Programming 2

Program term 1.2

01 (wk-46)	enumerations / structures / classes
02 (wk-47)	2-dim arrays / flow control
03 (wk-48)	lists / dictionaries
04 (wk-49)	file I/O / error handling
05 (wk-50)	Class Libraries / Layered Architecture
06 (wk-51)	program structure
07 (wk-52)	Christmas holiday
08 (wk-53)	Christmas holiday

----

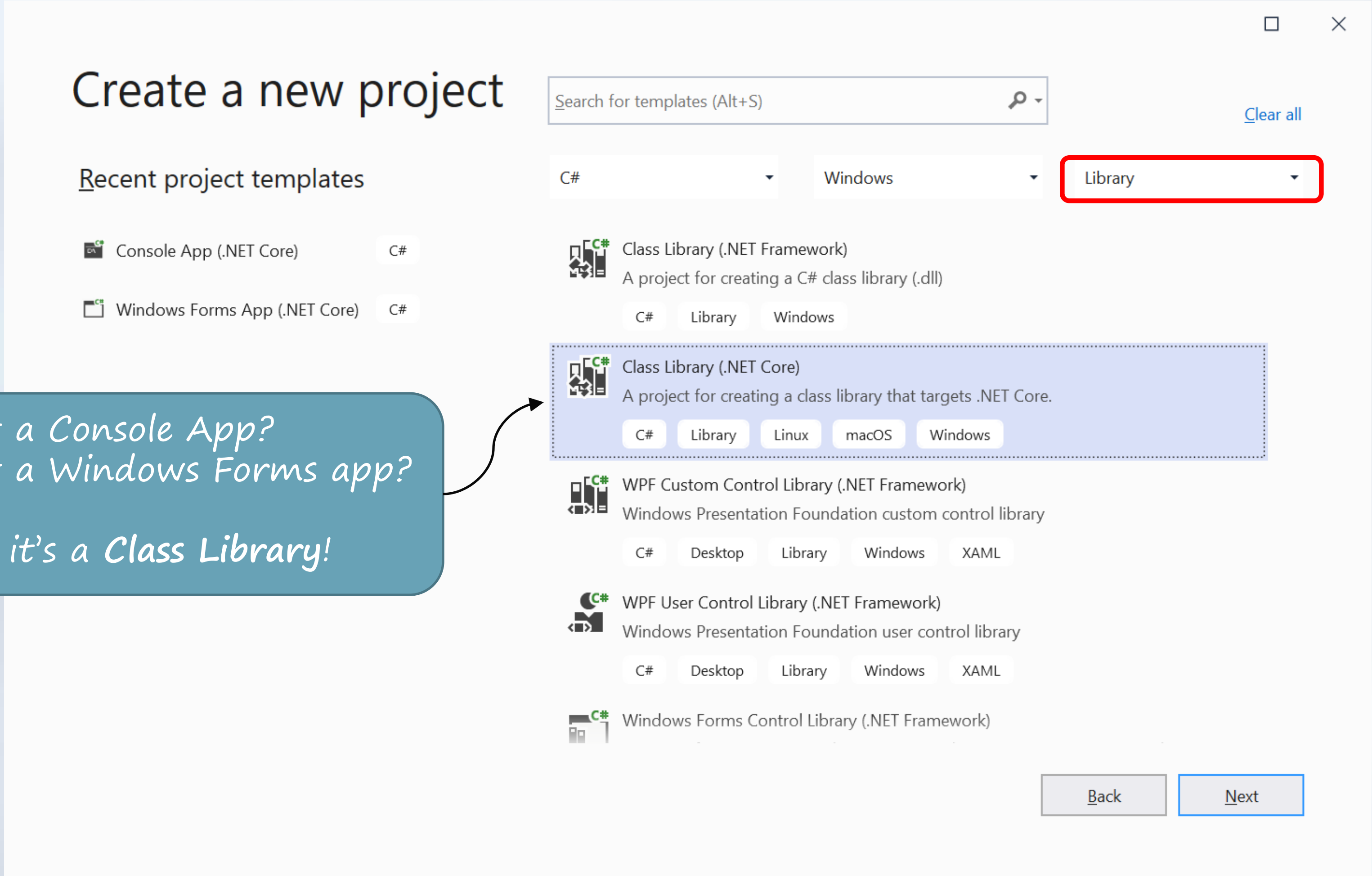
09 (wk-01)	practice exam
10 (wk-02)	*exams*
11 (wk-03)	*retake exams*
12 (wk-04)	*retake exams*

Class Libraries

Common methods

- In the first week we created a few common Read-methods (ReadInt, ReadString)
- These methods can be used in multiple projects
- Until now we had to copy them…
- If one of the methods need to be changed (bugfix, or to make it more efficient, …) we need to do this in several projects
- There is a better way → make a **Class Library** with common methods

Creating a Class Library (DLL)



Create a new project

Search for templates (Alt+S)

Clear all

Recent project templates

- Console App (.NET Core) — C#
- Windows Forms App (.NET Core) — C#

Filters: C# | Windows | Library

Class Library (.NET Framework)
A project for creating a C# class library (.dll)
C# · Library · Windows

Class Library (.NET Core)
A project for creating a class library that targets .NET Core.
C# · Library · Linux · macOS · Windows

WPF Custom Control Library (.NET Framework)
Windows Presentation Foundation custom control library
C# · Desktop · Library · Windows · XAML

