진행 모니터

진행 바는 윈도우내에 위치할 수 있는 아주 단순한 컴포넌트이다. 이에 반해, ProgressMonitor **는 진행 바를 포함하고 있는 완전한 다이얼로그 박스이다(그림**6-38**참조). 다이얼로그는** OK **와** Cancel **버튼을 포함하고 있다. 만일 여러분이 어느 버튼이든 누르면 다이얼로그는 닫혀진다. 게다가, 여러분의 프로그램은 사용자가 다이얼로그를 종료할 것인지 모니터링된 행위를 중단할 것인지 질의할 수 있다(** 클래스 이름이 J**로 시작하지 않는 다는 것에 주의하라).**

그림 6-38 : 진행 모니터 다이얼로그

여러분은 다음 사항들을 제공함으로써 진행 모니터를 구성할 수 있다.

- **팝업될 다이얼로그상의 부모 컴포넌트**
- **다이얼로그상에 디스플레이될 객체(** 문자열이나 아이콘, 컴포넌트여야 함)
- **객체아래 디스플레이될 추가적인 노트**
- **최소와 최대 값**

그러나, 진행 모니터는 그 자체로 진행이나 취소를 알 수 없다. 여러분은 여전히 setProgress **메소드를 호출하여 진행 값을 주지적으로 설정할 필요가 있다.(** 이 메소드는 JProgressBar 클래스의 setValue **메소드와 동일하다.) 여러분이 진행 값을 갱신했을 때, 여러분은 프로그램 사용자가 취소 버튼을 클릭했는지 알기 위해** isCancel **메소드를 호출할 것이다.**

모니터링된 활동이 끝날 때, 여러분은 다이얼로그를 종료하기 위해 close **메소드를 호출한다. 여러분은 다시** start**를 호출하여 같은 다이얼로그를 재사용할 수 있다. 우리는 여전히 모의 활동의 진행상황과 진행 모니터를 갱신을 지켜보기 위해 타이머를 구동시킬 필요가 있다. 다음은 타이머 콜백이다.**

```
public void actionPerformed(ActionEvent event)
        {   // start activity
            activity = new SimulatedActivity(1000);
             activity.start();

            // launch progress dialog
            progressDialog = new ProgressMonitor(
                 ProgressMonitorFrame.this,
                "Waiting for Simulated Activity",
```

```
            null, 0, activity.getTarget());

        // start timer
        activityMonitor.start();

        startButton.setEnabled(false);
    }
```

종료를 위해서는 2개의 조건들이 존재한다. 활동이 완료되거나 사용자가 취소했을 때이다. 각각의 경우에, 우리는 다음을 종료한다.

- 활동을 모니터링하는 타이머
- 진행 다이얼로그
- 활동 그 자체(쓰레드를 종료하는 것에 의해)

만일 예제 6-15의 프로그램을 수행한다면, 여러분은 진행 모니터 다이얼로그의 흥미로운 기능을 볼 수 있다. 다이얼로그는 바로 나타나지 않는다. 대신에, 활동이 이미 완료되었거나 다이얼로그가 나타나는 데 걸리는 시간보다 적은 시간에 완료되는지 살펴보기 위해 잠간 동안 대기한다. setMillisToDecidePopup 메소드를 사용하여 다이얼로그 객체의 구성과 파업을 보여줄 것인지 결정하는 사이에 대기하기 위한 밀리 초 단위의 시간을 설정한다. 기본 값은 500 밀리 초이다. setMillisToPopup은 다이얼로그가 팝 업을 하는 데 필요한 측정 시간이다. 스윙 디자이너들은 이 값을 2초를 기본으로 설정했다. 분명 스윙 다이얼로그가 항상 우리가 원하는 시점에 바로 나타나지 않는 다는 사실을 명심하라. 여러분은 아마도 이 값에 손 댈 필요가 없을 것이다.

예제 6-15는 작동중인 진행 모니터에서 모의 활동의 진행을 측정하는 것을 보여준다. 여러분들이 볼 수 있듯이, 진행 모니터는 사용하기 편리하다. 그리고 오직 여러분은 모니터링하길 원하는 쓰레드에 주기적으로 질의만 하면 된다.

예제 6-15 : ProgressMonitorTest.java

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

public class ProgressMonitorTest
{  public static void main(String[] args)
```

```
   {  JFrame frame = new ProgressMonitorFrame();
      frame.show();
    }
}

class ProgressMonitorFrame extends JFrame
{  public ProgressMonitorFrame()
   {  setTitle("ProgressMonitorTest");
      setSize(300, 200);
      addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
            {  System.exit(0);
            }
         } );

      Container contentPane = getContentPane();

      // this text area holds the activity output
      textArea = new JTextArea();

      // set up a button panel
      JPanel panel = new JPanel();
      startButton = new JButton("Start");
       panel.add(startButton);

      contentPane.add(new JScrollPane(textArea), "Center");
      contentPane.add(panel, "South");

      // set up the button action

       startButton.addActionListener(
          new ActionListener()
             {  public void actionPerformed(ActionEvent event)
                {  // start activity
                   activity = new SimulatedActivity(1000);
                    activity.start();
```

```
                // launch progress dialog
                progressDialog = new ProgressMonitor(
                      ProgressMonitorFrame.this,
                    "Waiting for Simulated Activity",
                     null, 0, activity.getTarget());

                // start timer
                 activityMonitor.start();

                 startButton.setEnabled(false);
              }
           });

  // set up the timer action

   activityMonitor = new Timer(500,
        new ActionListener()
          {  public void actionPerformed(ActionEvent event)
              {  int current = activity.getCurrent();

                 // show progress
                 textArea.append(current + "\n");
                  progressDialog.setProgress(current);

                // check if task is completed or canceled
                  if (current == activity.getTarget()
                      || progressDialog.isCanceled())
                   {  activityMonitor.stop();
                      progressDialog.close();
                      activity.interrupt();
                      startButton.setEnabled(true);
                   }
              }
           });
}
```

```
    private Timer activityMonitor;
    private JButton startButton;
    private ProgressMonitor progressDialog;
    private JTextArea textArea;
    private SimulatedActivity activity;
}

class SimulatedActivity extends Thread
{   public SimulatedActivity(int t)
    {   current = 0;
        target = t;
    }

    public int getTarget()
    {   return target;
    }

    public int getCurrent()
    {   return current;
    }

    public void run()
    {   while (current < target && !interrupted())
        {   try
            {   sleep(100);
            }
            catch(InterruptedException e)
            {   return;
            }
            current++;
        }
    }

    private int current;
    private int target;
```

}

입력 스트림의 진행 상태 모니터링

스윙 패키지는 유용한 스트림 필터로 ProgressMinitorInputStream **을 포함하고 있다. 이 필터는 얼마나 많은 스트림이 읽혀졌는지 모니터링하는 다이얼로그를 자동으로 팝 업하는 데 사용할 수 있다.**

이 필터는 아주 사용하기 쉽다. 여러분은 필터링된 스트림의 일반적인 순서 사이에 ProgressMinitorInputStream**을 끼워 넣는다.(** 스트림에 대한 보다 자세한 정보는 제1권의 12장을 참조하라.)

예를 들면, 여러분이 파일로부터 텍스트를 읽는 다고 가정해 보자. FileInputStream**으로 시작한다:**

```
FileInputStream fileIn = new FileInputStream( f );
```

일반적으로 fileIn**을** InputStreamReader**로 변환한다.**

```
InputStreamReader inReader = new InputStreamReader(fileIn);
```

그러나 스트림을 모니터링 하기위해서, 먼저 파일 입력 스트림을 진행 모니터를 갖는 스트림으로 바꾼다.

```
ProgressMinitorInputStream progressIn
        = new ProgressMinitorInputStream(parent, caption, fileIn);
```

여러분은 부모 컴포넌트와 갭션 그리고 당연히 모니터할 스트림을 제공할 필요가 있다. 진행 모니터 스트림의 read **메소드는 단순히 바이트들을 전달하고 진행 다이얼로그를 갱신한다.**

여러분은 이제 여러분의 필터 순서를 구성한다.

