Telerik Academy

Correct Use of Variables, Data, Expressions and Constants

Correctly Organizing Data and Expressions

Telerik Software Academy
Learning & Development
http://academy.telerik.com

Telerik Academy

Table of Contents

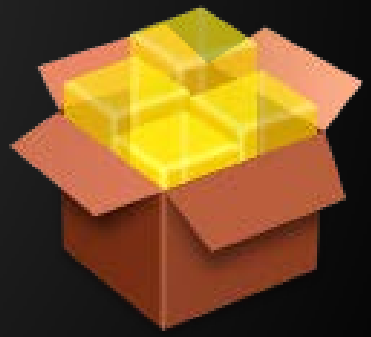
Telerik Academy

Principles for Initialization

Best Practices

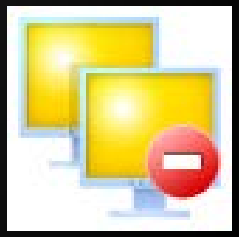
Initially Assigned Variables in C#

- Static variables
- Instance variables of class instances
- Instance variables of initially assigned struct variables
- Array elements
- Value parameters
- Reference parameters
- Variables declared in a catch clause or a foreach statement



Initially Unassigned Variables in C#

- Instance variables of initially unassigned struct variables
- Output parameters
  - Including the `this` variable of struct instance constructors
- Local variables
  - Except those declared in a `catch` clause or a `foreach` statement



Guidelines for Initializing Variables

- When the problems can happen?
  - The variable has never been assigned a value
  - The value in the variable is outdated
  - Part of the variable has been assigned a value and a part has not
    - E.g. `Student` class has initialized name, but faculty number is left unassigned
- Developing effective techniques for avoiding initialization problems can save a lot of time

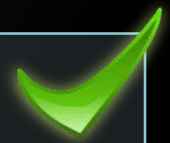
Variable Initialization

- Initialize all variables before their first usage
  - Local variables should be manually initialized
  - Declare and define each variable close to where it is used
  - This C# code will result in compiler error:

```
int value;
Console.WriteLine(value);
```

  - We can initialize the variable at its declaration:

```
int value = 0;
Console.WriteLine(value);
```



Variable Initialization (2)

- Pay special attention to counters and accumulators
  - A common error is forgetting to reset a counter or an accumulator

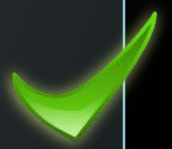
```
int sum = 0;
for (int i = 0; i < array.GetLength(0); i++)
{
  for (int j = 0; j < array.GetLength(1); j++)
  {
    sum = sum + array[i, j];
  }
  Console.WriteLine(
    "The sum of the elements in row {0} is {1}", sum);
}
```

The sum must be reset after the end of the inner for loop

Variable Initialization (3)

- Check the need for reinitialization
  - Make sure that the initialization statement is inside the part of the code that's repeated
- Check input parameters for validity
  - Before you assign input values to anything, make sure the values are reasonable

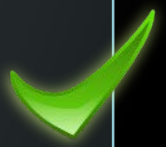
```
int input;
bool validInput =
  int.TryParse(Console.ReadLine(), out input);
if (validInput)
{
  …
```



Partially Initialized Objects

- Ensure objects cannot get into partially initialized state
  - Make all fields private and require valid values for all mandatory fields in all constructors
  - Example: `Student` object is invalid unless it has `Name` and `FacultyNumber`

```
class Student
{
  private string name, facultyNumber;
  public Student(string name, string facultyNumber)
  { … }
}
```

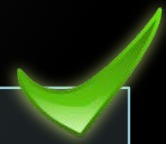


Variables – Other Suggestions

- Don't define unused variables
  - Compilers usually issues warnings
- Don't use variables with hidden purpose
  - Incorrect example:

```
int mode = 1;
…
if (mode == 1) …; // Read
if (mode == 2) …; // Write
if (mode == 3) …; // Read and write
```

