Поясніть призначення узагальнених типів. Наведіть приклади.

За допомогою узагальнених типів (Generics) можна створювати класи, інтерфейси або методи, які мають можливість працювати з будь-яким типом даних, а не обмежуватися конкретним типом. Це забезпечує більшу гнучкість та безпеку використання, оскільки код може бути написаний так, щоб працювати з будь-яким типом даних, а не лише з конкретним.

Основні переваги узагальнених типів включають::

Гнучкість: Здатність працювати з різними типами даних без необхідності створення кожного разу нового коду для кожного типу.

Безпека типів: Компілятор визначає та перевіряє типи даних на етапі компіляції, що робить код більш безпечним та менш вразливим до

помилок.

Підтримка повторного використання: Можливість використовувати один і той же узагальнений код для різних типів даних сприяє повторному використанню коду.

# Поясніть призначення оператору default.

Оператор default використовується для отримання значення за замовчуванням для будь-якого типу даних. Це дає можливість ініціалізувати змінні чи отримувати значення за замовчуванням для типів, які можуть бути невідомими на етапі написання коду.

# Поясніть призначення обмежень where в узагальнених типах. Наведіть приклади.

Обмеження where (where constraints) в узагальнених типах використовуються для установлення обмежень на типи, які можуть бути передані як параметри в узагальнені класи, інтерфейси або методи.

Використання обмежень where дозволяє зміцнити контроль над типами, які можуть бути використані, і встановлює певні вимоги, які повинні виконуватися для цих типів.

# Поясніть сутність нумератору колекції.

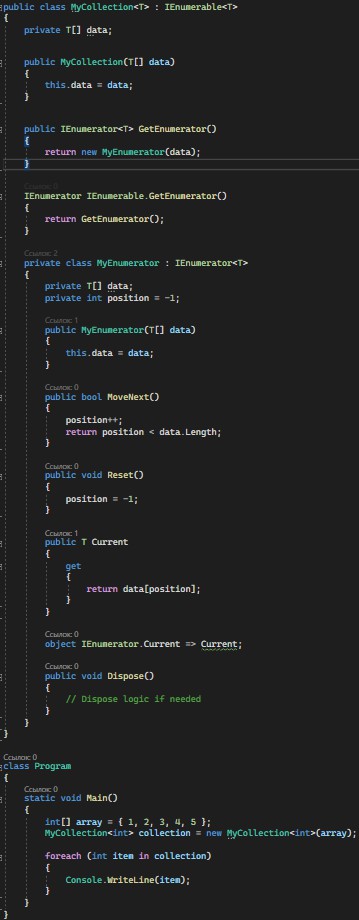
Нумератор (Enumerator) представляє собою об'єкт, який дозволяє отримувати доступ до елементів колекції послідовно, не розкриваючи її внутрішньої реалізації. Цей об'єкт реалізує інтерфейс IEnumerator і дозволяє зручно перебирати елементи за допомогою циклу foreach.

Основні методи та властивості інтерфейсу IEnumerator включають::

* + Current: Повертає поточний елемент у колекції.
  + MoveNext(): Переміщує нумератор на наступний елемент у колекції. Повертає true, якщо був переміщений, і false, якщо немає більше елементів.
  + Reset(): Повертає нумератор на початок колекції.

# Наведіть способи опису нумератору в колекції, приклад створення власного нумератору.

Щоб визначити нумератор для своєї колекції, треба включити в клас або інтерфейс метод, який реалізує інтерфейс IEnumerable та повертає нумератор. Нумератор повинен реалізувати інтерфейс IEnumerator, який включає методи Current, MoveNext, та Reset.



# Роз’ясніть сутність поняття делегата

Делегат - це тип, який представляє посилання на метод. В інших словах, він дозволяє передавати метод як аргумент і викликати його пізніше через це посилання. Делегати дозволяють створювати більш гнучкі та розширені моделі обробки подій, а також використовувати паттерн "посредник" (mediator pattern) для взаємодії об'єктів.

