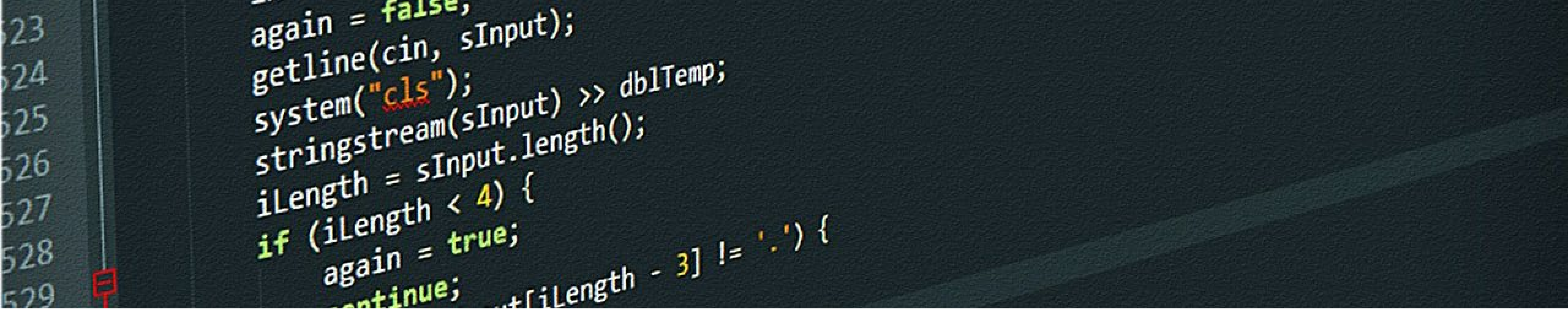


```
again = false;
getline(cin, sInput);
system("cls");
stringstream(sInput) >> dblTemp;
iLength = sInput.length();
if (iLength < 4) {
    again = true;
    ntinue;
          t[iLength - 3] != '.') {
```

Thomas

C23-05 Exception handling

Advanced algorithms and programming

v5

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Films created using material from R. Peters and P. Puschmann / HTW Berlin

Exceptions

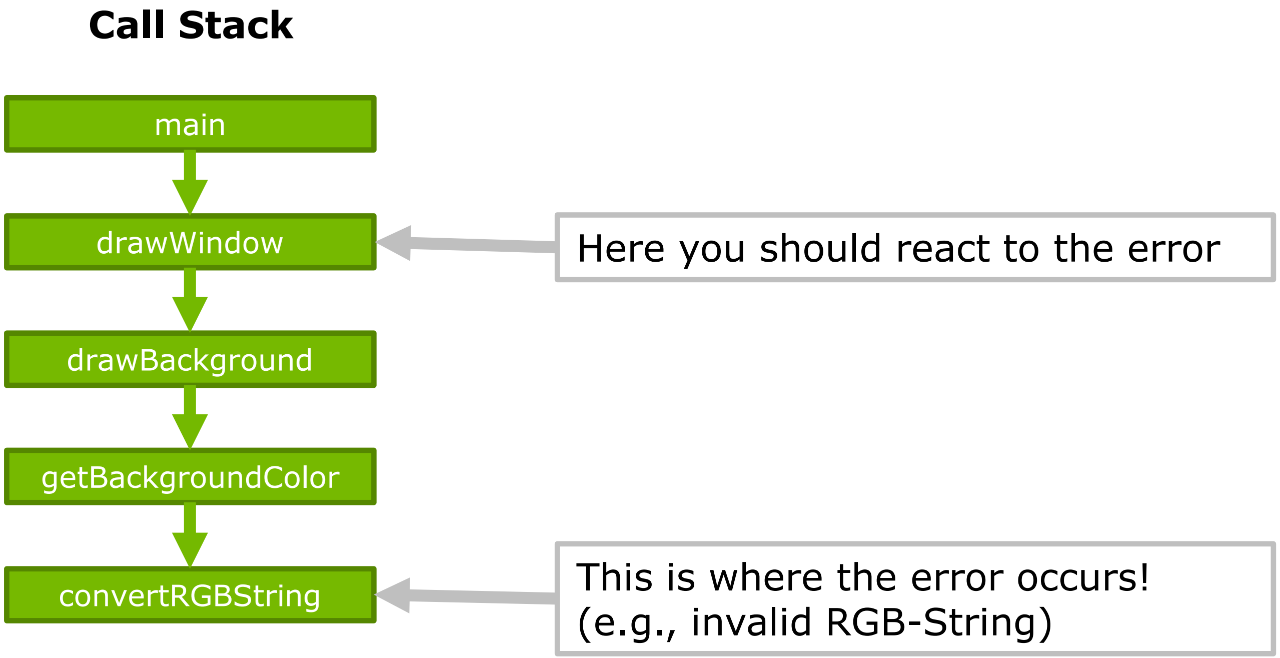
Exceptions are an error handling concept in C++

