# Seminarblatt 3

## Aufzählungstypen (ENUMs)

Wenn eine Variable, ein Attribut eine begrenzte Menge von Werten oder Zuständen annehmen können soll, dann kann man einen solchen Wertetyp durch Aufzählung definieren (**ENUM**).

***SOURCE CODE***

```
// DECLARATION
enum ConnectionState {
    Connecting, Established, Closing, Interrupted };

// USE
ConnectionState state = ConnectionState::Closing;

switch (state)
{
case Established:
    cout << "State is Established";
    break;
case Connecting:
    cout << "State is Connecting";
    break;
case Closing:
     cout << "State is Closing";
     break;
default:
    cout << "State is interrupted";
}
```

Die Verwendung von ENUMs ist vorzuziehen gegenüber Integer-Konstanten.

- Es erlaubt die Überprüfung der Typsicherheit durch den Compiler!
- Macht den Code lesbarer!

## Ausnahmen (Exceptions)

Wenn eine Programmsituation keine sinnvolle Fortführung erlaubt, tritt eine Ausnahme auf.
Das ermöglicht eine Unterscheidung zwischen Rückgabewerten und Fehlersituationen.
Vorhersehbaren und unvorhersehbaren Fehlersituationen können in C++ mit dem Konstrukt der **Exception**s behandelt werden.
Ausnahmen können verursacht werden (**THROW**). Codeabschnitte können angeben, dass sie Ausnahmen erwarten (**TRY**), abfangen und behandeln (**CATCH**).

Wenn für eine Ausnahme keine Behandlung definiert ist, dann wird ein **Default-Handler** aktiviert, der das Programm mit einer Fehlerausgabe abgebrochen.

| SOURCE CODE | OUTPUT |
|---|---|
| `#include <iostream>`<br>`using namespace std;`<br><br>`class Test {`<br>`public:`<br>`  Test() {`<br>`  cout << "Constructor of Test" << endl;`<br>`  }`<br>`  ~Test() {`<br>`  cout << "Destructor of Test" << endl;`<br>`  }`<br>`  void check() {`<br>`  // do something`<br>`  cout << "causing a problem" << endl;`<br>`  throw exception();`<br>`  }`<br>`};`<br><br>`int main() {`<br>`  try {`<br>`  Test t1;`<br>`  t1.check();`<br>`  // do something else`<br>`  cout << "after problem" << endl;`<br>`  } catch (exception &e) {`<br>`  cout << "Caught exception" << endl;`<br>`  }`<br>`}` | *Constructor of Test*<br><br>*Causing a problem*<br><br>*Destructor of Test*<br><br>*Caught exception* |

Ausnahmen sind Objekte, und können Informationen über die Art des Fehlers beinhalten.
Konstruktor: **`invalid_argument(const string& what);`**
Die Klasse **`exception`** aus der Standardbibliothek dient als Basisklasse. Spezielle Fehler können als Unterklassen von **`exception`** definiert werden [cppreference.com/w/cpp/error/exception](cppreference.com/w/cpp/error/exception)
Einige sind bereits in der Standardbibliotek <exception> definiert:

| logic_error | runtime_error |
|---|---|
| invalid_argument | range_error |
| domain_error | overflow_error |
| length_error | underflow_error |
| out_of_range | regex_error |
| future_error | system_error |
| | tx_exception |
| | format_error |

Der Catch block hat einen Parameter (wie eine Methode), und wird nur bei passenden Exceptions aktiviert. Aber die Vererbung erlaubt das Fangen von Unterklassen!

Beispiel: