itelio

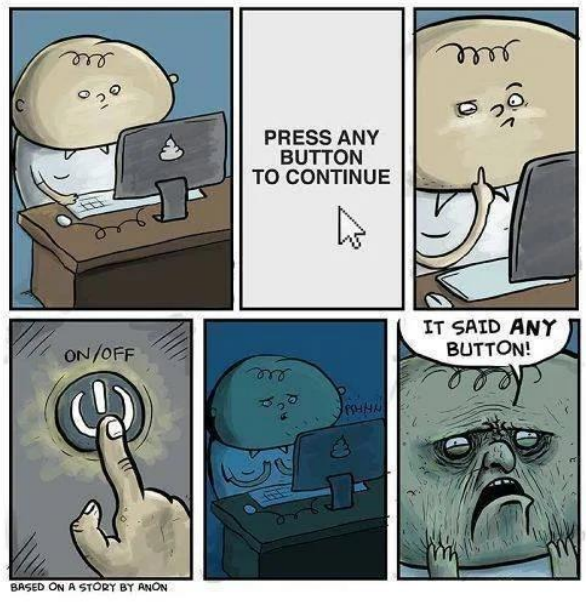
Fehlerbehandung
Debugging
Logging

Agenda

1. Fehlerbehandlung
    1. Warum treten Fehler auf?
    2. Auswirkungen
    3. Definition Exceptions
    4. Exception Handling
    5. Übung
2. Debugging
    1. Einführung Debugging – Visual Studio
    2. Breakpoints
    3. Call Stack
    4. Übung
3. Logging
    1. Warum / Was / Wohin
    2. Log4net

Fehlerbehandlung – Warum treten Fehler auf?

- Eingaben von Benutzer fehlerhaft
  - Benutzer tätigt unerwartete Aktionen
  - Kann etwas falsch eingegeben werden, so wird das früher oder später passieren.



Fehlerbehandlung – Warum treten Fehler auf?

- Fehler im Code
  - Statistische Fehlerdichte zwischen 0,5 – 10 Fehler pro 1000 Zeilen Code.
  - https://de.wikipedia.org/wiki/Fehlerquotient
  - Codeänderungen -> Neue Anforderungen
- Hardware
  - Neue Hardware verhält sich anders / wird nicht unterstützt
  - Festplatte voll
- …
- Murphys Law -> „Alles was schiefgehen kann wird auch schiefgehen“

Fehlerbehandlung - Auswirkungen

- Was passiert wenn Fehler nicht behandelt werden?
  - Absturz
    - Aktuelle Arbeit des Benutzers wird nicht gespeichert
    - Zeitverlust
    - Negativer Eindruck/Bewertung der Software
  - Daten werden verfälscht
    - Kann schlimmer als nicht vorhandene Daten sein.
    - Beispiel: Mars Climate Orbiter
    - https://de.wikipedia.org/wiki/Mars_Climate_Orbiter
  - Schadenersatzforderungen
  - Je nach Anwendungsgebiet eventuell menschlicher Schaden
    - Airback Steuerung
    - ...

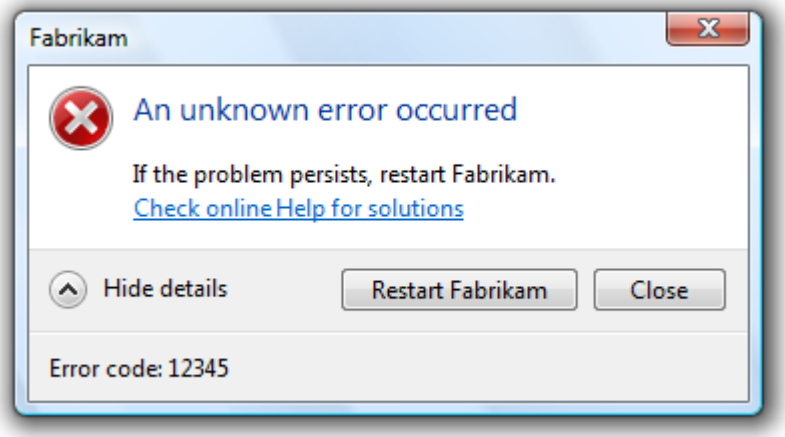
Fehlerbehandlung – Was tun?