Reasons for the development of the exceptions:

- Processing of runtime errors via the return values is cumbersome, costly and error-prone
- A lot of source code has to be processed if return values indicating errors have to be changed
- Constructors cannot report errors by return values (they have no return value)

Exceptions
Principle

The principle using the example of a "drawWindow" function



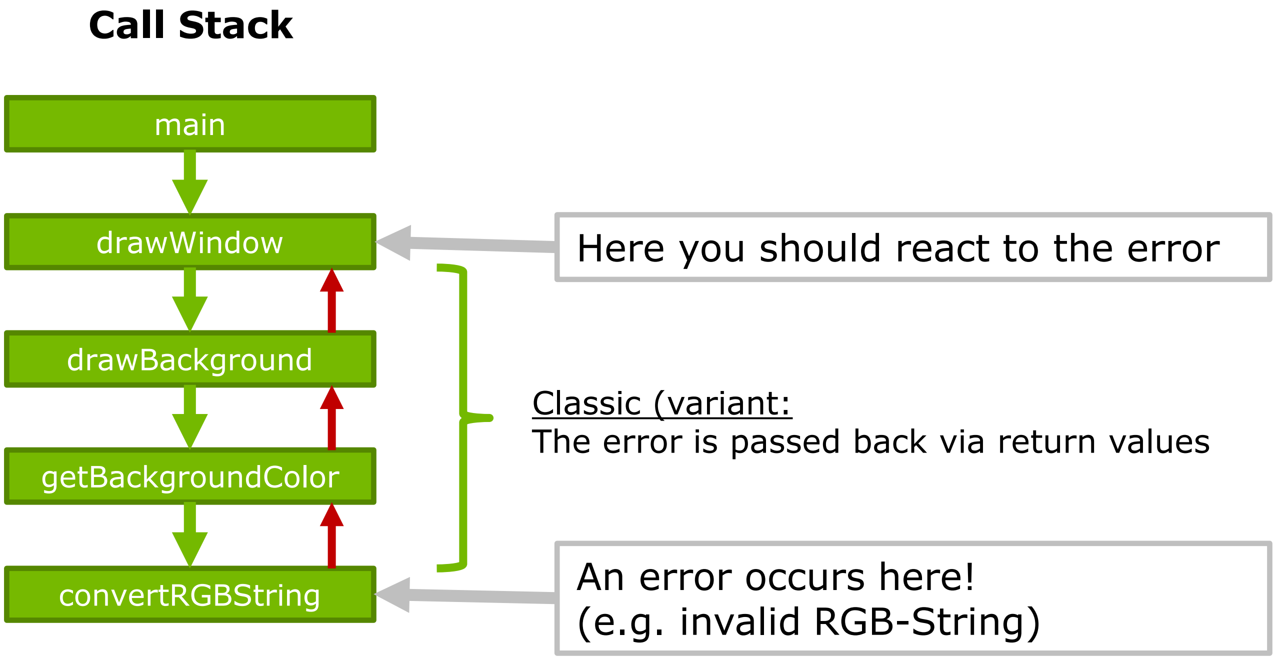
Here you should react to the error

This is where the error occurs!
(e.g., invalid RGB-String)

Exceptions

Principle

The principle using the example of a "drawWindow" function



Here you should react to the error

Classic (variant:
The error is passed back via return values

An error occurs here!
(e.g. invalid RGB-String)

Exceptions

Principle

1. The keyword `throw` throws an exception
2. The exception moves through the call stack of the program until it is processed ("caught") by a `try-catch` block

