stem;

using System.Collections.Generic;

using System.Linq;

public partial class Set

{

private List<int> elements;// список, который хранит элементы множества

private readonly int id;//уникальный идентификатор множества, который генерируется при создании объекта. Поле readonly гарантирует, что идентификатор объекта будет постоянным после его инициализации. Это полезно для таких случаев, когда нужно обеспечить уникальность объекта

private static readonly int maxSize;// максимальный размер множества (устанавливается как статическая константа в статическом конструкторе). модификатор static указывает, что это поле принадлежит классу, а не конкретному экземпляру. Это означает, что maxSize является общим для всех объектов класса Set.

public static int TotalSetsCreated { get; set; } //статическое поле, которое хранит количество созданных объектов класса Set.Это значит, что значение TotalSetsCreated будет общим для всех экземпляров класса Set

public Set()// Этот конструктор используется для создания множества без начальных элементов.

{

elements = new List<int>();//инициализируется пустой список элементов

id = ComputeHash();//вызывается метод ComputeHash(), который генерирует уникальный идентификатор для текущего объекта множества

IncrementSetCount();//Метод IncrementSetCount() увеличивает статическое поле TotalSetsCreated, чтобы отслеживать общее количество созданных множеств.

}

public Set(List<int> initialElements)//Этот конструктор используется для создания множества, инициализированного списком элементов. Подходит, когда уже имеется список значений, которые нужно включить в множество.

{

elements = new List<int>(initialElements);

id = ComputeHash();

IncrementSetCount();

}

public Set(int[]? initialElements = null)//Этот конструктор похож на предыдущий, но принимает массив целых чисел (int[]) в качестве начальных элементов множества. Если массив не передан (значение по умолчанию null), создается пустое множество

{

elements = initialElements != null ? new List<int>(initialElements) : new List<int>();//Если передан массив initialElements, он преобразуется в список List<int>. Если массив отсутствует (null), создается пустой список.

id = ComputeHash();

IncrementSetCount();

}

private Set(bool isPrivate, List<int> initialElements) // Закрытый конструктор с инициализацией элементов.Этот конструктор объявлен как private, что означает, что он не может быть вызван напрямую за пределами класса. Его можно вызвать только внутри методов класса.

{

elements = new List<int>(initialElements); // Инициализация элементов

id = ComputeHash();

IncrementSetCount();

}

static Set() //Статический конструктор в классе Set используется для того, чтобы один раз инициализировать статические поля. Эти поля относятся ко всему классу в целом, а не к отдельным объектам.

{

maxSize = 100;

TotalSetsCreated = 0;

}

private void IncrementSetCount() //Этот метод увеличивает счётчик статического поля TotalSetsCreated, которое отслеживает общее количество созданных объектов класса Set.

{

TotalSetsCreated++;

}

public List<int> Elements //В данном случае свойство Elements предоставляет доступ к приватному полю elements, которое представляет собой список целых чисел (List<int>).

{

//методы используются для управления доступом к приватным полям класса, предоставляя гибкий и безопасный способ чтения и записи данных.

get { return elements; }// используется для того, чтобы получить значение поля elements.

set//используется для того, чтобы установить новое значение для поля elements.При этом новое значение передаётся через специальную переменную value, которая автоматически создаётся, когда вы присваиваете новое значение свойству:

{

if (value.Count <= maxSize)

{

elements = value;

}

else

{

throw new InvalidOperationException($"Превышен максимальный размер множества ({maxSize} элементов).");//Если количество элементов больше, чем допускается, выбрасывается исключение (InvalidOperationException), чтобы предотвратить превышение размера множества.

}

}

}

public int ID => id; //Это свойство используется для получения значения приватного поля id.

//=> — это сокращённая форма записи get-метода.

private int ComputeHash()

{

return elements.GetHashCode();

}

public static Set CreatePrivateSet(List<int> initialElements)//это статический метод в классе Set, который используется для создания экземпляра класса Set с помощью закрытого конструктора.

{

return new Set(true, initialElements); // Вызов закрытого конструктора с инициализацией.Значение true, передаваемое в закрытый конструктор Set, имеет смысл, если его назначение — указывать на то, что создаваемый объект является частным множеством.

}

public void ModifySet(ref int element, out bool added)//изменяет множество, добавляя новый элемент, если это возможно. ключевое слово ref означает, что метод получает ссылку на переменную element, и изменения, внесённые в элемент внутри метода, отразятся на переменной, переданной при вызове.

{

if (elements.Count < maxSize && !elements.Contains(element))

{

elements.Add(element); //Если оба условия верны (т.е. множество не переполнено и элемент не содержится в множестве), элемент добавляется в коллекцию elements, и added устанавливается в true, что указывает на успешное добавление.

added = true;

}

else

{

added = false;

}

}

public override bool Equals(object obj)

{

if (obj is Set otherSet)//Если obj является объектом типа Set, мы присваиваем его переменной otherSet.

{

return elements.SequenceEqual(otherSet.elements);//SequenceEqual — это метод из LINQ, который проверяет, содержат ли два списка одинаковые элементы в одинаковом порядке. Он возвращает true, если списки равны, и false, если нет.

}

return false;

}

public override int GetHashCode()

{

int hash = 17;

foreach (int element in elements)//С помощью цикла foreach мы проходим по каждому элементу в списке elements, который хранит элементы множества.

