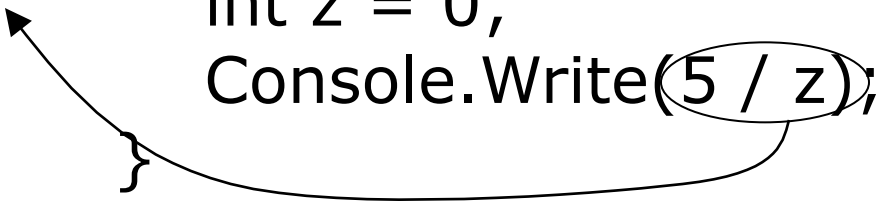
Exceptions

Overview

- An exception is an unusual circumstance, such as an error condition, that must be handled in a non-standard way.
- C# and many other languages (including Java), exceptions serve as a standard mechanism for handling execution error conditions.
- Provides for handling unusual cases at the appropriate level(s).
  - For example, an airplane could have a stuck wing flap detected in a very low-level method monitoring a wing flap's position.
  - No ability to communicate information directly to the pilot.
  - A wing flap exception can be reported upward to the higher levels that call the low-level method where the exception is detected.
  - The higher level layers, with broader status information, could automatically adjust other control surfaces and report the wing flap exception to the pilot.

A Little Demo

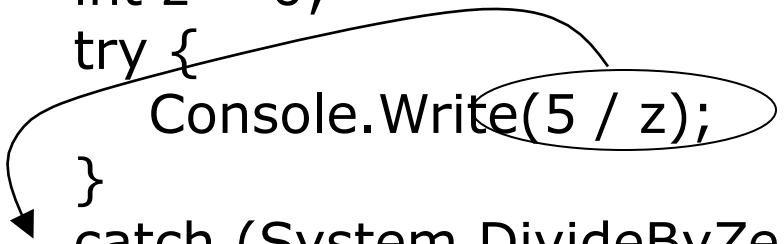
```
public class Test  {
    public static void Main()  {
        int z = 0;
        Console.Write(5 / z);
    }
}
```



Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
at Test.Main() in Program.cs:line 6

Exception Example

```
public class Test {
    public static void Main() {
        int z = 0;
        try {
            Console.Write(5 / z);
        }
        catch (System.DivideByZeroException e) {
            System.Console.WriteLine("You're dividing by zero!");
        }
    }
}
```



This will catch and handle any **`DivideByZeroException`**.
Other exceptions will still get the language system's default behavior.

You're dividing by zero!

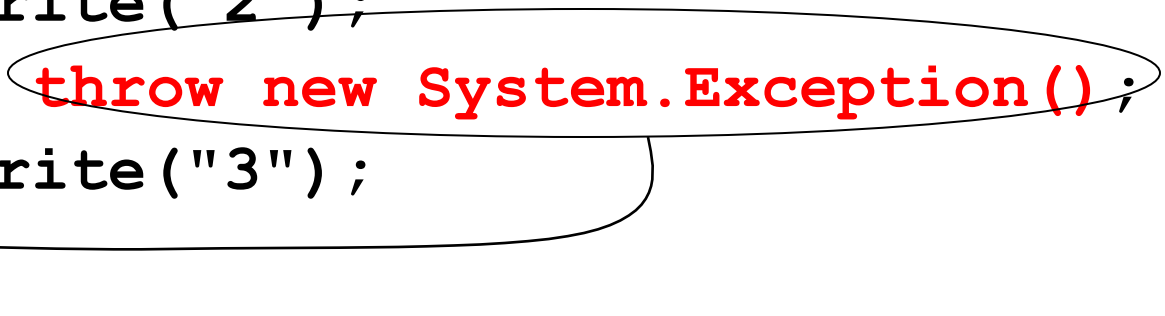
Exception Mechanism

The exception mechanism consists of three parts:

1. *definition* of the exception class
2. *throwing* the exception when exceptional condition is detected
3. *catching* or handling the exception (*try* and *catch*).