- Fehler im Vorfeld verhindern
  - Logische Fehler / Codeänderungen…
    - Unit Tests
  - Benutzereingaben validieren

- Unerwartete Fehler
  - Fehlercodes auswerten und reagieren
  - **Exception Handling**
  - **Fehler beseitigen -> Debugging**

Fehlerbehandlung – Was kam vor Exceptions

- Error Codes
- Beispiel
  - C/C++
    - Aktives Abfragen ob ein Fehler passiert ist
    - GetLastError()
  - https://docs.microsoft.com/en-us/windows/desktop/debug/system-error-codes--0-499-

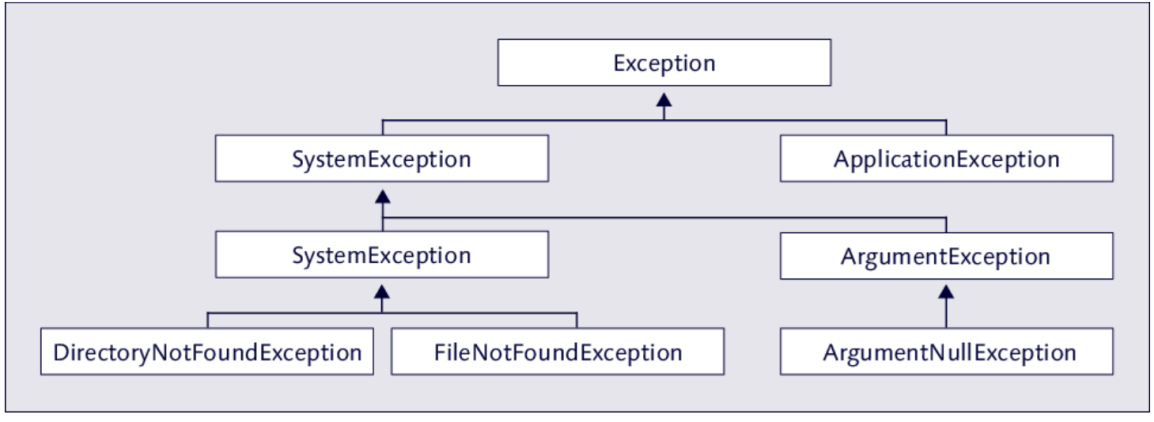


Ausnahmen/Exceptions

- Definition
  - „Represents errors that occur during application execution."
  - https://docs.microsoft.com/de-de/dotnet/api/system.exception

- Was ist in einer Exception enthalten?
  - Message – Lesbare Fehlermeldung, oft übersetzt in die lokale Sprache.
  - Stacktrace (Wie sind wir zu dieser Stelle der Codeausführung gekommen)
  - Zusätzliche Fehlerinformationen / Daten
  - Informationen über innere Exceptions die aufgetreten sind.

- Exception Hierarchie



http://openbook.rheinwerk-verlag.de/visual_csharp_2012/1997_07_001.html

Ausnahmen/Exceptions – Try und Catch Block

itelio

```
public string LoadFile(string myFile)
{
    try
    {
        var text = File.ReadAllText(myFile);
        return text;
    }
    catch(Exception ex)
    {
         Console.WriteLine("Exception: " + ex.ToString());
    }
    return string.Empty;
}
```

- Wird eine Exception innerhalb des try Blocks ausgelöst, so springt die Codeausführung direkt in den catch Block.

Ausnahmen/Exceptions – Try und Catch Block

- Spezielle Exception abfangen

```
public string LoadFile(string myFile)
{
    try
    {
        return File.ReadAllText(myFile);
    }
    catch(FileNotFoundException ex)
    {
        Console.WriteLine("Das ist eine IO Exeption: " + ex.ToString());
    }
    return string.Empty;
}
```

- Nur die FileNotFoundException wird abgefangen, andere Exceptions werfen die Exception eine Ebene höher.