  - Use enumeration instead:



```
enum ResourceAccessMode { Read, Write, ReadWrite }
```

Telerik Academy

Retuning Result from a Method

- Always assign the result of a method in some variable before returning it. Benefits:
  - Improved code readability
    - The returned value has self-documenting name
  - Simplified debugging
  - Example:

```
int salary = days * hoursPerDay * ratePerHour;
return salary;
```

The intent of the formula is obvious

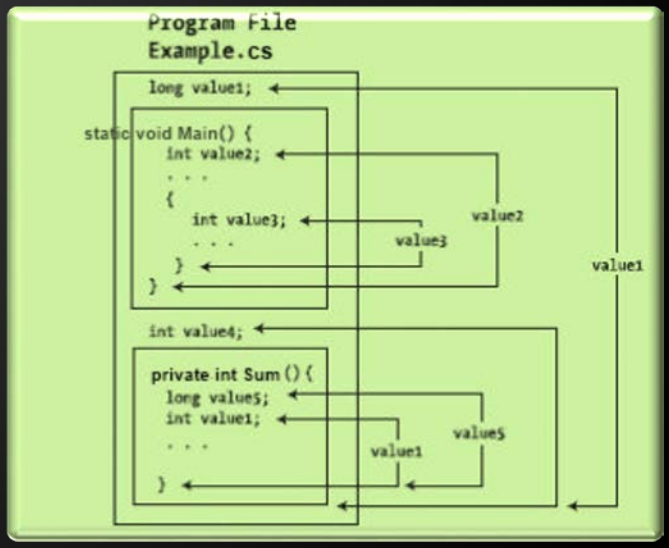
We can put a breakpoint at this line and check if the result is correct

  - Incorrect example:

```
return days * hoursPerDay * ratePerHour;
```

Telerik Academy

Scope, Lifetime, Span



Scope of Variables

- Scope – a way of thinking about a variable's celebrity status
  - How famous is the variable?
  - Global (static), member variable, local
  - Most famous variables can be used anywhere, less famous variables are much more restricted
  - The scope is often combined with visibility
- In C# and Java, a variable can also be visible to a package or a namespace

Visibility of Variables

- Variables' visibility is explicitly set restriction regarding the access to the variable
  - `public`, `protected`, `internal`, `private`
- Always try to reduce maximally the variables scope and visibility
  - This reduces potential coupling
  - Avoid public fields (exception: `readonly` / `const` in C# / `static final` in Java)
  - Access all fields through properties / methods

Exceeded Scope – Example

```
public class Globals
{
    public static int state = 0;
}
public class ConsolePrinter
{
    public static void PrintSomething()
    {
        if (Globals.state == 0)
        {
            Console.WriteLine("Hello.");
         }
        else
        {
            Console.WriteLine("Good bye.");
        }
    }
}
```

Span of Variables

- Variable span
  - The of lines of code (LOC) between variable usages
  - Average span can be calculated for all usages
  - Variable span should be kept as low as possible
  - Define variables at their first usage, not earlier
  - Initialize variables as late as possible
  - Try to keep together lines using the same variable

Always define and initialize variables just before their first use!

Calculating Span of Variable

```
1   a = 1;
2   b = 1;          } span = 1
3   c = 1;          }
4   b = a + 1;      } span = 0
5   b = b / c;      }
```

- One line between the first reference to b and the second
- There are no lines between the second reference to b and the third
- The average span for b is (1+0)/2 = 0.5

Variable Live Time

- Variable live time
  - The number of lines of code (LOC) between the first and the last variable usage in a block
  - Live time should be kept as low as possible
- The same rules apply as for minimizing span:
  - Define variables at their first usage
  - Initialize variables just before their first use
  - Try to keep together lines using the same variable