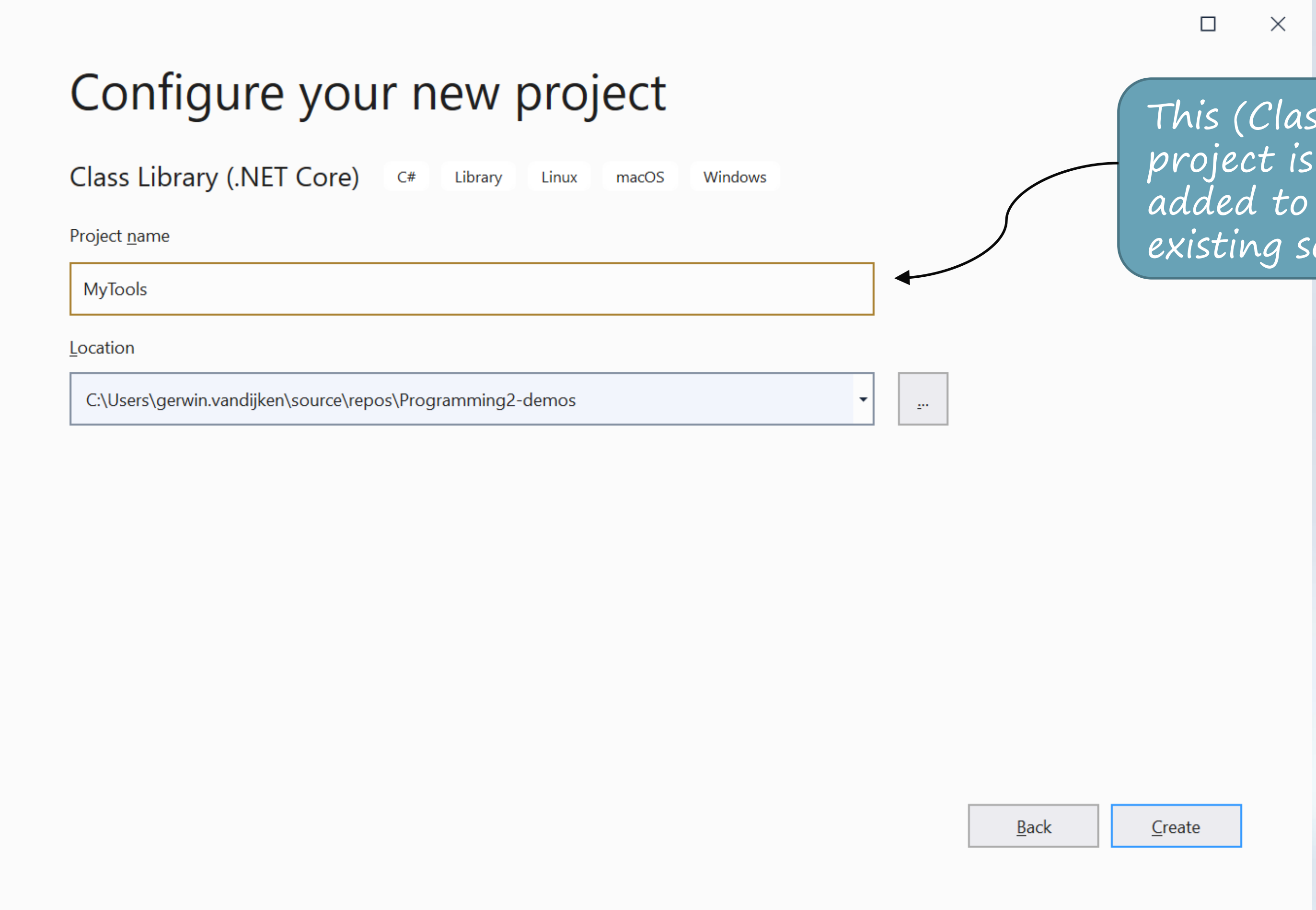
WPF User Control Library (.NET Framework)
Windows Presentation Foundation user control library
C# · Desktop · Library · Windows · XAML

Windows Forms Control Library (.NET Framework)

Is it a Console App?
Is it a Windows Forms app?

No, it's a ***Class Library!***

Creating a Class Library



Configure your new project

Class Library (.NET Core)

