Chapter 11 – Exception Handling

Exception-Handling Overview

- Uses of exception handling
  - It is a mechanism to process exceptions from program components
  - Handle exceptions in a uniform manner
  - With it, we can remove error-handling code from "main stream line" of execution
- Code that might generate errors are put in try blocks
  - Code for error handling are put in a catch clause
  - The finally clause always executes
    - The finally clause can be used to process the uncaught errors
- throws clause written in a method definition specifies exceptions the method will throw out

Introduction

- Exception: Indication of problem during execution
  - E.g., divide by zero
- Exception handling
  - Goal:
    - Create application that can handle or resolve exception
    - Enable clear, robust and more fault-tolerant programs
- Keywords
  - Try
    - Include codes in which exceptions might occur
  - Catch
    - Represent types of exceptions the catch can handle
  - Finally
    - codes present here will always execute

11.3 Example: DivideByZeroException

- Error catching
  - Method Convert.ToInt32 will automatically detect for invalid representation of an integer
    - Method generates a FormatException
  - CLR automatic detection for division by zero
    - Occurrence will cause a DivideByZeroException

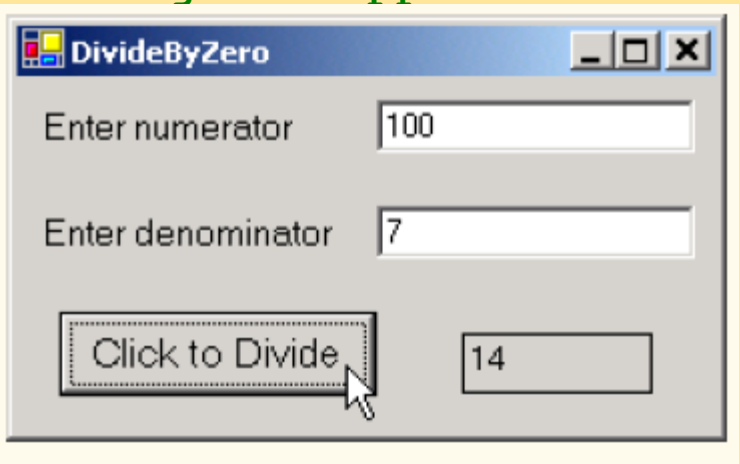
15.2 Exception-Handling Overview

- Termination model of exception handling
  - The block in which the exception occurs expires

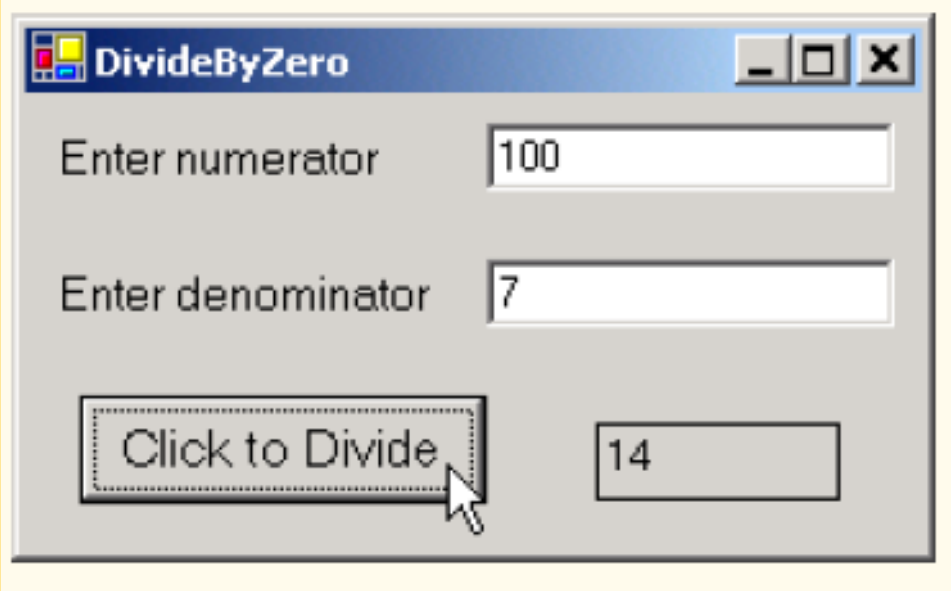
15.3 Exception-Handling Example: Divide by Zero

