Exceptions in C++

Usage, pitfalls, pros and cons

dkfz.
DEUTSCHES
KREBSFORSCHUNGSZENTRUM
IN DER HELMHOLTZ-GEMEINSCHAFT

Catching exceptions

- Multiple catch blocks allowed
- Only one is executed
  (no Java - finally)
- The first one that matches is executed → put base classes at the end
- Always catch by reference

```
try
{
  Poco::XML::DOMParser parser;
  Poco::AutoPtr<Document> pDoc =
          parser.parse("xml.xml");
}
catch (Poco::XML::Exception& exc)
{
  std::cerr << exc.displayText() <<
  std::endl;
}
catch (Poco::Exception& exc)
{
  std::cerr << exc.displayText() <<
  std::endl;
}
catch (...)
{
  std::cerr << "an error occured" <<
  std::endl;
}
```

Throwing exceptions

- Everything can be thrown – limit yourself to real exception classes
- Throw exceptions by value
- Do not throw exceptions in destructors (!)
- Exceptions may be re-thrown

```
mitk::Image::Pointer
  segmentAll(mitk::Image::Pointer image)
{
  if(image.IsNull())
    throw std::invalid_argument("Image is 0!");
  // ...
}

void f()
{
  try {
   // ...
  }
  catch (MyException& e) {
    e.addInfo("f() failed");
    throw;
  }
}
```

Exception declarations

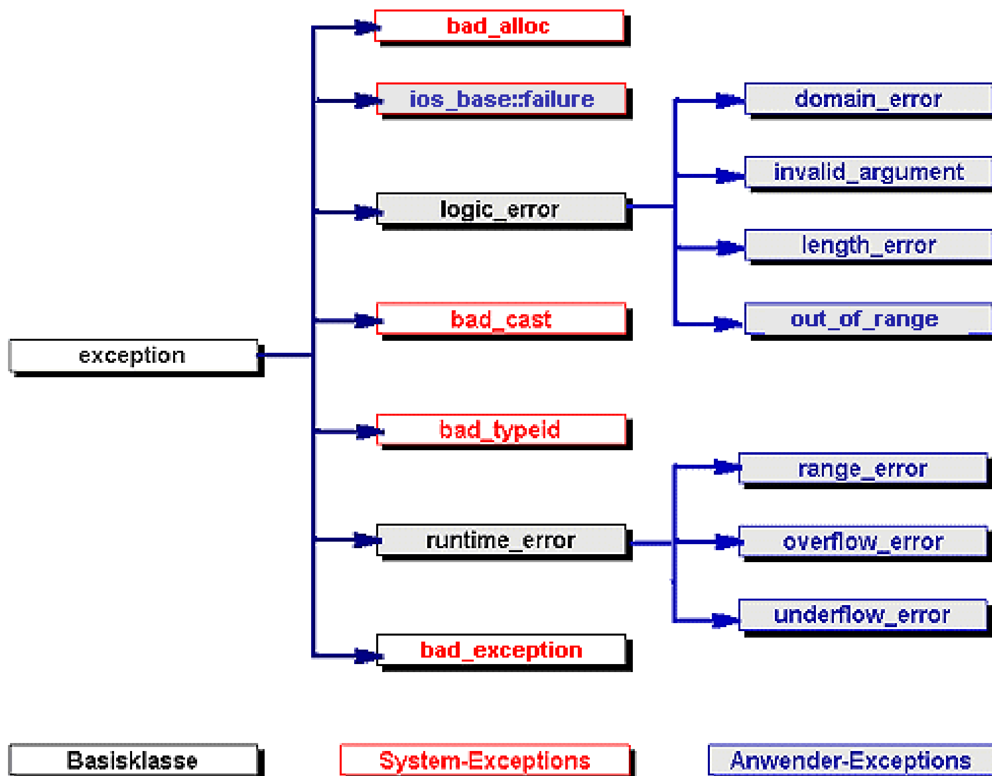
- Use throw to declare which exception can be thrown by your function
- Violations do not cause compiler errors (!)
- If violated, `unexpected()` will be called at runtime

```
mitk::Image::Pointer
  segmentAll(mitk::Image::Pointer image)
  throw (std::invalid_argument)
{
  // this function should only throw
  // std::invalid_argument exceptions
}
```

```
mitk::Image::Pointer
  segmentAll(mitk::Image::Pointer image)
  throw () // no exceptions allowed
{
  try { // ... }
  catch( … ) {
   // ...
  }
}
```

Predefined std exceptions

dkfz.




```
// out_of_range
std::bitset<8> bits;
try
{
  bits.set(9);   // out_of_range!
}
catch(const std::out_of_range& ex)
{
  std::cout << ex.what() << std::endl;
}
```

Pitfalls (I): memory leaks

dkfz.

- Always use Smart Pointer in try/catch blocks

```
void f()
{
  try {
   mitk::Image* image = new mitk::Image;
   // any code throwing an exception

   delete image; // never executed
  }
  catch (…)
  {}
}

void f()
{
  try {
   mitk::Image::Pointer image =
          mitk::Image::New();
  }
  catch (…)
  {}
} // image is deleted here
```

Pitfalls (II): Using exceptions across threads

- Threads can concurrently throw excpetions
- … but: An exception should never leave a thread`s main function (abnormal program termination)

```
int thread_main_function(void*
  thread_data)
{
  throw std::exception("thread error");
  // program termination
}
```

Pitfalls (III): Using exceptions across shared libraries

dkfz.

- On Windows: No problem when using the same compiler
- On Linux gcc uses RTTI to resolve catch blocks
  - if the std::type_info is not exported, user defined exceptions will not work

```
// KaputtException.h (shared library)
struct KaputtException : public std::exception
{
  KaputtException(const std::string&);
};

void throwKaputtException();
```

```
// main.cpp (executable)
#include "KaputtException.h"

int main()
{
  try
  {
    throwKaputtException();
  }
  catch (const KaputtException& e)
  {
    // will not be caught
  }
}
```

Advantages of exception handling

dkfz.

Exceptions...

✓ separate error-handling code from the normal program flow and thus make the code more readable

✓ are the only clean way to report an error from a constructor

✓ are hard to ignore

✓ are easily propagated from deeply nested functions

✓ can be user defined types that carry much more information than an error code

Potential drawbacks

Exceptions...

- create multiple invisible exit points
- can lead to resource leaks
- are hard to introduce to legacy code
- can cause problems in DLLs and in threaded programs
- are easily abused for performing tasks that belong to normal program flow