**Міністерство освіти і науки України**

**Національний авіаційний університет**

**Навчально-науковий інститут комп’ютерних**

**інформаційних технологій**



Лабораторна робота №2

з дисципліни «Об’єктно орієнтоване програмування»

«ДОСЛІДЖЕННЯ КОЛЕКЦІЙ ТА УЗАГАЛЬНЕНИХ КОЛЕКЦІЙ C#»

Виконав студент:

групи ПІ-223Б

Карпов І.Б.

Перевірив викладач:

Дишлевий О.П.

Київ 2022

**Мета:** дослідити класи-колекції та узагальнені колекції у мові програмування С#.

**Завдання**

1. Описати клас, заданий варіантом у мові програмування C#;
2. Створити список об’єктів класу, вказаного в п.1, використовуючи будь-яку бібліотечну узагальнену (Generic), звичайну (Non-Generic) колекцію C#, а також масив. Продемонструвати основні операції з колекціями та масивами: додавання, видалення, оновлення, пошук елементу та прохід по набору даних. Порівняти поведінку та пояснити відмінності;
3. Створити бінарне дерево об’єктів класу, вказаного в п.1. При цьому для бінарного дерева передбачити можливість зберігати дані будь-якого типу (створити узагальнений тип). Елементами дерева не можуть бути структури;
4. Доповнити бінарне дерево з п.3 операцією порівняння (використати узагальнені типи та інтерфейси IComparer<T> і/чи IComparable<T>). Продемонструвати функціонал, реалізований з інтерфейсів порівняння через виклик методів сортування. А також реалізувати власний ітератор, використовуючи інтерфейси IEnumerable та IEnumerator. Обхід дерева реалізувати відповідно до варіанту.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Варіант | Дані | Методи та властивості | Кількість об’єктів | Порядок обходу дерева |
| 5 | Прямокутник:  Колір заповнення, колір контуру, довжини сторін | Обчислення площі;  Обчислення периметру;  Виведення інформації про фігуру | 5 | Зворотній порядок (postorder) |

**Хід роботи**

Код програми та приклад роботи зберігається на GitHub за цим посиланням:  
<https://github.com/IvanKarpov-1/course2-term1-OOP-lab2>

В ході виконання програми було описано клас (Rectangle), заданий варіантом. Клас реалізовує інтерфейси IComparable та IComparable<Rectangle> для використання методів сортування.

Використовуючи будь-яку бібліотечну узагальнену, звичайну колекцію C#, а також масив, було створено список об’єктів даного класу. Було реалізовано та продемонстровано основні операції з колекціями та масивом. А саме: додавання, видалення, оновлення (сортування), пошук елементу та прохід по набору даних.

Порівняння основних операцій в узагальненій, не узагальненій колекціях та в масиві:

* Додавання:

В колекціях додавання елементу відбувається шляхом виклику методу Add з передачею об’єкта в якості параметра. Щоб додати об’єкт до масиву, слід викликати методо Append (повертає IEnumerable), передати в нього об’єкт для додавання, перетворення результату назад в масив, шляхом виклику методу ToArray та присвоїти результат даних операцій початковому масиву: \_rectangles = \_rectangles.Append(rectangle).ToArray();

Видалення

В колекціях видалення елементу відбувається таким чином: \_rectangles.RemoveAt(\_rectangles.Count - 1);

В масиві, так: \_rectangles = \_rectangles.Take(\_rectangles.Length - 1).ToArray();

* Оновлення:

Оновлення колекцій та масиву відбувається шляхом їх сортування. Сортування в колекціях відбувається шляхом виклику методо Sort (для цього елементи колекцій повинні реалізовувати інтерфейс IComparable). Сортування масиву відбувається шляхом виклику статичного методу Sort класу Array (для цього повинен бути реалізований інтерфейс IComparable<in T>).

* Пошук:

Пошук в не узагальненій колекції здійснюється шляхом отримання індексу елемента, якщо такий є, та отримання елементу з колекції за заданим індексом з явним приведенням типу:

var temp = \_rectangles.IndexOf(rectangle);

return temp == -1 ? null : (Rectangle)\_rectangles[temp];

Пошук в узагальненій колекції відбувається шляхом виклику метода Find для колекції з заданим предикатом (для цього випадку, елементи колекції повинні перевизначити метод Equals):

return \_rectangles.Find(x => x.Equals(rectangle));

Пошук в масиві здійснюється так само, як і в не узагальненій колекції, тільки в кінці не потрібно приводити типи.

