Ausnahmen (Exceptions)

try-Anweisung


```
FileStream s = null;
try {
    s = new FileStream(curName, FileMode.Open);
    ...
} catch (FileNotFoundException e) {
    Console.WriteLine("file {0} not found", e.FileName);
} catch (IOException) {
    Console.WriteLine("some IO exception occurred");
} catch {
    Console.WriteLine("some unknown error occurred");
} finally {
    if (s != null) s.Close();
}
```

- *catch*-Klauseln werden in der Reihenfolge ihrer Aufschreibung getestet.
- optionale *finally*-Klausel wird immer ausgeführt.
- Exception-Name in *catch*-Klausel kann entfallen.
- Exception-Typ muß von *System.Exception* abgeleitet sein.
  Fehlt er, wird *System.Exception* angenommen.

System.Exception

Properties

e.Message	die Fehlermeldung als String; wird eingestellt durch *new Exception(msg);*
e.Source	Name der Applikation oder des Objekts, das die Ausnahme ausgelöst hat
e.StackTrace	die Methodenaufrufkette als String
e.TargetSite	das Methodenobjekt, das die Ausnahme ausgelöst hat
...	

Methoden

e.ToString()	liefert den Namen der Ausnahme
...	

Auslösen von Ausnahmen

Durch ungültige Operation (implizit)

Division durch 0
Indexüberschreitung
null-Zugriff
...

Durch throw-Anweisung (explizit)

```
throw new FunnyException(10);
```

```
class FunnyException : ApplicationException {
    public int errorCode;
    public FunnyException(int x) { errorCode = x; }
}
```

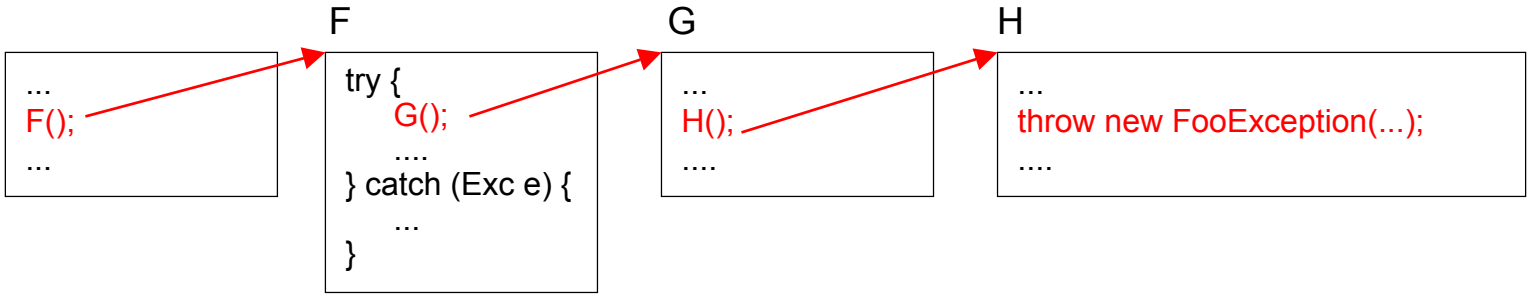
(Aufruf einer Methode, die eine Ausnahme auslöst und nicht behandelt)

```
s = new FileStream(...);
```

Exception-Hierarchie (Auszug)

- **Exception**
  - **SystemException**
    - ArithmeticException
      - DivideByZeroException
      - OverflowException
      - ...
    - NullReferenceException
    - IndexOutOfRangeException
    - InvalidCastException
    - ...
  - **ApplicationException**
    - ... eigene Exceptions
    - ...
  - **IOException**
    - FileNotFoundException
    - DirectoryNotFoundException
    - ...
  - **WebException**
  - ...

Suche nach passender catch-Klausel



Ruferkette wird rückwärts nach passender catch-Klausel durchsucht.
Wenn keine gefunden => Programmabbruch mit Fehlermeldung und Stack-Trace

Ausnahmen müssen in C# nicht behandelt werden (im Gegensatz zu Java)
Keine Unterscheidung zwischen *Checked Exceptions* und *Unchecked Exceptions*
Grund (wahrscheinlich): nicht alle .NET-Sprachen haben Checked Exceptions (z.B. C++)

+ bequemer
- weniger robuste Software

Ausnahmen müssen nicht im Methoden-kopf spezifiziert werden

Java

```
void myMethod() throws IOException {
    ... throw new IOException(); ...
}
```

