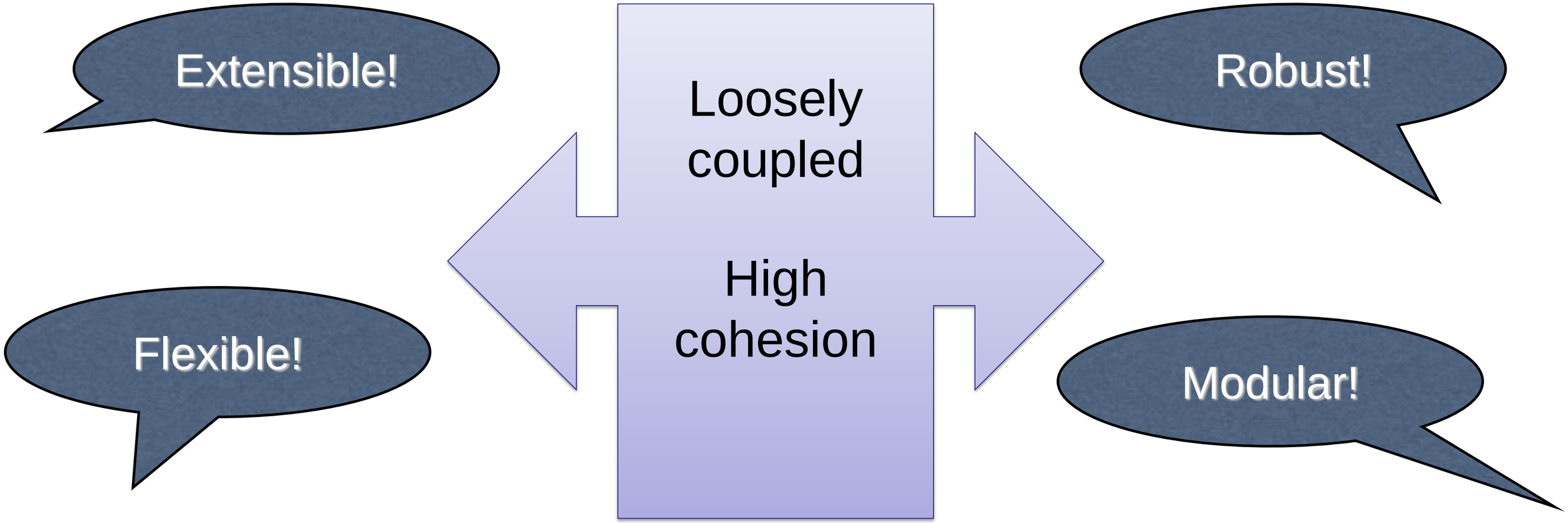
# Achieving Good Object-Oriented Design

by Willem van Straten

SWINBURNE UNIVERSITY OF TECHNOLOGY

# Good design is often described in terms of design goals



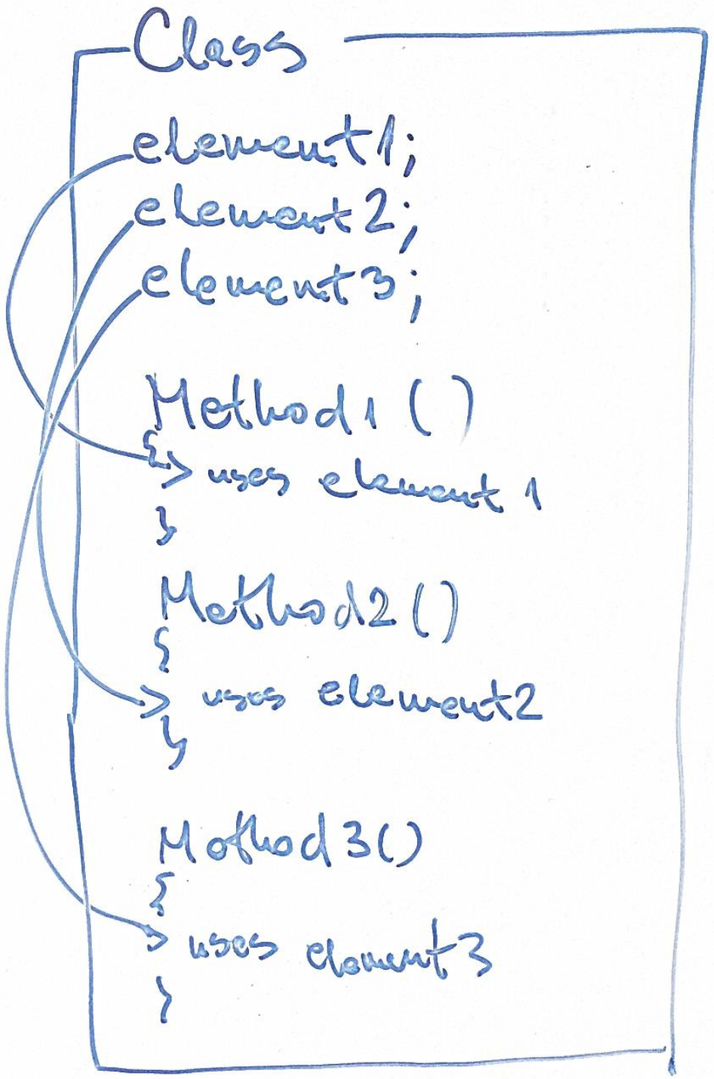
# What is cohesion ??

- Well-defined roles

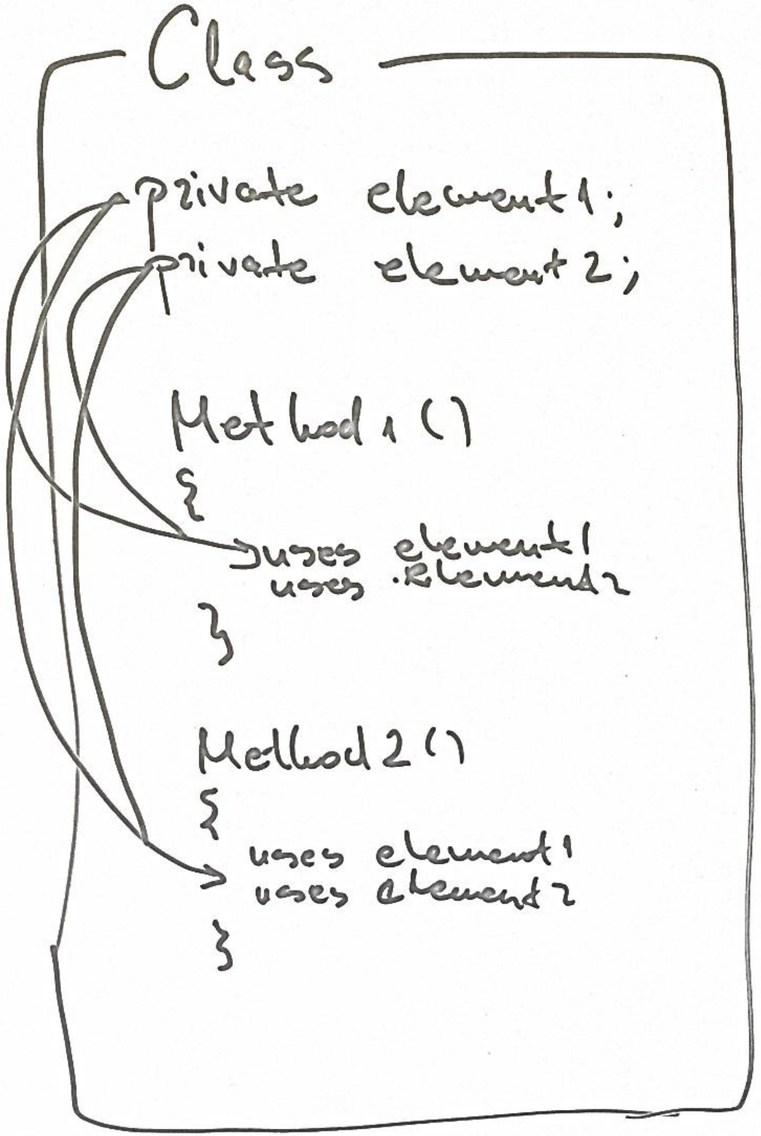
➡ We would say that something is highly cohesive if it has a clear boundary and all of it is contained in one place.

Example: A football is very cohesive

## Low Cohesion




## High Cohesion




All methods contribute to a single well-defined task in the class

# What is coupling?

- extent to which a component of your software depends on other components

**Loose coupling** - components depend on each other to the least extent practically possible…

**Tight coupling** is where components are so tied to one another, that you cannot possibly change the one without changing the other.

