Error Management

Types of programming errors

Exceptions & Unit Tests

Overview

- Types of programming errors
- Testing Software
  - Test Case Example
- Exception Handling
  - Temperature Example
- Unit Tests

Types of programming errors

- Compiler Error
  - Easy to find
- Runtime Error
  - Debugging
  - Using Exceptions
- Logical Error
  - easy to miss
  - sometimes hard to find

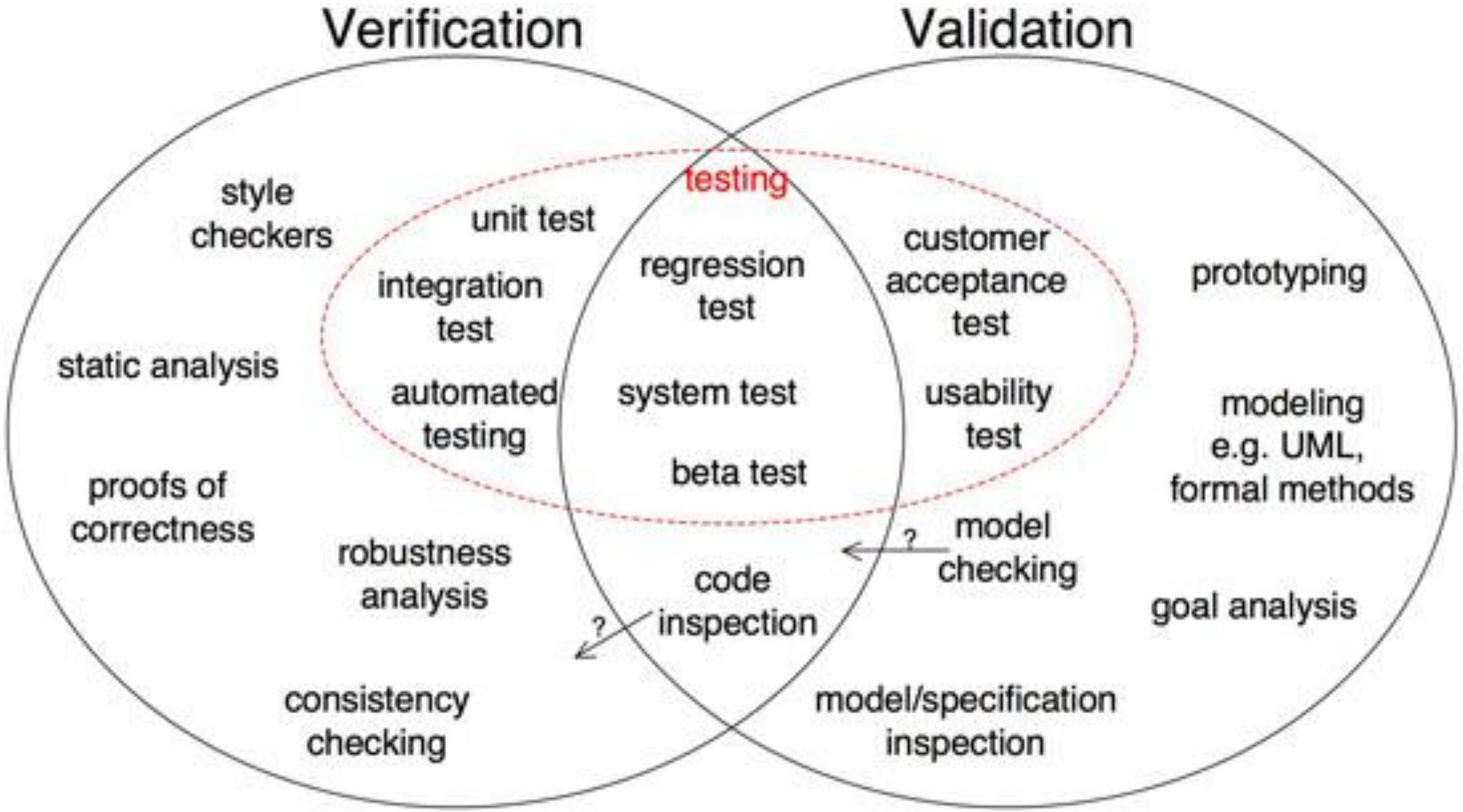
Types of program errors explained

- **Syntax errors / Compiler time errors**
  - errors due to the fact that the syntax of the language is not respected
  - errors due to an improper use of program statements.
- **Runtime errors**:
  - dynamic semantic errors, that cannot be detected by the compiler, so the program compiles, starts and than it crashes with an exception
- **Logical errors**:
  - errors due to the fact that the specification is not respected, the result is incorrect

Testing Software

executing a program with the intent of **finding the software bugs**

Validate & Verify



Testing Software

- Software testing is
  - **a process of validating and verifying** that a software application:
    - meets the technical requirements as designed
    - works as expected

Test Design

- Sometimes it’s referred as
  “verifying the test basis via the test design”.

- test basis:
  - includes documents such as
    the requirements and design specifications
- test design:
  - creating and writing test suites for testing a software

Requirements

- Test Cases have to be
  - Requriement oriented
    - anforderungsbezogen
  - Reproducible
    - reproduzierbar
  - Comprehensible
    - nachvollziehbar
  - Verifiable
    - überprüfbar

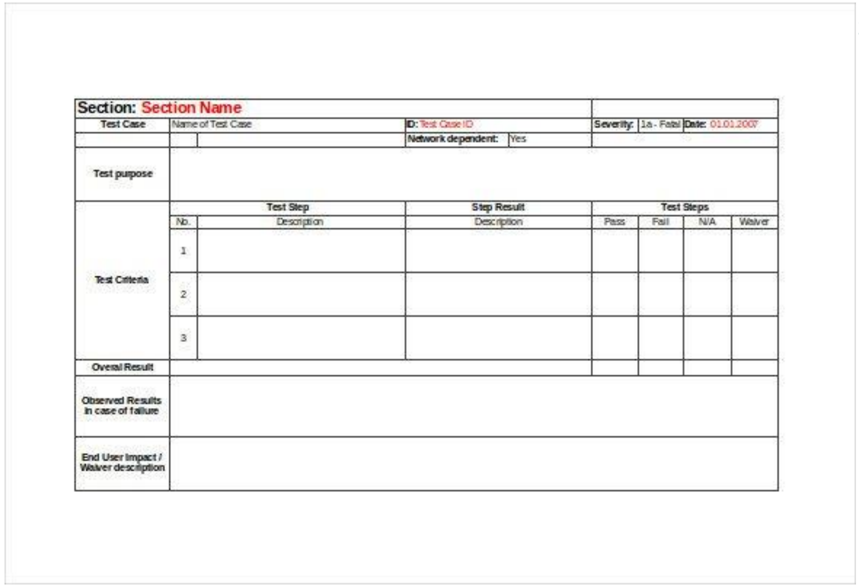
Chronology

1. Receive an Assignment & define the Performance Requirements

2. Design the Class Diagram & the **Test Cases**

3. Implement the Software
   using **Exceptions** for Error Management

4. Test the Software
5. Implement the Test Cases using **Unit Tests**



Section: Section Name							
Test Case	Name of Test Case		ID: Test Case ID	Severity:	1a - Fatal	Date:	01.01.2007
			Network dependent: Yes				
Test purpose							
Test Criteria	Test Step		Step Result	Test Steps			
	No.	Description	Description	Pass	Fail	N/A	Waiver
	1						
	2						
	3						
Overal Result							
Observed Results in case of failure							
End User Impact / Waiver description							

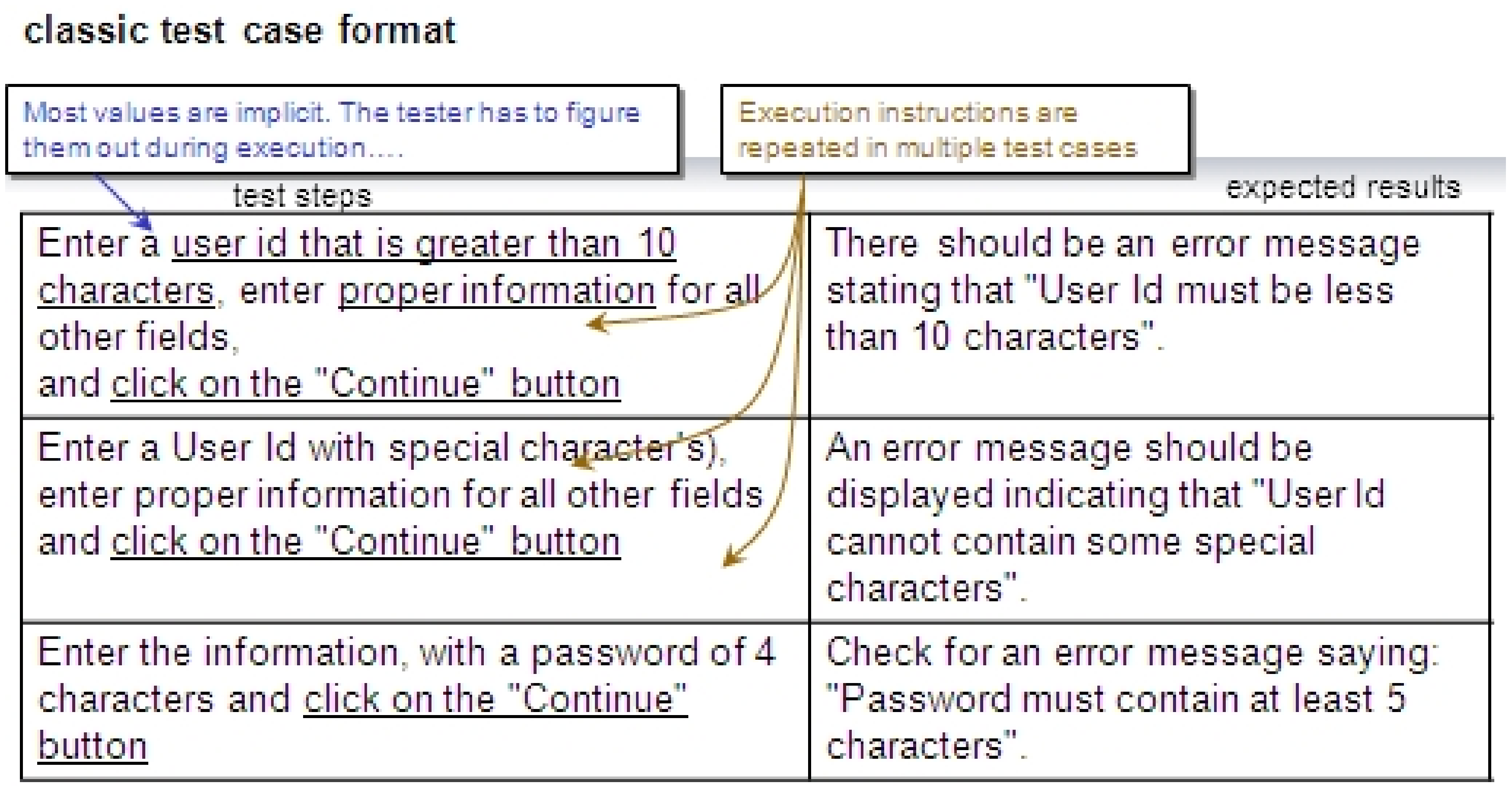
Test Case Examples

Define test cases and expected results

Write your program

See if you meet your expected results

Example for designing Test Cases:



classic test case format

Most values are implicit. The tester has to figure them out during execution....