Rufer von *myMethod* müssen

- *IOException* behandeln, oder
- *IOExceptions* in eigenem Methodenkopf spezifizieren

C#

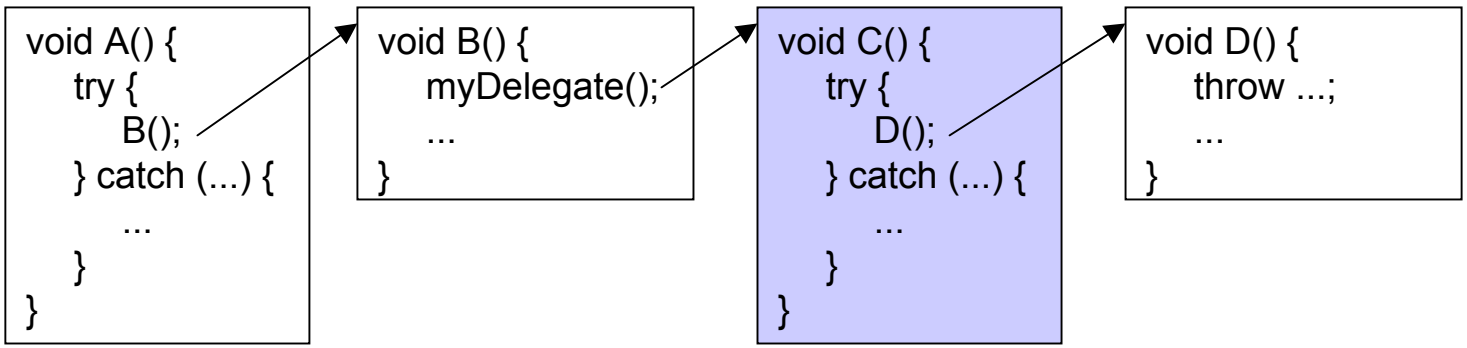
```
void myMethod() {
    ... throw new IOException(); ...
}
```

Rufer von *myMethod* können *IOException* behandeln, müssen es aber nicht.

+ kürzer und bequemer
- weniger sicher und robust

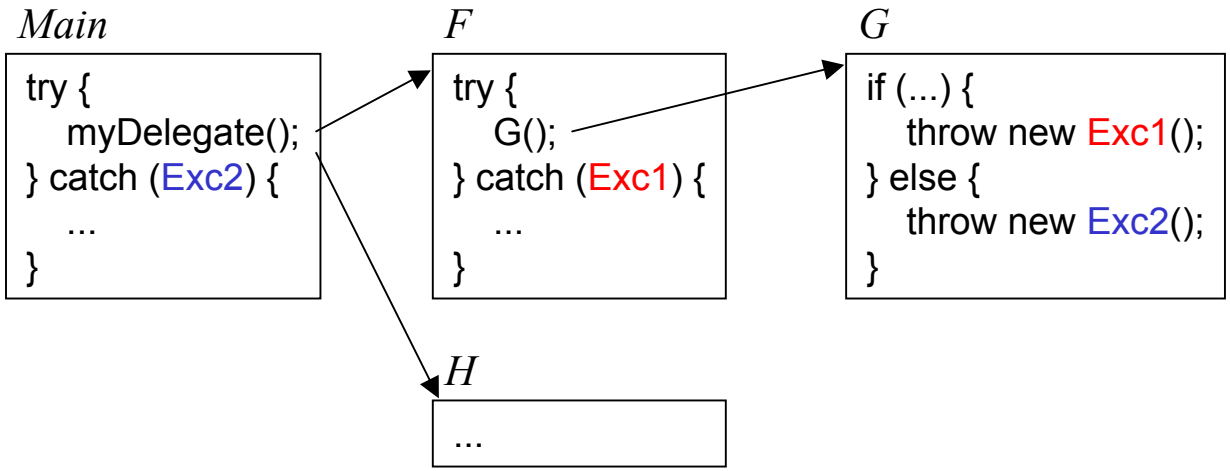
Ausnahmen in Delegates

Delegates werden bei der Suche nach catch-Klausel wie normale Methoden behandelt.



als Delegate aufgerufen

Ausnahmen in Multicast-Delegates



- tritt in *G()* die Ausnahme *Exc1* auf, wird auch noch *H()* aufgerufen
- tritt in *G()* die Ausnahme *Exc2* auf, wird *H()* nicht mehr aufgerufen, weil *Exc2* erst in *Main()* abgefangen wird.