Things you didn't know about C# and .NET

Marius Bancila

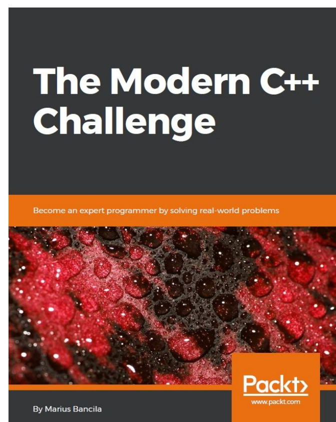
ApexVox - Cluj-Napoca, 2 Nov 2019

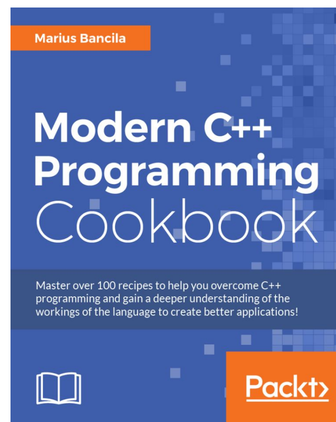
https://mariusbancila.ro

https://github.com/mariusbancila

mariusbancila

marius.bancila





Jared Parsons liked

Jeremy Kuhne
@Tofudebeast

Been writing .NET code for over 15 years, been on the .NET team for 4. Not a week goes by that I don't learn something new about writing .NET code. Sort of cool to be still saying "oh, really?" this far in. 🤯

4:30 AM · Apr 24, 2019 · Twitter Web App

Agenda

- Boxing and unboxing
- Nullables
- Null testing
- Types
- Rounding
- Conversion
- Enumerations
- Exceptions
- Testing privates
- Type aliases
- Partial function applications

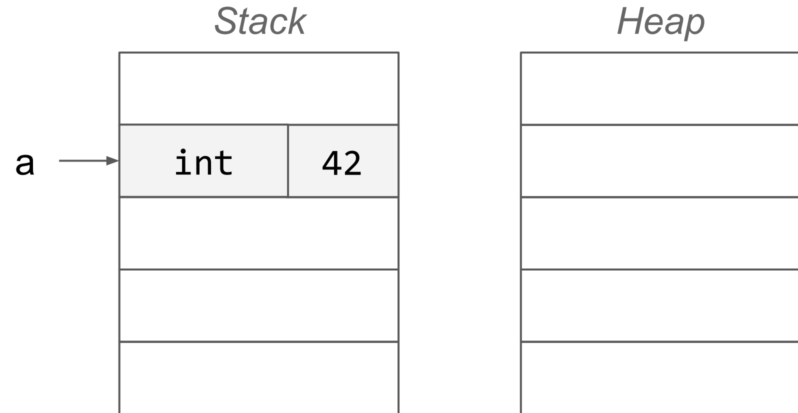
Value types & boxing

Value types semantics

- 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

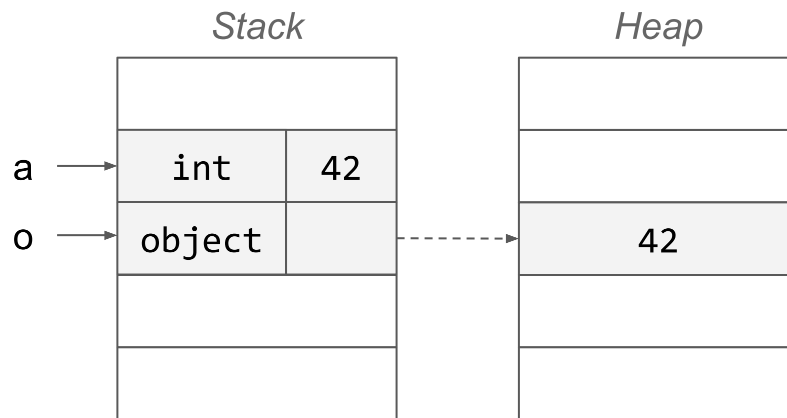
Boxing and unboxing

```
int a = 42;
```



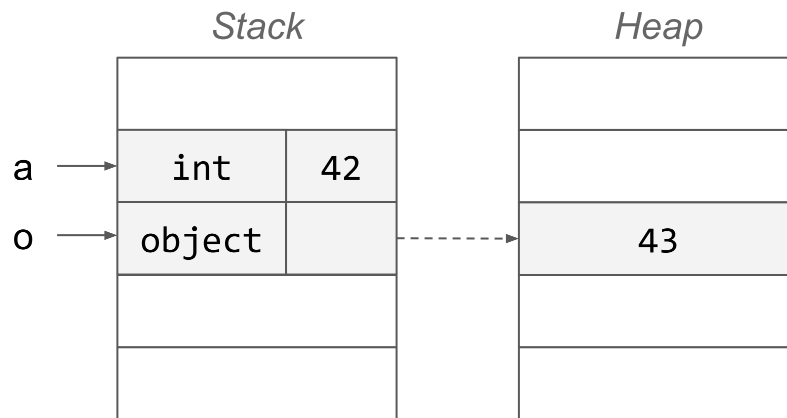
Boxing and unboxing

```
int a = 42;

object o = a;    // boxing
```