- Common programming mistake
- Throws ArithmeticException

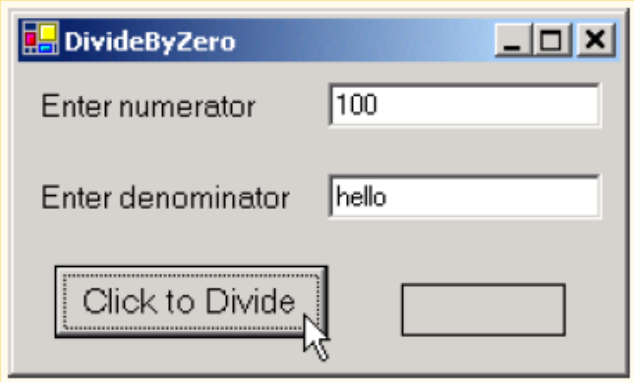
```
3     using System;
4     Using System.Drawing;
5     using System.Collections;
6     using System.ComponentModel;
7     using System.Windows.Forms;
8     using System.Data;
12    public class DivideByZeroTest : System.Windows.Forms.Form{
14       private System.Windows.Forms.Label numeratorLabel;
15       private System.Windows.Forms.TextBox numeratorTextBox;
16       private System.Windows.Forms.Label denominatorLabel;
17       private System.Windows.Forms.TextBox denominatorTextBox;
18       private System.Windows.Forms.Button divideButton;
19       private System.Windows.Forms.Label outputLabel;
20       // required designer variable
21       private System.ComponentModel.Container components = null;
23       public DivideByZeroTest() {
25          // required for Windows Form Designer support
26          InitializeComponent();
27       }
```

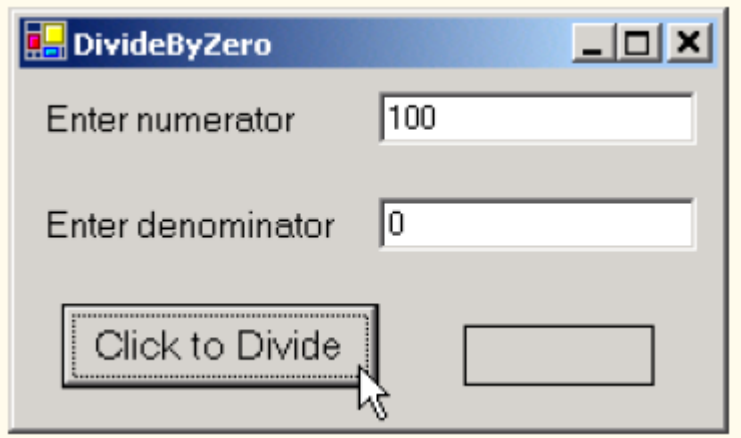


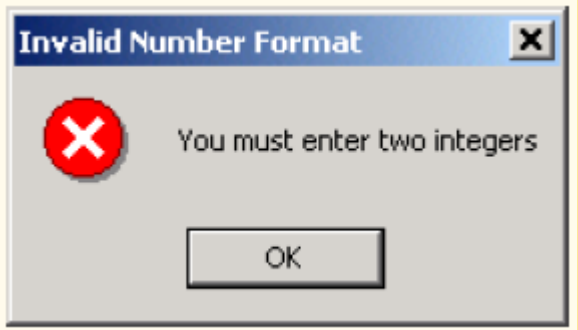
```
29      [STAThread]
30      static void Main() {
32         Application.Run( new DivideByZeroTest() );
33      }
37      private void divideButton_Click(object sender,
                                System.EventArgs e){
40         outputLabel.Text = "";
42         try    {
46           int numerator=Convert.ToInt32(numeratorTextBox.Text);
47           int denominator =
                 Convert.ToInt32(denominatorTextBox.Text );
51            int result = numerator / denominator;
52            outputLabel.Text = result.ToString();
53         } // end try
```