Exercise 1A - List line numbers executed and output

```
1.  Console.Write("1");
2.  try {
3.   Console.Write("2");
4.   if (true) throw new System.Exception();
5.   Console.Write("3");
6.  }
7.  catch (System.Exception e) {
8.   Console.Write("4");
9.  }
10. finally {
11.  Console.Write("5");
12. }
13. Console.Write("6");
```



Exercise 1B - List line numbers executed and output

```
1. using System;
2. public class Exercise1 {
3.  static void f1() {
4.    Console.Write("1");
5.    try {
6.      Console.Write("2");
7.      f2();
8.      Console.Write("3");
9.    }
10.   catch (Exception e) {
        Console.Write("4"); }
11.   finally { Console.Write("5"); }
12.   Console.Writeln("6");
13. }
14. static void f2 () {
15.   if (true) throw new Exception();
16. }
17. public static void Main() { f1(); }
18.}
```

Exercise 1C - List line numbers executed and output

```
1. using System;
2. public class Exercise1 {
3.  static void f1() {
4.    Console.Write("1");
5.    try {
6.      Console.Write("2");
7.      f2();
8.      Console.Write("3");
9.    }
10.   catch (Exception e) { Console.Write("4"); throw e; }
11.   finally {  Console.Write("5"); }
12.   Console.Writeln("6");
13. }
14. static void f2 () {
15.   if (true) throw new Exception();
16. }

17. public static void Main() {
18.       f1();
19. }
20.}
```

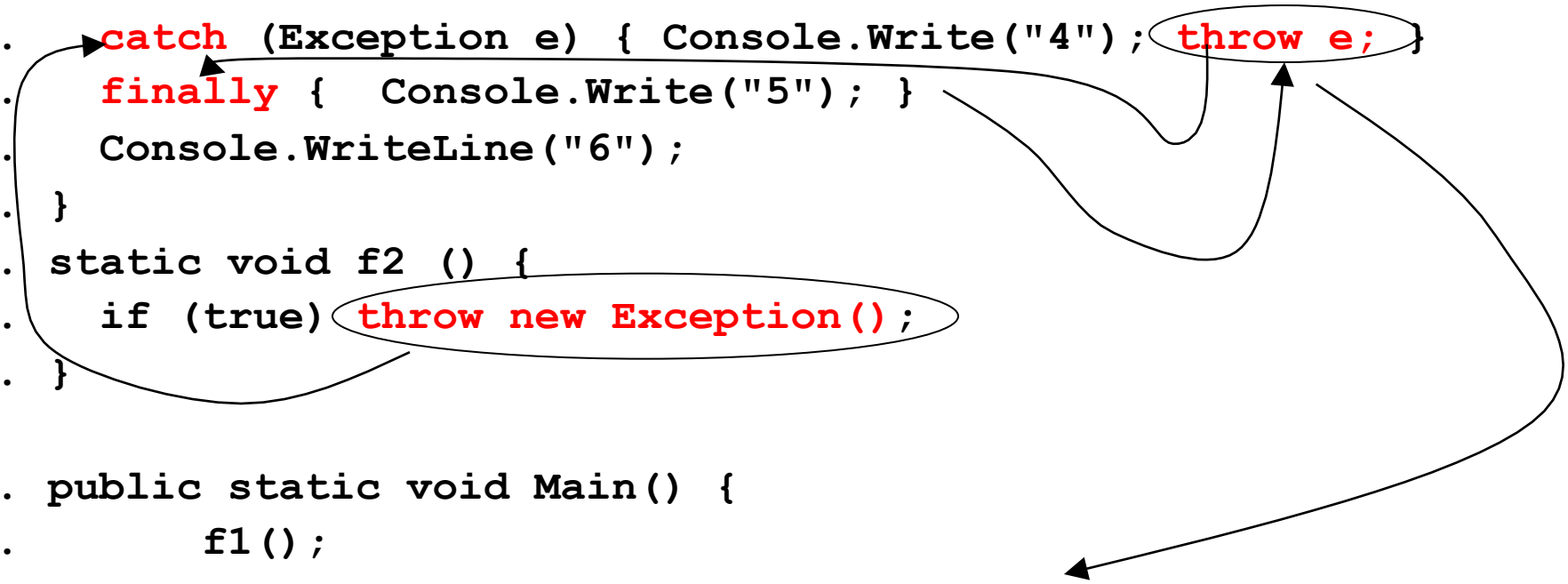
Exercise 1C - List line numbers executed and output

```
1. using System;
2. public class Exercise1 {
3.   static void f1() {
4.     Console.Write("1");
5.     try {
6.       Console.Write("2");
7.       f2();
8.       Console.Write("3");
9.     }
10.    catch (Exception e) { Console.Write("4"); throw e; }
11.    finally {  Console.Write("5"); }
12.    Console.WriteLine("6");
13.  }
14.  static void f2 () {
15.    if (true) throw new Exception();
16.  }

17.  public static void Main() {
18.        f1();
19.  }
20.}
```

