Exceptions

Kate Gregory

@gregcons www.gregcons.com/kateblog

Not Every Action Succeeds

Errors and failures happen

Some are predictable

Some are not

Different errors can be handled different ways

Expected Errors

Simplest way to handle expected errors is to test for them

Deal with them right where they are discovered

Prompt user for better input, for example

Expected Errors

Problem:

- Sometimes the code that finds the problem cannot deal with it
- Eg business layer code can’t get message to the user

One approach:

- Have the function return an indication of trouble
- Eg UpdateTimesheet() returns true or false

Expected Errors: What If?

What if the function already returns something?

- sqrt(), FindEmployee(), etc

What if the function can't return a value (eg constructor)?

What if the developer who calls the function forgets to check the return value?

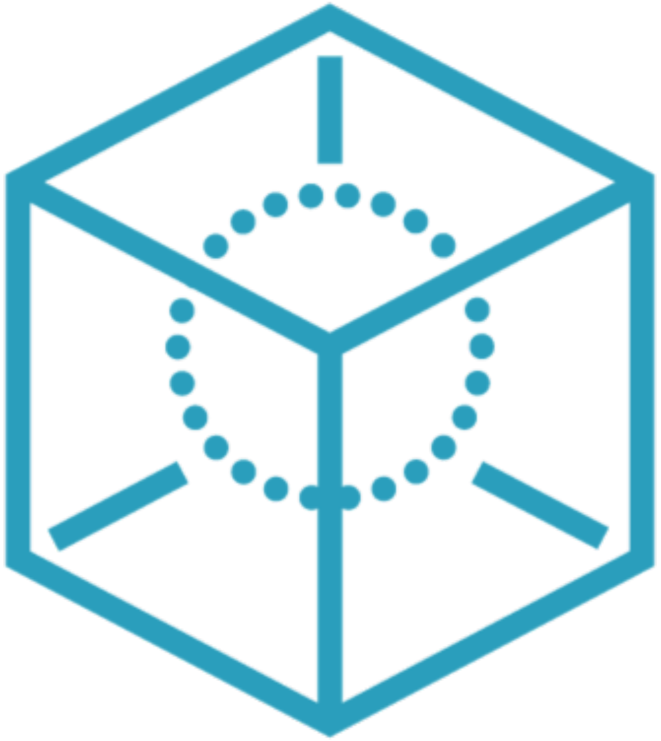
Exceptions

Transfer flow of execution

Developer can’t forget to check return value

Deal with things as close to the problem as possible

You need to know about stack unwinding



Wrap code that might throw in a try block

- as small as possible

Add one or more catch blocks after the try

Catch more specific exceptions first

Catch exceptions by reference

- Great for catching a derived exception

No finally

- That cleanup code belongs in a destructor
- Destructors run no matter how control leaves the block

```
try
{
    //risky stuff
}
catch (out_of_range& oor)
{
    // react
}
catch (exception& e)
{
    //react
}
```

- Braces around try block
- The caught exception is a local variable in the catch block
- Catch blocks are checked in order, so most general goes last

What to Throw and Catch

C++ allows you to throw and catch anything

- int, string, instance of a class

Puts quite a burden on the developer

Documentation might help

- If it exists
- If it mentions the exception

What to Throw and Catch

The Standard Library includes an exception class

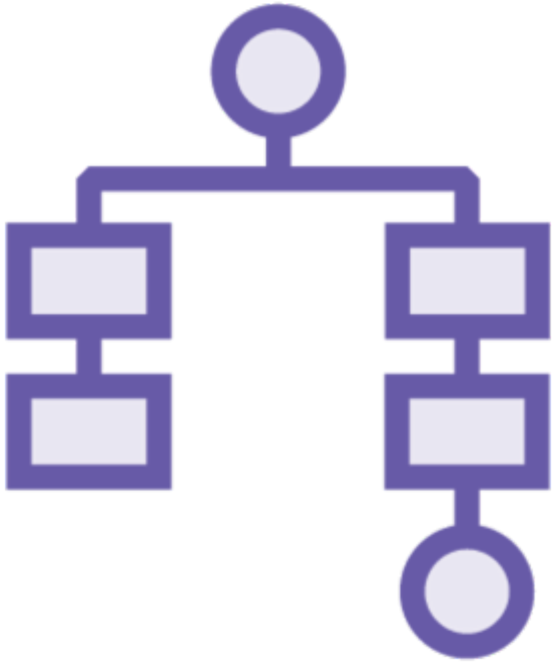
Base class to a hierarchy of exceptions

Uses classes derived from it when you need to throw

Use these exceptions yourself

Or derive your exceptions from them

std::exception



Has a member function: what() - returns a string

Has a number of derived classes

- logic_error
- runtime_error

These are “marker classes”

Unwinding the Stack

When an exception is thrown

If in a try, everything local to try block goes out of scope

- Destructors go off
- Control goes to the catch

If not, everything local to the function goes out of scope

- control returns to where that function was called from
- Recurse to “if in a try” above

If you get all the way out of main(), the user gets a dialog

- But it’s more interesting when you end up in a catch

RAII Revisited

No RAII

```
try
{
   auto x = new X(Stuff);
   //risky stuff
   delete x;
}

catch (exception& e)
{
    //react
}
```

RAII

```
try
{
  auto x = make_unique<X>(Stuff);
  //risky stuff
}

catch (exception& e)
{
    //react
}
```

Exceptions Have a Cost

Little or no cost to set up a try/catch if the exception is not thrown

If it’s thrown, time is used up (much more than an if)

Don’t use for data entry validation (eg Feb 30th)

Exceptions Have a Cost

More useful with deep calling hierarchy

- A calls b calls c calls d calls e….
- Each must test return value, prevent further calculations if something went wrong
- That can take time too

Using exceptions makes neater code that runs faster when everything goes well

- Best for rare occurrences like disk full, network fell down etc

You Can Mark a Function as noexcept

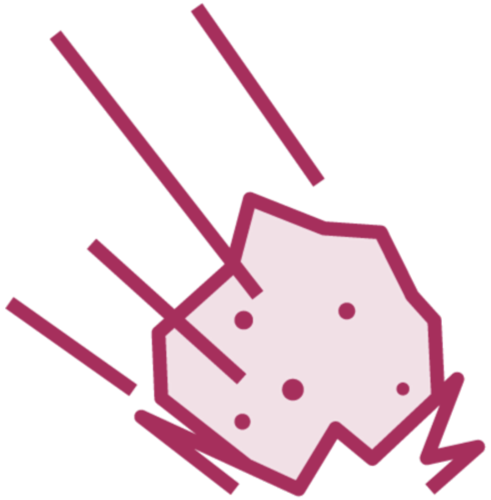
Appears to mean “won’t throw an exception”

Really means “won’t throw an exception worth catching”

Advantages: expressivity, performance

noexcept(false)

noexcept Functions That Throw?



App terminates

No stack unwinding

Enabling Moves with noexcept

If a move operation throws, the enclosing operation can't be rolled back

Some moving operations in std:: **will only call** noexcept **functions**

- Move ctor, move op=, swap

If your move operations (or things they call) are not noexcept, **you'll get a copy instead**

Mark these noexcept **if you can**

Summary

Exceptions handle unusual (exceptional) errors

- try
- throw
- catch

Between the throw and the catch, locally-scoped objects are cleaned up

- Destructors run

The std::exception class is very useful

- Most standard library code throws objects derived from it

Mark functions noexcept if they don't throw