SIT232 - OBJECT ORIENTED DEVELOPMENT

Session 9. Exception Handling

Outline

- Session 09. Exception Handling
  - Objectives
  - Managing Errors
  - Exceptions
  - Exceptions – Catching
  - Exceptions – Throwing
  - Exceptions – Rethrowing
  - Microsoft.Net Exception Classes
  - Custom Exception Hierarchies
  - Guidelines

SESSION 9. EXCEPTION HANDLING

Objectives

- At the end of this session you should:
  - Understand the alternatives for error detection and handling;
  - Understand how exceptions work;
  - Understand exception hierarchies and be able to develop classes within them; and
  - Know when and how to apply exceptions in your programs and be able to do so.

Error Handling

- Managing errors works towards improving the **robustness** of a program
  - **Error detection**: identify when an error has occurred
  - **Error handling**: correct for that error

Error Handling

- Handling errors without exceptions usually uses:
  - Return values (success/fail)
  - Status code (failed? what happened?)
    - Example status codes for Unix ‘errno’:

Name	Code Number	Description
	0	(represents no error occurred)
ENOENT	2	No such file or directory
EIO	5	I/O error
ENOMEM	12	Out of memory
EACCESS	13	Permission denied
EINVAL	22	Invalid argument
EMFILE	24	Too many open files
ENOSPC	28	No space left on device
EROFS	30	Read-only file system
ERANGE	34	Math result not representable

Error Handling

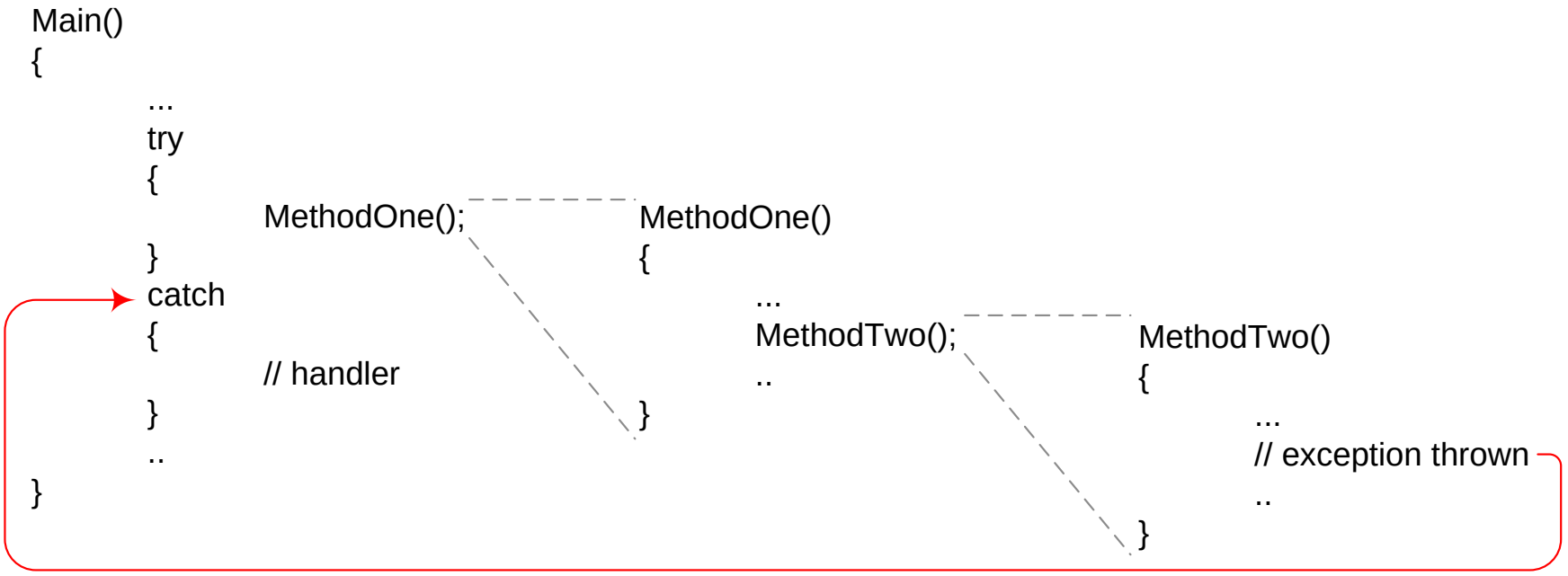
- Example C/C++ code for this structure:

```
if(open("someFile.txt", O_RDONLY) == -1)
{
        perror("someFile.txt");
        return -1;
}
```

- What happens when the error handling routine is several method calls back?
  - Pass the error message back, one method at a time
    - If you miss one, an error state has been “forgotten” (application will usually die)

Exceptions

- Exceptions are different
  - When an error is detected an object is created with information about that error
    - Object is “thrown” directly to the error handling routine



Exceptions – Catching

- We catch exceptions using a try/catch block:
  - try block contains code that may throw an exception
  - *catch blocks handle different exceptions that are thrown (zero or more)*
  - *finally block executes regardless of exception or not*
- Note: Order is important!

```
try
{
    code that may generate an exception
}[
catch(type[  variable_name])
{
    error handling code
}]
[...]
[catch
{
    error handling code
}]
[finally
{
    code always executed after the try/catch
}]
```

Exceptions – Catching

- C# simple data types have TryParse() and Parse() methods that *show alternative techniques for error handling*, e.g.,

```
Console.Write("Enter a number: ");
int value = 0;
if(int.TryParse(Console.ReadLine(), out value) == true)
    Console.WriteLine("Thank you.");
else
    Console.WriteLine("That wasn't a number!");
```

