Ollscoil na Gaillimhe
University of Galway

CT326 Programming III

LECTURE 7

NESTED CLASSES

DR ADRIAN CLEAR
SCHOOL OF COMPUTER SCIENCE

Objectives for today

- Understand nested classes
- Demonstrate the use of static nested classes, inner classes, and anonymous inner classes

https://vevox.app/#/m/178336857

Nested Classes

- You can define a class as a member of another class. Such a class is called a nested class and is illustrated here:

```
class EnclosingClass {
    ...
    class ANestedClass {
        ...
    }
}
```

Relationship to enclosing class

- You use nested classes to reflect and to enforce the relationship between two classes.
  - You should define a class within another class when the nested class makes sense only in the context of its enclosing class or when it relies on the enclosing class for its function.
  - For example, a text cursor might make sense only in the context of a text component.
- As a member of its enclosing class, a nested class has a special privilege: It has unlimited access to its enclosing class's members, even if they are declared private.

Static and inner classes

- Like other class members, a nested class can be declared static (or not).
  - A static nested class is called just that: a static nested class.
  - A non-static nested class is called an inner class.

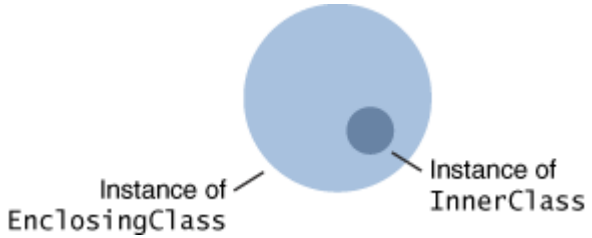
```
class EnclosingClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

Static and inner classes

- As with static methods and variables, which we call class methods and variables, a static nested class is associated with its enclosing class.
  - And like class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class — it can use them only through an object reference.
- As with instance methods and variables, an inner class is associated with an instance of its enclosing class and has direct access to that object's instance variables and methods.

Nested inner classes

- The interesting feature about the relationship between these two classes is not that the Inner Class is syntactically defined within Enclosing Class.
- Rather, it's that an instance of Inner Class can exist only within an instance of Enclosing Class and that it has direct access to the instance variables and methods of its enclosing instance.
- You may encounter nested classes of both kinds (static and inner) in the Java platform API and be required to use them.
- However, most nested classes that you write will probably be inner classes.



Using Anonymous Inner Classes

- Tokenizer
  - Partition **`String`** into individual substrings
  - Use *delimiter*
  - Java offers **`java.util.StringTokenizer`**

```
1    // Fig. 10.20: TokenTest.java
2    // Testing the StringTokenizer class of the java.util package
3
4    // Java core packages
5    import java.util.*;
6    import java.awt.*;
7    import java.awt.event.*;
8
9    // Java extension packages
10   import javax.swing.*;
11
12   public class TokenTest extends JFrame {
13      private JLabel promptLabel;
14      private JTextField inputField;
15      private JTextArea outputArea;
16
17      // set up GUI and event handling
18      public TokenTest()
19      {
20         super( "Testing Class StringTokenizer" );
21
22         Container container = getContentPane();
23         container.setLayout( new FlowLayout() );
24
25         promptLabel =
26            new JLabel( "Enter a sentence and press Enter" );
27         container.add( promptLabel );
28
29         inputField = new JTextField( 20 );
30
31         inputField.addActionListener(
32
33            // anonymous inner class
34            new ActionListener() {
35
```

TokenTest.java

Line 29

inputField contains **String** to be parsed by **StringTokenizer**

```
36             // handle text field event
37             public void actionPerformed( ActionEvent
38             {
39                String stringToTokenize =
40                   event.getActionCommand();
41                StringTokenizer tokens =
42                   new StringTokenizer( stringToTokenize );
43
44                outputArea.setText( "Number of elements: " +
45                   tokens.countTokens() + "\nThe tokens are:\n" );
46
47                while ( tokens.hasMoreTokens() )
48                   outputArea.append( tokens.nextToken() + "\n" );
49             }
50
51          }  // end anonymous inner class
52
53       ); // end call to addActionListener
54
55       container.add( inputField );
56
57       outputArea = new JTextArea( 10, 20 );
58       outputArea.setEditable( false );
59       container.add( new JScrollPane( outputArea ) );
60
61       setSize( 275, 260 );  // set the window size
62       show();               // show the window
63    }
64
65    // execute application
66    public static void main( String args[] )
67    {
68       TokenTest application = new TokenTest();
69
```

Use **`StringTokenizer`** to parse **`String stringToTokenize`** with default delimiter “ **`\n\t\r`**”

Count number of tokens

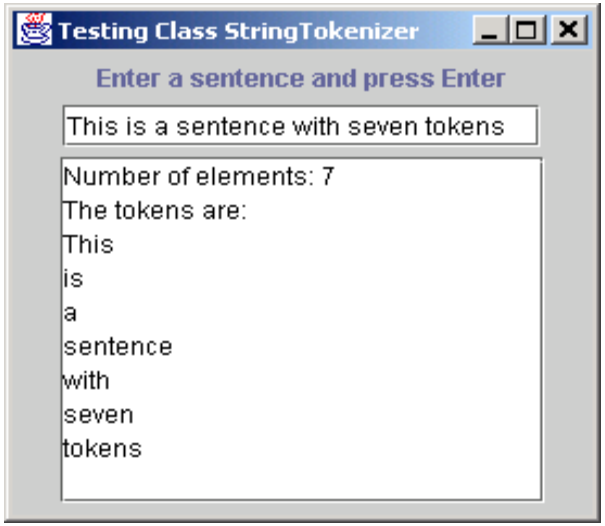
Line 45

Lines 47-48

Append next token to **`outputArea`**, as long as tokens exist

```
70          application.addWindowListener(
71
72             // anonymous inner class
73             new WindowAdapter() {
74
75                // handle event when user closes window
76                public void windowClosing( WindowEvent windowEvent )
77                {
78                   System.exit( 0 );
79                }
80
81             }  // end anonymous inner class
82
83          ); // end call to addWindowListener
84
85       }  // end method main
86
87    }  // end class TokenTest
```

TokenTest.java



In-class demo

- Create a class to represent an array data structure of specified size, populated with ascending integer values
- Includes a method, `printEven` that prints the even values of the array
- Uses a nested inner class to iterate over even numbers of the array.

Next time…

- Enums