```
InputStreamReader inReader = new InputStreamReader(progressIn);
```

이제 모든 것이 다 끝났다. 파일을 읽을 때, 진행 모니터는 자동으로 팝업된다. 이것은 아주 좋은 스트림 필터링의 응용 프로그램이다.

경고 : 진행 모니터 스트림은 스트림의 총 바이트 숫자를 결정하기 위해 InputStream **클래스의** available **메소드를 사용한다. 그러나** available **메소드는 단지 블록킹되지 않고 이용할 수 있는 스트림내의 총 바이트 숫자를 알려준다. 진행 모니터는 파일들과** HTTP URL**들에 대해**

서는 아주 잘 작동한다. 왜냐하면 이들 파일의 길이는 미리 알려져 있기 때문이다. 그러나 이들 모니터가 모든 스트림들에 대해 작동하는 것은 아니다.

그림 6-39 : 입력 스트림을 위한 진행 모니터

예제 6-16의 프로그램은 파일의 라인들을 카운트한다. 만일 여러분이 큰 파일을 읽는다면, 진행 다이얼로그가 팝업된다.

예제 6-16 : ProgressMonitorInputStreamTest.java

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

public class ProgressMonitorInputStreamTest
{   public static void main(String[] args)
   {   JFrame frame = new TextFrame();
       frame.show();
    }
}

class TextFrame extends JFrame
   implements ActionListener
{   public TextFrame()
    {   setTitle("ProgressMonitorInputStreamTest");
        setSize(300, 200);
        addWindowListener(new WindowAdapter()
           {   public void windowClosing(WindowEvent e)
               {   System.exit(0);
               }
           } );
```

```
    Container contentPane = getContentPane();

   // this text area contains line counts and error messages
    textArea = new JTextArea();
    contentPane.add(new JScrollPane(textArea), "Center");

    // set up menu

   JMenuBar menuBar = new JMenuBar();
     setJMenuBar(menuBar);
   JMenu fileMenu = new JMenu("File");
     menuBar.add(fileMenu);
   openItem = new JMenuItem("Open");
     openItem.addActionListener(this);
     fileMenu.add(openItem);
   exitItem = new JMenuItem("Exit");
     exitItem.addActionListener(this);
     fileMenu.add(exitItem);
}

public void actionPerformed(ActionEvent evt)
{  Object source = evt.getSource();
   if (source == openItem)
   {  // have user select file

       JFileChooser chooser = new JFileChooser();
        chooser.setCurrentDirectory(new File("."));
        chooser.setFileFilter(
            new javax.swing.filechooser.FileFilter()
           {  public boolean accept(File f)
              {  String fname = f.getName().toLowerCase();
                 return fname.endsWith(".txt")
                    || f.isDirectory();
              }
              public String getDescription()
             { return "Text Files"; }
```

```
            });
        int r = chooser.showOpenDialog(this);
        if (r == JFileChooser.APPROVE_OPTION)
            readFile(chooser.getSelectedFile());
      }
    else if (source == exitItem)
         System.exit(0);
  }

public void readFile(final File f)
{  /* important: the monitored activity must be in a new
       thread. We define a thread class on the fly; the thread
        action is in the run method
     */

    Thread readThread = new Thread()
     {   public void run()
         {   try
          {  // set up stream and reader filter sequence

              FileInputStream fileIn = new FileInputStream(f);
               ProgressMonitorInputStream progressIn
                  = new ProgressMonitorInputStream(TextFrame.this,
                     "Reading " + f.getName(), fileIn);
               InputStreamReader inReader
                  = new InputStreamReader(progressIn);
              BufferedReader in = new BufferedReader(inReader);

             // read file and count lines

              int count = 0;
               String line;
             while ((line = in.readLine()) != null)
                   count++;
            textArea.append(f + ": " + count + " lines\n");
               fileIn.close();
```

```
                }
                catch(IOException e)
               {   textArea.append("Error " + e + "\n");
                }
            }
        };

        readThread.start();
    }

    private JTextArea textArea;
    private JMenuItem openItem;
    private JMenuItem exitItem;
}
```

javax.swing.JProgressBar

- JProgressBar()
- JProgressBar(int direction)
- JProgressBar(int min, int max)
- JProgressBar(int direction, int min, int max)

  주어진 방향, **최소**,**최대**, **초기 값을 갖는 수평 슬라이더를 구성한다**.

  전달인자 : direction SwingConstants.HORIZONTAL **이나** SwingConstants.VERTICAL**중의 하나**. **기본은 수평이다**.

  min, max **진행 바의 최소와 최대 값**. **기본은** 0**과** 100**이다**.

- int getMinimum()
- int getMaximum()
- void setMinimum(int value)
- void setMaximum(int value)

  최소와 최대 값을 가져오거나 설정한다.

- int getValue()
- void setValue(int value)

  현재 값을 가져오거나 설정한다.

- string getString()
- void setString(String s)

  진행 바에 디스플레이 할 문자열을 가져오거나 설정한다. **만일 문자열이** null**이면** ,

기본 문자열로 “n%”가 디스플레이 된다.

- boolean isStringPainted()
- void setStringPainted(boolean b)

  “String painted” 속성을 가져오거나 설정한다. 만일 이 속성이 true이면 문자열은 진행 바의 상단에 출력된다. 기본은 false이고 출력되는 문자열은 없다.

Javax.swing.ProgressMonitor

- ProgressMonitor(Component parent, Object message, String note, int min, int max)

  진행 모니터 다이얼로그를 구성한다.

전달인자:	parent	해당 다이얼로그 팝업이 놓여질 부모 컴포넌트
	message	다이얼로그에 디스플레이될 메시지 객체
	note	메시지 아래에 디스플레이될 선택 사항인 문자열, 만일 이 값이 null이면, 노트를 위한 별도의 공백은 설정되지 않는다. 그리고 나중에 setNote를 호출하여 효과를 없앤다.
	min, max	진행 바의 최소, 최대 값

- void setNote(String note)

  노트 텍스트를 변경한다.

- void setProgress(int value)

  진행 바 값을 주어진 값으로 설정한다.

- void close()

  해당 다이얼로그를 닫는다.

- boolean isCanceled()

  사용자가 해당 다이얼로그를 취소했다면 true를 반환한다.

javax.swing.ProgressMonitorInputStream

- ProgressMonitorInputStream(Component parent, Object message, InputStream in)

  진행 모니터 다이얼로그와 연관된 입력 스트림 필터를 구성한다.

전달인자:	parent	해당 다이얼로그 팝업이 놓여질 부모 컴포넌트
	message	다이얼로그에 디스플레이될 메시지 객체
	in	모니터링될 입력 스트림

툴 바와 툴 팁

툴 바는 프로그램에서 가장 공통적으로 사용되는 명령어들을 빠르게 접근하도록 해주는 버

튼 바이다(그림6-40).

그림 6-40 : 툴 바

툴 바를 특별하게 만드는 것은 이들을 임의의 위치로 옮길 수 있다는 것이다. 여러분은 툴 바를 프레임의 4 경계중 하나로 드레깅 할 수 있다(그림 6-41 참조). 여러분이 마우스 버튼을 해제할 때, 툴 바는 새로운 위치에 놓여진다(그림 6-42 참조).

노트 : 만일 툴 바가 보더 레이아웃을 갖는 컨테이너 내부나 "North", "East", "South" 그리고 "West" 제약을 지원하는 다른 모든 레이아웃 관리자 내부에 있다면 툴 바 드레깅은 작동한다.

그림 6-41 : 툴 바 드레깅

그림 6-42 : 툴 바를 다른 보더로 드레깅

심지어 툴 바는 프레임으로부터 완전히 분리될 수 있다. 분리된 툴 바는 자신의 프레임 내부에 포함된다(그림 6-43 참조). 여러분이 분리된 툴 바를 포함하고 있는 프레임을 닫을 때, 툴 바는 본래의 프레임안으로 복귀한다.

그림 6-43 : 툴 바 떼어내기

툴 바는 프로그램 하기 쉽다. 여러분은 툴 바에 컴포넌트들을 추가한다.

```
JToolBar ber = new JToolBar();
bar.add(blueButton);
```

여러분은 분리자로 일련의 버튼 그룹을 분리할 수 있다.

```
bar.addSeperator();
```

가령, 그림 6-40의 툴 바는 3번째와 4번째 버튼 사이에 분리자를 가지고 있다.

그런 다음, 컨테이너에 툴 바를 추가한다.

```
contentPane.add(bar, “North”);
```

대부분의 툴 바가 버튼들로 구성되지만, 툴 바에 추가할 수 있는 컴포넌트에 대한 제약은 없다. 예를 들면, 툴 바 내부에 컴보 박스들은 아주 일반적이다.

