C#
FUNDAMENTALS

SESSIE 4/5

PLAN DAG 4

- Eventuele uitloop
- Exceptions
- Input / output
- Debugging
- Unit testing

EXCEPTIONS

- Exceptions are thrown during the so called unhappy path
- You can specify a backup plan for when exceptions occur (using a try/catch block)
- If you don’t there is a default exception handler

EXCEPTION HANDLING - TRY/CATCH

- Try block: contains the code that might throw an exception
- Catch block: specifies what exception to catch, there can be more than one catch block
- If the catch block doesn't catch an exception thrown in a try, the executor will check for surrounding try catch, or try catch higher in the stack. If there's none, it will use the default exception handler

```
int x = 0;
    try
    {
        int y = 100 / x;
    }
    catch (ArithmeticException e)
    {
        Console.WriteLine($"ArithmeticException
Handler: {e}");
    }
```

MULTI-CATCH

- Most specific catch block should come before less specific catch block
- The exception can only be caught by one block

```
int x = 0;
    try
    {
      int y = 100 / x;
    }
    catch (ArithmeticException e)
    {
      Console.WriteLine($"ArithmeticException Handler: {e}");
    }
    catch (Exception e)
    {
      Console.WriteLine($"Generic Exception Handler: {e}");
    }
```

FINALLY

- Finally block executed after try or catch block
- Sometimes finally block is not ran when an unhandled exception occurs
- Commonly used to clean up resources, like streams and database connections

```
string path = @"c:\users\public\test.txt";
System.IO.StreamReader file = new System.IO.StreamReader(path);
char[] buffer = new char[10];
try
{
    file.ReadBlock(buffer, index, buffer.Length);
}
catch (System.IO.IOException e)
{
    Console.WriteLine("Error reading from {0}. Message = {1}", path, e.Message);
}

finally
{
    if (file != null)
    {
        file.Close();
    }
}
```

DIFFERENT KIND OF EXCEPTIONS

- Usage error: should be avoided by coding well, such as NullReferenceException
- Program errors: can't be solved by writing better code, for example, wrong user input
- System failures: can't be handled using code, e.g. OutOfMemoryException

DO'S AND DON'TS IN EXCEPTION HANDLING

DO'S

- Use finally to clean up resources when "using" is not an option
- Check for conditions that might throw exceptions instead of catching these exceptions, especially when it could occur frequently
- Throw exceptions instead of returning error codes
- Rollback on exception

DONT'S

- Don't catch exception that come from poor coding, such as NullReferenceException
- Don't catch Exception, too broad
- Don't make custom exceptions when you can use a built in one just as well

CUSTOM EXCEPTIONS

- Convention: end the name with – Exception
- Use these three constructors: Exception(), Exception(string message), Exception(string message, Exception e)

```
static void Main(string[] args)
{
    try
    {
        string naam = "Dummynaam0";
        ValideerNaam(naam);
        Console.WriteLine(naam);
    }
    catch (InvalidNameException ex)
    {
        Console.WriteLine(ex.Message);
    }
}

private static void ValideerNaam(string controleernaam)
{
    // Zoek naar nummers en bijzondere karakters
    Regex regex = new Regex("^[a-zA-Z]+$");

    if (!regex.IsMatch(controleernaam))
        throw new InvalidNameException();
}

public class InvalidNameException : Exception
{
    public InvalidNameException()
    {
        Console.WriteLine("Ongeldige naam");
    }
}
```

EXERCISE

- Create an application that asks for a favorite number
- Show the favorite number squared
- Catch the exception you might get when someone enters text
- And then prompt again (optional)
- Optionally, also catch the exception you get when the number is too big and it overflows (use a ulong as type for the number otherwise it won't ever overflow)

FILES

READING AND WRITING TO FILES AND DATA STREAM

Use namespace System.IO

Classes

Directory – Create, move, and enumate through directories.

File - Creation, copying, deletion, moving, and opening of a single file.

FileStream – (A)synchronous read and write operations

StreamWriter - Writing characters to a stream

StreamReader - Reads characters from a byte stream

Path - File or directory path information.

CREATE DIRECTORIES AND FILES

Create and change directories

```
string path = @“c:\Backup";

DirectoryInfo directory;
Directory.CreateDirectory(path);

directory.Delete();
directory.Exists(path)
string [] fileEntries = directory.GetFiles(path);
```

Create Files

```
string path = @“c:blabla\Backup.txt";
if(!File.Exists(path)){
  using (StreamWriter sw = File.CreateText(path))
  {
    sw.WriteLine("Hello World");
  }
}
```

Open the file to read from

```
using (StreamReader sr = File.OpenText(path))
 {
    string text = "";
    while ((text = sr.ReadLine()) != null)
    {
        Console.WriteLine(text);
     }
}
```

READ AND WRITE TO FILES

Read a text file with the File.ReadAllText() method

```
string textfile = @”C:\Test.txt”;
string text = File.ReadAllText(textFile);
Console.WriteLine(text);
```