{

hash = hash \* 31 + element.GetHashCode();

}

return hash;

}

public override string ToString()

{

return $"Множество: {string.Join(", ", elements)}, ID: {id}, Количество элементов: {elements.Count}";

//Этот метод объединяет все элементы списка elements в одну строку, разделяя их запятыми. Например, если elements содержит {1, 2, 3}, то результат будет "1, 2, 3".

}

public static void DisplayClassInfo()

{

Console.WriteLine($"Класс Set. Всего создано множеств: {TotalSetsCreated}. Максимальный размер множества: {maxSize}");

}

// Методы работы с множествами

public void Add(int element)// Добавляет новый элемент в множество, если он еще не существует и если количество элементов не превышает максимальное значение.

{

if (elements.Count < maxSize && !elements.Contains(element))

{

elements.Add(element);

}

else

{

Console.WriteLine("Множество переполнено или элемент уже существует.");

}

}

public void Remove(int element)//удаление элемента

{

elements.Remove(element);

}

public Set Intersection(Set otherSet)// пересечение множеств

{

Set result = new Set();

foreach (int element in elements)

{

// Проверяем, содержит ли otherSet этот элемент

if (otherSet.elements.Contains(element))

{

result.Add(element);

}

}

return result;

}

public Set Difference(Set otherSet)// разность

{

Set result = new Set();

foreach (int element in elements)

{

if (!otherSet.elements.Contains(element))

{

result.Add(element);

}

}

return result;

}

public bool IsOnlyEvenElements()

{

return elements.All(x => x % 2 == 0) && elements.Count > 0;//=> указывает, что после него следует выражение, которое будет применяться к x.

}

public bool IsOnlyOddElements()

{

return elements.All(x => x % 2 != 0) && elements.Count > 0;

}

public bool HasNegativeElements()

{

return elements.Any(x => x < 0);

}

}

class Program

{

static void Main()

{

// Создание множеств

Set set1 = new Set(new List<int> { 2, 4, 6 });

Set set2 = new Set(new List<int> { -2, -3, 5 });

Set set3 = new Set(new List<int> { 1, 3, 5 });

// Вызов свойств

Console.WriteLine("Созданные множества:");

Console.WriteLine(set1);

Console.WriteLine(set2);

Console.WriteLine(set3);

// Вызов статического метода для создания множества с использованием закрытого конструктора

Set privateSet = Set.CreatePrivateSet([7, 8, 9]);

Console.WriteLine("Созданное частное множество:");

Console.WriteLine(privateSet);

// Вывод количества созданных объектов

Console.WriteLine($"\nВсего создано множеств: {Set.TotalSetsCreated}");

// Сравнение объектов

Console.WriteLine("\nСравнение множеств:");

Console.WriteLine($"Множество 1 и 2 равны: {set1.Equals(set2)}");

Console.WriteLine($"Множество 2 и 3 равны: {set2.Equals(set3)}");

Console.WriteLine($"Множество 1 и 3 равны: {set1.Equals(set3)}");

// Проверка типов созданных объектов

Console.WriteLine($"\nТип множества 1: {set1.GetType()}");

Console.WriteLine($"Тип множества 2: {set2.GetType()}");

Console.WriteLine($"Тип множества 3: {set3.GetType()}");

// Проверка условий и вывод результатов

Console.WriteLine("\nМножества, содержащие только четные элементы:");

if (set1.IsOnlyEvenElements())

Console.WriteLine(set1);

Console.WriteLine("\nМножества, содержащие только нечетные элементы:");

if (set2.IsOnlyOddElements())

Console.WriteLine(set2);

if (set3.IsOnlyOddElements())

Console.WriteLine(set3);

Console.WriteLine("\nМножества, содержащие отрицательные элементы:");

if (set2.HasNegativeElements())

Console.WriteLine(set2);

// Использование метода ModifySet с ref и out параметрами

int newElement = 7;

bool wasAdded;

set3.ModifySet(ref newElement, out wasAdded);

Console.WriteLine($"\nЭлемент {newElement} был добавлен в множество 3: {wasAdded}");

Console.WriteLine("Обновленное состояние множества:");

Console.WriteLine(set3.ToString());

int elementToRemove = 2;

set1.Remove(elementToRemove);

Console.WriteLine($"Элемент {elementToRemove} был удален.");

Console.WriteLine(set1.ToString());

Set intersectionSet = set2.Intersection(set3);

Console.WriteLine("Пересечение множеств:");

Console.WriteLine(intersectionSet.ToString());

Set differenceSet = set1.Difference(set2);

Console.WriteLine("Разность первого множества относительно второго:");

Console.WriteLine(differenceSet.ToString());

// Переопределённый ToString

Console.WriteLine("\nИнформация о множестве 1: " + set1.ToString());

// Создание анонимного типа

var anonymousSet = new

{

Elements = new List<int> { 3, 4, 6 },

ID = new Random().Next(1000, 9999) // Генерация случайного ID

};

// Вывод анонимного типа

Console.WriteLine("\nАнонимный тип:");

Console.WriteLine($"Элементы: {string.Join(", ", anonymousSet.Elements)}, ID: {anonymousSet.ID}");

Console.WriteLine("\nВся информация о классе:");

Set.DisplayClassInfo();

}

}