

LANDESBERUFSSCHULE 4 SALZBURG

Informatik

Ausnahmebehandlung

LBS 4

Inhalt

Ausnahmebehandlung -- Exception.....	3
Mögliche Fehler in Programmen:	4
Abfolge der Ausnahmebehandlung:	5
Ablauf bei Ausnahmebedingungen:.....	5
Schritte:	5
Schlüsselwörter:	6
Ablauf einer Exception:.....	6
Beispiel C++ Error-Handling mit dem Datentyp string:	7
Beispiel C++ eigene Fehlerklasse MathError:	8
Standardfehlerklassen:.....	Fehler! Textmarke nicht definiert.

Ausnahmebehandlung -- Exception

Die Fehlerbehandlung ist ein sehr wichtiger Aspekt bei der Softwareentwicklung. Es gibt viele Situationen die nicht vorhersehbar sind – Dateien fehlen, Inhalt ist nicht konform, Speichermangel usw. – diese müssen behandelt, abgefangen werden. Das Ziel ist Programmabstürze und fehlerhaftes Verhalten der Software zu vermeiden. Die Erreichung der Zielsetzung und die Fehlerbehandlung sind aber völlig unabhängige Aspekte.

Prinzipiell ist die Verwendung von Exceptions/Ausnahmen für den Fall gedacht, dass Entwickler zu einem bestimmten Zeitpunkt zur Laufzeit feststellen können, ob ein Problem vorliegt. Jedoch wissen sie nicht, wie sie dieses Problem behandeln sollen. Folgende Logik versteckt sich nun hinter dem exception-handling Mechanismus, um die Problemerkennung von der Problembehandlung zu trennen:

1. Der Teil der Software, der ein Problem erkennt, wirft eine Exception (daher kommt auch das Keyword `throw` zum Einsatz, dass wir noch kennen lernen werden).
2. Der Teil der Software, der sich bereit erklärt, bestimmte Probleme behandeln zu können, signalisiert diese Bereitschaft durch eine Deklaration, eine Exception fangen zu können (daher kommt auch das Keyword `catch`, das wir noch kennen lernen werden).
3. Wenn eine Exception geworfen wird, dann wird in der Methoden und Funktionen-Aufrufhierarchie des Programms so lange zurück zum Aufrufenden gesprungen (dies nennt sich auch Stack Unwinding), bis ein fangender Teil gefunden wird. Diesem wird die Exception dann zur Behandlung zugeführt.

Eine Exception zu werfen bedeutet nicht, dass unbedingt etwas völlig Katastrophales passiert wäre. Es ist auch keineswegs gesagt, dass Exceptions „so gut wie nie“ auftreten bzw. auftreten dürfen. Exceptions werden immer dann gebraucht, wenn etwas passiert, was dem regulär geplanten Programmablauf zuwiderläuft.

Zur Fehlerbehandlung verwendet man am besten nachstehenden Ansatz:

- Fehler erkennen und melden
 - eine Ausnahme/Exception werfen
 - Schlüsselwort `throw`
- Codestellen auf Ausnahmen überprüfen
 - Schlüsselwort `try`
- Behandlung der aufgetretenen Ausnahme
 - Abfangen des Fehlers
 - Schlüsselwort `catch`

Exceptions treten erst zur Laufzeit auf. Besonders bei arithmetischen Operationen, Überschreitung von Arraygrenzen oder bei einer Typkonvertierung. Programme können trotz dieser Fehler weiterarbeiten, wenn der Fehler richtig behandelt wird. Ein weiteres Ziel des exception-handlings ist es, die Übersichtlichkeit zu verbessern, indem man sicheren Quellcode von möglicherweise fehlerverursachendem Quellcode trennt.

Das Abfangen der Fehler – die Reaktion – kann an anderer Stelle erfolgen.

Ein Fehler liegt nur dann vor, wenn Ausnahmesituationen nicht im Programm berücksichtigt sind und eine unkontrollierte Programmbeendigung erfolgt.

Mögliche Fehler in Programmen:

- Division durch Null,
- Bereichsüberschreitung eines Arrays,
- Syntaxfehler bei Eingaben,
- Zugriff auf eine nichtgeöffnete Datei,
- Fehlschlag der Speicherbeschaffung oder Nichteinhaltung der Vorbedingung einer Methode


Für diese Fehler müssen Strategien bereitgestellt werden, wobei manche Situationen nicht behandelbar sind (z.B.: OS-Fehlersituation, Stack-Overflow, ..).

Mögliche Varianten sind

- Programmabbruch
- Parameterübergabe
- Error-Funktion

Für die Qualität eines Programmes ist es wichtig, Fehlerbehandlungen so zu implementieren, dass jede mögliche Fehlersituation berücksichtigt wird.

Abfolge der Ausnahmebehandlung 2:

- 
1. Fehlererkennung
 - Überprüfen von Programmteilen
 2. Reaktion auf Fehler
 - Selbständig ohne Benutzereingabe
 3. Weiterführung
 - Fortführen
 - Beendigung mit Sicherung