Execution instructions are repeated in multiple test cases

test steps	expected results
Enter a user id that is greater than 10 characters, enter proper information for all other fields, and click on the "Continue" button	There should be an error message stating that "User Id must be less than 10 characters".
Enter a User Id with special character's), enter proper information for all other fields and click on the "Continue" button	An error message should be displayed indicating that "User Id cannot contain some special characters".
Enter the information, with a password of 4 characters and click on the "Continue" button	Check for an error message saying: "Password must contain at least 5 characters".

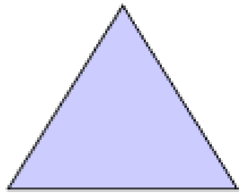
Example

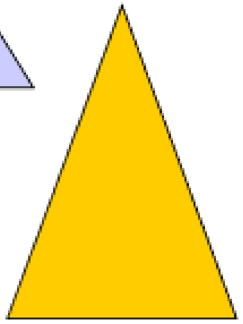
Wie kann man das folgende Programm testen?

Das Programm liest drei Werte ein. Diese werden als Längen von Dreiecksseiten interpretiert. Das Programm druckt eine Meldung mit der Feststellung aus, ob das Dreieck ungleichseitig, gleichschenklig oder gleichseitig ist.

(aus „Myers: Methodisches Testen von Programmen“)

- *gleichseitig: drei gleich lange Seiten*
- *gleichschenklig: mindestens zwei gleich lange Seiten*





Possible Test Cases

Kategorien von Testfällen:

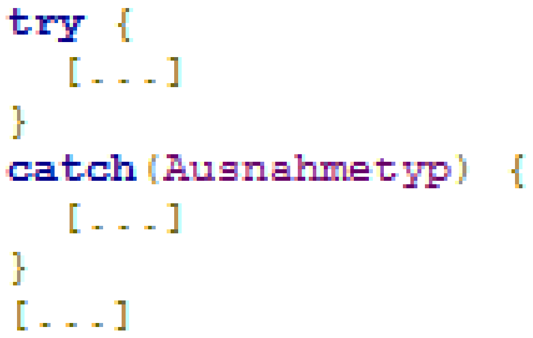
- 3 ganze Zahlen für ein zulässiges ungleichseitiges Dreieck
- 3 ganze Zahlen für ein zulässiges gleichseitiges Dreieck
- 3 ganze Zahlen für ein zulässiges gleichschenkliges Dreieck
- 3 ganze Zahlen: eine Seite gleich Null, alle drei Seiten gleich Null
- 3 ganze Zahlen: mind. eine Seite hat einen negativen Wert
- 3 ganze Zahlen: die Summe zweier Zahlen ist gleich oder kleiner der dritten
- auch nicht ganzzahlige Werte
- nicht typgerechte Eingabewerte

Beispiel für einen konkreten Testfall:

- Eingabe: 4,4,2
- Testbedingung: Die Ausgabe ist: „Das Dreieck ist gleichschenklig.“

Exception Handling

using Excpetions



```
try {
   [...]
}
catch(Ausnahmetyp)  {
   [...]
}
[...]
```

Before Exceptions

- If the user offers you incorrect values:
  - Write a Error Message to the Console
  - Show a Message Box with the Error Message

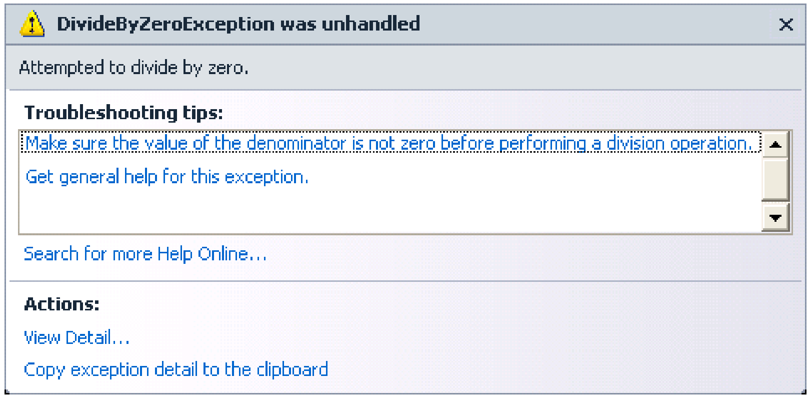
- Which one now?!
  - Throw an Exception an decide later ;-)

Exception Handling

- An exception is a problem that **arises during** the **execution of a program**.

- A C# exception is
  a response to **an exceptional circumstance**
  that arises while a program is running,

  such as an attempt
  to divide by zero.



DivideByZeroException was unhandled

Attempted to divide by zero.

Troubleshooting tips:

Make sure the value of the denominator is not zero before performing a division operation.

Get general help for this exception.

Search for more Help Online...

Actions:

View Detail...

Copy exception detail to the clipboard

Syntax & Keywords

- Exceptions provide a way to transfer control from one part of a program to another.

- C# exception handling is built upon four keywords: **try**, **catch**, **finally**, and **throw**.

Try & Catch

- **try**: A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.

- **catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem.

  The catch keyword indicates the catching of an exception.

Finally & Throw

- **finally**: The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown.

  For example, if you open a file, it must be closed whether an exception is raised or not.

- **throw**: A program throws an exception when a problem shows up. This is done using a throw keyword.

Syntax

```
try {
    // statements causing exception
}
catch( ExceptionName e1 ) {
    // error handling code
} catch( ExceptionName e2 ) {
    // error handling code
} catch( ExceptionName eN ) {
    // error handling code
} finally {
    // statements to be executed
}
```

Exception Classes in C#

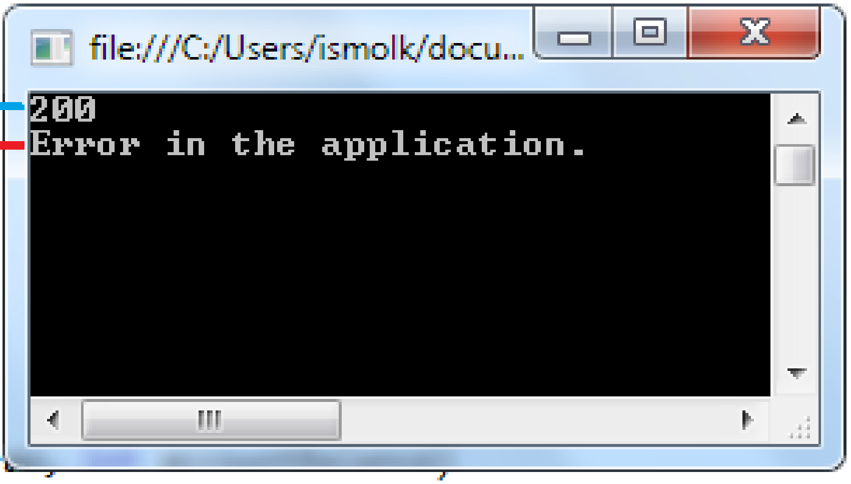
- C# exceptions are represented by classes.
- The exception classes in C# are mainly directly or indirectly derived from the **System.Exception** class.
  - Some of the exception classes derived from the System.Exception class are the
    - **System.ApplicationException** and
    - **System.SystemException** classes.

Predefined exception classes:

Exception Class	Description
System.IO.IOException	Handles I/O errors.
System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.NullReferenceException	Handles errors generated from deferencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.
System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

Withdraw Money

```
try
{
    WithdrawMoney(moneyToWithdraw, accountBalance);
}
catch (InsufficientFundsException e)
{
    Console.WriteLine(e.errorCode);
    Console.WriteLine(e.Message);
}

Console.ReadKey();
```

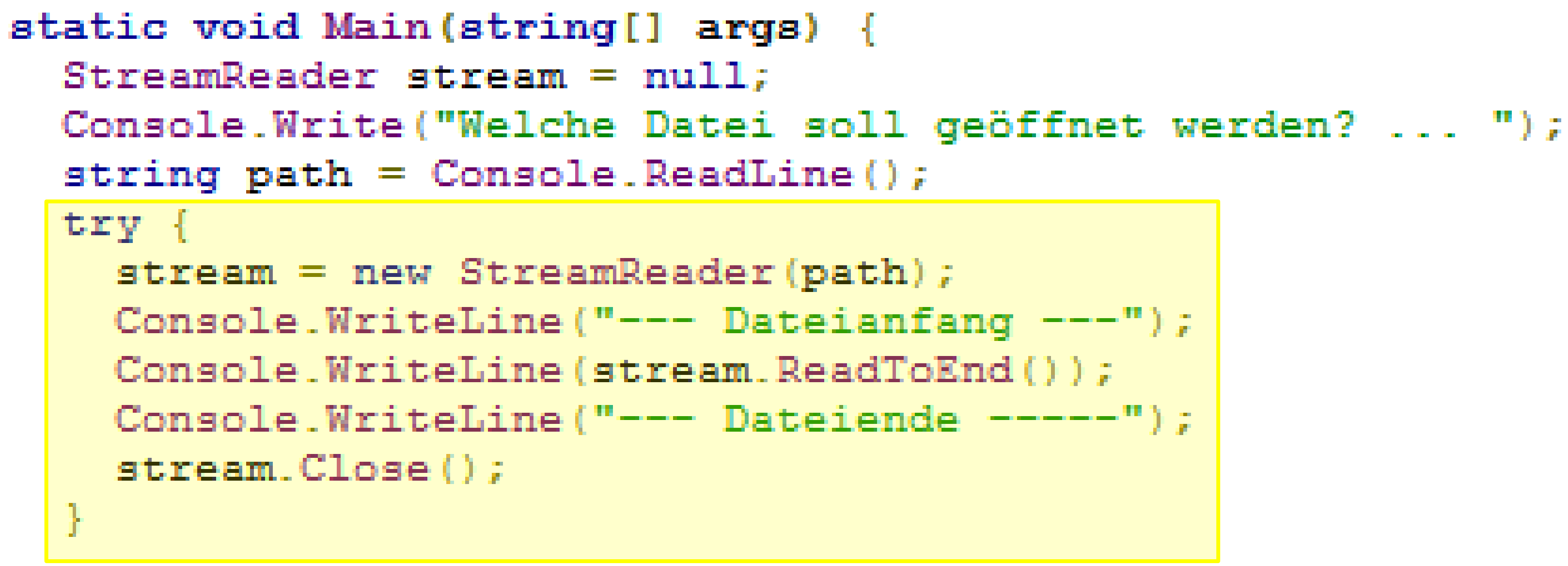


```
200
Error in the application.
```

```
itic void WithdrawMoney(int moneyToWithdr

    if (moneyToWithdraw > accountBalance)
        throw new InsufficientFundsException(200);
    else
        accountBalance = accountBalance - moneyToWithdraw;
```

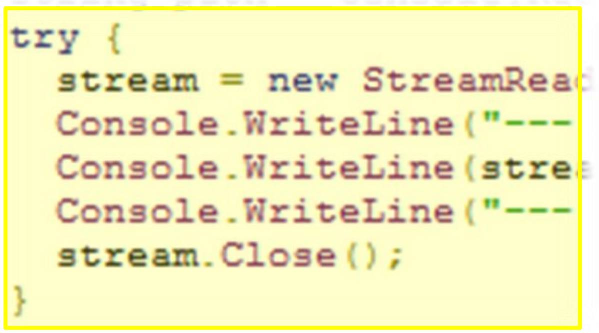
Exception Handling for Files:

- Open a File
- Read to end
- Close File

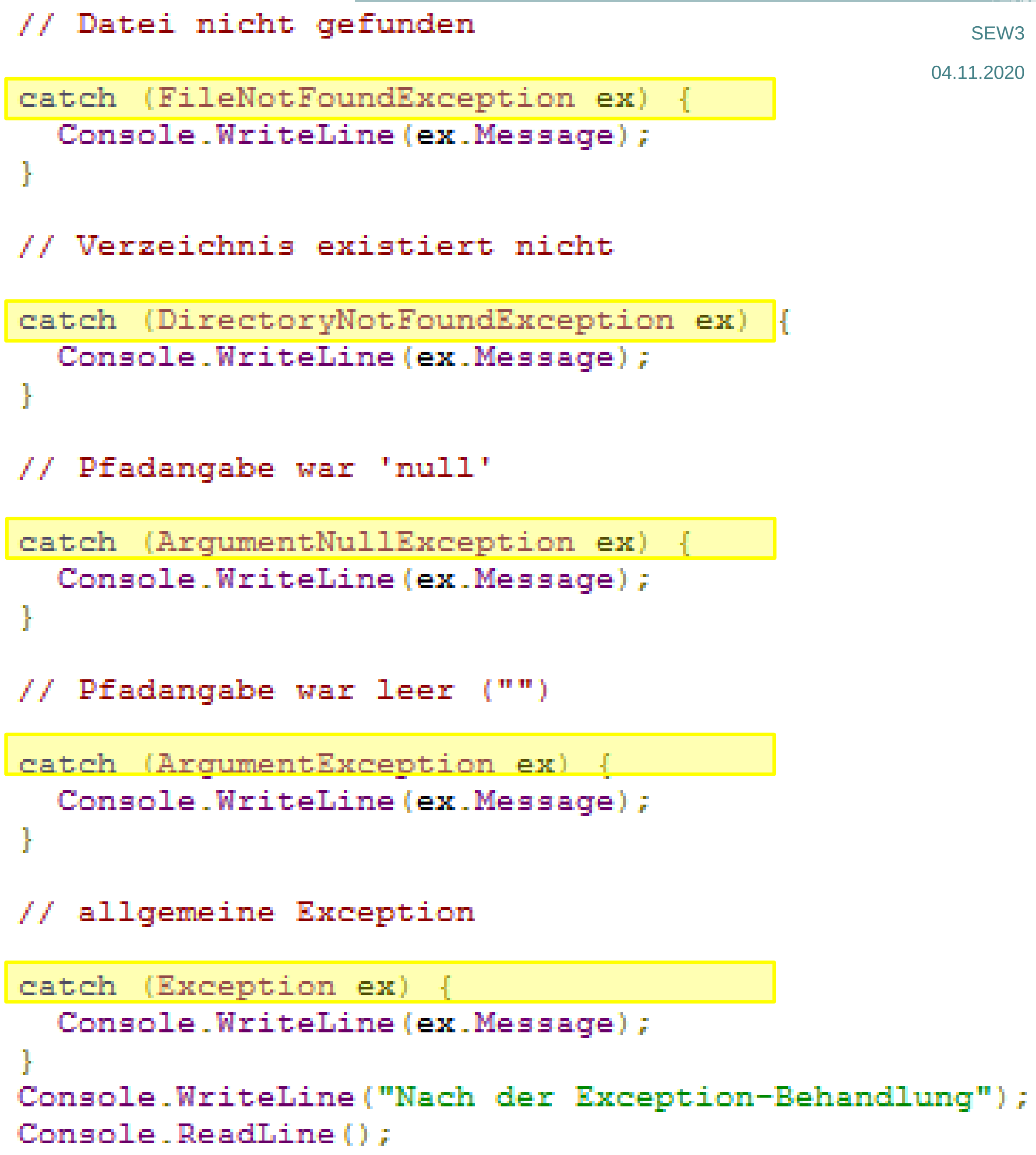


```
static void Main(string[] args) {
  StreamReader stream = null;
  Console.Write("Welche Datei soll geöffnet werden? ... ");
  string path = Console.ReadLine();
  try {
    stream = new StreamReader(path);
    Console.WriteLine("--- Dateianfang ---");
    Console.WriteLine(stream.ReadToEnd());
    Console.WriteLine("--- Dateiende -----");
    stream.Close();
  }
```

Catch different exceptions



```
try {
  stream = new StreamRea
  Console.WriteLine("---
  Console.WriteLine(stre
  Console.WriteLine("---
  stream.Close();
}
```



```
// Datei nicht gefunden

catch (FileNotFoundException ex) {
  Console.WriteLine(ex.Message);
}

// Verzeichnis existiert nicht

catch (DirectoryNotFoundException ex) {
  Console.WriteLine(ex.Message);
}

// Pfadangabe war 'null'

catch (ArgumentNullException ex) {
  Console.WriteLine(ex.Message);
}

// Pfadangabe war leer ("")

catch (ArgumentException ex) {
  Console.WriteLine(ex.Message);
}

// allgemeine Exception

catch (Exception ex) {
  Console.WriteLine(ex.Message);
}
Console.WriteLine("Nach der Exception-Behandlung");
Console.ReadLine();
```

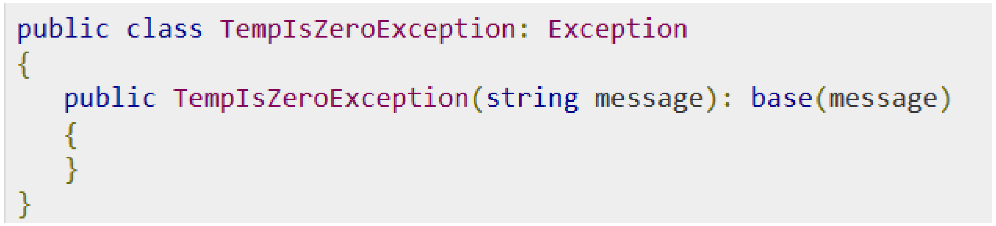
Write your own Exceptions:

```
class CustomException : Exception
{
    public CustomException(string message)
    {

    }

}
private static void TestThrow()
{
    CustomException ex =
        new CustomException("Custom exception in TestThrow()");

    throw ex;
}
```

Catch the CustomException

```
static void TestCatch()
{
    try
    {
        TestThrow();
    }
    catch (CustomException ex)
    {
        System.Console.WriteLine(ex.ToString());
    }
}
```

Write a TemperatureIsZeroExceptions



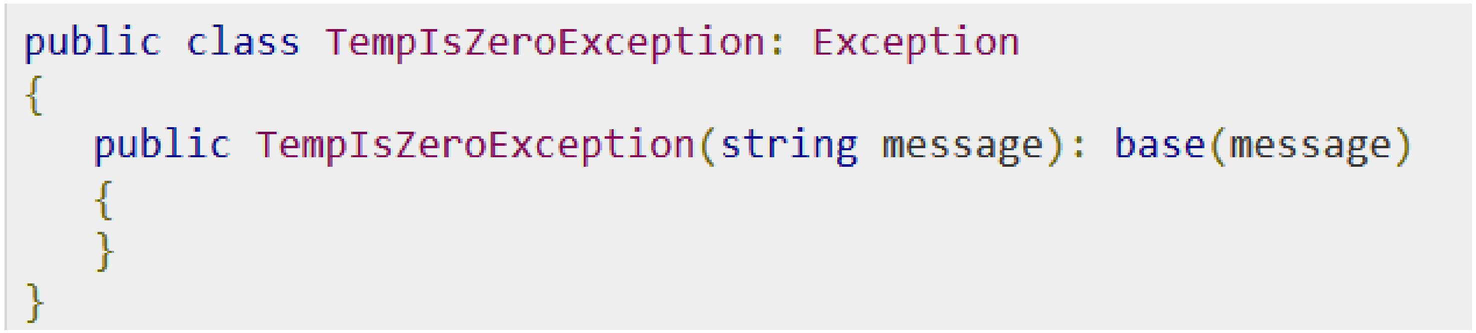
```
public class TempIsZeroException: Exception
{
    public TempIsZeroException(string message): base(message)
    {
    }
}
```

Temperatur Example

Create a class Temperature with an attribute temperatur.

Use your Temperatur-Class, set the value to zero

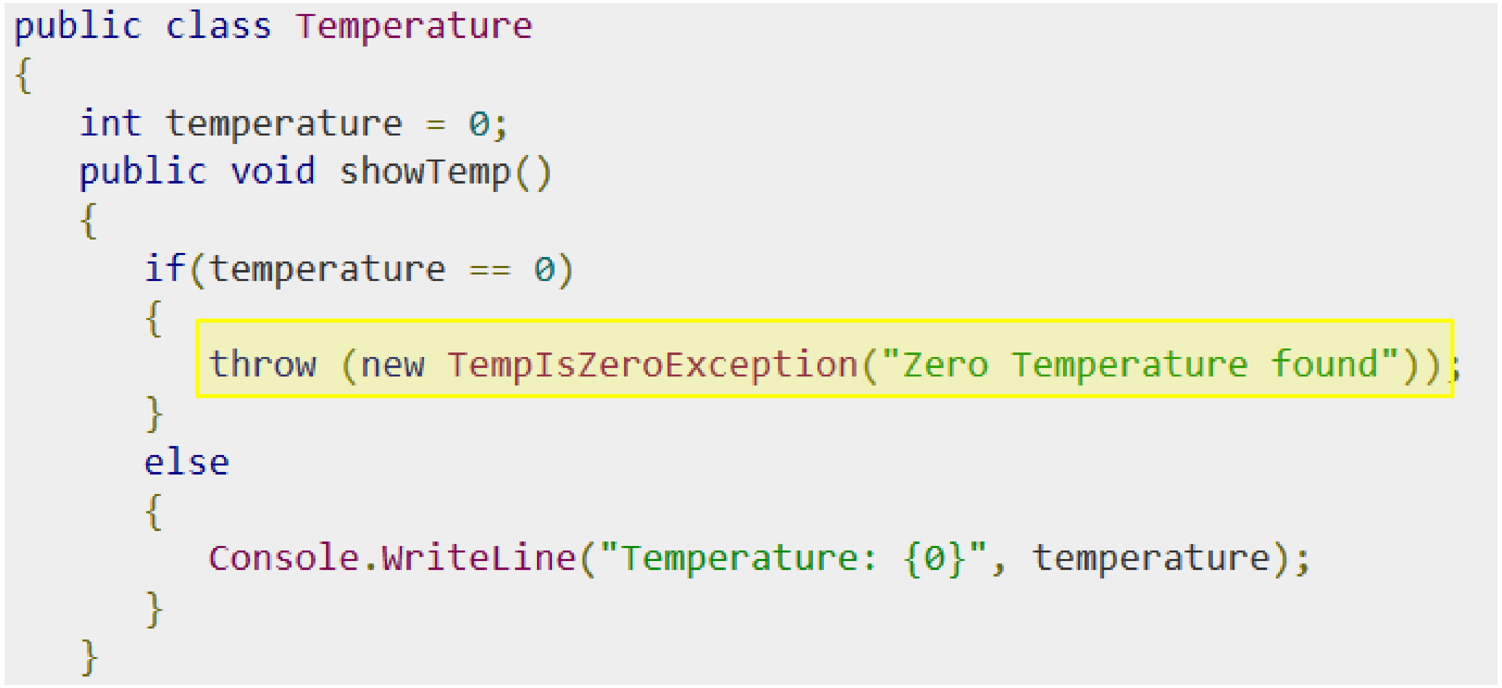
TempIsZeroException



```
public class TempIsZeroException: Exception
{
    public TempIsZeroException(string message): base(message)
    {
    }
}
```