Project name: MyTools

Location: C:\Users\gerwin.vandijken\source\repos\Programming2-demos

This (Class Library) project is being added to an existing solution

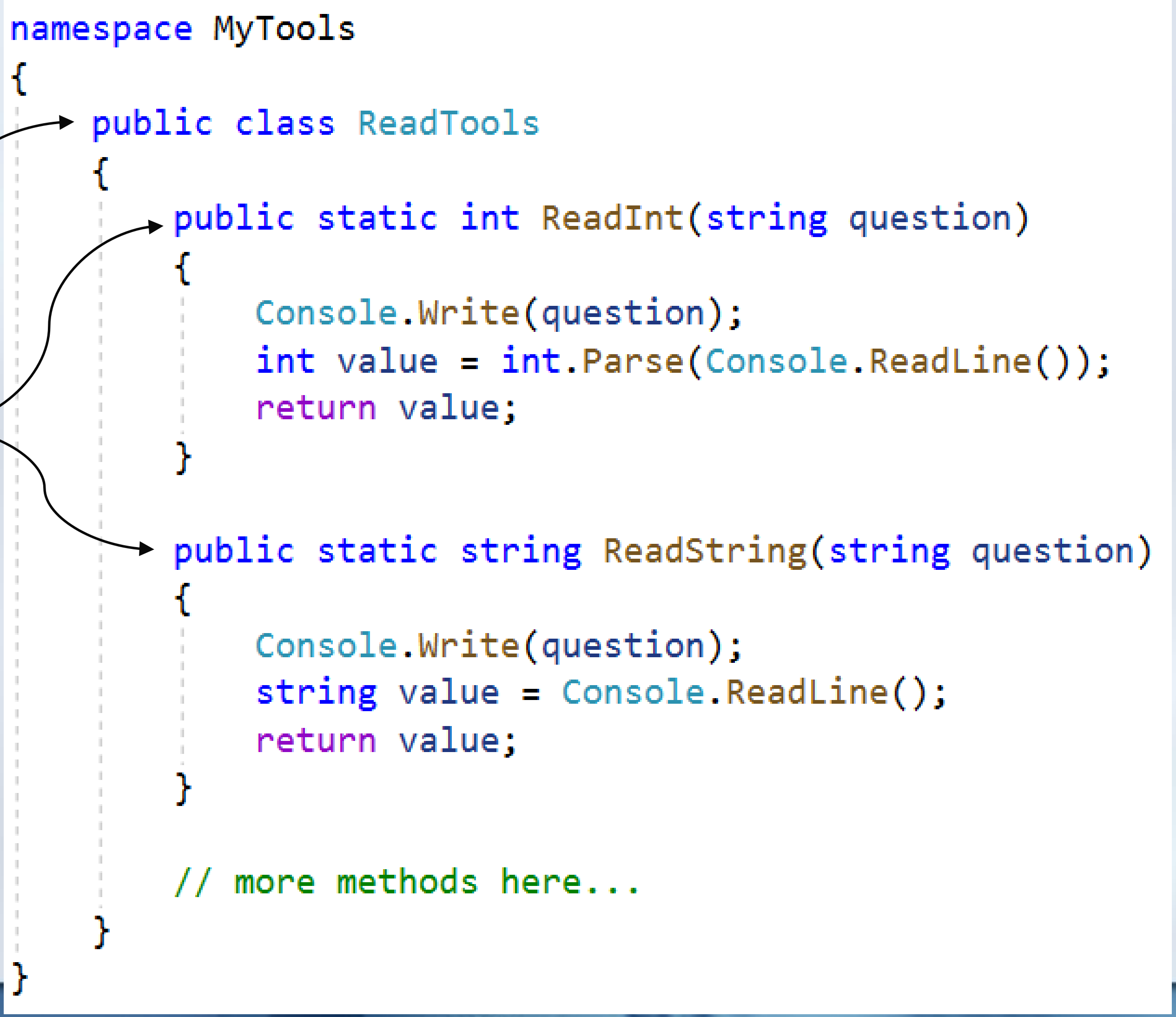
Creating a Class Library

It's just like Math.Abs(...):
* namespace: system
* class: Math
* (static) method: Abs

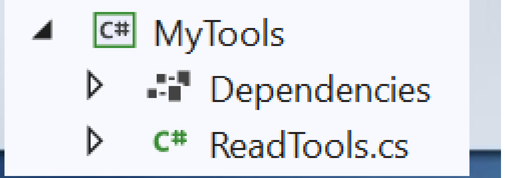
A Class Library contains ***public classes*** ...

... with ***public methods***

You will see this in the solution Explorer (no Program.cs)

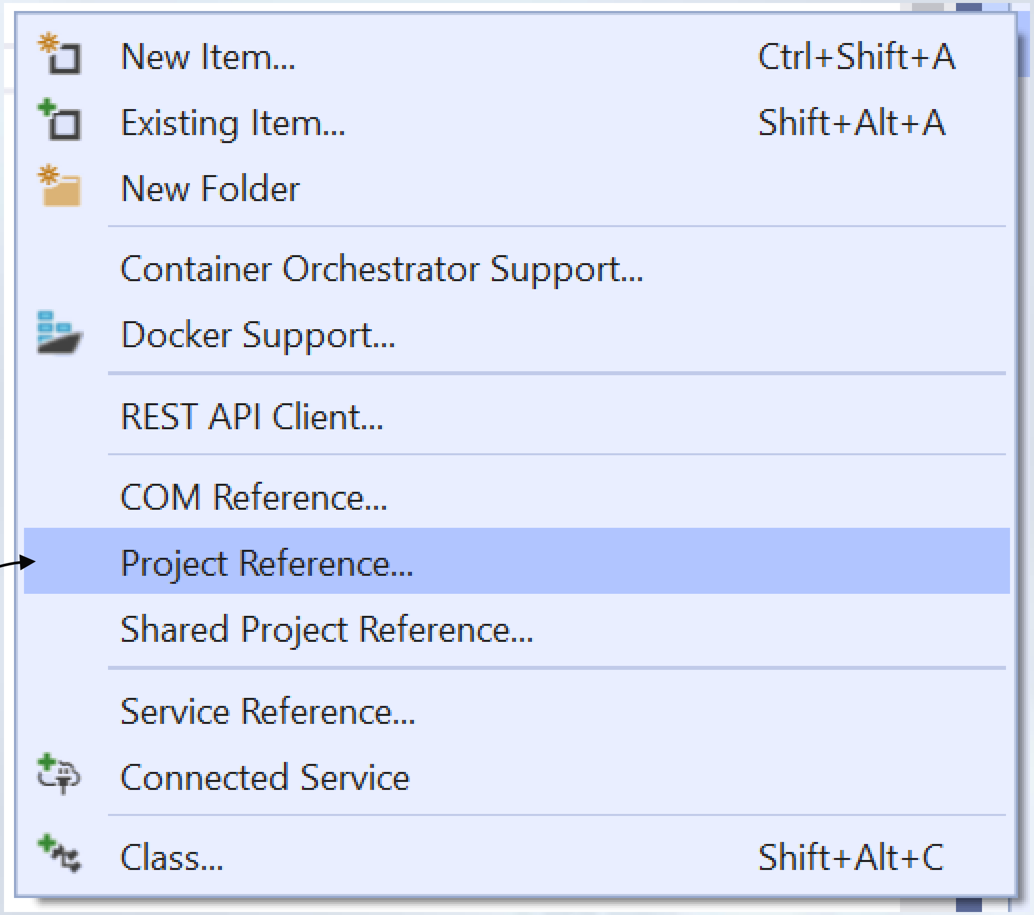


```
namespace MyTools
{
    public class ReadTools
    {
        public static int ReadInt(string question)
        {
            Console.Write(question);
            int value = int.Parse(Console.ReadLine());
            return value;
        }

        public static string ReadString(string question)
        {
            Console.Write(question);
            string value = Console.ReadLine();
            return value;
        }

        // more methods here...
    }
}
```