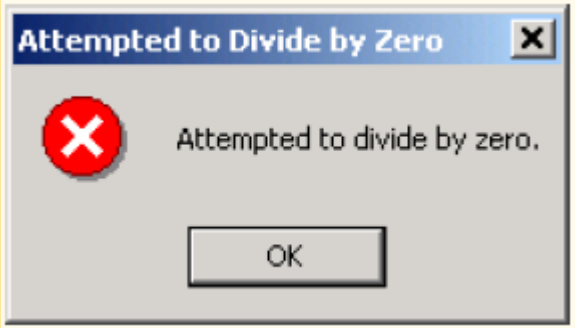


```
55          catch (FormatException ) {
57              MessageBox.Show( "You must enter two integers",
58                  "Invalid Number Format",
59                  MessageBoxButtons.OK, MessageBoxIcon.Error );
60          }
62          catch (DivideByZeroException divideByZeroException){
64              MessageBox.Show( divideByZeroException.Message,
65                  "Attempted to Divide by Zero",
66                  MessageBoxButtons.OK, MessageBoxIcon.Error );
67          }
68      } // end method divideButton_Click
69  } // end class DivideByZeroTest
```









11.2 Exception Handling Overview

- Chained exceptions
  - Rethrow exception to its caller (or loop) if catch cannot handle it
  - Uncaught exceptions thrown outer of the caller (or loop) until the main()
- Catch type
  - Must be of class Exception or one that extends it directly or indirectly

11.5 Finally Block

- Finally block
  - Ideal for placing resource deallocation code
  - Execute immediately after catch handler or try block
  - Must be present if no catch block is present
  - But it is optional if more than one or more catch handler exist

11.5 Finally Block

- The Throw expression
  - An exception object
  - Must be of either class Exception or one of its derived class
  - Customize the exception type thrown from methods

```
3     using System;
5     class UsingExceptions  {
8        static void Main( string[] args ){
11        Console.WriteLine( "Calling DoesNotThrowException" );
12        DoesNotThrowException();

15        Console.WriteLine("\nCallingThrow ExceptionWithCatch");
16        ThrowExceptionWithCatch();

19        Console.WriteLine"\nCalling ThrowExceptionWithoutCatch");
21          // call ThrowExceptionWithoutCatch
22        try {
24             ThrowExceptionWithoutCatch();
25          }
28        catch  {
30             Console.WriteLine( "Caught exception from " +
31                "ThrowExceptionWithoutCatch in Main" );
32          }
```

```
35      Console.WriteLine("\nCalling ThrowExceptionCatchRethrow");
38      try {
40         ThrowExceptionCatchRethrow();
41       }
44      catch  {
46         Console.WriteLine( "Caught exception from " +
47                "ThrowExceptionCatchRethrow in Main" );
48          }
49       } // end method Main
```

```
51      public static void DoesNotThrowException() {
54         try {
56            Console.WriteLine( "In DoesNotThrowException" );
57         }
59         catch {
61            Console.WriteLine( "This catch never executes" );
62         }
64         finally {
66            Console.WriteLine(
67               "Finally executed in DoesNotThrowException" );
68         }
69         Console.WriteLine( "End of DoesNotThrowException" );
70      }
```

```
Calling DoesNotThrowException
In DoesNotThrowException
Finally executed in DoesNotThrowException
End of DoesNotThrowException
```

```
72      public static void ThrowExceptionWithCatch() {
75         try {
77            Console.WriteLine( "In ThrowExceptionWithCatch" );
78            throw new Exception("Exception in
                                   ThrowExceptionWithCatch" );
80         }
82         catch ( Exception error ) {
84            Console.WriteLine( "Message: " + error.Message );
85         }
87         finally {
89            Console.WriteLine(
90               "Finally executed in ThrowExceptionWithCatch" );
91         }
92         Console.WriteLine( "End of ThrowExceptionWithCatch" );
93      } // end method ThrowExceptionWithCatch
```

```
Calling ThrowExceptionWithCatch
In ThrowExceptionWithCatch
Message: Exception in ThrowExceptionWithCatch
Finally executed in ThrowExceptionWithCatch
End of ThrowExceptionWithCatch
```

```
95      public static void ThrowExceptionWithoutCatch() {
98         try {
100           Console.WriteLine("In ThrowExceptionWithoutCatch" );
101           throw new Exception(
102              "Exception in ThrowExceptionWithoutCatch" );
103        }
105        finally {
107           Console.WriteLine( "Finally executed in " +
108              "ThrowExceptionWithoutCatch" );
109        }
111        Console.WriteLine( "This will never be printed" );
112     }
```

```
Calling ThrowExceptionWithoutCatch
In ThrowExceptionWithoutCatch
Finally executed in ThrowExceptionWithoutCatch
Caught exception from ThrowExceptionWithoutCatch in Main
```

```
114     public static void ThrowExceptionCatchRethrow() {
117        try {
119           Console.WriteLine("In ThrowExceptionCatchRethrow" );
120           throw new Exception(
121              "Exception in ThrowExceptionCatchRethrow" );
122        }
124        catch ( Exception error ) {
126           Console.WriteLine( "Message: " + error.Message );
128           throw error;
130        }
132        finally {
134           Console.WriteLine( "Finally executed in " +
135              "ThrowExceptionCatchRethrow" );
136        }
138        Console.WriteLine( "This will never be printed" );
139     } // end method ThrowExceptionCatchRethrow
140  } // end class UsingExceptions
```

```
Calling ThrowExceptionCatchRethrow
In ThrowExceptionCatchRethrow
Message: Exception in ThrowExceptionCatchRethrow
Finally executed in ThrowExceptionCatchRethrow
Caught exception from ThrowExceptionCatchRethrow in Main
```