# Loose coupling

- Achieved through abstraction and interface

## COHESION AND COUPLING

Cohesion is used to describe a single software component, Coupling is used to describe the relationship between components

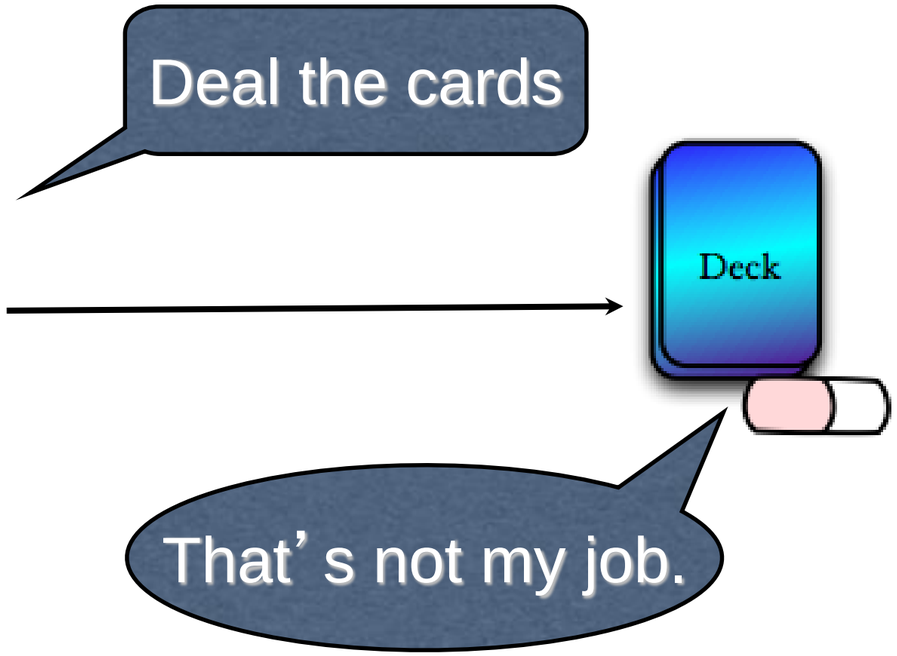
Developers must learn **how** to achieve good object-oriented design

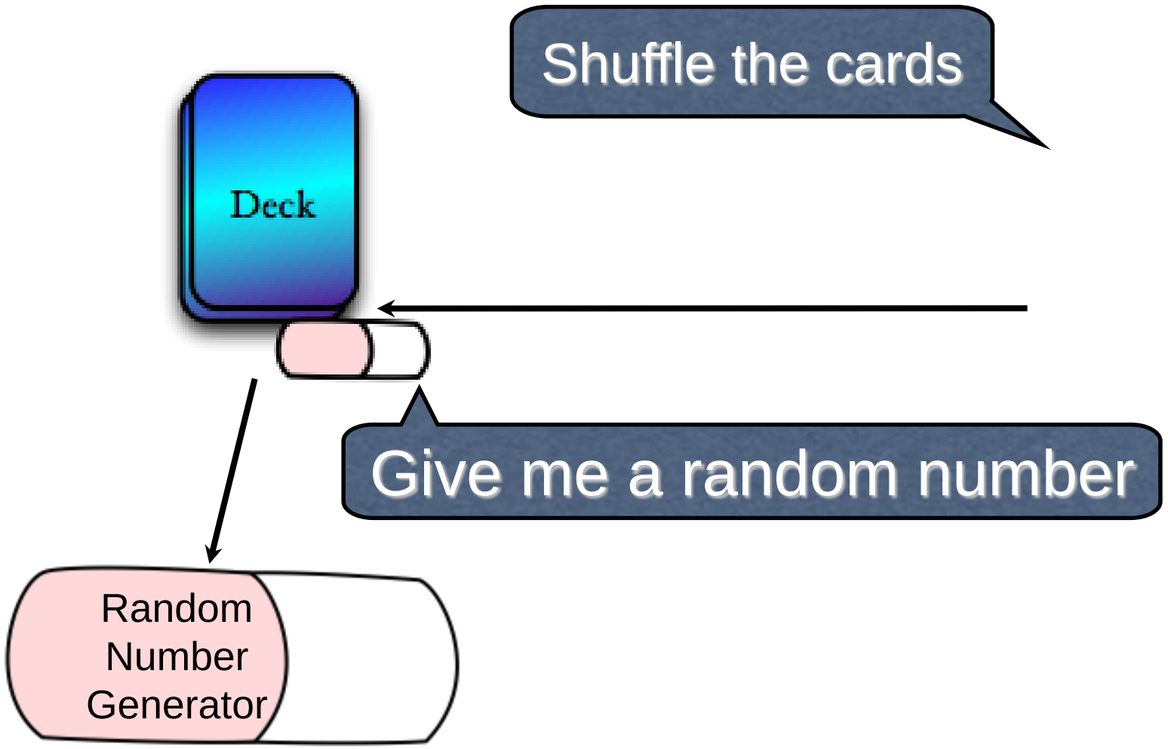
A → B

Developers need principles
- or rules of thumb -
to guide design decisions


Remember these three simple rules

# Rule 1: Classes should be lazy

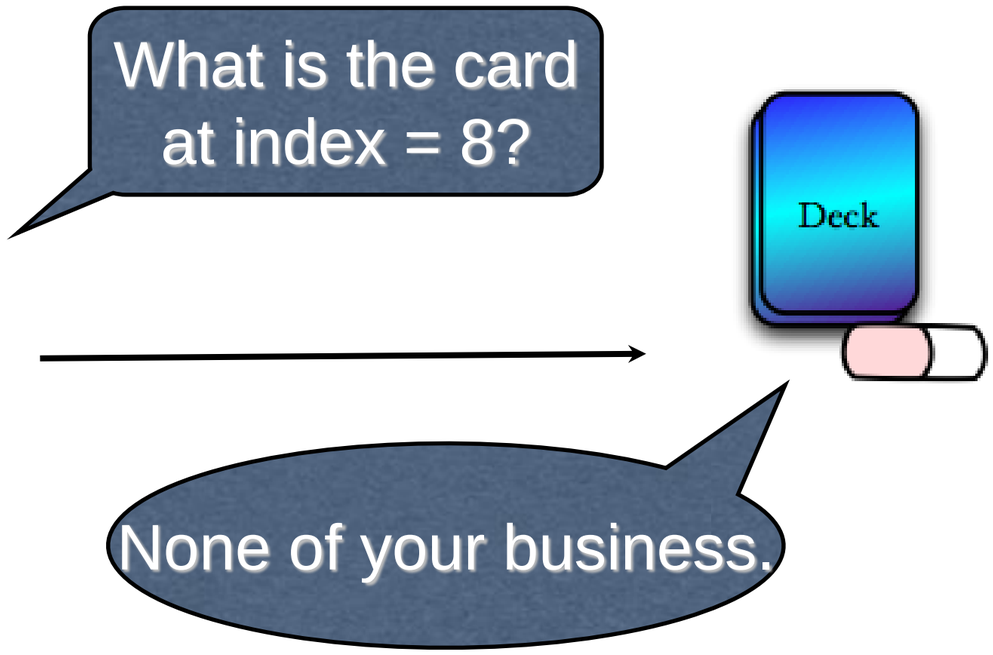


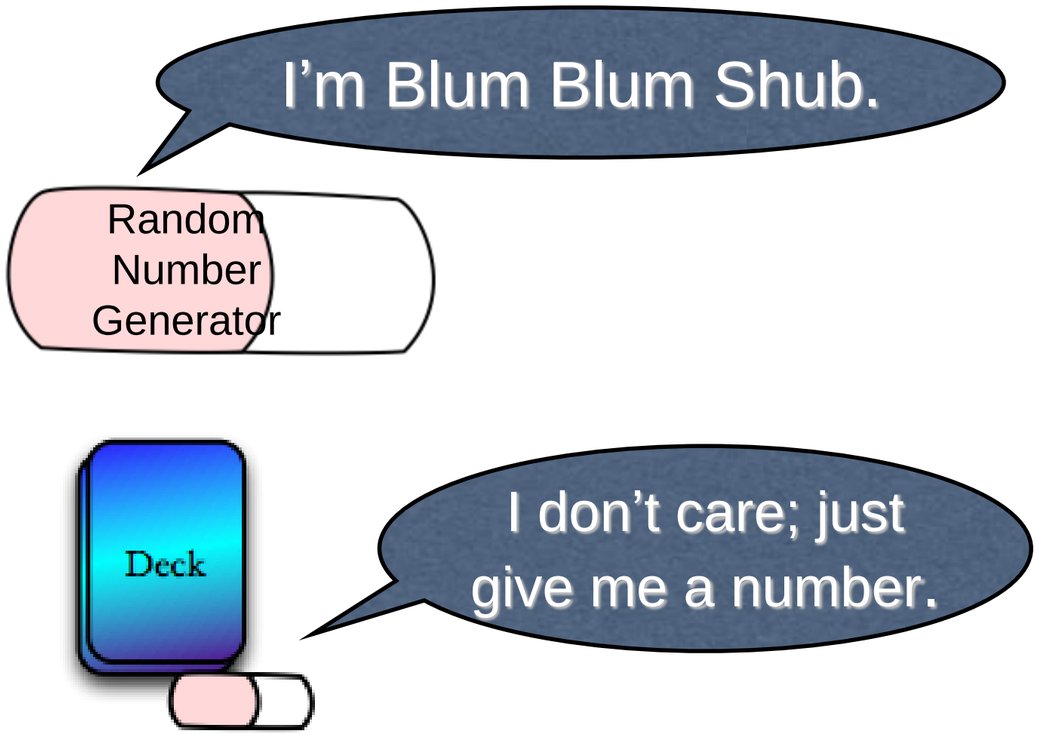




- do as little as possible (single, well-defined role)
- get others to work for you (collaborate)

do little: small, focused classes are easier to understand and more likely to be reused