* Прохід:

Прохід по колекціям і масиву реалізовується шляхом реалізації інтерфейсів IEnumerable та IEnumerator.

Було створено бінарне дерево з передбаченою можливістю зберігання даних будь-якого типу, тобто узагальнене бінарне дерево. За умовою, елементи дерева не можуть бути структурами, тому потрібно обмежити узагальнений тип Т: BinaryTree<T> where T : class.

В операції додавання елементу до дерева, для визначення гілки, куди слід додати, був використаний метод CompareTo інтерфейсу IComparable<T>.

Був реалізований власний ітератор з зворотнім обходом дерева:

public IEnumerable<T> PostOrder()

{

if (RootNode == null) yield break;

var stack = new Stack<TreeNode<T>>();

var node = RootNode;

while (stack.Count > 0 || node != null)

{

if (node == null)

{

node = stack.Pop();

if (stack.Count > 0 && node.RightNode == stack.Peek())

{

stack.Pop();

stack.Push(node);

node = node.RightNode;

}

else

{

yield return node.Data;

node = null;

}

}

else

{

if (node.RightNode != null) stack.Push(node.RightNode);

stack.Push(node);

node = node.LeftNode;

}

}

}

public IEnumerator<T> GetEnumerator()

{

return PostOrder().GetEnumerator();

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

**Відповіді на контрольні запитання**

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

Колекції – це сукупності об’єктів. У FCL є 5 типів колекцій: неузагальнені, спеціальні, колекції з порозрядною організацією, узагальнені, паралельні.

1. **Що таке операції «упаковка» і «розпаковка»? Наведіть приклади.**

Упаковка являє собою процес перетворення типу значення в тип object або в будь-який інший тип інтерфейсу, реалізований цим типом значення. Коли тип значення упаковується загальномовним середовищем виконання (CLR), він інкапсулює значення всередині екземпляра System.Object і зберігає його в купі. Операція розпакування витягує тип значення об'єкта. Упаковка є неявною; розпакування є явним.

int i = 123;  
object o = i;

o = 123;

i = (int)o;

1. **Наведіть основні інтерфейси, які успадковуються колекціями та їх призначення.**

* IEnumerable – надає ітератор, який підтримує простий вибір елементів колекції
* ICollection – визначає методи, які дають змогу визначити кількість елементів у колекції, а також методи синхронізації для колекцій.
* IList – представляє інтерфейс колекції об'єктів, кожен із яких може бути отриманий за індексом. Також визначає методи модифікації колекції.
* IDictionary – представляє інтерфейс колекції пар «ключ-значення».
* ICloneable – визначає спосіб, що дозволяє створити копію об'єкта.
* IComparer – визначає метод, який здійснює порівняння двох об'єктів.

1. **Наведіть класи колекцій, які реалізовують типову поведінку основних структур даних.**

Вбудовані масиви, ArrayList, Hashtable, SortedList, Stack, Queue, BitArray

1. **Поясніть призначення Енумератору (ітератору) колекцій.**

Енумератор потрібен для того, щоб можна було перебирати колекцію в циклі foreach.

1. **Як реалізувати власний Енумератор (ітератор)? В яких випадках він потрібний?**

Для реалізацію власного нумератора, потрібно реалізувати інтерфейси IEnumerable та IEnumerator.

1. **Дані якого формату зберігаються у хеш-таблиці?**

Дані в форматі ключ-значення.

1. **Поясніть особливості узагальнених типів .Net.**

Тип безпеки. Генерики перекладають тягар безпеки типів з вас на компілятор. Немає необхідності писати код для перевірки правильного типу даних, оскільки він виконується під час компіляції. Зменшується необхідність приведення типу та ймовірність помилок під час виконання.

Менше коду, і код легше повторно використовувати. Немає необхідності успадковувати базовий тип і перевизначати члени. Наприклад, LinkedList<T> готовий до негайного використання.Краща продуктивність. Загальні типи колекцій, як правило, ефективніші для зберігання та маніпулювання типами значень, оскільки немає потреби включати типи значень у рамки.