- Mehere Exception Typen behandeln

```
public string LoadFile(string myFile)
{
    try
    {
        if (string.IsNullOrWhiteSpace(myFile))
           throw new ArgumentException("Keine Datei angegeben.");
        return File.ReadAllText(myFile);
    }
    catch(FileNotFoundException ex)
    {
        Console.WriteLine("Das ist eine IO Exeption: " + ex.ToString());
    }
    catch(ArgumentException ex)
    {
        Console.WriteLine(ex.ToString());
    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception: " + ex.ToString());
    }
    return string.Empty;
}
```

- Benutzerdefinierte Exceptions implementieren

```
public class FileNameEmptyException : Exception
{
    public FileNameEmptyException()
        : base("Dateiname ist leer")
    {  }
}
```

```
…
public string LoadFile(string a_file)
{
    if (string.IsNullOrWhiteSpace(a_file))
        throw new FileNameEmptyException();

    return File.ReadAllText(a_file);
}
…
```

Ausnahmen/Exceptions – Finally

- Wird in jedem Fall durchlaufen, auch wenn im Try Block eine Exception passiert.
- Nützlich um Ressourcen aufzuräumen

```
public void UploadFile(string a_data)
{
    var tempFile = Path.GetTempFileName();
    try
    {
        File.WriteAllBytes(tempFile, Encoding.UTF8.GetBytes(a_data));
        //Upload
        //...
    }
    catch(Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
    finally
    {
        if (File.Exists(tempFile))
            File.Delete(tempFile);
    }
}
```

Ausnahmen/Exceptions

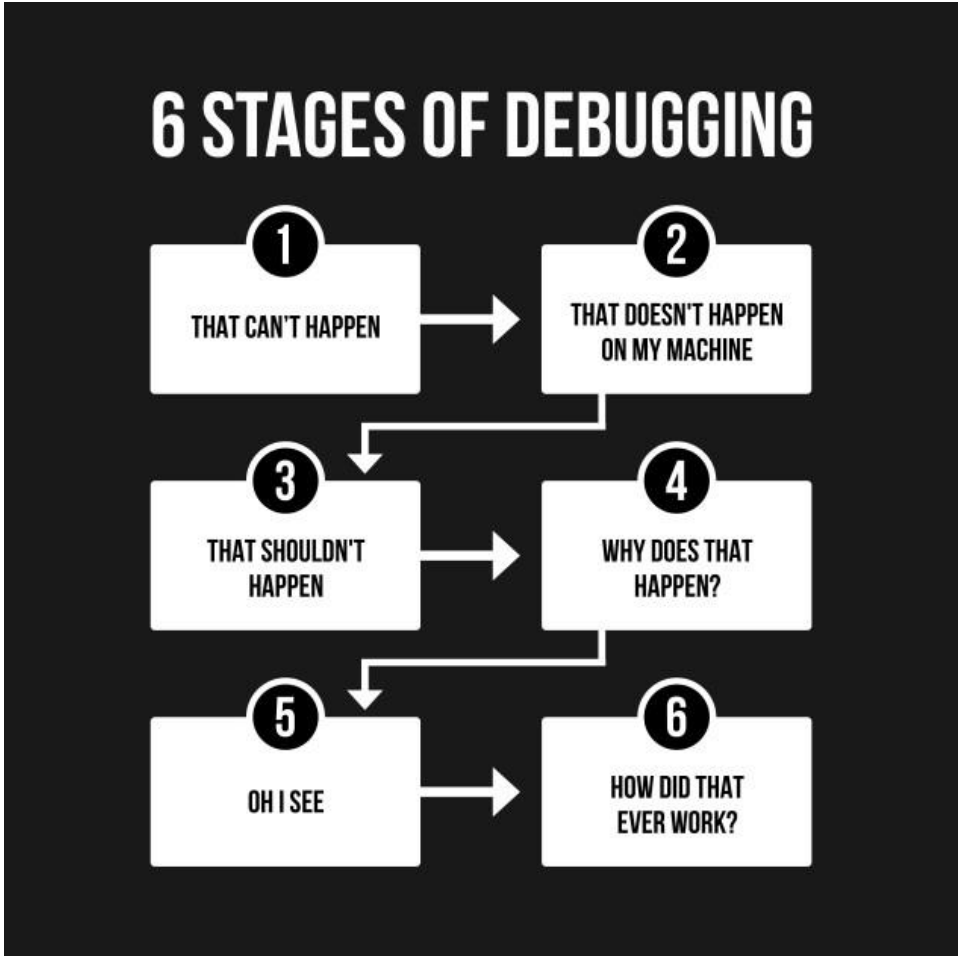
- Übung
  - Übung01_Exceptions - Texteditor
  - Übung02_Exceptions – Taschenrechner

Debugging - Definition

- Vorgang bei der Programmherstellung, bei dem das Programm getestet wird und die entdeckten Fehler beseitigt werden.

