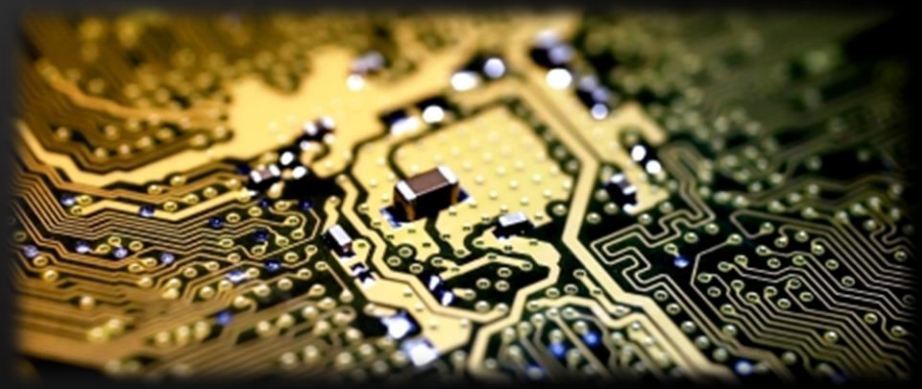
Telerik Academy

academy.telerik.com/.../software-engineering

High-Quality Programming Code Construction

Revealing the Secrets of Self-Documenting Code

Svetlin Nakov
Manager Technical Training
http://www.nakov.com
Telerik Software Academy
academy.telerik.com



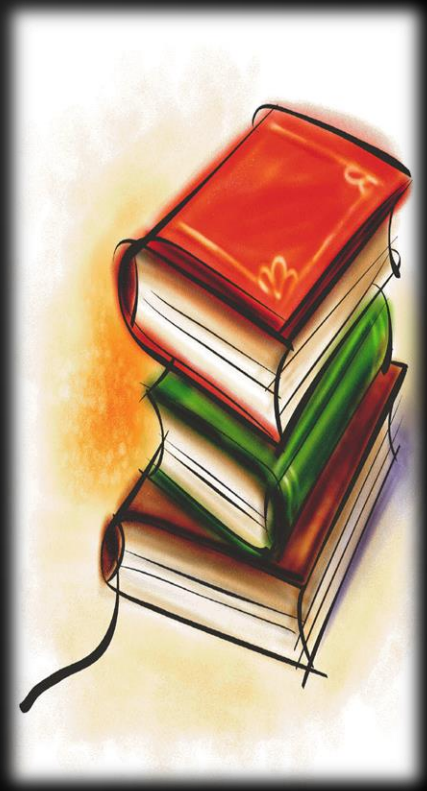
Telerik Academy

Table of Contents

Telerik Academy

Table of Contents (2)

- Using Constants
- Using Control-Flow Constructs Correctly
  - Conditional Statements
  - Loops
- Defensive Programming
  - Assertions and Exceptions
- Comments and Documentation
- Code Refactoring



Telerik Academy

What is High-Quality Programming Code?

Telerik Academy

Why the Code Quality Is Important?

```
static void Main()
{
  int value=010, i=5, w;
  switch(value){case
10:w=5;Console.WriteLine(w);break;case 9:i=0;break;
          case 8:Console.WriteLine("8 ");break;
     default:Console.WriteLine("def ");{
               Console.WriteLine("hoho ");     }
     for (int k = 0; k < i; k++, Console.WriteLine(k -
'f'));break;} { Console.WriteLine("loop!"); }
}
```

What does this code do? Is it correct?

Why the Code Quality Is Important? (2)

```
static void Main()
{
  int value = 010, i = 5, w;
  switch (value)
  {
    case 10: w = 5; Console.WriteLine(w); break;
    case 9: i = 0; break;
    case 8: Console.WriteLine("8 "); break;
    default:
      Console.WriteLine("def ");
      Console.WriteLine("hoho ");
      for (int k = 0; k < i; k++, Console.WriteLine(k -'f'));
      break;
    }
    Console.WriteLine("loop!");
}
```

Now the code is formatted, but is still unclear.

Software Quality

- External quality
  - Does the software behave correctly?
  - Are the produced results correct?
  - Does the software run fast?
  - Is the software UI easy-to-use?

- Internal quality
  - It the code easy to read and understand?
  - It the code well structured?
  - Is the code easy to modify?

What is High-Quality Programming Code?

- High-quality programming code:
  - Easy to read and understand
    - Easy to modify and maintain
  - Correct behavior in all cases
    - Well tested
  - Well architectured and designed
  - Well documented
    - Self-documenting code
  - Well formatted

Code Conventions

- Code conventions are formal guidelines about the style of the source code:
  - Code formatting conventions
    - Indentation, whitespace, etc.
  - Naming conventions
    - `PascalCase` or `camelCase`, prefixes, suffixes, …
  - Best practices
    - Classes, interfaces, enumerations, structures, inheritance, exceptions, properties, events, constructors, fields, operators, etc.

