Writing better C# code

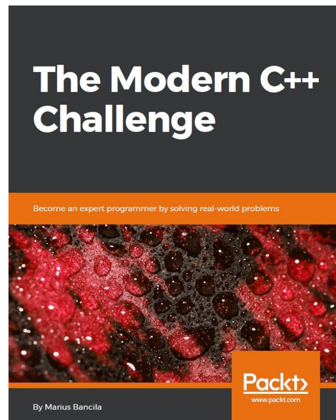
Marius Bancila

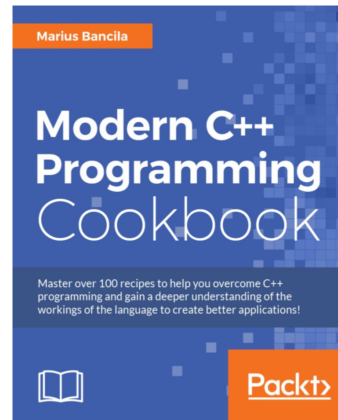
https://mariusbancila.ro

https://github.com/mariusbancila

mariusbancila

marius.bancila





Definition of “better”

- Simple
- Readable
- Consistent
- Performant

© Marius Bancila | ApexVox 2019

Agenda

- Struct vs class
- Nullables
- Freeing resources
- Using proper collections
- Yield return
- Properties vs fields
- Const vs readonly vs static
- Using strings
- Exception handling
- Declarative programming

Struct vs. class

Value types vs. reference types

© Marius Bancila | ApexVox 2019

Example

```
struct Point3D
{
   public int X {get; private set;}
   public int Y {get; private set;}
   public int Z {get; private set;}

   public Point3D(int x = 0, int y = 0, int z = 0)
   {
      X = x;
      Y = y;
      Z = z;
   }
}
```

```
class Point3D
{
   public int X {get; private set;}
   public int Y {get; private set;}
   public int Z {get; private set;}

   public Point3D(int x = 0, int y = 0, int z = 0)
   {
      X = x;
      Y = y;
      Z = z;
   }
}
```

© Marius Bancila | ApexVox 2019

Example

```
struct Point3D
{
   public int X;
   public int Y;
   public int Z;
}
```

```
class Point3D
{
   public int X;
   public int Y;
   public int Z;
}
```

Value types vs reference types

- Passed by value
- Assignments copy the value
- No inheritance
- Allocated on the stack or inlined
- Arrays of value types are allocated inline
- Boxed when cast to a reference type; unboxed when cast back

- Passed by reference
- Assignments copy the reference
- Support inheritance
- Arrays are out-of-line, refs to heap objects
- Allocated on the heap and GC

The stack / heap is an implementation detail

Rules

- Use `struct` when the type is
  - small (under 16 bytes)
  - Immutable
  - Short-lived
  - Represents a single value
  - Not boxed frequently
- Use `class` in all other cases

Records (future)

- Syntactic sugar for simple classes/structs
- Intended for Plain Old CLR Objects (POCO)
- Implements `IEquatable<T>` to support type-safe equality comparisons
  - Used by generic types to test equality
  - For value types avoids boxing and unboxing

```
public class User(int Id, string Name, string Email);
```

```
var user = new User(42, "marius", "marius@bancila.ro");

var (id, name, email) = user;
```

© Marius Bancila | ApexVox 2019

```csharp
using System;

public class User : IEquatable<User>
{
   public int Id { get; }
   public string Name { get; }
   public string Email { get; }

   public User(int Id, string Name, string Email)
   {
      this.Id = Id;
      this.Name = Name;
      this.Email = Email;
   }

   public bool Equals(User other)
   {
      return Equals(Id, other.Id) && Equals(Name, other.Name) && Equals(Email, other.Email);
   }

   public override bool Equals(object other)
   {
      return (other as User)?.Equals(this) == true;
   }

   public override int GetHashCode()
   {
      return (Id.GetHashCode() * 17 + Name.GetHashCode() + Email.GetHashCode());
   }

   public void Deconstruct(out int Id, out string Name, out string Email)
   {
      Id = this.Id;
      Name = this.Name;
      Email = this.Email;
   }

   public User With(int Id = this.Id, string Name = this.Name, string Email = this.Email) => new User(Id, Name, Email);
}
```