Measuring the Live Time of a Variable

```
25 recordIndex = 0;
26 while (recordIndex < recordCount) {
27 ...
28 recordIndex = recordIndex + 1;
...
62 total = 0;
63 done = false;
64 while ( !done ) {
...
69 if ( total > projectedTotal ) {
70 done = true;
```

recordIndex
(line 28-line 25 + 1) = 4

total
(line 69-line 62 + 1) = 8

done
(line 70-line 63 + 1) = 8

- Average live time for all variables:
  - (4 + 8 + 8) / 3 ≈ 7

Unneeded Large Variable Span and Live Time

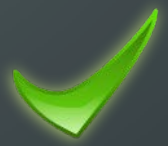
```
1  int count;
2  int[] numbers = new int[100];
3  for (int i = 0; i < numbers.Length; i++)
4  {
5    numbers[i] = i;
6  }
7  count = 0;
8  for (int i = 0; i < numbers.Length / 2; i++)
9  {
10   numbers[i] = numbers[i] * numbers[i];
11 }
12 for (int i = 0; i < numbers.Length; i++)
13 {
14   if (numbers[i] % 3 == 0)
15   {
16     count++;
17   }
18 }
19 Console.WriteLine(count);
```

live time = 19

span = (5+8+2) / 3 = 5

Reduced Span and Live Time

```
1  int[] numbers = new int[100];
2  for (int i = 0; i < numbers.Length; i++)
3  {
4    numbers[i] = i;
5  }
6  for (int i = 0; i < numbers.Length / 2; i++)
7  {
8    numbers[i] = numbers[i] * numbers[i];
9  }
10 int count = 0;
11 for (int i = 0; i < numbers.Length; i++)
12 {
13   if (numbers[i] % 3 == 0)
14   {
15     count++;
16   }
17 }
18 Console.WriteLine(count);
```

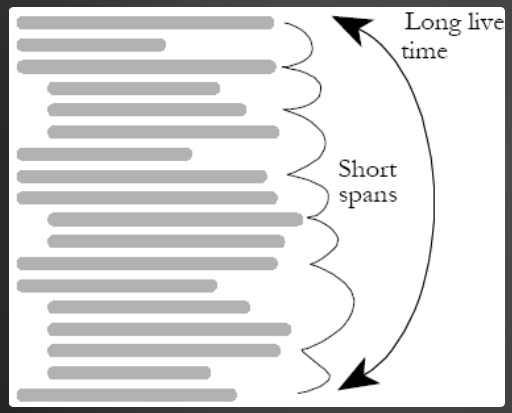


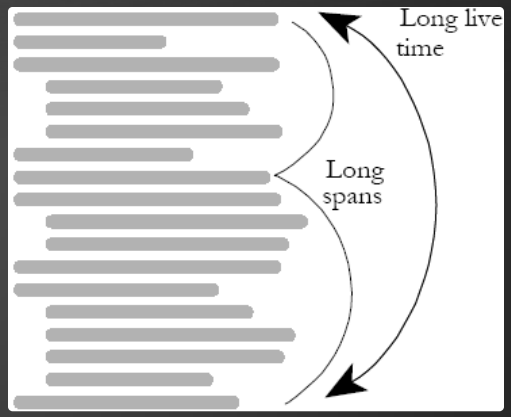
live time = 9

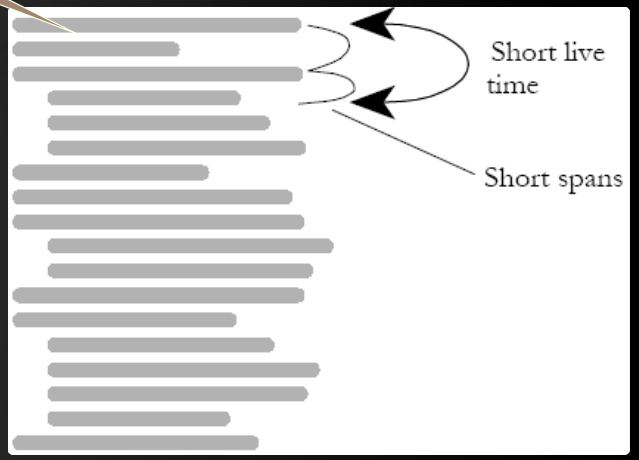
span=
(4+2) / 3 = 2

Keep Variables Live As Short a Time

The best case







- Advantages of short time and short span
  - Gives you an accurate picture of your code
  - Reduces the chance of initialization errors
  - Makes your code more readable