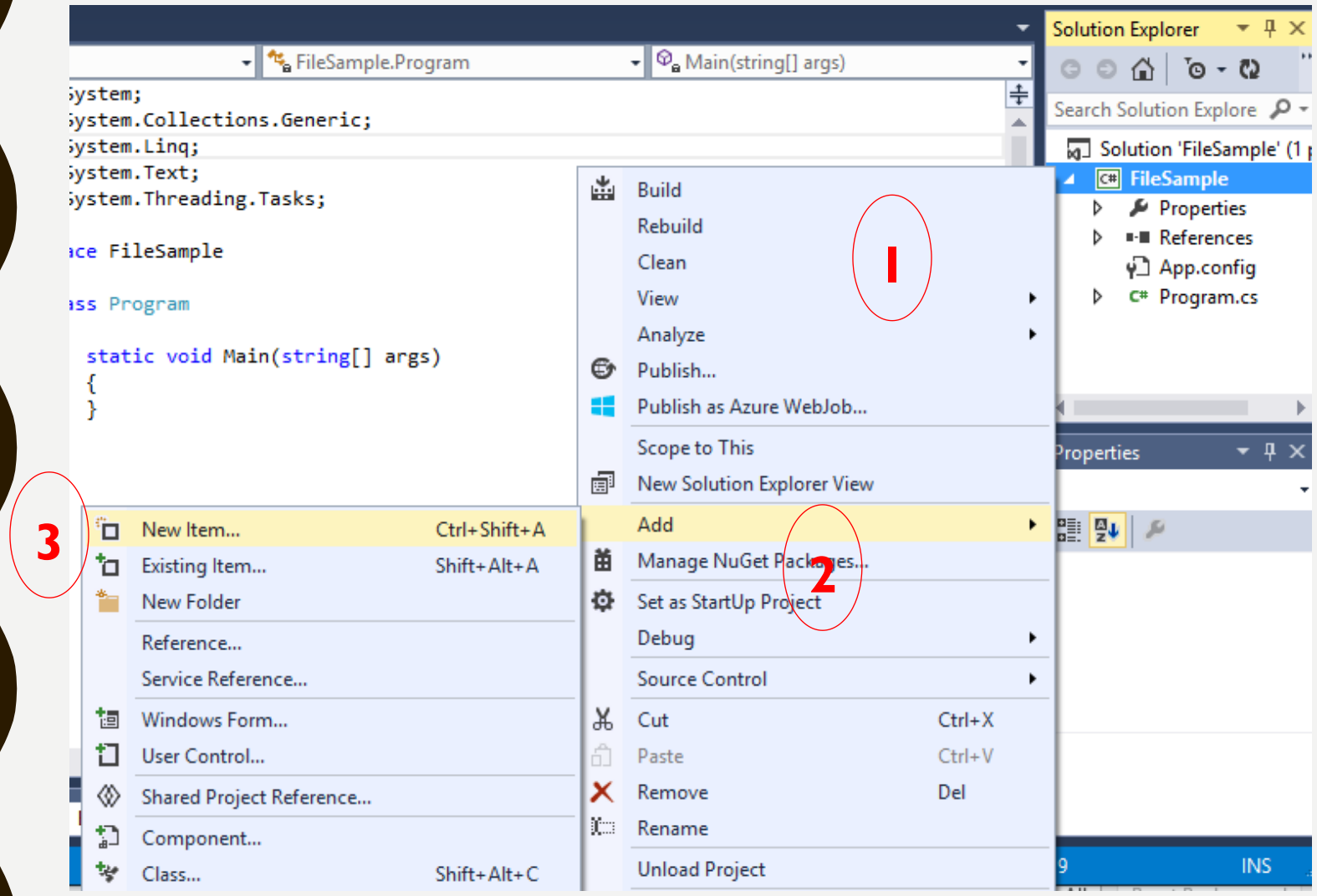
Read a text file line by line.

```
string[] lines = File.ReadAllLines(textFile);
foreach (string line in lines)
Console.WriteLine(line);
```

Create a text file with the File.CreateText method

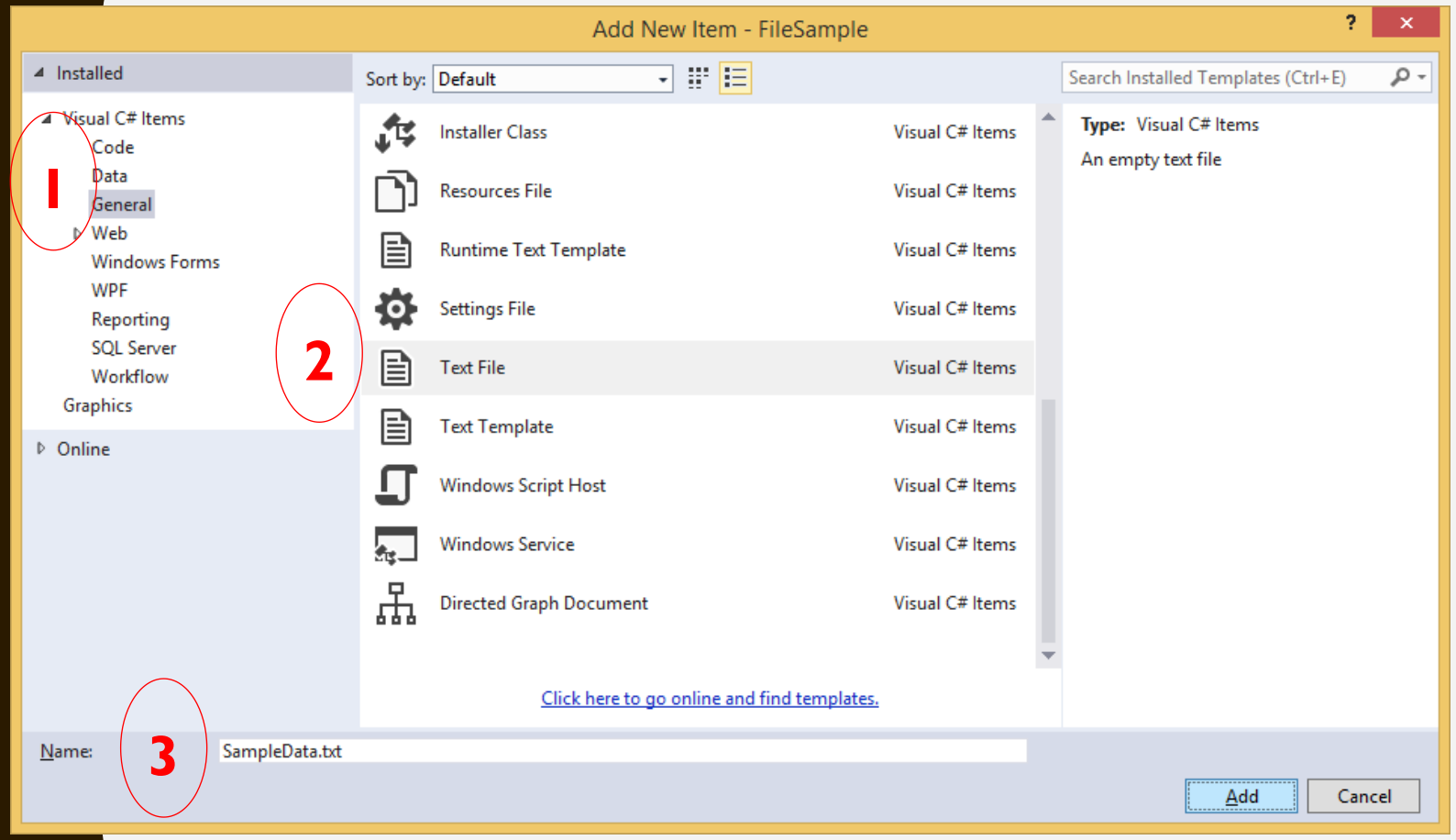
```
string textfile = @”C:\Test.txt”;
if (!File.Exists(path))
{
        using (StreamWriter sw = File.CreateText(path))
        {
                sw.WriteLine("Hello World");
        }
}
```