17,18,3-7,14,15, 10,11,19

124
Unhandled Exception: System.Exception:
Exception of type 'System.Exception' was thrown.
at Exercise1.f1() in Program.cs:line 10
at Exercise1.Main() in Program.cs:line 20
5



Define

- Generally, an exception *class* inherits from the parent class ***Exception***.
- The exception class definition minimally defines a class constructor but may also provide methods for analysis of the conditions causing the exception.
- The following is a typical class definition for ***counterException*** with a constructor that copies a string message ***complaint*** presumably detailing the cause of the exception.
- An example of the constructor's use illustrates a new counterException created with the complaint that ***count failed***.

Define and Use Example

Define

```
class counterException : Exception {
 String complaint;
 public counterException(String complaint){
    this.complaint = complaint;
 }
 public override String ToString(){
    return "counter Exception " + complaint;
 }
}
```

Use

```
new counterException("count failed.");
```

Throw

- When an error condition is detected, it can be handled immediately where detected or can *throw* the exception to the *calling* method.
- Exception handling is a mechanism for transferring control from where an error occurred to where it can be handled most appropriately.
- After the exception is thrown, the throwing method terminates and execution control is immediately passed backward along the *chain of callers* from where the exception was thrown.
- Any method along the calling chain can:
  a) handle the exception and continue with execution,
  b) handle the exception *and* again throw the exception to the calling method to handle
  c) or do nothing, terminate and let the calling method deal with the exception.
- The d*own* method below is an example of throwing an exception, in this case when the counter *n* becomes negative.

Throw Example

```
public int down() {
  if (n <= 0)
     throw new counterException(
          n + " count Down failed.");
  return --n;
}
```

•The d*own* method is an example of throwing an exception, in this case when the counter *n* becomes negative.

•If exception is thrown, execution does not reach

```
return --n;
```

Catch

- Handling an exception consists of *try*ing a method that throws an exception and *catch*ing a thrown exception.
- Catching an exception prevents the exception from being passed to a higher level calling method.
- If an exception is not caught, the method terminates immediately and control is passed to the higher level calling method so that it might catch the exception.
- Consider the following example which causes a *counterException* to be thrown by method d*own* when the counter becomes negative.
  - The *try* and *catch* go together to define the method that throws the exception, d*own*, and the exception to be caught, *counterException*.
  - Only if an exception is thrown will it be caught and the *catch* statement(s) executed.
  - Any statements following the catch would be executed as normal.