Code Conventions (2)

- Microsoft has official code conventions called
  - Design Guidelines for Developing Class Libraries: http://msdn.microsoft.com/en-us/library/ms229042.aspx
- Large organizations follow strict conventions
  - Code conventions can vary in different teams
  - Most conventions developers follow the official Microsoft's recommendations but extend them
- High-quality code goes beyond code conventions
  - Software quality is not just a set of conventions – its is a way of thinking

Telerik Academy

Naming Identifiers

Naming Classes, Interfaces, Enumerations, Methods, Variables and Constants

General Naming Guidelines

- Always use English
  - How you will feel if you read Vietnamese code with variables named in Vietnamese?
  - English is the only language that all software developers speak
- Avoid abbreviations
  - Example: `scrpCnt` vs. `scriptsCount`
- Avoid hard-to-pronounce names
  - Example: `dtbgRegExPtrn` vs. `dateTimeBulgarianRegExPattern`

Use Meaningful Names

- Always prefer using meaningful names
  - Names should answer these questions:
    - *What does this class do? What is the intent of this variable? What is this variable / class used for?*
  - Examples:
    - `FactorialCalculator`, `studentsCount`, `Math.PI`, `configFileName`, `CreateReport`
  - Incorrect examples:
    - `k`, `k2`, `k3`, `junk`, `f33`, `KJJ`, `button1`, `variable`, `temp`, `tmp`, `temp_var`, `something`, `someValue`

Fake Meaningful Names

- Junior developers often use meaningful names that are in fact meaningless
  - Bad naming examples:
    - `Topic6Exercise12`, `LoopsExercise12`, `Problem7`, `OOPLecture_LastExercise`
  - Yes, `Topic6Exercise12` indicates that this is solution to exercise 12, but what is it about?
  - Better naming:
    - `MaximalNumbersSubsequence`

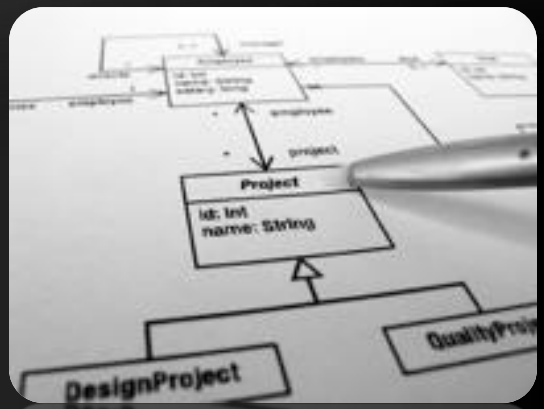
Naming Types

- Naming types (classes, structures, etc.)
  - Use `PascalCase` character casing
  - Examples:
    - `RecursiveFactorialCalculator`, `TreeSet`, `XmlDocument`, `IEnumerable`, `Color`, `TreeNode`, `InvalidTransactionException`, `MainForm`
  - Incorrect examples:
    - `recursiveFactorialCalculator`, `recursive_factorial_calculator`, `RECURSIVE_FACTORIAL_CALCULATOR`

Telerik Academy

Naming Classes and Structures

- Use the following formats:
  - [Noun]
  - [Adjective] + [Noun]
- Examples:
  - `Student`, `FileSystem`, `BinaryTreeNode`, `Constants`, `MathUtils`, `TextBox`, `Calendar`
- Incorrect examples:
  - `Move`, `FindUsers`, `Fast`, `Optimize`, `Extremly`, `FastFindInDatabase`, `Check`



Naming Interfaces

- Following formats are acceptable:
  - 'I' + [Verb] + 'able'
  - 'I' + [Noun], 'I' + [Adjective] + [Noun]

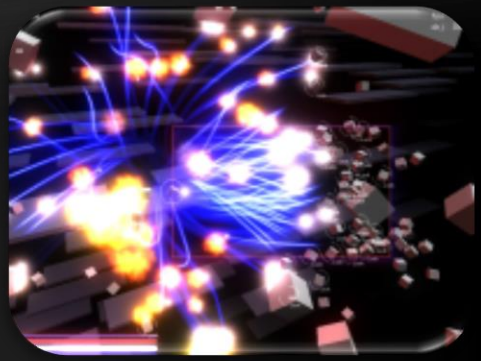
- Examples:
  - `IEnumerable`, `IFormattable`, `IDataReader`, `IList`, `IHttpModule`, `ICommandExecutor`
- Incorrect examples:
  - `List`, `FindUsers`, `IFast`, `IMemoryOptimize`, `Optimizer`, `FastFindInDatabase`, `CheckBox`

Naming Enumerations

- Several formats are acceptable:
  - [Noun] or [Verb] or [Adjective]
- Use the same style for all members
- Examples:
  - `enum Days {Monday, Tuesday, Wednesday, …}, enum AppState {Running, Finished, …}`
- Incorrect examples:
  - `enum Color {red, green, blue, white}, enum PAGE_FORMAT {A4, A5, A3, LEGAL, …}`

Naming Special Classes

- Exceptions
  - Add `'Exception'` as suffix
  - Use informative name
  - Example: `FileNotFoundException`
  - Incorrect example: `FileNotFoundError`



- Delegate Classes
  - Add `'Delegate'` or `'EventHandler'` as suffix
  - Example: `DownloadFinishedDelegate`
  - Incorrect example: `WakeUpNotification`

The Length of Class Names

- How long could be the name of a class / struct / interface / enum?
  - The name should be as long as required
  - Don't abbreviate the names if this could make them unclear
  - Your IDE has autocomplete, right?