# Rule 2: Classes should be antisocial

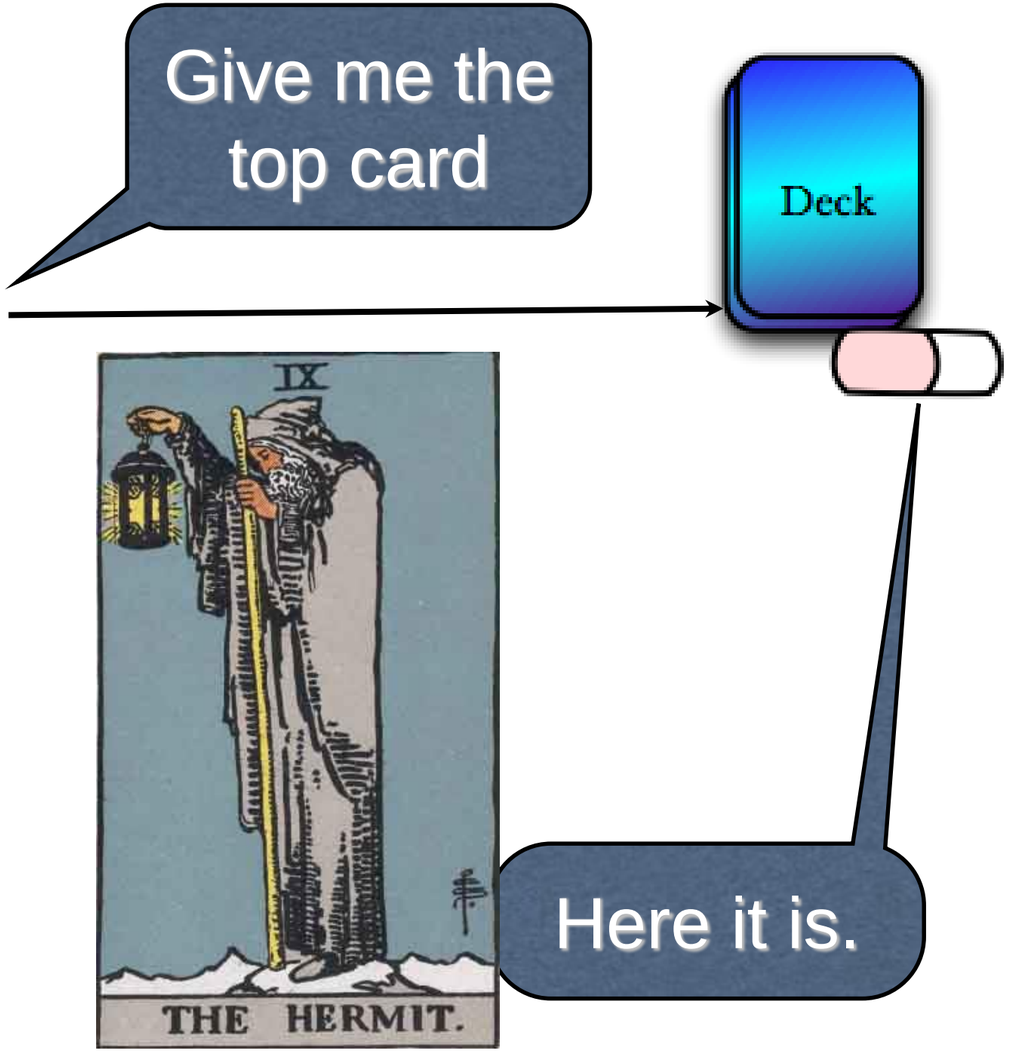



- share as little as possible about self (encapsulation)

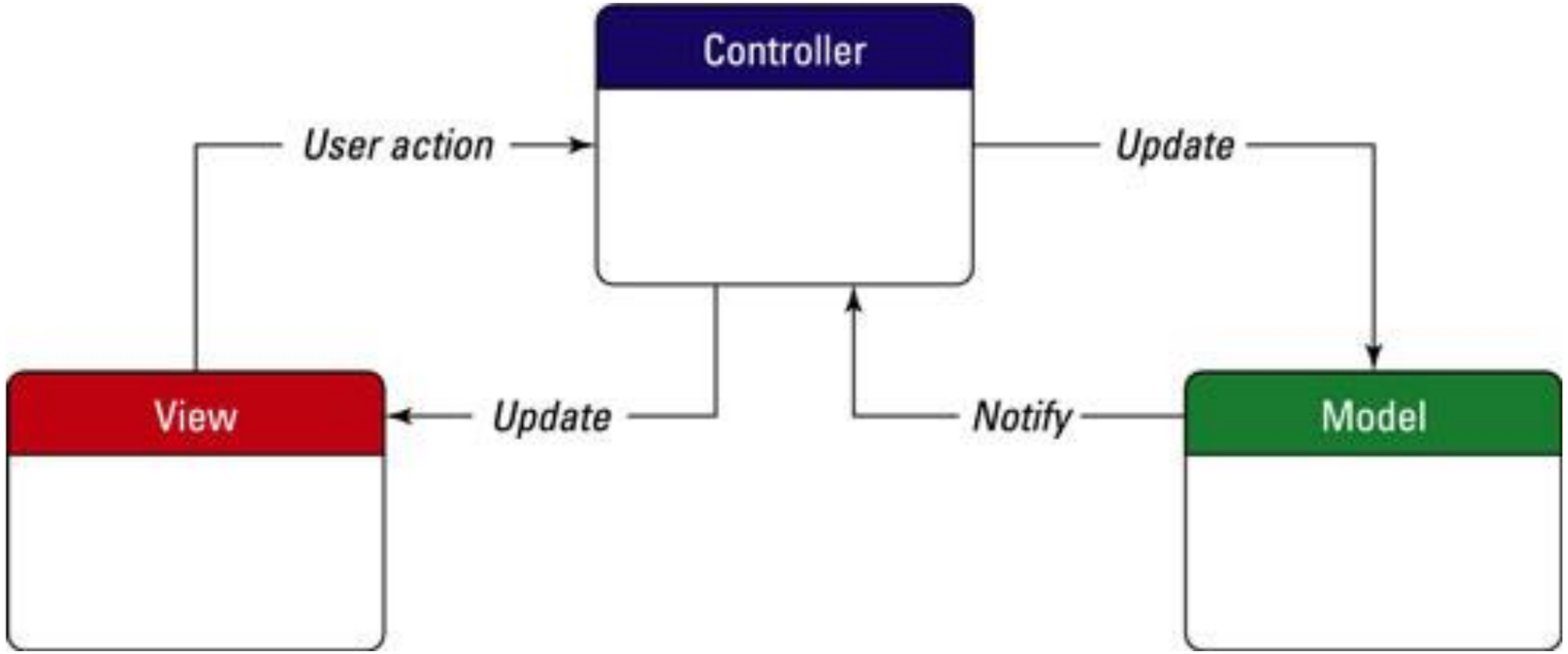
- know as little as possible about others (depend on abstractions)
- minimize inter-dependence so that changes do not cascade and classes can be understood in isolation

# Rule 3: Derived classes should be conformist

# Laziness motivates Separation of Concerns

## leading to model-view-controller (MVC) architecture




**Model:** data and operations that implement functionality
**View:** graphical representation of Model state and available operations
**Controller:** directs user input to Model operations (sometimes via View)

# More on MVC…

| | |
|---|---|
| Handle data and business logic | ➤ **Model** |
| Present data to the user in any supported format and layout | ➤ **View** |
| Receive user requests and call appropriate resources to carry them out | ➤ **Controller** |

