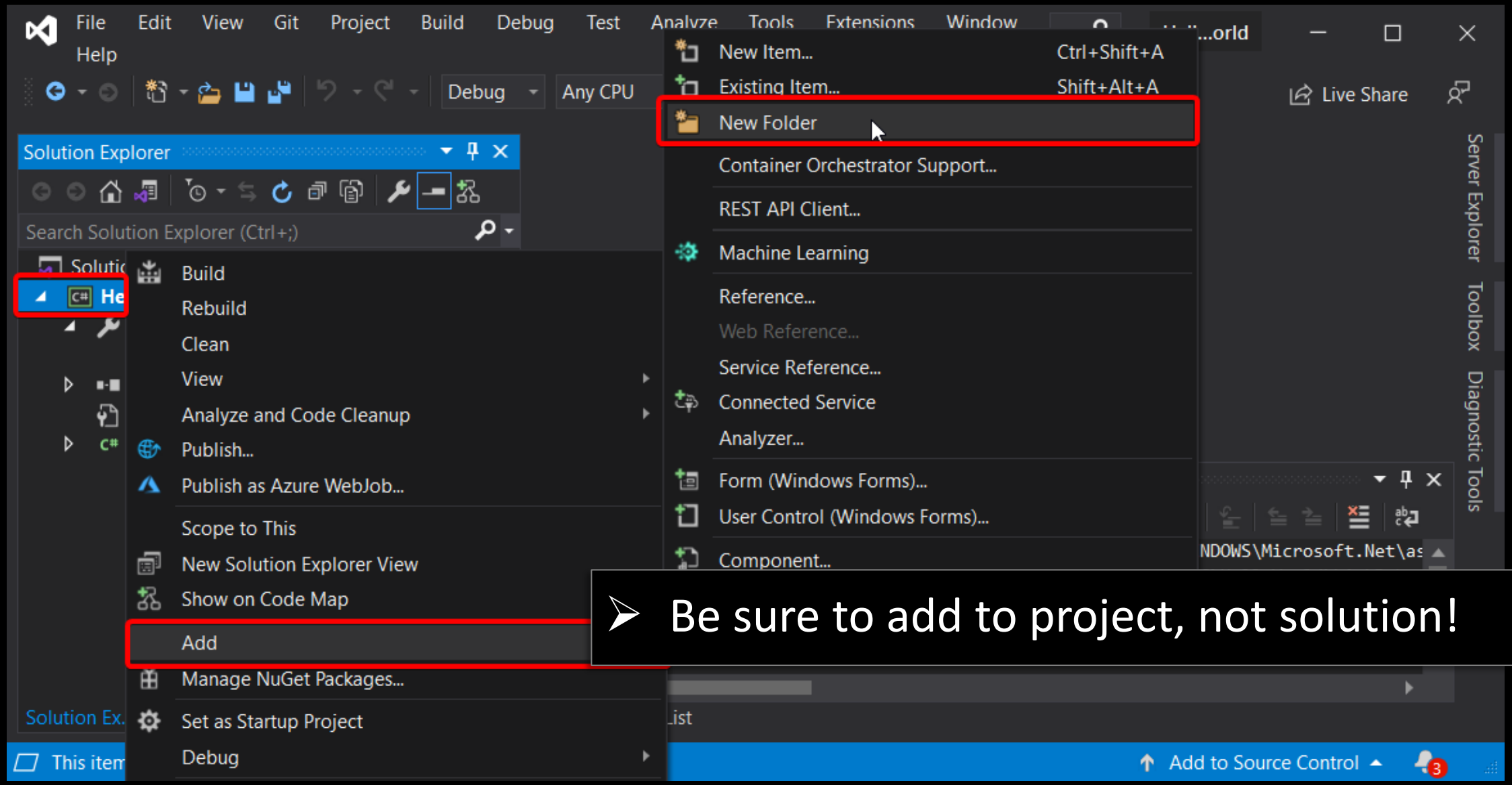
TOOL DEVELOPMENT

OBJECT ORIENTED PROGRAMMING IN C#

C# syntax – hands on

CREATE FOLDER ‘Model’



➢ Be sure to add to project, not solution!

C# syntax – hands on

CREATE CLASS ‘Hero’ INSIDE Model FOLDER

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help

Hell...orld

Debug | Any CPU | Start | Live Share

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'HelloWorld' (1 of 1 project)
  - HelloWorld
    - Properties
    - References
    - Model
    - App.confi
    - Program.c

Solution Ex... | Class V

This item does not

Add
- Scope to This
- New Solution Explorer View
- Show on Code Map
- Exclude From Project
- Cut Ctrl+X
- Copy Ctrl+C
- Paste Ctrl+V
- Delete Del
- Rename F2
- Copy Full Path

New Item... Ctrl+Shift+A
Existing Item... Shift+Alt+A
New Folder
Container Orchestrator Support...
REST API Client...
Machine Learning
Form (Windows Forms)...
User Control (Windows Forms)...
Component...
Class...

