# Лабораторная работа № 4 обобщения

**Цель лабораторной работы**

Познакомиться с основой объектного подхода в языке C#, научиться работать с обобщениями

**Теоретический материал**

**Обобщённые типы (Generics)** — это механизм языка C#, позволяющий создавать классы, интерфейсы, методы и делегаты, работающие с разными типами данных без потери типобезопасности. Основная цель Generics — устранение дублирования кода и повышение производительности за счёт исключения упаковки/распаковки (boxing/unboxing) при работе с типами-значениями.

**Проблема, решаемая Generics**

До появления Generics (в .NET 2.0) для создания универсальных коллекций использовались **классы с типом**object (нетипизированные коллекции, например, ArrayList).

**Недостатки object:**

* + Упаковка/распаковка при работе с типами-значениями (int, double и др.).
  + Отсутствие проверки типов на этапе компиляции.

**Пример:**

ArrayList list = new ArrayList();

list.Add(10); // Упаковка (boxing)

int num = (int)list[0]; // Распаковка (unboxing)

Generics решают эти проблемы, предоставляя **строгую типизацию** и **оптимизацию производительности**.

**Основные понятия Generics**

**Обобщённые классы**

Класс, параметризованный типом T:

public class GenericList<T>

{

private T[] \_items;

public void Add(T item) { /\* ... \*/ }

public T Get(int index) { /\* ... \*/ }

}

**Использование:**

GenericList<int> intList = new GenericList<int>();

intList.Add(42); // Тип T = int

GenericList<string> stringList = new GenericList<string>();

stringList.Add("Hello"); // Тип T = string

**Обобщённые методы**

Методы могут быть параметризованы независимо от класса:

public static T Max<T>(T a, T b) where T : IComparable<T>

{

return a.CompareTo(b) > 0 ? a : b;

}

**Вызов:**

int maxInt = Max(10, 20); // T = int

string maxStr = Max("A", "B"); // T = string

**Обобщённые интерфейсы**

Пример интерфейса с обобщённым параметром:

public interface IRepository<T>

{

void Add(T entity);

T GetById(int id);

}

**Реализация:**

public class UserRepository : IRepository<User>

{

public void Add(User user) { /\* ... \*/ }

public User GetById(int id) { /\* ... \*/ }

}

**Ограничения обобщённых типов (Constraints)**

Для обеспечения безопасности типов Generics поддерживают **ограничения** (where), указывающие, какие типы допустимы в качестве параметров T.

| **Ограничение** | **Описание** | **Пример** |
| --- | --- | --- |
| where T : struct | T — тип-значение | int, double |
| where T : class | T — ссылочный тип | string, object |
| where T : new() | T имеет конструктор по умолчанию | T obj = new T(); |
| where T : BaseClass | T наследуется от BaseClass | T : Stream |
| where T : Interface | T реализует интерфейс Interface | T : IComparable |
| where T : U | T должен быть производным от U | T : U (вложенные Generics) |

**Пример:**

public class Sorter<T> where T : IComparable<T>

{

public void Sort(T[] array)

{

Array.Sort(array); // Использует IComparable<T>

}

}

**Преимущества Generics**

1. **Типобезопасность**
   * Компилятор проверяет соответствие типов, исключая ошибки приведения (InvalidCastException).
2. **Производительность**
   * Нет упаковки/распаковки для типов-значений.
   * Специализированный код генерируется для каждого типа T.
3. **Повторное использование кода**
   * Один обобщённый алгоритм работает с разными типами.
4. **Поддержка в стандартной библиотеке**
   * Коллекции (List<T>, Dictionary<TKey, TValue>),
   * LINQ (IEnumerable<T>),
   * Асинхронные методы (Task<T>).

**Примеры применения Generics**

**Обобщённая коллекция**

public class Stack<T>

{

private List<T> \_items = new List<T>();

public void Push(T item) => \_items.Add(item);

public T Pop()

{

T item = \_items[^1];

\_items.RemoveAt(\_items.Count - 1);

return item;

}

}

**Использование:**

var intStack = new Stack<int>();

intStack.Push(10);

int value = intStack.Pop(); // Безопасное приведение

**Задание на лабораторную работу**

1. Разработать библиотеку классов (тип проекта).
2. В библиотеке классов создать обобщенный класс согласно варианту.
3. В этом же решении создать проект «Консольное приложение».
4. Подключить библиотеку классов к консольному приложению.
5. В консольном приложении продемонстрировать работу с разработанным обобщением: создать экземпляры класса, вызвать методы.
6. Код должен быть аккуратно оформленный, чистый и понятный.