Unsociability promotes the use of
Abstractions and Interfaces

Conformity guides the use of
Inheritance and Polymorphism

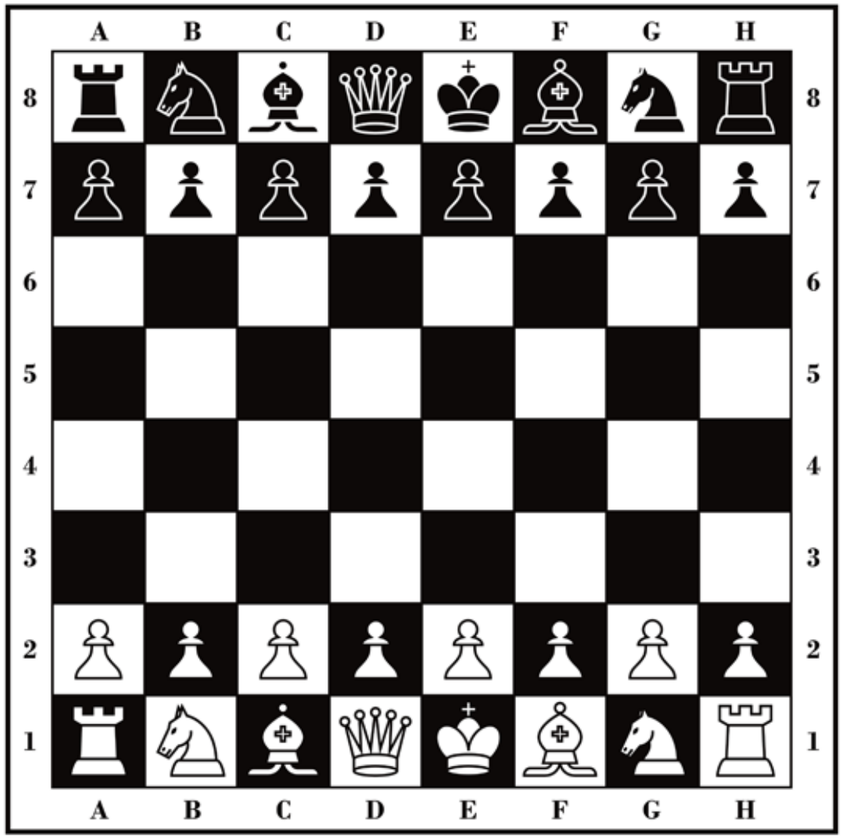
# Responsibility driven design

- Software design involves decomposing a problem into smaller, interacting pieces
- Developers use strategies to conceptualise the problem and explore alternative designs
- Make objects central using Roles, Responsibilities, and Collaborations

# Guidelines

Step 1: Identify **candidate** roles

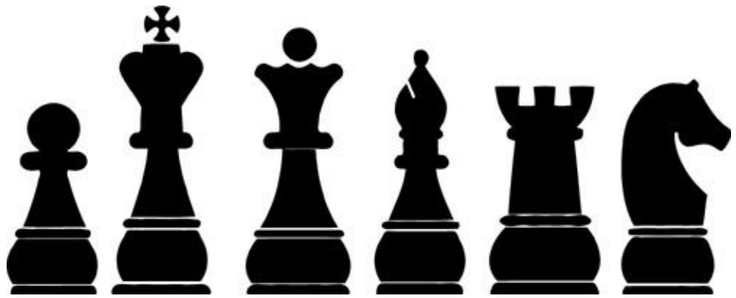
Picture the problem domain and identify **candidate** roles



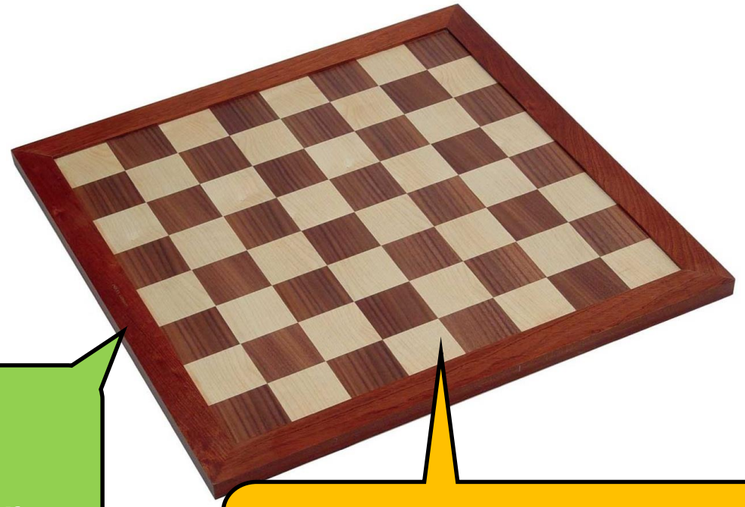
# Guidelines

Step 2: Define **responsibilities** for each candidate role

Each candidate role has responsibilities



I'm a Bishop...
I'm responsible for...

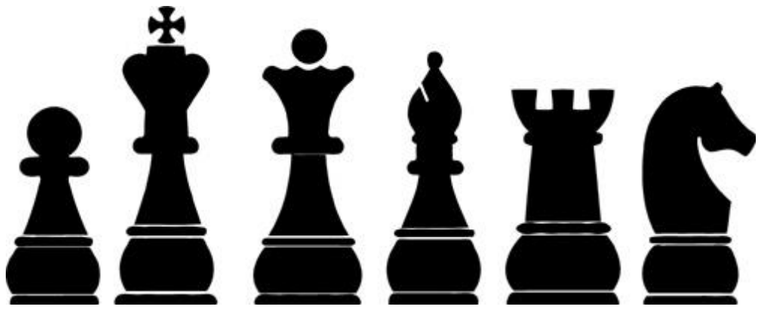


I'm a Board...
I'm responsible for...

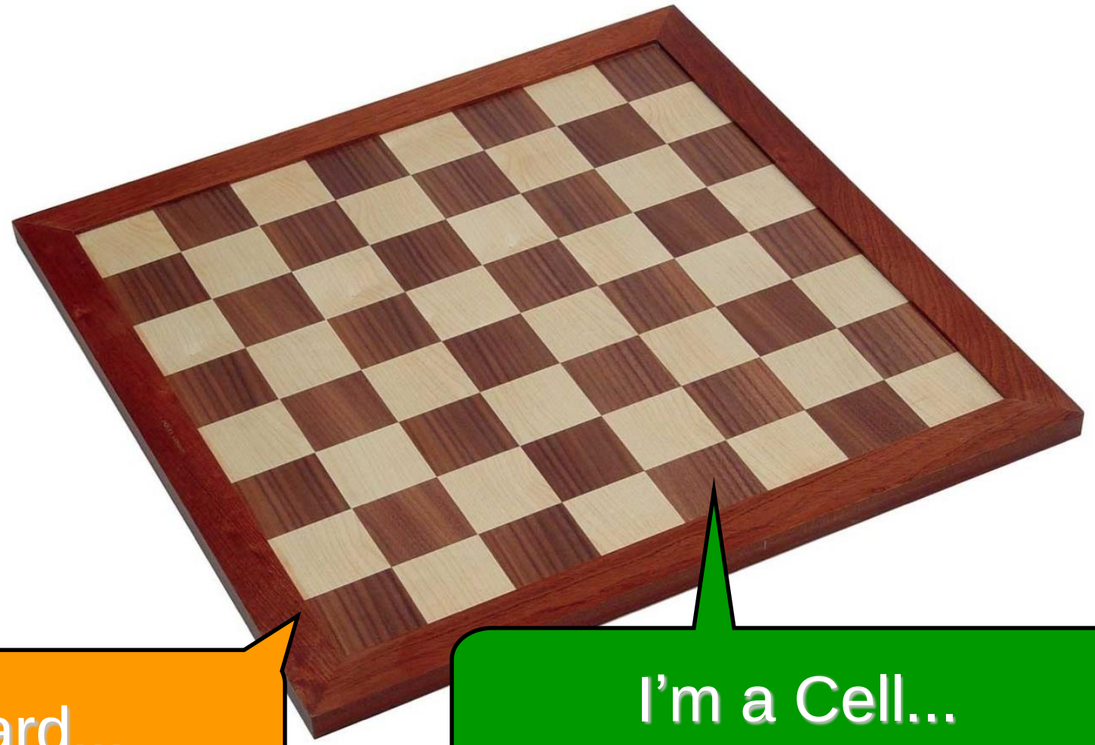
I'm a Cell...
I'm responsible for...

# Guidelines

- Responsibilities include **knowing** things



I’m a King...
I know my colour...