INDOWS\Microsoft.Net\as
RC\HelloWorld\bin\Debug
with code 0 (0x0).
3741510 (0xc000013a).

dd to Source Control

Server Explorer | Toolbox | Diagnostic Tools

C# CLASS

BASICS

```
public class Hero
{
    // FIELD (called data member in C++)
    private string _nemesis = string.Empty;

    // PROPERTY (Getter and/or Setter)
    0 references
    public string Nemesis
    {
        get { return _nemesis; }
        set { _nemesis = value; }
    }

    // AUTO-PROPERTY
    0 references
    public int Health { get; set; }

    // CONSTRUCTOR
    0 references
    public Hero()
    {
    }

    // NO DESTRUCTOR IS NEEDED!!!
    // METHOD (called member function in C++)
    0 references
    public void SaveTheWorld()
    {
    }

    0 references
    private void AskForHelp()
    {
    }
}
```

HEADERS & DESTRUCTORS

- No Headers
  - C# doesn't use header files
  - Why not?
    - .NET is compiled to an intermediate language, the assemblies contain all the information about the classes and methods in it

- No Destructors
  - C# is managed,
  - therefore the Garbage Collector (GC) handles all (de)allocations
  - No need for a destructor

- No need to use the → operator, instead use the . operator (DOT) to access the members

ACCESS SPECIFIERS

- Because of the lack of headers, every class, method, field and property must have an access specifier.

public	➢ Can be accessed from **anywhere**, same as C++
private	➢ Can **not** be accessed from **outside the class**, same as C++ ➢ Default access specifier for fields and properties
protected	➢ Can **not** be accessed from **outside the class**, **except by a child class**, same as C++
internal	➢ Can only be accessed by classes **in the same namespace** (ex. Library only) ➢ Default access specifier for a class in C# programming

PROPERTIES

MEMBER VARIABLES & EXPOSURE

```
public class Hero
{
    // FIELD (called data member in C++)
    private string _nemesis = string.Empty;

    // PROPERTY (Getter and/or Setter)
    0 references
    public string Nemesis
    {
        get { return _nemesis; }
        set { _nemesis = value; }
    }

    // AUTO-PROPERTY
    0 references
    public int Health { get; set; }

    // CONSTRUCTOR
    0 references
    public Hero()
    {
    }

    // NO DESTRUCTOR IS NEEDED!!!
    // METHOD (called member function in C++)
    0 references
    public void SaveTheWorld()
    {
    }

    0 references
    private void AskForHelp()
    {
    }
}
```

C# syntax

MEMBER VARIABLES?

```
//private field
private string _myNemises;

//public property to access field
0 references
public string MyNemises
{
    get { return _myNemises; }
    set { _myNemises = value; }
}
```

- Convention: _lowerCamelCase
- private
- Initialized by default

- Convention: UpperCamelCase
- public
- used to expose fields

```
public class Hero
{

    propfull
    propfull
}
```

propfull
Code snippet for property and backing field
Note: Tab twice to insert the 'propfull' snippet.

AUTO IMPLEMENTED PROPERTY

```
//public property (hidden field)
0 references
public int Health { get; set; }
```

- Convention: UpperCamelCase
- public
- auto generates (hidden) field

```
public class Hero
{
    prop
    prop
    propa
    propdp
    propfull
```

prop
Code snippet for an automatically implemented property
Language Version: C# 3.0 or higher
Note: Tab twice to insert the 'prop' snippet.

CONTROLLING INPUT / OUTPUT

- auto implemented property:

```
public int Health { get; private set; }
```

- full property:

```
//private field
private string _myNemises;

//public property to access field
3 references
public string MyNemises
{
    get
    {
        //extra check
        if (string.IsNullOrWhiteSpace(_myNemises))
            return "(unknown)";
        return _myNemises;
    }
    private set { _myNemises = value; }
}
```

ACCESS A PROPERTY

```
public class Hero
{
    //private field
    private string _myNemises;

    //public property to access field
    3 references
    public string MyNemises
    {
        get
        {
            //extra check
            if (string.IsNullOrWhiteSpace(_myNemises))
                return "(unknown)";
            return _myNemises;
        }
        set { _myNemises = value; }
    }
}
```

```
static void Main(string[] args)
{
    //declare + initialize
    Hero myHero = new Hero();

    //GET nemisis
    Console.WriteLine(myHero.MyNemises);

    //SET nemisis
    myHero.MyNemises = "Batman";

    //GET nemisis
    Console.WriteLine(myHero.MyNemises);
}
```