주의 : JDK1.2에서, 툴 바 버튼은 여러분들이 윈도우 룩앤필을 사용할 때 아주 널리 사용된다. 툴 바에 추가할 각 버튼에 대한 호출의 개선 방법은 다음과 같다.

```
button.setMargin(new Inset(0, 0, 0, 0) );
```

만일 여러분이 1권의 8장을 보았다면, 액션과 이들을 유발하는 사용자 인터페이스를 분리하기 위해 Action 인터페이스의 사용을 추천한 것을 상기할 것이다. Action 인터페이스는 ActionListener 인터페이스를 확장한다. 그리고 이름과 아이콘 그리고 도움 정보를 유지하기 위해 작은 룩업 테이블을 지원한다. AbstractAction 클래스는 Action 인터페이스의 모든 메소드들을 구현한다. 단 하위 클래스에 여러분이 제공해야 하는 Action 인터페이스의 actionPerformed 메소드는 예외이다. 다음은 기본적인 액션이다:

```
Action exitAction
        = new AbstractAction("Exit", new ImageIcon("exit.gif"))
            {   public void actionPerformed(ActionEvent event)
                {   System.exit(0);
                }
             };
```

여러분은 메뉴들에 Action 객체들을 추가할 수 있다:

```
menu.add(exitAction);
```

이 연산은 자동으로 주어진 아이템이 선택됐을 때 자동으로 호출되는 actionPerformed 메소드와 이름, 아이콘을 갖는 메뉴 아이템을 생성한다. 1권의 8장은 여러분에게 같은 액션 객체를 키스트로크에 부착하는 방법을 보여준다.

또한 JToolBar 클래스는 액션을 추가하기 위한 메소드를 가지고 있다. 불운하게도, 액션 이름이 아이콘 아래에 디스플레이 되기 때문에 최종 툴 바는 다소 매력적으로 보이지 않는다 (그림 6-44 참조).

그림 6-44 : 액션들을 갖는 툴 바

희망적인 것은, 스윙의 향후 버전에는 이러한 액션 이름을 가리는 옵션이 툴 바에 추가될 것이라는 사실이다. 현재로서, 우리는 여러분에게 툴 바 버튼을 액션없이 만들기 위해 도움 클래스를 사용하길 제안한다.

```
class ToolBarButton extends JButton
{  public ToolBarButton(Action a)
   {   super((Icon)a.getValue(Action.SMALL_ICON));
       addActionListener(a);
   }
}
```

그런 다음, 여러분은 이들 버튼을 갖는 툴 바를 구동할 수 있다.

```
bar.add( new ToolBatButton(exitAction) );
```

툴 팁들

툴 바에 단점은 사용자가 종종 툴 바내에 작은 아이콘들의 의미를 혼돈한다는 것이다. 이 문제를 풀기위해, 툴 팁이 고안되었다. 툴 팁은 커져가 버튼상에 놓일 때 작동된다. 툴 팁 텍스트는 사각형 내부에서 디스플레이 된다. 사용자가 마우스를 바깥으로 이동할 때, 툴 팁은 제거된다.

그림 6-45 : 툴 팁

스윙에서, 여러분은 setToolTipText **메소드를 호출하여 모든** JComponent**에 툴 팁을 추가할 수 있다**:

```
exitButton.setToolTipText("Exit");
```

예제 6-17**의** ToolBarButton **클래스는 액션 테이블에서 적합한 툴 바 팁을 찾기 위한 시도를 한다. 만일 액션이** Action.SHORT_DESCRIPTION **키와 연관된 값을 가지고 있다면 , 그 값이 사용된다. 만일 없다면, 액션 이름이 툴 팁에 디스플레이 된다.**

예제 6-17**은 같은 메뉴와 툴 바에** Action **객체들을 추가하는 방법을 보여주는 프로그램이다.**

액션 이름들은 메뉴상에서 메뉴 아이템 이름으로, 툴 바에서 툴 팁으로 나타난다.

예제 6-17 : ToolBatTest.java

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

public class ToolBarTest
{  public static void main(String[] args)
   {  JFrame frame = new ToolBarFrame();
      frame.show();
   }
}

/* the color action sets the background of its target to a
   given color
*/

class ColorAction extends AbstractAction
{  public ColorAction(String name, Icon icon,
      Color c, Component t)
   {  putValue(Action.NAME, name);
      putValue(Action.SMALL_ICON, icon);
      putValue(Action.SHORT_DESCRIPTION, name + " background");
      putValue("Color", c);
      target = t;
   }

   public void actionPerformed(ActionEvent evt)
   {  Color c = (Color)getValue("Color");
      target.setBackground(c);
      target.repaint();
   }
```

```
    private Component target;
}

/* the tool bar button is a button with an icon and no text
   suitable for addition into a tool bar. The tool tip is set
   to the short description of the action, or to the name
   if the short description is not available
*/

class ToolBarButton extends JButton
{   public ToolBarButton(Action a)
    {   super((Icon)a.getValue(Action.SMALL_ICON));

        String toolTip
            = (String)a.getValue(Action.SHORT_DESCRIPTION);
        if (toolTip == null)
            toolTip = (String)a.getValue(Action.NAME);
        if (toolTip != null)
            setToolTipText(toolTip);
        addActionListener(a);
    }
}

class ToolBarFrame extends JFrame
{   public ToolBarFrame()
    {   setTitle("ToolBarTest");
        setSize(300, 200);
        addWindowListener(new WindowAdapter()
            {   public void windowClosing(WindowEvent e)
                {   System.exit(0);
                }
            } );

       // add a panel for color change

        Container contentPane = getContentPane();
```

```
JPanel panel = new JPanel();
 contentPane.add(panel, "Center");

// set up actions

Action blueAction = new ColorAction("Blue",
      new ImageIcon("blue-ball.gif"),
      Color.blue, panel);
Action yellowAction = new ColorAction("Yellow",
      new ImageIcon("yellow-ball.gif"),
      Color.yellow, panel);
Action redAction = new ColorAction("Red",
      new ImageIcon("red-ball.gif"),
      Color.red, panel);

 Action exitAction
    = new AbstractAction("Exit", new ImageIcon("exit.gif"))
        {   public void actionPerformed(ActionEvent event)
              {   System.exit(0);
              }
         };

// populate tool bar

JToolBar bar = new JToolBar();
 bar.add(new ToolBarButton(blueAction));
  bar.add(new ToolBarButton(yellowAction));
 bar.add(new ToolBarButton(redAction));
  bar.addSeparator();
 bar.add(new ToolBarButton(exitAction));
 contentPane.add(bar, "North");

 // populate menu

JMenu menu = new JMenu("Color");
  menu.add(yellowAction);
```

```
        menu.add(blueAction);
        menu.add(redAction);
        menu.add(exitAction);
      JMenuBar menuBar = new JMenuBar();
        menuBar.add(menu);
        setJMenuBar(menuBar);
    }
}
```

javax.swing.JToolBar

- void add(Action a)
  주어진 액션으로부터 이름, 아이콘, 그리고 액션 콜백을 갖는 새로운 버튼을 툴 바에 생성한다. 그리고 버튼을 툴 바의 끝에 추가한다. 이름이 아이콘 아래에 디스플레이 된다는 것을 기억하라.
- void addSeperator()
  툴 바의 끝에 분리자를 추가한다.

javax.swing.JComponent

- void setToolTipText(String text)
  마우스를 컴포넌트 위에 위치할 때 툴 팁으로서 디스플레이 될 텍스트를 설정한다.

컴포넌트 구성

우리는 다른 컴포넌트들을 구성하는 데 도움을 주는 컴포넌트들의 표현을 갖는 고급 스윙 기능들을 논의하면서 마무리 한다. 이들에는 영역을 다중 부분으로 나누는 방법인 스플릿 패인(split pane), **다중 패널들을 사용자가 탭으로 펼칠 수 있게 해주는 탭있는 패인**(tabbed pane), **다중 내부 프레임**(internal frame)**을 출력해주는 응용 프로그램을 구현하는 데 사용하는 데스크탑 페인**(desktop pane)**등 이 있다.**

분할 패인

분할 패인은 컴포넌트를 두개의 부분으로 나눌 때 사용한다. 그림 6-46**은 두개의 분할 패인을 갖는 프레임웍을 보여준다. 바깥쪽의 패인은 수평으로 나뉘었고 버튼 상에 텍스트 영역을 갖고, 상단에 다른 분할 패인이 있는 레벨을 가지고 있다.**

그림 6- 46 : 두개의 중첩된 분할 패인을 갖는 프레임

여러분은 방향을 명시함으로써 분할 패인을 구성할 수 있다. JSplitPane.HORIZONTAL_SPLIT ,

JSplitPane.VERTICAL_SPLIT, 예를 들면,

```
JSplitPane innerPane = new JSplitPane(JSplitPane. HORIZONTAL_SPLIT, planetList, planetImage);
```

이것이 여러분이 해야 할 모든 것들이다. 만일 여러분이 원한다면, 분리바에 "한번에 확장(one touch expand)" 아이콘을 추가할 수 있다. 여러분은 이들 아이콘들을 그림 6-46의 상위 패인에서 볼 수 있다. 만일 여러분이 이들중 하나를 클릭한다면, 분리자는 항상 사각형의 팁 방향으로 움직인다. 패인들중의 하나를 완전하게 확장한다.

이러한 기능을 추가하려면, 다음을 호출한다.

