CS360 Lecture 9
Exception Handling

Tuesday, March 2, 2004

Reading

Nested Classes: Section 10.9
Exception Handling: Chapter 15

Nested Classes

Java allows classes to be declared inside of other classes. If these classes are not static then they are called inner classes. Inner classes are primarily used in event handling.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TimeTestWindow extends JFrame {
   private Time time;
   private JLabel hourLabel, minuteLabel, secondLabel;
   private JTextField hourField, minuteField,secondField, displayField;
   private JButton exitButton;

   public TimeTestWindow()
   {
      super( "Inner Class Demonstration" );

      time = new Time();

      Container container = getContentPane();
      container.setLayout( new FlowLayout() );

      hourLabel = new JLabel( "Set Hour" );
      hourField = new JTextField( 10 );
      container.add( hourLabel );
      container.add( hourField );
      minuteLabel = new JLabel( "Set Minute" );
      minuteField = new JTextField( 10 );
      container.add( minuteLabel );
      container.add( minuteField );

      secondLabel = new JLabel( "Set Second" );
      secondField = new JTextField( 10 );
      container.add( secondLabel );
      container.add( secondField );

      displayField = new JTextField( 30 );
      displayField.setEditable( false );
      container.add( displayField );

      exitButton = new JButton( "Exit" );
```

```
      container.add( exitButton );

      ActionEventHandler handler = new ActionEventHandler();

      hourField.addActionListener( handler );
      minuteField.addActionListener( handler );
      secondField.addActionListener( handler );
      exitButton.addActionListener( handler );

   }

   public void displayTime()
   {
      displayField.setText("The time is: " + time);
   }

   public static void main( String args[] )
   {
      TimeTestWindow window = new TimeTestWindow();

      window.setSize( 400, 140 );
      window.setVisible( true );

   }
   private class ActionEventHandler implements ActionListener {

      public void actionPerformed( ActionEvent event )
      {
         if ( event.getSource() == exitButton )
            System.exit( 0 );

         else if ( event.getSource() == hourField )
         {
            time.setHour( Integer.parseInt(event.getActionCommand()) );
            hourField.setText( "" );
         }

         else if (event.getSource() == minuteField)
         {
            time.setMinute(Integer.parseInt(event.getActionCommand()));
            minuteField.setText( "" );
         }

         else if (event.getSource() == secondField)
         {
            time.setSecond(Integer.parseInt(event.getActionCommand()));
            secondField.setText( "" );
         }

         displayTime();
      }
   }
}
```

What class files will be created for the above program?

Anonymous Inner Classes

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TimeTestWindow2 extends JFrame {
   private Time time;
   private JLabel hourLabel, minuteLabel, secondLabel;
   private JTextField hourField, minuteField,secondField, displayField;

   public TimeTestWindow2()
   {
      super("Anonymous Inner Class Demonstration");

      time = new Time();
      createGUI();
      registerEventHandlers();
   }

   private void createGUI()
   {
      Container container = getContentPane();
      container.setLayout( new FlowLayout() );

      hourLabel = new JLabel( "Set Hour" );
      hourField = new JTextField( 10 );
      container.add( hourLabel );
      container.add( hourField );

      minuteLabel = new JLabel( "Set minute" );
      minuteField = new JTextField( 10 );
      container.add( minuteLabel );
      container.add( minuteField );

      secondLabel = new JLabel( "Set Second" );
      secondField = new JTextField( 10 );
      container.add( secondLabel );
      container.add( secondField );

      displayField = new JTextField( 30 );
      displayField.setEditable( false );
      container.add( displayField );
   }

   private void registerEventHandlers()
   {
      hourField.addActionListener(

         new ActionListener() {
            public void actionPerformed( ActionEvent event )
            {
              time.setHour(Integer.parseInt(event.getActionCommand()));
               hourField.setText( "" );
               displayTime();
            }
```

```
         }
      );

      minuteField.addActionListener(

         new ActionListener() {
            public void actionPerformed( ActionEvent event )
            {
               time.setMinute( Integer.parseInt(
                                          event.getActionCommand() ) );
               minuteField.setText( "" );
               displayTime();
            }
         }
      );
      secondField.addActionListener(

         new ActionListener() {
            public void actionPerformed( ActionEvent event )
            {
               time.setSecond( Integer.parseInt(
                                          event.getActionCommand() ) );
               secondField.setText( "" );
               displayTime();
            }
         }
      );
   }

   public void displayTime()
   {
      displayField.setText("The time is: " + time);
   }

   public static void main( String args[] )
   {
      TimeTestWindow2 window = new TimeTestWindow2();

      window.addWindowListener(

         new WindowAdapter() {
            public void windowClosing( WindowEvent event )
            {
               System.exit( 0 );
            }
         }
      );
      window.setSize( 400, 105 );
      window.setVisible( true );
   } // end main
} // end class TimeTestWindow2
```

What class files are created when the above program is compiled?

Exception Handling

Question: What is an exception?
An exception is an indication of a problem that occurs during a program's execution.

Handling an exception allows a program to either:
- continue executing as if no problem occurred
- notify the user and terminate the program

```
Perform a task

If the task did not execute correctly
 Perform error processing

Perform next task

If the task did not execute correctly
 Perform error processing
```

Here we mix program logic with error-handling logic. Makes the program harder to:
- read
- modify
- maintain
- debug

Could also degrade the program's performance.