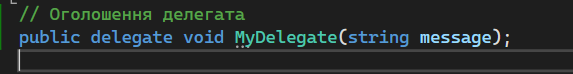
Основні аспекти делегатів включають:

* + Сигнатура: Делегати мають сигнатуру, що визначає типи параметрів і тип повертання значення для методу, який вони можуть вказувати. Посилання на метод: Делегат може утримувати посилання на один чи більше методів, які відповідають його сигнатурі.
  + Інкапсуляція методів: Делегати інкапсулюють методи, що дозволяє передавати метод як аргумент, зберігати його у властивостях, повертати з методів і т.д.
  + Виклик методів через делегат: Делегати можуть викликати метод, на який вони посилаються, викликаючи сам делегат. Це робиться за допомогою оператора ().

# Наведіть приклад опису делегата та виклику методу, використовуючи делегат.

1. **Наведіть склад класу делегату та поясніть, чим забезпечується контроль типів в делегатах.**

Клас делегата в C# включає ряд компонентів, які визначають його структуру та дозволяють контролювати типи викликаних методів**.**

* + Кожен клас делегата складається з наступних компонентів:
  + delegate (ключове слово): Вказує на те, що ми оголошуємо делегат.
  + результат (тип): Визначає тип поверненого значення методу, який може бути викликано за допомогою цього делегата. делегат
  + (ім'я): Назва делегата, яка визначає новий тип делегата. параметри
  + (параметри): Параметри, які приймає метод, який може бути викликаний за допомогою цього делегата.

В цьому прикладі MyDelegate - це делегат, який може вказувати на будь-який метод, який приймає один параметр типу string і не повертає значення (тип void).

Контроль типів в делегатах забезпечується визначенням сигнатури (типів параметрів та типу повертаного значення) для методів, які можуть бути викликані за допомогою даного делегата. Якщо типи параметрів чи тип повертаного значення не відповідають визначеним в делегаті, то компілятор видасть помилку. Це дозволяє виявити помилки на етапі компіляції і забезпечує безпечність типів під час виклику методів через делегат.

# Поясніть сутність поняття анонімного методу

Анонімний метод - це метод, який визначається без імені прямо в тілі коду. Він часто використовується для передачі функціональності або обробки подій без створення окремого методу з вказаним ім'ям.

* + Основні особливості анонімних методів: Визначення без імені: Анонімні методи не мають імені і визначаються безпосередньо в тілі коду. Вони звичайно використовуються в тому ж контексті, де і визиваються.
  + Використання делегатів: Анонімні методи часто використовуються як значення для делегатів. Зазвичай вони призначаються делегатові або використовуються безпосередньо для обробки подій.