Exceptions – Catching

- C# simple data types have TryParse() and Parse() methods that show alternative techniques for error handling, e.g.,

```
Console.Write("Enter a number: ");
int value = 0;
try
{
    value = int.Parse(Console.ReadLine());
    Console.WriteLine("Thank you.");
}
catch(FormatException)
{
    Console.WriteLine("That wasn't a number!");
}
```

Exceptions – Catching

- C# simple data types have TryParse() and Parse() methods that show alternative techniques for error handling, e.g.,

```
Console.Write("Enter a number: ");
int value = 0;
try
{
    value = int.Parse(Console.ReadLine());
    Console.WriteLine("Thank you.");
}
catch (FormatException fe)
{
    Console.WriteLine(fe.Message);
}
```

Exceptions – Throwing

- Exceptions are thrown when an error is detected

```
throw new type();
throw new type(message);
throw new type(message, inner_exception);
```

Exceptions – Rethrowing

- Sometimes you also need to "re-throw" an exception you have caught:
  - Exception partially handled, changing the ***exceptional state***
  - Need to ***tidy up/free resources*** that otherwise wouldn't be
  - ***Encapsulating an exception thrown*** by a class that *is the target of a delegation*
- Two options:
  - *Create a new exception as per previous slide*
  - *Re-throw the caught exception:* `throw;`

Microsoft.Net Exception Classes

- Microsoft.Net provides many different exception types that should be used where appropriate:
  - **AccessViolationException** – attempt to read or write protected memory;
  - **ArgumentException** – one or more arguments were invalid;
  - **DivideByZeroException** – attempt made to divide by zero;
  - **FileNotFoundException** – specified file does not exist;
  - **IndexOutOfRangeException** – array index is out of range;
  - **InvalidCastException** – a data type casting is not valid (usually because the types are unrelated);

Microsoft.Net Exception Classes

- Microsoft.Net provides many different exception types that should be used where appropriate (cont.):
  - **InvalidOperationException** – a method call is (currently) invalid;
  - **NotImplementedException** – method is not yet implemented;
  - **NotSupportedException** – the functionality defined by a method is not supported for that particular object, e.g., invalid reading/writing to a file;
  - **NullReferenceException** – when an attempt is made to access an attribute/operation of an object when the reference is set to null;
  - **OutOfMemoryException** – the system has run out of memory;

Microsoft.Net Exception Classes

- Microsoft.Net provides many different exception types that should be used where appropriate (cont.):
  - **OverflowException** – converting a value, such as with the Convert object, results in a loss of data, e.g., attempting to convert the value 123456 to a byte (which has range 0-255);
  - **RankException** – attempt to access a dimension of an array that does not exist;
  - **StackOverflowExceptio**n – the call stack cannot grow any larger;
  - **UnauthorizedAccessException** – permission denied;

Custom Exception Hierarchies

- The *provided exception classes will often not be adequate* – this is not unusual or a fault
  - Can create our own exception classes instead

- Rules:
  - **Create a new class with the suffix Exception in the name**, i.e., _____Exception;
  - The class *must be derived from the Exception class*;
  - A **minimum of three constructors** must be provided:
    - Parameter-less constructor;
    - Constructor with a single parameter: string message;
    - Constructor with two parameters: string message, Exception inner;

Custom Exception Hierarchies

- Example:

```
class InvalidBirthdateException : Exception
{
    public InvalidBirthdateException()
    {
    }

    public InvalidBirthdateException(string message)
        : base(message)
    {
    }

    public InvalidBirthdateException(string message, Exception inner)
        : base(message, inner)
    {
    }
}
```

Guidelines

- From McConnell, S., “Code Complete”, Second Edition, Microsoft Press, 2004:
  - Use *exceptions to notify other parts of the program about errors* that should not be ignored
  - *Throw an exception only for conditions* that are truly exceptional
  - *Don’t use an exception to pass the buck*
  - *Avoid throwing exceptions in constructors and destructors* unless you catch them in the same place

Guidelines

- From McConnell, S., “Code Complete”, Second Edition, Microsoft Press, 2004 (cont.):
  - *Throw exceptions at the right level of abstraction*
  - *Include in the exception message* all the information that led to the exception
  - *Avoid empty catch blocks*
  - *Know the exceptions your library code throws*
  - Consider *building a centralized exception* reporter
  - *Standardize your project’s use of excep*tions
  - Consider *alternatives to exceptions*

Summary

- Session 09. Exception Handling
  - Objectives
  - Managing Errors
  - Exceptions
  - Exceptions – Catching
  - Exceptions – Throwing
  - Exceptions – Rethrowing
  - Microsoft.Net Exception Classes
  - Custom Exception Hierarchies
  - Guidelines

Testing

Which of the following guide lines for using exceptions is correct?

a) Throw an exception only for conditions that are truly exceptional

b) Don't use an exception to pass the buck

c) Throw exceptions at the right level of abstraction

d) All of the above

e) None of the above

Testing

What does the following statement do?

```
throw;
```

a) It causes the program to immediately terminate

b) It throws an exception that has previously been caught

c) It causes the debugger to be invoked

d) It indicates to the programmer that a method has not yet been implemented

e) It presents an error message to the user

Testing

Which of the following is NOT an exception type defined by Microsoft.Net?

a) ArgumentException

b) BadDataException

c) DivideByZeroException

d) InvalidOperationException

e) OverflowException

Testing

Which of the following is NOT a common signature for an exception constructor?

a) new type(message)

b) new type(message, inner_exception, line_number)

c) new type()

d) new type(message, inner_exception)

e) None of the above

Summary

- Training Videos:
  - C#: Exceptions