Synchronous vs. Asynchronous Errors

Synchronous errors are those that occur during the program's flow of control such as divide by zero, array index out of bounds, etc.
Asynchronous are those that occur independent of the program's flow of control such as mouse clicks, network message arrivals, etc.

Exception handling is geared to situations in which the method that detects a problem is unable to handle it. The method then throws an exception.

Example

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class DivideByZeroTest extends Jframe implements ActionListener
{
   private JTextField inputField1, inputField2, outputField;
   private int number1, number2, result;

   public DivideByZeroTest()
   {
      super( "Demonstrating Exceptions" );
```

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

   container.add(new JLabel( "Enter numerator ",
                             SwingConstants.RIGHT ) );
   inputField1 = new JTextField();
   container.add( inputField1 );

   container.add(new JLabel("Enter denominator and press Enter ",
                             SwingConstants.RIGHT ) );
   inputField2 = new JTextField();
   container.add( inputField2 );
   inputField2.addActionListener( this );

   container.add( new JLabel( "RESULT ",
                               SwingConstants.RIGHT ) );
   outputField = new JTextField();
   container.add( outputField );

   setSize( 425, 100 );
   setVisible( true );
}

public void actionPerformed( ActionEvent event )
{
   outputField.setText( "" );
   try
   {
      number1 = Integer.parseInt( inputField1.getText() );
      number2 = Integer.parseInt( inputField2.getText() );

      result = quotient( number1, number2 );
      outputField.setText( String.valueOf( result ) );
   }

   catch ( NumberFormatException numberFormatException )
   {
      JOptionPane.showMessageDialog( this,
                     "You must enter two integers",
                     "Invalid Number Format",
                     JOptionPane.ERROR_MESSAGE );
   }

   catch ( ArithmeticException arithmeticException )
   {
      JOptionPane.showMessageDialog( this,
                     arithmeticException.toString(),
                     "Arithmetic Exception",
                     JOptionPane.ERROR_MESSAGE );
   }
}

public int quotient( int numerator, int denominator )
   throws ArithmeticException
{
   return numerator / denominator;
}
```

```
   public static void main( String args[] )
   {
      DivideByZeroTest application = new DivideByZeroTest();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }
}
```

Parts of exception handling:

- throws: specified in the method that could throw an exception
- try: encloses code that might throw an exception and the code that should not execute if an exception occurs
- catch: encloses code that will execute if the specified exception occurs

How do we know what exception to throw or catch?

Exception Hierarchy

Exceptions are objects. There are two categories of exceptions:

- Checked exceptions
- Unchecked exceptions

Checked exceptions are tested by the constructor. If a method that you are using throws a checked exception, then you must also place that method within a try-catch clause. If the method call is within a method, then that method must throw the appropriate exception.

`finally` Clause

This is placed at the end of a try-catch block. It is executed regardless of whether an exception gets thrown or not.

The finally clause is optional and usually contains resources-release code.

Stack Trace

```
public class UsingExceptions {

   public static void main( String args[] )
   {
      try {
         method1(); // call method1
      }
      catch ( Exception exception ) {
         System.err.println( exception.getMessage() + "\n" );
         exception.printStackTrace();

         StackTraceElement[] traceElements = exception.getStackTrace();

         System.out.println(  "\nStack trace from getStackTrace:" );
         System.out.println( "Class\t\tFile\t\t\tLine\tMethod" );

         for(int i=0; i<traceElements.length; i++ )
            {
            StackTraceElement currentElement =  traceElements[ i ];
```

```
            System.out.print( currentElement.getClassName() + "\t" );
            System.out.print( currentElement.getFileName() + "\t" );
            System.out.print( currentElement.getLineNumber() + "\t" );
            System.out.print( currentElement.getMethodName() + "\n" );
         }
      }
   }

   public static void method1() throws Exception
   {
      method2();
   }

   public static void method2() throws Exception
   {
      method3();
   }

   public static void method3() throws Exception
   {
      throw new Exception( "Exception thrown in method3" );
   }
}

Exception thrown in method3

java.lang.Exception: Exception thrown in method3
        at UsingExceptions.method3(UsingExceptions.java:51)
        at UsingExceptions.method2(UsingExceptions.java:45)
        at UsingExceptions.method1(UsingExceptions.java:39)
        at UsingExceptions.main(UsingExceptions.java:8)

Stack trace from getStackTrace:
Class           File                    Line    Method
UsingExceptions UsingExceptions.java    51      method3
UsingExceptions UsingExceptions.java    45      method2
UsingExceptions UsingExceptions.java    39      method1
UsingExceptions UsingExceptions.java    8       main
```

Declaring New Exception Types

Most programmers use existing exception classes from the Java API or from the third party vendors.

If you do need to create an exception class, then you should extend the class Exception and specify two constructors.

```
public class MyException extends Exception
{
	public MyException()
	{
	}

	public MyException( String reason )
	{
		super( reason );
	}
}

public class NonZero
{
	public NonZero( int number )
		throws MyException
	{
		if( number == 0 )
			throw new MyException( "The argument was zero" );

		System.out.println( "Number " + number + " is non zero" );
	}

	public static void main( String[] args )
	{
		try
		{
			NonZero nz1 = new NonZero( 1 );
			NonZero nz0 = new NonZero( 0 );
			System.out.println( "All OK" );
		}
		catch( MyException e )
		{
			System.out.println( e );
		}
	}
}
```