11.4 .NET Exception Hierarchy

- .Net Framework
  - Class Exception is base class
  - Derived class:
    - ApplicationException
      - Programmer use to create data types specific to their application
      - Reduce the chance of program stopping execution
    - SystemException
      - CLR can generate at any point during execution
      - Runtime exceptin
      - Example: IndexOutOfRangeException

11.6 Exception Properties

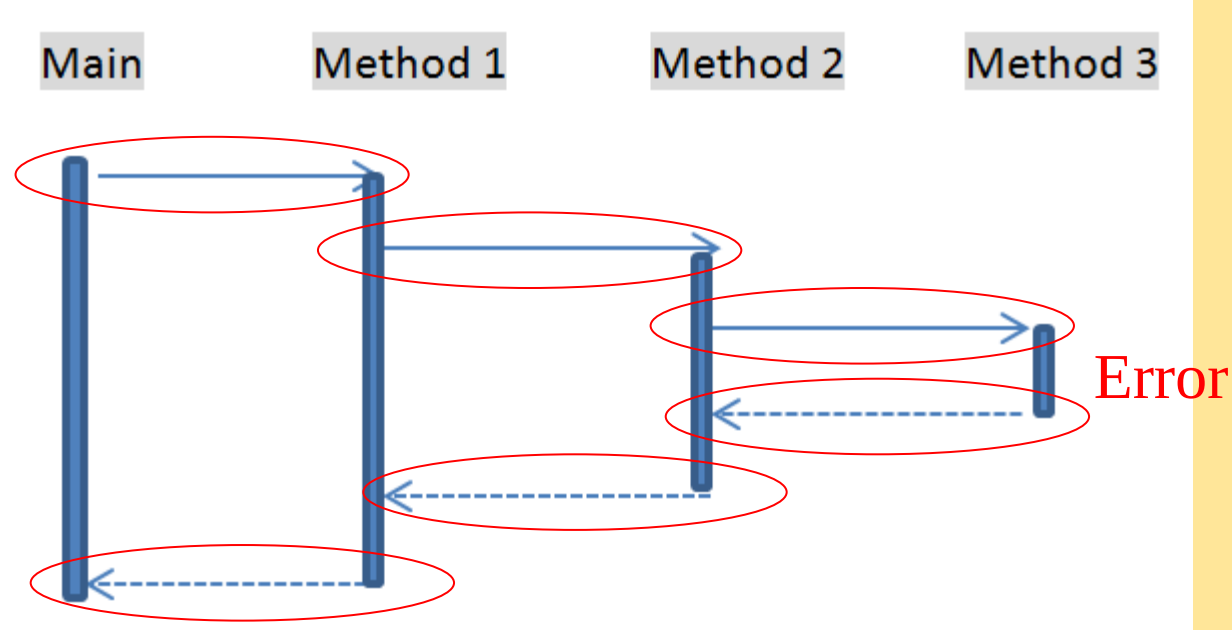
- Properties for a caught exception
  - Message
    - Stores the error message associated with an Exception object
      - May be a default message or customized
  - StackTrace
    - Contain a string that represents the method call stack
    - Represent sequential list of methods that were not fully processed when the exception occurred
    - The exact location is called the throw point

