```
// Fig. 14.1: DivideByZeroException.java
// Definition of class DivideByZeroException.
// Used to throw an exception when a
// divide-by-zero is attempted.
public class DivideByZeroException
             extends ArithmeticException {
   public DivideByZeroException()
   {
      super( "Attempted to divide by zero" );
   }

   public DivideByZeroException( String message )
   {
      super( message );
   }
}
```

Fig. 14.1 A simple exception handling example with divide by zero (part 1 of 4).

```
// Fig. 14.1: DivideByZeroTest.java
// A simple exception handling example.
// Checking for a divide-by-zero-error.
import java.text.DecimalFormat;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DivideByZeroTest extends JFrame
                              implements ActionListener {
   private JTextField input1, input2, output;
   private int number1, number2;
   private double result;

   // Initialization
   public DivideByZeroTest()
   {
      super( "Demonstrating Exceptions" );

      Container c = getContentPane();
      c.setLayout( new GridLayout( 3, 2 ) );

      c.add( new JLabel( "Enter numerator ",
                         SwingConstants.RIGHT ) );
      input1 = new JTextField( 10 );
      c.add( input1 );

      c.add(
         new JLabel( "Enter denominator and press Enter ",
                     SwingConstants.RIGHT ) );
      input2 = new JTextField( 10 );
      c.add( input2 );
```

```
49          input2.addActionListener( this );
```

Fig. 14.1 A simple exception handling example with divide by zero (part 2 of 4).

```
50
51          c.add( new JLabel( "RESULT ", SwingConstants.RIGHT ) );
52          output = new JTextField();
53          c.add( output );
54
55          setSize( 425, 100 );
56          show();
57       }
58
59       // Process GUI events
60       public void actionPerformed( ActionEvent e )
61       {
62          DecimalFormat precision3 = new DecimalFormat( "0.000" );
63
64          output.setText( "" ); // empty the output JTextField
65
66          try {
67             number1 = Integer.parseInt( input1.getText() );
68             number2 = Integer.parseInt( input2.getText() );
69
70             result = quotient( number1, number2 );
71             output.setText( precision3.format( result ) );
72          }
73          catch ( NumberFormatException nfe ) {
74             JOptionPane.showMessageDialog( this,
75                "You must enter two integers",
76                "Invalid Number Format",
77                JOptionPane.ERROR_MESSAGE );
78          }
79          catch ( DivideByZeroException dbze ) {
80             JOptionPane.showMessageDialog( this, dbze.toString(),
81                "Attempted to Divide by Zero",
82                JOptionPane.ERROR_MESSAGE );
83          }
84       }
85
86       // Definition of method quotient. Used to demonstrate
87       // throwing an exception when a divide-by-zero error
88       // is encountered.
89       public double quotient( int numerator, int denominator )
90          throws DivideByZeroException
91       {
92          if ( denominator == 0 )
93             throw new DivideByZeroException();
94
95          return ( double ) numerator / denominator;
96       }
97
98       public static void main( String args[] )
99       {
```

```
100         DivideByZeroTest app = new DivideByZeroTest();
101
```

Fig. 14.1 A simple exception handling example with divide by zero (part 3 of 4).

```
102         app.addWindowListener(
103            new WindowAdapter() {
104               public void windowClosing( WindowEvent e )
105               {
106                  e.getWindow().dispose();
107                  System.exit( 0 );
108               }
109            }
110         );
111      }
112   }
```