ADD TEXT FILE



1. Right-click project
2. Click Add
3. Click New Item…

NAME NEW TEXT FILE



1. Select "General"
2. Select "Text File"
3. Name it.

UPDATE TEXT FILE AND PROPERTIES

FileSample - Microsoft Visual Studio

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP

Shahed Chowdhuri SC

Quick Launch (Ctrl+Q)

Start Debug

Server Expl...
- Azure
- Data Connect
- Servers
- SharePoint Co

SampleData.txt Program.cs

```
1 my sample data line 1
2 another line
```

146 %

enter sample data

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'FileSample' (1 project)
- FileSample
  - Properties
  - References
  - App.config
  - Program.cs
  - SampleData.txt

Properties

SampleData.txt File Properties

Build Action	Content
Copy to Output Directory	Copy always
Custom Tool	
Custom Tool Namespace	
File Name	SampleData.txt
Full Path	C:\ work\presentations\FileSar

Copy to Output Directory

Specifies the source file will be copied to the output directory.

Set to "Copy always"

Toolbox Server Ex... Web Publish Act... Error List Output Find Results 1 Find Symbol Res... Package Manag...

Ready

READ FILE, HANDLE EXCEPTIONS

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Use System.IO namespace for StreamReader

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            using (StreamReader sr = new StreamReader("SampleData.txt"))
            {
                String line = sr.ReadToEnd();
                Console.WriteLine(line);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine("The file could not be read:");
            Console.WriteLine(e.Message);
        }

        Console.ReadKey();
```

Handle possible exceptions with try-catch block

EXERCISE: CREATE DIRECTORY AND READ FILES

Make 3 folders on your c drive. Movies, Vacation and Files.

Create a list of your favorite movies.

Write the list to a textfile in the folder Movies.

Make a new textfile in your project with your 3 favorite vacation destinations.
Every vacation should be on a new line.
Read the file, line by line and display it on your screen.

DEMO (READ TEXT FILE)

FileSample - Microsoft Visual Studio

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            using (StreamReader sr = new StreamReader("SampleData.txt"))
            {
                String line = sr.ReadToEnd();
                Console.WriteLine(line);
            }
        }
        catch (Exception e)
        {
            Console.WriteLine("The file could not be read:");
            Console.WriteLine(e.Message);
        }

        Console.ReadKey();
    }
}
}
```

file:///C:/_work/presentati...

```
my sample data line 1
another line
```

USE CODE TO WALK FILES

- Several strategies, but for now we'll only discuss recursion to alk a directory tree
- This could go wrong when the three is too large or deep
- Let's not worry too much about that now

```
static void WalkDirectoryTree(System.IO.DirectoryInfo root)
  {
      System.IO.FileInfo[] files = null;
      System.IO.DirectoryInfo[] subDirs = null;

      // First, process all the files directly under this folder
      try
      {
          files = root.GetFiles("*.*");
      }
      // This is thrown if even one of the files requires permissions greater
      // than the application provides.
      catch (UnauthorizedAccessException e)
      {
          // This code just writes out the message and continues to recurse.
          // You may decide to do something different here. For example,
you
          // can try to elevate your privileges and access the file again.
           Console.WriteLine(e.Message);
      }
catch (System.IO.DirectoryNotFoundException e)
      {
          Console.WriteLine(e.Message);
      }

      if (files != null)
      {
          foreach (System.IO.FileInfo fi in files)
          {
              // In this example, we only access the existing FileInfo object. If we
              // want to open, delete or modify the file, then
              // a try-catch block is required here to handle the case
              // where the file has been deleted since the call to TraverseTree().
              Console.WriteLine(fi.FullName);
          }
         }
          // Now find all the subdirectories under this directory.
          subDirs = root.GetDirectories();

          foreach (System.IO.DirectoryInfo dirInfo in subDirs)
          {
              // Recursive call for each subdirectory.
              WalkDirectoryTree(dirInfo);
          }

  }
```