11.6 Exception Properties

- InnerException property
  - "Wrap" exception objects caught in code
    - Then throw new exception types

```
3     using System;
6     class Properties {
8        static void Main( string[] args ) {
12          try {
14             Method1();
15          }
19          catch ( Exception exception ){
21             Console.WriteLine("exception.ToString(): \n{0}\n",
23                exception.ToString() );
24             Console.WriteLine( "exception.Message: \n{0}\n",
25                exception.Message );
26             Console.WriteLine( "exception.StackTrace: \n{0}\n",
27                exception.StackTrace );
28             Console.WriteLine("exception.InnerException: \n{0}",
30                exception.InnerException );
31          }
32       }
```

```
34      public static void Method1() {
36          Method2();
37      }
39      public static void Method2() {
41          Method3();
42      }
44      public static void Method3() {
47          try {
49              Convert.ToInt32( "Not an integer" );
50          }
52          catch ( FormatException error ) {
54              throw new Exception(
55                  "Exception occurred in Method3", error );
56          }
57      }
58  }
```



```
exception.ToString():
System.Exception: Exception occurred in Method3 --->
   System.FormatException: Input string was not in a correct format.
   at System.Number.ParseInt32(String s, NumberStyles style,
      NumberFormatInfo info)
   at System.Convert.ToInt32(String s)
   at Properties.Method3() in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 60
   --- End of inner exception stack trace ---
   at Properties.Method3() in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 66
   at Properties.Method2() in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 51
   at Properties.Method1() in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 45
   at Properties.Main(String[] args) in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 16
```

© 2002 Prentice Hall.
All rights reserved.

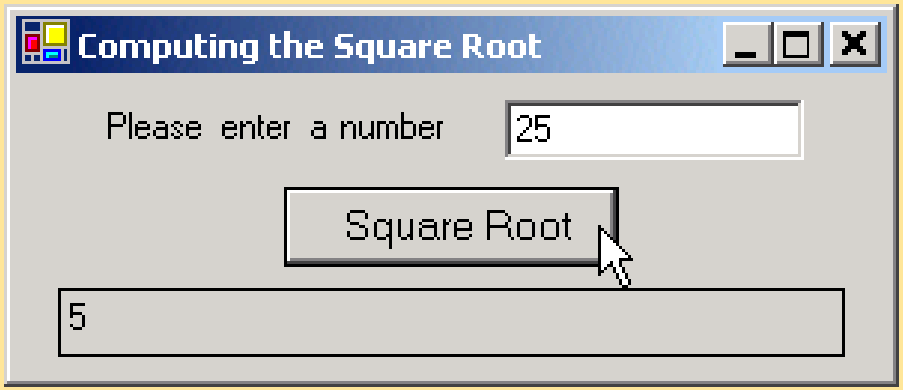
```
exception.Message:
Exception occurred in Method3

exception.StackTrace:
   at Properties.Method3() in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 66
   at Properties.Method2() in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 51
   at Properties.Method1() in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 45
   at Properties.Main(String[] args) in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 16

exception.InnerException:
System.FormatException: Input string was not in a correct format.
   at System.Number.ParseInt32(String s, NumberStyles style,
      NumberFormatInfo info)
   at System.Convert.ToInt32(String s)
   at Properties.Method3() in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_8\
         properties\properties.cs:line 60
```