```
innerPane.setOneTouchExpandable(true);
```

"레이아웃 지속(continuous layout)" 기능은 양쪽 컴포넌트의 내용을 계속해서 다시 그리는 기능이다. 보다 최신의 룩을 볼 수 있지만 느릴 수 있다. 다음 호출을 사용하여 이 기능을 가동시킬 수 있다.

```
innerPane.setContinuousLayout(true);
```

예제 프로그램에서, 우리는 기본으로 하부 분리자를 남겨 두었다. 여러분이 이를 드레그할 때, 오직 검은 윤곽만 이동한다. 여러분이 작업을 끝냈을 때, 컴포넌트들은 다시 그려진다.

예제 6-18의 프로그램은 아주 간결한 구조를 갖는다. 행성들을 나열하는 리스트 박스를 보여준다. 사용자가 선택할 때, 행성 이미지가 오른쪽에 출력되고 하단부의 텍스트 영역에 설명이 위치한다. 여러분이 프로그램을 수행할 때, 분리자들을 정리하고 "한번에 확장" 기능과 "레이아웃 지속"기능을 시도해 보라.

예제 6-18 : SplitPaneTest.java

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

public class SplitPaneTest
```

```
{  public static void main(String[] args)
   {   JFrame frame = new SplitPaneFrame();
       frame.show();
   }
}

class SplitPaneFrame extends JFrame
   implements ListSelectionListener
{  public SplitPaneFrame()
   {   setTitle("SplitPaneTest");
       setSize(400, 300);
       addWindowListener(new WindowAdapter()
          {  public void windowClosing(WindowEvent e)
             {   System.exit(0);
             }
          } );

      // set up components for planet names, images, descriptions

       planetList = new JList(planets);
        planetList.addListSelectionListener(this);

       planetImage = new JLabel();

       description = new JTextArea();

      // set up split panes

        JSplitPane innerPane
          = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
              planetList, planetImage);

        innerPane.setContinuousLayout(true);
        innerPane.setOneTouchExpandable(true);

       JSplitPane outerPane
```

```
            = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                innerPane, description);

        getContentPane().add(outerPane, "Center");
    }

    public void valueChanged(ListSelectionEvent event)
    {   JList source = (JList)event.getSource();
        Planet value = (Planet)source.getSelectedValue();

        // update image and description

        planetImage.setIcon(value.getImage());
        description.setText(value.getDescription());
    }

    private JList planetList;
    private JLabel planetImage;
    private JTextArea description;
    private Planet[] planets =
        {   new Planet("Mercury", 2440, 0),
            new Planet("Venus", 6052, 0),
            new Planet("Earth", 6378, 1),
            new Planet("Mars", 3397, 2),
            new Planet("Jupiter", 71492, 16),
            new Planet("Saturn", 60268, 18),
            new Planet("Uranus", 25559, 17),
            new Planet("Neptune", 24766, 8),
            new Planet("Pluto", 1137, 1),
        };
}

class Planet
{  public Planet(String n, double r, int m)
    {   name = n;
        radius = r;
```

```
        moons = m;
        image = new ImageIcon(name + ".gif");
    }

    public String toString()
    {   return name;
    }

    public String getDescription()
    {   return "Radius: " + radius + "\nMoons: " + moons + "\n";
    }

    public ImageIcon getImage()
    {   return image;
    }

    private String name;
    private double radius;
    private int moons;
    private ImageIcon image;
}
```

javax.swing.JSplitPane

- JSplitPane()
- JSplitPane(int direction)
- JSplitPane(int direction , boolean continuousLayout)
- JSplitPane(int direction , Component first, Component second)
- JSplitPane(int direction , boolean continuousLayout ,Component first, Component second)

  새로운 분할 패인을 생성한다.

  전달인자: direction HORIZONTAL_SPLIT **이나** VERTICAL_SPLIT **중의 하나**

  continuousLayout **분리자가 이동했을 때 컴포넌트들이 지속적으로 갱신된다면** true**이다**.

  first, second **추가될 컴포넌트**

- boolean isOneTouchExpandable()

- void setOneTouchExpandable(Boolean b)
  "한번에 확장(one touch expand)" 속성을 가져오거나 설정한다. 이 속성이 설정될 때, 분리자는 완전히 확장하는 것과 컴포넌트를 확장하는 것 , 두개의 아이콘을 갖는다.
- boolean isContonuousLayout()
- void setContonuousLayout(Boolean b)
  "레이아웃 지속(continuous layout)" 속성을 가져오거나 설정한다. 이 속성이 설정될 때, 컴포넌트들은 분리자가 이동할 때 지속적으로 갱신된다.
- void setLeftComponent(Component c)
- void setTopComponent(Component c)
  분할 패인내에 첫번째 컴포넌트로서 c를 설정하는 같은 효과를 제공하는 메소드들이다.
- void setRightComponent(Component c)
- void setBottomComponent(Component c)
  분할 패인내에 두번째 컴포넌트로서 c를 설정하는 같은 효과를 제공하는 메소드들이다.

탭 달린 패인

탭 달린 패인은 복잡한 다이얼로그를 연관된 옵션들의 하위 셋으로 쪼개기 위한 친숙한 사용자 인터페이스 디바이스이다. 또한 여러분은 탭을 사용하여 사용자들로 하여금 일련의 문서들이나 이미지들을 통해 넘길 수 있게 할 수 있다(그림 6-47).

그림 6-47 : 탭 달린 패인

탭 달린 패인을 생성하기 위해, 여러분은 먼저 JtabbedPane 객체를 생성해야 한다. 그런 다음, 여기에 탭을 추가한다.

```
JTabbedPane tabbedPane = new JTabbedPane();
tabbedPane.addTeb(title, icon, component);
```

addTab 메소드의 마지막 전달인자는 Component 타입이다. 만일 여러분이 복수개의 컴포넌트를 같은 탭에 추가하길 원한다면, 먼저 JPanel같은 컴테이너에 이들을 담어야 한다.

아이콘은 선택사항이다. addTab 메소드는 아이콘을 요구하지 않는다.

```
tabbedPane.addTeb(title,   component);
```

탭 컬렉션으로부터 한 탭을 제거할 때는 다음을 사용한다.

```
tabPane.removeTabAt(index);
```

여러분이 새로운 탭을 탭 컬렉션에 추가할 때, 자동적으로 디스플레이 되지 않는다. 여러분은 setSelectedIndex 메소드로 이를 선택해야 한다. 가령, 다음은 방금 최종적으로 추가한 탭을 보여주는 방법이다.

```
tabbedPane.setSelectedIndex (tabbedPane.getTabCount() -1);
```

예제 프로그램은 탭있는 패인으로 유용한 기술을 보여준다. 때때로, 여러분은 컴포넌트를 디스플레이하기 전에 컴포넌트를 갱신하길 원한다. 우리의 예제 프로그램에서, 사용자가 실제 탭상에 클릭했을 때만 행성 이미지를 로드할 것이다.

사용자가 새로운 탭상에 클릭했을 때마다 통보하기 위해, 여러분은 탭있는 패인에 ChangeListener를 설치해야 한다. 컴포넌트들이 아니라 탭있는 패인 그 자체에 리스너를 설치해야 한다는 사실에 주의하라.

```
tabbedPane.addChangeListener (listener);
```

사용자가 탭을 선택할 때, 변경 리스너의 stateChange 메소드가 호출된다. 여러분은 이벤트의 소스로서 탭있는 패인을 추출한다. 어느 패인이 선택될 것인지 발견하기 위해 getSelectedIndex 메소드를 호출한다.

```
public void stateChanged(ChangeEvent event)
{       JTabbedPane pane = (JTabbedPane)event.getSource();
        int n = pane.getSelectedIndex();
        …
}
```

예제 6-19 에서, 우리는 먼저 모든 탭 컴포넌트들을 null로 설정한다. 새로운 탭이 선택됐을 Eom 우리는 해당 컴포넌트가 여전히 null인지 검사한다. 만일 null이라면, 이를 이미지로 대체한다. (이것은 사용자가 탭상에 클릭했을 때 계속해서 발생한다. 여러분이 비어있는 패인을 보지는 않을 것이다.) 재미를 위해, 우리는 또한 방문한 패인들을 나타내기 위해 아이콘을 노란 볼에서 붉은 볼로 변경할 것이다.

