// Fig. 16.16: RandomCharacters.java
// Class RandomCharacters demonstrates the Runnable interface
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class RandomCharacters extends JApplet implements ActionListener {
    private String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private final static int SIZE = 3;
    private JLabel outputs[];
    private JCheckBox checkboxes[];
    private Thread threads[];
    private boolean suspended[];

    // set up GUI and arrays
    public void init()
    {
        outputs = new JLabel[ SIZE ];
        checkboxes = new JCheckBox[ SIZE ];
        threads = new Thread[ SIZE ];
        suspended = new boolean[ SIZE ];

        Container container = getContentPane();
        container.setLayout( new GridLayout( SIZE, 2, 5, 5 ) );
    }
}
// create GUI components, register listeners and attach components to content pane
for ( int count = 0; count < SIZE; count++ ) {
    outputs[ count ] = new JLabel();
    outputs[ count ].setBackground( Color.GREEN );
    outputs[ count ].setOpaque( true );
    container.add( outputs[ count ] );

    checkboxes[ count ] = new JCheckBox( "Suspended" );
    checkboxes[ count ].addActionListener( this );
    container.add( checkboxes[ count ] );
}

} // end method init

// create and start threads each time start is called (i.e., after init and when user revisits Web page containing this applet)
public void start() {
    for ( int count = 0; count < threads.length; count++ ) {
        // create Thread; initialize object that implements Runnable
        threads[ count ] =
            new Thread( new RunnableObject(), "Thread " + ( count + 1 ) );

        threads[ count ].start(); // begin executing Thread
    }
}
```java
53 }  
54 // determine thread location in threads array
55 private int getIndex( Thread current )  
56 {  
57     for ( int count = 0; count < threads.length; count++ )  
58         if ( current == threads[ count ] )  
59             return count;  
60       
61       return -1;  
62 }  
63 // called when user switches Web pages; stops all threads
64 public synchronized void stop()  
65 {  
66     // set references to null to terminate each thread's run method  
67     for ( int count = 0; count < threads.length; count++ )  
68         threads[ count ] = null;  
69       
70     notifyAll(); // notify all waiting threads, so they can terminate  
71 }  
72 // handle button events
73 public synchronized void actionPerformed( ActionEvent event )  
74 {  
```

**Outline**

RandomCharacter.java

Line 66

Line 70

Line 72

Method `stop` stops all threads

Set thread references in array `threads` to `null`

Invoke method `notifyAll` to ready waiting threads
for (int count = 0; count < checkboxes.length; count++) {
    if (event.getSource() == checkboxes[count]) {
        suspended[count] = !suspended[count];
        // change label color on suspend/resume
        outputs[count].setBackground(
            suspended[count] ? Color.RED : Color.GREEN);
        // if thread resumed, make sure it starts executing
        if (!suspended[count])
            notifyAll();
            return;
    }
} // end method actionPerformed

// private inner class that implements Runnable to control threads
private class RunnableObject implements Runnable {

    // place random characters in GUI, variables currentThread and
    // index are final so can be used in an anonymous inner class
    public void run() {

        // call notifyAll to start ready threads

        Class RunnableObject implements Runnable interface

        Declare method run
```java
// get reference to executing thread
final Thread currentThread = Thread.currentThread();

// determine thread's position in array
final int index = getIndex( currentThread );

// loop condition determines when thread should stop; loop
// terminates when reference threads[index] becomes null
while ( threads[index] == currentThread ) {
    // sleep from 0 to 1 second
    try {
        Thread.sleep( ( int ) ( Math.random() * 1000 ) );
    }
    // determine whether thread should suspend execution;
    // synchronize on RandomCharacters applet object
    synchronized( RandomCharacters.this ) {
        while ( suspended[index] && threads[index] == currentThread ) {
            // temporarily suspend thread execution
            RandomCharacters.this.wait();
        }
    } // end synchronized statement
```
} // end try

// if thread interrupted during wait/sleep, print stack trace
try {
    catch (InterruptedException exception) {
        exception.printStackTrace();
    }
}

// display character on corresponding JLabel
SwingUtilities.invokeLater(
    new Runnable() {
        // pick random character and display it
        public void run() {
            char displayChar =
                alphabet.charAt((int)(Math.random() * 26));

            outputs[index].setText(
                currentThread.getName() + ": " + displayChar);
        }
    }
); // end call to SwingUtilities.invokeLater

Anonymous inner class implements Runnable interface
} // end while

System.err.println( currentThread.getName() + " terminating" );

} // end method run

} // end private inner class RunnableObject

} // end class RandomCharacters