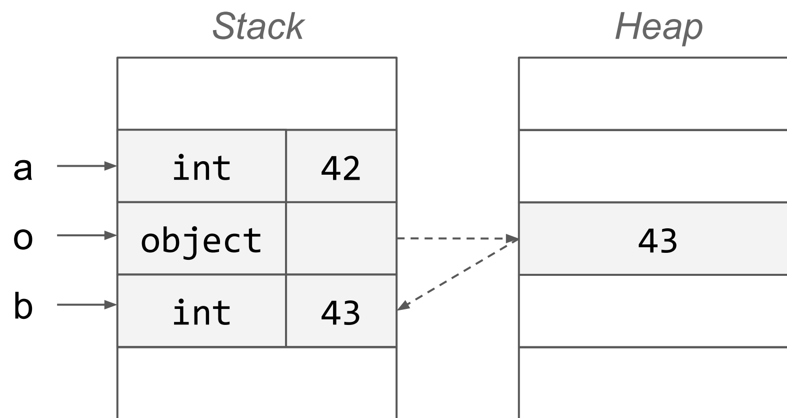
Boxing and unboxing

```
int a = 42;

object o = a;    // boxing

o = 43;
```



Boxing and unboxing

```
int a = 42;

object o = a;    // boxing

o = 43;

int b = (int)o; // unboxing

Console.WriteLine(a);   // 42
Console.WriteLine(b);   // 43
```



Boxing

```
public struct Point
{
   public int X { get; }
   public int Y { get; }

   public Point(int x, int y)
   {
      X = x; Y = y;
   }
}

public class Program
{
    public static void Main(string[] args)
    {
        Point p1 = new Point(0, 1);
        Point p2 = new Point(2,1 );

        if(p1.Equals(p2)) { }
    }
}
```

```
IL_0000: nop
IL_0001: ldloca.s 0
IL_0003: ldc.i4.0
IL_0004: ldc.i4.1
IL_0005: call instance void Point::.ctor(int32, int32)
IL_000a: ldloca.s 1
IL_000c: ldc.i4.2
IL_000d: ldc.i4.1
IL_000e: call instance void Point::.ctor(int32, int32)
IL_0013: ldloca.s 0
IL_0015: ldloc.1
IL_0016: box Point
IL_001b: constrained. Point
IL_0021: callvirt instance bool [mscorlib]System.Object::Equals(object)
IL_0026: stloc.2
// sequence point: hidden
IL_0027: ldloc.2
IL_0028: brfalse.s IL_002c
```

System.Object

```
public class Object
{
   public Object();
   ~Object();
   protected Object MemberwiseClone();
   public static bool Equals(Object objA, Object objB);
   public static bool ReferenceEquals(Object objA, Object objB);

   public virtual string ToString();
   public virtual bool Equals(Object obj);
   public virtual int GetHashCode();
   public Type GetType();
}
```

`IEquatable<T>`

- For types that perform equality of values
- Used by generic collections when testing equality (`Contains`, `IndexOf`, `LastIndexOf`, and `Remove`)
- Helps avoid boxing of value types

```
public interface IEquatable<T>
{
   bool Equals(T other);
}
```

IEquatable<T>

```
public struct Point : IEquatable<Point>
{
   public int X { get; }
   public int Y { get; }

   public Point(int x, int y)
   {
      this.X = x; this.Y = y;
   }

   public bool Equals(Point other)
   {
      return X == other.X && Y == other.Y;
   }

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

   public override int GetHashCode()
   {
      return X.GetHashCode() * 17 + Y.GetHashCode();
   }
}
```

```
IL_0000: nop
IL_0001: ldloca.s 0
IL_0003: ldc.i4.0
IL_0004: ldc.i4.1
IL_0005: call instance void Point::.ctor(int32, int32)
IL_000a: ldloca.s 1
IL_000c: ldc.i4.2
IL_000d: ldc.i4.1
IL_000e: call instance void Point::.ctor(int32, int32)
IL_0013: ldloca.s 0
IL_0015: ldloc.1
IL_0016: call instance bool Point::Equals(valuetype Point)
IL_001b: stloc.2
// sequence point: hidden
IL_001c: ldloc.2
IL_001d: brfalse.s IL_0021
```

`IEquatable<T>` best practices

- For value types always implement it
- Override `Equal(object)` => better performance
- `Equals(T)` and `Equals(Object)` must return consistent results
- If you implement `IEquatable<T>` and values can be ordered then implement `IComparable<T>` too
- If you implement `IComparable<T>` implement `IEquatable<T>` too

Nullable value types

- Useful when no value is a valid value of a value type
- `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
  - If no value the result of boxing is `null`

Quiz

```
var obj = new int?();

Console.WriteLine(obj.GetType());
```

What is the output?

A. int
B. System.Int32
C. System.Nullable<System.Int32>
D. Exception
E. None of the above

System.Object

```
public class Object
{
    public Object();
    ~Object();
    protected Object MemberwiseClone();
    public static bool Equals(Object objA, Object objB);
    public static bool ReferenceEquals(Object objA, Object objB);

    public virtual string ToString();
    public virtual bool Equals(Object obj);
    public virtual int GetHashCode();
    public Type GetType();
}
```

`Nullable<T>` boxing and unboxing

- `GetType()` is not virtual, therefore boxing is performed
- Wrapper value is boxed
- If no value the result of boxing is `null`
- Calling a method on a null object => exception

null tests

null-checks

- Typical use `!= null` and `== null`
- Prefer `is object` over `!= null`
- Prefer `is null` over `== null`
- Why
  - Efficient code
  - Works in all versions
  - Works with nullable types (even in generics as of C# 8)
  - Unlike `==` and `!=`, *does not work with value types*: you get errors

null-checks

```
class foo{}

foo f = new foo();

if (f == null) {Console.WriteLine("is null");}
if (f is null) {Console.WriteLine("is null");}
if (f != null) {Console.WriteLine("not null");}
if (f is object) {Console.WriteLine("is object");}
```

```
public static void M(IntPtr u)
{
    if (u == null)    { /* no warning or error */}
    if (u != null)    { /* no warning or error */}
    if (u is null)    { /* error */}
    if (u is object) { /* warning */}
}
```

```
int value = 42;

if (value == null)    { /* warning */ }
if (value != null)    { /* warning */ }
if (value is null)    { /* error */}
if (value is object) { /* warning */}
```

Jared Parsons
@jaredpar

Think developers should embrace `is object` as the canonical non-null test in C#. It works in every version, won't compile when the expression type is a struct, logical opposite of `is null`, emits efficient IL, ... the only fault you can say is the name isn't obvious.