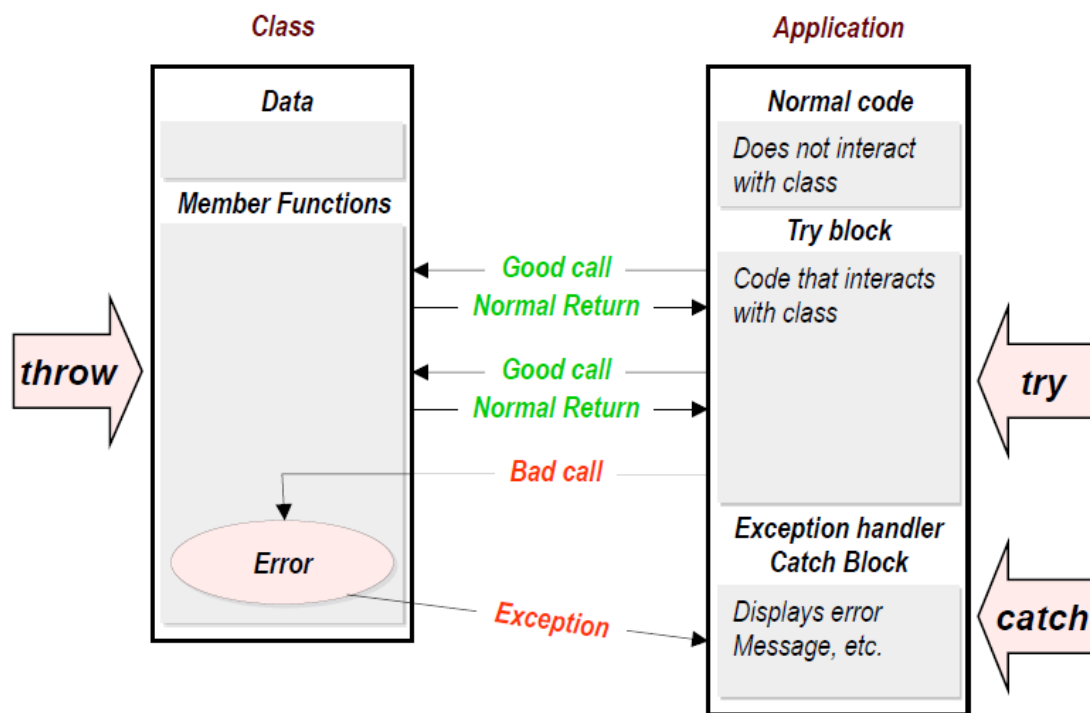
In C++ kann eine Ausnahme als Objekt erzeugt werden. Dieses kann an anderer Stelle bearbeitet werden. Die Ausnahmen werden weitergereicht, bis die Bearbeitungsfunktion gefunden wird.

Ablauf bei Ausnahmebedingungen:

Schritte:

1. Beobachter Applikationsblock: **try**
2. Erzeugen der Ausnahme: **throw**
3. Methoden werden statt **throw** mit **noexcept(false)** deklariert
4. Behandlung der Ausnahme: **catch**

Das Schlüsselwort **noexcept** kennzeichnet Funktionen und Methoden welche keine Exception auslösen.



Schlüsselwörter:

- try
 - umfasst den Block, wo „Exceptiones“ erzeugt werden
- throw , noexcept
 - der throw-block ist in der Definition der Methode/Funktion die eine Ausnahme erzeugen kann
- catch
 - fängt das „Exception-Objekt“ ab und führt eine Ausnahmebehandlung durch

Ablauf einer Fehlerbehandlung:

- Wird eine Fehlerbedingung erreicht, so wird ein Exception-Objekt erzeugt (throw) und an eine neutrale Stelle kopiert.
 - es wird immer eine Kopie des exception-Objektes erzeugt
- dann erfolgt die Suche nach dem Ende des Try-Blocks. Wird kein Ende gefunden wird an die nächsthöhere aufrufende Funktion übergeben (aufrollen des Stacks – unwinding).
- nach dem Ende geht die Kontrolle zum nächsten catch-Block

Catch kann alle möglichen Datentypen abfangen. Nur der Typ *void* ist nicht erlaubt!

Beispiel C++ Error-Handling mit dem Datentyp string:

```
//Klasse zum Berechnen division.h
class Division{
public:
double calc(double a, double b) noexcept(false);
};

//Definition calc(int a, int b) division.cpp
double Division::calc(double a, double b) noexcept(false){
if(b == 0.0){
    throw string("Division durch 0");
}
if(b < 0){
    throw string("Der Nenner ist Negativ");
}
return a/b;
}
```

```
#include <iostream>
using namespace std;
int main(){
Division d;
    double a,b;
    double erg;
    try{
        cout << "Geben Sie die Zahlen für die Division ein" << endl;
        cin >> a >> b;
        erg = d.calc(a,b);
    }
    catch(string& e)
    {
        cout << "Fehler bei der Division. Meldung: " << e << endl;
        return(1);
    }
    cout << "Ergebnis:" << erg << endl;
    return 0;
}
```

Weiteres kann man eigene Klassen zur Fehlerbehandlung erstellen. In diesen Klassen kann eine spezielle Fehlerroutine enthalten sein. Nachstehend ist ein Beispiel angeführt.

Beispiel C++ eigene Fehlerklasse MathError:

```
//Error Klasse
class MathError {
private:
    string message;
public:
    MathError(const string& s):message(s){}
    string& getMessage(){return message;}
};
```

```
//Klasse zum Berechnen
class Division{
public:
    double calc(double a, double b) noexcept(false);
};
```

```
double Division::calc(double a, double b) noexcept(false){
    if(b == 0.0){
        throw MathError("Division durch 0");
    }
    if(b < 0){
        throw MathError("der Nenner ist Negativ");
    }
    return a/b;}

```

```
#include <iostream>
using namespace std;

int main(){
    division d;
    int a,b;
    double erg;
```



```
try{
    cout << "Geben Sie die Zahlen für die Division ein" << endl;
    cin >> a >> b;
    erg = d.calc(a,b);
}
catch(MathError& e)
{
    cout << "Fehler bei der Division. Meldung: " << e.getMessage() << endl;
    return(1);
}
cout << "Ergebnis:" << erg << endl;
    return 0;
}
```