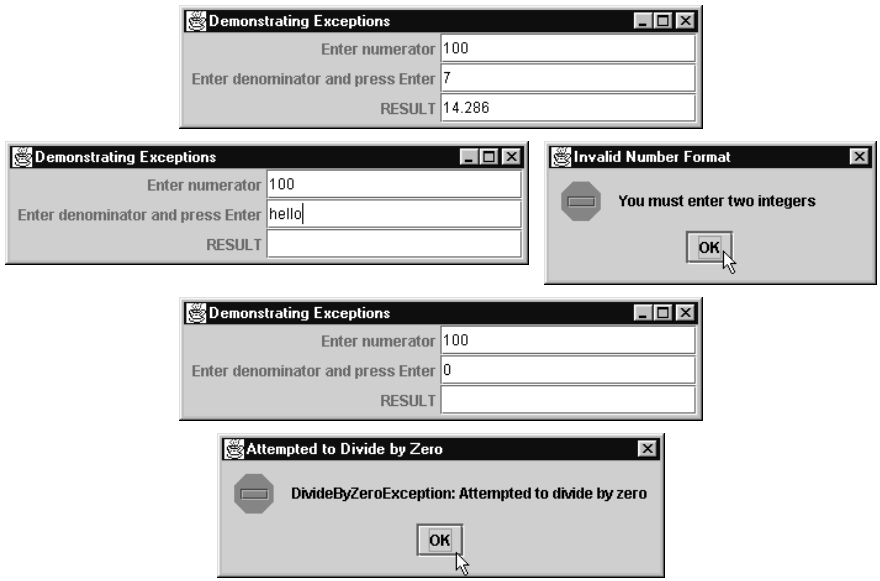

Fig. 14.1 A simple exception handling example with divide by zero (part 4 of 4).

The `java.lang` package errors

`Error` (all in **`java.lang`** except for **`AWTError`**, which is in **`java.awt`**)

- **`LinkageError`**
  - **`ClassCircularityError`**
  - **`ClassFormatError`**
  - **`ExceptionInInitializerError`**
  - **`IncompatibleClassChangeError`**
    - **`AbstractMethodError`**
    - **`IllegalAccessError`**
    - **`InstantiationError`**
    - **`NoSuchFieldError`**
    - **`NoSuchMethodError`**
  - **`NoClassDefFoundError`**
  - **`UnsatisfiedLinkError`**
  - **`VerifyError`**
- **`ThreadDeath`**
- **`VirtualMachineError (Abstract class)`**
  - **`InternalError`**
  - **`OutOfMemoryError`**
  - **`StackOverflowError`**
  - **`UnknownError`**
- **`AWTError`** (in **`java.awt`**)

Fig. 14.2 The **`java.lang`** package errors .

The `java.lang` package exceptions

```
Exception
   ClassNotFoundException
   CloneNotSupportedException
   IllegalAccessException
   InstantiationException
   InterruptedException
   NoSuchFieldException
   NoSuchMethodException
   RuntimeException
      ArithmeticException
      ArrayStoreException
      ClassCastException
      IllegalArgumentException
         IllegalThreadStateException
         NumberFormatException
      IllegalMonitorStateException
      IllegalStateException
      IndexOutOfBoundsException
         ArrayIndexOutOfBoundsException
         StringIndexOutOfBoundsException
      NegativeArraySizeException
      NullPointerException
      SecurityException
```

Fig. 14.3 The `java.lang` package exceptions.

The `java.util` package exceptions

```
Exception
   RuntimeException
      EmptyStackException
      MissingResourceException
      NoSuchElementException
   TooManyListenersException
```

Fig. 14.4 The `java.util` package exceptions.

The `java.io` package exceptions

```
Exception
    IOException
       CharConversionException
       EOFException
       FileNotFoundException
       InterruptedIOException
       ObjectStreamException
          InvalidClassException
          InvalidObjectException
          NotActiveException
          NotSerializableException
          OptionalDataException
          StreamCorruptedException
          WriteAbortedException
       SyncFailedException
       UnsupportedCodingException
       UTFDataFormatException
```

Fig. 14.5 The `java.io` package exceptions .

The `java.awt` package exceptions

```
Exception
      AWTException
      RuntimeException
         IllegalStateException
            IllegalComponentStateException
```

Fig. 14.6 The `java.awt` package exceptions .