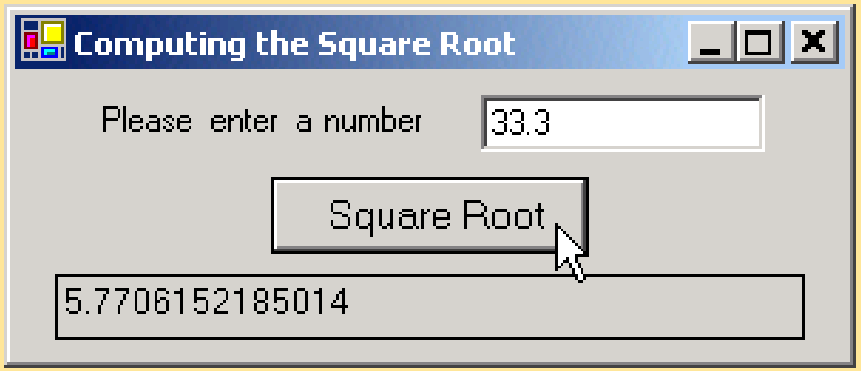
11.7 Programmer-Defined Exception Classes

- Creating customized exception types (class)
  - The class is derived from class ApplicationException
  - Should end with "Exception"
  - It defines three constructors
    - A default constructor
    - A constructor that receives a string argument
    - A constructor that takes a string argument and an Exception argument

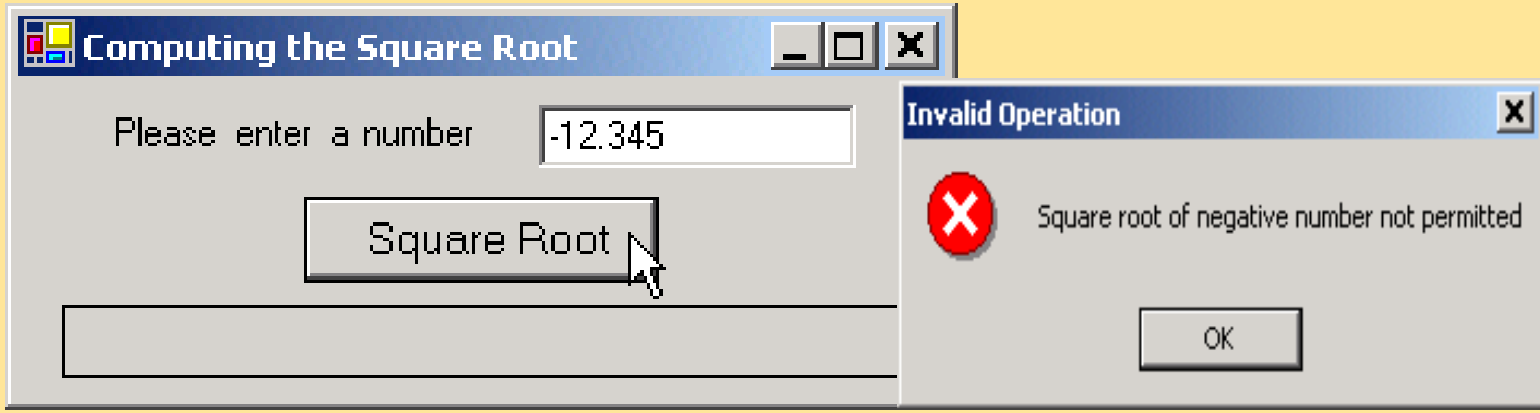
```
4    using System;
7    class NegativeNumberException : ApplicationException{
10       public NegativeNumberException()
             : base( "Illegal operation for a negative number" ) {
13       }
15       public NegativeNumberException( string message )
             : base( message ) {
18       }
21       public NegativeNumberException(
           string message, Exception inner):base(message, inner ){
25       }
26   } // end class NegativeNumberException
```

```
3     using System;
4     using System.Drawing;
5     using System.Collections;
6     using System.ComponentModel;
7     using System.Windows.Forms;
8     using System.Data;
10    public class SquareRootTest : System.Windows.Forms.Form {
12       private System.Windows.Forms.Label inputLabel;
13       private System.Windows.Forms.TextBox inputTextBox;
14       private System.Windows.Forms.Button squareRootButton;
15       private System.Windows.Forms.Label outputLabel;
17       private System.ComponentModel.Container components = null;
19       public SquareRootTest() {
22          InitializeComponent();
23       }
24       // Visual Studio .NET generated code
```