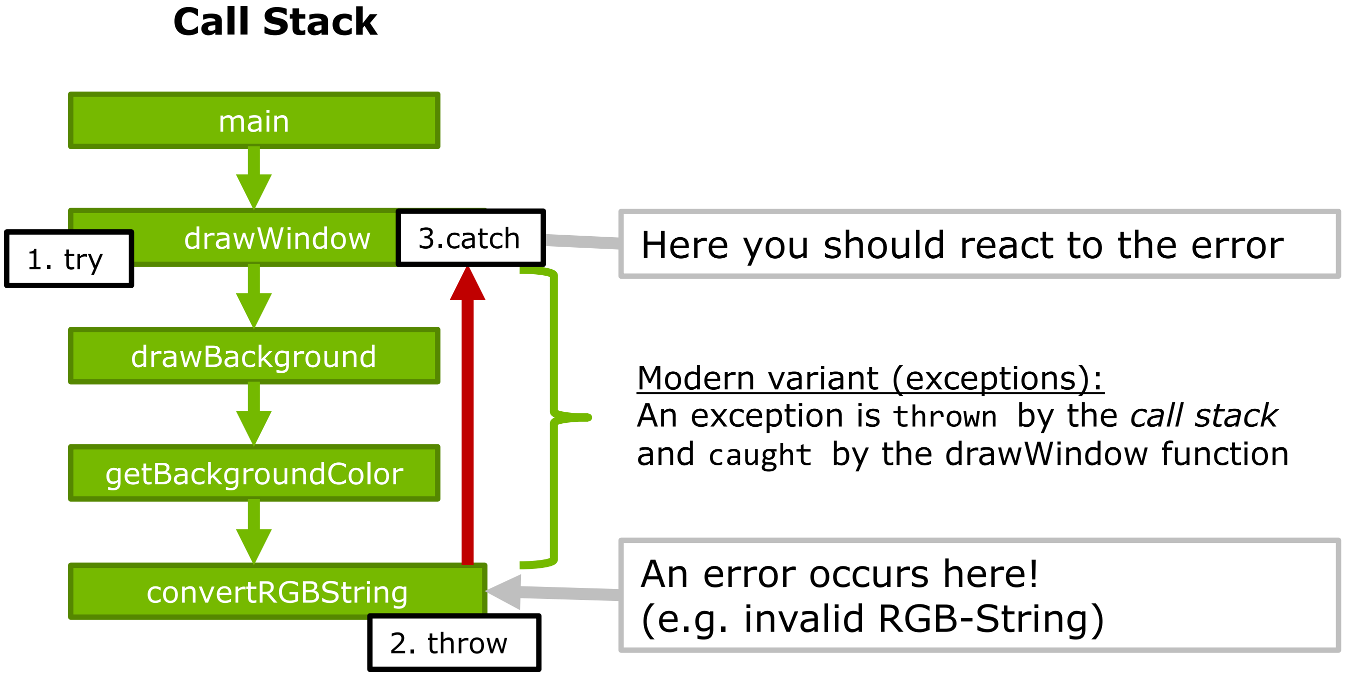
   (The "*call stack*" of a program is the hierarchy of the nested function calls in the program)

3. If an exception is not "caught", it ends up in the runtime system, leading to program crash!

Exceptions

Principle

The principle using the example of a "drawWindow" function



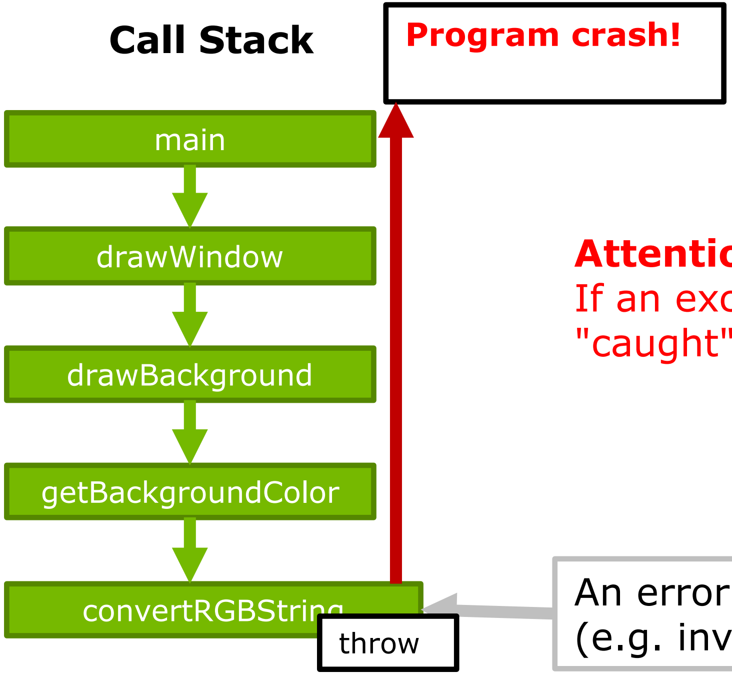
Here you should react to the error

Modern variant (exceptions):
An exception is `thrown` by the *call stack* and `caught` by the drawWindow function

An error occurs here!
(e.g. invalid RGB-String)

Exceptions
Principle

The principle using the example of a "drawWindow" function



Attention:
If an exception is "thrown" and <u>not</u> "caught", the program crashes.