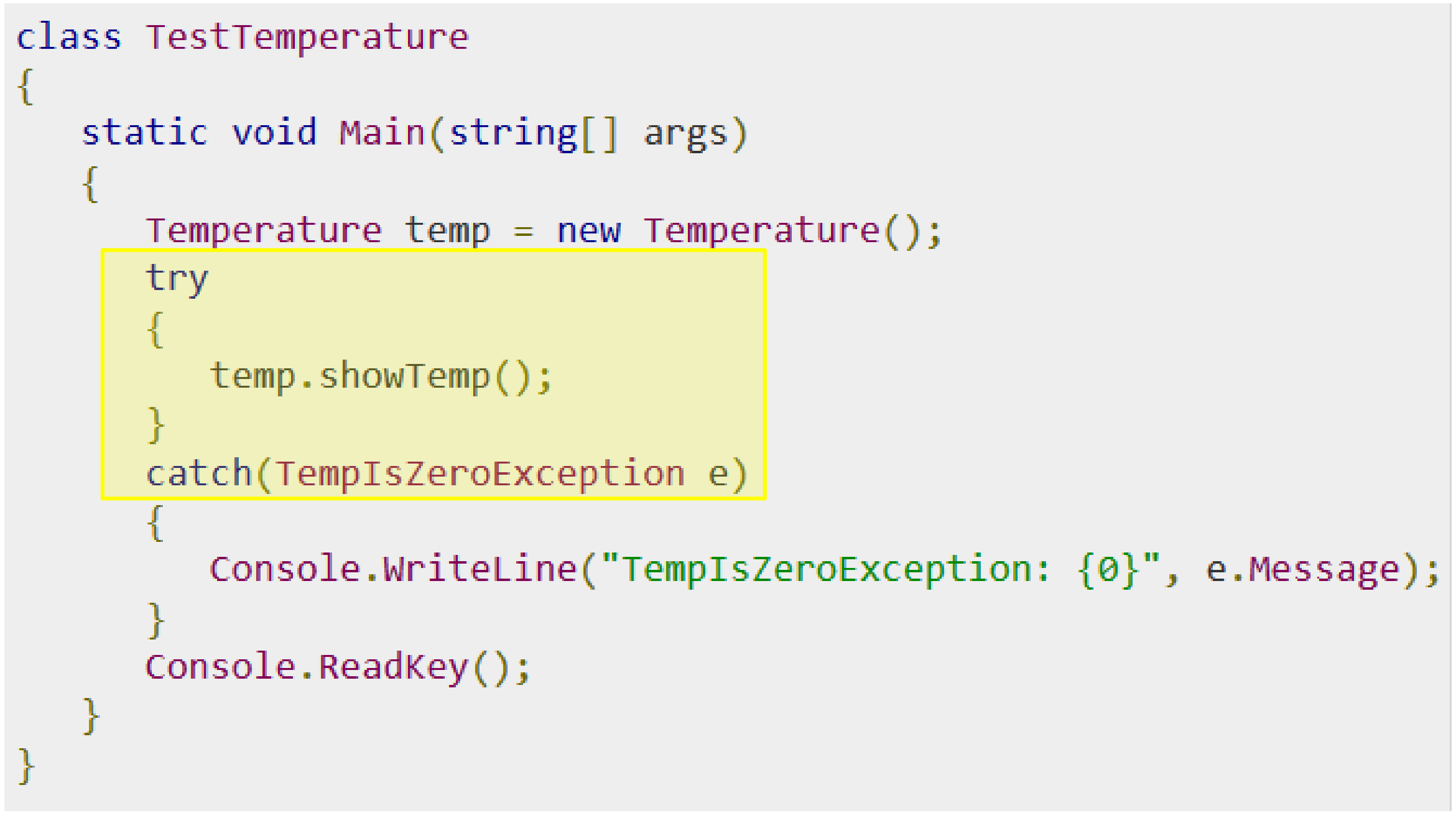
- Create a class Temperature
  - with an attribute temperatur.
  - If the Temperatur is zero, throw the exception
- Use your Temperatur-Class:
  - set the value to zero
  - Catch the exception and
    print the error message to the console

Throw User Defined Exceptions



```
public class Temperature
{
   int temperature = 0;
   public void showTemp()
   {
      if(temperature == 0)
      {
         throw (new TempIsZeroException("Zero Temperature found"));
      }
      else
      {
         Console.WriteLine("Temperature: {0}", temperature);
      }
   }
```

Catch User Defined Exceptions



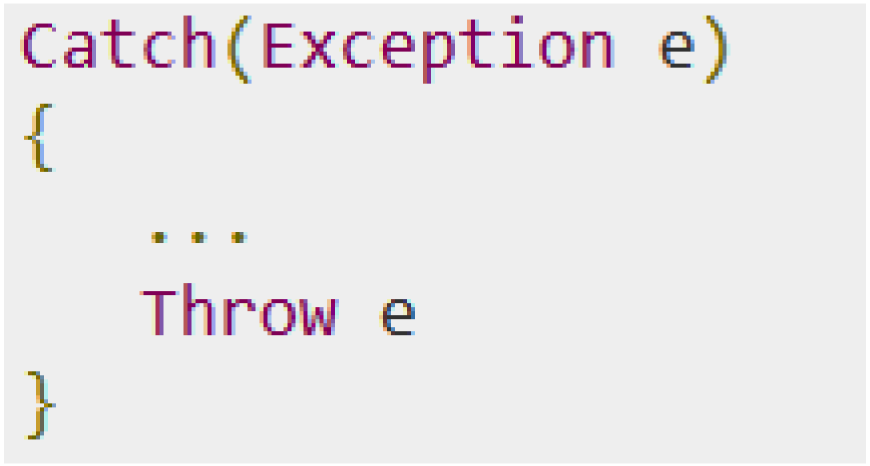
```
class TestTemperature
{
    static void Main(string[] args)
    {
        Temperature temp = new Temperature();
        try
        {
            temp.showTemp();
        }
        catch(TempIsZeroException e)
        {
            Console.WriteLine("TempIsZeroException: {0}", e.Message);
        }
        Console.ReadKey();
    }
}
```

Console Output:
TempIsZeroException: Zero Temperature found

Throwing Objects

- You can throw an object if it is either directly or indirectly derived from the **System.Exception** class.

- You can use a throw statement in the catch block to throw the present object as:



```
Catch(Exception e)
{
    ...
    Throw e
}
```

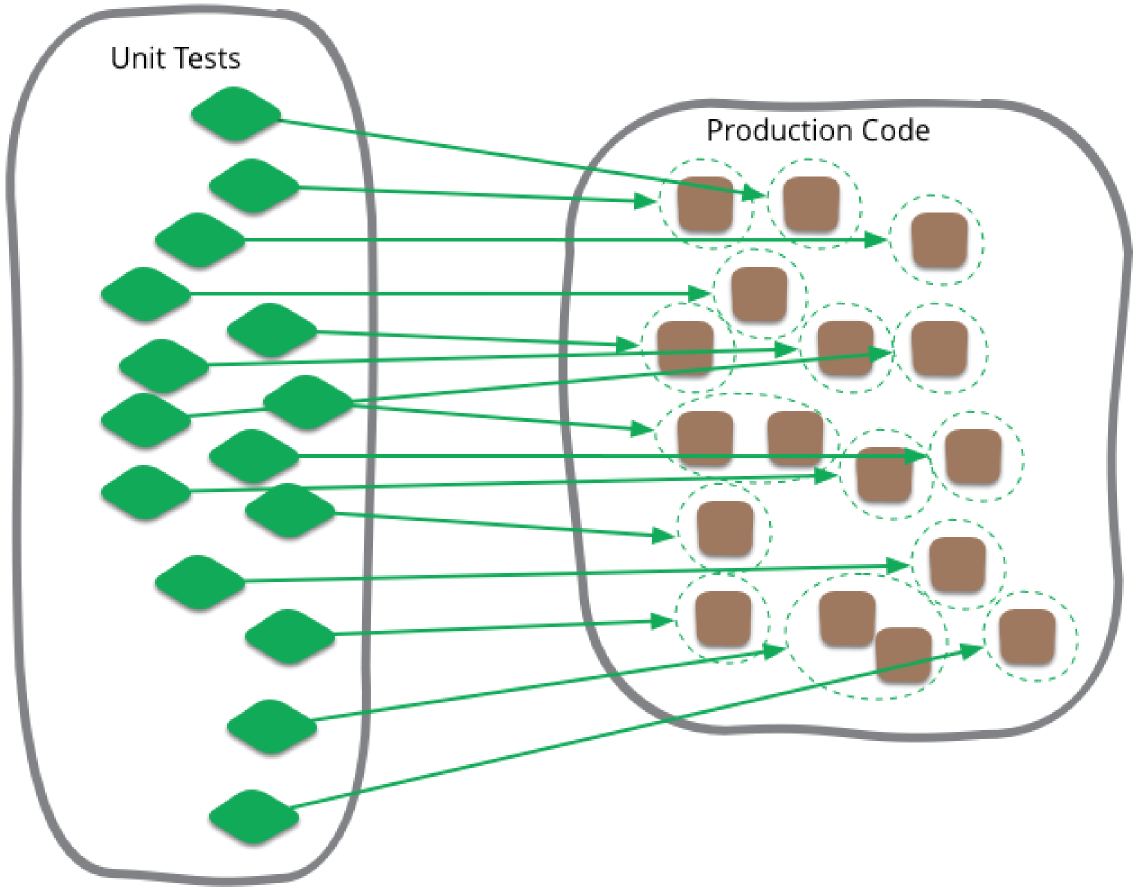
Example

- Write a class Person with Name and Age.
- Throw an Exception if the age is a negative Number.
- Main: Create a person with an invalide Age.

- Write your own Exception InvalideAgeException
- Throw your own Exception in Person
- Main: Write a Try – Catch Block
  - Read the Age again and set a correct Age.