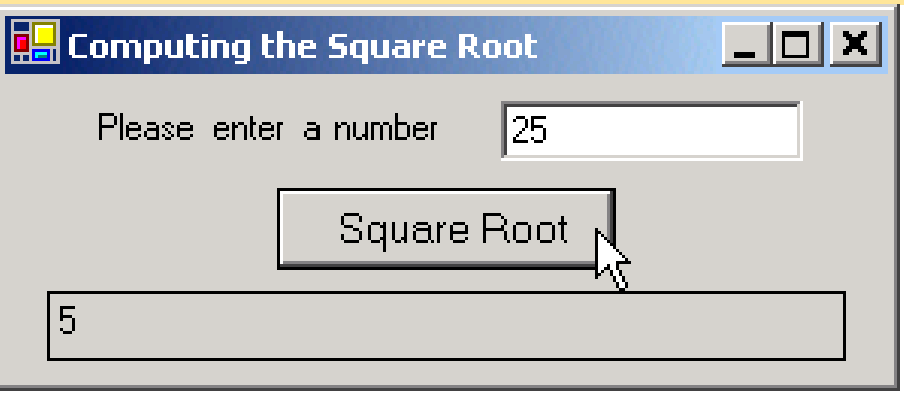


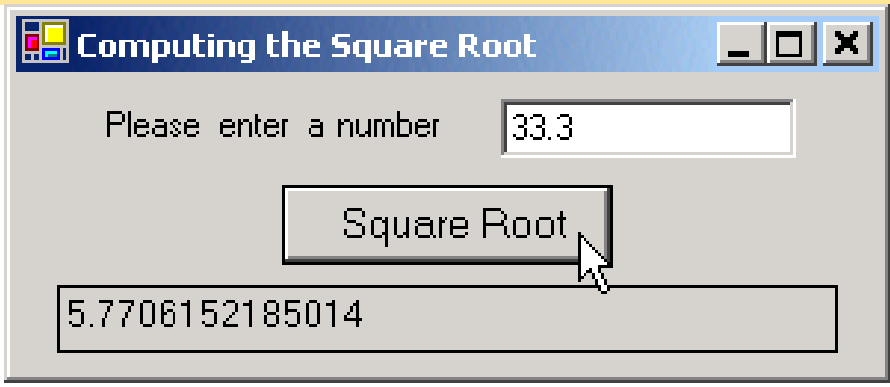


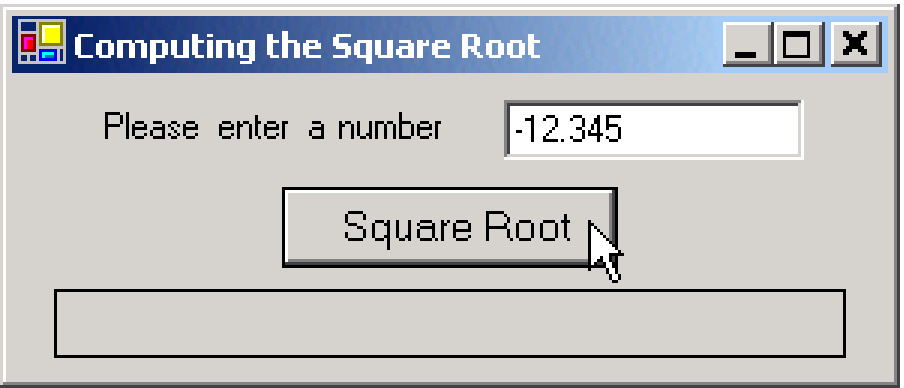
```
26      [STAThread]
27      static void Main() {
29         Application.Run( new SquareRootTest() );
30      }
33      public double SquareRoot( double operand ) {
36         if ( operand < 0 )
37            throw new NegativeNumberException(
38               "Square root of negative number not permitted" );
40         return Math.Sqrt( operand );
41      }
44      private void squareRootButton_Click(
45         object sender, System.EventArgs e ) {
47         outputLabel.Text = "";
49         try {
51            double result =
52               SquareRoot( Double.Parse(inputTextBox.Text ) );
53            outputLabel.Text = result.ToString();
54         }
```