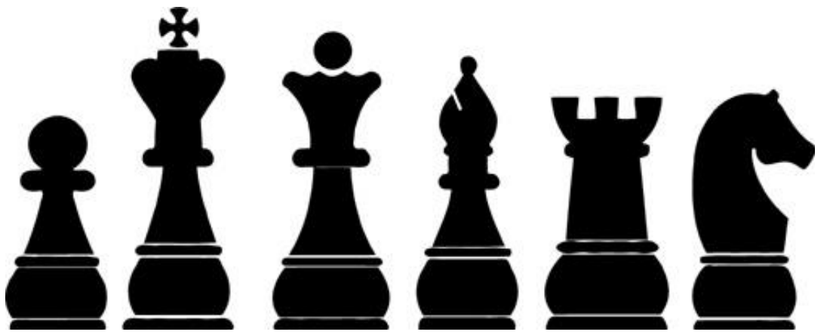


I’m a Board...
I know all of the cells.

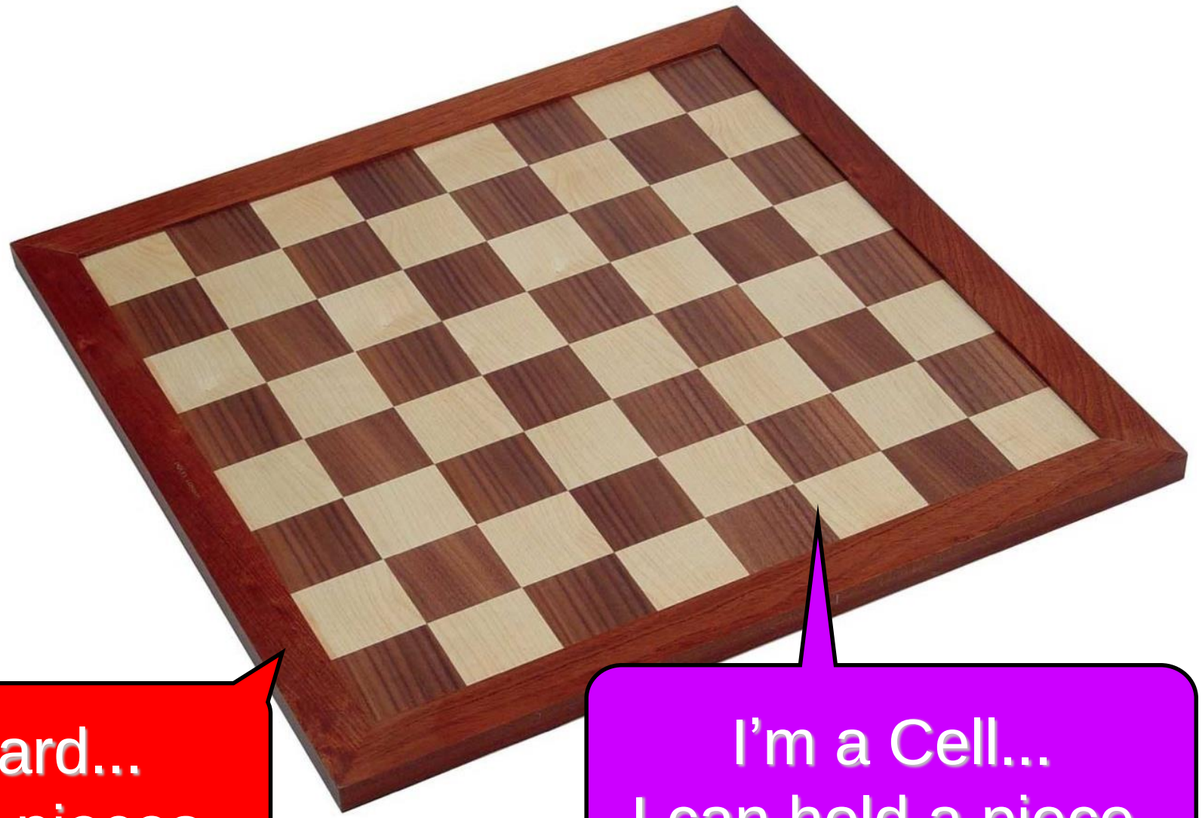
I’m a Cell...
I know my occupant.

# Guidelines

- Responsibilities include **doing** things



I'm a Pawn...
I can be a Queen.

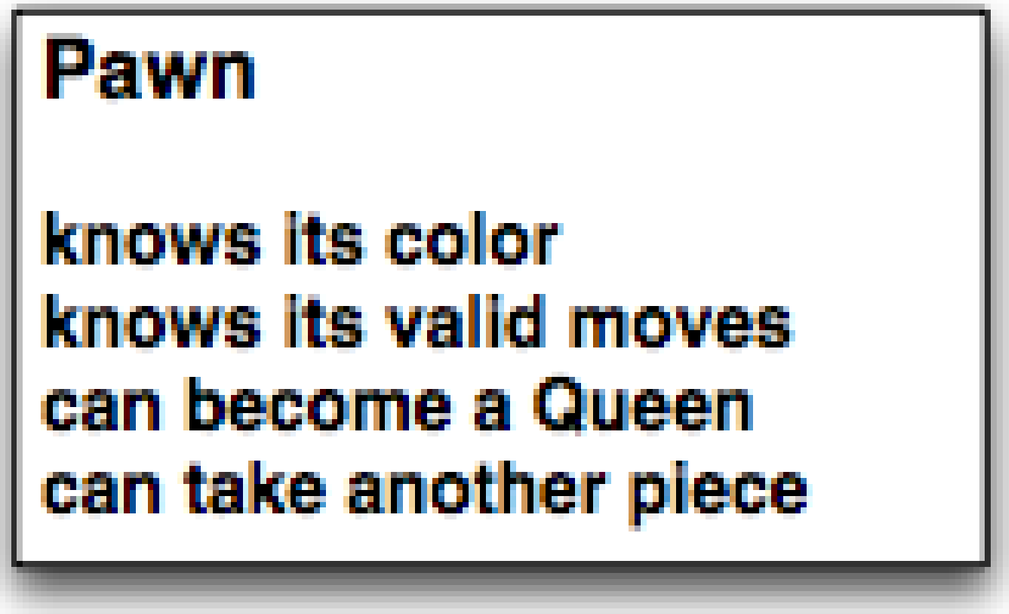


I'm a Board...
I can move pieces.

I'm a Cell...
I can hold a piece.

# Use CRC cards to communicate the responsibilities of candidate roles in your design

- CRC = candidate role, responsibility, collaborations



# Guidelines

Step 3: **Collaborate** with other objects to meet responsibilities

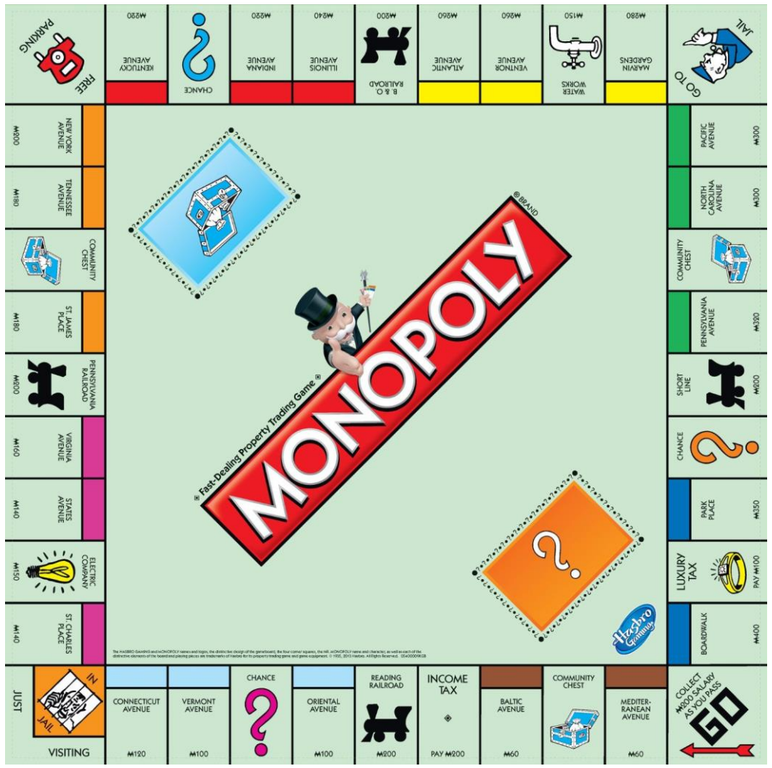
| **Class**: Pawn | |
|---|---|
| **Responsibility:** | **Collaborator:** |
| Knows its colour | |
| Knows its valid moves | Cell |
| Can become Queen | |
| Can take another piece | |

