Poznan University of Technology
Object Oriented Programming
Exceptions

Jakub Piotr Hamerliński, M.Eng.

https://www.linkedin.com/in/hamerlinski

https://github.com/hamerlinski

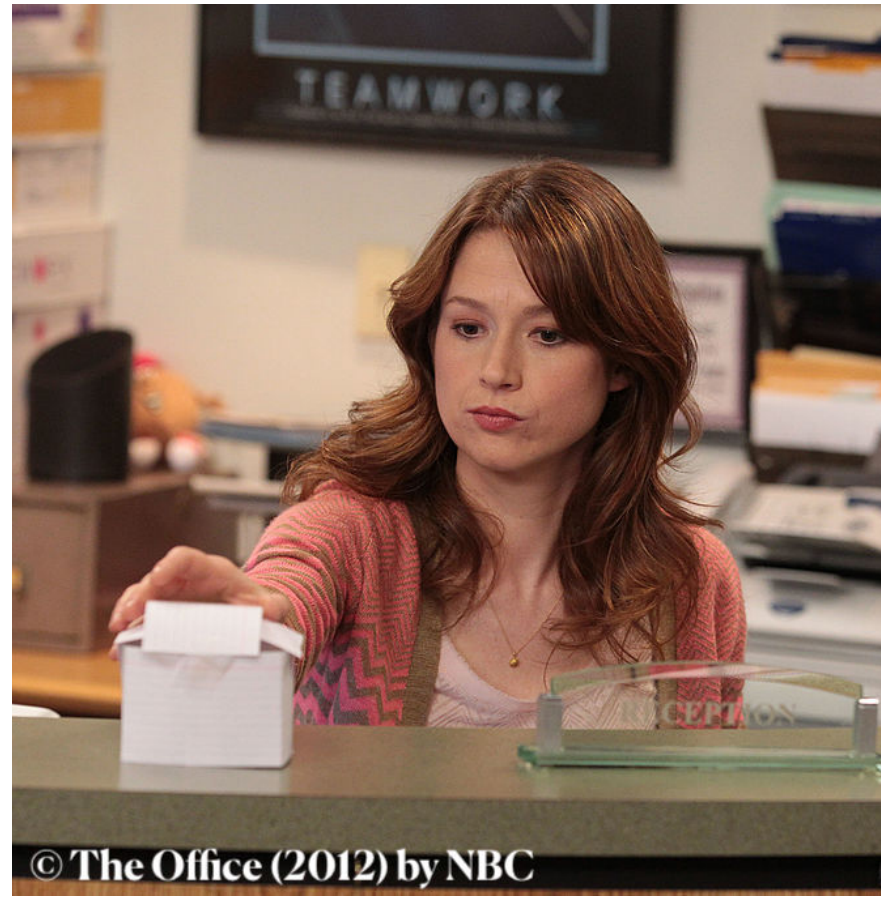
Object Oriented Programming

Agenda

01. Writing safe code
02. Exceptions
03. Standard exceptions classes
04. Task

> "Disposable cameras are fun. Although it does seem wasteful and you don't ever get to see your pictures. If it's an important even that you want to remember, I recommend using a real camera. But I don't care if I forget today..."

Erin Hannon



Writing safe code

Object Oriented Programming

Writing safe code

Creating programs resistant to errors and exceptions is an essential aspect of software development. To ensure your program can handle unexpected situations, you need to implement proper exception handling techniques.

Object Oriented Programming

Writing safe code

Here are some basic strategies for creating programs resistant to errors:

01. **Defensive programming**: Write code that anticipates potential issues and handles them gracefully. This includes validating input data, using assertions, and employing modular design principles.
02. **Code reviews**: Regularly review your code and the code of your peers to identify potential issues and areas of improvement.
03. **Testing**: Thoroughly test your code, including unit tests, integration tests, and stress tests, to identify and resolve any issues before deployment.

Object Oriented Programming

Writing safe code

04. **Logging and monitoring**: Implement robust logging and monitoring systems to track errors and exceptions, helping you identify and resolve issues quickly.
05. **Proper resource management**: Ensure that resources such as memory, file handles, and network connections are properly allocated and released to avoid resource leaks and other issues.

Exceptions

Object Oriented Programming

Exceptions

In object-oriented languages like C++, various methodologies and techniques can be used to manage exceptions.

Now let's discuss exception handling in C++:

01. **Defining and reporting exceptions**: In C++, exceptions can be defined using classes. You can create a custom exception class that inherits from the standard std::exception class, or you can use one of the predefined exception classes provided by the C++ Standard Library.

Example of a custom exception:

```
#include <stdexcept>
#include <string>
class CustomException : public std::runtime_error {
 public:
  explicit CustomException(const std::string& message)
      : std::runtime_error(message) {}
};
```

Object Oriented Programming

Exceptions

02. **Exception throwing**: When an error occurs, you can throw an exception using the `throw` keyword, followed by an instance of the exception class.