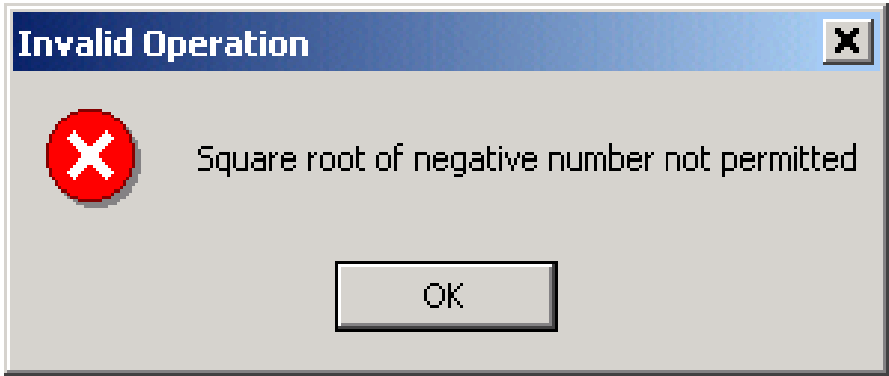


```
56          catch ( FormatException notInteger ) {
58             MessageBox.Show( notInteger.Message,
59                "Invalid Operation", MessageBoxButtons.OK,
60                MessageBoxIcon.Error );
61          }
63          catch ( NegativeNumberException error ) {
65             MessageBox.Show( error.Message, "Invalid Operation",
66                MessageBoxButtons.OK, MessageBoxIcon.Error );
67          }
68       }
69    }
```









11.8 Handling Overflows with Operators checked and unchecked

- C# provides operators ***checked*** and ***unchecked*** to specify whether integer arithmetic error occurs
- Calculation that could overflow
  - Use checked when performing calculations that may result in overflow
    - maximum for int is 2,147,483,647
  - The CLR throws an overflowException if overflow occur during calculation
- Unchecked
  - The result is of the overflow will be truncated and no exception occurs

```
3     using System;
5     class Overflow {
7        static void Main( string[] args ) {
9           int number1 = Int32.MaxValue;  // 2,147,483,647
10          int number2 = Int32.MaxValue;  // 2,147,483,647
11          int sum = 0;
12          Console.WriteLine(
13             "number1: {0}\nnumber2: {1}", number1, number2 );
15          try {
17             Console.WriteLine(
18                "\nSum integers in checked context:" );
19             sum = checked( number1 + number2 );
20          }
22          catch ( OverflowException overflowException ) {
24             Console.WriteLine( overflowException.ToString() );
25          }
26          Console.WriteLine(
27             "\nsum after checked operation: {0}", sum );
```

```
28          Console.WriteLine(
29             "\nSum integers in unchecked context:" );
30          sum = unchecked( number1 + number2 );
31          Console.WriteLine(
32             "sum after unchecked operation: {0}", sum );
33       }
34    }
```

```
number1: 2147483647
number2: 2147483647

Sum integers in checked context:
System.OverflowException: Arithmetic operation resulted in an
overflow.
   at Overflow.Overflow.Main(String[] args) in
      f:\books\2001\csphtp1\csphtp1_examples\ch11\fig11_09\
         overflow\overflow.cs:line 24

sum after checked operation: 0

Sum integers in unchecked context:
sum after unchecked operation: -2
```