GAME: CATCH THE MOUSE

Task 0

- Write the code to create a folder structure that resembles a house. For each room at a txt. Also add a park folder with a mouse wonderland txt file to it. In one of the text files of the house, add our mouse Micky. But Micky is clever… He is disguised as 13931125

Task 1

- We want to catch the mouse. The mouse is hiding somewhere in the file directory under the resources in the main folder. However, we don't have a stupid mouse. He's not hiding out in the open. The mouse name is Micky, but as mice do, he has a secret name. You can unlock the secret name by completing the method in the CatchTheMouse class. There should be a method ChangeNameToSecretName. This method needs to be implemented in a way that it will convert the name the following way: every letter has to be replaced with it's corresponding number in the alphabet. So a = 1, b = 2, etc. Ab will become 12.

GAME: CATCH THE MOUSE

Task 2

- Now we have the code to find the secret name of the mouse, we can look for the mouse. Walk the files in the resources folder under the main folder of the project. Do this in the method FindTheMouse in the CatchTheMouse class. In one of these files you'll find the mouse and a stressed programmer that wants the mouse to move out.

Task 3

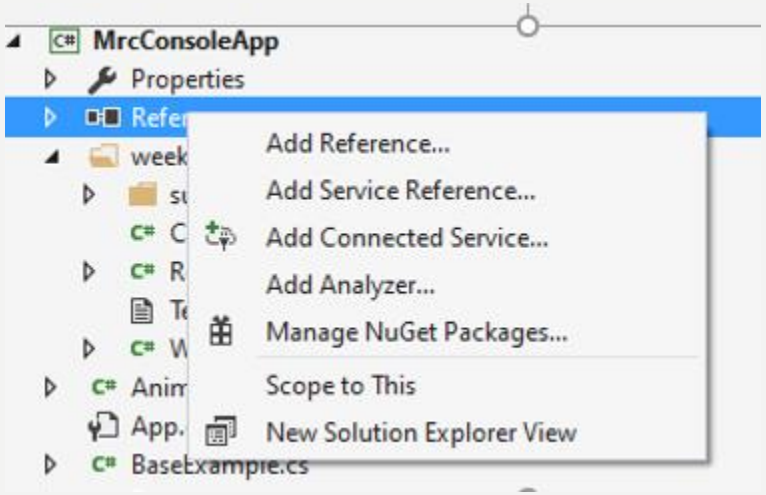
- Now that you have found the mouse, it is time to catch it and release it elsewhere. Of course, we don't want to kill the mouse. So you'll have to do the following things in the method AnimalFriendlyMouseRemoval in the CatchTheMouse class. :

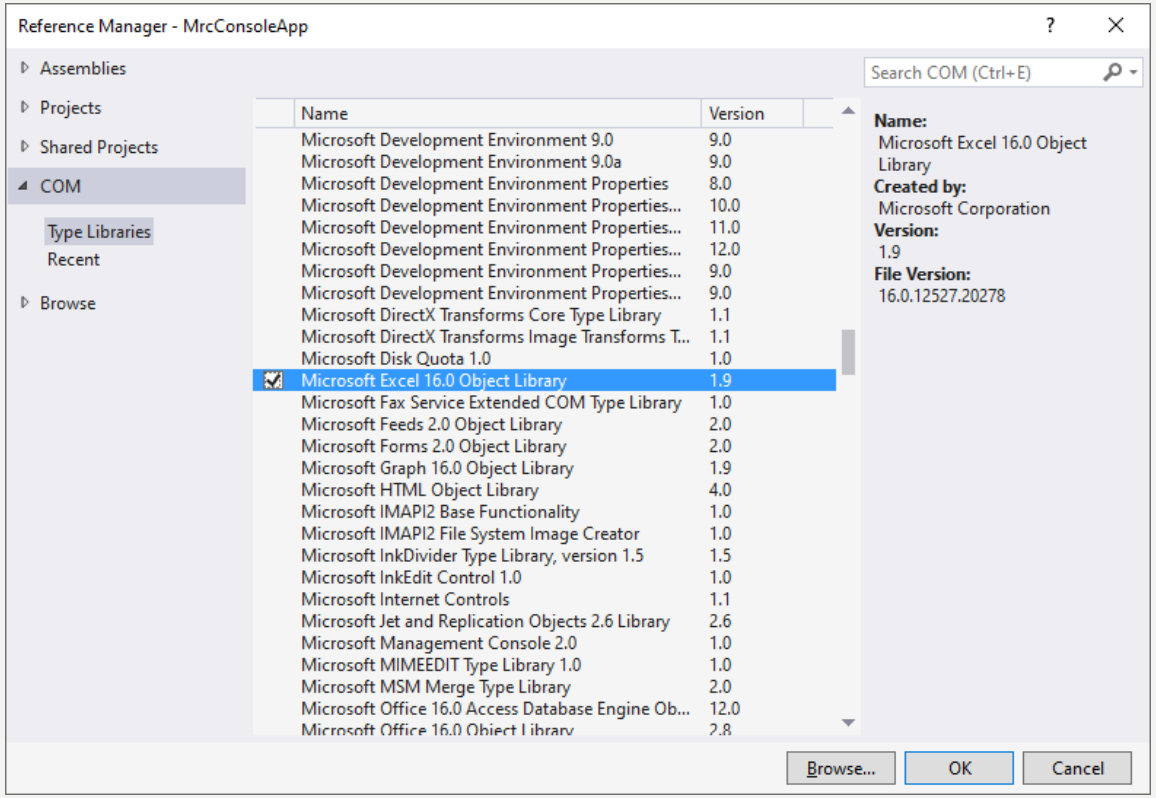
– remove the mouse from the file where you found it, you can assume that the mouse is always on it's own line in the file
– add the mouse to the file /park/farAwayFromMyHouseMouseWonderland.txt
– change the location of the mouse to the new location