예제 6-19 : TabbedPaneTest.java

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

public class TabbedPaneTest
{  public static void main(String[] args)
   {  JFrame frame = new TabbedPaneFrame();
      frame.show();
   }
}

class TabbedPaneFrame extends JFrame
   implements ChangeListener
{  public TabbedPaneFrame()
   {  setTitle("TabbedPaneTest");
      setSize(400, 300);
      addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
            {  System.exit(0);
            }
         } );

      tabbedPane = new JTabbedPane();
      tabbedPane.addChangeListener(this);

      /* we set the components to null and delay their
         loading until the tab is shown for the first time
      */

      ImageIcon icon = new ImageIcon("yellow-ball.gif");

      tabbedPane.addTab("Mercury", icon, null);
      tabbedPane.addTab("Venus", icon, null);
```

```
        tabbedPane.addTab("Earth", icon, null);
        tabbedPane.addTab("Mars", icon, null);
        tabbedPane.addTab("Jupiter", icon, null);
        tabbedPane.addTab("Saturn", icon, null);
        tabbedPane.addTab("Uranus", icon, null);
        tabbedPane.addTab("Neptune", icon, null);
        tabbedPane.addTab("Pluto", icon, null);

         getContentPane().add(tabbedPane, "Center");
    }

   public void stateChanged(ChangeEvent event)
   {  JTabbedPane pane = (JTabbedPane)event.getSource();

     // check if this tab still has a null component

      if (pane.getSelectedComponent() == null)
       {   // set the component to the image icon

           int n = pane.getSelectedIndex();
           String title = pane.getTitleAt(n);
          ImageIcon planetIcon = new ImageIcon(title + ".gif");
           pane.setComponentAt(n, new JLabel(planetIcon));

           // indicate that this tab has been visited--just for fun

           pane.setIconAt(n, new ImageIcon("red-ball.gif"));
       }
    }

   private JTabbedPane tabbedPane;
}
```

javax.swing.JTabbedPane

- JTabbedPane()
- JTabbedPane(int placement)

탭 있는 패인을 생성한다.

전달인자: placement swingconstants.TOP, swingconstants.LEFT, swingconstants.RIGHT **또는** swingconstants.BUTTOM**중의 하나이다.**

- void addTab(String Title, Component component)
- void addTab(String Title, Icon icon, Component component)
- void addTab(String Title, Icon icon, Component component, String tooltip)

  탭 있는 패인 끝에 탭을 하나 추가한다.
- void addTab(String Title, Icon icon, Component component, String tooltip, int index)

  탭있는 패인의 주어진 인덱스 위치에 탭을 하나 추가한다.
- void removeTabAt(int index)

  주어진 인덱스에 위치한 탭을 제거한다.
- void setSelectedIndex(int index)

  선택된 탭의 인덱스를 선택한다.
- int getSelectedIndex()

  선택된 탭의 인덱스를 반환한다.
- Component getSelectedComponent()

  컴포넌트를 반환한다.
- String getTitleAt(int index)
- void setTitleAt(int index, String title)
- Icon getIconAt(int index)
- Void setIconAt(int index, Icon icon)
- Component getComponentAt(int index)
- void setComponentAt(int index, Component c)

  주어진 인덱스에 있는 컴포넌트나 제목, 아이콘을 가져오거나 설정한다.
- int indexOfTab(Icon icon)
- int indexOfTab(String title)
- int indexOfComponent(Component c)

` **주어진 제목, 아이콘, 또는 컴포넌트를 갖는 탭의 인덱스를 반환한다.**
- int getTabCount()

  해당 탭있는 패인의 전체 탭 숫자를 반환한다.
- void addChangeListener(ChangeListener listener)

  사용자가 다른 탭을 선택할 때 통보받는 변경 리스너를 추가한다.

데스크 탑 패인과 내부 프레임들

많은 응용 프로그램들은 큰 프레임 내부에 모두 포함되는 복수 개의 윈도우에 정보를 제공한다. 만일 응용 프로그램 프레임을 최소화한다면, 프레임내에 모든 윈도우가 동시에 숨겨진

다. 윈도우 환경하에서, 이러한 사용자 인터페이스를 다중 도큐먼트 인터페이스, MDI(Multiple Document Interface)라 부른다. 그림 6-48은 이러한 인터페이스를 사용하는 기본적인 응용 프로그램을 보여준다.

그림 6-48 : 다중 도큐먼트 인터페이스 응용 프로그램

한 동안, 이러한 사용자 인터페이스 스타일은 아주 유명했었다. 그러나 최근에 들어 그다지 널리 사용되지 않았다. 특히, 대부분의 브라우져는 별도의 최상위 프레임에 각 웹 페이지를 디스플레이한다.(눈에 띠는 예외상황으로 그림 6-46의 오페라 브라우져가 있다.) 어느 것이 더 좋을 까? 이 두 접근방법에는 장,단점이 모두 존재한다. MDI는 윈도우가 어지럽게 되는 것을 줄여준다. 그러나 별도의 최상위 윈도우를 갖는 것은 윈도우들을 통해 이동하기 위해 호스트 윈도우 시스템의 핫 키와 버튼들을 사용할 수 있다는 것을 의미한다.

자바 세계에서는 풍부한 호스트 윈도우 시스템들에 의존할 수 없다. 이것은 여러분의 응용 프로그램이 해당 프레임들을 관리하는데 많은 노력을 기울이게 한다.

그림 6-49는 세 개의 내부 프레임들을 갖는 자바 응용 프로그램을 보여준다. 이들 중 두개는 보더상에 최대화와 아이콘화 기능을 갖고 있다. 세 번째는 아이콘화된 상태이다.

메탈 룩앤필로 보면, 내부 프레임은 프레임들을 주변으로 이동하는 데 사용하는 "그래버(grabber)" 영역을 가지고 있다. 여러분은 구석에 있는 리사이즈를 드레깅해서 윈도우의 크기를 조절할 수 있다.

이러한 기능을 부여하기 위해, 다음의 단계들을 따라야 한다.

1. 응용 프로그램을 위해 일반 JFrame 윈도우를 사용한다.
2. JDesktopPane에 대한 JFrame의 컨텐트 경로를 설정한다.

   ```
   desktop = new JDesktopPane()
   setContentPane(desktop);
   ```

3. JInternalFrame 윈도우를 구성한다. 여러분은 리사이징 아이콘이건 프레임 닫기건 원하는 대로 명시할 수 있다. 일반적으로 여러분은 모든 아이콘을 원한다.

   ```
   JInternalFrame iframe = new JInternalFrame(title ,
           true, // resizable
   ```

```
        true, // closable
        true, // maximizable
        true ); // iconifiable
```

4. 프레임의 컨텐트 패인에 컴포넌트들을 추가한다.

   iframe.getContentPane().add(c);

5. 프레임 아이콘을 설정한다. 아이콘은 프레임웍 상단 좌측 코너에 나타난다.

   iframe.setFrameIcon(icon);

노트 : 메탈 룩앤필의 현재 버전에서 프레임 아이콘은 아이콘화된 프레임에 나타나지 않는다.

---

6. 내부 프레임의 크기를 설정한다. 일반 프레임으로서, 내부 프레임은 초기 0 by 0 픽셀 크기를 갖는다. 여러분이 모든 내부 프레임들을 각기 상단에 디스플레이되길 원하지 않기 때문에, 여러분은 다음 프레임에 대한 가변 위치를 사용해야 한다. 프레임의 위치와 크기 모두를 설정하기 위해 reshape 메소드를 사용한다.

   iframe.reshape(nextFrameX, nextFrameY, width, height);

7. Jframes으로 프레임을 가시적으로 만들 필요가 있다.

   iframe.setVisible(true);

노트 : 스윙의 초기 버전에서, 내부 프레임은 자동적으로 가시화되었고 이 호출은 필요하지 않았다.

---

8. JDesktopPane에 프레임을 추가한다.

   desktop.add(iframe);

9. 여러분은 아마도 새로운 프레임을 ***선택된 프레임****(selected frame)*으로 만들길 원할 것이다. 데스크 탑상의 내부 프레임들에서, 오직 선택된 프레임만이 키보드 포커스를 받는다. 메탈 룩앤필에서, 선택된 프레임은 파란 제목 바를 가지고 있다. 반면에 다른 프레임들은 회색 제목 바를 가지고 있다. 여러분은 프레임을 선택하기 위해 setSelected 메소드를 사용한다. 그러나, "선택된" 속성은 거부(vetoed)될 수 있다-현재 선택된 프레임은 포커스를 포기하길 거부할 수 있다. 이 경우, setSelected 메소드는 이를 여러분에게 처리하도록 하기 위해 PropertyVetoException을 발생시킨다.