Best Practices

- Initialize variables used in a loop immediately before the loop
- Don't assign a value to a variable until just before the value is used
  - Never follow the old C / Pascal style of declaring variables in the beginning of each method
- Begin with the most restricted visibility
  - Expand the visibility only when necessary
- Group related statements together

Group Related Statements – Example

```
void SummarizeData(…)
{
  …
  GetOldData(oldData, numOldData);
  GetNewData(newData, numNewData);
  totalOldData = Sum(oldData, numOldData);
  totalNewData = Sum(newData, numNewData);
  PrintOldDataSummary(oldData, totalOldData);
  PrintNewDataSummary(newData, totalNewData);
  SaveOldDataSummary(totalOldData, numOldData);
  SaveNewDataSummary(totalNewData, numNewData);
  …
}
```

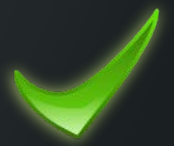
You have to keep track of oldData, newData, numOldData, numNewData, totalOldData and totalNewData

- Six variables for just this short fragment

Better Grouping– Example

```
void SummarizeDaily( … )
{
  GetOldData(oldData, numOldData);
  totalOldData = Sum(oldData, numOldData);
  PrintOldDataSummary(oldData, totalOldData);
  SaveOldDataSummary(totalOldData, numOldData);
  …
  GetNewData(newData, numNewData);
  totalNewData = Sum(newData, numNewData);
  PrintNewDataSummary(newData, totalNewData);
  SaveNewDataSummary(totalNewData, numNewData);
  …
}
```

The two blocks are each shorter and individually contain fewer variables



- Easier to understand, right?

Telerik Academy

Using Variables

Best Practices

Single Purpose

- Variables should have single purpose
  - Never use a single variable for multiple purposes!
  - Economizing memory is not an excuse
- Can you choose a good name for variable that is used for several purposes?
  - Example: variable used to count students or to keep the average of their grades
  - Proposed name: `studentsCountOrAvgGrade`

Variables Naming

- The name should describe the object clearly and accurately, which the variable represents
  - Bad names: `r18pq`, `__hip`, `rcfd`, `val1`, `val2`
  - Good names: `account`, `blockSize`, `customerDiscount`
- Address the problem, which the variable solves – "what" instead of "how"
  - Good names: employeeSalary, employees
  - Bad names: myArray, customerFile, customerHashTable

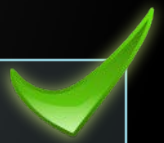
Telerik Academy

Poor and Good Variable Names

```
x = x - xx;
xxx = aretha + SalesTax(aretha);
x = x + LateFee(x1, x) + xxx;
x = x + Interest(x1, x);
```

- What do x1, xx, and xxx mean?
- What does aretha mean ?

```
balance = balance - lastPayment;
monthlyTotal = NewPurchases + SalesTax(newPurchases);
balance = balance + LateFee(customerID, balance) +
  monthlyTotal;
balance = balance + Interest(customerID, balance);
```



Naming Considerations

- Naming depends on the scope and visibility
  - Bigger scope, visibility, longer lifetime → longer and more descriptive name:

```
protected Account[] mCustomerAccounts;
```

  - Variables with smaller scope and shorter lifetime can be shorter:

```
for (int i=0; i<customers.Length; i++) { … }
```