The `java.net` package exceptions

```
Exception
   IOException
      BindException
      MalformedURLException
      ProtocolException
      SocketException
         ConnectException
         NoRouteToHostException
      UnknownHostException
      UnknownServiceException
```

Fig. 14.7 The `java.net` package exceptions.

```
1  // Fig. 14.8: UsingExceptions.java
2  // Demonstration of the try-catch-finally
3  // exception handling mechanism.
4  public class UsingExceptions {
5     public static void main( String args[] )
6     {
7        try {
8           throwException();
9        }
10       catch ( Exception e )
11       {
12          System.err.println( "Exception handled in main" );
13       }
14
15       doesNotThrowException();
16    }
```

Fig. 14.8 Demonstration of the **try-catch-finally** exception handling mechanism (part 1 of 2).

```
17
18    public static void throwException() throws Exception
19    {
20       // Throw an exception and immediately catch it.
21       try {
22          System.out.println( "Method throwException" );
23          throw new Exception();  // generate exception
24       }
25       catch( Exception e )
26       {
27          System.err.println(
28             "Exception handled in method throwException" );
29          throw e;  // rethrow e for further processing
30
31          // any code here would not be reached
32       }
33       finally {
34          System.err.println(
35             "Finally executed in throwException" );
36       }
37
38       // any code here would not be reached
39    }
40
41    public static void doesNotThrowException()
42    {
43       try {
44          System.out.println( "Method doesNotThrowException" );
45       }
46       catch( Exception e )
47       {
48          System.err.println( e.toString() );
49       }
```

```
50         finally {
51            System.err.println(
52               "Finally executed in doesNotThrowException" );
53         }
54
55         System.out.println(
56            "End of method doesNotThrowException" );
57      }
58   }
```

```
Method throwException
Exception handled in method throwException
Finally executed in throwException
Exception handled in main
Method doesNotThrowException
Finally executed in doesNotThrowException
End of method doesNotThrowException
```

Fig. 14.8 Demonstration of the **`try-catch-finally`** exception handling mechanism (part 2 of 2).

```
1  // Fig. 14.9: UsingExceptions.java
2  // Demonstration of stack unwinding.
3  public class UsingExceptions {
4     public static void main( String args[] )
5     {
6        try {
7           throwException();
8        }
9        catch ( Exception e ) {
10          System.err.println( "Exception handled in main" );
11       }
12    }
13
14    public static void throwException() throws Exception
15    {
16       // Throw an exception and catch it in main.
17       try {
18          System.out.println( "Method throwException" );
19          throw new Exception();      // generate exception
20       }
21       catch( RuntimeException e ) {  // nothing caught here
22          System.err.println( "Exception handled in " +
23                              "method throwException" );
24       }
25       finally {
26          System.err.println( "Finally is always executed" );
27       }
28    }
29 }
```

Fig. 14.9 Demonstration of stack unwinding (part 1 of 2).

```
Method throwException
Finally is always executed
Exception handled in main
```

Fig. 14.9 Demonstration of stack unwinding (part 2 of 2).

```
1  // Fig. 14.10: UsingExceptions.java
2  // Demonstrating the getMessage and printStackTrace
3  // methods inherited into all exception classes.
4  public class UsingExceptions {
5     public static void main( String args[] )
6     {
7        try {
8           method1();
9        }
10       catch ( Exception e ) {
11          System.err.println( e.getMessage() + "\n" );
```

Fig. 14.10 Using `getMessage` and `printStackTrace` (part 1 of 2).

```
12          e.printStackTrace();
13       }
14    }
15
16    public static void method1() throws Exception
17    {
18       method2();
19    }
20
21    public static void method2() throws Exception
22    {
23       method3();
24    }
25
26    public static void method3() throws Exception
27    {
28       throw new Exception( "Exception thrown in method3" );
29    }
30 }
```

```
Exception thrown in method3
java.lang.Exception: Exception thrown in method3
        at UsingExceptions.method3(UsingExceptions.java:28)
        at UsingExceptions.method2(UsingExceptions.java:23)
        at UsingExceptions.method1(UsingExceptions.java:18)
        at UsingExceptions.main(UsingExceptions.java:8)
```

Fig. 14.10 Using `getMessage` and `printStackTrace` (part 2 of 2).