© Marius Bancila | ApexVox 2019

Nullables

© Marius Bancila | ApexVox 2019

Nullable value types

- `Nullable<T>` or `T?`
- `Nullable<T>` is a struct
- `T` can only be a value type
- Boxing is done with the underlying value, not the nullable value
- Use `Nullable<T>` when no value is a valid value of a value type

```
struct Foo
{
   int      Value;
   int?     Priority;
   Point3D? Point;
}
```

Nullable reference types

- `T?` where `T` is a reference type
- Enables reference types to be non-nullable
- Produce a warning when assigning `null` or when dereferencing
- Post-fix `!` operator aka *null-forgiving operator*
- Opt-in from project settings (`NullableReferenceTypes`)

```
string  s1 = null;          // warning
string? s2 = null;          // OK

string? s3 = "nullable";
string  s4 = s3;            // warning
```

```
public void Foo(string? str)
{
   if (!str.IsNullOrEmpty())
   {
      var length = str!.Length;   // null-forgiving operator
   }
}
```

Freeing resources

© Marius Bancila | ApexVox 2019

Best practices

- Implement `IDisposable`
- Always dispose objects when no longer needed
- Use the `using` statement

```
using (ResourceType resource = expression) statement

// equivalent to

{
    ResourceType resource = expression;
    try {
        statement;
    }
    finally {
        resource.Dispose();
    }
}
```

```
// value types
((IDisposable)resource).Dispose();

// nullable value types or reference types
if (resource != null)
   ((IDisposable)resource).Dispose();

// dynamic
if (((IDisposable)resource) != null)
   ((IDisposable)resource).Dispose();
```

Don't

```
int counter = 0;
string line;

var file = new System.IO.StreamReader(@"c:\test.txt");
while((line = file.ReadLine()) != null)
{
   System.Console.WriteLine(line);
   counter++;
}

file.Close();
```

Source:
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/file-system/how-to-read-a-text-file-one-line-at-a-time

Don't

```
int counter = 0;
string line;

var file = new System.IO.StreamReader(@"c:\test.txt");
while((line = file.ReadLine()) != null)
{
   System.Console.WriteLine(line);
   counter++;
}

file.Close();
```

Source:
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/file-system/how-to-read-a-text-file-one-line-at-a-time

Do

```
int counter = 0;
string line;

using(var file = new System.IO.StreamReader(@"c:\test.txt"))
{
   while((line = file.ReadLine()) != null)
   {
      System.Console.WriteLine(line);
      counter++;
   }
}
```

Do (future)

```
int counter = 0;
string line;

using var file = new System.IO.StreamReader(@"c:\test.txt");

while((line = file.ReadLine()) != null)
{
    System.Console.WriteLine(line);
    counter++;
}
```

Before and after (future)