- The enclosing type gives a context for naming:

```
class Account { Name: string { get; set; } }
// not AccountName
```

Optimum Name Length

- Somewhere between the lengths of `x` and `maximumNumberOfPointsInModernOlympics`
- Optimal length – 10 to 16 symbols
  - Too long

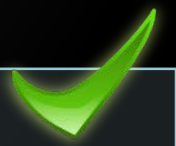
```
numberOfPeopleOfTheBulgarianOlympicTeamFor2012
```

  - Too short

```
a, n, z, щ
```

  - Just right

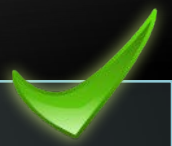
```
numTeamMembers, teamMembersCount
```



Naming Specific Data Types

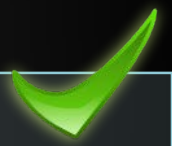
- Naming counters

```
UsersCount, RolesCount, FilesCount
```



- Naming variables for state

```
ThreadState, TransactionState
```

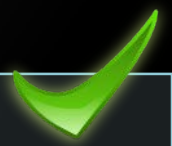


- Naming temporary variables
  - Bad examples:

```
a, aa, tmpvar1, tmpvar2
```

  - Good examples:

```
index, value, count
```

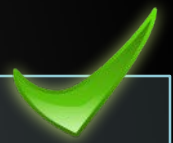


Telerik Academy

Naming Specific Data Types (2)

- Name Boolean variables with names implying "Yes / No" answers

```
canRead, available, isOpen, valid
```

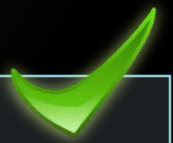


- Booleans variables should bring "truth" in their name
  - Bad examples:

```
notReady, cannotRead, noMoreData
```

  - Good examples:

```
isReady, canRead, hasMoreData
```



Telerik Academy

Naming Specific Data Types (3)

- Naming enumeration types
  - Use build in enumeration types (`Enum`)

```
Color.Red, Color.Yellow, Color.Blue
```

  - Or use appropriate prefixes (e.g. in JS / PHP)

```
colorRed, colorBlue, colorYellow
```

- Naming constants – use capital letters

```
MAX_FORM_WIDTH, BUFFER_SIZE
```

- C# constants should be in PascalCase:

```
Int32.MaxValue, String.Empty, InvariantCulture
```

Naming Convention

- Some programmers resist to follow standards and conventions
  - But why?
- Conventions benefits
  - Transfer knowledge across projects
  - Helps to learn code more quickly on a new project
  - Avoid calling the same thing by two different names

Naming Convention (2)

- When should we use a naming convention?
  - Multiple developers are working on the same project
  - The source code is reviewed by other programmers
  - The project is large
  - The project will be long-lived
- You always benefit from having some kind of naming convention!

Telerik Academy

Language-Specific Conventions

- C# and Java / JavaScript conventions
  - `i` and `j` are integer indexes
  - Constants are in `ALL_CAPS` separated by underscores (sometimes `PascalCase` in C#)
  - Variable and method names use uppercase in C# and lowercase in JS for the first word
  - The underscore _ is not used within names
    - Except for names in all caps

Standard Prefixes

- Hungarian notation – not used
- Semantic prefixes (ex. `btnSave`)
  - Better use `buttonSave`
- Do not miss letters to make name shorter
- Abbreviate names in consistent way throughout the code
- Create names, which can be pronounced (not like `btnDfltSvRzlts`)
- Avoid combinations, which form another word or different meaning (ex. `preFixStore`)

Kinds of Names to Avoid

- Document short names in the code
- Remember, names are designed for the people, who will read the code
  - Not for those who write it
- Avoid variables with similar names, but different purpose it

```
UserStatus / UserCurrentStatus
```