Unit Tests

The gol of unit testing is to isolate each part of the program and show that the individual parts are correct



Automated Testing

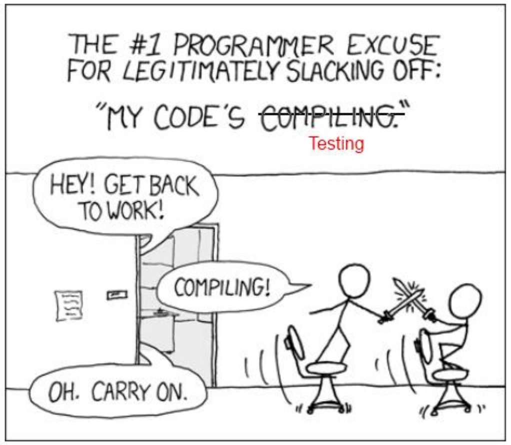
functional correctness

testing individual modules

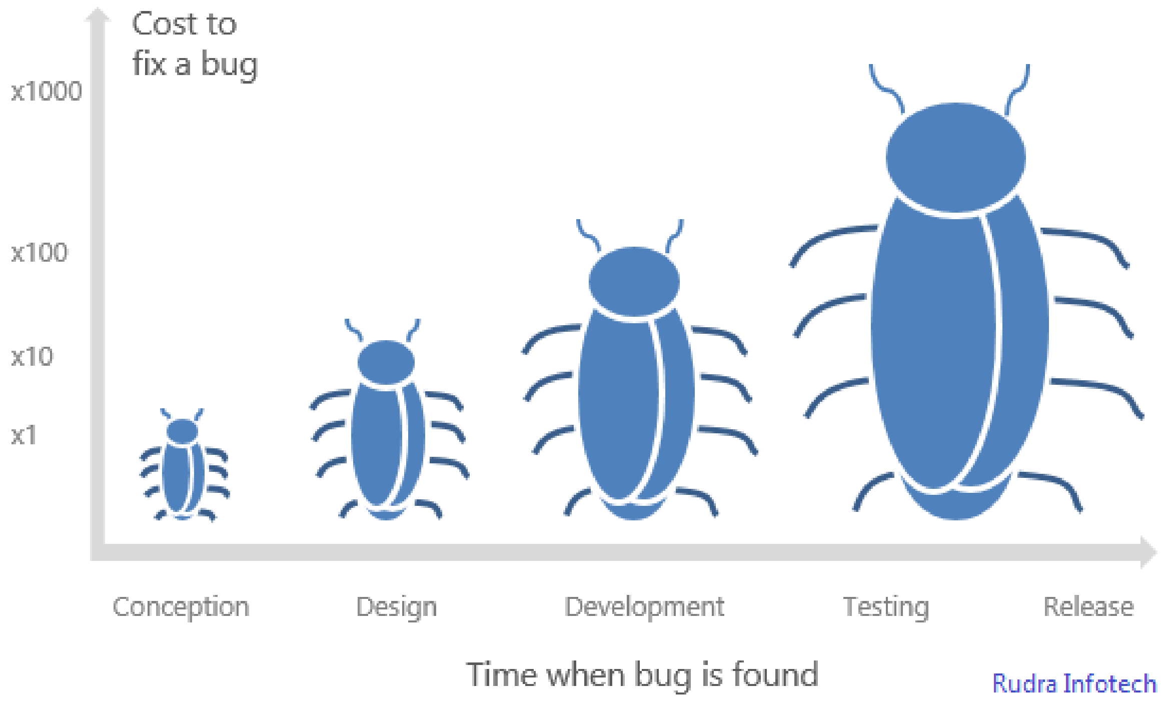
- **isolate each unit** of the system
  to **identify**, **analyze** and **fix** the **defects**

Unit Testing Techniques:

- **Black Box Testing**
  - The user input and output are tested, using the user interface.
- **White Box Testing**
  - Used to test each one of those functions behaviour.

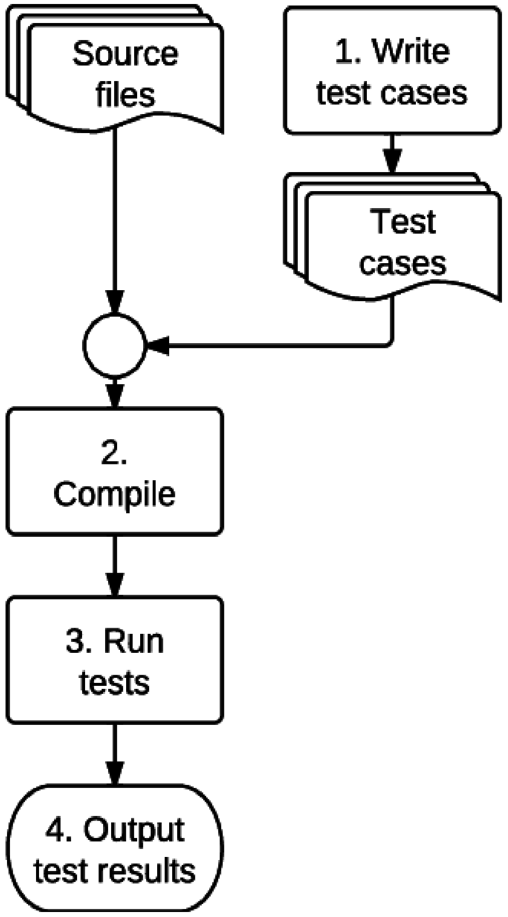

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S ~~COMPILING~~"
Testing
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

Time when Bug is found



For instance, a bug identified during conception costs something around zero, but when this same bug is found only after implementation or test, the average cost of repair can get to something between 10 and 1000 times more than in the previous step. When customers find this bug in production environment, the cost of the problem considers all side effects related to it and. That is where things can get serious.

Unit Testing Lifecycle



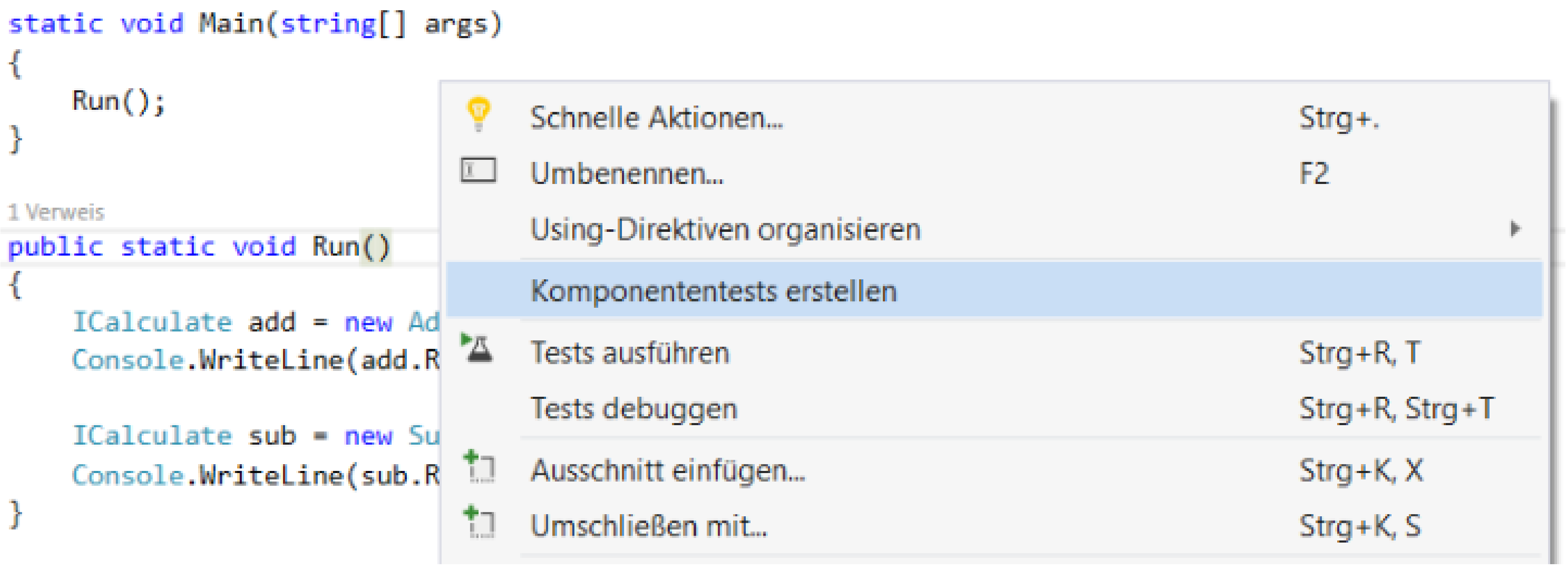
- Create Sourcecode
- Write Test Cases

- Compile
- Run Tests

- Validate Test Results
- Correct Code if necessary

Create a Unit Tests in C#

- Right Mouse Click on a Method
- „Komponententests erstellen“

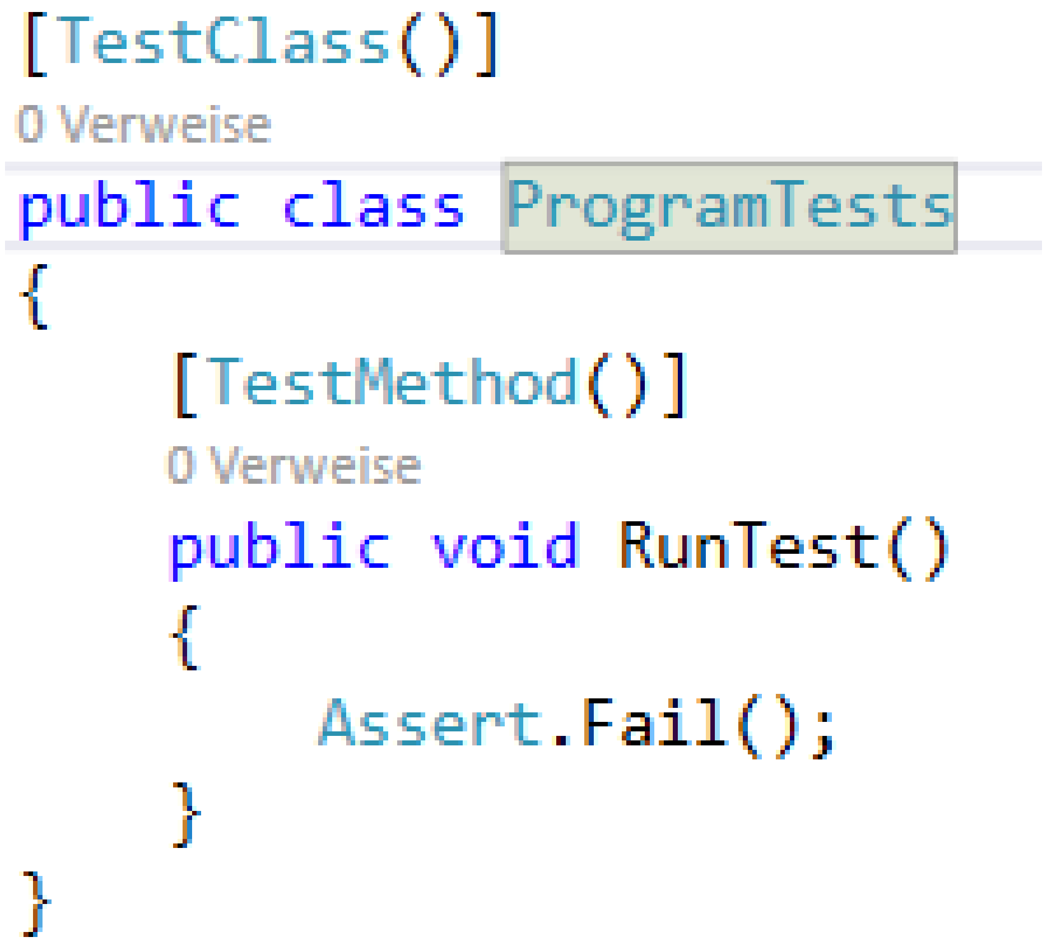


```
static void Main(string[] args)
{
    Run();
}

1 Verweis
public static void Run()
{
    ICalculate add = new Ad
    Console.WriteLine(add.R

    ICalculate sub = new Su
    Console.WriteLine(sub.R
}
```

Schnelle Aktionen... Strg+.
Umbenennen... F2
Using-Direktiven organisieren
Komponententests erstellen
Tests ausführen Strg+R, T
Tests debuggen Strg+R, Strg+T
Ausschnitt einfügen... Strg+K, X
Umschließen mit... Strg+K, S