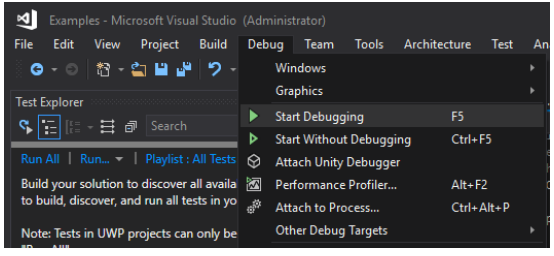
- Der Begriff „debugging“ (zu deutsch *entwanzen*) basiert auf der von Grace Hopper eingeführten Bezeichnung für Fehler in Computersystemen. 1947 hatte während der Arbeiten am Mark II eine Motte für den Ausfall eines Relais dieses Computers gesorgt. Grace Hopper hat die (tote) Motte in das Logbuch geklebt und mit dem Satz „First actual case of bug being found.“ („Das erste Mal, dass tatsächlich ein Bug gefunden wurde.“) kommentiert. Der Begriff „Bug“ (für Insekt, Käfer, Schädling) war im Englischen unter Ingenieuren bereits seit längerer Zeit als Bezeichnung für Fehlfunktionen in Gebrauch. Mit *Bugfix* (engl. *fix* für reparieren, ausbessern) wird die Behebung eines Programmfehlers bezeichnet.
- https://de.wikipedia.org/wiki/Debugger

