Multithreading with GUI

- All Swing applications have a single thread, called the event dispatch thread, to handle interactions with the application's GUI components.
- All tasks that require interaction with an application's GUI are placed in an event queue and are executed sequentially by the event dispatch thread.
- Swing GUI components are not thread safe.
- Thread safety in GUI applications is achieved by ensuring that Swing components are accessed from only a single thread—the event dispatch thread.
  - Called thread confinement.

Multithreading with GUI (cont.)

- Java SE 6 provides class SwingWorker (in package `javax.swing`) to perform long-running computations in a worker thread and to update Swing components from the event dispatch thread based on the computations' results.
  - `Implements the Runnable interface, meaning that a SwingWorker object can be scheduled to execute in a separate thread.`
- Some common `SwingWorker` methods are described on next slide.

Method	Description
`doInBackground`	Defines a long computation and is called in a worker thread.
`done`	Executes on the event dispatch thread when `doInBackground` returns.
`execute`	Schedules the `SwingWorker` object to be executed in a worker thread.
`get`	Waits for the computation to complete, then returns the result of the computation (i.e., the return value of `doInBackground`).
`publish`	Sends intermediate results from the `doInBackground` method to the `process` method for processing on the event dispatch thread.
`process`	Receives intermediate results from the `publish` method and processes these results on the event dispatch thread.
`setProgress`	Sets the progress property to notify any property change listeners on the event dispatch thread of progress bar updates.

Performing Computations in a Worker Thread

- Class `BackgroundCalculator` extends `SwingWorker` (line 8), overriding the methods `doInBackground` and `done`.
- Method `doInBackground` (lines 21–24) computes the *n*th Fibonacci number in a worker thread and returns the result.
- Method `done` (lines 27–43) displays the result in a `JLabel`.

```
1  // Fig. 26.24: BackgroundCalculator.java
2  // SwingWorker subclass for calculating Fibonacci numbers
3  // in a background thread.
4  import javax.swing.SwingWorker;
5  import javax.swing.JLabel;
6  import java.util.concurrent.ExecutionException;
7
8  public class BackgroundCalculator extends SwingWorker< Long, Object >
9  {
10    private final int n; // Fibonacci number to calculate
11    private final JLabel resultJLabel; // JLabel to display the result
12
```

```
13      // constructor
14      public BackgroundCalculator( int number, JLabel label )
15      {
16         n = number;
17         resultJLabel = label;
18      } // end BackgroundCalculator constructor
19
20      // long-running code to be run in a worker thread
21      public Long doInBackground()
22      {
23         return nthFib = fibonacci( n );
24      } // end method doInBackground
25
```

```
26      // code to run on the event dispatch thread when doInBackground returns
27      protected void done()
28      {
29         try
30         {
31            // get the result of doInBackground and display it
32            resultJLabel.setText( get().toString() );
33         } // end try
34         catch ( InterruptedException ex )
35         {
36            resultJLabel.setText( "Interrupted while waiting for results." );
37         } // end catch
38         catch ( ExecutionException ex )
39         {
40            resultJLabel.setText(
41               "Error encountered while performing calculation." );
42         } // end catch
43      } // end method done
44
```

```
45      // recursive method fibonacci; calculates nth Fibonacci number
46      public long fibonacci( long number )
47      {
48         if ( number == 0 || number == 1 )
49            return number;
50         else
51            return fibonacci( number - 1 ) + fibonacci( number - 2 );
52      } // end method fibonacci
53   } // end class BackgroundCalculator
```

Performing Computations in a Worker Thread (cont.)

- Class `FibonacciNumbers` displays a window containing two sets of GUI components—one set to compute a Fibonacci number in a worker thread and another to get the next Fibonacci number in response to the user's clicking a `JButton`.

```
1  // Fig. 26.25: FibonacciNumbers.java
2  // Using SwingWorker to perform a long calculation with
3  // results displayed in a GUI.
4  import java.awt.GridLayout;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import javax.swing.JButton;
8  import javax.swing.JFrame;
9  import javax.swing.JPanel;
10 import javax.swing.JLabel;
11 import javax.swing.JTextField;
12 import javax.swing.border.TitledBorder;
13 import javax.swing.border.LineBorder;
14 import java.awt.Color;
15 import java.util.concurrent.ExecutionException;
16
```

```
17  public class FibonacciNumbers extends JFrame
18  {
19     // components for calculating the Fibonacci of a user-entered number
20     private final JPanel workerJPanel =
21        new JPanel( new GridLayout( 2, 2, 5, 5 ) );
22     private final JTextField numberJTextField = new JTextField();
23     private final JButton goJButton = new JButton( "Go" );
24     private final JLabel fibonacciJLabel = new JLabel();
25
26     // components and variables for getting the next Fibonacci number
27     private final JPanel eventThreadJPanel =
28        new JPanel( new GridLayout( 2, 2, 5, 5 ) );
29     private long n1 = 0; // initialize with first Fibonacci number
30     private long n2 = 1; // initialize with second Fibonacci number
31     private int count = 1; // current Fibonacci number to display
32     private final JLabel nJLabel = new JLabel( "Fibonacci of 1: " );
33     private final JLabel nFibonacciJLabel =
34        new JLabel( String.valueOf( n2 ) );
35     private final JButton nextNumberJButton = new JButton( "Next Number" );
36
```

```
37       // constructor
38       public FibonacciNumbers()
39       {
40          super( "Fibonacci Numbers" );
41          setLayout( new GridLayout( 2, 1, 10, 10 ) );
42
43          // add GUI components to the SwingWorker panel
44          workerJPanel.setBorder( new TitledBorder(
45             new LineBorder( Color.BLACK ), "With SwingWorker" ) );
46          workerJPanel.add( new JLabel( "Get Fibonacci of:" ) );
47          workerJPanel.add( numberJTextField );
48          goJButton.addActionListener(
49             new ActionListener()
50             {
51                public void actionPerformed( ActionEvent event )
52                {
53                   int n;
54
```