ACCESS A PROPERTY

```
public class Hero
{
    //private field
    private string _myNemises;

    //public property to access field
    3 references
    public string MyNemises
    {
        get
        {
            //extra c
            if (strin
                retur
            return _myNemises;
        }
        private set { _myNemises = value; }
    }
}
```

```
static void Main(string[] args)
{
    //declare + initialize
    Hero myHero = new Hero();

    //GET nemisis
    Console.WriteLine(myHero.MyNemises);

    //SET nemisis
    myHero.MyNemises = "Batman";
```

string Hero.MyNemises { get; private set; }

CS0272: The property or indexer 'Hero.MyNemises' cannot be used in this context because the set accessor is inaccessible

CONSTRUCTORS

PAREMETERS & CONSTRUCTOR OVERLOADING

```
ctor
ctor
1 reference
```

ctor
Code snippet for constructor
Note: Tab twice to insert the 'ctor' snippet.

```
public class Hero
{
    // FIELD (called data member in C++)
    private string _nemesis = string.Empty;

    // PROPERTY (Getter and/or Setter)
    0 references
    public string Nemesis
    {
        get { return _nemesis; }
        set { _nemesis = value; }
    }

    // AUTO-PROPERTY
    0 references
    public int Health { get; set; }

    // CONSTRUCTOR
    0 references
    public Hero()
    {
    }

    // NO DESTRUCTOR IS NEEDED!!!
    // METHOD (called member function in C++)
    0 references
    public void SaveTheWorld()
    {
    }

    0 references
    private void AskForHelp()
    {
    }
}
```

CONSTRUCTOR OVERLOADING

```
Hero hero = new Hero("Billy", 4);
println(hero.Health); //?

Hero myHero = new Hero("Bob",1, 13);
hero.MyNemises = "none! "; //not in ctor

Hero fastHero = new Hero("Fast",10)
        { MyNemises="Bob", Health=123 };
//object initializers:
        //call existing constructor,
        //add more properties between {},
        //using name = value
```

```
    }

    1 reference
    public int Health  { get; set; }
    2 references
    public int Level { get; private set; }

    1 reference
    public Hero(string name, int level)
        : this(name, level, 10)
    {
    }

    2 references
    public Hero (string name, int level, int health)
    {
        this.Level = level;
        this.Name = name;
        this.MyNemises = "";
        this.Health = health;
    }
}
```

METHODS

REMEMBER: NO DESTRUCTOR

```
public class Hero
{
    // FIELD (called data member in C++)
    private string _nemesis = string.Empty;

    // PROPERTY (Getter and/or Setter)
    0 references
    public string Nemesis
    {
        get { return _nemesis; }
        set { _nemesis = value; }
    }

    // AUTO-PROPERTY
    0 references
    public int Health { get; set; }

    // CONSTRUCTOR
    0 references
    public Hero()
    {
    }

    // NO DESTRUCTOR IS NEEDED!!!
    // METHOD (called member function in C++)
    0 references
    public void SaveTheWorld()
    {
    }

    0 references
    private void AskForHelp()
    {
    }
}
```

C# syntax

METHODS INHERITED FROM SYSTEM.OBJECT

Object Browser | Program.cs

Browse: My Solution

<Search>

- HelloWorld
  - HelloWorld
    - Program
      - Base Types
        - Object
- Microsoft.CSharp
- mscorlib
- System
  - Microsoft.CSharp
  - Microsoft.VisualBasic
  - Microsoft.Win32
  - Microsoft.Win32.SafeHandles
  - System
    - FileStyleUriParser
    - FtpStyleUriParser
    - GenericUriParser
    - GenericUriParserOptions
    - GopherStyleUriParser
    - HttpStyleUriParser
    - LdapStyleUriParser
    - NetPipeStyleUriParser
    - NetTcpStyleUriParser
    - NewsStyleUriParser
    - Uri

- ~Object()
- Equals(object)
- Equals(object, object)
- GetHashCode()
- GetType()
- MemberwiseClone()
- Object()
- ReferenceEquals(object, object)
- ToString()

public class **Object**

Member of System

Summary:

Supports all classes in the .NET Framework class hierarchy and provides low-

USING THE 'Hero' CLASS

- Print the Name of a Hero class instance:

```
Console.WriteLine(myHero.Name);
```

```
D:\SRC\HelloWorld\bin\Debug\HelloWorld.exe
Flash
```