An error occurs here!
(e.g. invalid RGB-String)

Exceptions

Throwing an exception

Syntax

```
throw <variable/object/pointer/etc...>;
```

Means: "An error has occurred here!

- With `throw`, all types of information can be transferred (thrown), which can be created in memory (`int` , `string`, `double`, ..., objects, pointers, ...)
  The data type determines the type of error (usually you use your own classes)
- The command `throw` interrupts the program flow!
  Program continues where the exception is caught
- The thrown element "flies" upwards in the hierarchy of function calls,
  even up to the runtime system if it is not caught within the program

Exceptions

Catching Exceptions

`try` - blocks define the area where exceptions can occur, which should be processed afterwards

```
try
{
     /* Exceptions thrown directly here or in subfunctions,
        can be caught in a subsequent 'catch' block! */
}
```

`catch` - Blocks process an exception type. A `try` block can be followed by several `catch` blocks

```
catch(< TYPE> e)
{
     /* Error handling of
        Error type: <TYPE> */
}
```

```
catch(...)
{
    // Intercepts all exceptions!
}
```

Attention:
No differentiation of error type possible

Exceptions

Catching Exceptions

Typical application pattern:

```
try
{
     /* Exceptions thrown here or in subfunctions,
        can be caught in the following 'catch'-blocks! */
}
catch(TYPE1 e)
{
     /* Error handling of error type: <TYPE1> */
}
catch(TYPE2 e)
{
     /* Error handling of error type: <TYPE2> */
}
```

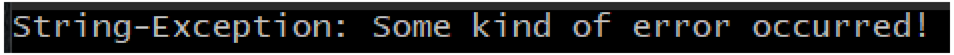
```
catch( ... )
{
     /* General error handling, if an exception has not been thrown so far
        was intercepted. */
}
```

Exceptions

Examples

Example 1

```
#include <iostream>

int main()
{
    try
    {
        throw "Some kind of error occurred!";
        std::cout << "This part will no longer be executed!" << std::endl;
    }

    catch (const char* str)
    {
        std::cout << "String-Exception: " << str << std::endl;
    }
}
```



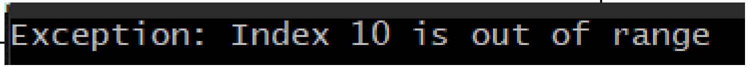
```
String-Exception: Some kind of error occurred!
```

Exceptions

Examples

Example 2

```
#include <iostream>
int main()
{
    char* mystring = nullptr;
    try
    {
        mystring = new char[10];
        if (mystring == nullptr) throw "Allocation failure";
        // Loop deliberatly runs into an exception
        for (int n = 0; n <= 100; n++)
        {
            if (n > 9) throw n;
            mystring[n] = 'z’;
        }
        mystring = nullptr;
    }
    catch (int i)
    {
        std::cout << "Exception: ";
        std::cout << "Index " << i << " is out of range" <<
std::endl;
    }
    catch (const char* str)
    {
        std::cout << "Exception: " << str << std::endl;
    }
    delete[] mystring;
}
```



```
Exception: Index 10 is out of range
```

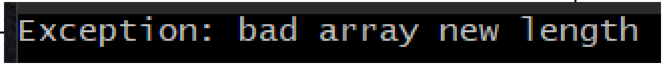
Exceptions

Examples

Example 3

Exceptions in the standard library are derived from the type `std::exception`.

```
#include <iostream>
#include <exception>

int main()
{
    int i{ -1 };
    try
    {
        // throws bad_alloc standard exception
        // try also with ...= new int[-1]
        int* myarray = new int[i];
    }
    catch (std::exception& e)
    {
        std::cout << "Exception: " << e.what() << std::endl;
    }
}
```



```
Exception: bad array new length
```

Exceptions

"Best Practice" for Exceptions

- Throw objects (instead of basic data types)
  To distinguish errors, a separate class should be implemented for each of them

- Generally avoid `catch(...) {}` blocks!
  In exceptional cases they are useful as last catch block to catch unhandled errors