Example of throwing an exception:

```
void FunctionThatThrows() {
  // Some code...
  throw CustomException("An error occurred");
}
```

Object Oriented Programming

Exceptions

03. **Exception catching and handling**: To catch an exception, you can use the `try` and `catch` blocks. The `try` block contains the code that might throw an exception, while the `catch` block handles the exception.

Example of catching and handling an exception:

```
#include <iostream>
int main() {
  try {
    FunctionThatThrows();
  } catch (const CustomException& e) {
    std::cerr << "Caught an exception: " << e.what() << std::endl;
  }
  return 0;
}
```

Object Oriented Programming

Exceptions

04. **Exception handling applications**: Proper exception handling allows your program to continue running even when errors occur. This can help ensure your program's stability, maintainability, and reusability. Examples of exception handling applications include validating user input, handling file I/O errors, and managing network communication errors.
05. **Importance of exception handling for programming modules intended for reuse**: Exception handling is crucial for modules designed for reuse, as it allows other developers to use your code without having to worry about unexpected errors. By providing clear and informative exceptions, you can help other developers understand the possible issues they might encounter and how to handle them properly.

Object Oriented Programming

Exceptions

In summary, using proper exception handling techniques in object-oriented languages like C++ helps create programs that are more robust, maintainable, and reusable. By anticipating potential issues, providing informative error messages, and handling exceptions gracefully, you can significantly improve the quality of your code.

Standard exceptions classes

Object Oriented Programming

Standard exceptions classes

In C++, the Standard Library provides a set of standard exception classes that can be used to represent common error situations. These exception classes inherit from the std::exception class, which is the base class for all standard exceptions. Some of the most common standard exceptions include std::logic_error, std::range_error, and others, which are described below:

01. `std::exception`: This is the base class for all standard C++ exception classes. It provides a virtual member function `what()` that returns a null-terminated character sequence (C-style string) describing the error.
02. `std::logic_error`: This class represents errors that result from a fault in the program's logic. These errors can potentially be detected and fixed before the program is run. `std::logic_error` is derived from `std::exception` and has several derived classes of its own:
    I. `std::domain_error`: Thrown when a function receives an argument that is outside its domain of validity.

Object Oriented Programming

Standard exceptions classes

II. `std::invalid_argument`: Thrown when a function receives an invalid argument, such as a value that does not meet the required constraints.
III. `std::length_error`: Thrown when a length error occurs, such as when an object exceeds its maximum allowable size.
IV. `std::out_of_range`: Thrown when an indexed container (e.g., `std::vector`, `std::string`) is accessed with an index that is out of bounds.

Object Oriented Programming

Standard exceptions classes

03. std::runtime_error: This class represents errors that occur during the program's execution and cannot be detected before running the program. `std::runtime_error` is derived from `std::exception` and has several derived classes:

I. `std::range_error`: Thrown when a mathematical operation produces a result that is outside the representable range of the result type (e.g., when a floating-point overflow occurs).
II. `std::overflow_error`: Thrown when an arithmetic operation results in an overflow, such as when the result of an integer operation is too large to be represented by the result type.
III. `std::underflow_error`: Thrown when an arithmetic operation results in an underflow, such as when the result of a floating-point operation is too small to be represented by the result type.
IV. `std::system_error`: Thrown when an error occurs within the operating system or another low-level API.

Object Oriented Programming

Standard exceptions classes

04. `std::bad_alloc`: Derived directly from `std::exception`, this exception is thrown when memory allocation fails, such as when the new operator cannot allocate the requested memory.
05. `std::bad_cast`: Also derived directly from `std::exception`, this exception is thrown when an invalid dynamic_cast is performed, such as when attempting to cast a base class pointer to a derived class when the pointed-to object is not actually an instance of the derived class.

These standard exceptions can be used to represent common error situations in your C++ programs. You can catch and handle these exceptions using `try` and `catch` blocks, just as you would with any other exception.

Task

Object Oriented Programming

Task

Logarithm class calculates the logarithm of a given number with a given base. The class has a constructor that takes two double arguments, representing the base and the number. The class also has a calculate method that returns the calculated logarithm.

The current implementation of the Logarithm class does not handle exceptions.
Your task is to modify the class to ensure that any possible exceptions are handled correctly.

Object Oriented Programming

Task

Here's the initial implementation of the Logarithm class:

```cpp
#include <iostream>
#include <cmath>
class Logarithm {
 public:
  Logarithm(double base, double number) {
    base_ = base;
    number_ = number;
  }
  double Calculate() const {
    return std::log(number_) / std::log(base_);
  }
 private:
  double base_;
  double number_;
};
```

Object Oriented Programming

Task

Hints:

Consider what kind of exceptions can occur when calculating logarithms.
Create custom exception classes to handle specific error cases.
Use try-catch blocks where appropriate to handle exceptions.

Thank you

Feel free to reach me via LinkedIn

Fin