

HarmoniIT



**HarmoniIT**

# OO Course

**Rob Brinkman**  
**WL | Delft Hydraulics**

## Subjects

General OO principles

Classes and Interfaces

Exception handling

Using UML

Using VisualStudio.Net

Accessing FORTRAN

# Sample program

```
public static void Main()
{
    Rectangle r1 = new Rectangle();
    r1.Width = 5;
    r1.Height = 3;
    double area = r1.GetArea();
    Console.WriteLine(area);
}
```

# Class Example

```
public class Rectangle
{
    public double Width;
    public double Height;
    public double GetArea()
    {
        return Width * Height;
    }
}
```

# Properties

```
public class Rectangle
{
    private double _width;

    public int Width
    {
        get {return _width;}
        set
        {
            if (value < 0) {
                _width = 0;
            } else {
                _width = value;
            }
        }
    }
}
```

```
public class Rectangle
{
    private double _width = 0;
    private double _height = 0;

    public Rectangle(int width, int height)
    {
        _width = width;
        _height = height;
    }
}
```

# Static method

```
public class Rectangle
{
    public static Rectangle Biggest (Rectangle r1, Rectangle r2)
    {
        if (r1.GetArea() > r2.GetArea())
        {
            return r1;
        }
        else
        {
            return r2;
        }
    }
}
```

# Sample program

```
public static void Main()  
{  
    Rectangle r1 = new Rectangle (5, 3);  
    Rectangle r2 = new Rectangle (4, 4);  
  
    Rectangle biggest = Rectangle.Biggest (r1, r2);  
  
    Console.WriteLine  
        (biggest.Width, biggest.Height);  
}
```

# Exercise 1:

- Open VisualStudio.Net
- Create a new, blank solution
- Add a project to the solution
  - C#, Console Application
- Let the application write "Hello OpenMI" to the console
- Execute the program

## Exercise 2

- Create a new C# console application
- Add a class Rectangle, which holds properties for the left, right, top and bottom side
- Add a method to compute the area
- Add another class containing the Main method
- Let this class instantiate a rectangle object
- Let this class read from the Console to fill required properties of the rectangle
  - use methods `string Console.ReadLine()` and `int Convert.ToInt32(string)`
- Write useful information to the Console
- Test your application

# Exception handling

```
int i = -1;
bool inputOK = false;
while (!inputOK)
{
    try {
        string input = Console.ReadLine();
        i = Convert.ToInt32(string);
        inputOK = true;
    }
    catch (Exception e) {
        Console.WriteLine (e.Message);
    }
}

Console.WriteLine(String.Format("You entered {1}", i));
```

# Another class

```
public class Circle
{
    private double _radius = 0;

    public double Radius {
        get {return _radius;}
        set {_radius = value;}
    }

    public double GetArea()
    {
        return System.Math.PI * _radius * _radius;
    }
}
```

# Inheritance

```
public class Shape
```

```
{
```

```
    public virtual double GetArea() {return 0;}
```

```
}
```

```
public class Rectangle : Shape
```

```
{
```

```
    ... properties Width and Height
```

```
    public override double GetArea() {return ... ;}
```

```
}
```

```
public class Circle : Shape
```

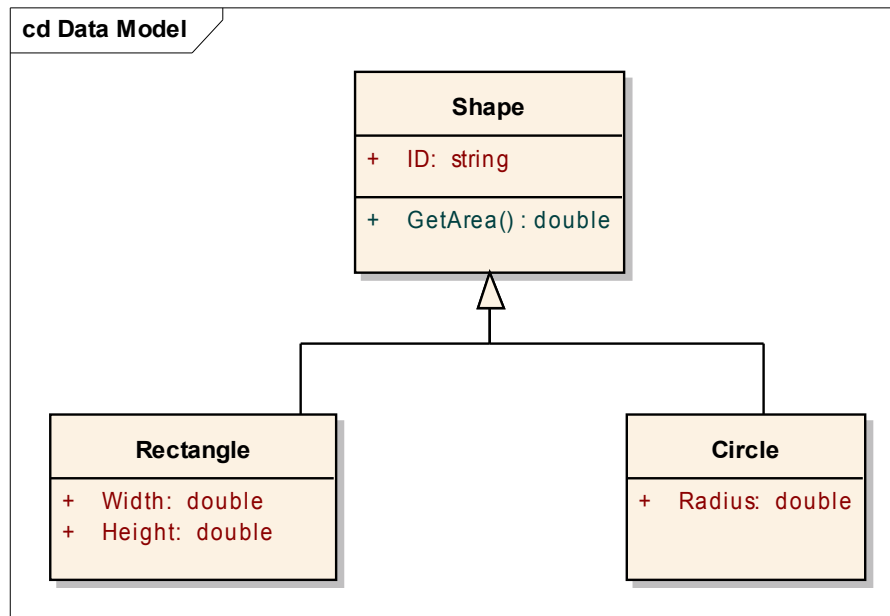
```
{
```

```
    ... property Radius
```

```
    public override double GetArea() {return ...;}
```

```
}
```