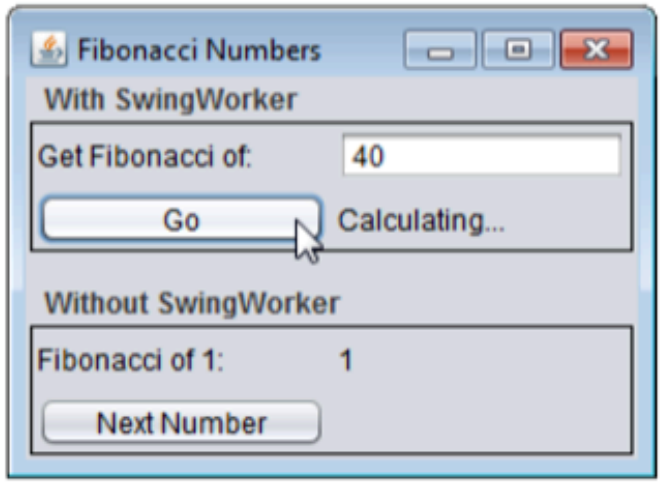
```
55                  try
56                  {
57                     // retrieve user's input as an integer
58                     n = Integer.parseInt( numberJTextField.getText() );
59                  } // end try
60                  catch( NumberFormatException ex )
61                  {
62                     // display an error message if the user did not
63                     // enter an integer
64                     fibonacciJLabel.setText( "Enter an integer." );
65                     return;
66                  } // end catch
67
```

```
68                  // indicate that the calculation has begun
69                  fibonacciJLabel.setText( "Calculating..." );
70
71                  // create a task to perform calculation in background
72                  BackgroundCalculator task =
73                     new BackgroundCalculator( n, fibonacciJLabel );
74                  task.execute(); // execute the task
75               } // end method actionPerformed
76            } // end anonymous inner class
77         ); // end call to addActionListener
78         workerJPanel.add( goJButton );
79         workerJPanel.add( fibonacciJLabel );
80
```

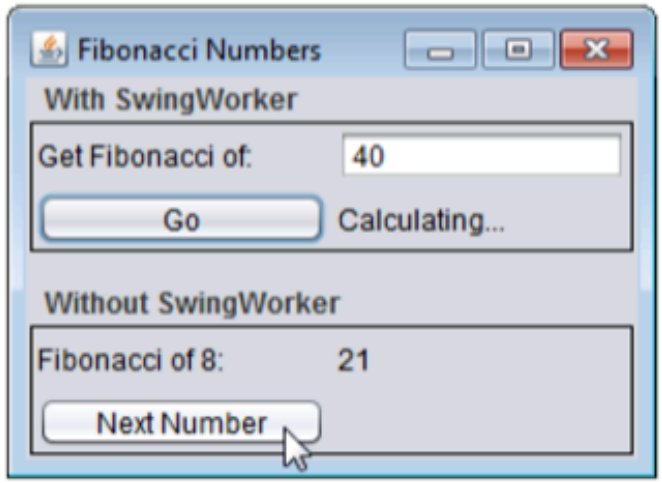
```
81          // add GUI components to the event-dispatching thread panel
82          eventThreadJPanel.setBorder( new TitledBorder(
83             new LineBorder( Color.BLACK ), "Without SwingWorker" ) );
84          eventThreadJPanel.add( nJLabel );
85          eventThreadJPanel.add( nFibonacciJLabel );
86          nextNumberJButton.addActionListener(
87             new ActionListener()
88             {
89                public void actionPerformed( ActionEvent event )
90                {
91                   // calculate the Fibonacci number after n2
92                   long temp = n1 + n2;
93                   n1 = n2;
94                   n2 = temp;
95                   ++count;
96
97                   // display the next Fibonacci number
98                   nJLabel.setText( "Fibonacci of " + count + ": " );
99                   nFibonacciJLabel.setText( String.valueOf( n2 ) );
100               } // end method actionPerformed
101            } // end anonymous inner class
102         ); // end call to addActionListener
```

```
103         eventThreadJPanel.add( nextNumberJButton );
104
105         add( workerJPanel );
106         add( eventThreadJPanel );
107         setSize( 275, 200 );
108         setVisible( true );
109      } // end constructor
110
111      // main method begins program execution
112      public static void main( String[] args )
113      {
114         FibonacciNumbers application = new FibonacciNumbers();
115         application.setDefaultCloseOperation( EXIT_ON_CLOSE );
116      } // end main
117   } // end class FibonacciNumbers
```

a) Begin calculating Fibonacci of 40 in the background



b) Calculating other Fibonacci values while Fibonacci of 40 continues calculating



c) Fibonacci of 40 calculation finishes