- Examples: `FileNotFoundException`, `CustomerSupportNotificationService`
- Incorrect examples: `FNFException`, `CustSuppNotifSrvc`

Naming Projects / Folders

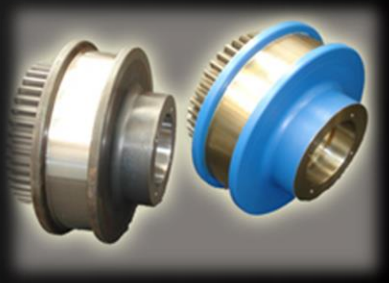
- Project naming guidelines
  - Use PascalCase
- Following formats are preferable:
  - `ProductName`
  - `Company [., -, _] ProductName`
- Name project folders to match the project name / its component names
- Examples: `Sitefinity`, `Telerik JustCode`
- Incorrect example: `ConsoleApplication17`

Naming Namespaces

- Namespaces naming guidelines
  - Use PascalCase
- Following formats are acceptable:
  - Company . Product . Component . …
  - Product . Component . …
- Example:
  - `Telerik.WinControls.GridView`
- Incorrect example:
  - `Telerik_WinControlsGridView`, `Classes`

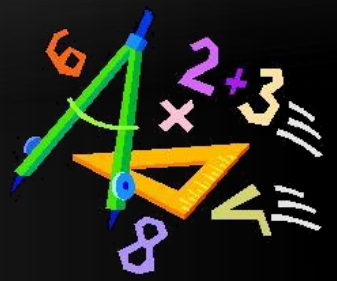
Naming Assemblies

- Assembly names should follow the root namespace in its class hierarchy
- Examples:
  - `Telerik.WinControls.GridView.dll`
  - `Oracle.DataAccess.dll`
  - `Interop.CAPICOM.dll`
- Incorrect examples:
  - `Telerik_WinControlsGridView.dll`
  - `OracleDataAccess.dll`



Naming Methods

- Methods naming guidelines
  - Method names should be meaningful
  - Should answer the question:
    - What does this method do?
  - If you cannot find a good name for a method, think about does it have clear intent
- Examples: `FindStudent`, `LoadReport`, `Sin`
- Incorrect examples: `Method1`, `DoSomething`, `HandleStuff`, `SampleMethod`, `DirtyHack`



Naming Methods (2)

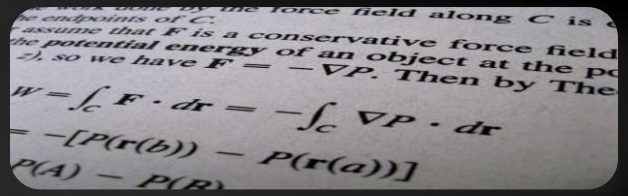
- Use PascalCase character casing
  - Example: `LoadSettings`, not `loadSettings`
- Prefer the following formats:
  - [Verb]
  - [Verb] + [Noun], [Verb] + [Adjective] + [Noun]

- Examples: `Show`, `LoadSettingsFile`, `FindNodeByPattern`, `ToString`, `PrintList`
- Incorrect examples: `Student`, `Counter`, `White`, `Generator`, `Approximation`, `MathUtils`

Telerik Academy

Methods Returning a Value

- Methods returning values should describe the returned value
- Examples:



  - `ConvertMetersToInches`, not `MetersInches` or `Convert` or `ConvertUnit`
  - `Meters2Inches` is still acceptable
  - `CalculateSin` is good
    - `Sin` is still acceptable (like in the `Math` class)
  - Ensure that the unit of measure of obvious
    - Prefer `MeasureFontInPixels` to `MeasureFont`

Telerik Academy

Single Purpose of All Methods

- Methods should have a single purpose!
  - Otherwise they cannot be named well
  - How to name a method that creates annual incomes report, downloads updates from internet and scans the system for viruses?
  - `CreateAnnualIncomesReportDownloadUpdatesAndScanForViruses` is a nice name, right?
- Methods that have multiple purposes (weak cohesion) are hard to be named
  - Need to be refactored instead of named

Telerik Academy

Consistency in Methods Naming

- Use consistent naming in the entire project
  - `LoadFile`, `LoadImageFromFile`, `LoadSettings`, `LoadFont`, `LoadLibrary`, but not `ReadTextFile`
- Use consistently the opposites at the same level of abstraction:
  - `LoadLibrary` vs. `UnloadLibrary`, but not `FreeHandle`
  - `OpenFile` vs. `CloseFile`, but not `DeallocateResource`
  - `GetName` vs. `SetName`, but not `AssignName`

Telerik Academy

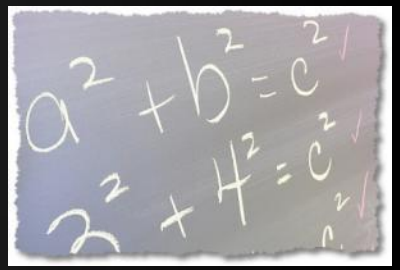
The Length of Method Names

- How long could be the name of a method?
  - The name should be as long as required
  - Don't abbreviate
  - Your IDE has autocomplete
- Examples:
  - `LoadCustomerSupportNotificationService`, `CreateMonthlyAndAnnualIncomesReport`
- Incorrect examples:
  - `LoadCustSuppSrvc`, `CreateMonthIncReport`