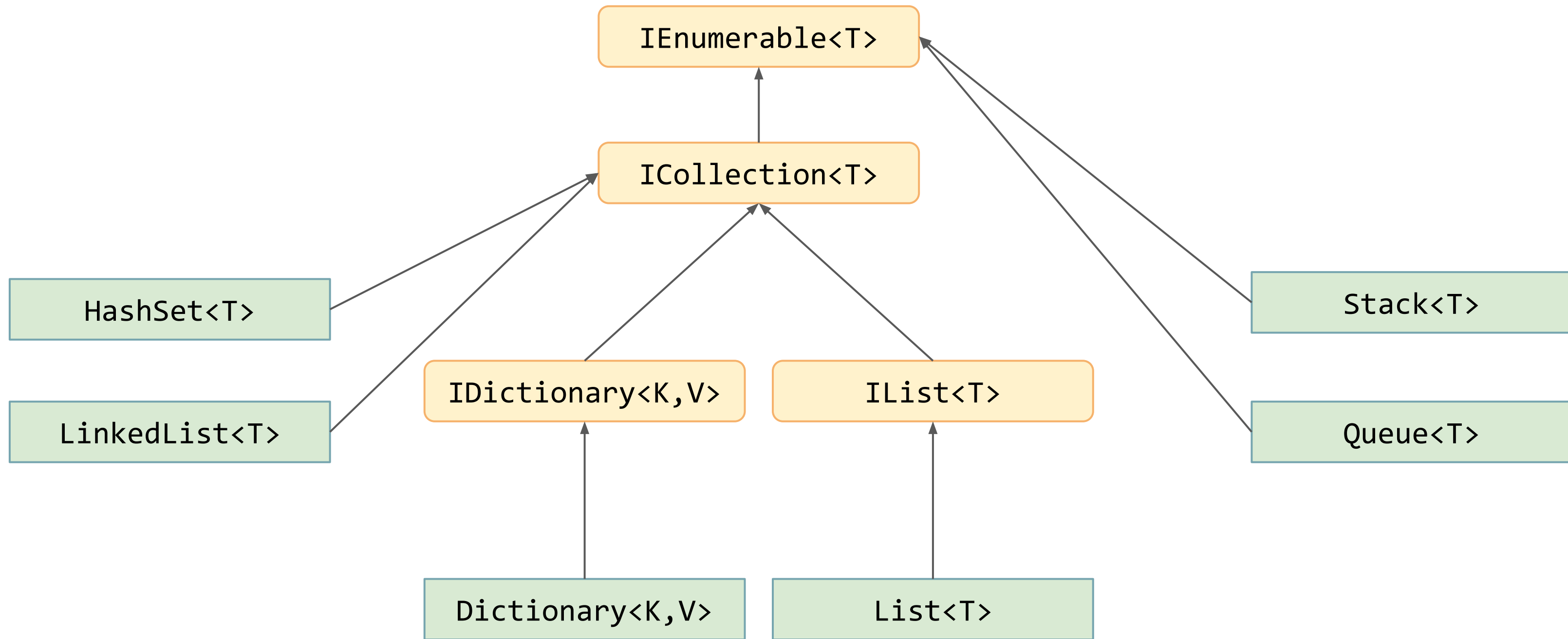
```
public void Foo() {
  using (var con = new SqlConnection(constr)) {
    con.Open();
    using (var command = con.CreateCommand()) {
      command.CommandText = "SELECT * FROM FOO";
      using (var reader = command.ExecuteReader()) {
        while (reader.Read()) {
          ProcessRecord(reader);
        }
      }
    }
  }
}
```

```
public void Foo() {
    using var con = new SqlConnection(constr) ;
    con.Open();

    using var command = con.CreateCommand() ;
    command.CommandText = "SELECT * FROM FOO";

    using var reader = command.ExecuteReader() ;
    while (reader.Read()) {
        ProcessRecord(reader);
    }
}
```