◢ C# MyTools
    ▷ Dependencies
    ▷ C# ReadTools.cs

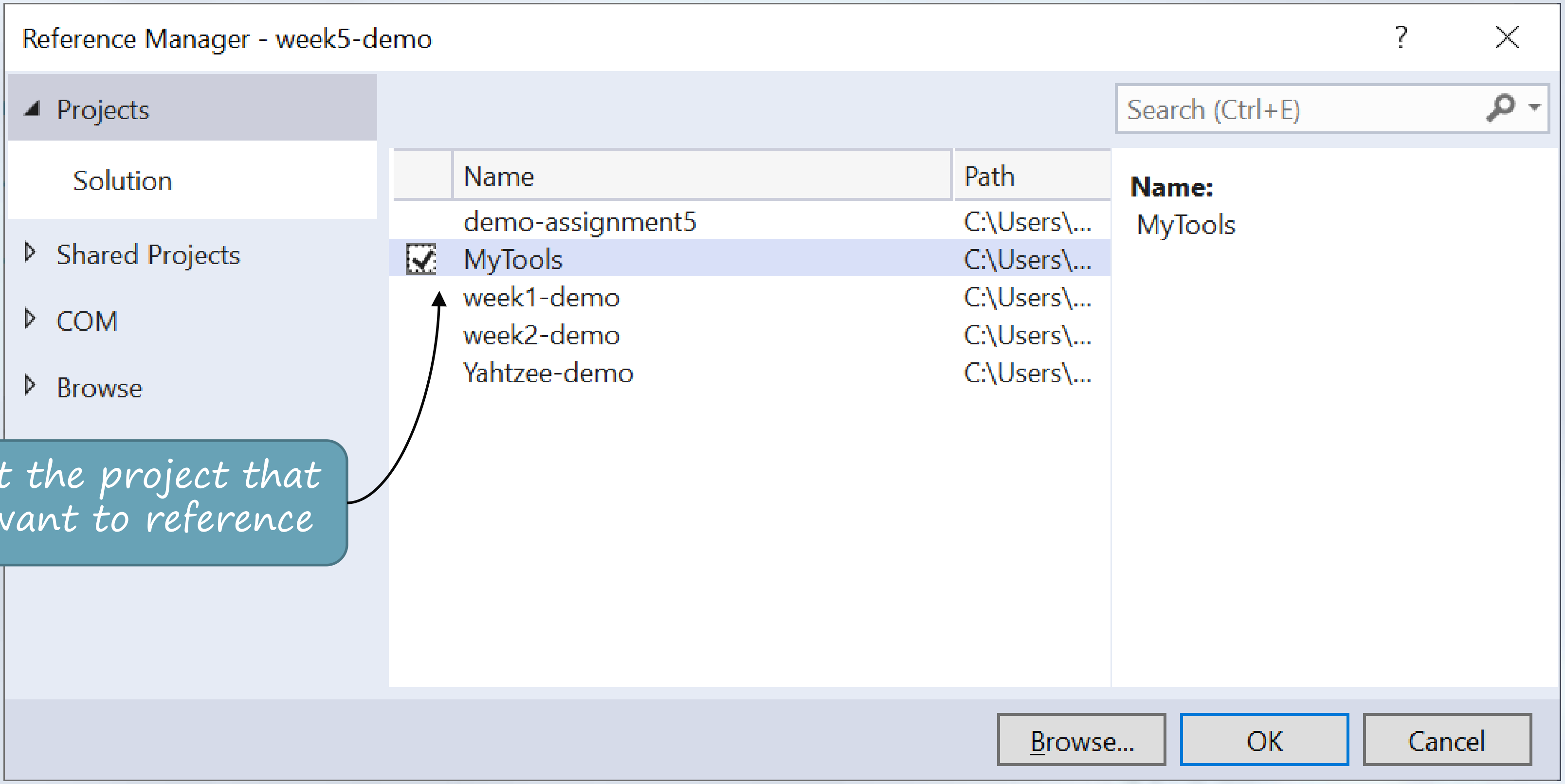
Using a Class Library



A project needs a reference to a Class Library

(right-mouse click on project, Add | Project Reference...)