WORKING WITH EXCEL FILES

- Add a reference
- Create the COM objects
- Read
- Clean up

ADD REFERENCE





READ FROM EXCEL

```
public static void ReadExcelFile(string path, int sheet)
{
    //create COM objects
    _Application excel = new Microsoft.Office.Interop.Excel.Application();
    Workbook wb = excel.Workbooks.Open(path);
    Worksheet ws = wb.Worksheets[sheet];

    //read cell
    //excel starts counting from one, so A1 = 1,1
    //value2 is best to use, cause that will give back the underlying value of the cell
    int i = 1;
    int j = 1;
    // this will only read cells 1,1 2,2 3,3
    while (ws.Cells[i, j].Value2 != null)
    {
        Console.WriteLine(ws.Cells[i, j].Value2);
        i++;
        j++;
    }

    //clean up
    wb.Close();
    excel.Quit();


}
```

EXERCISE

- Create a player class that contains the name of the player and the age of the player and some other relevant stuff for a yahtzee player
- Create an excel file with columns that correspond with your player object
- Read from this excel file and create player objects for each row
- Add the players to a collection

WRITING TO EXCEL FILES

```
public static void WriteExcelFile()
{
    //create COM objects
    _Application excel = new Microsoft.Office.Interop.Excel.Application();
    Workbook wb = excel.Workbooks.Add("");
    Worksheet ws = wb.ActiveSheet;

    // Add table headers going cell by cell.
    ws.Cells[1, 1] = "First Name";
    ws.Cells[1, 2] = "Last Name";
    ws.Cells[1, 3] = "Full Name";
    ws.Cells[1, 4] = "Salary";

    //save
    excel.Visible = false;
    excel.UserControl = false;
    wb.SaveAs(@"C:\Users\maaik\OneDrive\Documenten\Visual Studio 2017\Projects\MrcConsoleApp\MrcConsoleApp\week 4\Map2.xlsx",
        false, false, Microsoft.Office.Interop.Excel.XlSaveAsAccessMode.xlNoChange,
        Type.Missing, Type.Missing, Type.Missing, Type.Missing, Type.Missing);

    //cleanup
    wb.Close();
    excel.Quit();
}
```

OTHER OPTIONS

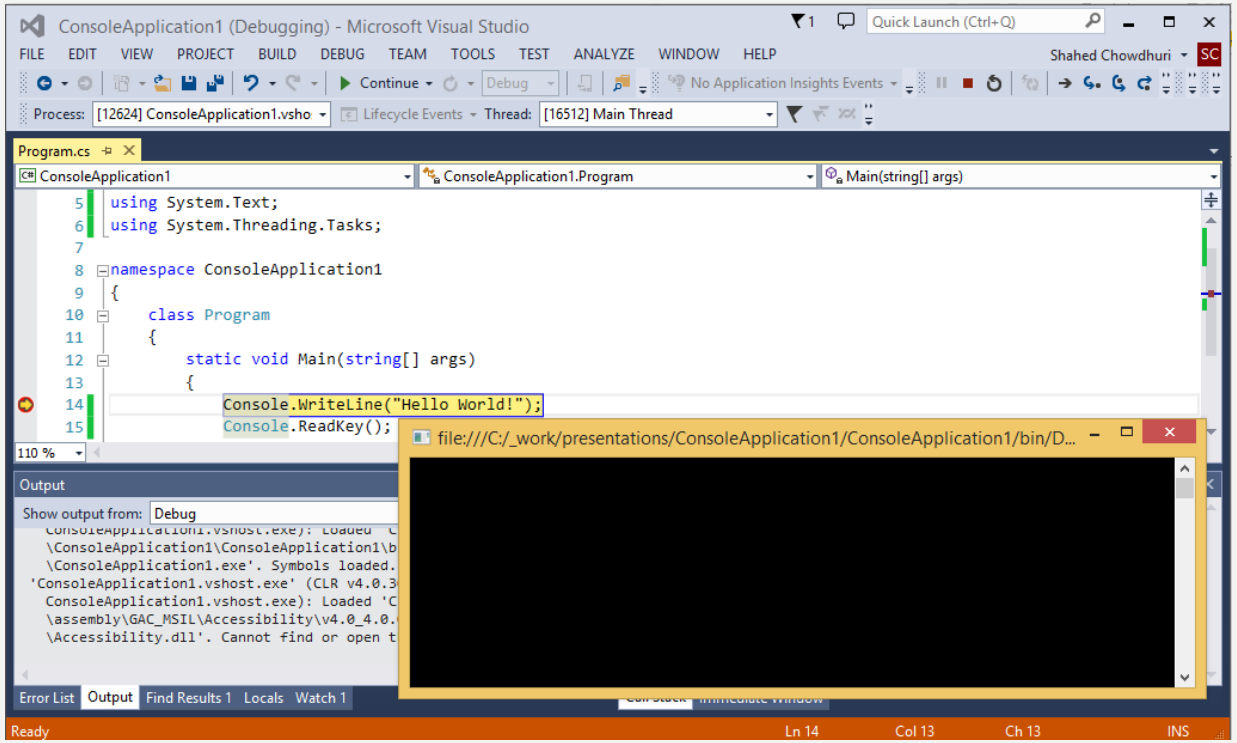
```
//Format A1:D1 as bold, vertical alignment = center.
ws.get_Range("A1", "D1").Font.Bold = true;
ws.get_Range("A1", "D1").VerticalAlignment =
    Microsoft.Office.Interop.Excel.XlVAlign.xlVAlignCenter;

// Create an array to multiple values at once.
string[,] saNames = new string[5, 2];

saNames[0, 0] = "John";
saNames[0, 1] = "Smith";
saNames[1, 0] = "Tom";

saNames[4, 1] = "Johnson";

//Fill A2:B6 with an array of values (First and Last Names).
ws.get_Range("A2", "B6").Value2 = saNames;

//Fill C2:C6 with a relative formula (=A2 & " " & B2).
Range rng = ws.get_Range("C2", "C6");
rng.Formula = "=A2 & \" \" & B2";

//Fill D2:D6 with a formula(=RAND()*100000) and apply format.
rng = ws.get_Range("D2", "D6");
rng.Formula = "=RAND()*100000";
rng.NumberFormat = "$0.00";
```