Debugging - Definition

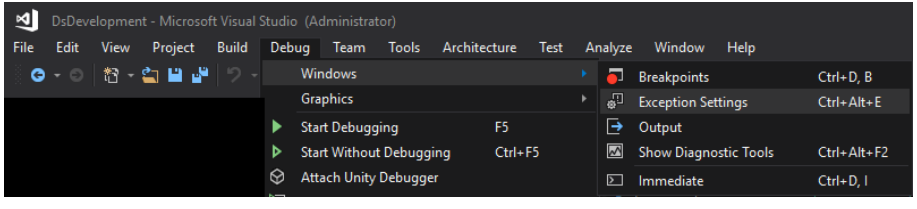


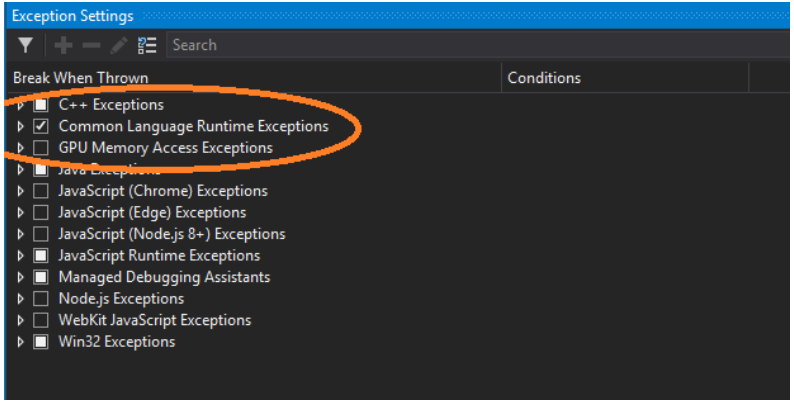
Debugging – Visual Studio

- Start Debugging F5



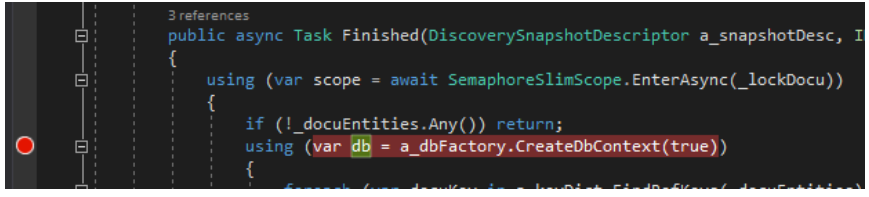
- Exceptions einschalten



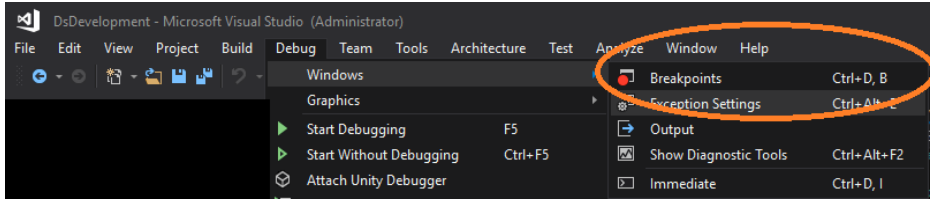


Debugging – Visual Studio - Breakpoints

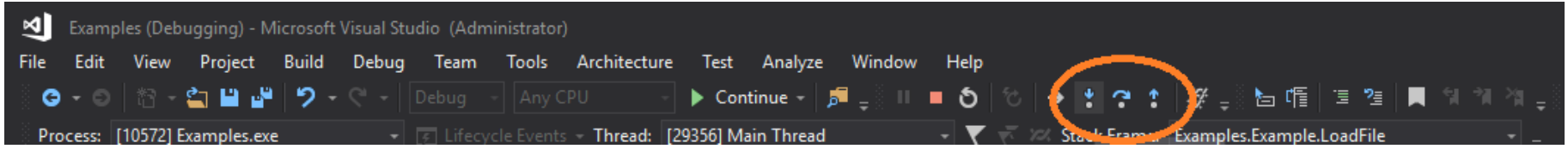
- Läuft eine Applikation im Debug Modus und passiert der ausgeführte Code einen Breakpoint, so bleibt die Codeausführung stehen. (F5/F10/F11)
- Der aktuelle Status kann ausgelesen werden (CallStack/Variablen…)
- Breakpoint können mit F9 oder durch einen Klick am Rand des Codeeditors eingefügt werden.



- Breakpoints können deaktiviert und Bedingungen enthalten
- Eine Liste aller Breakpoints kann man im Breakpoints Tool Fenster finden



Debugging – Visual Studio



- F10 – Single Step
  - Schritt im aktuellem Codeblock „weiterschreiten“

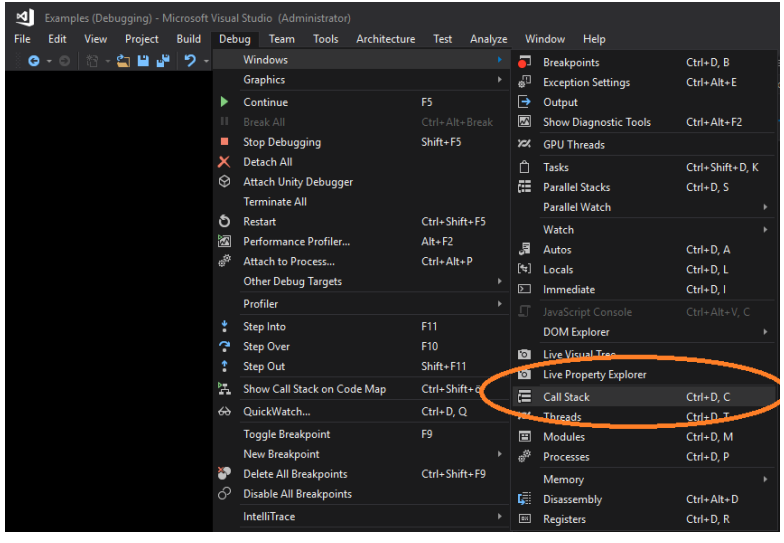
- F11 – Step Into
  - In eine aktuelle Funktion einsteigen

