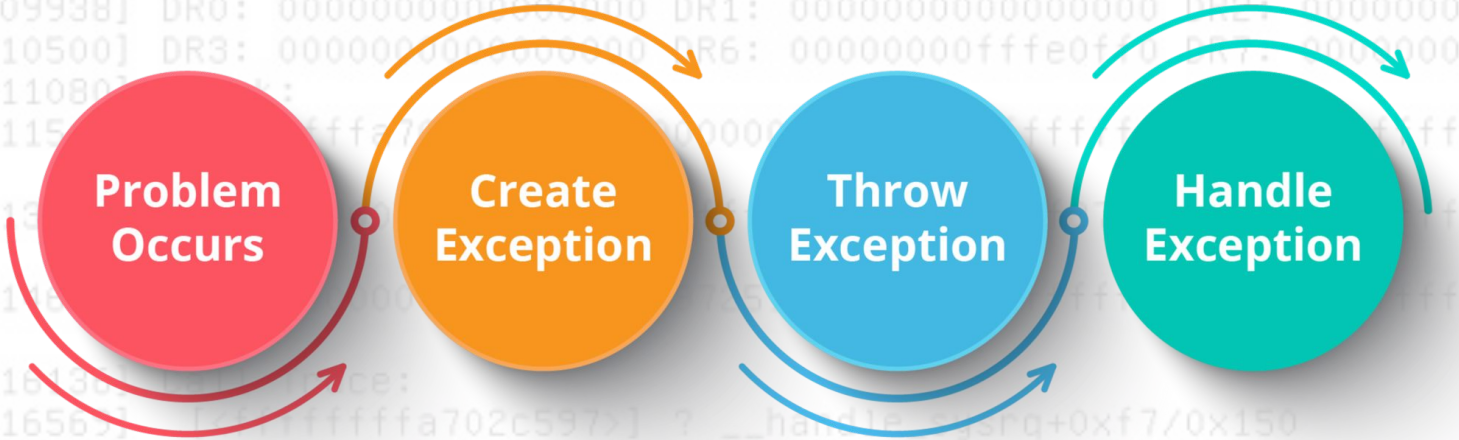

Exceptions and Error Handling

Ryan Hessler & Brandon Mitchell

Overview

- Exceptions
  - Two major ways they are used
    - Control flow
    - Handling abnormal situations
- Errors
  - Descriptive errors are necessary for efficient debugging
    - Tracebacks
    - Descriptive error names

Purpose of Exceptions

- Don’t directly help with debugging
- Capture (and handle) unexpected behaviors at runtime or during programming
  - At runtime, recover from unexpected situation
  - During programming, guard against incorrect method usages, incorrect arguments, etc.

Exceptions at the Low Level

1. Exception is thrown
2. Can it be handled?
    a. Trace back up the method call stack
    b. Search for a block of code that can handle the exception (exception handler)
3. Handle the exception
4. If no exception handler, terminate
    a. Some languages allow the uncaught exception handler to be overridden

History of Exception/Error Handling

- First occurrence of error handling
  - Lisp 1.5 (1962)
  - Errors returned a keyword instead of terminating
  - MacLisp (1972) included CATCH and THROW keywords
    - Introduced user-defined exception handling
  - In the 70s, New Implementation of Lisp (NIL)
    - Introduced UNWIND-PROTECT
      - Behaved similarly to FINALLY

Exception Handling in Different Languages

- Most languages are similar
  - try-catch-finally
  - throw
- Some languages have different syntax or ways to handle exceptions
  - Python uses except instead of catch
  - raise replaces throw in some languages
  - JavaScript
    - try-catch-finally
    - .then(), .catch(), and .finally() for exception handling with promises
  - Scheme
    - raise, raise-result-error, and similar functions
    - call-with-exception-handler and with-handlers for handling exceptions