- Print the whole object:

```
Console.WriteLine(myHero);
```

-

```
D:\SRC\HelloWorld\bin\Debug\HelloWorld.exe
HelloWorld.Model.Hero
```

C# syntax - exercise

PRINTING AN OBJECT

```
Console.WriteLine(myHero);
```

```
D:\SRC\HelloWorld\bin\Debug\HelloWorld.exe
HelloWorld.Model.Hero
```

- Every object inherits from (is a) System.Object
  - Inherits the ToString() function
  - This is automatically being called when printing an object
  - default: namespace.classname
  - override:

```
public class Hero
{
    //override default ToString function (< System.Object):
    0 references
    public override string ToString()
    {
        return $"This is a level {Level} hero called {Name}.";
    }
```

```
D:\SRC\HelloWorld\bin\Debug\HelloWorld.exe
This is a level 4 hero called Flash.
```

WORKING WITH STRING TYPE

STRING TYPE IN C#

STRING CONCATENATION

- using the + operator:

```
var myString = "Hello";
myString += " World!";
```

  - this creates a new string!!
  - reason: immutable (state of an object doesn't change after creation)
  - !! Modifying/Building a lot of strings can cause performance issues !! (Constant creation and Garbage collection)

- using string interpolation (→ C# 6 or higher):

```
string name = "Uzzi";
string firstName = "Jack";
int age = 42;
string description = $"Your character is called {firstName} {name}, and is {age} years old.";
Console.WriteLine(description);
```

USING THE STRING TYPE

- Check out StringBuilder
- Check out the System.String MSDN documentation page
- You can iterate over a string
- Find a lot of helpful static methods:
  - string.Format(...)
  - string.IsNullOrEmpty(...)
  - string.IsNullOrWhiteSpace(...)
  - ...

VALUE TYPES VS REFERENCE TYPES

WHAT, WHEN, WHERE, HOW

OBJECT TYPES IN C#

- C# has pointers,
- but you'll almost never use them (only in unsafe mode)

- No need to use the -→ operator,
  instead we use the . operator (DOT) to access the members

- Object types in C#:
  - Value Types
    - Structs, enumerations, numerical types, ...
  - Reference Types
    - Classes, arrays, string, ...

object types in C#

VALUE TYPES

- Holds the data in its own memory allocation
- Always passed by value
- (Pass by reference? Use the 'ref' keyword)

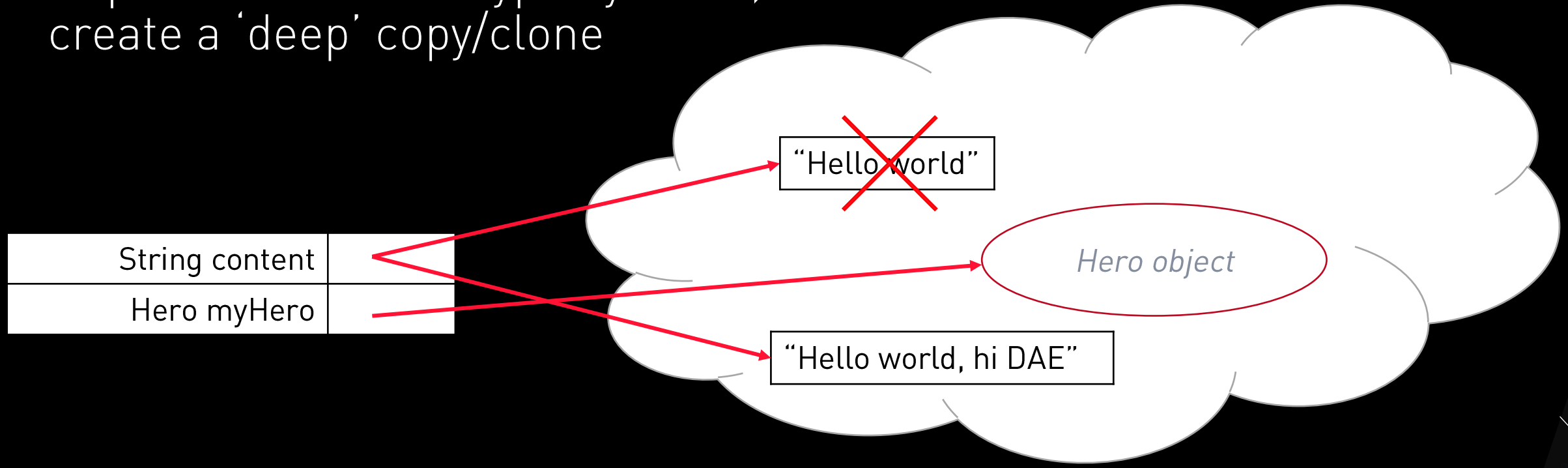
int wholeNumber	13
double myNumber	3.1415

```
1 reference
public void test()
{
    int number = 4;
    calculate(number, 2);
    //number value has not changed!
}

1 reference
public void calculate(int value, int multiplier)
{
    value = value * multiplier;
    //do something with the calculated value
}
```