# Inheritance - UML



# Using Inheritance (1)

```
public static Main()
{
    Rectangle r1 = new Rectangle(5,4);
    r1.ID = "My rectangle";

    Circle c1 = new Circle(10);
    r1.ID = "My Circle";

    if (c1 is Shape) {Console.WriteLine (c1.ID + " is a shape");}
    if (c1 is Rectangle) {Console.WriteLine (c1.ID + " is a rectangle");}
}
```

## Using Inheritance (2)

```
public static Main()
{
    Shape s0 = new Shape();
    Shape s1 = new Rectangle (7, 4);
    Shape s2 = new Circle (5);

    double area =
        s0.GetArea() + s1.GetArea() + s2.GetArea();

    Console.WriteLine (area);
}
```

## Exercise 3

- Open the project Exercise3
- Change the MainClass so that it will not crash
  - Use a try catch block

# Exercise 4

- Open Exercise 4
- Implement a base class Geometry and subclasses Rectangle and Circle, all with appropriate properties and methods
- Create a main class, which creates some rectangles and circles; treat them as geometries
- Let the main class write the combined area to the console

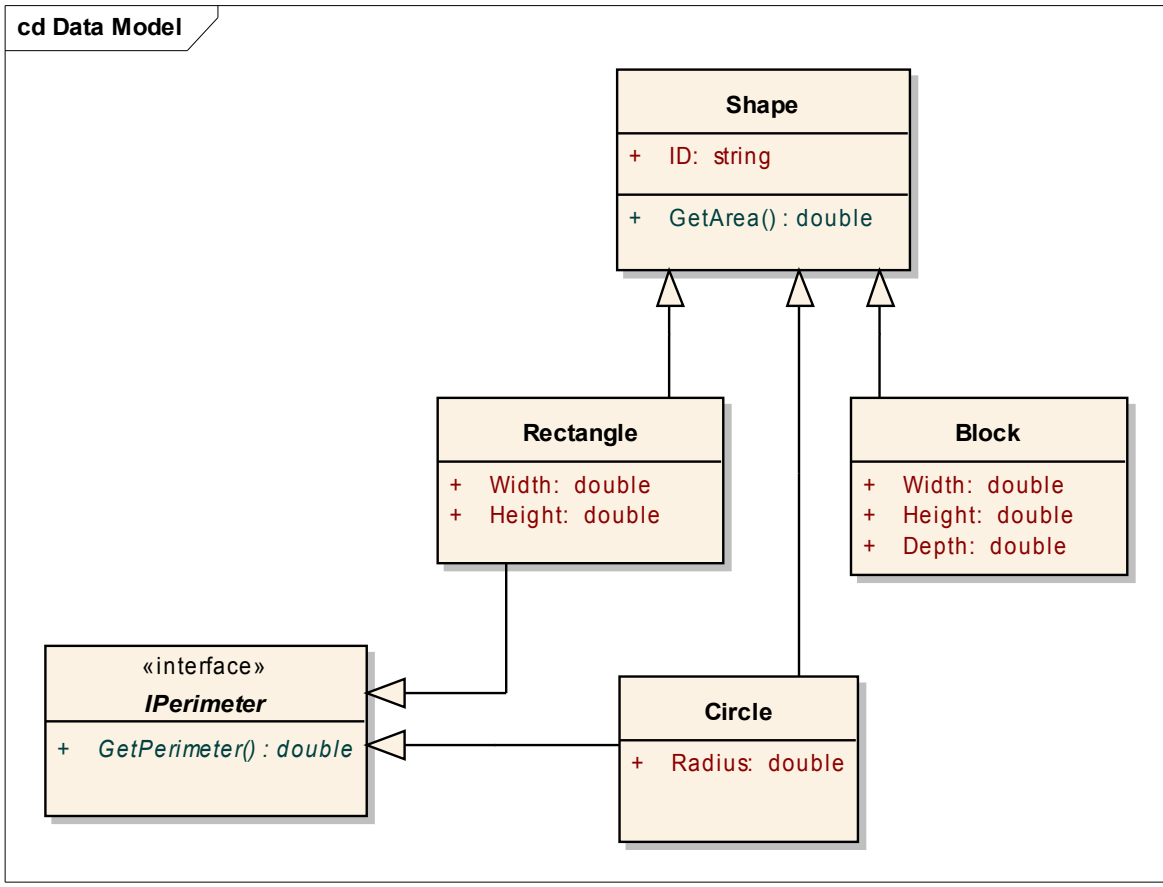
# Interfaces

```
public interface IPerimeter
{
    double GetPerimeter();
}
```

```
public class Rectangle : Geometry, IPerimeter
{
    ... existing properties and methods

    public double GetPerimeter()
    {
        return 2 * (Width + Height);
    }
}
```