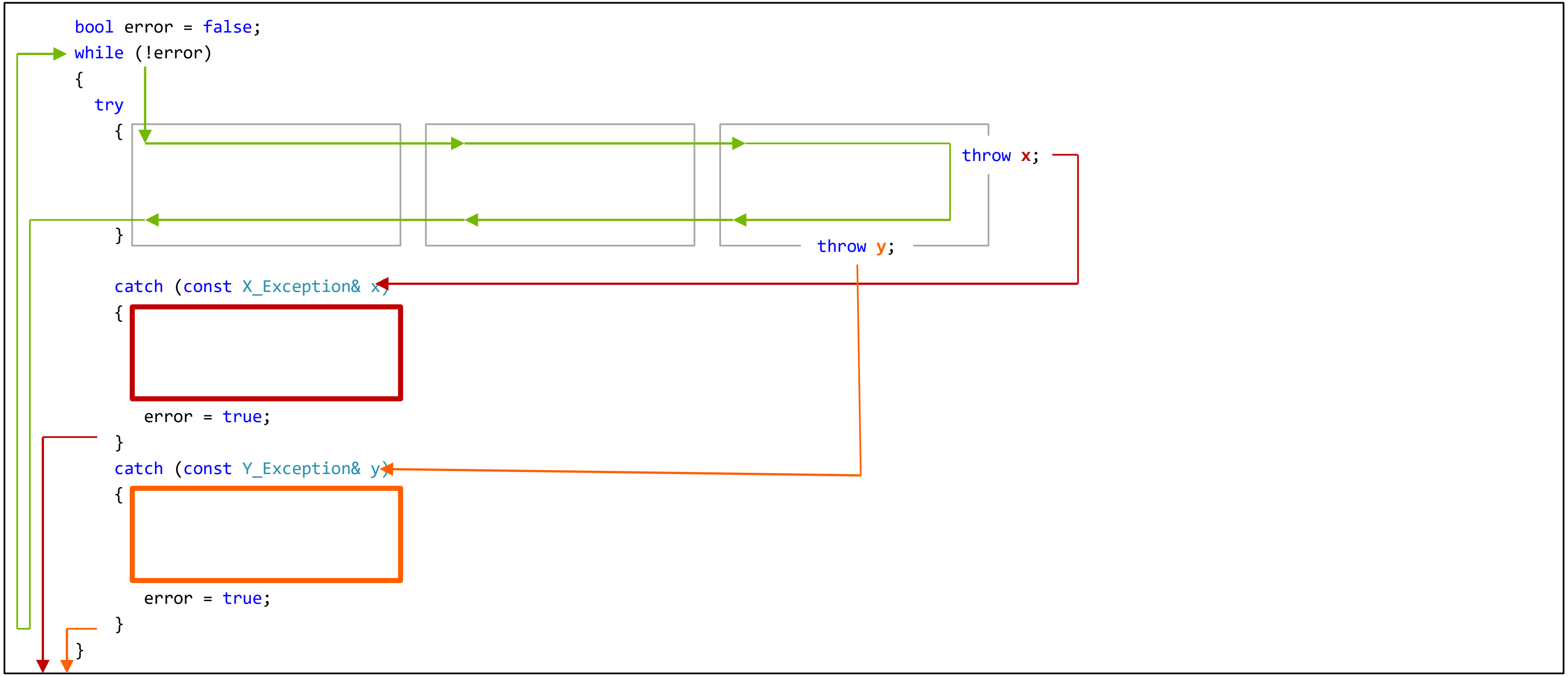
- When catching exceptions use `const references` to prevent unnecessary copying of objects

```
catch (const TYPE& e) { }
```

- Perform error handling as 'close' as possible to the error source

Exceptions

Program structure example



```
bool error = false;
while (!error)
{
  try
    {
                                                                                          throw x;

    }                                                                          throw y;

    catch (const X_Exception& x)
    {

       error = true;
    }
    catch (const Y_Exception& y)
    {

       error = true;
    }
}
```

Exceptions

Examples

Example 4

Complete example with own exception classes

Exceptions.h

```
#pragma once
#include <string>

class InitException
{
public:
    InitException(const std::string& className, const std::string& reason)
        : m_className(className), m_reason(reason)
    {}

    std::string getError() const
    {
        return "Class \"" + m_className + "\" could not be initialized ("
            + m_reason + ")\n";
    }

private:
    std::string m_className;
    std::string m_reason;
};
```