Telerik Academy

Naming Method Parameters

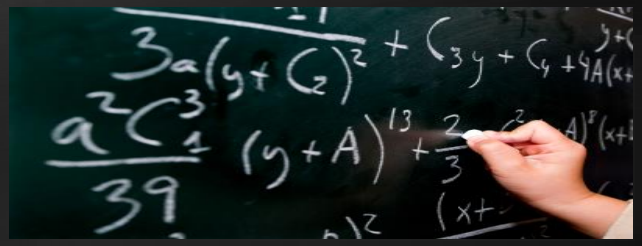
- Method parameters names
  - Preferred form: [Noun] or [Adjective] + [Noun]
  - Should be in camelCase
  - Should be meaningful
  - Unit of measure should be obvious



- Examples: `firstName`, `report`, `usersList`, `fontSizeInPixels`, `speedKmH`, `font`
- Incorrect examples: `p`, `p1`, `p2`, `populate`, `LastName`, `last_name`, `convertImage`

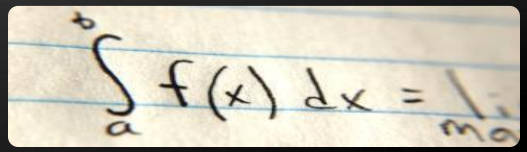
Naming Variables

- Variable names
  - Should be in `camelCase`
  - Preferred form: [Noun] or [Adjective] + [Noun]
  - Should explain the purpose of the variable
    - If you can't find good name for a variable check if it has a single purpose
    - Exception: variables with very small scope, e.g. the index variable in a 3-lines long for-loop
  - Names should be consistent in the project



Telerik Academy

Naming Variables – Examples



- Examples:
  - `firstName`, `report`, `usersList` , `fontSize`, `maxSpeed`, `font`, `startIndex`, `endIndex`, `charsCount`, `configSettingsXml`, `config`, `dbConnection`, `createUserSqlCommand`
- Incorrect examples:
  - `foo`, `bar`, `p`, `p1`, `p2`, `populate`, `LastName`, `last_name`, `LAST_NAME`, `convertImage`, `moveMargin`, `MAXSpeed`, `_firstName`, `__temp`, `firstNameMiddleNameAndLastName`

Telerik Academy

More about Naming Variables

- The name should be addressed to the problem we solve, not to the means we use to solve it
  - Prefer nouns from the business domain to computer terms
- Examples:
  - `accounts`, `customers`, `customerAddress`, `accountHolder`, `paymentPlan`, `vipPlayer`
- Incorrect examples:
  - `accountsLinkedList`, `customersHashtable`, `paymentsPriorityQueue`, `playersArray`

Naming Boolean Variables

- Boolean variables should imply `true` or `false`
- Prefixes like `is`, `has` and `can` are useful
- Use positive boolean variable names
  - Incorrect example:

```
if (! notFound) { … }
```

- Examples:
  - `hasPendingPayment`, `customerFound`, `validAddress`, `positiveBalance`, `isPrime`
- Incorrect examples:
  - `notFound`, `run`, `programStop`, `player`, `list`, `findCustomerById`, `isUnsuccessfull`

Telerik Academy

Naming Special Variables

- Naming Counters
  - Establish a convention, e.g. [Noun] + 'Count'
  - Examples: `ticketsCount`, `customersCount`
- State
  - Establish a convention, e.g. [Noun] + 'State'
  - Examples: `blogParseState`, `threadState`
- Variables with small scope and span
  - Short names can be used, e.g. `index`, `i`, `u`

Temporary Variables

- Do really temporary variables exist?
  - All variables in a program are temporary because they are used temporarily only during the program execution, right?
- Temporary variables can always be named better than `temp` or `tmp`:

```
// Swap a[i] and a[j]
int temp = a[i];
a[i] = a[j];
a[j] = temp;
```

```
// Swap a[i] and a[j]
int swapValue = a[i];
a[i] = a[j];
a[j] = swapValue;
```

Telerik Academy

The Length of Variable Names

- How long could be the name of a variable?
  - Depends on the variable scope and lifetime
  - More "famous" variables should have longer and more self-explaining name
- Acceptable naming examples:

```
for (int i=0; i<users.Length; i++)
  if (i % 2 == 0)
    sum += users[i].Weight;
```

```
class Student {
  public string lastName;
}
```

- Unacceptable naming examples:

```
class PairOfLists {
  public int Count { get; set; }
}
```

```
class Student {
  private int i;
}
```

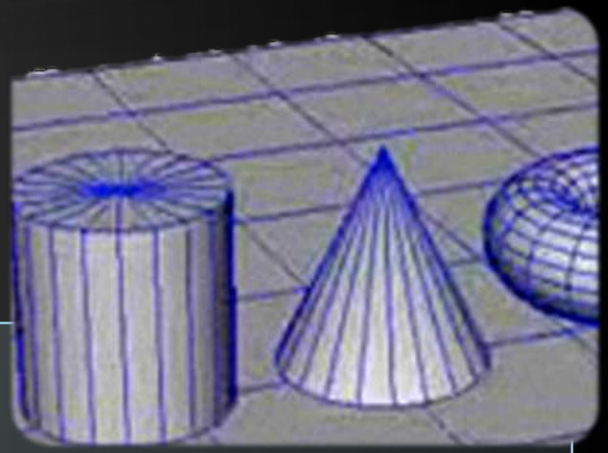
Telerik Academy

Prefixes / Suffixes and Hungarian Notation

- In C# prefix / suffix notations are not popular
- Hungarian notation
  - Using prefixes to denote the variable types, e.g. `lpcstrText`, `lpdwFlags`, `cbMultiByte`, `hWnd`
- The Hungarian notation works well in unmanaged languages like C++
  - Do not use Hungarian notation in C# and .NET
- Don't use prefixes / suffixes to denote the variable data type

