Supplement: SwingWorker and JProgressBar
For Introduction to Java Programming
By Y. Daniel Liang

This supplement introduces two advanced Swing components SwingWorker and JProgressBar. You can read this supplement after completing the chapter on threads.

1 SwingWorker

.Side Remark: event dispatch thread
All Swing GUI events are processed in a single event dispatch thread. If an event requires a long time to process, the thread cannot attend to other tasks in the queue. To solve this problem, you should run the time-consuming task for processing the event in a separate thread. Java 6 introduced SwingWorker. SwingWorker is an abstract class that implements Runnable. You can define a task class that extends SwingWorker, run the time-consuming task, and update the GUI using the results produced from the task. Figure 1 defines SwingWorker.

![interface]
java.lang.Runnable

javax.swing.SwingWorker<T, V>

- doInBackground(): T
- done(): void
- execute(): void
- get(): T
- isDone(): boolean
- cancel(): boolean
- publish(data V...): void
- process(data: java.util.List<V>): void
- setProgress(progress: int): void
- getProgress(): void

Figure 1
The SwingWorker class can be used to process time-consuming tasks.

This section demonstrates basic use of SwingWorker. The next section gives an example involving advanced features of SwingWorker.

.Side Remark: doInBackground()  
.Side Remark: done()  
To use SwingWorker, your task class should override doInBackground() to perform a time-consuming task and...
override the done() method to update GUI components if necessary. Listing 1 gives an example that lets the user specify a number and displays the number of prime numbers less than or equal to the specified number, as shown in Figure 2.

![Figure 2](image.png)

You can compare the effect of using versus not using SwingWorker.

**Listing 1 SwingWorkerDemo.java**

```java
***PD: Please add line numbers in the following code***
<Side Remark line 6: GUI components>
<Side Remark line 13: create UI>
<Side Remark line 14: left panel>
<Side Remark line 27: right panel>
<Side Remark line 44: add listener>
<Side Remark line 46: create task>
<Side Remark line 47: create task>
<Side Remark line 51: add listener>
<Side Remark line 66: constructor>
<Side Remark line 72: override doInBackground()>
<Side Remark line 77: override done()>
<Side Remark line 87: getNumberOfPrimes>
<Side Remark line 116: main method omitted>
```

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class SwingWorkerDemo extends JApplet {
    private JButton jbtComputeWithSwingWorker = new JButton("Compute");
    private JTextField jtfLimit1 = new JTextField(8);
    private JTextField jtfResult1 = new JTextField(6);
    private JButton jbtCompute = new JButton("Compute");
    private JTextField jtfLimit2 = new JTextField(8);
    private JTextField jtfResult2 = new JTextField(6);

    public SwingWorkerDemo() {
        JPanel panel1 = new JPanel(new GridLayout(2, 1));
        panel1.setBorder(BorderFactory.createTitledBorder(
            "Using SwingWorker");
        JPanel panel11 = new JPanel();
        panel11.add(new JLabel("The number of prime numbers <= "));
        panel11.add(jtfLimit1);
        panel11.add(new JLabel("is");
        panel11.add(jtfResult1);
        JPanel panel12 = new JPanel();
        panel12.add(jbtComputeWithSwingWorker);
        panel1.add(panel11);
        panel1.add(panel12);

        JPanel panel2 = new JPanel(new GridLayout(2, 1));
        panel2.setBorder(BorderFactory.createTitledBorder(
            "Without Using SwingWorker");
        JPanel panel21 = new JPanel();
        panel21.add(new JLabel("The number of prime numbers <= "));
        JPanel panel22 = new JPanel();
        panel22.add(jbtCompute);
        panel2.add(panel21);
        panel2.add(panel22);
    }

    public void main(String args[]) {
        // code for main method
    }

    int getNumberOfPrimes(int limit) {
        // code for calculate number of primes
    }

    protected void actionPerformed(ActionEvent e) {
        // code for adding listener
    }

    protected void doInBackground() {
        // code for executing task in background
    }

    protected void done() {
        // code for updating GUI components
    }
}
```

© Copyright Y. Daniel Liang, 2005
```java
panel21.add(new JLabel("is"));
panel21.add(jtfResult2);
JPanel panel22 = new JPanel();
panel22.add(jbtCompute);
panel2.add(panel21);
panel2.add(panel22);
setLayout(new GridLayout(1, 2));
add(panel1);
add(panel2);

jbtComputeWithSwingWorker.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        new ComputePrime(Integer.parseInt(jtfLimit1.getText()),
                          jtfResult1).execute(); // Execute SwingWorker
    }
});

jbtCompute.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int count = ComputePrime.getNumberOfPrimes(
                   Integer.parseInt(jtfLimit2.getText()));
        jtfResult2.setText(count + "");
    }
});