Загальні делегати дозволяють зворотні виклики, безпечні для типу, без необхідності створювати кілька класів делегатів. Наприклад, загальний делегат Predicate<T> дозволяє вам створити метод, який реалізує ваші власні критерії пошуку для певного типу, і використовувати ваш метод із методами типу Array , такими як Find , FindLast і FindAll .

Генерики оптимізують динамічно згенерований код. Коли ви використовуєте генерики з динамічно згенерованим кодом, вам не потрібно генерувати тип. Це збільшує кількість сценаріїв, у яких можна використовувати легкі динамічні методи замість створення цілих збірок. Щоб отримати додаткові відомості, перегляньте розділ Як: визначити та виконати динамічні методи та DynamicMethod .

1. **Поясніть сутність обмежень узагальнених типів.**

Обмеження інформують компілятор про можливості, які повинен мати аргумент типу. Без будь-яких обмежень аргумент типу може бути будь-яким типом. Компілятор може приймати лише члени System.Object , який є основним базовим класом для будь-якого типу .NET. Якщо код клієнта використовує тип, який не задовольняє обмеження, компілятор видає помилку. Обмеження вказуються за допомогою where контекстного ключового слова.

Обмеження визначають можливості та очікування параметра типу. Оголошення цих обмежень означає, що ви можете використовувати операції та виклики методів типу обмежень. Якщо ваш загальний клас або метод використовує будь-яку операцію над загальними членами, окрім простого призначення або виклику будь-яких методів, які не підтримуються System.Object , ви застосуєте обмеження до параметра типу. Наприклад, обмеження базового класу повідомляє компілятору, що лише об’єкти цього типу або похідні від цього типу будуть використовуватися як аргументи типу. Як тільки компілятор отримає цю гарантію, він може дозволити викликати методи такого типу в загальному класі.

1. **Що може бути реалізовано, як узагальнення в C#?**

Класи, структури, інтерфейси, методи, делегати.

1. **Порівняйте універсальні, узагальнені колекції та масиви.**

Універсальні колекції зберігають елементи не в вигляді конкретно типізованих об’єктів, а як об’єкти типу object.

Узагальнена колекція застосовується, якщо всі елементи в колекції мають однаковий тип даних. Універсальна колекція забезпечує строгу типізацію, дозволяючи додавати дані лише необхідного типу.

Масив – це структура, що представляє впорядковану колекцію значень або об’єктів одного типу фіксованої довжини.

1. **Коли варто створювати власні класи колекцій?**

Варто.

1. **В чому різниця між інтерфейсами IComparer та IComparable? Для чого вони призначені? В яких випадках варто використовувати кожний із них?**

IComparable<T> – представляє метод порівняння двох об’єктів визначеного типу. Це необхідно, коли потрібно надати можливість упорядкування для об'єкта.

IComparer<T> – надає додаткові механізми порівняння. Наприклад, може знадобитися впорядкування класу за декількома полями або властивостями, за зростанням та зменшенням для одного й того ж поля або обох.

1. **В чому різниця між універсальними інтерфейсами IComparer та IComparable та узагальненими IComparer<T>, IComparable<T>? Для чого призначені ці узагальнені інтерфейси? В яких випадках варто використовувати кожний із них?**

Різниця між універсальними та узагальненими інтерфейсами полягає в тому, що версія коду, який реалізує універсальні інтерфейси може бути використаною з будь якими типами, що також реалізують універсальні інтерфейси, і це не є безпечно. А версія, що реалізує узагальнені інтерфейси є безпечною, оскільки може бути використано лише тип, який вказаний в параметрі T.

IComparable<T> – представляє метод порівняння двох об’єктів визначеного типу. Це необхідно, коли потрібно надати можливість упорядкування для об'єкта.

IComparer<T> – надає додаткові механізми порівняння. Наприклад, може знадобитися впорядкування класу за декількома полями або властивостями, за зростанням та зменшенням для одного й того ж поля або обох.

1. **Для чого призначені інтерфейси IEnumerable, IEnumerator та IEnumarable<T>, IEnumerator<T>?**

Для того, щоб перебирати об’єкти колекції в циклі foreach.