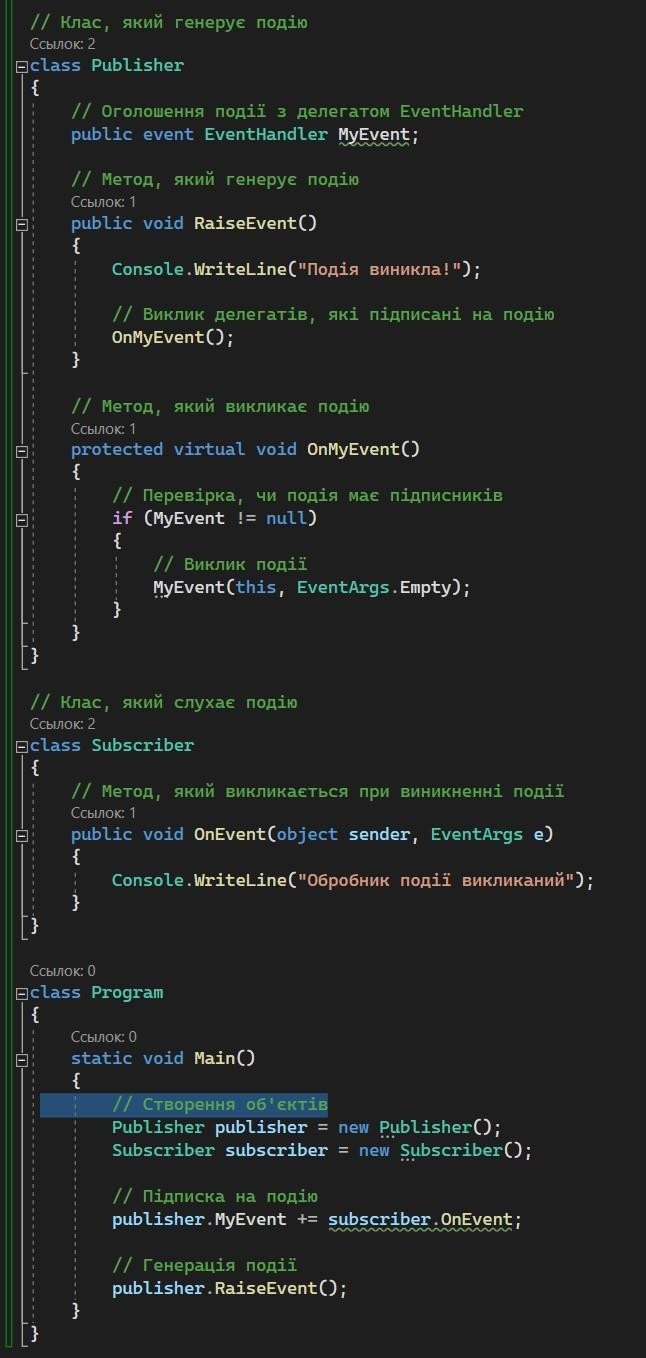
# Поясніть сутність лямбда-виразу, наведіть приклади лямбда-виразів

Лямбда-вираз представляє собою короткий та зручний спосіб створення анонімних методів. Вони дозволяють оголошувати функції безпосередньо в тілі коду, що робить код більш компактним і читабельним. Зазвичай лямбда-вирази використовуються для передачі функціональності в якості параметра, використання їх у методах LINQ або для визначення анонімних делегатів..

Основні особливості лямбда-виразів:

* + Синтаксис: Вони мають короткий та чіткий синтаксис, що дозволяє швидко визначати функції.
  + Параметри: Можуть мати нуль або більше параметрів.
  + Типи: Зазвичай не вимагають явного вказання типів параметрів, оскільки типи можуть бути виведені автоматично компілятором.
  + Вирази: Лямбда-вирази можуть включати один або кілька виразів, які визначають поведінку функції.
  + Визначення делегата: Лямбда-вирази можуть використовуватися для створення об'єктів делегатів.

# Наведіть приклад опису події та генерування події.

Події (events) є механізмом, який дозволяє об'єктам повідомляти інші об'єкти про виникнення певних подій. Події дозволяють реалізовувати механізм підписки та відписки на події, щоб реагувати на зміни стану об'єкта. Ось приклад опису події та генерування події:

У цьому прикладі:

Клас Publisher має подію MyEvent, оголошену з делегатом EventHandler. Подія виникає методом RaiseEvent, який викликає метод OnMyEvent, що перевіряє, чи є підписники на подію, і викликає її.

Клас Subscriber має метод OnEvent, який буде викликаний при виникненні події.

У методі Main створюються об'єкти Publisher і Subscriber, а після цього підписується метод OnEvent до події MyEvent. При виклику методу RaiseEvent, який генерує подію, викликається метод OnEvent підписника, і виводиться повідомлення.

# Поясніть, яким чином виконується підписання на події та скасування підписки.

Для підписки на події та відписки використовуються оператори += та

-= відповідно. Під час підписки об'єкт (підписник) додає свій метод до списку делегатів, пов'язаних із подією. При відписці об'єкт вилучає свій метод із цього списку.