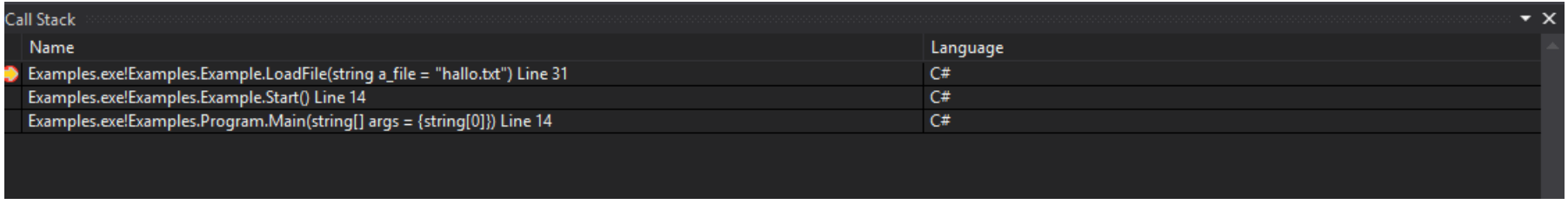
Debugging – Call Stack

- Beantwortet die Fragen
  - Welche Codestelle hat die aktuelle Funktion aufgerufen?
  - Von woher komme ich?

- Springen zwischen Call Stack Einträgen möglich
  - Lokale Variablen…

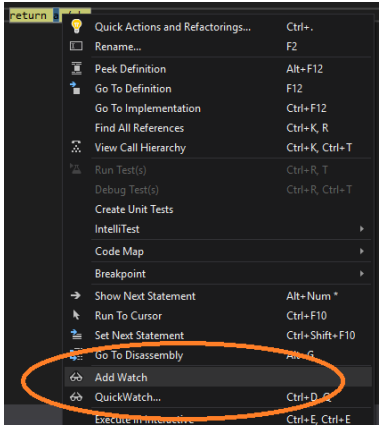
Debugging – Visual Studio - CallStack

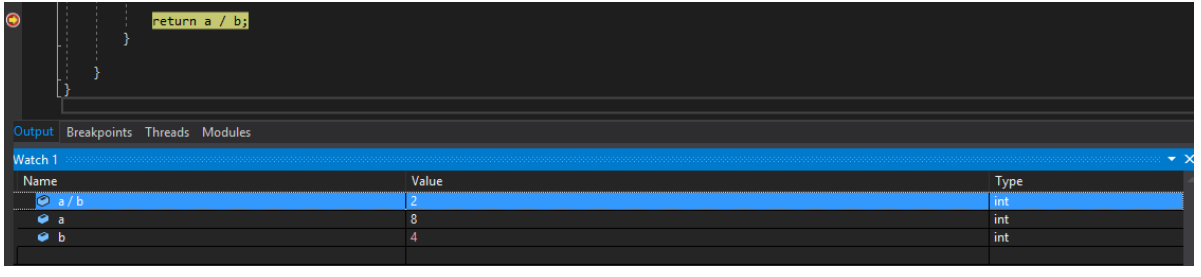




Debugging - Watch

- Ermöglicht das beobachten von bestimmten ausgewählten Variablen bzw. Statements.