Using a Class Library



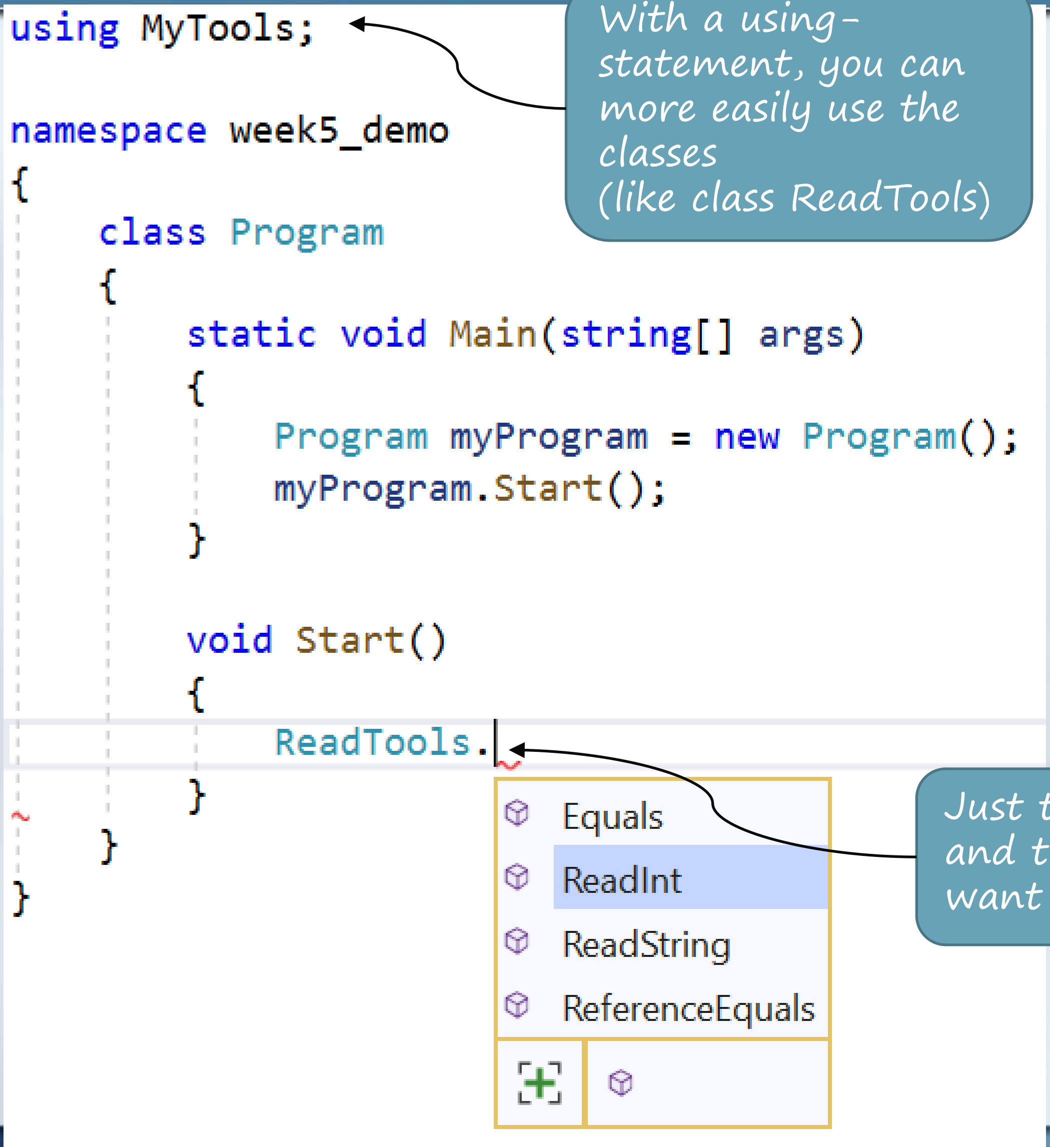
Reference Manager - week5-demo

	Name	Path
	demo-assignment5	C:\Users\...
☑	MyTools	C:\Users\...
	week1-demo	C:\Users\...
	week2-demo	C:\Users\...
	Yahtzee-demo	C:\Users\...

Name:
MyTools

Select the project that you want to reference

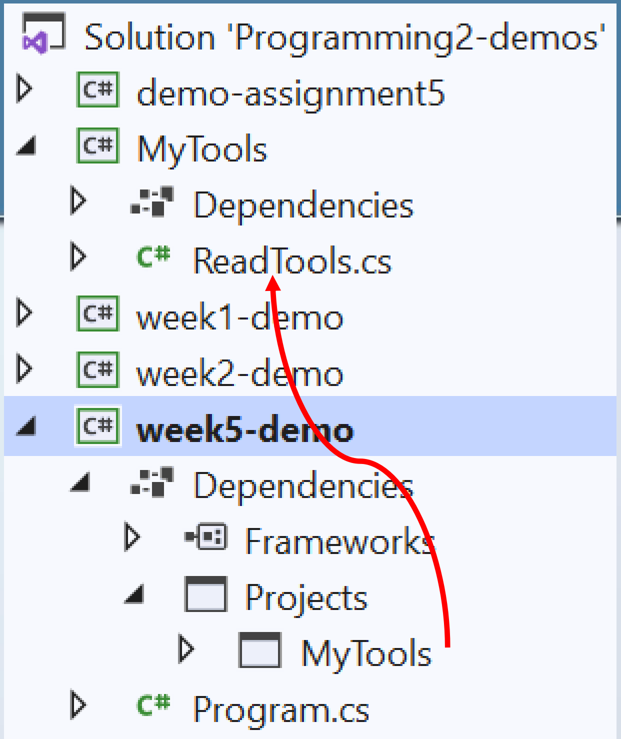
Using a Class Library



```
using MyTools;

namespace week5_demo
{
    class Program
    {
        static void Main(string[] args)
        {
            Program myProgram = new Program();
            myProgram.Start();
        }

        void Start()
        {
            ReadTools.
        }
    }
}
```

With a using-statement, you can more easily use the classes
(like class ReadTools)