# Interfaces - UML



# Sample Program

```
public static class Main() {  
    double perimeter = 0;  
    Geometry r1 = new Rectangle (3, 4);  
    if (r1 is IPerimeter) {  
        perimeter += ((IPerimeter) r1).GetPerimeter();  
    }  
    Geometry c1 = new Circle (5);  
    if (c1 is IPerimeter) {  
        perimeter += ((IPerimeter) c1).GetPerimeter();  
    }  
    Geometry b1 = new Block (5,4,2);  
    if (b1 is IPerimeter) {  
        perimeter += ((IPerimeter) b1).GetPerimeter();  
    }  
  
    Console.WriteLine (perimeter);  
}
```

# Inheritance vs Interfaces

## Inheritance

- Default Implementation
- Single inheritance
- Base class can be instantiated
- Not for existing class

## Interfaces

- No default implementation
- Multiple interfaces
- Interface cannot be instantiated
- Can be applied to existing class

# Object Orientation

- **What is an Object?**
- An object is a software bundle of related variables and methods. Software objects are often used to model real-world objects you find in everyday life.
- **What is a Class?**
- A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind.
- **What is Inheritance?**
- A class inherits state and behavior from its superclass. Inheritance provides a powerful and natural mechanism for organizing and structuring software programs.
- **What is an Interface?**
- An interface is a contract in the form of a collection of method and constant declarations. When a class implements an interface, it promises to implement all of the methods declared in that interface.

# Collections

using System.Collections;

```
ArrayList list = new ArrayList();  
list.Add (new Rectangle(5,4));  
list.Add (new Circle(3));
```

```
double p1 = 0;  
double p2 = 0;
```

```
for (int i = 0; i < list.Count; i++) {  
    p1 += ((IPerimeter) list[i]).GetPerimeter();  
}
```

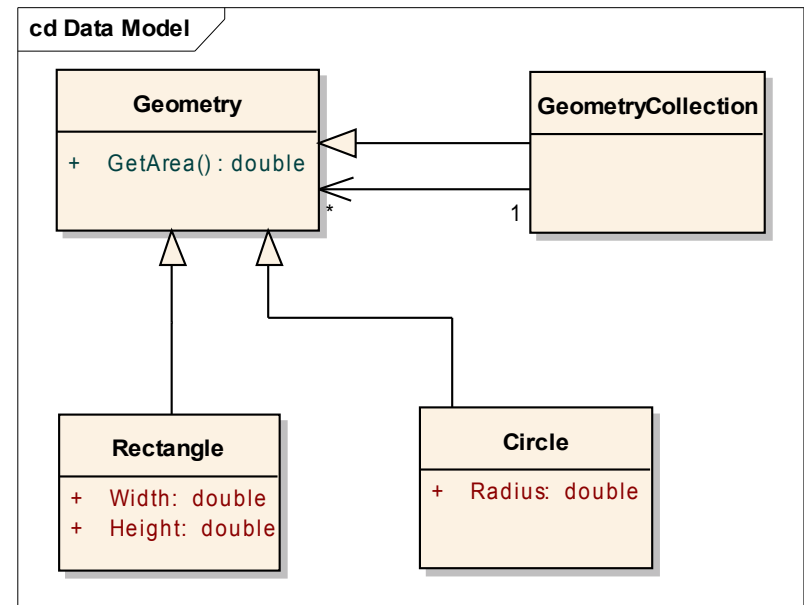
```
foreach (IPerimeter perimeter in list) {  
    p2 += perimeter.GetPerimeter();  
}
```