Using the proper collection

© Marius Bancila | ApexVox 2019

Generic collections

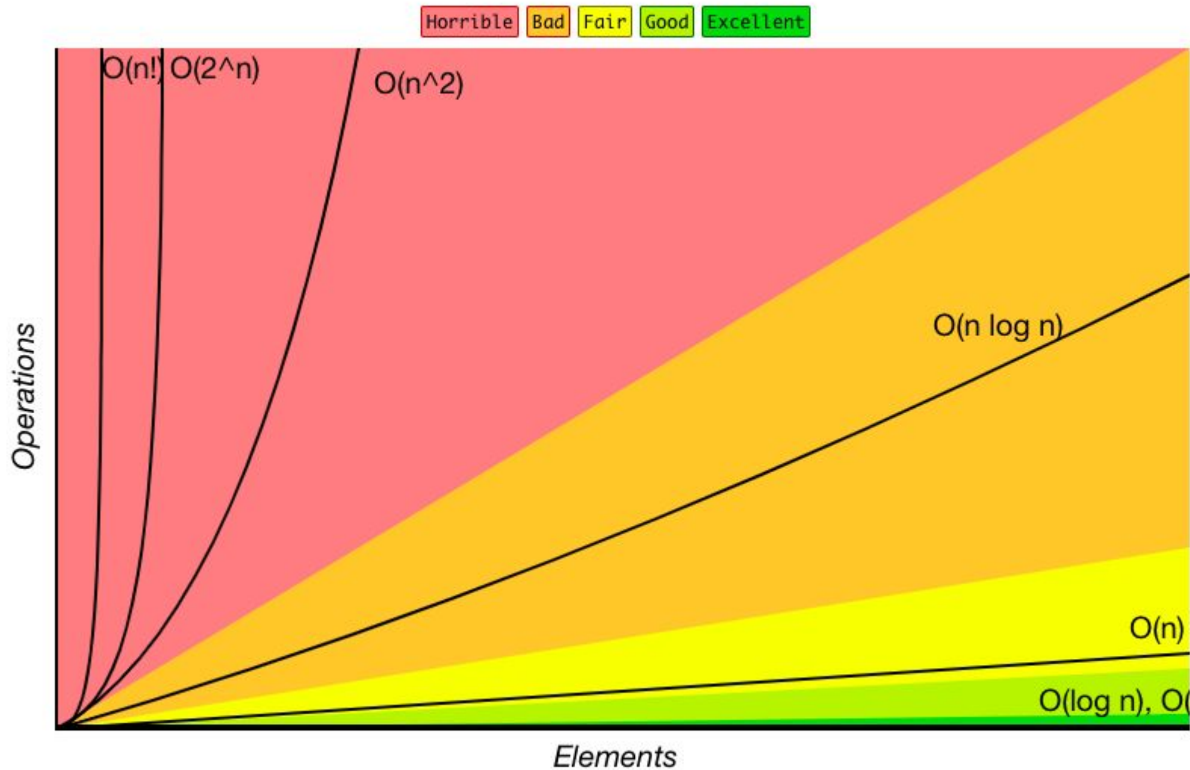


© Marius Bancila | ApexVox 2019

Typical collections

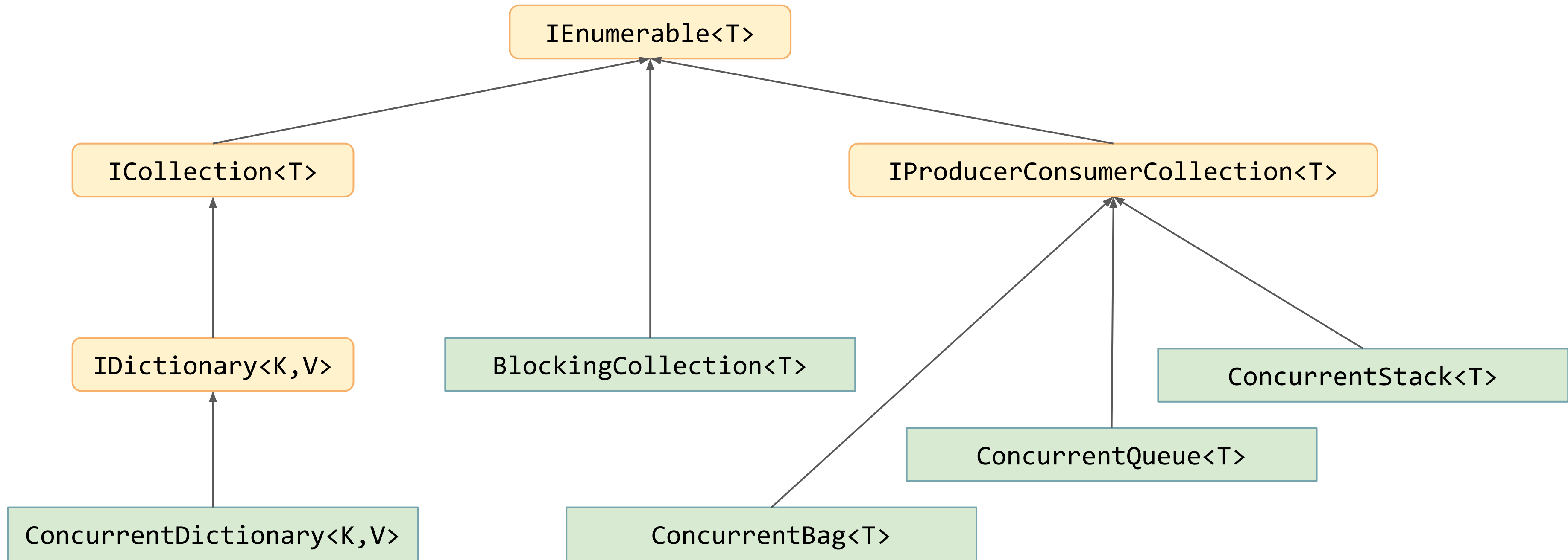
Name	Implemented as	Add/Insert	Remove	Queue/ Push	Dequeue/ Pop/Peek	Contains/ Find
`List`	Array	O(1)/O(n)*	O(n)			O(1)
`LinkedList`	Double linked list	O(1)	O(1)	O(1)	O(1)	O(n)
`Stack`	Array	O(1)/O(n)*		O(1)	O(1)	
`Queue`	Array	O(1)/O(n)*		O(1)	O(1)	
`Dictionary`	Hashtable	O(1)/O(n)**	O(1)/O(n)**			O(1)/O(n)**
`HashSet`	Hashtable	O(1)/O(n)**	O(1)/O(n)**			O(1)/O(n)**