1:30 AM · Apr 8, 2019 · Twitter Web Client

115 Retweets **320** Likes

Jared Parsons
@jaredpar

Replying to @tomasaschan

The != requires context to understand. Is it calling a custom operator, a lifted nullable operator or doing reference equality? The `is object` and `is null` versions require no context. They are null reference checks

8:34 AM · Apr 8, 2019 · Twitter Web Client

Jared Parsons
@jaredpar

Replying to @ben_a_adams

I think devs aren't happy at first due to the lack of familiarity. Once it's explained what it does they generally warm up. Particularly because every other option is inferior ;)

1:55 AM · Apr 8, 2019 · Twitter Web Client

Types

Quiz

How many category of types are in C#?

Quiz

How many category of types are in C#?

Value types

Reference types

Pointer types

Pointer types

- Pointer to an unmanaged type
  - `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `char`, `float`, `double`, `decimal`, `bool`
  - enums
  - pointer types
  - structs that contain fields of unmanaged types only
- Only in unsafe contexts
  - CLR cannot verify its safety (=> unverifiable code)
  - Unsafe contexts (types, methods, code blocks) declared with the keyword `unsafe` + `/unsafe` compiler switch
  - Allows pointer arithmetic

Unsafe code

```
unsafe
{
    char* pointerToChars = stackalloc char[123];

    for (int i = 65; i < 123; i++)
    {
        pointerToChars[i] = (char)i;
    }

    Console.Write("Uppercase letters: ");

    for (int i = 65; i < 91; i++)
    {
        Console.Write(pointerToChars[i]);
    }
}

// Output:
// Uppercase letters: ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

```
public unsafe struct Node
{
    public int Value;
    public Node* Left;
    public Node* Right;
}

class Test
{
   unsafe static void Main()
   {
      double d = 123.456e23;
      unsafe
      {
         byte* pb = (byte*)&d;
         for (int i = 0; i < sizeof(double); ++i)
            Console.Write("{0:X2} ", *pb++);
         Console.WriteLine();
      }
   }
} // Output: BA FF 51 A2 90 6C 24 45
```

Never write unsafe code

Unless

- Implementing time-critical algorithms. Maybe.
- Interfacing with native APIs (e.g. operating system APIs) that require pointers. Maybe. Prefer:
  - `IntPtr`
  - `SafeFileHandle`, `SafeRegistryHandle`, `SafeMemoryMappedFileHandle`, etc.

Rounding

Quiz

```
Math.Round(-2.5) ==
```

Quiz

```
Math.Round(-2.5) == -2
```

Quiz

```
Math.Round(-2.5) ==  -2

Math.Round(-1.5) ==
```

Quiz

```
Math.Round(-2.5) ==  -2

Math.Round(-1.5) ==  -2
```

Quiz

```
Math.Round(-2.5) ==  -2

Math.Round(-1.5) ==  -2

Math.Round(-0.5) ==
```

Quiz

```
Math.Round(-2.5) ==  -2

Math.Round(-1.5) ==  -2

Math.Round(-0.5) ==   0
```

Quiz

```
Math.Round(-2.5) ==  -2

Math.Round(-1.5) ==  -2

Math.Round(-0.5) ==   0

Math.Round( 0.5) ==
```

Quiz

```
Math.Round(-2.5) ==  -2
Math.Round(-1.5) ==  -2
Math.Round(-0.5) ==   0
Math.Round( 0.5) ==   0
```

Quiz

```
Math.Round(-2.5) ==  -2
Math.Round(-1.5) ==  -2
Math.Round(-0.5) ==   0
Math.Round( 0.5) ==   0
Math.Round( 1.5) ==
```

Quiz

```
Math.Round(-2.5) ==  -2
Math.Round(-1.5) ==  -2
Math.Round(-0.5) ==   0
Math.Round( 0.5) ==   0
Math.Round( 1.5) ==   2
```

Quiz

```
Math.Round(-2.5) ==  -2
Math.Round(-1.5) ==  -2
Math.Round(-0.5) ==   0
Math.Round( 0.5) ==   0
Math.Round( 1.5) ==   2
Math.Round( 2.5) ==
```

Quiz