REFERENCE TYPES

- Store a reference to the data (Sort of Managed Pointers)
- Always passed by reference in most cases
  - string → Immutable → reference is passed by value
  - Eg.: `content += ", hi DAE";`
- To pass a reference type by value, create a 'deep' copy/clone



REFERENCE TYPES: CODE EXAMPLE

```
static void Main(string[] args)
{
    Hero myHero = new Hero("Flash", 4);
    Console.WriteLine(myHero);
    upgradeHero(myHero); // change values
    Console.WriteLine(myHero);

    Console.ReadKey();
}

1 reference
private static void upgradeHero(Hero myHero)
{
    myHero.Name = "upgraded";
    myHero.Level += 4;
}
```

D:\SRC\HelloWorld\bin\Debug\HelloWorld.exe

```
This is a level 4 hero called Flash.
This is a level 8 hero called upgraded.
```

SO WHAT IF I WANT MY VALUE TYPE TO BE **NULL**?

- Value types can never be null (default values)

int wholeNumber	0
double myNumber	0.0

- Solution: **Nullable types**
  - `int? number = null;`
  - ~~`int notNullable = number;`~~
  - `int notNullable = number ?? 0;`

- Nullable types: value or reference type??
  - value type;
  - struct with 'HasValue' boolean that indicates if there is a value

C# CONVERSION AND PARSING

SWITCHING BETWEEN TYPES

STRING TO NUMBER

- Static methods in very .NET numerical Type
  - Parse
    - ```
      string input = ...;
      int nr = int.Parse(input);
      double d = double.Parse(input);
      ```
    - no check for valid input ➔ wrong format = crash!
  - TryParse (No check, can crash if the string has a wrong format)
    - ```
      string input = ...;
      int nr;
      int.TryParse(input, out nr);
      ```
    - checks for valid input (returns boolean)

      double d;
      ```
      bool ok = double.TryParse(input, out d);
      ```

WHAT ABOUT 'CULTURAL' DIFFERENCES?

- Notation rules per region:
  - Belgian double → 1,234 (comma)
  - English double → 1.123 (point)
  - Think about dates!!
  - Small detail but it can break your application!!

- Solution: CultureInfo

CULTUREINFO

- Provides information about a specific culture (aka Locale)
  - Notation Rules for different types (Numbers, Date, Time, ...)
  - PARSING Rules for different types

- Several .NET methods can take a CultureInfo object as arguments
  - Determines the cultural interpretation of the data

- `int.Parse(string, cultureInfo)`
  - Default behavior = PC Locale Settings
  - Check & pick the correct CultureInfo for your data

CULTUREINFO - CODE EXAMPLE

```
string n = "1,234";
double k = double.Parse(n, new CultureInfo("nl-BE"));
Console.WriteLine(k);
```

[Comma] is used as a decimal separator in Dutch locale

```
string n = "1.234";
double k = double.Parse(n, new CultureInfo("nl-BE"));
Console.WriteLine(k);
```

FormatException

[Point] is not used as a numerical separator in Dutch locale

```
string n = "1,234";
double k = double.Parse(n, new CultureInfo("en"));
Console.WriteLine(k);
```

file:///D:/DA

1234

[Comma] is used as a thousands separator in English locale

```
string n = "1.234";
double k = double.Parse(n, new CultureInfo("en"));
Console.WriteLine(k);
```

file:///D:/DAE/Le

1,234

[Point] is used as a decimal separator in English locale

CultureInfo Codes: http://msdn.microsoft.com/en-us/goglobal/bb896001.aspx

CULTUREINFO – ON THAT FITS ALL?

- Easiest Solution → CultureInfo.InvariantCulture (Culture – Insensitive)
  - For background IO operations,
  - not for UI purposes

- CultureInfo.Invariant
  - Associated with the English language not with any country/region
  - Commonly used for internal IO/Parsing/Conversion operations

```
string n = "1.234";
double k = double.Parse(n, CultureInfo.InvariantCulture);
Console.WriteLine(k);
```