* - beyond capacity   ** - if collision occurs



http://bigocheatsheet.com/

Concurrent collections



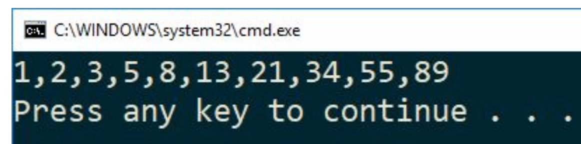
© Marius Bancila | ApexVox 2019

Yield return

When to use it

- When working with infinite sets

```
public static IEnumerable<int> GetFibonacciStream()
{
    int v1 = 0, v2 = 1;

    while (true)
    {
        int v = v1 + v2;
        v1 = v2;
        v2 = v;

        yield return v;
    }
}

static void Main(string[] args)
{
    Console.WriteLine(
        string.Join(
            ",",
            GetFibonacciStream().Take(10)));
}
```



When to use it

- When generating items in a collection is time-consuming and you could consume them while others are produced

```
void ConsumeLoop() {
   foreach (Consumable item in ProduceList())        // might have to wait here
       item.Consume();
}

IEnumerable<Consumable> ProduceList() {
   while (KeepProducing())
       yield return ProduceExpensiveConsumable();    // expensive
}
```

Async example

```
public static async Task<IEnumerable<int>> GetFibonacciStream()
{
   return await Task.Run(() =>
   {
      int v1 = 0, v2 = 1;
      var list = new List<int>();

      for(int i = 0; i < 10; ++i)
      {
         int v = v1 + v2;
         v1 = v2;
         v2 = v;

         list.Add(v);
      }

      return list;
   });
}
```

```
static void Main(string[] args)
{
   Console.WriteLine(
      string.Join(
         ",",
         GetFibonacciStream().Result.Take(10)));
}
```

Async example

```
public static async IAsyncEnumerable<int> GetFibonacciStream()
{
   int v1 = 0, v2 = 1;

   while (true)
   {
      int v = v1 + v2;
      v1 = v2;
      v2 = v;

      yield return v;
   }
}
```

© Marius Bancila | ApexVox 2019

Async streams

- Produce and consume asynchronous streams of objects
- `IAsyncEnumerable<T>`

```
static async IAsyncEnumerable<int> FetchData()
{
    for (int i = 1; i <= 10; i++)
    {
        await Task.Delay(1000); // get data such as from an IoT device
        yield return i;
    }
}

static async Task Main(string[] args)
{
    await foreach(var data in FetchData())
    {
        Console.WriteLine(data);
    }

    Console.ReadLine();
}
```

return type

yield return

await on the loop
not the function

© Marius Bancila | ApexVox 2019

Properties vs fields

© Marius Bancila | ApexVox 2019

The arguments

- Properties can be used in interfaces
- Properties allow validation
- Many .NET data bindings support only properties
- Public fields break encapsulation
- Changing a field to a property breaks binary compatibility
- More fine-grained access control with properties

Expression-bodied members

- Supported for
  - properties, read-only properties, indexers
  - methods, constructors, finalizers
- Not supported for
  - fields, events, nested-types

```
class foo
{
   public int Answer => 42; // expression-bodied property
}

class bar
{
   public int Answer = 42; // field with with initializer
}
```

© Marius Bancila | ApexVox 2019

Quiz

```
class foo
{
   public List<int> Items => new List<int>();
}

class Program
{
   static void Main(string[] args)
   {
      var f = new foo();
      foreach(var i in Enumerable.Range(1, 5))
         f.Items.Add(i);

      Console.WriteLine(string.Join(",", f.Items));
   }
}
```

```
C:\WINDOWS\system32\cmd.exe

Press any key to continue . . .
```

Quiz

```
class foo
{
    public List<int> Items => new List<int>();
}

class Program
{
    static void Main(string[] args)
    {
        var f = new foo();
        foreach(var i in Enumerable.Range(1, 5))
            f.Items.Add(i);

        Console.WriteLine(string.Join(",", f.Items));
    }
}
```

```
C:\WINDOWS\system32\cmd.exe

Press any key to continue . . .
```