```
Math.Round(-2.5) ==  -2
Math.Round(-1.5) ==  -2
Math.Round(-0.5) ==   0
Math.Round( 0.5) ==   0
Math.Round( 1.5) ==   2
Math.Round( 2.5) ==   2
```

Quiz

Expression	Crossed-out answer	???
`Math.Round(-2.5) ==`	-2	-3
`Math.Round(-1.5) ==`	-2	-2
`Math.Round(-0.5) ==`	0	-1
`Math.Round( 0.5) ==`	0	1
`Math.Round( 1.5) ==`	2	2
`Math.Round( 2.5) ==`	2	3

Quiz

Expression	~~Guess~~	???
`Math.Round(-2.5) ==`	~~-2~~	**-2**
`Math.Round(-1.5) ==`	~~-2~~	**-1**
`Math.Round(-0.5) ==`	~~0~~	**0**
`Math.Round( 0.5) ==`	~~0~~	**0**
`Math.Round( 1.5) ==`	~~2~~	**1**
`Math.Round( 2.5) ==`	~~2~~	**2**

Rounding algorithms

Algorithm / Value	-5.5	-2.5	-1.6	-1.1	-1.0	1.0	1.1	1.6	2.5	5.5
Up	-6.0	-3.0	-2.0	-2.0	-1.0	1.0	2.0	2.0	3.0	6.0
Down	-5.0	-2.0	-1.0	-1.0	-1.0	1.0	1.0	1.0	2.0	5.0
Ceiling	-5.0	-2.0	-1.0	-1.0	-1.0	1.0	2.0	2.0	3.0	6.0
Floor	-6.0	-3.0	-2.0	-2.0	-1.0	1.0	1.0	1.0	2.0	5.0
Half up	-6.0	-3.0	-2.0	-1.0	-1.0	1.0	1.0	2.0	3.0	6.0
Half down	-5.0	-2.0	-2.0	-1.0	-1.0	1.0	1.0	2.0	2.0	5.0
Half even	-6.0	-2.0	-2.0	-1.0	-1.0	1.0	1.0	2.0	2.0	6.0
Half odd	-5.0	-3.0	-2.0	-1.0	-1.0	1.0	1.0	2.0	3.0	5.0

Rounding algorithms

Algorithm / Value	-5.5	-2.5	-1.6	-1.1	-1.0	1.0	1.1	1.6	2.5	5.5
Up	-6.0	-3.0	-2.0	-2.0	-1.0	1.0	2.0	2.0	3.0	6.0
Down	-5.0	-2.0	-1.0	-1.0	-1.0	1.0	1.0	1.0	2.0	5.0
Ceiling	-5.0	-2.0	-1.0	-1.0	-1.0	1.0	2.0	2.0	3.0	6.0
Floor	-6.0	-3.0	-2.0	-2.0	-1.0	1.0	1.0	1.0	2.0	5.0
Half up	-6.0	-3.0	-2.0	-1.0	-1.0	1.0	1.0	2.0	3.0	6.0
Half down	-5.0	-2.0	-2.0	-1.0	-1.0	1.0	1.0	2.0	2.0	5.0
Half even	**-6.0**	**-2.0**	**-2.0**	**-1.0**	**-1.0**	**1.0**	**1.0**	**2.0**	**2.0**	**6.0**
Half odd	-5.0	-3.0	-2.0	-1.0	-1.0	1.0	1.0	2.0	3.0	5.0

- Up
  - rounds away from zero
- Down
  - rounds towards zero
- Ceiling
  - rounds towards positive infinity
- Floor
  - rounds towards negative infinity
- Half up
  - rounds towards "nearest neighbour" unless both neighbours are equidistant, in which case round up
- Half down
  - rounds towards "nearest neighbour" unless both neighbours are equidistant, in which case round down
- **Half even**
  - **rounds towards the "nearest neighbour" unless both neighbours are equidistant, in which case, round towards the even neighbour**
- Half odd
  - rounds towards the "nearest neighbour" unless both neighbours are equidistant, in which case, round towards the odd neighbour

Half even algorithm

+5.0	+5.1	+5.4	+5.5	+5.6	+5.9	+6.0	+6.1	+6.4	+6.5	+6.6	+6.9	+7.0
+5	+5	+5	+6	+6	+6	+6	+6	+6	+6	+7	+7	+7

Round-Half-Even (Odd and Even Positive Values)

-7.0	-6.9	-6.6	-6.5	-6.4	-6.1	-6.0	-5.9	-5.6	-5.5	-5.4	-5.1	-5.0
-7	-7	-7	-6	-6	-6	-6	-6	-6	-6	-5	-5	-5

Round-Half-Even (Odd and Even Negative Values)

Rounding Algorithms 101 Redux
https://www.eetimes.com/document.asp?doc_id=1274515#A10

Rounding options

```
public enum MidpointRounding
{
    ToEven = 0,
    AwayFromZero = 1
}
```

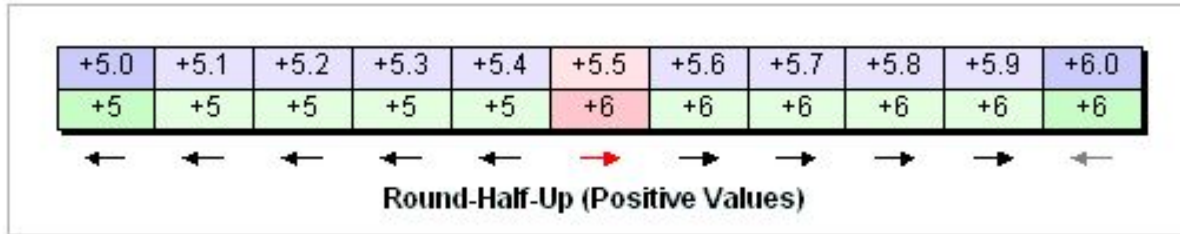
```
Math.Round(-2.5, MidpointRounding.AwayFromZero) == -3
Math.Round(-1.5, MidpointRounding.AwayFromZero) == -2
Math.Round(-0.5, MidpointRounding.AwayFromZero) == -1
Math.Round( 0.5, MidpointRounding.AwayFromZero) ==  1
Math.Round( 1.5, MidpointRounding.AwayFromZero) ==  2
Math.Round( 2.5, MidpointRounding.AwayFromZero) ==  3
```

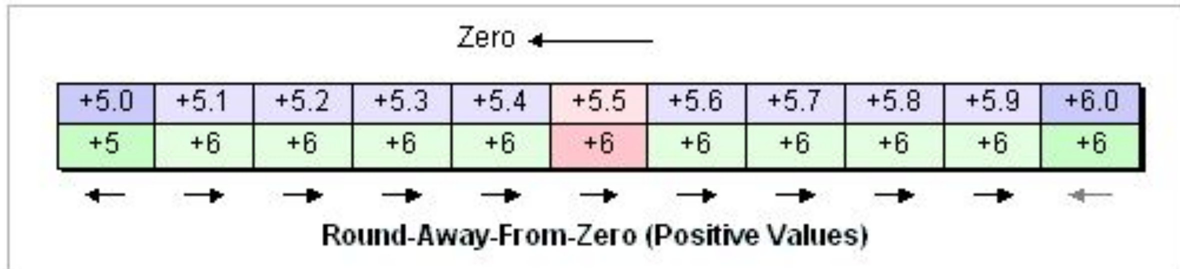
“Away from zero”