# Implementing Responsibilities

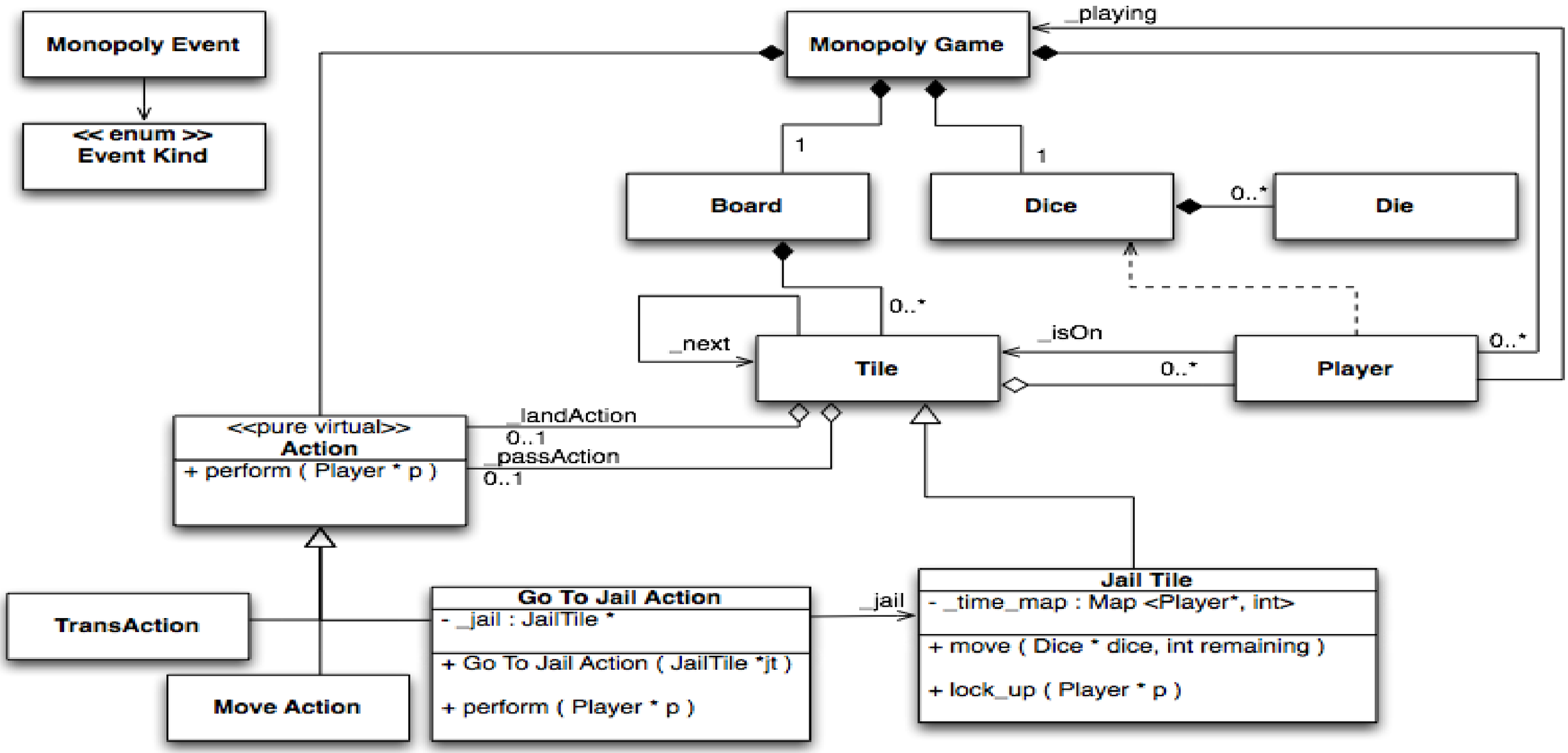
- The responsibility to know something can be implemented as a field in the class. To make it accessible publicly, this can be achieved with a Property or set of Accessor Methods.
- The responsibility to be able to do something can be implemented as a method.

# Time to put on your thinking cap

If you are to develop a monopoly game, what are the roles and responsibilities involved?



# You may have this …



# Make objects central using Roles, Responsibilities, and Collaborations

Procedural thinking is not enough, you must focus on objects

Responsibility driven design focuses on roles and interactions

# Any questions?

# Add-ons (1) to this week lecture

# Operator Overloading

# Operator Overloading

- Use operator (+, -, *,…) to calculate user-defined types such as adding 2 objects.
- functions with special names the keyword **operator** followed by the symbol

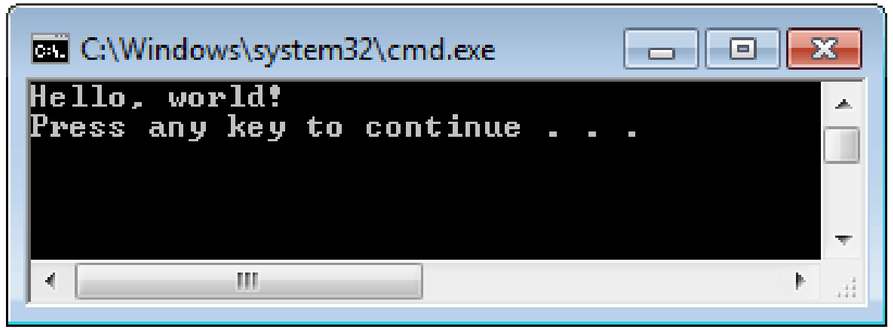
```
// Overload + operator to add two Box objects.
public static Box operator+ (Box b, Box c)
{
        Box box = new Box();
        box.length = b.length + c.length;
        box.height = b.height + c.height;
        return box;
}
```

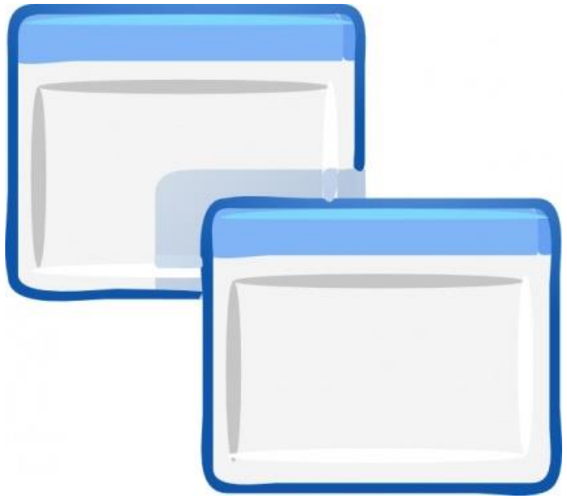
```
// Add two object
Box3 = Box1 + Box2;

Console.WriteLine("Box3 length : {0}",
Box3.Length);

Console.WriteLine("Box3 height : {0}",
Box3.Height);
```