Exceptions

Examples

Example 4

Rectangle.h

```
#pragma once
#include <string>

class Rectangle
{
public:
    Rectangle(int x = 0, int y = 0, int width = 1, int height = 1);
    void setCoords(int x, int y) { m_x = x; m_y = y; }
    void setDimensions(int width, int height);
private:
    int m_x, m_y;
    int m_width, m_height;
};

class RectangleDimensionError
{
public:
    std::string getError() const
    {
        return "Rectangle has an illegal height or width!";
    }
};
```

Exceptions

Examples

Example 4

Rectangle.cpp

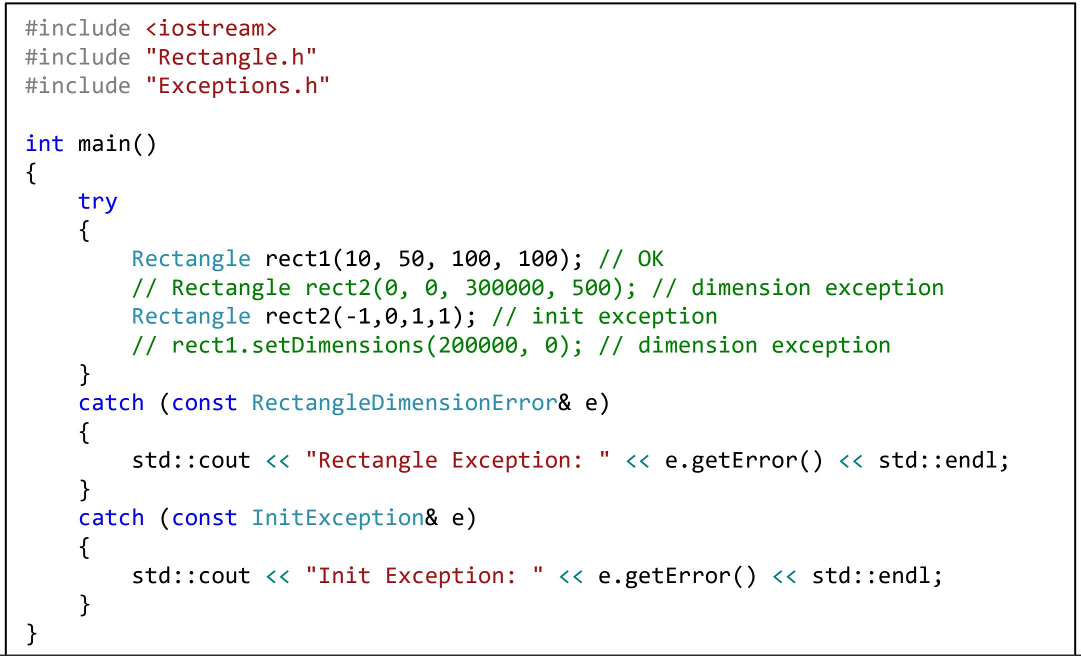
```cpp
#include "Exceptions.h"
#include "Rectangle.h"

Rectangle::Rectangle(int x, int y, int width, int height) : m_x(x), m_y(y)
{
    if (x < 0 || y < 0)
    {
        throw InitException("Rectangle", "Invalid position");
    }
    setDimensions(width, height);
}

void Rectangle::setDimensions(int width, int height)
{
    if (width < 0 || height < 0 || width > 100000 || height > 100000)
    {
        throw RectangleDimensionError();
    }
    m_width = width;
    m_height = height;
}
```

Exceptions

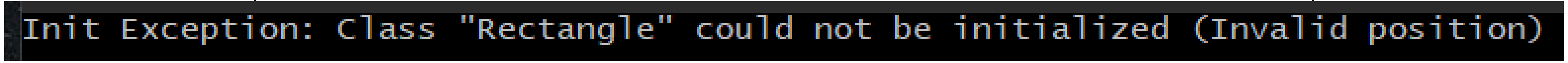
Examples

Example 4

main.cpp



```cpp
#include <iostream>
#include "Rectangle.h"
#include "Exceptions.h"

int main()
{
    try
    {
        Rectangle rect1(10, 50, 100, 100); // OK
        // Rectangle rect2(0, 0, 300000, 500); // dimension exception
        Rectangle rect2(-1,0,1,1); // init exception
        // rect1.setDimensions(200000, 0); // dimension exception
    }
    catch (const RectangleDimensionError& e)
    {
        std::cout << "Rectangle Exception: " << e.getError() << std::endl;
    }
    catch (const InitException& e)
    {
        std::cout << "Init Exception: " << e.getError() << std::endl;
    }
}
```



```
Init Exception: Class "Rectangle" could not be initialized (Invalid position)
```

Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

www.htw-berlin.de