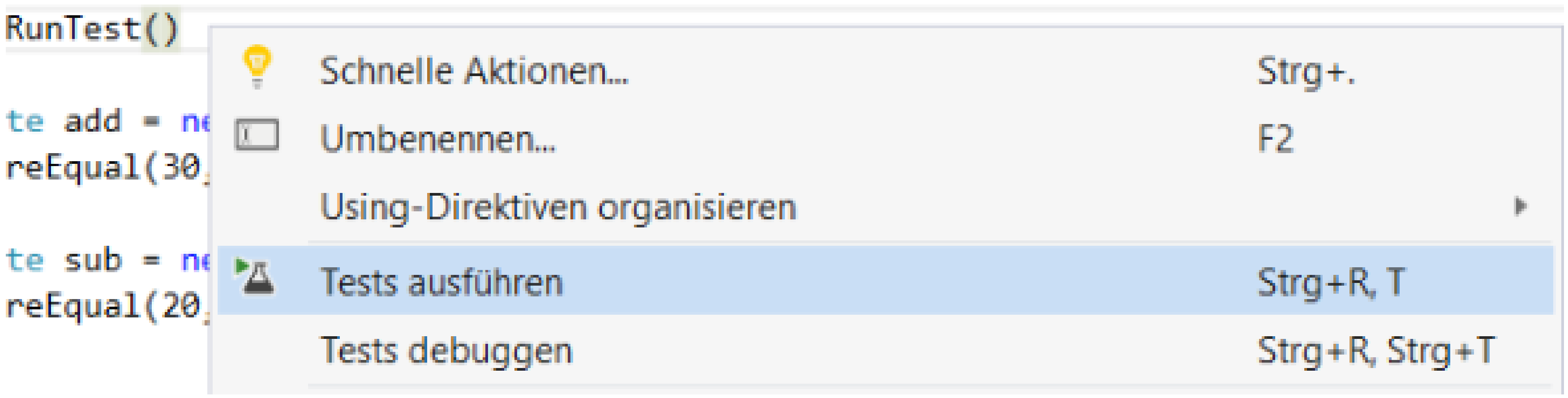
- Test Class with Test Method will be created

Test Class with Test Methods



```
[TestClass()]
0 Verweise
public class ProgramTests
{
    [TestMethod()]
    0 Verweise
    public void RunTest()
    {
        Assert.Fail();
    }
}
```

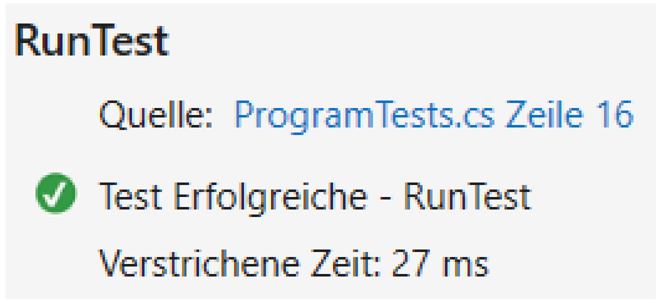
Run Unit Tests



```
RunTest()

te add = ne
reEqual(30,

te sub = ne
reEqual(20,
```

Schnelle Aktionen... Strg+.
Umbenennen... F2
Using-Direktiven organisieren
Tests ausführen Strg+R, T
Tests debuggen Strg+R, Strg+T

- Right Mouse Click on a Test Method
- Run Test



RunTest

Quelle: ProgramTests.cs Zeile 16

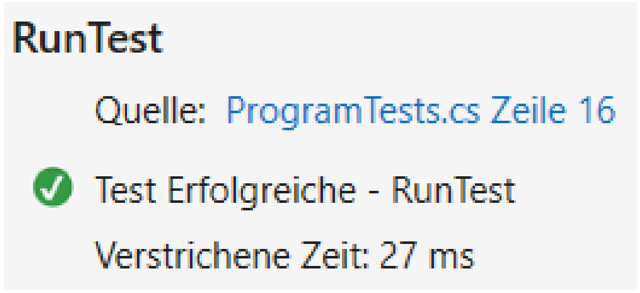
✓ Test Erfolgreiche - RunTest

Verstrichene Zeit: 27 ms

Class Assert

- unit test method
  - exercises the code of a method
  - but it **reports the correctness** of the code's behavior **only if you include Assert statements**

- Assert class contains
  - a set of static methods
    that evaluate a Boolean condition
  - the assertion passes,
    when the evaluation is true



RunTest

Quelle: ProgramTests.cs Zeile 16

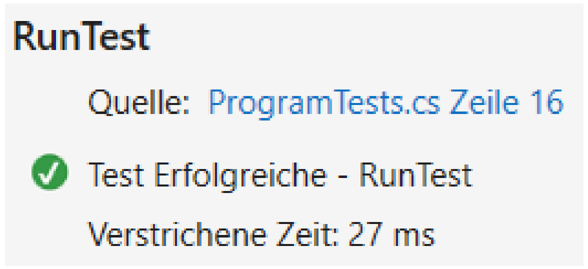
✓ Test Erfolgreiche - RunTest

Verstrichene Zeit: 27 ms

Assert Methods:

• `AreEqual()`	`AreNotEqual()`
• `IsNull()`	`IsNotNull()`
• `IsInstanceOfType()`	`IsNotInstanceOfType()`
• `IsTrue()`	`IsFalse()`

Test Successful

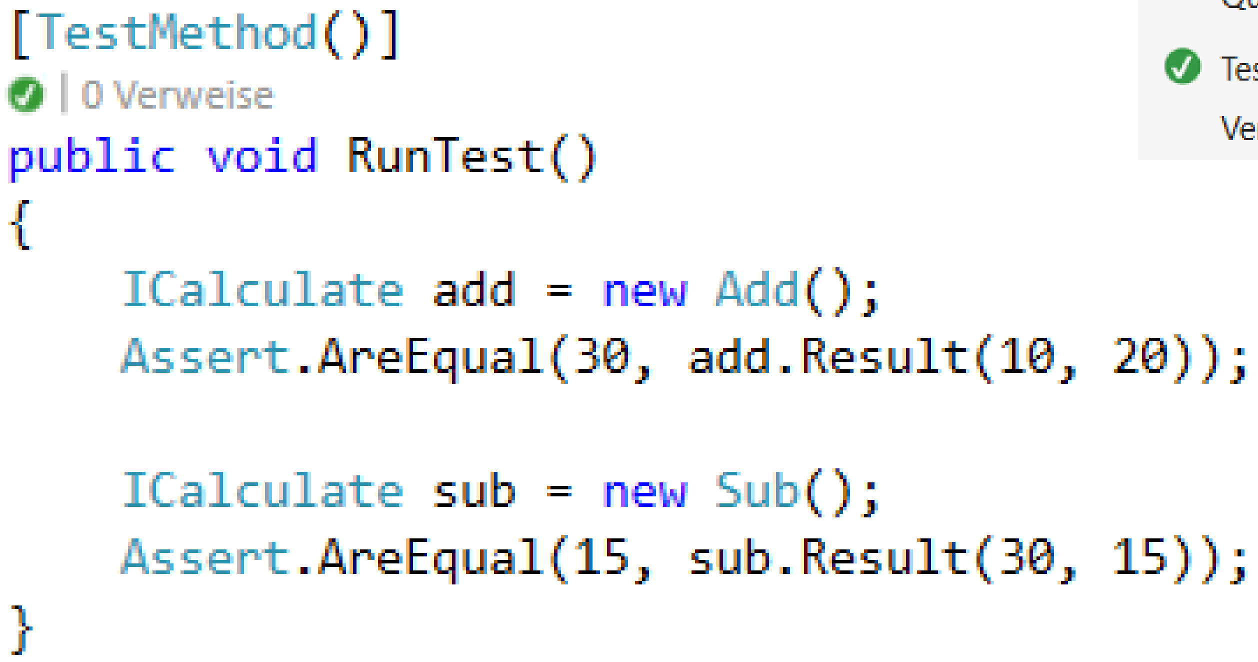


RunTest

Quelle: ProgramTests.cs Zeile 16

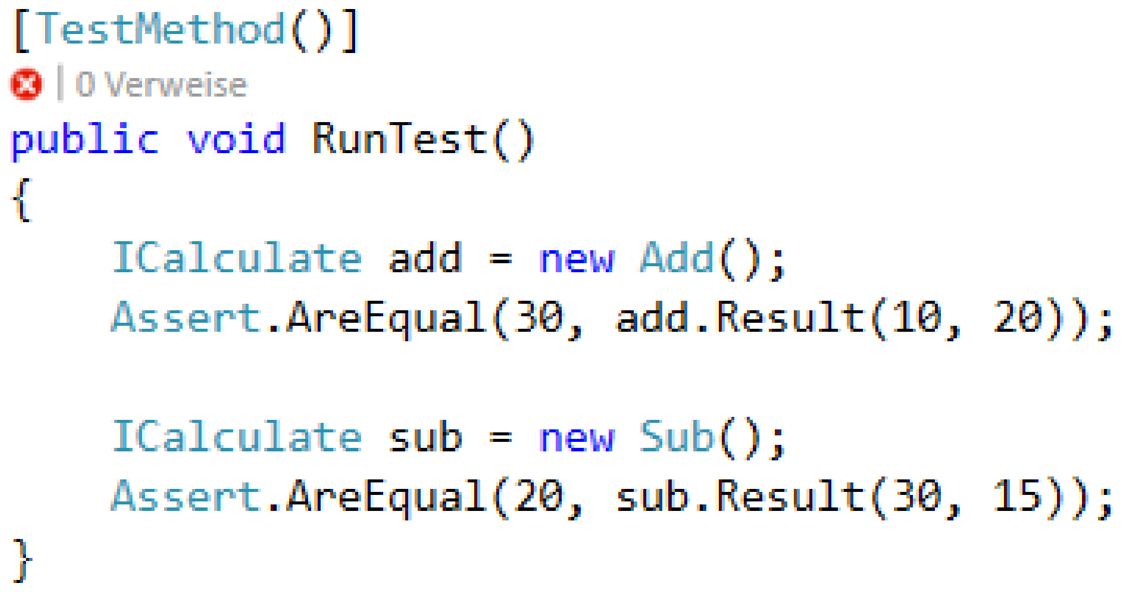
✓ Test Erfolgreiche - RunTest

Verstrichene Zeit: 27 ms

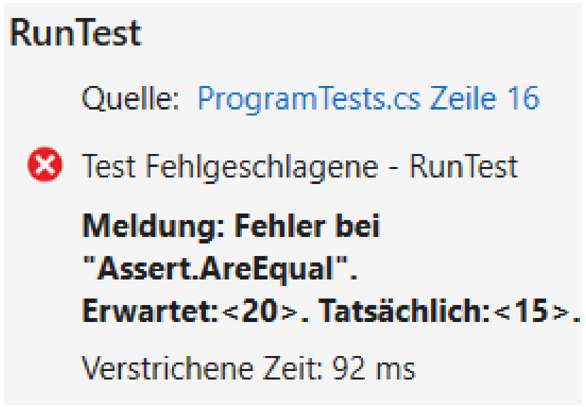


```
[TestMethod()]
✓ | 0 Verweise
public void RunTest()
{
    ICalculate add = new Add();
    Assert.AreEqual(30, add.Result(10, 20));

    ICalculate sub = new Sub();
    Assert.AreEqual(15, sub.Result(30, 15));
}
```

Test Failed



```
[TestMethod()]
ⓧ | 0 Verweise
public void RunTest()
{
    ICalculate add = new Add();
    Assert.AreEqual(30, add.Result(10, 20));

    ICalculate sub = new Sub();
    Assert.AreEqual(20, sub.Result(30, 15));
}
```