```
try
{       iframe.setSelected(true);
}
```

```
catch(PropertyVetoException e)
{           // attempt was vetoed
}
```

10. 여러분은 아마도 기존의 프레임과 중첩되는 걸 원하지 않기 때문에 다음의 내부 프레임의 위치를 아래로 이동하길 원할 것이다. 프레임들간의 좋은 구별 방법은 제목바의 높이이다. 이 높이는 다음처럼 구할 수 있다.

```
int frameDistance = iframe.getHeight() – iframe.getContentPane().getHeight()
```

11. 다음 내부 프레임 위치를 결정하기 위해 구한 거리를 사용한다.

```
nextFrameX += frameDistance;
nextFrameY += frameDistance;
if ( nextFrameX + width > desktop.getWidth() )
        nextFrameX = 0;
if ( nextFrameY + width > desktop.getHeight() )
        nextFrameY = 0;
```

캐스캐이딩과 틸링(tiling)

윈도우에서, 윈도우의 캐스캐이딩과 틸링은 표준 명령어이다(그림 –50과 6-51). 자바 JDesktopPane과 JInternalFrame 클래스들은 이들 기능을 지원하지 않는다. 예제 6-20에서, 우리는 이들 기능들을 직접 구현하는 방법을 보여준다.

모든 윈도우를 캐스캐이딩하기 위해, 여러분은 윈도우들을 같은 크기로 재조정하고 그들의 위치를 배치해야 한다. JDesktopPane 클래스의 getAllFrames 메소드는 모든 내부 프레임들의 배열을 반환한다.

```
JInternalFrame[] frames = desktop. getAllFrames();
```

그러나, 여러분은 프레임의 상태에 주의를 기울여야 한다. 내부 프레임은 다음 상태들중 하나일 수 있다.

- 아이콘(Icon)
- 크기조정 가능(Resizable)
- 최대화(Maximum)

여러분은 현재 어느 내부 프레임들이 아이콘화 되어 있는지 찾기 위해 isIcon 메소드를 사용

한다. 그러나 만일 프레임이 최대화 상태이면, 여러분은 먼저 setMaximum(false)를 호출하여 해당 프레임을 크기 조정 가능하게 설정해야 한다. 이것은 거부할 수 있는 또 다른 속성이다. 그러므로 여러분은 PropertyVetoException을 감지해야 한다.

```
for (int i = 0; i < frames.length; i++)
      {  if (!frames[i].isIcon())
         {  try
           {  /* try to make maximized frames resizable
                 this might be vetoed
              */
              frames[i].setMaximum(false);
             frames[i].reshape(x, y, width, height);

              x += frameDistance;
              y += frameDistance;
            // wrap around at the desktop edge
            if (x + width > desktop.getWidth()) x = 0;
            if (y + height > desktop.getHeight()) y = 0;
            }
           catch(PropertyVetoException e)
            {}
         }
      }
```

그림 6-50 : 캐스캐이드된 내부 프레임들

그림 6-51 : 타일된 내부 프레임들

타일링된 프레임들은 다소 애매하다. 특히, 프레임의 숫자가 완전히 제곱한 수가 아닐 때 그러하다. 먼저, 아이콘이 아닌 프레임들의 숫자를 계산한다. 그런 다음, 다음처럼 컬럼의 숫자를 계산한다.

```
int cols = (int)Math.sqrt(frameCount);
```

그런 다음, 열의 숫자를 구한다.

```
int rows = frameCount / cols;
```

마지막 컬럼들이 갖는 rows + 1 열을 제외한다.

```
extra = framwCount % cols;
```

다음은 데스크 탑상에 모든 프레임들을 타일링하는 루프이다.

```
int width = desktop.getWidth() / cols;
      int height = desktop.getHeight() / rows;
      int r = 0;
      int c = 0;
      for (int i = 0; i < frames.length; i++)
       {  if (!frames[i].isIcon())
          {  try
             {  frames[i].setMaximum(false);
                frames[i].reshape(c * width,
                   r * height, width, height);
                r++;
                if (r == rows)
                {  r = 0;
                   c++;
                   if (c == cols - extra)
                   {  // start adding an extra row
                      rows++;
                      height = desktop.getHeight() / rows;
                   }
                }
             }
             catch(PropertyVetoException e)
             {}
          }
```

```
    }
```

예제 프로그램은 다른 일반 프레임 연산을 보여준다 : 현재 프레임으로부터 아이콘화 되지 않은 다른 프레임으로 선택을 이동한다. JDesktopPane**클래스는 선택된 프레임을 반환하는 메소드가 없다. 대신에, 여러분은 모든 프레임을 탐색하면서 현재 선택된 프레임을 찾을 때까지** isSelected**를 호출해야 한다. 그런 다음, 연속해서 아이콘화 되지 않은 다음 프레임을 찾으면서 다음 호출을 통해 선택을 시도한다.**

```
frames[next].setSelected(true);
```

전에 보았던 것처럼, 이 메소드는 PropertyVetoException**을 발생시킨다. 만일 여러분이 본래의 프레임으로 복귀하면, 선택가능한 다른 프레임은 존재하지 않는다. 그리고 여러분은 이를 포기한다. 다음은 완전한 루프이다.**

```
for (int i = 0; i < frames.length; i++)
    {   if (frames[i].isSelected())
     {  /* find next frame that isn't an icon and can be
           selected
        */
        try
       {   int next = i + 1;
          while (next != i && frames[next].isIcon())
              next++;
          if (next == i) return;
            // all other frames are icons or veto selection
            frames[next].setSelected(true);
            frames[next].toFront();
            return;
        }
        catch(PropertyVetoException e)
        {}
      }
    }
```

프로퍼티 설정 거부(veto)

이제 여러분은 거부 예외상황의 모든 것을 살펴 보았다. 여러분은 어떻게 여러분의 프레임에서 거부를 하는 지 궁금해 할 것이다. JInternalFrame클래스는 속성들의 설정을 모니터링하기 위해 일반 자바 빈 메커니즘을 사용한다. 우리는 8장에서 이 메커니즘을 논의했었다. 지금, 우리는 여러분의 프레임들이 속성 변화를 거부하는 요청을 하는 방법을 보여주길 원한다.

프레임들은 일반적으로 아이콘화나 포커스를 읽는 것에 불복하기 위해 거부를 사용하길 원하지 않는다. 그러나 프레임들이 그들 자신을 닫는 것에 동의하는지 검사하는 것은 아주 일반적인 일이다. 여러분은 JInternalFrame 클래스의 setClosed 메소드로 프레임을 종료한다. 메소드가 거부 가능하기 때문에, 변경을 처리하기 전에 모두 거부 가능한 변경 리스너(vetoable change listener)에 등록해야 한다. 이것은 각각의 리스너들에게 PropertyVetoException을 발생시킬 기회와 모든 설정을 변경하기 전에 setClosed의 호출을 중단할 수 있는 기회를 제공한다.

우리의 예제 프로그램에서, 우리는 종료 윈도우에 동의할 것인지 사용자에게 묻는 팝업 다이얼로그를 제공한다(그림 6-52). 만일 사용자가 동의하지 않는 다면 윈도우는 개방된 상태로 남는다.

그림 6-52 : 사용자는 종료 속성을 거부 할 수 있다.

다음은 이 같은 통보를 달성하는 방법이다.

1. 각 프레임에 리스너 객체를 추가한다. 객체는 VetoableChangeListener 인터페이스를 구현한 몇몇 클래스에 속해야 한다. 프레임을 구성한 다음에 바로 수행하는 것이 좋다. 우리의 예제에서, 우리는 내부 프레임들을 구성하기 위해 프레임 클래스를 사용한다. 다른 옵션은 익명의 내부 클래스를 사용할 수 있다는 것이다.

```
iframe.addVetoableChangeListener(lis tener);
```

2. VetoableChangeListener 인터페이스에서 요구하는 유일한 메소드인 vetoableChange를 구현한다. 이 메소드는 PropertyChangeEvent 객체를 받는다. 변경될 속성의 이름을 발견하기 위해 getName메소드를 사용한다(가령, 거부 메소드 호출이 setClosed(true)일 때 “closed”처럼). 8장에서 보게 되듯이 , 속성 이름은 메소드 이름에서 “set” 접두사를 제거하고 다음 문자를 소문자로 변경하는 것에 의해서 얻어진다.

제출된 새로운 값을 얻기 위해 getNewValue 메소드를 사용한다.