```
var arr = new double[] { 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0 };

var rou = arr.Select(n => Math.Round(n, MidpointRounding.AwayFromZero));
```

```
arr: 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6.0

rou:   5,   5,   5,   5,   5,   6,   6,   6,   6,   6,   6
```

“Away from zero” is “half up”





Rounding Algorithms 101 Redux
https://www.eetimes.com/document.asp?doc_id=1274515#A10

- Up
  - rounds away from zero
- Down
  - rounds towards zero
- Ceiling
  - rounds towards positive infinity
- Floor
  - rounds towards negative infinity
- **Half up**
  - **rounds towards "nearest neighbour" unless both neighbours are equidistant, in which case round up**
- Half down
  - rounds towards "nearest neighbour" unless both neighbours are equidistant, in which case round down
- Half even
  - rounds towards the "nearest neighbour" unless both neighbours are equidistant, in which case, round towards the even neighbour
- Half odd
  - rounds towards the "nearest neighbour" unless both neighbours are equidistant, in which case, round towards the odd neighbour

Rounding algorithms in .NET

- Ceiling
  - `Math.Ceiling(...)`
- Floor
  - `Math.Floor(...)`
- Half up
  - `Math.Round(..., MidpointRounding.AwayFromZero)`
- Half even
  - `Math.Round(...)` / `Math.Round(..., MidpointRounding.ToEven)`

But why “half even”?

- Also known as *Banker’s rounding*
- Default in IEEE 754
- On average, equal numbers of half-quantities are rounded up and down
- Average of rounded values is the closest to the average of raw values

Conversions

Casting vs Converting

```
double n = 42.1;

int i1 = (int)n;
int i2 = Convert.ToInt32(n);

Console.WriteLine(i1); // 42
Console.WriteLine(i2); // 42
```

```
double n = 42.7;

int i1 = (int)n;
int i2 = Convert.ToInt32(n);

Console.WriteLine(i1); // 42
Console.WriteLine(i2); // 43
```

Casting vs Converting

- int cast
  - Same as `Math.Ceiling()` for negatives
  - Same as `Math.Floor()` for positives
- Convert.ToInt32()
  - Same as `Math.Round()`

Conversion to enums

- A literal zero converts to any enum

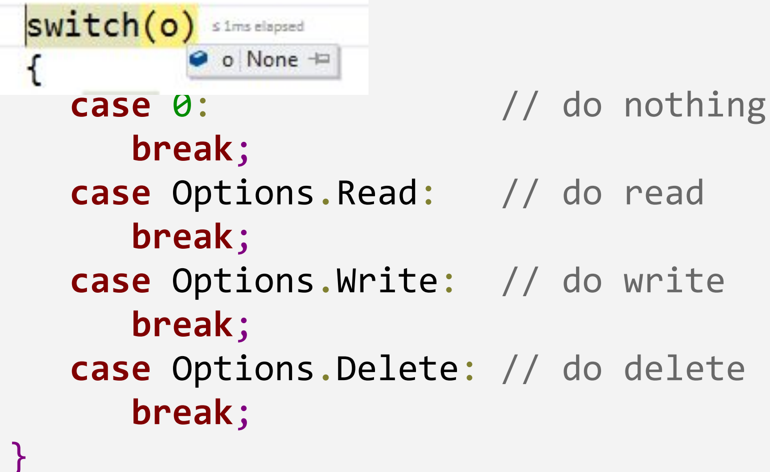
```
enum Options
{
   None,       // 0
   Read,       // 1
   Write,      // 2
   Delete      // 3
}

Options o = Options.None;

// same as

Options o = 0;
```

```
switch(o)
{
   case 0:               // do nothing
      break;
   case Options.Read:    // do read
      break;
   case Options.Write:   // do write
      break;
   case Options.Delete:  // do delete
      break;
}
```

Conversion to enums

- A literal zero converts to any enum

```
enum Options
{
   None,       // 0
   Read,       // 1
   Write,      // 2
   Delete      // 3
}

Options o = Options.None;

// same as

Options o = 0;
```



```
switch(o)
{
   case 0:                // do nothing
      break;
   case Options.Read:     // do read
      break;
   case Options.Write:    // do write
      break;
   case Options.Delete:   // do delete
      break;
}
```

Conversion to enums

- A literal zero converts to any enum

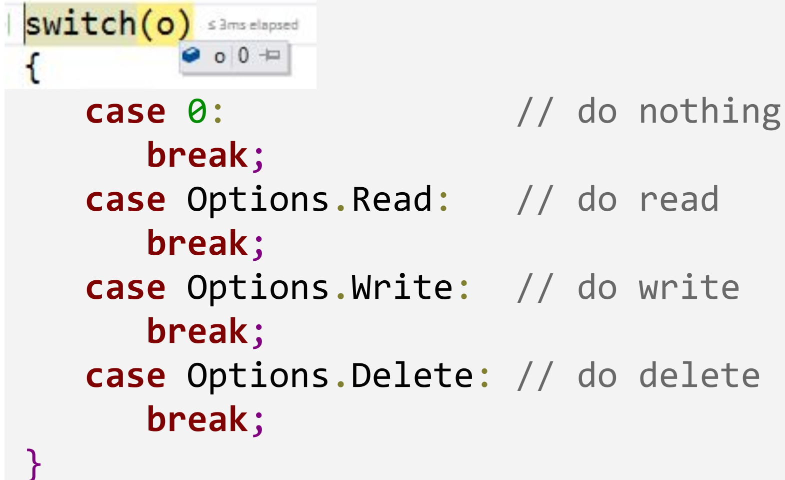
```
enum Options
{
   None = 1,   // 1
   Read,       // 2
   Write,      // 3
   Delete      // 4
}