EXERCISE

- Create a shopping list
- Make headers for the item, the amount and the price excluding vat
- Calculate the 9% vat
- Bonus: Sum the total

DEBUGGING

- Find errors in logic that the compiler
- Use breakpoints (Press F9) to stop on a particular line of code
- Use Locals and Watch window to find variable values



DEMO DEBUGGING

EXERCISE

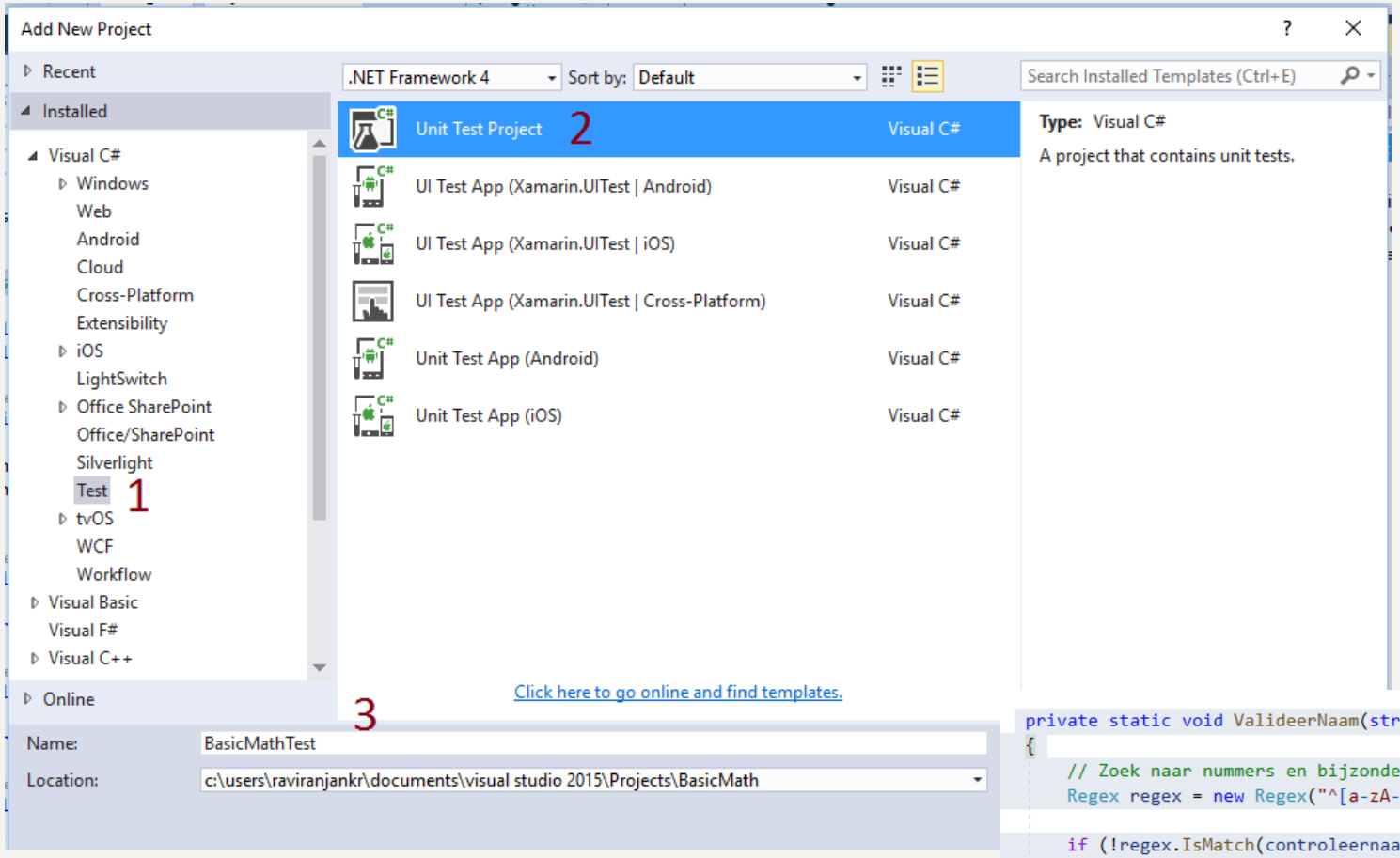
- Add breakpoints to your application
- Look at the variables changing
- See if you can step into the line
- See if you can step over the line