```
String name = event.getPropertyName();
Object value = event.getNewValue();
if(name.equals("closed") && value.equals(Boolean.TRUE) )
{       ask user for confirmation
}
```

3. 속성 변경을 블록하기 위해 PropertyVetoException을 발생시킨다. 만일 여러분이 변화를 거부하길 원하지 않는다면 그냥 반환한다.

```
class DesktopFrame extends Jframe implements VetoableChangeListener
{       …
        public void vetoableChange(PropertyChangeEvent event)
                throws PropertyVetoException
        {       …
                if(not ok)
                        throw new PropertyVetoException("reason", event);
                // return normally if ok
        }
}
```

내부 프레임에서의 다이얼로그

만일 여러분이 내부 프레임들을 사용한다면, 다이얼로그를 위해 JDialog 클래스를 사용하지 말라. 이들 다이얼로그는 다음 두가지 단점들이 있다:

- 윈도우 시스템에서 새로운 프레임을 생성하기 때문에 무겁다.
- 윈도우 시스템은 다이얼로그들을 생성한 내부 프레임과 다이얼로그간의 상대적 위치를 모른다.

대신, 단순한 다이얼로그를 위해 JoptionPane 클래스의 showInternalXxxDialog 메소드를 사용한다. 이들은 내부 프레임상에 가벼운 윈도우를 위치시킨다는 점을 제외하고 showXxxDialog 메소드와 똑같이 작동한다.

보다 복잡한 다이얼로그에 대해서는, JInternalFrame으로 이를 구성한다. 불운하게도, 대화(modal) 다이얼로그는 내부에서 지원하지 않는다.

우리의 예제 프로그램에서, 우리는 프레임을 종료할 지를 사용자에게 확인받기 위해 내부 다이얼로그를 사용한다.

```
int result = JOptionPane.showInternalConfirmDialog(iframe, “OK to close?”)
```

노트 : 만일 여러분이 프레임을 종료할 때 이를 단순히 통지하길 원한다면 , 거부 메커니즘을 사용하지 말라. 대신에 , InternalFrameListener를 설치하라. 내부 프레임 리스너는 단지 WindowListener처럼 작동한다. 내부 프레임이 닫혀질 때, 침숙한 windowClosing메소드 대신에 internalFrameClosing 메소드가 호출된다. 다른 6개의 내부 프레임 통지(개방/종료, 아이콘화/비아이콘화, 활동/비활동) 또한 윈도우 리스너 메소드들에서 지원한다.

아웃트라인 드레깅

내부 프레임에 대한 개발자의 한가지 불평사항은 성능이 빠르지 않다는 것이다. 지금까지 가장 늦은 연산은 데스크탑을 가로질러 복잡한 컨텐트를 갖는 프레임을 드레그하는 것이다. 데스크탑 관리자는 프레임에게 드레그되는 동안 다시 그리도록 요청한다. 이는 아주 늦다.

실제, 여러분이 적당히 작성된 비디오 드라이버를 이용하여 윈도우나 X 윈도우를 사용한다면 같은 문제를 경험할 것이다. 비디오 하드웨어는 프레임 내부의 이미지를 드레깅 과정동안 다른 스크린 위치에 매핑시키는 것에 의해 드레그 연산을 지원하기 때문에 대부분의 시스템상에서는 윈도우 드레깅이 빠르게 나타난다.

사용자가 크게 실망하지 않고 성능을 향상하기 위해, 여러분은 “아웃트라인 트레깅”을 설정할 수 있다. 사용자가 프레임을 끌 때, 오직 프레임의 아웃트라인만이 계속해서 갱신된다 (그림 6-53 참조). 내부는 오직 사용자가 프레임을 최종 위치에 놓았을 때 다시 그려진다.

그림 6-53 : 아웃트라인 드레깅

아웃트라인 드레깅을 작동시키기 위해, 다음 구문을 사용한다.

```
desktop.putClientProperty(“JDesktopPane.dragMode”, “outline”);
```

노트 : 이 속성은 JSplitPane 클래스의 “지속 레이아웃”과 동일하다. 이러한 행동이 JSplitPane 클래스의 메소드로서 노출되는 것은 심각한 문제이다. 그러나 데스크 탑 패인을 위한 속성내에 숨겨져 있다.

예제 프로그램에서, 여러분은 아웃트라인 드레깅을 온/오프하는 토글을 선택하기 위해 Window -> Drag Outline 토글을 사용할 수 있다.

노트 : 데스크 탑상의 내부 프레임들은 DesktopManager 클래스에 의해서 관리된다. 여러분은 일반 프로그램을 위해 이 클래스를 알 필요는 없다. 새로운 데스크 탑 관리자를 설치하여 다른 데스크 탑 행위를 구현하는 것은 가능하다. 그러나 우리는 이를 다루지는 않는다.

예제 6-20은 HTML 페이지를 보여주는 내부 프레임을 갖는 데스크 탑을 구동시킨다. File->Open 메뉴 옵션은 로컬 HTML 파일을 새로운 내부 프레임에 읽기 위한 파일 다이얼로그를 팝업한다. 만일 여러분이 임의의 링크를 누른다면 연결된 문서가 다른 내부 프레임내에 디스플레이된다. Window -> Cascade와 Window->Tile 명령어를 시도해 보라. 이 예제로 고급 스윙 기능들에 대한 논의를 마친다.

예제 6-20 : InternalFrameTest.java

```
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;

public class InternalFrameTest
{  public static void main(String[] args)
   {  JFrame frame = new DesktopFrame();
      frame.show();
    }
}



class DesktopFrame extends JFrame
```

```
   implements ActionListener, VetoableChangeListener
{   public DesktopFrame()
    {   setTitle("InternalFrameTest");
        setSize(600, 400);
        addWindowListener(new WindowAdapter()
            {   public void windowClosing(WindowEvent e)
                {   System.exit(0);
                }
            } );

        desktop = new JDesktopPane();
        setContentPane(desktop);

        // set up menus

        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        JMenu fileMenu = new JMenu("File");
        menuBar.add(fileMenu);
        openItem = new JMenuItem("Open");
        openItem.addActionListener(this);
        fileMenu.add(openItem);
        exitItem = new JMenuItem("Exit");
        exitItem.addActionListener(this);
        fileMenu.add(exitItem);
        JMenu windowMenu = new JMenu("Window");
        menuBar.add(windowMenu);
        nextItem = new JMenuItem("Next");
        nextItem.addActionListener(this);
        windowMenu.add(nextItem);
        cascadeItem = new JMenuItem("Cascade");
        cascadeItem.addActionListener(this);
        windowMenu.add(cascadeItem);
        tileItem = new JMenuItem("Tile");
        tileItem.addActionListener(this);
        windowMenu.add(tileItem);
```

```
    dragOutlineItem = new JCheckBoxMenuItem("Drag Outline");
      dragOutlineItem.addActionListener(this);
      windowMenu.add(dragOutlineItem);
  }

  public void createInternalFrame(Component c, String t)
 {   JInternalFrame iframe = new JInternalFrame(t,
          true,  // resizable
          true,  // closable
          true,  // maximizable
          true);  // iconifiable

      iframe.getContentPane().add(c);
      desktop.add(iframe);

     iframe.setFrameIcon(new  ImageIcon("document.gif"));

    // add listener to confirm frame closing
      iframe.addVetoableChangeListener(this);

     // position  frame
    int  width = desktop.getWidth() / 2;
      int height = desktop.getHeight() / 2;
    iframe.reshape(nextFrameX, nextFrameY, width, height);

      iframe.show();

    // select  the  frame--might be vetoed
      try
      {    iframe.setSelected(true);
      }
     catch(PropertyVetoException  e)
      {}

   /* if this is the first time, compute distance between
          cascaded  frames
```

```
      */

    if (frameDistance == 0)
        frameDistance = iframe.getHeight()
            - iframe.getContentPane().getHeight();

    // compute placement for next frame

    nextFrameX += frameDistance;
    nextFrameY += frameDistance;
   if (nextFrameX + width > desktop.getWidth())
        nextFrameX = 0;
   if (nextFrameY + height > desktop.getHeight())
        nextFrameY = 0;
 }

public void cascadeWindows()
{   JInternalFrame[] frames = desktop.getAllFrames();
    int x = 0;
    int y = 0;
   int width = desktop.getWidth() / 2;
   int height = desktop.getHeight() / 2;

  for (int i = 0; i < frames.length; i++)
     {   if (!frames[i].isIcon())
         {   try
           {   /* try to make maximized frames resizable
                  this might be vetoed
                */
                frames[i].setMaximum(false);
              frames[i].reshape(x, y, width, height);

                x += frameDistance;
                y += frameDistance;
              // wrap around at the desktop edge
              if (x + width > desktop.getWidth()) x = 0;
```