© Marius Bancila | ApexVox 2019

Quiz

```
class foo
{
    public List<int> Items { get { return new List<int>(); } }
}

class Program
{
    static void Main(string[] args)
    {
        var f = new foo();
        foreach(var i in Enumerable.Range(1, 5))
            f.Items.Add(i);

        Console.WriteLine(string.Join(",", f.Items));
    }
}
```

```
C:\WINDOWS\system32\cmd.exe

Press any key to continue . . .
```

© Marius Bancila | ApexVox 2019

Quiz: The fix

```
class foo
{
   public List<int> Items { get; } = new List<int>();
}

class Program
{
   static void Main(string[] args)
   {
      var f = new foo();
      foreach(var i in Enumerable.Range(1, 5))
         f.Items.Add(i);

      Console.WriteLine(string.Join(",", f.Items));
   }
}
```

Read-only property with initializer

```
class foo
{
   public List<int> Items { get; } = new List<int>();
}

class foo
{
   private readonly List<int> items = new List<int>();
   public List<int> Items { get { return items; } }
}
```

© Marius Bancila | ApexVox 2019

Const vs readonly vs static

The details

- `const`
  - Compile-time constant values
  - Immutable
  - Must be assigned a value at initialization
  - Only for primitive or built-in types
  - Implies static
  - A constant referenced from another assembly is compiled into the calling assembly
- `static`
  - Belongs to the type not the instance
- `readonly`
  - For fields
  - Assignment only in the declaration of the class or constructor
  - Can have different values for different instances
- `static readonly`
  - Runtime constant values
  - Part of the type, not the instance
  - Cannot be changed outside the declaration (only in the static constructor)
  - Is a reference to a storage location

Example

```
class foo
{
    private const string Str1 = "apex";
    private static string Str2 = "vox";
    public readonly string Str3 = "better";
    public static readonly string Str4 = "code";
}
```

```
.class private auto ansi beforefieldinit foo
    extends [mscorlib]System.Object
{
    // Fields
    .field private static literal string Str1 = "apex"
    .field private static string Str2
    .field public initonly string Str3
    .field public static initonly string Str4

    // Methods
    .method public hidebysig specialname rtspecialname
        instance void .ctor () cil managed
    {
        // Method begins at RVA 0x2050
        // Code size 19 (0x13)
        .maxstack 8

        IL_0000: ldarg.0
        IL_0001: ldstr "better"
        IL_0006: stfld string foo::Str3
        IL_000b: ldarg.0
        IL_000c: call instance void [mscorlib]System.Object::.ctor()
        IL_0011: nop
        IL_0012: ret
    } // end of method foo::.ctor

    .method private hidebysig specialname rtspecialname static
        void .cctor () cil managed
    {
        // Method begins at RVA 0x2064
        // Code size 21 (0x15)
        .maxstack 8

        IL_0000: ldstr "vox"
        IL_0005: stsfld string foo::Str2
        IL_000a: ldstr "code"
        IL_000f: stsfld string foo::Str4
        IL_0014: ret
    } // end of method foo::.cctor

} // end of class foo
```

Obfuscation

- Compile-time constant values (const) cannot be obfuscated
- Runtime constant values (readonly static) can be obfuscated

© Marius Bancila | ApexVox 2019

What to use

- If the value does not change => `const`
- If the value might change => `readonly`
- If you need a value to be part of the type => `static`
- If you need constants to obfuscate => `static readonly`

Using strings

String formatting

- Don't concatenate with `+`

```
string message = "User " + user + " with role " + role + " is authenticated";
string text = string.Empty;
for(int i = 0; i < 100; ++i)
   text += i.ToString() + ",";  // creates 100 temporaries
```

- Use `StringBuilder` to concatenate strings

```
var sb = new StringBuilder();
for(int i = 0; i < 100; ++i)
   sb.Append(i).Append(',');
```

- Use `String.Format` / interpolation (`$` prefix)

```
string message = String.Format("User {0} with role {1} is authenticated", user, role);
string message = $"User {user} with role {role} is authenticated";
```