Catch Example

Catch

```
try {
      aCounter.down( );
}
catch (counterException ce) {
      Console.Writeln("" + ce);
}
```

Throw

```
public int down() {
  if (n <= 0)
      throw new counterException(
            n + " count Down failed.");
  return --n;
}
```

Throwing exceptions to caller

- C# does not have checked exceptions, where the complier can verify the caller handles any thrown exceptions (see Java-style example).
- Checked exceptions force the programmer to handle an exception.
- When an exception is not handled in a method it is thrown to the higher level calling method.
- The earlier *down* method not handle or *catch* the *counterException* thrown so the exception is thrown back to the calling method.
- The direct effect of d*own* throwing an exception is to terminate its execution, passing execution to the *catch,* so that the remaining statements of *down* (i.e. return --*n)* are not executed.

```
public int down throws counterException () {
   if (n <= 0)
      throw new counterException(
            n + " count Down failed.");
   return --n;
}
```

Finally

- try – Invokes a method that throws an exception.
- catch – Point of execution of exception throw.
- finally - Always executed at end of a *try* block.

```
   using System;
1. class counterException : Exception {          // Define
2.  String complaint;
3.   public counterException(String c)  {
4.     this.complaint = c;
5.   }
6.   public override String ToString( ) {
7.     return "counter Exception “ + complaint;
8.   }
9. }
10. class counter {
11.   int n = 0;
12.   public int zero() { return n=0; }
13.   public int up() { return ++n; }
14.   public int down(){                         // Throw
15.     if (n <= 0)
16.        throw new counterException
17.          (n + " count Down failed.");
18.     return --n;
19.   }
20.}
```

```
21.public class Example {
22. public static void Main() {
23.   counter aCounter = new counter( );
24.   aCounter.zero( );
25.   aCounter.up();
26.   try {  aCounter.down( ); }
27.   catch (counterException ce) {   // Catch
28.         Console.Write(ce);
29.   }
30.   try {  aCounter.down( ); }
31.   catch (counterException ce) {   // Catch
32.         Console.Write(ce);
33.   }
34.   finally  {
35.        Console.Write("Finally");
36.   }
37. }
38.}
```

Exercise 2

```
1.  class counterException : Exception {
2.    String complaint;
3.    public counterException(String c){ this.complaint = c; }
4.    public override String ToString( ) { return "counter Exception " +
                                                          complaint; }
5.  }

6.  class counter {
7.    int n = 0;
8.    public int zero() { return n=0; }
9.    public int up() { return ++n; }
10.   public int down() {
11.     if (n <= 0) throw new counterException (n + " count Down failed.");
12.     return --n;
13.   }
14. }

29. public class Example {
30.  public static void Main( ) {
31.   counter aCounter = new counter( );
32.   aCounter.zero( );
33.   aCounter.up();
34.   try {  aCounter.down( ); }
35.   catch (counterException ce) { Console.Write(ce); }
36.   try {  aCounter.down( ); }
37.   catch (counterException ce) { Console.Write(ce); }
38.   finally  { Console.Write("Finally"); }
39.  }
40. }
```

1. List the sequence of line numbers executed and output.

2. The *catch* is defined in *main* method but is executed by the *throw*. Is the *catch* visible to the *throw?*
3. What does the *throw* and *catch* resemble?

4. List the sequence of line numbers executed and output.
5. What occurs at Line 21?

```
1.  class counterException : Exception {
2.    String complaint;
3.    public counterException(String c){ this.complaint = c; }
4.    public override String ToString( ) { return "counter Exception “ +
                                                          complaint; }
5.  }

6.  class counter {
7.    int n = 0;
8.    public int zero() { return n=0; }
9.    public int up() { return ++n; }
10.   public int down() {
11.     if (n <= 0) throw new counterException (n + " count Down failed.");
12.     return --n;
13.   }
14. }

29. public class Example {
30.  public static void Main( ) {
31.   counter aCounter = new counter( );
32.   aCounter.zero( );
33.   aCounter.up();
34.   aCounter.down( );
35.   aCounter.down( );
36.   Console.Write(“Completed”);
37.  }
38. }
```