UNIT TESTING

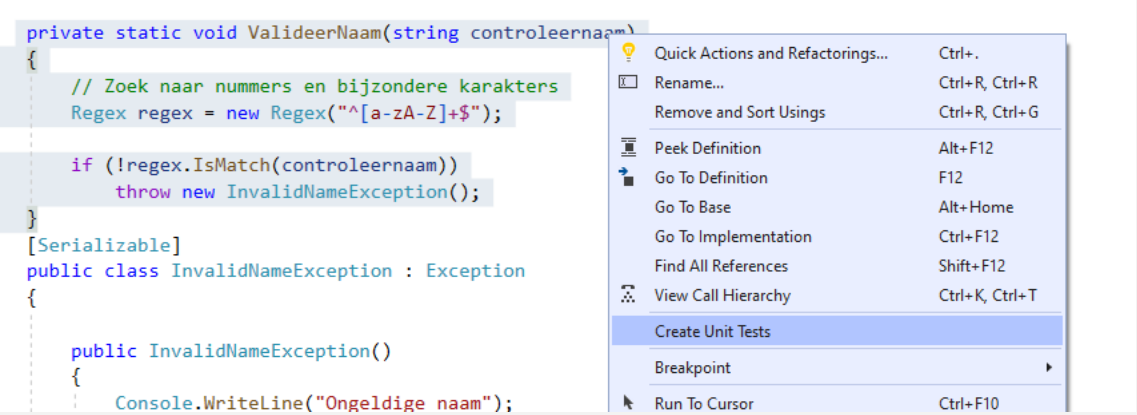
- Test small pieces of code
- The good news: we've been doing this partly in our main all the time!
- Mirror the structure and execution of a small part of code to test expected results
- Avoid logic in tests
- Use stub or mock object to mimmic real data
- Run tests
- Debug unit testing

```
[TestMethod()]
public void DebitTest()
{
    BankAccount b = new BankAccount("maaike", 100);
    b.Debit(50);
    Assert.AreEqual(50, b.Balance);
}
```