Options o = Options.None;

// NOT same as

Options o = 0;
```

```
switch(o)
{
   case 0:                // do nothing
      break;
   case Options.Read:     // do read
      break;
   case Options.Write:    // do write
      break;
   case Options.Delete:   // do delete
      break;
}
```

Conversion to enums

- A literal zero converts to any enum

```
enum Options
{
   None = 1,   // 1
   Read,       // 2
   Write,      // 3
   Delete      // 4
}

Options o = Options.None;

// NOT same as

Options o = 0;
```



Conversion to enums

```
Options o = 0;
switch (o)
{
   case Options.None: // 1
      Console.WriteLine("do nothing");
      break;
   case Options.Read: // 2
      Console.WriteLine("do read");
      break;
   case Options.Write: // 3
      Console.WriteLine("do write");
      break;
   case Options.Delete: // 4
      Console.WriteLine("do delete");
      break;
}
```

Quiz 1/2

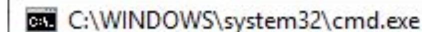
- What is the output of the following program?

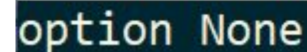
```
enum Options
{
    None,
    Read,
    Write,
    Delete
}
```

```
static void Foo(object o)
{
    Console.WriteLine($"object {o}");
}

static void Foo(Options o)
{
    Console.WriteLine($"option {o}");
}

static void Main(string[] args)
{
    Foo(0);
}
```

Quiz 1/2

- What is the output of the following program?






```
enum Options
{
    None,
    Read,
    Write,
    Delete
}
```

```
static void Foo(object o)
{
    Console.WriteLine($"object {o}");
}

static void Foo(Options o)
{
    Console.WriteLine($"option {o}");
}

static void Main(string[] args)
{
    Foo(0);
}
```

Quiz 2/2

- What is the output of the following program?

```
enum Options
{
    None = 1,
    Read,
    Write,
    Delete
}
```

```
static void Foo(object o)
{
    Console.WriteLine($"object {o}");
}

static void Foo(Options o)
{
    Console.WriteLine($"option {o}");
}

static void Main(string[] args)
{
    Foo(0);
}
```

Quiz 2/2

- What is the output of the following program?

C:\WINDOWS\system32\cmd.exe

```
option 0
```

```
enum Options
{
    None = 1,
    Read,
    Write,
    Delete
}
```

```
static void Foo(object o)
{
    Console.WriteLine($"object {o}");
}

static void Foo(Options o)
{
    Console.WriteLine($"option {o}");
}

static void Main(string[] args)
{
    Foo(0);
}
```

But why?

- To avoid explicit cast for bit-flag enumerations

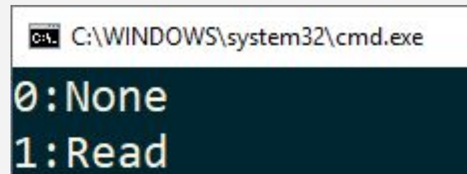
```
[Flags]
enum Options
{
   None   = 0,
   Read   = 1,
   Write  = 2,
   Delete = 4
}

Options o = 0;

// avoid an explicit cast

Options o = (Options)0;
```

Extension methods on enums

- It’s possible

```
enum Options
{
   None,
   Read,
   Write,
   Delete
}

static class OptionsExtensions
{
   public static string AsText(this Options o)
   {
      return $"{(int)o}:{o}";
   }
}
```

```
Console.WriteLine(Options.None.AsText());
Console.WriteLine(Options.Read.AsText());
```



Exceptions

Exception dispatching

```
public void Foo()
{
   try
   {
      ExecuteFunctionThatThrows();
   }
   catch(Exception ex)
   {
      // perhaps do something, like logging

      // rethrow to preserve info
      throw;
   }
}
```

Exception dispatching

```
public void Foo()
{
   Exception lastException;
   try
   {
      ExecuteFunctionThatThrows();
   }
   catch(Exception ex)
   {
      lastException = ex;
   }

   // do something you cannot do in the catch block

   // rethrow
   if(lastException != null)
     throw lastException;
}
```

Exception dispatching

```
public void Foo()
{
    ExceptionDispatchInfo exceptionDispatchInfo = null;
    try
    {
        ExecuteFunctionThatThrows();
    }
    catch(Exception ex)
    {
        exceptionDispatchInfo = ExceptionDispatchInfo.Capture(ex);
    }

    // do something you cannot do in the catch block

    // rethrow
    if (exceptionDispatchInfo != null)
        exceptionDispatchInfo.Throw();
}
```

Unit testing

Unit-testing internals

```
namespace demolib
{
   internal class InternalClass
   {
      public int Run() { return 42; }

      internal int Add(int a, int b) { return a + b; }
   }
}
```

Unit-testing internals

```
[TestClass]
public class UnitTests
{
   [TestMethod]
   public void TestInternalClass()
   {
      var obj = new InternalClass(); // 'InternalClass' is inaccessible due to its protection level

      var r1 = obj.Run();            // 'InternalClass.Run()' is inaccessible due to its protection level
      Assert.AreEqual(42, r1);

      var result = obj.Add(1, 2);    // 'InternalClass.Add(int, int)' is inaccessible due to its protection level
      Assert.AreEqual(3, result);
   }
}
```

Unit-testing internals

- Add attribute `InternalsVisibleTo()` in the tested assembly

```
using System.Runtime.CompilerServices;

[assembly:InternalsVisibleTo("demotest")]
```

Unit-testing internals

- If the tested assembly is a strong-named signed assembly you must
  - Sign the testing assembly too
  - Specify the public key from the file that was used to sign the testing assembly