RunTest

Quelle: ProgramTests.cs Zeile 16

ⓧ Test Fehlgeschlagene - RunTest

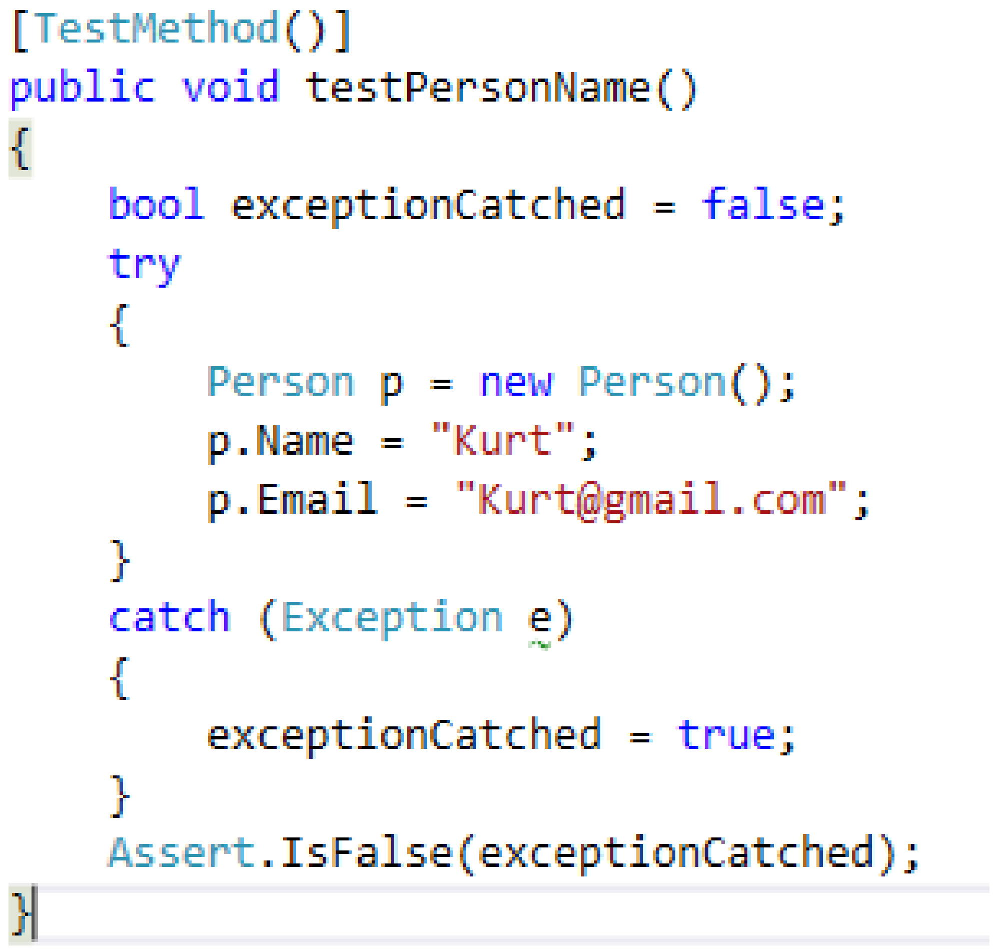
Meldung: Fehler bei "Assert.AreEqual". Erwartet:<20>. Tatsächlich:<15>.

Verstrichene Zeit: 92 ms

Expected Exception

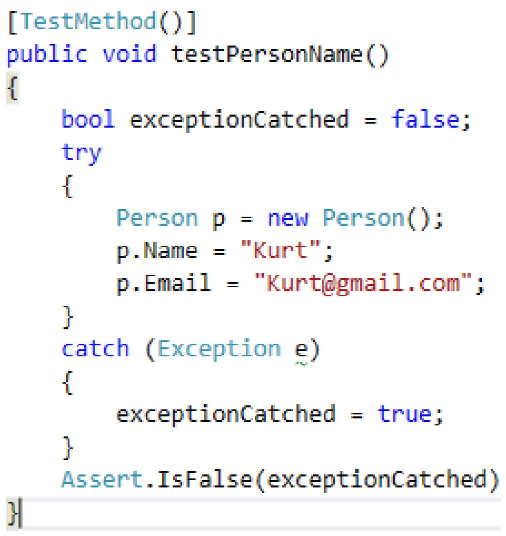
```
[TestClass()]
public class DivisionClassTest
{
    [TestMethod()]
    [ExpectedException(typeof(System.DivideByZeroException))]
    public void DivideTest()
    {
        DivisionClass target = new DivisionClass();
        int numerator = 4;
        int denominator = 0;
        int actual;
        actual = target.Divide(numerator, denominator);
    }
}
```

Excercise: Simplify the Test Method:

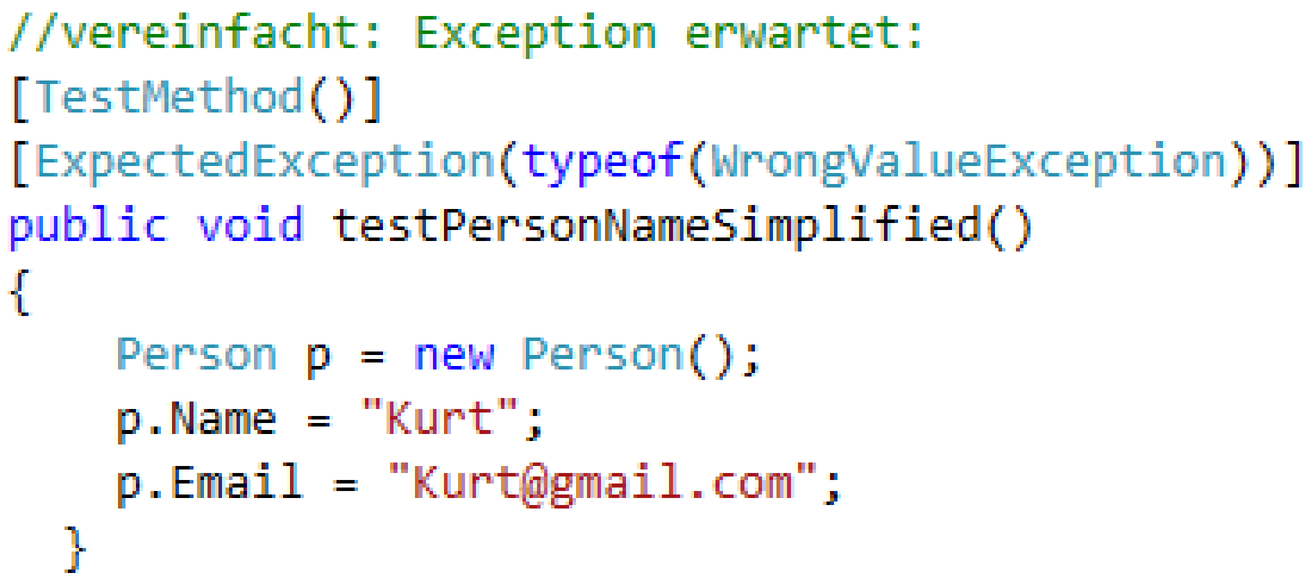


```
[TestMethod()]
public void testPersonName()
{
    bool exceptionCatched = false;
    try
    {
        Person p = new Person();
        p.Name = "Kurt";
        p.Email = "Kurt@gmail.com";
    }
    catch (Exception e)
    {
        exceptionCatched = true;
    }
    Assert.IsFalse(exceptionCatched);
}
```

Exception expected



```
[TestMethod()]
public void testPersonName()
{
    bool exceptionCatched = false;
    try
    {
        Person p = new Person();
        p.Name = "Kurt";
        p.Email = "Kurt@gmail.com";
    }
    catch (Exception e)
    {
        exceptionCatched = true;
    }
    Assert.IsFalse(exceptionCatched)
}
```



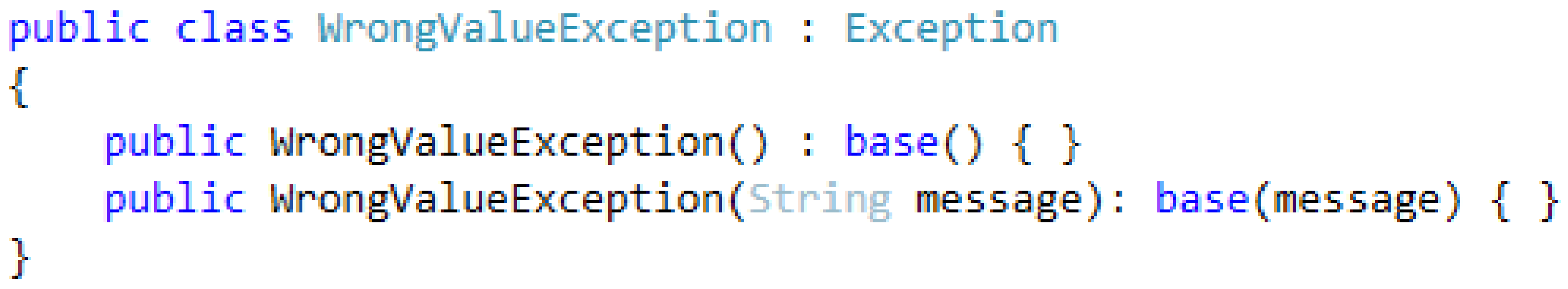
```
//vereinfacht: Exception erwartet:
[TestMethod()]
[ExpectedException(typeof(WrongValueException))]
public void testPersonNameSimplified()
{
    Person p = new Person();
    p.Name = "Kurt";
    p.Email = "Kurt@gmail.com";
  }
```

Excercise: Exceptions & Unit Test

- Write a Class Person
  - with a Name (min 3 chars)
  - with an Email Adress (contains an @)
  - with an Adress (zip code (max 4 digits) , street & city)
  - with a Telefonnumber (starts with 0043 in Austria)

  - Throw an Exception, if the input values are not vaild
  - Write your individual own exceptions
  - Test the Program with an Unit Test
  - Split the Test Cases in serveral TestMethods

WrongValueException

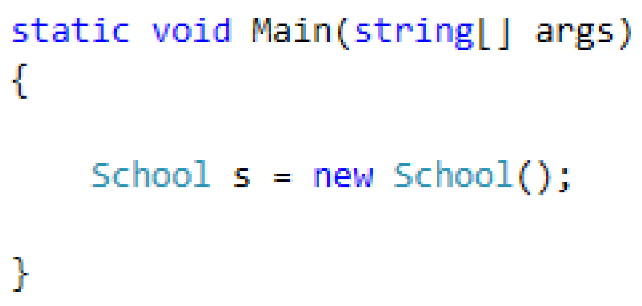


```
public class WrongValueException : Exception
{
    public WrongValueException() : base() { }
    public WrongValueException(String message): base(message) { }
}
```