Naming Properties

- Name properties in C# using a noun, noun phrase, or an adjective
- Use Pascal Case
- Examples:



```
public int Length { … }
public Color BackgroundColor { … }
public CacheMode CacheMode { … }
public bool Loaded { … }
```

- Incorrect examples:

```
public int Load { … }
public Color BackgroundColor { … }
public bool Font { … }
```

Naming Constants

- Use CAPITAL_LETTERS for `const` fields and PascalCase for `readonly` fields
- Use meaningful names that describe their value
- Examples:

```
private const int READ_BUFFER_SIZE = 8192;
public static readonly PageSize DefaultPageSize = PageSize.A4;
private const int FONT_SIZE_IN_POINTS = 16;
```

- Incorrect examples:

```
public const int MAX = 512; // Max what? Apples or Oranges?
public const int BUF256 = 256; // What about BUF256 = 1024?
public const string GREATER = "&gt;"; // GREATER_HTML_ENTITY
public const int FONT_SIZE = 16; // 16pt or 16px?
public const PageSize PAGE = PageSize.A4; // Maybe PAGE_SIZE?
```

Names to Avoid

- Don't use numbers in the identifiers names
  - Example:
    - `PrintReport` and `PrintReport2`
    - What is the difference?
  - Exceptions:
    - When the number is part of the name itself, e.g. `RS232Port`, `COM3`, `Win32APIFunctions`

- Don't use Cyrillic or letters from other alphabets
  - E.g. `FindСтудентByName`, `CalcΩ2Protein`

Telerik Academy

Never Give a Misleading Name!

- Giving a misleading name is worse than giving a totally unclear name
- Example:
  - Consider a method that calculates the sum of all elements in an array
  - It should be named `Sum` or `CalculateSum`
  - What about naming it `CalculateAverage` or `Max` or `CheckForNegativeNumber`?
  - It's crazy, but be careful with "copy-paste"

Don't Believe Microsoft!

- Microsoft sometimes use really bad naming in their API libraries (especially in Win32 API)
  - Examples:
    - `Navigate` and `Navigate2` methods in Internet Explorer ActiveX control (`MSHTML.DLL`)
    - `WNetAddConnection3` method in Multiple Provider Router Networking API (`MPR.DLL`)
    - `LPWKSTA_USER_INFO_1` structure in Win32
  - Don't follow them blindly, just think a bit!

Telerik Academy

What's Wrong with This Code?

```
FileStream fs = new FileStream(FILE_NAME, FileMode.CreateNew);
// Create the writer for data.
BinaryWriter w = new BinaryWriter(fs);
// Write data to Test.data.
for (int i = 0; i < 11; i++)
{
  w.Write( (int) i);
}
w.Close();
fs.Close();
// Create the reader for data.
fs = new FileStream(FILE_NAME, FileMode.Open, FileAccess.Read);
BinaryReader r = new BinaryReader(fs);
// Read data from Test.data.
for (int i = 0; i < 11; i++)
{
  Console.WriteLine(r.ReadInt32());
}
r.Close();
fs.Close();
```

Source: http://msdn.microsoft.com/en-us/library/36b93480.aspx

Naming Should Be Clear inside Its Context

- Naming should be clear inside its context
  - Method names should be clear in their class
    - Methods should not repeat their class name
  - Parameter names should be clear in the context of their method name
- Example of good class / property naming:

```
class Font
{
  public string Name { get; } // not FontName
}
```

Naming Should Be Clear inside Its Context (2)

- Example of good method & parameter naming:

```
double CalcDistance(Point p1, Point p2) { … }
```

  - Name like `CalcDistanceBetweenPoints` is excessive – parameters say "between points"

- Parameter names still can be improved:

```
double CalcDistance(Point firstPoint,
  Point secondPoint) { … }
```

- Name like `Calc` can still be correct:

```
class DistanceCalculator
{ static double Calc(Point p1, Point p1) { … } }
```

Naming Should Be Clear inside Its Context (3)

- Incorrect example of method naming:

```
class Program
{
  int Calculation(int value)
}
```

  - It is unclear in this context what kind of calculation is performed by the method

- Incorrect example of parameter naming:

```
class FileDownloader
{
  byte[] Download(string f);
}
```

What "f" means? An URL or FTP server or file path?