DEMO UNIT TESTING



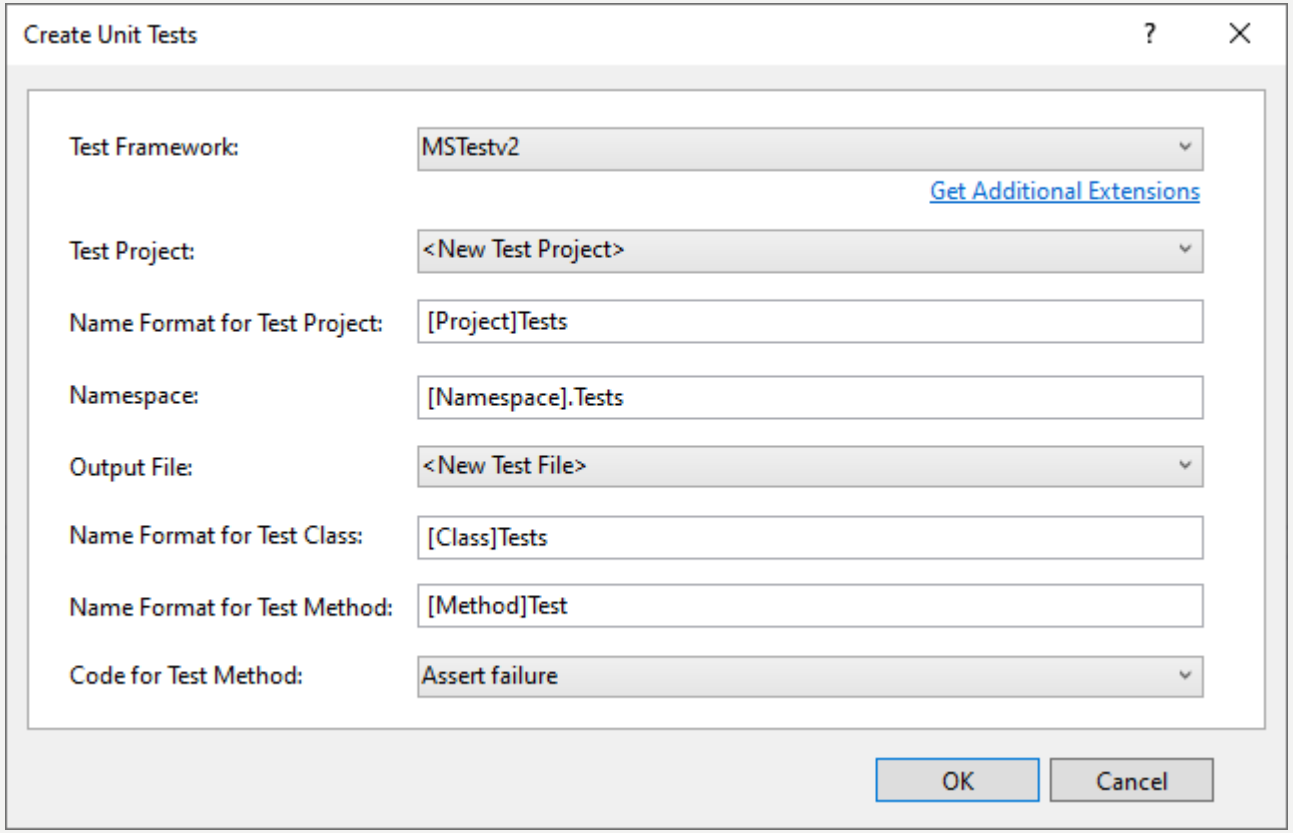
Create unit test, right click method



CLASS FOR TEST PURPOSES

```
{
    public class BankAccount
    {

        private double balance;

        public BankAccount(string nameClient, double balance)
        {
            NameClient = nameClient;
            this.balance = balance;
        }

        public string NameClient { get; set; }
        public double Balance { get { return balance; }  }
        public double MinBalance { get; set; }

        public void Debit(double debitAmount)
        {
            if((Balance - debitAmount) < MinBalance)
            {
                throw new ArgumentOutOfRangeException("balance too low");
            }
            balance -= debitAmount;
        }

        public void Credit(double creditAmount)
        {
            if (creditAmount <= 0)
            {
                throw new ArgumentOutOfRangeException("ccredit amount too low");
            }
            balance += creditAmount;
        }

    }
}
```

RIGHT CLICK + CREATE UNIT TESTS



DEFAULT TEST

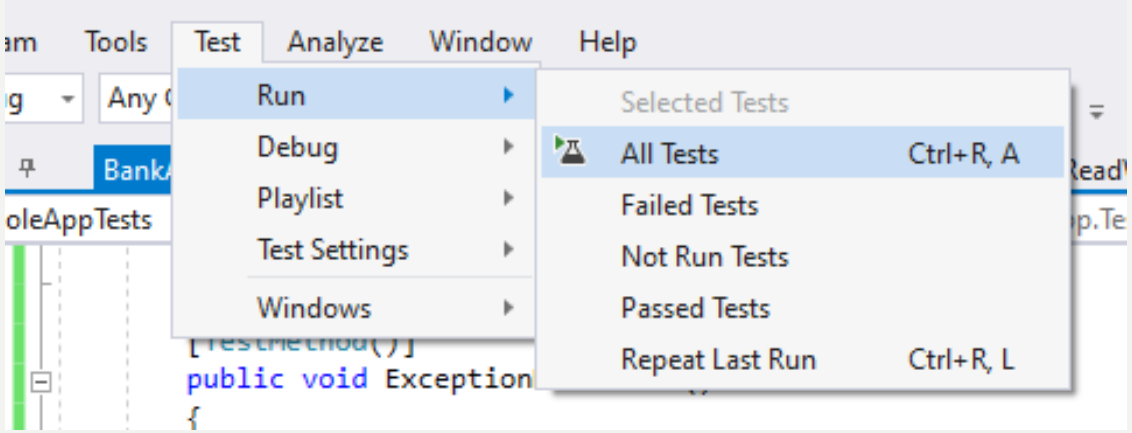
```
{
    [TestClass()]
    public class BankAccountTests
    {
        [TestMethod()]
        public void CreditTest()
        {
            Assert.Fail();
        }
    }
}
```

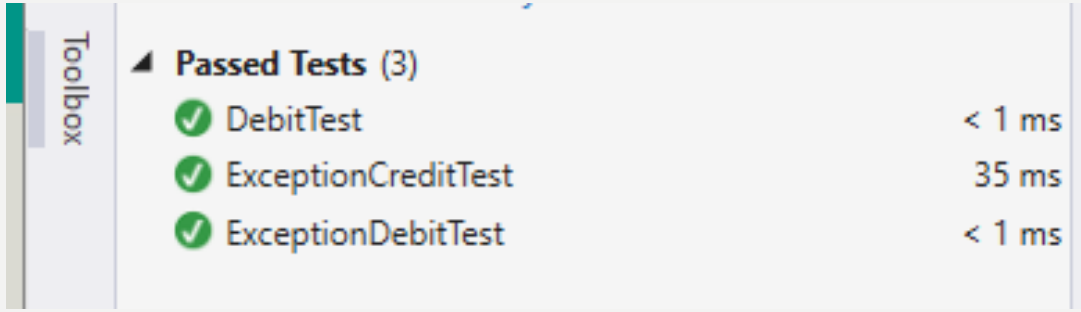
WRITE SOME LOGICAL TEST

```
[TestMethod()]
public void DebitTest()
{
    BankAccount b = new BankAccount("maaike", 100);
    b.Debit(50);
    Assert.AreEqual(50, b.Balance);
}
```

```
[TestMethod()]
public void ExceptionDebitTest()
{
    BankAccount b = new BankAccount("maaike", 100);
    try
    {
        b.Debit(101);
    }
    catch (ArgumentOutOfRangeException e)
    {
        StringAssert.Contains(e.Message, "too low");
        return;
    }
    catch (Exception e)
    {
        StringAssert.Contains(e.Message, "");
        return;
    }
    Assert.Fail("no exception thrown for debit with amount 0");
}
```

RUN THEM!


Tools
Test
Analyze
Window
Help
Run
Debug
Playlist
Test Settings
Windows
Selected Tests
All Tests Ctrl+R, A
Failed Tests
Not Run Tests
Passed Tests
Repeat Last Run Ctrl+R, L
public void Exception


Toolbox
Passed Tests (3)
DebitTest < 1 ms
ExceptionCreditTest 35 ms
ExceptionDebitTest < 1 ms

EXERCISE

Create a class with a method that multiplies two number

Add a method that divides two number

Create unit tests to check if the methods works, test as many scenarios as possible

NOG VRAGEN?

Stel ze via:

maaike.vanputten@brightboost.nl

Of WhatsApp:

0683982426