- Avoid names, that sounds similar

```
decree / degree / digRee
```

- Avoid digits in names

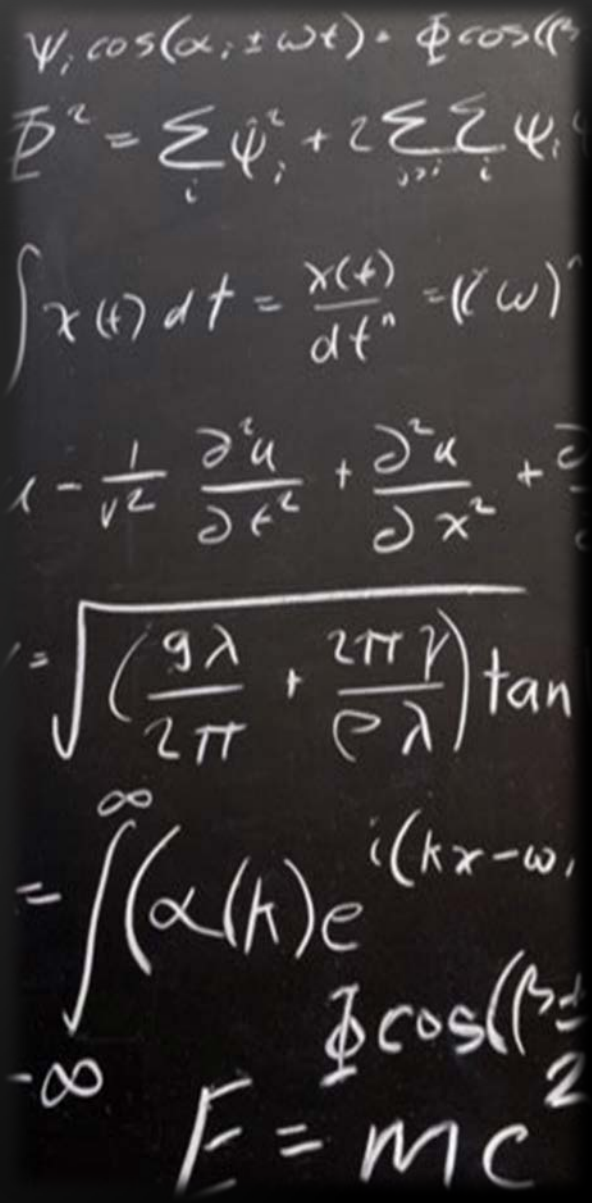
Kinds of Names to Avoid (2)

- Avoid words, which can be easily mistaken
  - E.g. `ads1`, `adc1`, `adct1`, `atc1`
- Avoid using non-English words
- Avoid using standard types and keywords in the variable names
  - E.g. `int`, `class`, `void`, `return`
- Do not use names, which has nothing common with variables content
- Avoid names, that contains hard-readable symbols / syllables, e.g. Csikszentmihalyi

Telerik Academy

Using Expressions

Best Practices



Avoid Complex Expressions

- Never use complex expressions in the code!
  - Incorrect example:

```
for (int i=0; i<xCoords.length; i++) {
  for (int j=0; j<yCoords.length; j++) {
    matrix[i][j] =
      matrix[xCoords[findMax(i)+1]][yCoords[findMin(j)-1]] *
      matrix[yCoords[findMax(j)+1]][xCoords[findMin(i)-1]];
  }
}
```

What shall we do if we get at this line `IndexOutOfRangeException`?

There are 10 potential sources of `IndexOutOfRangeException` in this expression!