Telerik Academy

Code Formatting

Telerik Academy

Why Code Needs Formatting?

```
public   const    string                    FILE_NAME
="example.bin"  ;  static void Main   (            ){
FileStream   fs=     new FileStream(FILE_NAME,FileMode
.   CreateNew)   // Create the writer      for data  .
;BinaryWriter w=new BinaryWriter    (    fs      );//
Write data to                              Test.data.
for(  int i=0;i<11;i++){w.Write((int)i);}w   .Close();
fs   .   Close  (  ) // Create the reader    for data.
;fs=new FileStream(FILE_NAME,FileMode.             Open
,  FileAccess.Read)     ;BinaryReader                r
= new BinaryReader(fs);  // Read data from  Test.data.
 for (int i = 0; i < 11; i++){ Console      .WriteLine
(r.ReadInt32                                      ())
;}r       .    Close   (   );  fs .  Close  (  )  ;  }
```

Telerik Academy

Code Formatting Fundamentals

- Good formatting goals
  - To improve code readability
  - To improve code maintainability

- Fundamental principle of code formatting:

> The formating of the source code should disclose its logical structure.

  - Any formatting style that follows the above principle is good
  - Any other formatting is not good

Formatting Blocks in C#

- Put { and } alone on a line under the corresponding parent block
- Indent the block contents by a single [Tab]
  - Don't indent with spaces
- Example:

```
if ( some condition )
{
  // Block contents indented by a single [Tab]
  // Don't use spaces for indentation
}
```

Methods Indentation

- Methods should be indented with a single [Tab] from the class body
- Methods body should be indented with a single [Tab] as well

```
public class IndentationExample
{
    private int Zero()
    {
        return 0;
    }
}
```

The entire method is indented with a single [Tab]

Method body is also indented

Telerik Academy

Brackets in Methods Declaration

- Brackets in the method declaration should be formatted as follows:

```
private static ulong CalcFactorial(uint num)
```

- Don't use spaces between the brackets:

```
private static ulong CalcFactorial ( uint num )
```

```
private static ulong CalcFactorial (uint num)
```

- The same applies for `if`-conditions and `for`-loops:

```
if (condition) …
```

Separating Parameters

- Separate method parameters by comma followed by a space
  - Don't put comma before the space
  - Examples:

```
private void RegisterUser(string username, string password)
```

```
RegisterUser("nakov", "s3cr3t!p@ssw0rd");
```

  - Incorrect examples:

```
private void RegisterUser(string username,string password)
```

```
private void RegisterUser(string username ,string password)
```

```
private void RegisterUser(string username , string password)
```

Telerik Academy

Empty Lines between Methods

- Use empty line for separation between methods:

```
public class Factorial
{
  private static ulong CalcFactorial(uint num)
  {
    if (num == 0)
      return 1;
    else
      return num * CalcFactorial(num - 1);
  }

  static void Main()
  {
    ulong factorial = CalcFactorial(5);
    Console.WriteLine(factorial);
  }
}
```

Always use { and } after if (there is no space to do it here)

Empty line

Telerik Academy

Empty Lines in Method Body

- Use an empty line to separate logically related sequences of lines:

```
private List<Report> PrepareReports()
{
    List<Report> reports = new List<Report>();

    // Create incomes reports
    Report incomesSalesReport = PrepareIncomesSalesReport();
    reports.Add(incomesSalesReport);
    Report incomesSupportReport = PrepareIncomesSupportReport();
    reports.Add(incomesSupportReport);

    // Create expenses reports
    Report expensesPayrollReport = PrepareExpensesPayrollReport();
    reports.Add(expensesPayrollReport);
    Report expensesMarketingReport = PrepareExpensesMarketingReport();
    reports.Add(expensesMarketingReport);

    return reports;
}
```

Empty line

Empty line

Empty line

Misplaced Empty Lines – Example

```
public static void PrintList(List<int> ints)
{
  Console.Write("{ ");
  foreach (int item in ints)
  {
    Console.Write(item);

    Console.Write(" ");



  }
  Console.WriteLine("}");
}
static void Main()
{
  // ...
}
```

Formatting Types

- Formatting classes / structures / interfaces / enumerations
  - Indent the class body with a single [Tab]
  - Use the following order of definitions:
    - Constants, delegates, inner types, fields, constructors, properties, methods
    - Static members, public members, protected members, internal members, private members
  - The above order of definitions is not the only possible correct one

Formatting Types – Example

```
public class Dog
{
  // Static variables

  public const string SPECIES =
    "Canis Lupus Familiaris";

  // Instance variables

  private int age;

  // Constructors

  public Dog(string name, int age)
  {
    this.Name = name;
    this.age = age;
  }
```

(continues on the next slide)

Telerik Academy

Formatting Types – Example (2)

```
    // Properties

    public string Name { get; set; }

    // Methods

    public void Breath()
    {
        // TODO: breathing process
    }

    public void Bark()
    {
        Console.WriteLine("wow-wow");
    }

}
```

Telerik Academy

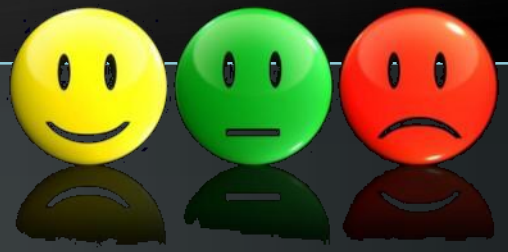
Formatting Conditional Statements and Loops

- Formatting conditional statements and loops
  - Always use { } block after `if` / `for` / `while`, even when a single operator follows
  - Indent the block body after `if` / `for` / `while`
  - Never put the block after `if` / `for` / `while` on the same line!
  - Always put the { on the next line
  - Never indent with more than one [Tab]