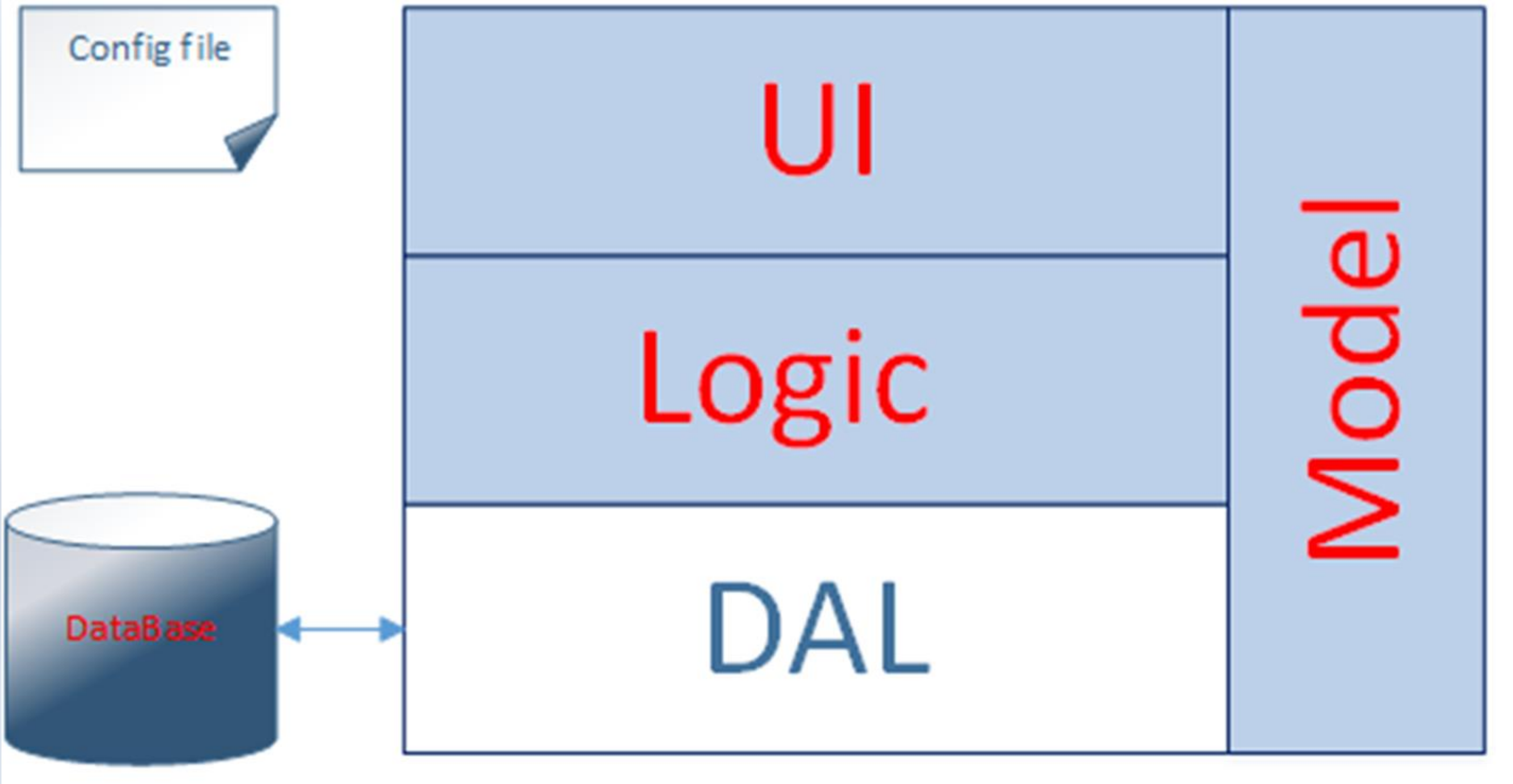
Just type ReadTools. and the method you want to use



Project week5-demo has a reference to library MyTools

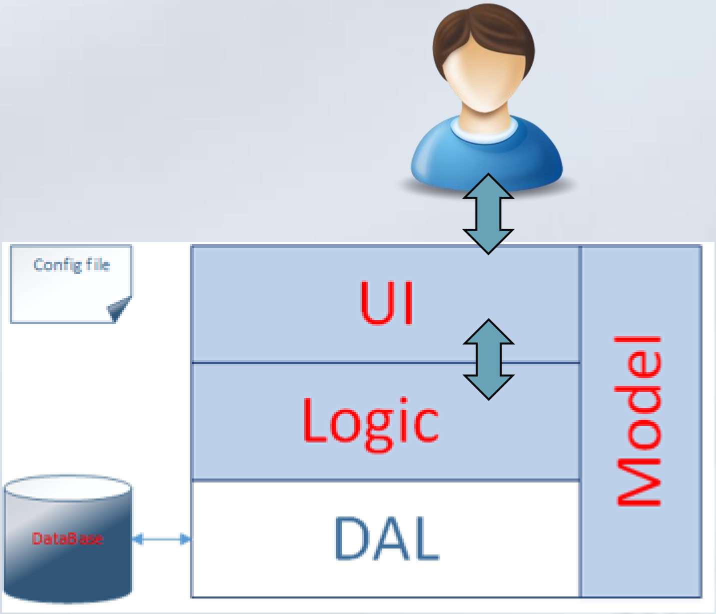
Layered Architecture

Layered Architecture



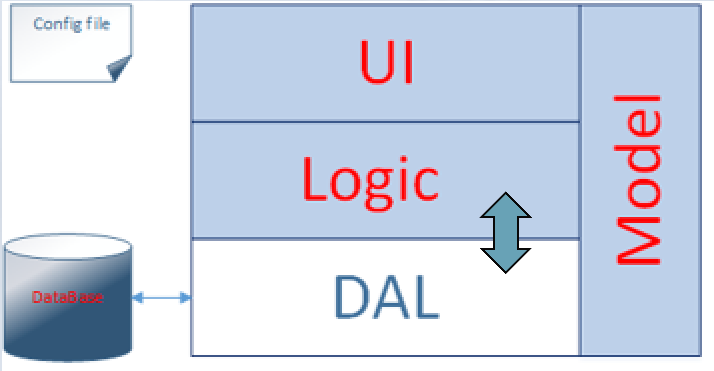
Layers: User Interface Layer (UI)