```
using System.Runtime.CompilerServices;

[assembly:InternalsVisibleTo("demotest, PublicKey="+
"0024000004800000940000000602000000240000525341310004000001000100291fb5c9c86bda" +
"4d78d59f9f8abe09ab5d9370dc77ebac20d03d6d4fdbf6c54cdaab8df6c4e467fa29ae00969835" +
"c660936b2539bfec4c4da8bb0ba418feeb8e19d20166482c473ee8a0acbba288867f7a66740284" +
"6266f8f9808ef85c68d0ee30c710e64e822ad77122da5fa47a6dcdd0ef1f3e9d9cddd18b74fefd" +
"619074ec")]
```

Unit-testing privates

```
namespace demolib
{
   class PrivateClass
   {
      protected int Run() { return 42; }

      private int Add(int a, int b) { return a + b; }
   }
}
```

Unit-testing privates

```
[TestMethod]
public void TestPrivateClass()
{
   var obj = new PrivateObject(typeof(PrivateClass));

   var r1 = (int)obj.Invoke("Run");
   Assert.AreEqual(42, r1);

   var r2 = (int)obj.Invoke("Add", 1, 2);
   Assert.AreEqual(3, r2);
}
```

Unit-testing private statics

```
namespace demolib
{
   class PrivateClass
   {
      protected int Run() { return 42; }

      private static int Add(int a, int b) { return a + b; }
   }
}
```

Unit-testing private statics

```
[TestMethod]
public void TestPrivateClass()
{
   var obj = new PrivateType(typeof(PrivateClass));

   var r2 = (int)obj.InvokeStatic("Add", 1, 2);
   Assert.AreEqual(3, r2);
}
```

Type aliases

Type aliases

```
namespace test
{
   class Program
   {
      static void Main(string[] args)
      {
         var d = new Dictionary<int, List<string>>();
      }
   }
}
```

```
namespace test
{
   using MyDictionary = Dictionary<int, List<string>>;

   class Program
   {
      static void Main(string[] args)
      {
         var d = new MyDictionary();

         Console.WriteLine(
            typeof(MyDictionary) ==
            typeof(Dictionary<int, List<string>>));
      }
   }
}
```

Type aliases

```
using System;
namespace demo_lib
{
   public class foo
   {
      public void SayHello()
      {
         Console.WriteLine("Hello from Library 1!");
      }
   }
}
```

```
using System;
namespace demo_lib
{
   public class foo
   {
      public void SayHello()
      {
         Console.WriteLine("Hello from Library 2!");
      }
   }
}
```

Type aliases

Solution 'extern_alias_demo' (3 projects)
- demo_lib1
  - Properties
  - References
  - foo.cs
- demo_lib2
  - Properties
  - References
  - foo.cs
- extern_alias_demo
  - Properties
  - References
    - Analyzers
    - demo_lib1
    - demo_lib2
    - Microsoft.CSharp
    - System
    - System.Core
    - System.Data
    - System.Data.DataSetExtensions
    - System.Net.Http
    - System.Xml
    - System.Xml.Linq
  - App.config
  - Program.cs

```
class Program
{
    static void Main(string[] args)
    {
        var f = new foo();
        f.SayHello();
    }
}
```

```
5        static void Main(string[] args)
6        {
7            var f = new foo();
```

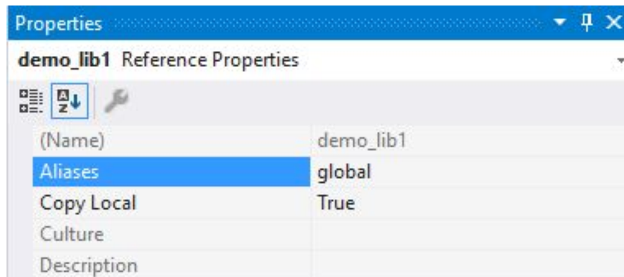
using demo_lib;
demo_lib.foo
Generate class 'foo' in new file
Generate class 'foo'
Generate nested class 'foo'
Generate new type...

CS0246 The type or namespace name 'foo' could not be found (are you missing a using directive or an assembly reference?)

```
using demo_lib;

namespace extern_alias_demo
...
```

Preview changes

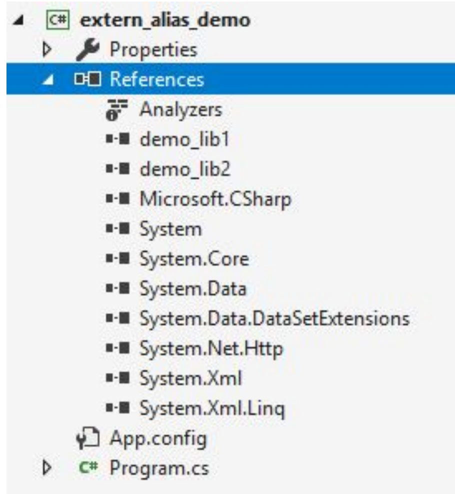
Type aliases: extern




```xml
<ItemGroup>
   <ProjectReference Include="..\demo_lib1\demo_lib1.csproj">
      <Project>{2771673b-5d23-4215-9b9e-fb4bf4e691c4}</Project>
      <Name>demo_lib1</Name>
      <Aliases>lib1</Aliases>
   </ProjectReference>
   <ProjectReference Include="demo_lib2\demo_lib2.csproj">
      <Project>{9f285780-9ee6-4ea1-a6c3-f52d92f61b1e}</Project>
      <Name>demo_lib2</Name>
      <Aliases>lib2</Aliases>
   </ProjectReference>
</ItemGroup>
```