Telerik Academy

Conditional Statements and Loops Formatting – Examples

- Example:



```
for (int i=0; i<10; i++)
{
  Console.WriteLine("i={0}", i);
}
```

- Incorrect examples:

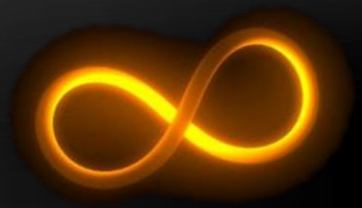
```
for (int i=0; i<10; i++)
  Console.WriteLine("i={0}", i);
```

```
for (int i=0; i<10; i++) Console.WriteLine("i={0}", i);
```

```
for (int i=0; i<10; i++) {
  Console.WriteLine("i={0}", i);
}
```

Breaking Long Lines

- Break long lines after punctuation
- Indent the second line by single [Tab]
- Do not additionally indent the third line
- Examples:



```
if (matrix[x, y] == 0 || matrix[x-1, y] == 0 ||
  matrix[x+1, y] == 0 || matrix[x, y-1] == 0 ||
  matrix[x, y+1] == 0)
{ …
```

```
DictionaryEntry<K, V> newEntry =
  new DictionaryEntry<K, V>(
  oldEntry.Key, oldEntry.Value);
```

Incorrect Ways To Break Long Lines

```
if (matrix[x, y] == 0 || matrix[x-1, y] ==
  0 || matrix[x+1, y] == 0 || matrix[x,
  y-1] == 0 || matrix[x, y+1] == 0)
{ …
```


```
if (matrix[x, y] == 0 || matrix[x-1, y] == 0 ||
      matrix[x+1, y] == 0 || matrix[x, y-1] == 0 ||
      matrix[x, y+1] == 0)
{ …
```

```
DictionaryEntry<K, V> newEntry
  = new DictionaryEntry<K, V>(oldEntry
  .Key, oldEntry.Value);
```

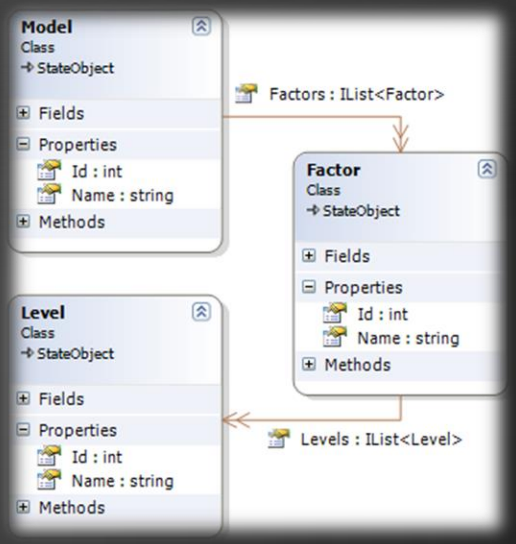
Alignments

- All types of alignments are considered harmful
  - Alignments are hard-to-maintain!
- Incorrect examples:

```
DateTime       date     = DateTime.Now.Date;
int            count    = 0;
Student        student  = new Strudent();
List<Student> students = new List<Student>();
```

```
matrix[x, y]                 == 0;
matrix[x + 1, y + 1]         == 0;
matrix[2 * x + y, 2 * y + x] == 0;
matrix[x * y, x * y]         == 0;
```

Telerik Academy



High-Quality Classes

How to Design High-Quality Classes? Abstraction, Cohesion and Coupling

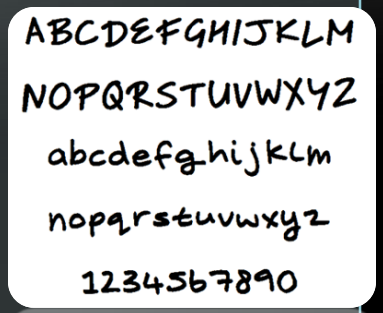
Telerik Academy

High-Quality Classes: Abstraction

- Present a consistent level of abstraction in the class contract (publicly visible members)
  - What abstraction the class is implementing?
  - Does it represent only one thing?
  - Does the class name well describe its purpose?
  - Does the class define clear and easy to understand public interface?
  - Does the class hide all its implementation details?

Good Abstraction – Example

```
public class Font
{
  public string Name { get; set; }
  public float SizeInPoints { get; set; }
  public FontStyle Style { get; set; }
  public Font(string name, float sizeInPoints,
    FontStyle style)
  {
    this.Name = name;
    this.SizeInPoints = sizeInPoints;
    this.Style = style;
  }
  public void DrawString(DrawingSurface surface,
    string str, int x, int y)
  { ... }
  public Size MeasureString(string str)
  { ... }
}
```

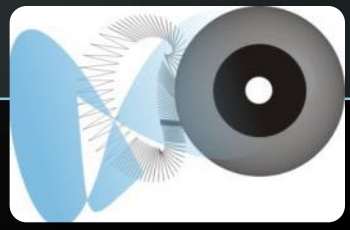


Bad Abstraction – Example

```
public class Program
{
  public string title;
  public int size;
  public Color color;
  public void InitializeCommandStack();
  public void PushCommand(Command command);
  public Command PopCommand();
  public void ShutdownCommandStack();
  public void InitializeReportFormatting();
  public void FormatReport(Report report);
  public void PrintReport(Report report);
  public void InitializeGlobalData();
  public void ShutdownGlobalData();
}
```

Does this class really represent a "program"? Is this name good?