- This layer contains the actual application
- Responsible for contact with the user (input and output)
- Communicates with the Logic layer



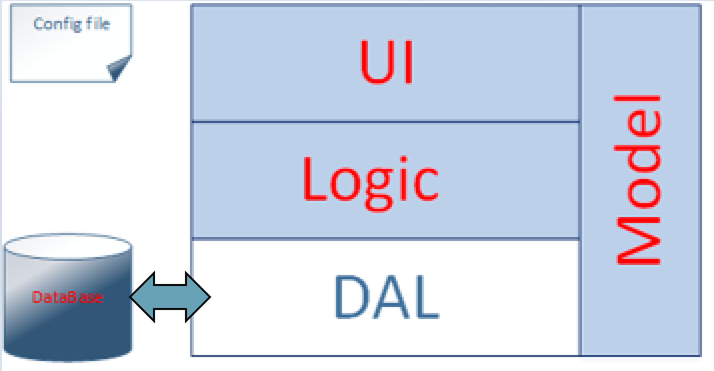
Layers: Logic Layer

- The logic layer contains the core of the system
- It contains the (bussiness) logic
- The logic layer contains classes with methods to process the core functionalities of the application
- The logic layer delegates all persistence/database functionality to the DAL layer



Layers: Data Access Layer (DAL)

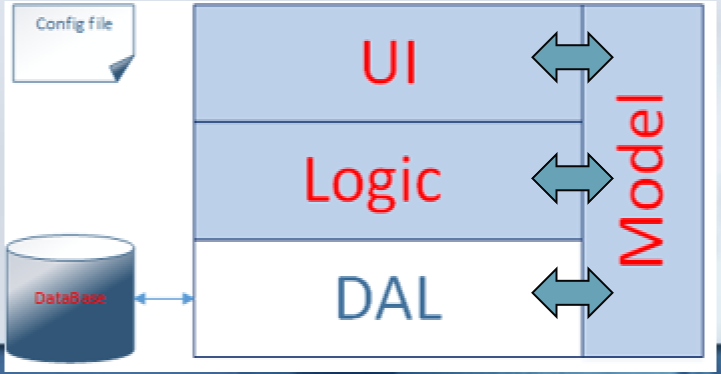
- A Library (DLL) to access the database
- The methods return ‘model’ objects
- DAL is responsible for converting (database) data to objects, and vice versa
- SQL is only used in the DAL layer *(create/insert, read/select, update, delete)*



Layers: Model

- Contains Model objects
  - Model objects represent the ‘things’ in the systeem
  - Model objects are used in all layers
  - e.g. Person, Meeting, Customer, Book, Card, Account,

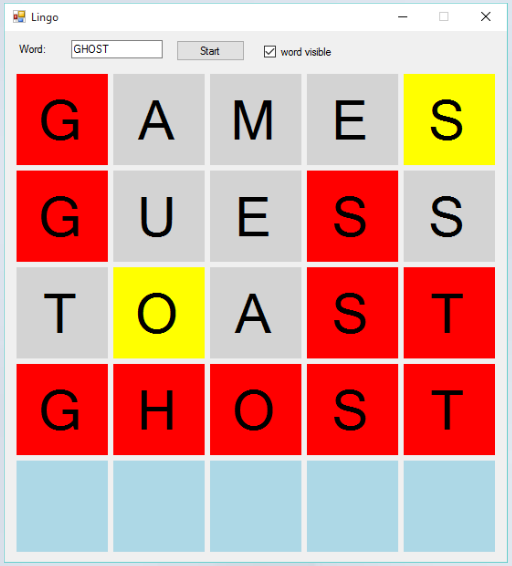
  - When a Model object is returned from the DAL layer, all fields of this object are filled *(with coherent data)* This means we don’t work with half-filled objects!



Exercise

Exercise: Lingo

- Design a Lingo game in which the user must guess a 5-letter word.
- The user gets 5 attempts; in each attempt the user enters a (5-letter) word and receives feedback on this word: which letters are correct, which letters are not correct, which letters are present but at wrong position.