/** Task class for SwingWorker */
static class ComputePrime extends SwingWorker<Integer, Object> {
    private int limit;
    private JTextField result; // Text field in the UI
    /** Construct a runnable Task */
    public ComputePrime(int limit, JTextField result) {
        this.limit = limit;
        this.result = result;
    }
    /** Code run on a background thread */
    protected Integer doInBackground() {
        return getNumberOfPrimes(limit);
    }
    /** Code executed after the background thread finishes */
    protected void done() {
        try {
            result.setText(get().toString()); // Display in text field
        } catch (Exception ex) {
            result.setText(ex.getMessage());
        }
    }
    /** Return the number of primes <= limit */
    public static int getNumberOfPrimes(int limit) {
        int count = 0; // Count the number of prime numbers
        int number = 2; // A number to be tested for primeness
        // Repeatedly find prime numbers
        while (number <= limit) {
            // Print the prime number and increase the count
            if (isPrime(number)) {
                count++; // Increase the count
            }
            // Check if the next number is prime
            number++;
        }
        return count;
    }
    /** Check whether number is prime */
    private static boolean isPrime(int number) {
        for (int divisor = 2; divisor < number / 2; divisor++) {
            if (number % divisor == 0) { // If true, number is not prime
                return false; // number is not a prime
            }
        }
        return true;
    }
}
```

---

© Copyright Y. Daniel Liang, 2005
The UI consists of two panels. The left panel demonstrates how to compute the number of prime numbers using a SwingWorker. The right panel demonstrates how to compute the number of prime numbers without using a SwingWorker. You enter a number (e.g., 100000) in the first text field in the panel and click the Compute button to display the number of primes in the second text field. When you click the Compute button in the left panel, a SwingWorker task is created and executed (lines 46–47). Since the task is run on a separate thread, you can continue to use the GUI. However, when you click the Compute button in the right panel, the GUI is frozen, because the getNumberOfPrimes method is executed on the event dispatch thread (lines 53–54).

The inner class ComputePrime is a SwingWorker (line 61). It overrides the doInBackground method to run getNumberOfPrimes in a background thread (lines 72–74). It also overrides the done method to display the result in a text field, once the background thread finishes (lines 77–84).

The ComputePrime class defines the static getNumberOfPrimes method for computing the number of primes (lines 87–103). When you click the Compute button in the left panel, this method is executed on a background thread (lines 46–47). When you click the Compute button in the right panel, this method is executed in the event dispatch thread (lines 53–54).

TIP:

Two things to remember when writing Swing GUI programs:

- Time-consuming tasks should be run in SwingWorker.
- Swing components should be accessed from the event dispatch thread only.

### 2 Displaying Progress Using JProgressBar

In the preceding example, it may take a long time to finish the computation in the background thread. It is better to inform the user the progress of the computation. You can use
JProgressBar is a component that displays a value graphically within a bounded interval. A progress bar is typically used to show the percentage of completion of a lengthy operation; it comprises a rectangular bar that is "filled in" from left to right horizontally or from bottom to top vertically as the operation is performed. It provides the user with feedback on the progress of the operation. For example, when a file is being read, it alerts the user to the progress of the operation, thereby keeping the user attentive.

JProgressBar is often implemented using a thread to monitor the completion status of other threads. The progress bar can be displayed horizontally or vertically, as determined by its orientation property. The minimum, value, and maximum properties determine the minimum, current, and maximum lengths on the progress bar, as shown in Figure 3. Figure 4 lists frequently used features of JProgressBar.

![Figure 3](image)

JProgressBar displays the progress of a task.

**<PD: UML Class Diagram>**

| javax.swing.JComponent
| javax.swing.JProgressBar
| +JProgressBar()
| +JProgressBar(min: int, max: int)
| +JProgressBar(orient: int)
| +JProgressBar(orient: int, min: int, max: int)
| +getMaximum(): int
| +setMaximum(n: int): void
| +getMinimum(): int
| +setMinimum(n: int): void
| +getOrientation(): int
| +setOrientation(orient: int): void
| +getPercentComplete(): double
| +getValue(): int
| +setValue(n: int): void
| +getString(): String
| +setString(s: String): void
| +isStringPainted(): boolean
| +setStringPainted(b: boolean): void

Creates a horizontal progress bar with min 0 and max 100.
Creates a horizontal progress bar with specified min and max.
Creates a progress bar with min 0 and max 100 and a specified orientation.
Creates a progress bar with a specified orientation, min, and max.

Gets the maximum value (default: 100).
Sets a new maximum value.
Gets the minimum value (default: 0).
Sets a new minimum value.
Gets the orientation value (default: HORIZONTAL).
Sets a new minimum value.
Returns the percent complete for the progress bar.
Returns the progress bar's current value.
Sets the progress bar's current value.
Returns the current value of the progress string.
Sets the value of the progress string.
Returns the value of the stringPainted property.
Sets the value of the stringPainted property, which determines whether the progress bar should render a progress percentage string (default: false).
**Figure 4**

*JProgressBar* is a Swing component with many properties that enable you to customize a progress bar.

Listing 2 gives an example that lets the user specify the number of primes, say \( n \), and displays the first \( n \) primes starting from 2, as shown in Figure 5. The program displays the primes in the text area and updates the completion status in a progress bar.

**Figure 5**
The user enters the number of prime numbers and clicks the Display Prime button to display the primes starting from 2 to the text area.

---

Listing 2 ProgressBarDemo.java