**Варианты заданий на лабораторную работу**

1. MyList<T>
   * Аналог List<T> с базовыми методами:
   * Add(T item), Remove(T item), Get(int index), Count
2. MyStack<T>
   * Классический стек с:
   * Push(T item), Pop(), Peek(), IsEmpty
3. MyQueue<T>
   * Очередь с:
   * Enqueue(T item), Dequeue(), Peek(), Count
4. CircularBuffer<T>
   * Кольцевой буфер фиксированного размера:
   * Add(T item), Get(int index), Size
5. Pair<T1, T2>
   * Контейнер для двух значений:
   * First (T1), Second (T2), Swap()
6. Calculator<T>
   * Базовые операции (ограничение where T : INumber<T>):
   * Add(T a, T b), Subtract(T a, T b), Multiply(T a, T b)
7. Range<T>
   * Диапазон значений (ограничение where T : IComparable):
   * Start, End, Contains(T value)
8. Vector2D<T>
   * 2D-вектор для int/double:
   * X, Y, Magnitude(), DotProduct(Vector2D<T> other)
9. MathStats<T>
   * Статистика для числовых типов:
   * Average(T[] array), Max(T[] array), Min(T[] array)
10. Matrix2x2<T>
    * Матрица 2x2:
    * Determinant(), Add(Matrix2x2<T> other), Multiply(Matrix2x2<T> other)
11. EqualityComparer<T>
    * Сравнение объектов:
    * AreEqual(T a, T b), GetHashCode(T obj)
12. SmartBox<T>
    * Контейнер с проверкой значения:
    * Value, IsNull, SetValue(T newValue)
13. Comparator<T>
    * Сравнение с компаратором:
    * Compare(T a, T b, Comparison<T> comparer)
14. Repository<T>
    * Имитация БД:
    * Add(T item), GetById(int id), GetAll()
15. Factory<T>
    * Создание объектов (ограничение where T : new()):
    * CreateInstance(), CreateArray(int size)
16. Cache<T>
    * Простой кэш:
    * Add(string key, T value), Get(string key), Clear()
17. Triple<T1, T2, T3>
    * Контейнер для трёх значений:
    * Item1, Item2, Item3, Rotate()
18. History<T>
    * Хранение истории значений:
    * Add(T item), Undo(), Redo(), Current
19. Either<T1, T2>
    * Хранение одного из двух типов:
    * IsLeft, LeftValue, RightValue, Switch(Action<T1> left, Action<T2> right)

**Пример разработки приложения**

Реализовать обобщённый класс:

RandomPicker<T> – случайный выбор

Методы: Add(T item), PickRandom(), Count

**1. Постановка задачи**

Класс RandomPicker<T> должен:

1. Хранить коллекцию элементов произвольного типа T
2. Позволять добавлять новые элементы
3. Возвращать случайный элемент из коллекции
4. Предоставлять информацию о количестве элементов

**2. Пошаговая реализация**

**Шаг 1: Создаем структуру класса**

public class RandomPicker<T>

{

// Внутренний список для хранения элементов

private readonly List<T> \_items = new List<T>();

// Генератор случайных чисел (static для избежания повторной инициализации)

private static readonly Random \_random = new Random();

// Дальше будут методы...

}

**Шаг 2: Реализуем метод Add()**

public void Add(T item)

{

if (item == null)

throw new ArgumentNullException(nameof(item));

\_items.Add(item);

}

**Проверка:**

* Запрещаем добавление null (выбрасываем исключение)
* Добавляем элемент во внутренний список

**Шаг 3: Реализуем метод PickRandom()**

public T PickRandom()

{

if (\_items.Count == 0)

throw new InvalidOperationException("No items to pick from");

int randomIndex = \_random.Next(\_items.Count);

return \_items[randomIndex];

}

**Логика:**

1. Проверяем, есть ли элементы
2. Генерируем случайный индекс
3. Возвращаем элемент по этому индексу

**Шаг 4: Реализуем свойство Count**

public int Count => \_items.Count;

**3. Пример использования**

class Program

{

static void Main()

{

// Создаем экземпляр для строк

var stringPicker = new RandomPicker<string>();

// Добавляем элементы

stringPicker.Add("Apple");

stringPicker.Add("Banana");

stringPicker.Add("Cherry");

Console.WriteLine($"Total items: {stringPicker.Count}"); // 3

// Выбираем случайный элемент 5 раз

for (int i = 0; i < 5; i++)

{

Console.WriteLine($"Random pick: {stringPicker.PickRandom()}");

}

// Пример с числами

var numberPicker = new RandomPicker<int>();

numberPicker.Add(10);

numberPicker.Add(20);

numberPicker.Add(30);

Console.WriteLine($"Random number: {numberPicker.PickRandom()}");

}

}

**5. Тестирование**

Рекомендуемые тест-кейсы:

1. Добавление элементов и проверка Count
2. Выбор случайного элемента из 1 элемента
3. Попытка выбора из пустой коллекции (должно быть исключение)
4. Добавление null (должно быть исключение)
5. Многократный вызов PickRandom() для проверки равномерности распределения

**Контрольные вопросы**

1. Что такое инкапсуляция?
2. Что такое свойство?
3. Что такое полиморфизм?
4. Что такое наследование?
5. Как при вызове конструктора класса-наследника передать управление конструктору базового класса?
6. Как при вызове одного конструктора класса передать управление другому конструктору класса?