Word: GHOST

☑ word visible

G	A	M	E	S
G	U	E	S	S
T	O	A	S	T
G	H	O	S	T

Exercise: Lingo

lingo word = ghost

player word = games

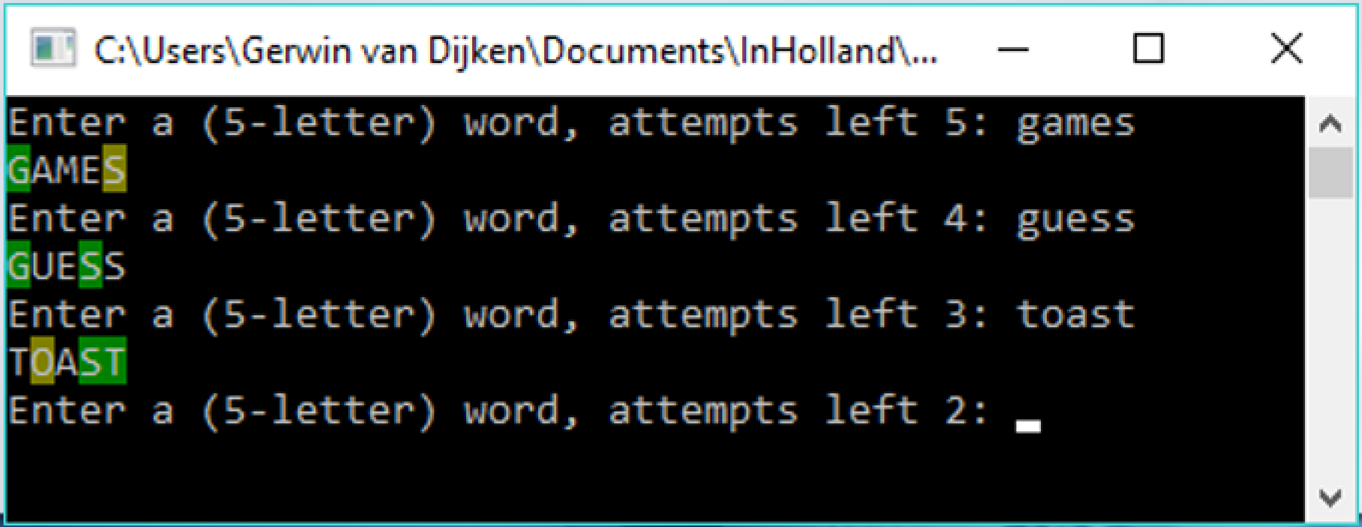
state[0] = **correct**
state[1] = incorrect
state[2] = incorrect
state[3] = incorrect
state[4] = **wrong-position**

player word = guess

state[0] = **correct**
state[1] = incorrect
state[2] = incorrect
state[3] = **correct**
state[4] = incorrect (!)

player word = toast

state[0] = incorrect (!)
state[1] = **wrong-position**
state[2] = incorrect
state[3] = **correct**
state[4] = **correct**



```
Enter a (5-letter) word, attempts left 5: games
GAMES
Enter a (5-letter) word, attempts left 4: guess
GUESS
Enter a (5-letter) word, attempts left 3: toast
TOAST
Enter a (5-letter) word, attempts left 2: _
```

Exercise: Lingo

- top-down *(stepwise refinement)*

  1) define the main task of the program
  2) define the subtasks (needed by main)
  3) define the subsubtasks

- bottom-up

  1) define the subtasks of the program
  2) define the main task of the program (call subtasks)

Exercise: Lingo

```
Start(filename)
        words = ReadWords(filename, 5)
        lingoWord = SelectWord(words)

        lingoGame = new LingoGame()
        lingoGame.Init(lingoWord)
        PlayLingo(lingoGame)

ReadWords(filename, wordLength)
        // read words with length <wordLength> from file...

SelectWord(words)
        // return random word from list
```

Exercise: Lingo

```
PlayLingo(lingoGame)
      attemptsLeft = 5
      wordLength = lingoGame.lingoWord.Length

      while attemptsLeft > 0 and !lingoGame.WordGuessed()
            playerWord = ReadPlayerWord(wordLength)
            letterResults = lingoGame.ProcessWord(playerWord)
            DisplayPlayerWord(playerWord, letterResults)

            attemptsLeft = attemptsLeft - 1

      return lingoGame.WordGuessed()
```

Exercise: Lingo

```
ReadPlayerWord(length)
      do
            word = ReadString()
      while (word.Length <> length)
      return word


DisplayPlayerWord(playerWord, letterResults)
    for i = 0 to playerWord.Length - 1
        if (letterResults[i] = LetterState.Correct)
            BackgroundColor = DarkGreen
        else if (letterResults[i] = LetterState.WrongPos)
            BackgroundColor = DarkYellow

        display playerWord[i]
        ResetColor()
```

Exercise: Lingo

```
[class LingoGame]

public enum LetterState { Correct,Incorrect,WrongPosition }

public string lingoWord
public string playerWord

Init(lingoWord)
      this.lingoWord = lingoWord
      this.playerWord = ""

WordGuessed()
      return lingoWord = playerWord
```

Exercise: Lingo

Lingo word	:	T	R	O	O	P
Player word	:	O	R	D	E	R

(reference letters: T O O P)

[class LingoGame]

```
ProcessWord(playerWord)
    this.playerWord = playerWord
    letterResults = new LetterState[lingoWord.Length]

    refLetters = new List<char>()
    for i = 0 to lingoWord.Length - 1
        if lingoWord[i] <> playerWord[i]
            refLetters.Add(lingoWord[i])

    ... (see next slide)
```

Exercise: Lingo

Lingo word : T R O O P

Player word : O R D E R

(reference letters: T O O P)

```
ProcessWord(playerWord)

    ... (see previous slide)

    for i = 0 to playerWord.Length - 1
        if lingoWord[i] = playerWord[i]
            letterResults[i] = LetterState.Correct
        else
            if refLetters.Contains(playerWord[i])
                letterResults[i] = LetterState.WrongPosition
                refLetters.Remove(playerWord[i])
            else
                letterResults[i] = LetterState.Incorrect

    return letterResults
```

Homework

- Read paragraphs ‘Yellow Book’
  (references can be found on Moodle)

- Assignments week 5
  (can be found on Moodle)