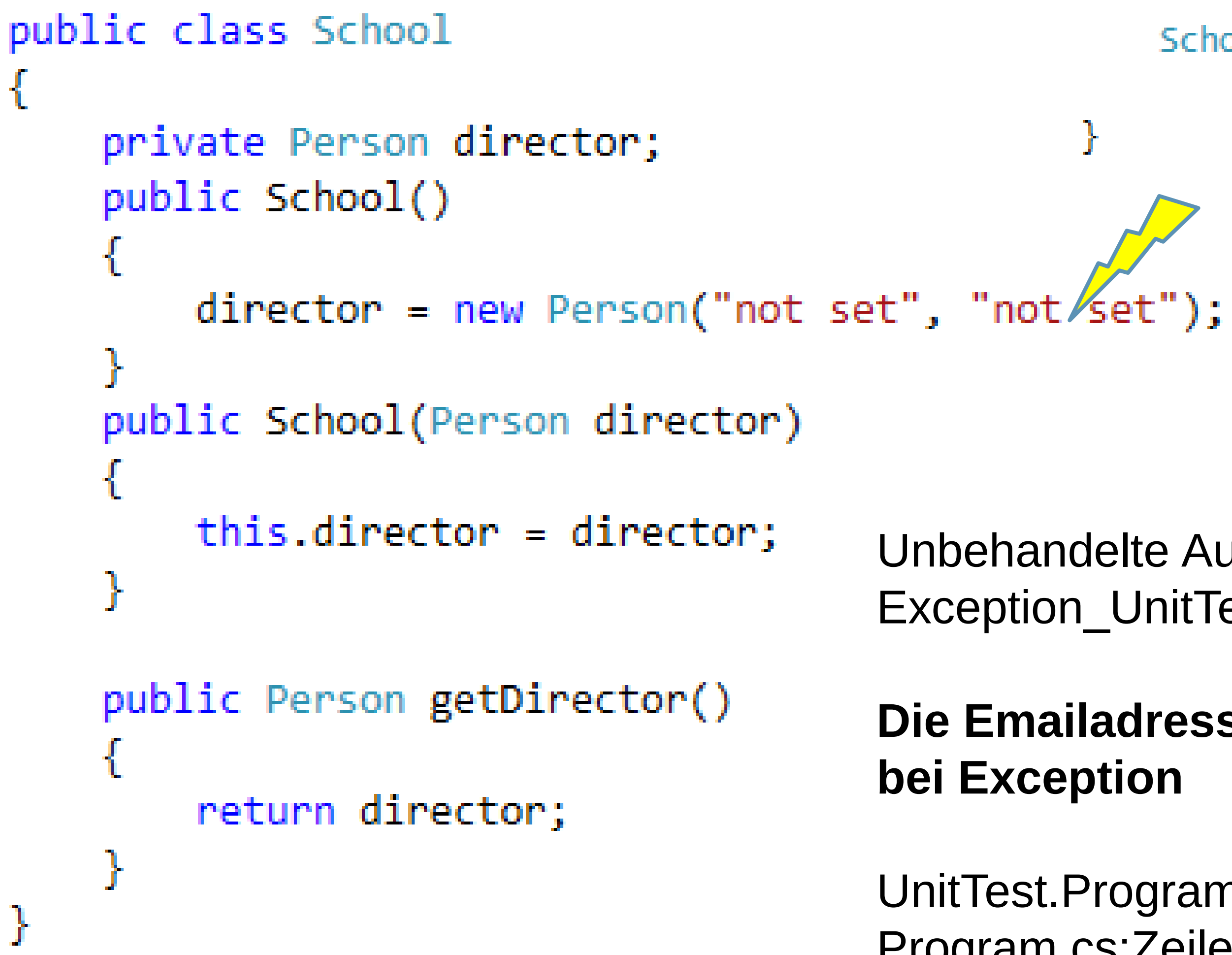
Class Person

```
public class Person
{
    public Person() { }
    public Person(string name, string email)
    {
        this.Name = name;
        this.Email = email;
    }

    private string name;
    public string Name {
        get { return name; }
        set {
            if (value.Length < 3)
                throw new WrongValueException("Der Name muss mindestens aus 3 Zeichen bestehen");
            else name = value;
        }
    }
    private string email;
    public string Email {
        get { return email; }
        set {
            if (!value.Contains("@"))
                throw new WrongValueException("Die Emailadresse muss ein @ enthalten");
            else email = value;
        }
    }
}
```

Class School



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

    School s = new School();

}
```



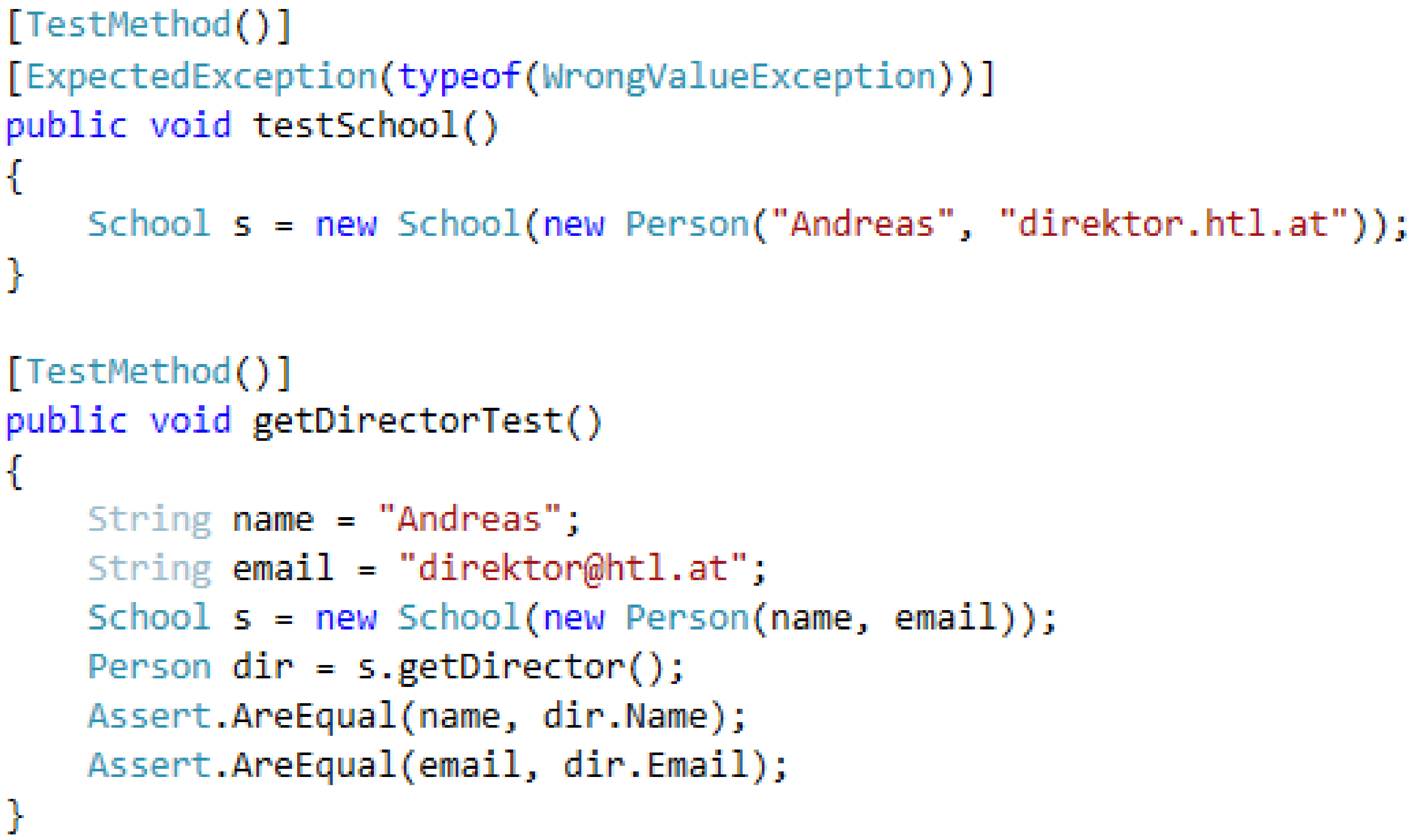
```
public class School
{
    private Person director;
    public School()
    {
        director = new Person("not set", "not set");
    }
    public School(Person director)
    {
        this.director = director;
    }

    public Person getDirector()
    {
        return director;
    }
}
```

Unbehandelte Ausnahme: Exception_UnitTest.WrongValueException:

Die Emailadresse muss ein @ enthalten bei Exception

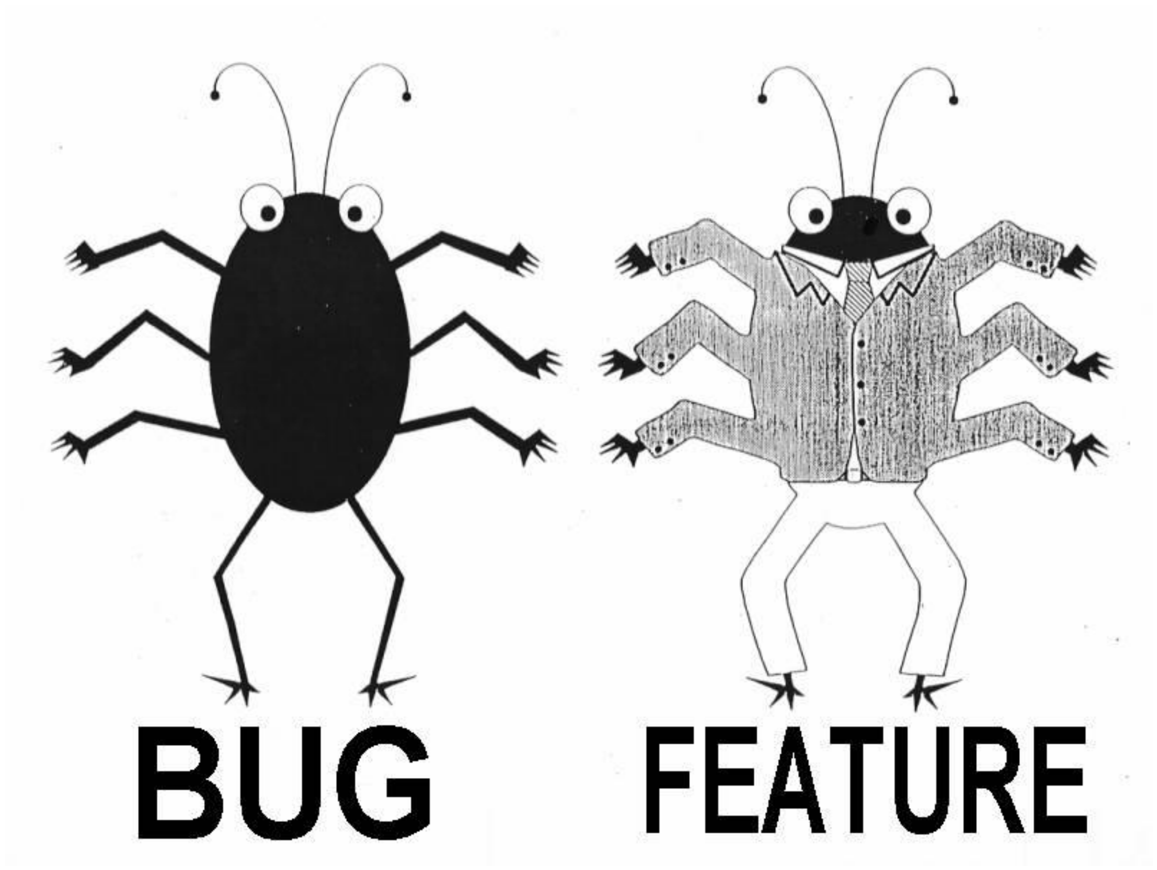
UnitTest.Program.Main(String[] args) in Program.cs:Zeile 79.

Unit Tests - Testing School & Director



```
[TestMethod()]
[ExpectedException(typeof(WrongValueException))]
public void testSchool()
{
    School s = new School(new Person("Andreas", "direktor.htl.at"));
}

[TestMethod()]
public void getDirectorTest()
{
    String name = "Andreas";
    String email = "direktor@htl.at";
    School s = new School(new Person(name, email));
    Person dir = s.getDirector();
    Assert.AreEqual(name, dir.Name);
    Assert.AreEqual(email, dir.Email);
}
```

It‘s not a bug, …


BUG
FEATURE