Debugging - Übungen

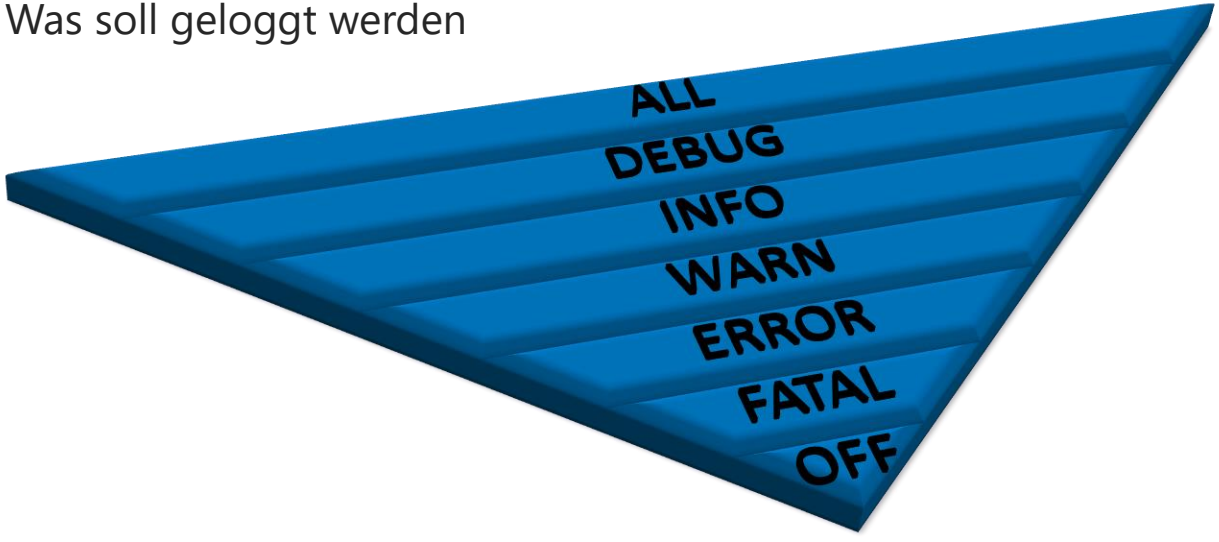
- Übung03_Exceptions - Debugging

Logging

- Warum benötigt man Logging
  - Nachvollziehbarkeit zu einem späteren Zeitpunkt.
  - Problem tritt nur beim Kunden auf.
    - Logdatei -> Übermittlung per E-Mail
  - Kein direkter Zugriff auf Kundenumgebung.
  - Kein Debugger/Visual Studio verfügbar.

- Was loggen?
  - Wann ist etwas passiert - Zeitstempel
  - Fehler – Exceptions
  - Programmablauf

- LogLevel (Beispiel log4net)
  - Filter – Was soll geloggt werden



- Wohin loggen?
  - Console – (Nur für Live-Debugging zu empfehlen, da Daten nicht gespeichert werden)
  - Datei
  - Datenbank
  - Cloud (Azure – Application Insight)
  - …

Logging - Implementierung

- Empfehlung: Keine eigenes Logging Framework implementieren

- Bibliotheken/Nugets
  - log4net
  - Serilog
  - Microsoft.Extension.Logging
  - …

Logging – Log4net

- Schnittstelle die Logging ermöglicht
  - ILog
    - Log.Debug()
    - Log.Error()
    - …
- Konfiguration
  - LogLevel setzen
  - Appender definieren
    - FileAppender
    - ConsoleAppender
  - …

Logging – Log4net - Implementierung

- Log4net „installieren“
  - Nuget Paketverwaltung
    - nach log4net suchen
    - installieren.

- Log4net konfigurieren
  - App.config erweitern
  - Beispiele: https://csharp.today/log4net-tutorial-great-library-for-logging/
  - Unter Properties -> In AssemblyInfo.cs folgende Zeile einfügen.
    - [assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config")]
  - Beim Start des Programms, z.B. in der Main Funktion folgende Zeile hinzufügen
    - log4net.Config.XmlConfigurator.Configure();

Logging – Log4net - Implementierung

- Einen Logger initialisieren
  - private static readonly ILog _log = LogManager.GetLogger(MethodBase.GetCurrentMethod().DeclaringType);
- Logger nutzen
  - _log.Debug(„…“)
  - _log.Error(„…“)

Logging – Log4net - AppConfig

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net"/>
  </configSections>
  <log4net>
    <root>
      <level value="ALL" />
      <appender-ref ref=„file" />
    </root>
    <appender name=„file" type="log4net.Appender.FileAppender">
      <file value="application.log" />
      <appendToFile value="true" />
      <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date %level %logger - %message%newline" />
      </layout>
    </appender>
  </log4net>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.2" />
    </startup>
</configuration>
```

Loggin - Übung

- Übung04_Logging

Quellen / Verweise / Interessantes...

- http://openbook.rheinwerk-verlag.de/visual_csharp_2012/1997_07_001.html
- https://stackify.com/log4net-guide-dotnet-logging/