String comparison

- Don't use `==` to compare strings

```
string text = "encyclopædia";

if(text == "encyclopaedia") {}
```

- Use `String.Equals` to compare strings

```
if(text.Equals("encyclopaedia")) {}

if(text.Equals("encyclopaedia", StringComparison.InvariantCultureIgnoreCase)) {}
```

© Marius Bancila | ApexVox 2019

String miscellaneous

- Use `String.Empty` instead of `""`
- Remember that strings are immutable

```
string text = "Hello, world!";
text.Replace("Hello", "Ola");        // wrong
text = text.Replace("Hello", "Ola"); // OK
```

- Interpolation + verbatim `$@"..."`

Exception handling

Things to do with an exception

```
try
{
   var text = File.ReadAllBytes(@"c:\temp\data.txt");
}
finally
{
   // just clean up
}
```

```
try
{
   var text = File.ReadAllBytes(@"c:\temp\data.txt");
}
catch (Exception ex)
{
   Console.WriteLine($"Cannot read data: {ex}");
   // do something else
}
```

```
try
{
   var text = File.ReadAllBytes(@"c:\temp\data.txt");
}
catch (Exception ex)
{
   throw new MyCustomException("Cannot read data", ex);
}
```

```
try
{
   var text = File.ReadAllBytes(@"c:\temp\data.txt");
}
catch (Exception ex)
{
   Console.WriteLine($"Cannot read data: {ex}");
   throw;
}
```

Use exceptions for abnormal situations

```
public User Validate(string username, string password)
{
  if(!UserExists(username))
    throw new UserNotFoundException();
  if(!IsValid(username, password))
    throw new InvalidCredentialsException();

  return GetUser(username);
}
```

Use exceptions for abnormal situations

```
public ValidationResult Validate(string username, string password)
{
  if(username == null)
    throw new ArgumentNullException("username");
  if(password == null)
    throw new ArgumentNullException("password");

  if(!UserExists(username))
    return new ValidationResult(ValidationStatus.UserNotFound);
  if(!IsValid(username, password))
    return new ValidationResult(ValidationStatus.InvalidPassword);

  var user = GetUser(username);
  if(user == null)
    throw new InvalidOperationException("GetUser should not return null.");


  return new ValidationResult(ValidationStatus.Valid, user);
}
```

© Marius Bancila | ApexVox 2019

Catch particular exception types

```
try
{
   var text = File.ReadAllBytes(@"c:\temp\data.txt");
}
catch(FileNotFoundException ex)
{
   Console.WriteLine($"The files does not exist: {ex}");
}
catch (DirectoryNotFoundException ex)
{
   Console.WriteLine($"The directory does not exist: {ex}");
}
catch (IOException ex)
{
   Console.WriteLine($"The files cannot be opened: {ex}");
}
catch (SecurityException ex)
{
   Console.WriteLine($"The called that does not have the required permissions: {ex}");
}
catch (Exception ex)
{
   Console.WriteLine($"An exception has occurred: {ex}");
}
```

Include three ctors in custom exception classes

```
class ParsingException : Exception
{
   public ParsingException()
      : base("A parsing exception occurred")
   { }

   public ParsingException(string message)
      : base(message)
   { }

   public ParsingException(string message, Exception innerException)
      : base(message, innerException)
   { }
}
```

© Marius Bancila | ApexVox 2019

More best practices...

- Do not introduce custom exceptions if you can use a predefined one
- Design classes so that exception can be avoided
- Handle common conditions without throwing exceptions
- Don't throw from a static constructor (class throws an `System.TypeInitializationException` when referenced)

Language Integrated Query (LINQ)

What vs. How

- Find the percentage of sick leave vacation for employees hired during the last six months

- Open the company HR records.
- Find the names of the employees hired in the past six months.
- Open the recording hours registry.
- Check the total number of vacation days for each employee.
- Check how many sick leave days each of them has, if any.
- Put all the findings in a list.
- Sort the list alphabetically by employee's last name.