```csharp
namespace extern_alias_demo
{
   extern alias lib1;
   extern alias lib2;

   class Program
   {
      static void Main(string[] args)
      {
         var f1 = new lib1::demo_lib.foo();
         f1.SayHello();

         var f2 = new lib2::demo_lib.foo();
         f2.SayHello();
      }
   }
}
```

Type alias: mscorlib




```xml
<PropertyGroup>
    <AdditionalExplicitAssemblyReferences/>
</PropertyGroup>
<ItemGroup>
   <Reference Include="mscorlib">
      <Aliases>global, mscorlib</Aliases>
   </Reference>
   <Reference Include="System" />
   <Reference Include="System.Core" />
   <Reference Include="Microsoft.CSharp" />
   <Reference Include="System.Data" />
</ItemGroup>
```

```csharp
namespace extern_alias_demo
{
   extern alias mscorlib;

   class Program
   {
      static void Main(string[] args)
      {
         var list = new mscorlib::System.Collections.Generic.List<int>();
      }
   }
}
```

Partial function application

Ever wrote something like this?

```
email.Headers = GetDecodedParameterValue("headers", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Text = GetDecodedParameterValue("text", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Html = GetDecodedParameterValue("html", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.From = GetDecodedParameterValue("from", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.To = GetDecodedParameterValue("to", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Cc = GetDecodedParameterValue("cc", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Subject = GetDecodedParameterValue("subject", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Dkim = GetDecodedParameterValue("dkim", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Spf = GetDecodedParameterValue("SPF", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Charsets = GetDecodedParameterValue("charsets", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.SpamScore = GetDecodedParameterValue("spam_score", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.SpamReport = GetDecodedParameterValue("spam_report", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.SenderIp = GetDecodedParameterValue("sender_ip", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.AttachmentsInfo = GetDecodedParameterValue("attachment-info", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
```

Reformatted...

```
email.Headers         = GetDecodedParameterValue("headers",         charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Text            = GetDecodedParameterValue("text",            charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Html            = GetDecodedParameterValue("html",            charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.From            = GetDecodedParameterValue("from",            charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.To              = GetDecodedParameterValue("to",              charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Cc              = GetDecodedParameterValue("cc",              charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Subject         = GetDecodedParameterValue("subject",         charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Dkim            = GetDecodedParameterValue("dkim",            charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Spf             = GetDecodedParameterValue("SPF",             charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Charsets        = GetDecodedParameterValue("charsets",        charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.SpamScore       = GetDecodedParameterValue("spam_score",      charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.SpamReport      = GetDecodedParameterValue("spam_report",     charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.SenderIp        = GetDecodedParameterValue("sender_ip",       charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.AttachmentsInfo = GetDecodedParameterValue("attachment-info", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
```

Can get rid of the duplicated part?

```
email.Headers         = GetDecodedParameterValue("headers",         charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Text            = GetDecodedParameterValue("text",            charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Html            = GetDecodedParameterValue("html",            charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.From            = GetDecodedParameterValue("from",            charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.To              = GetDecodedParameterValue("to",              charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Cc              = GetDecodedParameterValue("cc",              charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Subject         = GetDecodedParameterValue("subject",         charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Dkim            = GetDecodedParameterValue("dkim",            charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Spf             = GetDecodedParameterValue("SPF",             charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.Charsets        = GetDecodedParameterValue("charsets",        charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.SpamScore       = GetDecodedParameterValue("spam_score",      charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.SpamReport      = GetDecodedParameterValue("spam_report",     charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.SenderIp        = GetDecodedParameterValue("sender_ip",       charsets, parsersPerEncoding, string.Empty, desiredEncoding);
email.AttachmentsInfo = GetDecodedParameterValue("attachment-info", charsets, parsersPerEncoding, string.Empty, desiredEncoding);
```

The irrelevant details

```
private static string GetDecodedParameterValue(
    string parameterName,
    IEnumerable<KeyValuePair<string, Encoding>> encodings,
    Dictionary<Encoding, MultipartFormDataParser> parsersPerEncoding,
    string defaultValue,
    Encoding defaultEncoding)
{
    // do things
}
```

Partial function application

```
string decodedParam(string field) => GetDecodedParameterValue(field, charsets, parsersPerEncoding, string.Empty, desiredEncoding);

email.Headers         = decodedParam("headers");
email.Text            = decodedParam("text");
email.Html            = decodedParam("html");
email.From            = decodedParam("from");
email.To              = decodedParam("to");
email.Cc              = decodedParam("cc");
email.Subject         = decodedParam("subject");
email.Dkim            = decodedParam("dkim");
email.Spf             = decodedParam("SPF");
email.Charsets        = decodedParam("charsets");
email.SpamScore       = decodedParam("spam_score");
email.SpamReport      = decodedParam("spam_report");
email.SenderIp        = decodedParam("sender_ip");
email.AttachmentsInfo = decodedParam("attachment-info");
```

Partial function application

```
Func<string, string> decodedParam = (field) => GetDecodedParameterValue(field,
                                                                         charsets,
                                                                         parsersPerEncoding,
                                                                         string.Empty,
                                                                         desiredEncoding);
```

```
string decodedParam(string field) => GetDecodedParameterValue(field,
                                                              charsets,
                                                              parsersPerEncoding,
                                                              string.Empty,
                                                              desiredEncoding);
```

Q&A

Further readings

- Rounding Algorithms 101 Redux
  https://www.eetimes.com/document.asp?doc_id=1274515#A10
- 8 things you probably didn't know about C#
  https://damieng.com/blog/2012/10/29/8-things-you-probably-didnt-know-about-csharp
- Sharp Regrets: Top 10 Worst C# Features
  http://www.informit.com/articles/article.aspx?p=2425867
- Top 15 Underutilized Features of .NET
  https://www.automatetheplanet.com/top-15-underutilized-features-dotnet/
- Hidden Features of C#?
  https://stackoverflow.com/questions/9033/hidden-features-of-c