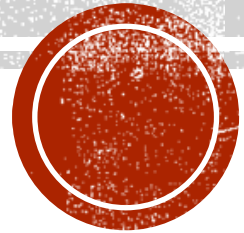
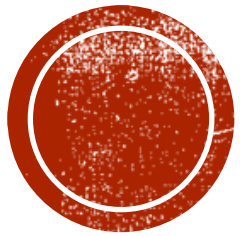
ЯЗЫКИ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ

Лекция 3





GENERIC

```
internal class ListNode<ElemT>
{
    internal ElemT Value { get; }

    internal ListNode<ElemT>? Next { get; set; } = null;

    internal ListNode(ElemT val) => Value = val;
}

public class List<T>
{
    private ListNode<T>? _firstNode;

    public void Append(T value)
    {
        var node = new ListNode<T>(value);
        if (_firstNode == null)
            _firstNode = node;
        else
        {
            var prevNode = _firstNode;
            var nextNode = _firstNode.Next;
            while (nextNode != null)
            {
                prevNode = nextNode;
                nextNode = nextNode.Next;
            }
            prevNode.Next = node;
        }
    }

    // ... other methods of list
}
```

Generic class

- Аналог шаблонов из C++
- Позволяют обобщить логику и не дублировать реализации для разных типов
- Окончательное “определение” типа (класса) формируется только в момент его использования

- *На слайде пример реализации односвязного списка*

Generic interface (1)

```
public interface ICrudRepository<T> where T : notnull
{
    Guid Create(T item);

    T? Read(Guid id);

    void Update(Guid id, T itemToUpdate);

    T? Remove(Guid id);
}
```

- Интерфейсы как и классы могут быть *«универсальными»*
- На тип может быть наложено ограничение в т.ч. Может требоваться реализация определенного интерфейса

- *На слайде пример интерфейса простого репозитория*

Generic Interface (2)

Реализация «универсального» интерфейса

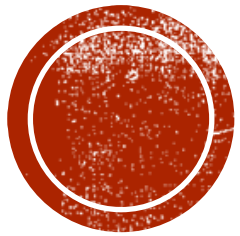
```
public class InMemoryStringRepository : ICrudRepository<string>
{
    private readonly Dictionary<Guid, string> _items = new();

    public Guid Create(string item)
    {
        var id = Guid.NewGuid();
        _items[id] = item;
        return id;
    }

    public string? Read(Guid id) => _items.TryGetValue(id, out var item) ? item : null;

    public void Update(Guid id, string itemToUpdate) => _items[id] = itemToUpdate;

    public string? Remove(Guid id)
    {
        var itemContains = _items.TryGetValue(id, out var item);
        if (itemContains)
            _items.Remove(id);
        return item;
    }
}
```



ИСКЛЮЧЕНИЯ

ИСКЛЮЧЕНИЯ В C#

- Базовый класс – **Exception**

- Наиболее распространенные:
  - **NullReferenceException**
  - **OutOfMemoryException**
  - **ArgumentException** и **ArgumentNullException**
  - **IndexOutOfRangeException** и **ArgumentOutOfRangeException**
  - **NotImplementedException**
  - **AggregateException**
  - **ArithmeticException**

СОЗДАНИЕ СОБСТВЕННОГО ИСКЛЮЧЕНИЯ

```
public class CustomException : Exception
{
    public CustomException() { }
    public CustomException(string? message) : base(message) { }
    public CustomException(string? message, Exception? innerException) : base(message, innerException) { }
}
```

```
public partial class CallExample
{
    private static void Foo() => throw new OutOfMemoryException("MyCustomMessage");
    private static void Bar() => throw new ArgumentException();

    public static void ExceptionsCatchExample()
    {
        try
        {
            Foo();
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
    }
```

```
public static void FullCatchBlockExample()
{
    try
    {
        Foo();
        Console.WriteLine("No Errors in Foo");
    }
    catch (OutOfMemoryException e) when(e.Message.Contains("MyCustomMessage"))
    {
        Console.WriteLine("Filtered OutOfMemoryException was thrown");
    }
    catch (OutOfMemoryException e)
    {
        Console.WriteLine("Other OutOfMemoryException was thrown");
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        throw;
    }
    finally
    {
        Console.WriteLine("Finally be called anyway.");
    }
}
```

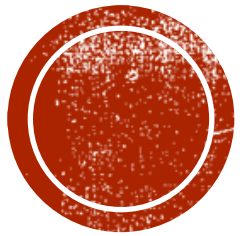
TRY-CATCH-FINALY

- Исключение последовательно пройдет через catch в порядке их написания
- Finaly выполняется в любом случае (даже если исключение не перехвачено)
- При помощи **throw;** можно выбросить пойманное исключение дальше после обработки в **catch** – такой вариант предпочтителен так как сохраняет **stack-trace**
- При помощи **when** можно накладывать дополнительные условия на **catch**

XML КОММЕНТАРИИ

```
/// <summary>
/// Interface for simple repository that allow save entities
/// </summary>
public interface ISaveRepository<T> where T : notnull
{
    /// <summary>
    /// Method for saving new entity to repository
    /// </summary>
    /// <param name="item">New entity for save. </param>
    /// <returns>Generated Id of new entity. </returns>
    /// <exception cref="ArgumentException"> If argument is already in repository. </exception>
    Guid Save(T item);
}

/// <inheritdoc cref="ISaveRepository{T}"/>
public class SaveRepositoryStub<T> : ISaveRepository<T> where T : notnull
{
    public Guid Save(T item) => Guid.NewGuid();
}
```



КОЛЛЕКЦИИ

IEnumerable<T>

```
public interface IEnumerable<T> : IEnumerable
{
    new IEnumerator<T> GetEnumerator();
}
```

```
public interface IEnumerator<T> : IEnumerator
{
    bool MoveNext();

    T Current { get; }

    void Reset();
}
```

- Простейший интерфейс коллекции, позволяющий перебирать данные в ней при помощи итератора.

- *На слайде – несколько упрощенный код из стандартной библиотеки*

```
public interface ICollection<T> : IEnumerable<T>
{
    int Count { get; }

    bool IsReadOnly { get; }

    void Add(T item);

    void Clear();

    bool Contains(T item);

    // CopyTo copies a collection into an Array, starting at a particular
    // index into the array.
    //
    void CopyTo(T[] array, int arrayIndex);


    bool Remove(T item);
}
```

ICOLLECTION<T>

- *На слайде – несколько упрощенный код из стандартной библиотеки*

FOREACH

```
public static void ForeachCall()
{
    var emails = new string[] { "petya@kek.ru", "sasha@rar.ru", "masha@bst.ru" };

    foreach (var email in emails)
    {
        Console.WriteLine(email);
    }
}
```

МАССИВЫ

- Массив – объект класса **Array**, а не указатель/область в памяти.
- Реализует **Ienumerable** и **Icollection**
- Обладает дополнительным набором полей и методов, например **Length** – размер массива.
- Подробнее – см https://docs.microsoft.com/ru-ru/dotnet/api/system.array?view=net-6.0

List<T>

- Список, в который можно добавлять элементы
- «Под капотом» является динамическим массивом
- Реализует **Ienumerable<T>** и **Icollection<T>**

ДРУГИЕ ПОЛЕЗНЫЕ КОЛЛЕКЦИИ

- ArrayList
- HashSet
- Dictionary
- Queue
- Stack
- LinkedList