© Marius Bancila | ApexVox 2019

Finding odd numbers

```
var collection = new int[] {1,2,3,4,5,6,7,8,9};

var temp = new List<int>();
foreach(var e in collection)
   if(e % 2 == 1) temp.Add(e);

var odds = temp.ToArray();
```

```
var collection = new int[] {1,2,3,4,5,6,7,8,9};

var odds = collection.Where(e => e % 2 == 1)
                     .ToArray();
```

© Marius Bancila | ApexVox 2019

Finding the sum of the first three odd numbers

```
var collection = new int[] {1,2,3,4,5,6,7,8,9};

var odds = new List<int>();
foreach(var e in collection)
   if(e % 2 == 1) odds.Add(e);

int sum = 0;
for(int i = 0; i < 3 && i < odds.Count; ++i)
   sum += odds[i];
```

```
var collection = new int[] {1,2,3,4,5,6,7,8,9};

var sum = collection.Where(e => e % 2 == 1)
                    .Take(3)
                    .Sum();
```

Find warm European countries

```
string[] warmCountries = { "Turkey", "Italy", "Spain", "Saudi Arabia", "Ethiopia" };
string[] europeanCountries = { "Denmark", "Germany", "Italy", "Portugal", "Spain" };
```

```
var countries = new List<string>();
foreach(var w in warmCountries)
{
   foreach(var e in europeanCountries)
   {
      if (w == e)
         countries.Add(w);
   }
}
```

```
var result = warmCountries.Join(europeanCountries,
                                w => w,
                                e => e,
                                (w, e) => w);

// or

var result = from w in warmCountries
             join e in europeanCountries on w equals e
             select w;
```

© Marius Bancila | ApexVox 2019

LINQ Pros

- Common syntax for querying different data sources
- Strongly typed
- Concise code, easy to understand and maintain
- Deferred execution

Tools

- ReSharper
  - Code quality analysis, quick-fixes, code editing helpers, code formatting & cleanup, code gen
- NDepend
  - Code rules, code coverage analysis, tech dept estimation
- Coverity
  - Static code analysis
  - Coverity Scan is free for OSS
- FxCop
  - Legacy: static post-build code analysis on compiled assemblies
  - Analyzers: source-code based analysis during compiler execution

© Marius Bancila | ApexVox 2019

Further readings

- Tips for Writing Better C# Code
  https://www.pluralsight.com/guides/tips-for-writing-better-c-code
- Some practices to write better C#/.NET code
  https://www.codeproject.com/Articles/539179/%2FArticles%2F539179%2FSome-practices-to-write-better-Csharp-NET-code
- Tips for writing clean and best code in C#
  https://gooroo.io/GoorooTHINK/Article/17142/Tips-for-writing-clean-and-best-code-in-C/26389#.XCPuohMzbs0
- 10 common traps and mistakes in C#
  https://codeaddiction.net/articles/38/10-common-traps-and-mistakes-in-c
- Buggy C# Code: The 10 Most Common Mistakes in C# Programming
  https://www.toptal.com/c-sharp/top-10-mistakes-that-c-sharp-programmers-make
- The Stack Is An Implementation Detail
  https://blogs.msdn.microsoft.com/ericlippert/2009/04/27/the-stack-is-an-implementation-detail-part-one/
  https://blogs.msdn.microsoft.com/ericlippert/2009/05/04/the-stack-is-an-implementation-detail-part-two/
- Exceptions in C#, Done Right
  https://www.codementor.io/j2jensen/exceptions-in-c-done-right-aieyu2jhp

Further readings

- 5 C# Collections that Every C# Developer Must Know
  https://programmingwithmosh.com/csharp/csharp-collections/
- Runtime Complexity of .NET Generic Collection
  http://c-sharp-snippets.blogspot.com/2010/03/runtime-complexity-of-net-generic.html
- Yield return in C#
  https://www.kenneth-truyers.net/2016/05/12/yield-return-in-c/
- Const vs Static vs Readonly in C#
  https://exceptionnotfound.net/const-vs-static-vs-readonly-in-c-sharp-applications/
- Why properties matter
  http://csharpindepth.com/articles/chapter8/propertiesmatter.aspx
- Choosing Between Class and Struct
  https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/choosing-between-class-and-struct
- Choosing the Right Collection
  https://www.codeproject.com/Articles/1095822/Choosing-the-Right-Collection
- What's Coming in C# 8.0? Records
  https://blog.cdemi.io/whats-coming-in-c-8-0-records/
- A C# 6 gotcha: Initialization vs. Expression Bodied Members
  http://thebillwagner.com/Blog/Item/2015-07-16-AC6gotchaInitializationvsExpressionBodiedMembers

Q&A

Thank you!