# Add-ons (2) to this week lecture

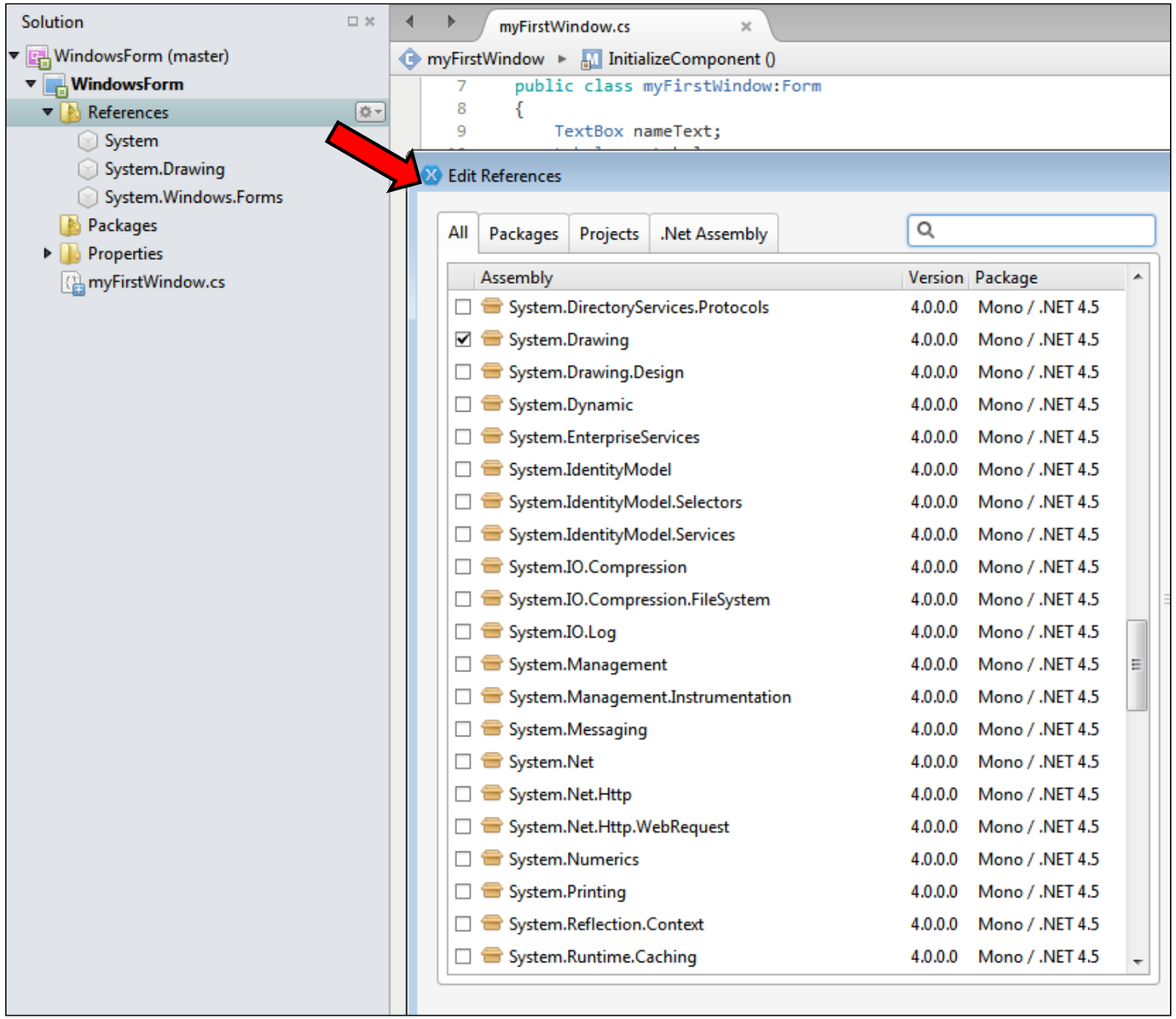
# **Windows Form**

# Other than console-based program, we can have Windows Form in Xamarin too!



# Introducing Windows Form using Xamarin



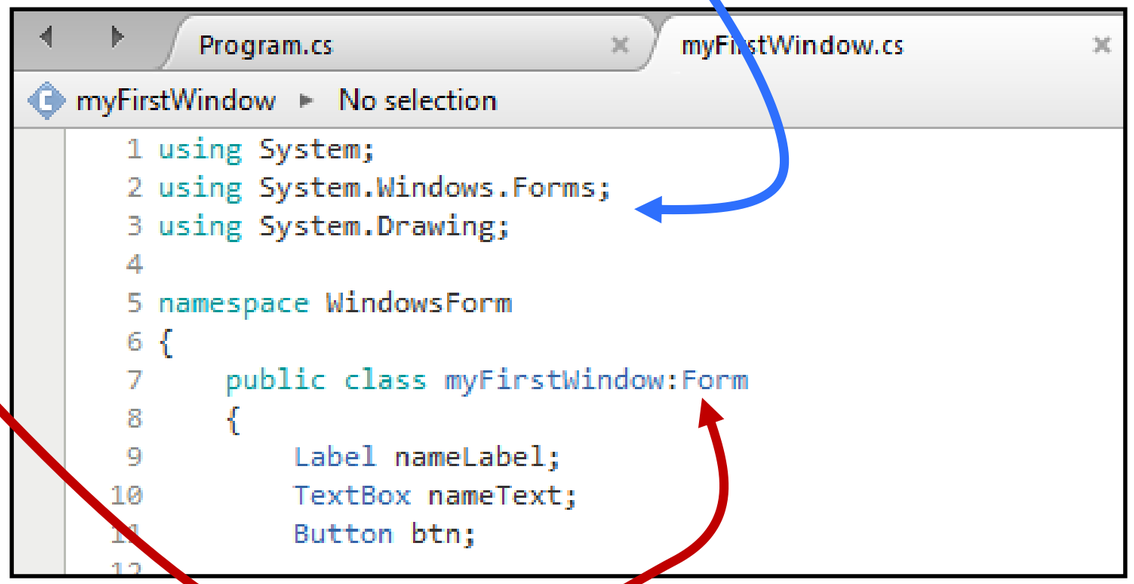

- Add references to System.Drawing and System.Windows.Forms

# Include appropriate libraries

- Include the following references to your codes

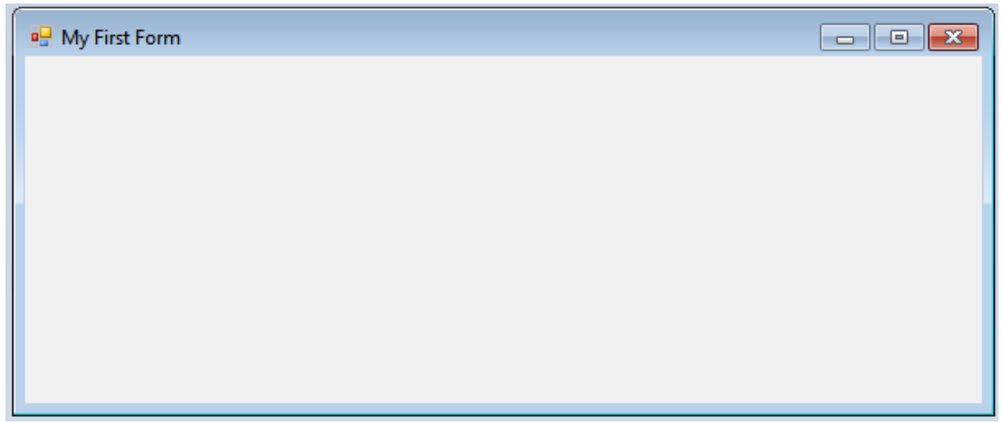
```
using System.Windows.Forms;
using System.Drawing;
```

- Inherit from the “Form” class

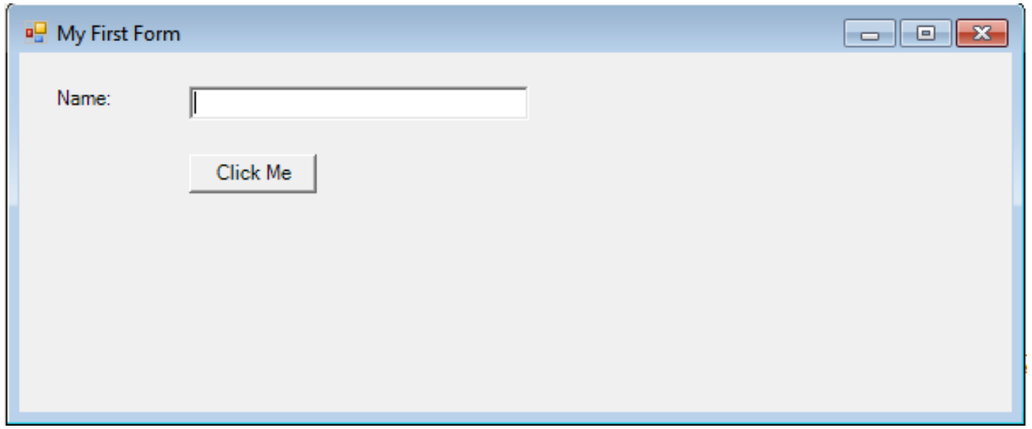


# Creating form and UI components

```
public static void Main(string[] args)
{

    myFirstWindow myApp = new myFirstWindow ();
    myApp.Text="My First Form";
    myApp.Width = 600;
    myApp.Height = 250;

    Application.Run (myApp);
}
```




```
/* Creating labels */
nameLabel = new Label ();
nameLabel.Text = "Name: ";
nameLabel.Width = 50;
nameLabel.Location = new Point (20, 20);


/* Creating textbox */
this.nameText=new TextBox();
this.nameText.Location = new Point (100, 20);
this.nameText.Width = 200;

/* Creating button */
Button btn = new Button ();
btn.Text = "Click Me";
btn.Location = new Point (30, 50);


// Add control to the form
this.Controls.Add (nameLabel);
this.Controls.Add (nameText);
this.Controls.Add (btn);
```



# Adding event handler

- Allows you to code some actions when certain components are triggered
- Example: When a button is clicked, a message box pops out.

```
// attaching event listener
btn.Click += new System.EventHandler (this.btnClick);
nameText.TextChanged += new System.EventHandler (this.txtChangedEvent);
```

Specify the events to handle

Custom method

# Code the events

```
// attaching event listener
btn.Click += new System.EventHandler (this.btnClick);
nameText.TextChanged += new System.EventHandler (this.txtChangedEvent);
```

```
protected void btnClick(object sender, System.EventArgs e)
{
    MessageBox.Show (nameText.Text);

}

protected void txtChangedEvent(object sender, System.EventArgs e)
{
    nameText.BackColor = Color.LightYellow;
}
```