Exception Handling in Modern Languages

- Generally, some form of try-catch-finally
- try
  - the code that might throw an exception
  - should have as little code as possible
- catch
  - the code that will handle the exception
  - can by typed to only catch certain exceptions
  - some languages allow multiple catch statements
- finally
  - will always run whether an exception occurs or not
  - meant for clean-up and freeing resources

Exception Handling in Modern Languages (Cont.)

```
try
{
     file.ReadBlock(buffer, index, buffer.Length);
}
catch (System.IO.IOException e)
{
     Console.WriteLine("Error reading from {0}. Message = {1}",
          path, e.Message);
}
finally
{
     if (file != null)
     {
          file.Close();
     }
}
```

Exception Handling in Modern Languages (Cont.)

- OOP languages have an exception hierarchy
- Custom exceptions can be created using this hierarchy
  - Done with inheritance
  - Can inherit from the most general (highest in the hierarchy) or more specific (lower)
- Polymorphism is at play!
  - Important to list most specific exceptions first and more general last

When Exceptions Should be Thrown

- Generally, when there is no better alternative
- Alternatives
  - Return a bool value to indicate success or failure
  - Return an invalid value that can be checked by the client
  - Set or return an error code
- As with many things, it varies from case to case
  - All possibilities should be considered

Exceptions as Control Flow

- This use of exceptions is generally frowned upon
  - Some languages were designed with this use in mind, like Python
- Exceptions can be used as a means of controlling the program execution sequence
  - Essentially a non-local goto
  - Where the program resumes can be difficult to find
  - Exceptions should be used for exceptional circumstances
- Trigger the debugger in some IDE's

```
try
{
     for (int i = 0; /* no test */ ; i++)
          array[i]++;
}
catch (ArrayIndexOutOfBoundsException e) {}
```

Terminating Exceptions and Uncaught Exceptions

- Some exceptions can’t be caught
  - Varies from language to language
  - Stack overflow is a common example
    - No way to properly recover
- Uncaught exceptions may terminate the program
  - May be appropriate to terminate program if no way to handle situation
    - Important to collect enough information to be useful

Questions?

Sources

Don't use exceptions for Flow Control. (1n.d.). Retrieved November 16, 2021, from http://c2.com/cgi-bin/wiki?DontUseExceptionsForFlowControl.

IBM. (2019, December 16). Which is better, a return code or an exception? IBM. Retrieved November 16, 2021, from https://www.ibm.com/support/pages/which-better-return-code-or-exception.

Microsoft. (2021, September 15). Try-catch-finally - C# reference. try-catch-finally - C# Reference | Microsoft Docs. Retrieved November 16, 2021, from https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/try-catch-finally.

Stack Exchange. (2012, August 18). Why are errors named as "exception" but not as "error" in programming languages? Software Engineering Stack Exchange. Retrieved November 16, 2021, from https://softwareengineering.stackexchange.com/questions/161488/why-are-errors-named-as-exception-but-not-as-error-in-programming-languages.

Stack Exchange. (2013, March 4). Are exceptions as control flow considered a serious antipattern? if so, why? Software Engineering Stack Exchange. Retrieved November 16, 2021, from https://softwareengineering.stackexchange.com/questions/189222/are-exceptions-as-control-flow-considered-a-serious-antipattern-if-so-why.

Wikimedia Foundation. (2021, April 23). Errno.h. Wikipedia. Retrieved November 16, 2021, from https://en.wikipedia.org/wiki/Errno.h.

Wikimedia Foundation. (2021, November 13). Defensive Programming. Wikipedia. Retrieved November 16, 2021, from https://en.wikipedia.org/wiki/Defensive_programming.

Wikimedia Foundation. (2021, November 15). Data validation. Wikipedia. Retrieved November 16, 2021, from https://en.wikipedia.org/wiki/Data_validation.

Wikimedia Foundation. (2021, October 16). Exception handling. Wikipedia. Retrieved November 16, 2021, from https://en.wikipedia.org/wiki/Exception_handling.