```java
***PD: Please add line numbers in the following code***

<Side Remark line 7: progress bar>
<Side Remark line 13: *JProgressBar* properties UI>
<Side Remark line 17: wrap word>
<Side Remark line 18: wrap line>
<Side Remark line 31: create task>
<Side Remark line 29: add button listener>
<Side Remark line 34: add property listener>
<Side Remark line 37: get property value>
<Side Remark line 42: execute task>
<Side Remark line 48: task class>
<Side Remark line 59: override *doInBackground*
<Side Remark line 66: override *process*
<Side Remark line 72: compute primes>
<Side Remark line 81: set progress property>
<Side Remark line 82: publish a prime>

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

public class ProgressBarDemo extends JApplet {
    private JProgressBar jpb = new JProgressBar();
    private JTextArea jtaResult = new JTextArea();
    private JTextField jtfPrimeCount = new JTextField(8);
    private JButton jbtDisplayPrime = new JButton("Display Prime");

    public ProgressBarDemo() {
        jpb.setStringPainted(true); // Paint the percent in a string
        jpb.setValue(0);
        jpb.setMaximum(100);
        jtaResult.setWrapStyleWord(true);
        jtaResult.setLineWrap(true);
        JPanel panel = new JPanel();
        panel.add(new JLabel("Enter the prime number count");

        override doInBackground() { ...
        override process() { ...

    }
}
```

---

© Copyright Y. Daniel Liang, 2005
panel.add(jtfPrimeCount);
panel.add(jbtDisplayPrime);
add(jpb, BorderLayout.NORTH);
add(new JScrollPane(jtaResult), BorderLayout.CENTER);
add(panel, BorderLayout.SOUTH);

jbtDisplayPrime.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        ComputePrime task = new ComputePrime(
            Integer.parseInt(jtfPrimeCount.getText()), jtaResult);
        task.addPropertyChangeListener(new PropertyChangeListener() {
            public void propertyChange(PropertyChangeEvent e) {
                if ("progress".equals(e.getPropertyName())) {
                    jpb.setValue((Integer)e.getNewValue());
                }
            }
        });
        task.execute(); // Execute SwingWorker
    }
});

/** Task class for SwingWorker */
static class ComputePrime extends SwingWorker<Integer, Integer> {
    private int count;
    private JTextArea result; // Textarea in the UI
    /** Construct a runnable Task */
    public ComputePrime(int count, JTextArea result) {
        this.count = count;
        this.result = result;
    }
    /** Code run on a background thread */
    protected Integer doInBackground() {
        publishPrimeNumbers(count);
        return 0; // doInBackground must return a value
    }
    /** Override process to display published prime values */
    protected void process(java.util.List<Integer> list) {
        for (int i = 0; i < list.size(); i++)
            result.append(list.get(i) + " ");
    }
    /** Publish the first n primes number */
    private void publishPrimeNumbers(int n) {
        int count = 0; // Count the number of prime numbers
        int number = 2; // A number to be tested for primeness
        // Repeatedly find prime numbers
        while (count < n) {
            // Print the prime number and increase the count
            if (isPrime(number)) {
                count++; // Increase the count
                setProgress(100 * count / n); // Update progress
                publish(number); // Publish the prime number
            }
            // Check if the next number is prime
            number++;
        }
        /** Check whether number is prime */
        private static boolean isPrime(int number) {
            for (int divisor = 2; divisor <= number / 2; divisor++) {
                if (number % divisor == 0) { // If true, number is not prime
                    return false; // number is not a prime
                }
            }
            return true; // number is prime
        }
    }
}
The SwingWorker class generates a PropertyChangeEvent whenever the setProgress method is invoked. The setProgress method (line 81) sets a new progress value between 0 and 100. This value is wrapped in the PropertyChangeEvent. The listener of this event can obtain the progress value using the getNewValue() method (line 37). The progress bar is updated using this new progress value (line 37).

The program creates a JProgressBar (line 7) and sets its properties (lines 13–15).

.Side Remark: override doInBackground

The inner class ComputePrime is a SwingWorker (line 48). It overrides the doInBackground method to run publishPrimeNumbers in a background thread (line 60). The publishPrimeNumbers method finds the specified number of primes starting from 2. When a prime is found, the setProgress method is invoked to set a new progress value (line 81). This causes a PropertyChangeEvent to be fired, which is notified to the listener.

.Side Remark: override process

When a prime is found, the publish method is invoked to send the data to the process method (line 82). The process method is overridden (lines 66–69) to display the primes sent from the publish method. The primes are displayed in the text area (line 68).