- Complex expressions are evil because:
  - Make code hard to read and understand, hard to debug, hard to modify and hard to maintain

Telerik Academy

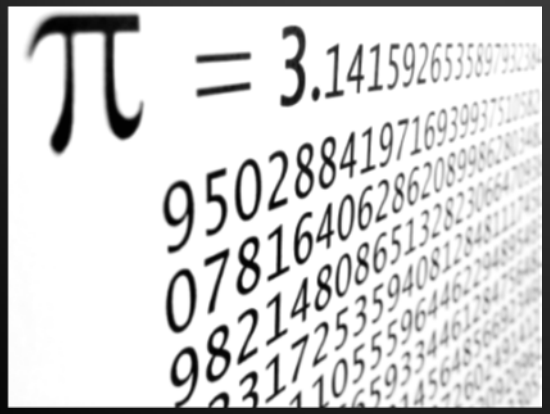
Simplifying Complex Expressions

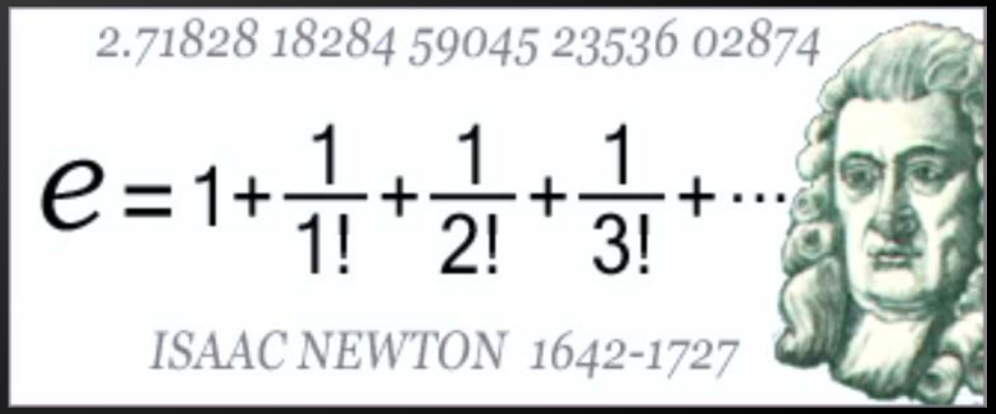
```
for (int i = 0; i < xCoords.length; i++)
{
  for (int j = 0; j < yCoords.length; j++)
  {
    int maxStartIndex = findMax(i) + 1;
    int minStartIndex = findMin(i) - 1;
    int minXcoord = xCoords[minStartIndex];
    int maxXcoord = xCoords[maxStartIndex];
    int minYcoord = yCoords[minStartIndex];
    int maxYcoord = yCoords[maxStartIndex];
    int newValue =
      matrix[maxXcoord][minYcoord] *
      matrix[maxYcoord][minXcoord];
    matrix[i][j] = newValue;
  }
}
```

Telerik Academy

Using Constants

When and How to Use Constants?





Telerik Academy

Avoid Magic Numbers and Strings

- What is magic number or value?
  - Magic numbers / values are all literals different than `0`, `1`, `-1`, `null` and `""` (empty string)
- Avoid using magic numbers / values

  - They are hard to maintain
    - In case of change, you need to modify all occurrences of the magic number / constant
  - Their meaning is not obvious
    - Example: what the number `1024` means?

The Evil Magic Numbers

```
public class GeometryUtils
{
  public static double CalcCircleArea(double radius)
  {
    double area = 3.14159206 * radius * radius;
    return area;
  }
  public static double CalcCirclePerimeter(double radius)
  {
    double perimeter = 6.28318412 * radius;
    return perimeter;
  }
  public static double CalcElipseArea(double axis1, double axis2)
  {
    double area = 3.14159206 * axis1 * axis2;
    return area;
  }
}
```

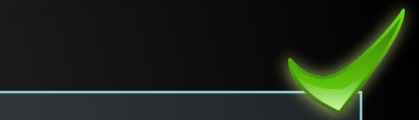
Turning Magic Numbers into Constants

```
public class GeometryUtils
{
    public const double PI = 3.14159206;
    public static double CalcCircleArea(double radius)
    {
        double area = PI * radius * radius;
        return area;
    }
    public static double CalcCirclePerimeter(double radius)
    {
        double perimeter = 2 * PI * radius;
        return perimeter;
    }
    public static double CalcElipseArea(
     double axis1, double axis2)
    {
        double area = PI * axis1 * axis2;
        return area;
    }
}
```