# Exercise 5

- Open Exercise 5
- Create an interface `IBounds`, which holds read only properties for left, right, top and bottom
- Let `Rectangle` and `Circle` implement this interface
  - Left meaning the most left point of the circle, etc.
- Create a class `SurroundingRectangle`, which inherits from `Rectangle` and holds a method `Add (IBounds bounds)`;
  - Left meaning the most left point of all bound objects added
- Write a main class to test your classes

# Exercise 6

- Open Exercise 6
- Implement classes as given in the UML
- Use an internal ArrayList in the GeometryCollection
- Write a main class to test you classes



# FORTRAN-C# (C# Side)

```
using System.Runtime.InteropServices;
```

```
[DllImport("Math.dll")]
```

```
private static extern double SQUARE (ref double x);
```

```
...
```

```
double x = 3;
```

```
double xx = SQUARE (ref x);
```

```
...
```

# FORTRAN-C# (FORTRAN Side)



```
double precision function SQUARE (x) result (fn_val)
!DEC$ ATTRIBUTES DLLEXPORT :: SQUARE
implicit none
double precision, intent(IN) :: x
fn_val = x * x
return
end function SQUARE
```

# FORTRAN-C# (Arrays)

```
double[] array = new double[] {2, 5, 6};  
int length = array.Length;  
double max = MAX (array, ref length);
```

```
[DllImport("Math.dll")]  
private static extern double MAX(double[] x, ref int n);
```

```
double precision function MAX(x, n) result (xmax)  
!DEC$ ATTRIBUTES DLLEXPORT :: MAX  
implicit none  
integer, intent(IN) :: n  
double precision, dimension (n) :: x
```

# FORTRAN-C# (Strings)

```
public double Process (string processType, double x)
{
    int length = type.Length;
    StringBuilder type = new StringBuilder(length);
    type.Append (type);
    return PROCESS (bound, length, ref x);
}
```

```
[DllImport("Math.dll")]
private static extern double PROCESS
    (StringBuilder type, int typeLength, ref double x);
```

```
double precision function PROCESS (type, x) result (fn_val)
!DEC$ ATTRIBUTES DLLEXPORT :: PROCESS
character(Len=*), intent(in) :: type
double precision :: x
```











# Exercise 7







- Open VisualStudio 6
- Create a new project for a Windows dll with exported symbols
- Modify and export the function in the file PolygonArea.f90
  
- Open Exercise 7
- Create a new class Polygon, which inherits from Geometry
- Let the GetArea() method access the FORTRAN dll for computing the area
- Write a main class to test your code

# Exercise 8

- Open the PolygonFunctions project
- Add and export a function POLYBOUND, which accepts
  - a string which bound should be calculated
  - an array of x coordinates
  - an array of y coordinates
- Open Exercise 8
- Modify the Polygon class so that it calls the new function
- Test your code

NUnit Test Runner

Options          

-  D:\OpenMI.org\OO-Exercises\OO-Exercises\Solution9\bin\Debug\Solution9.dll [0,156551s]
  -  Solution9 [0,1095857s]
    -  Tester [0,0939306s]
      -  Rectangle [0,0156551s]
      -  Circle [0,0156551s]
      -  ID [0s]

using NUnit.Framework;

[TestFixture]

public class Tester

[SetUp]

public void CreateGeometries()

[Test]

public void Test()

{

    Rectangle r1 = new Rectangle(4,5);

    Assert.AreEqual (20, r1.GetArea(), "Area");

}

# Exercise 9

- Open Exercise 9
- Convert the project to a dll in the project properties
- Add a reference to the NUnit.Framework
- Create a new class Tester, and implement methods to test your Geometry classes
- Run NUnit in the VisualStudio Add-in