Does this class really have a single purpose?



Telerik Academy

Establishing Good Abstraction

- Define operations along with their opposites
  - Example:
    - `Open()` and `Close()`
- Move unrelated methods in another class
  - Example:
    - In class `Employee` if you need to calculate `Age` by given `DateOfBirth`
    - Create static method `CalcAge` in a separate class `DateUtils`

Telerik Academy

Establishing Good Abstraction (2)

- Beware of breaking the interface abstraction due to evolution
  - Don't add public members inconsistent with class abstraction
  - Example: in class called Employee at some time we add method for accessing the DB with SQL

```
class Employee
{
  public string firstName;
  public string lastName;
  …
  public SqlCommand FindByPrimaryKeySqlCommand(int id);
}
```

Encapsulation

- Minimize the visibility of classes and members
  - Start from `private` and move to `internal`, `protected` and `public` if required
- Classes should hide their implementation details
  - Anything which is not part of the class interface should be declared `private`
  - Classes with good encapsulated classes are: less complex, easier to maintain, more loosely coupled
- Never declare fields public (except constants)
  - Use methods or properties to access fields

Encapsulation (2)

- Don't violate encapsulation semantically!
  - Don't rely on non-documented internal behavior or side effects
  - Wrong example:
    - Skip calling `ConnectToDB()` because you just called `FindEmployeeById()` which should open connection
  - Another wrong example:
    - Use `String.Empty` instead of `Titles.NoTitle` because you know both values are the same

Inheritance

- Don't hide methods in a subclass
  - Example: if the class `Timer` has private method `Start()`, don't define `Start()` in `AtomTimer`
- Move common interfaces, data, and behavior as high as possible in the inheritance tree
  - This maximizes the code reuse
- Be suspicious of base classes of which there is only one derived class
  - Do you really need this additional level of inheritance?

Inheritance (2)

- Be suspicious of classes that override a routine and do nothing inside
  - Is the overridden routine used correctly?
- Avoid deep inheritance trees
  - Don't create more than 6 levels of inheritance
- Avoid using a base class's protected data fields in a derived class
  - Provide protected accessor methods or properties instead

Inheritance (3)

- Prefer inheritance to extensive type checking:

```
switch (shape.Type)
{
  case Shape.Circle:
    shape.DrawCircle();
    break;
  case Shape.Square:
    shape.DrawSquare();
    break;
  ...
}
```

- Consider inheriting `Circle` and `Square` from `Shape` and override the abstract action `Draw()`

Class Methods and Data

- Keep the number of methods in a class as small as possible → reduce complexity
- Minimize direct method calls to other classes
  - Minimize indirect method calls to other classes
  - Less external method calls == less coupling
- Minimize the extent to which a class collaborates with other classes
  - Reduce coupling between classes

Class Constructors

- Initialize all member data in all constructors, if possible
  - Uninitialized data is error prone
  - Partially initialized data is even more evil
  - Incorrect example: assign `FirstName` in class `Person` but leave `LastName` empty
- Initialize data members in the same order in which they are declared
- Prefer deep copies to shallow copies (`ICloneable` should make deep copy)

Use Design Patterns

- Use private constructor to prohibit direct class instantiation
- Use deign patterns for common design situations
  - Creational patterns like singleton, factory method, abstract factory
  - Structural patterns like adapter, bridge, composite, decorator, façade
  - Behavioral patterns like command, iterator, observer, strategy, template method

Top Reasons to Create Class

- Model real-world objects with OOP classes
- Model abstract objects, processes, etc.
- Reduce complexity
  - Work at higher level
- Isolate complexity
  - Hide it in a class
- Hide implementation details → encapsulation
- Limit effects of changes
  - Changes affect only their class

Telerik Academy

Top Reasons to Create Class (2)

- Hide global data
  - Work through methods
- Group variables that are used together
- Make central points of control
  - Single task should be done at single place
  - Avoid duplicating code
- Facilitate code reuse
  - Use class hierarchies and virtual methods
- Package related operations together

Namespaces

- Group related classes together in namespaces
- Follow consistent naming convention

```
namespace Utils
{
    class MathUtils { … }
    class StringUtils { … }
}

namespace DataAccessLayer
{
    class GenericDAO<Key, Entity> { … }
    class EmployeeDAO<int, Employee> { … }
    class AddressDAO<int, Address> { … }
}
```

Telerik Academy

High-Quality Methods

How to Design and Implement High-Quality Methods? Understanding Cohesion and Coupling

Why We Need Methods?

- Methods are important in software development
  - Reduce complexity
    - Divide and conquer: complex problems can be split into composition of several simple ones
  - Improve code readability
    - Small methods with good method names make the code self-documenting
  - Avoid duplicating code
    - Duplicating code is hard to maintain

Why We Need Methods? (2)

- Methods simplify software development
  - Hide implementation details
    - Complex logic is encapsulated and hidden behind a simple interface
    - Algorithms and data structures are hidden and can be transparently replaced later
  - Increase the level of abstraction
    - Methods address the business problem, not the technical implementation:

```
Bank.accounts[customer].deposit(500);
```