Constants in C#

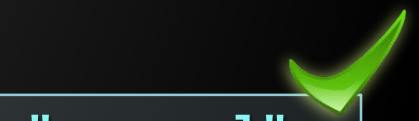
- There are two types of constants in C#
  - Compile-time constants:

```
public const double PI = 3.14159206;
```



    - Replaced with their value during compilation
    - No field stands behind them
  - Run-time constants:

```
public static readonly string ConfigFile = "app.xml";
```



    - Special fields initialized in the static constructor
    - Compiled into the assembly like any other class member

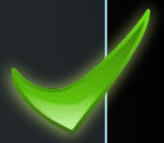
Constants in JavaScript

- JS does not support constants
  - Simulated by variables / fields in ALL_CAPS:

```
var PI = 3.14159206;

var CONFIG =
{
  COLOR : "#AF77EE",
  DEFAULT_WIDTH : 200,
  DEFAULT_HEIGHT : 300
};

document.getElementById("gameField").style.width =
  CONFIG.DEFAULT_WIDTH;
document.getElementById("gameField").style.
  backgroundColor = CONFIG.COLOR;
```



When to Use Constants?

- Constants should be used in the following cases:
  - When we need to use numbers or other values and their logical meaning and value are not obvious
  - File names

```
public static readonly string SettingsFileName =
  "ApplicationSettings.xml";
```

  - Mathematical constants

```
public const double E = 2.7182818284;
```

  - Bounds and ranges

```
public const int READ_BUFFER_SIZE = 5 * 1024 *1024;
```

When to Avoid Constants?

- Sometime it is better to keep the magic values instead of using a constant
  - Error messages and exception descriptions
  - SQL commands for database operations
  - Titles of GUI elements (labels, buttons, menus, dialogs, etc.)
- For internationalization purposes use resources, not constants
  - Resources are special files embedded in the assembly / JAR file, accessible at runtime

Telerik Academy

Correct Use of Variables, Data, Expressions and Constants

http://academy.telerik.com

Homework

1. Refactor the following code to use proper variable naming and simplified expressions:

```
public class Size
{
  public double wIdTh, Viso4ina;
  public Size(double w, double h)
  {
    this.wIdTh = w;
    this.Viso4ina = h;
  }
}

public static Size GetRotatedSize(
  Size s, double angleOfTheFigureThatWillBeRotaed)
{
  return new Size(
    Math.Abs(Math.Cos(angleOfTheFigureThatWillBeRotaed)) * s.wIdTh +
      Math.Abs(Math.Sin(angleOfTheFigureThatWillBeRotaed)) * s.Viso4ina,
    Math.Abs(Math.Sin(angleOfTheFigureThatWillBeRotaed)) * s.wIdTh +
      Math.Abs(Math.Cos(angleOfTheFigureThatWillBeRotaed)) * s.Viso4ina);
}
```

Homework (2)

2. Refactor the following code to apply variable usage and naming best practices:

```
public void PrintStatistics(double[] arr, int count)
{
    double max, tmp;
    for (int i = 0; i < count; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
    }
    PrintMax(max);

                                    // continue on the next slide
```

Homework (3)

```
    tmp = 0;
    max = 0;
    for (int i = 0; i < count; i++)
    {
        if (arr[i] < max)
        {
            max = arr[i];
        }
    }
    PrintMin(max);

    tmp = 0;
    for (int i = 0; i < count; i++)
    {
        tmp += arr[i];
    }
    PrintAvg(tmp/count);
}
```

Telerik Academy

Free Trainings @ Telerik Academy

- C# Programming @ Telerik Academy
  - csharpfundamentals.telerik.com

- Telerik Software Academy
  - academy.telerik.com

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com