# The complete codes for **myFirstWindow** class

myFirstWindow ► No selection

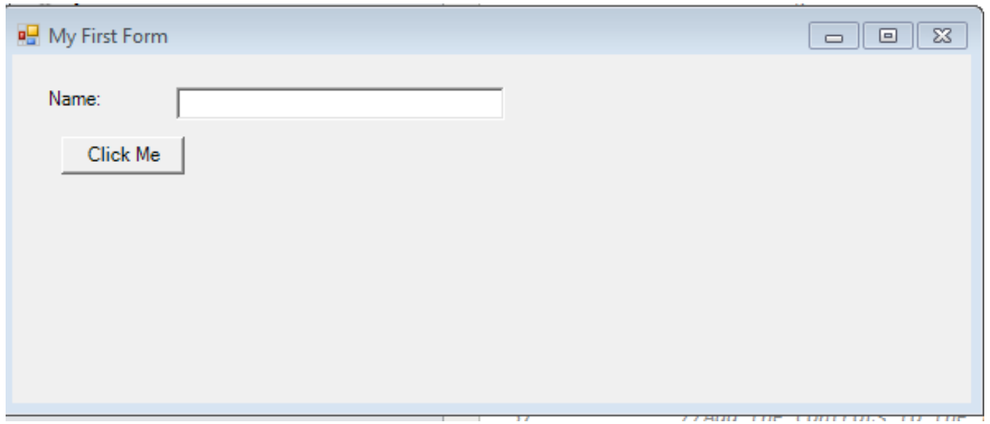
```
1 using System;
2 using System.Windows.Forms;
3 using System.Drawing;
4
5 namespace WindowsForm
6 {
7     public class myFirstWindow:Form
8     {
9         Label nameLabel;
10        TextBox nameText;
11        Button btn;
12
13        public myFirstWindow ()
14        {
15            // Creating Label and set the label's properties
16            nameLabel = new Label ();
17            nameLabel.Text = "Name:";
18            nameLabel.Width = 50;
19            nameLabel.Location = new Point (20, 20);
20
21            // Creating Textbox and set the textbox's properties
22            nameText = new TextBox ();
23            nameText.Location = new Point (100, 20);
24            nameText.Width = 200;
25
26            // Creating Button and set the button's properties
27            btn = new Button ();
28            btn.Text = "Click Me";
29            btn.Location = new Point (30, 50);
30
31            // Add the event handler for button created
32            btn.Click += new System.EventHandler (this.btnClick);
33
34            // Add the event handler for textbox created
35            nameText.TextChanged += new System.EventHandler (this.txtChangedEvent);
36
37            //Add the controls to the form
38            Controls.Add (nameLabel);
39            Controls.Add (nameText);
40            Controls.Add (btn);
41        }
42
43        // Custom Method for handling the button when it is clicked
44        void btnClick (object sender, EventArgs e)
45        {
46            MessageBox.Show (nameText.Text);
47        }
48
49        // Custom Method for handling the textbox when the text in the textbox changed
50        void txtChangedEvent (object sender, EventArgs e)
51        {
52            nameText.BackColor = Color.LightYellow;
53        }
54    }
55 }
```

# The **main program** to run the windows form

MainClass ▸ No selection

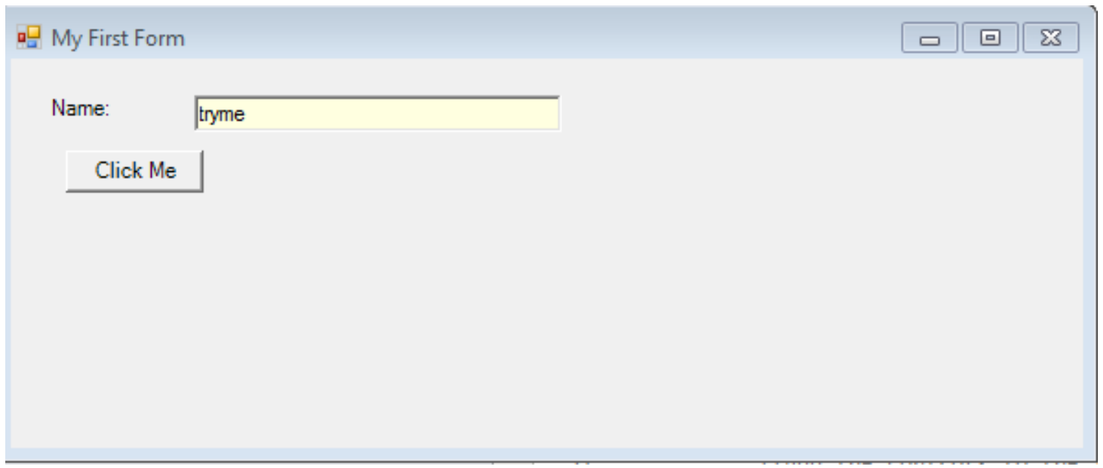
```
1  using System;
2  using System.Windows.Forms;
3  using System.Drawing;
4
5  namespace WindowsForm
6  {
7      class MainClass
8      {
9          public static void Main (string[] args)
10         {
11             myFirstWindow myApp = new myFirstWindow ();
12             myApp.Text = "My First Form";
13             myApp.Width = 600;
14             myApp.Height = 250;
15             Application.Run (myApp);
16         }
17             |
18     }
19 }
20
```

# Program Output

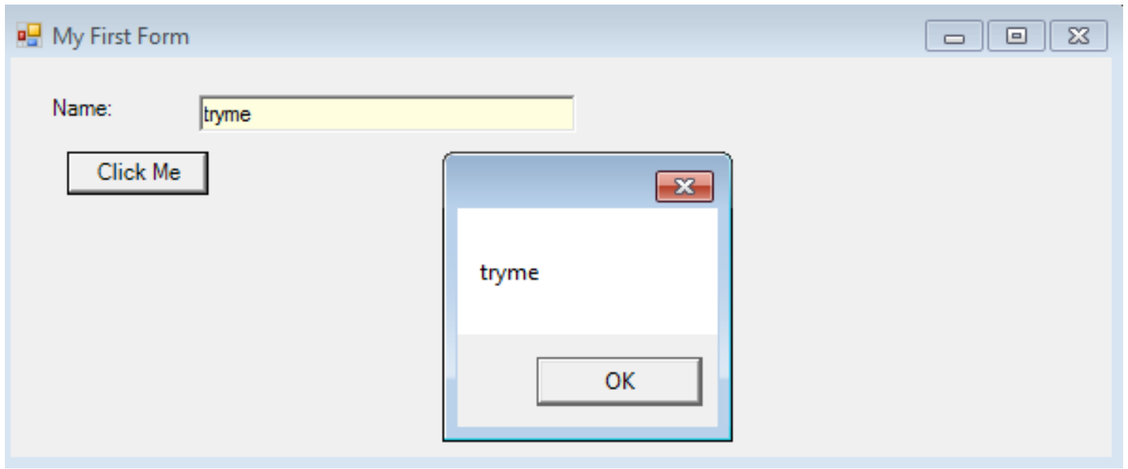



**When the program loads**

**When the text changed**




**When clicking the button**



# Any Questions?

# This Week's Tasks


Supplementary Task: Case Study: Iteration 1 and 2

Swin-Adventure Case Study Implementation

# Mid Term Review

# General Overview



- 100% of your **grade** comes from the Portfolio
- To get a Pass for this unit:
  - ✓ You must submit & complete **Pass Tasks 3, 7, 8, 10 & 12** by this Friday (Last Day)
  - ✓ You must pass the **Semester Test** (Week 6) or **Resit** (Week 8)
  - ✓ You must complete **Pass Task 13** (Week 10)
  - ✓ You must complete and submit a Passable **Learning Summary Report** before end of **Week 12.**

# Present your portfolio for assessment, where it is graded... not marked!

| | |
|---|---|
| **Pass**<br>-- Practices -- | • Complete 5 Assessed Pass Tasks + Pass Test + Pass Task 13 + Pass Task 14 (LSR*) |
| **Credit**<br>-- Understanding -- | • Fulfil all Pass requirements +<br>• Complete 2 Credit Tasks |
| **Distinction**<br>-- Application -- | • Fulfil all Credit requirements +<br>• Complete all (3) Distinction Tasks and Custom Program |
| **High Distinction**<br>-- Research -- | • Fulfil all Distinction requirement +<br>• Complete 1 HD Tasks aka research paper / report |

** LSR – Passable Learning Summary Report*

# Semester Test


- You must pass the Semester Test in test conditions
  - Test will assess core knowledge; i.e. things you will have already done in weekly work
  - You will get another chance to pass the test (ie. 1 resit) ... do your best to pass it the first time (get it out of the way).