```
            if (y + height > desktop.getHeight()) y = 0;
            }
            catch(PropertyVetoException e)
            {}
        }
    }
}

public void tileWindows()
{   JInternalFrame[] frames = desktop.getAllFrames();

   // count frames that aren't iconized
    int frameCount = 0;
   for (int i = 0; i < frames.length; i++)
    {   if (!frames[i].isIcon())
            frameCount++;
    }

    int rows = (int)Math.sqrt(frameCount);
   int cols = frameCount / rows;
   int extra = frameCount % rows;
      // number of columns with an extra row

   int width = desktop.getWidth() / cols;
   int height = desktop.getHeight() / rows;
    int r = 0;
    int c = 0;
  for (int i = 0; i < frames.length; i++)
    {   if (!frames[i].isIcon())
        {   try
            {   frames[i].setMaximum(false);
                frames[i].reshape(c * width,
                    r * height, width, height);
                r++;
                if (r == rows)
                {   r = 0;
```

```
                    c++;
                if (c == cols - extra)
                 {  // start adding an extra row
                      rows++;
                   height = desktop.getHeight() / rows;
                 }
               }
            }
            catch(PropertyVetoException e)
            {}
        }
    }
}

public void selectNextWindow()
{  JInternalFrame[] frames = desktop.getAllFrames();
  for (int i = 0; i < frames.length; i++)
    {  if (frames[i].isSelected())
      {  /* find next frame that isn't an icon and can be
                selected
            */
            try
          {  int next = i + 1;
             while (next != i && frames[next].isIcon())
                   next++;
             if (next == i) return;
               // all other frames are icons or veto selection
               frames[next].setSelected(true);
               frames[next].toFront();
               return;
            }
            catch(PropertyVetoException e)
            {}
        }
    }
}
```

```
public void vetoableChange(PropertyChangeEvent event)
    throws PropertyVetoException
{   JInternalFrame iframe = (JInternalFrame)event.getSource();
    String name = event.getPropertyName();
    Object value = event.getNewValue();

  // we only want to check attempts to close a frame

    if (name.equals("closed") && value.equals(Boolean.TRUE))
  {  // ask user if it is ok to close
       int result
          = JOptionPane.showInternalConfirmDialog(iframe,
             "OK to close?");

      // if the user doesn't agree, veto the close
       if (result == JOptionPane.CANCEL_OPTION)
          throw new PropertyVetoException("User canceled close",
               event);
    }
 }

public Component createEditorPane(URL u)
{  // create an editor pane that follows hyperlink clicks

   JEditorPane editorPane = new JEditorPane();
    editorPane.setEditable(false);
    editorPane.addHyperlinkListener(new HyperlinkListener()
       {   public void hyperlinkUpdate(HyperlinkEvent event)
           {   createInternalFrame(createEditorPane(event.getURL()),
                    event.getURL().toString());
           }
       });
    try
    {   editorPane.setPage(u);
    }
```

```
    catch(IOException e)
    {   editorPane.setText("Error: " + e);
     }
    return new JScrollPane(editorPane);
 }

public void actionPerformed(ActionEvent evt)
{   Object source = evt.getSource();
   if (source == openItem)
   {   // let user select file

       JFileChooser chooser = new JFileChooser();
         chooser.setCurrentDirectory(new File("."));
         chooser.setFileFilter(
            new javax.swing.filechooser.FileFilter()
           {   public boolean accept(File f)
              {   String fname = f.getName().toLowerCase();
                  return fname.endsWith(".html")
                     || fname.endsWith(".htm")
                     || f.isDirectory();
               }
              public String getDescription()
             { return "HTML Files"; }
            });
       int r = chooser.showOpenDialog(this);

       if (r == JFileChooser.APPROVE_OPTION)
      {   // open the file that the user selected

          String filename = chooser.getSelectedFile().getPath();
           try
         {   URL fileUrl = new URL("file:" + filename);
              createInternalFrame(createEditorPane(fileUrl),
                 filename);
           }
          catch(MalformedURLException e)
```

```
                {
                }
            }
        }
        else if (source == exitItem)
            System.exit(0);
        else if (source == nextItem)
            selectNextWindow();
        else if (source == cascadeItem)
            cascadeWindows();
        else if (source == tileItem)
            tileWindows();
        else if (source == dragOutlineItem)
        {   desktop.putClientProperty("JDesktopPane.dragMode",
              dragOutlineItem.isSelected() ? "outline" : null);
        }
    }

    private JDesktopPane desktop;

    private JMenuItem openItem;
    private JMenuItem exitItem;
    private JMenuItem nextItem;
    private JMenuItem cascadeItem;
    private JMenuItem tileItem;
    private JMenuItem dragOutlineItem;

    private int nextFrameX;
    private int nextFrameY;
    private int frameDistance;
}
```

javax.swing.JDesktopPane

- JInternalFrame[] getAllFrames()
  해당 데스크탑 패인에 있는 모든 내부 프레임을 얻는다.

javax.swing.JInternalFrame

- JInternalFrame()
- JInternalFrame(String title)
- JInternalFrame(String title , boolean resizable)
- JInternalFrame(String title , boolean resizable , boolean closable)
- JInternalFrame(String title , boolean resizable , boolean closable , boolean maximizable)
- JInternalFrame(String title , boolean resizable , boolean closable , boolean maximizable , boolean iconfiable)

  새로운 내부 프레임을 구성한다.

  전달인자:

	title	**제목 바에 디스플레이할 문자열**
	resizable	**만일 프레임이 크기 조정가능하면** true
	closable	**만일 프레임이 닫혀질 수 있으면** true
	maximizable	**만일 프레임이 최대화 될 수 있으면** true
	iconfiable	**만일 프레임이 아이콘화 될 수 있으면** true

- boolean isResizable()
- boolean isClosable()
- boolean isMaximizable()
- boolean isIconifiable()

  resizable, closable, maximizable , iconfiable **속성을 설정하거나 가져 온다. 속성이** true **이면 프레임 제목에 내부 프레임을 재조장하고, 닫고, 최대화하거나 아이콘화하는 아이콘이 나타난다.**
- boolean isIcon()
- void setIcon(boolean b)
- boolean isMaximum()
- void setMaximum (boolean b)
- boolean isClosed()
- void setClosed (boolean b)

  icon , maximum, **또는** closed **속성을 설정하거나 가져온다. 이 속성이** true**이면, 내부 프레임은 아이콘화 되거나, 최대화 되거나, 닫혀진다.**
- boolean isSelected()
- void setSelected (boolean b)

  selected **속성을 설정하거나 가져온다. 이 속성이** true**이면, 현재 내부 프레임이 데스크탑상에 선택된 프레임이 된다.**
- void moveToFront()
- void moveToBack()

  데스크탑의 앞이나 뒤로 해당 내부 프레임을 이동한다.

- void reshape(int x , int y, int width, int height)

  해당 내부 프레임의 이동하거나 재조정한다.

  전달인자: x,y **프레임의 최상단 좌측 구석 좌표**

  width, height **프레임의 폭과 넓이**

- Container getContentPane()
- void setContentPane(Container c)

  해당 내부 프레임의 컨텐트 패인을 설정하거나 가져온다.

- JDesktopPane getDesktopPane()

  해당 내부 프레임의 데스크탑 패인을 가져온다.

- Icon getFrameIcon()
- void setFrameIcon(Icon anIcon)

  제목 바에 디스플레이되는 프레임 아이콘을 가져오거나 설정한다.

- boolean isVisible()
- void setVisible(boolean b)

  가시화 속성을 가져오거나 설정한다.

- void show()

  해당 내부 프레임을 가시화하고 이를 앞쪽으로 가져온다.

javax.swing.JComponent

- void addVetoableChangeListener(VetoableChangeListener listener)

  제약받은 속성을 변경하려고 시도할 때 통보될 거부가능한 변경 리스너를 추가한다.

java.beans.VetoableChangeListener

- void vetoableChange(PropertyChangeEvent event)

  제약받은 속성의 set **메소드가 거부가능한 변경 리스너에게 통보할 때 호출된다.**

java.beans.PropertyChangeEvent

- String getPropertyName()

  변경할 속성의 이름을 반환한다.

- Object getNewValue()

  속성을 위해 제출된 새로운 값을 반환한다.

java.beans.PropertyVetoException

- PropertyVetoException(String reason, PropertyChangeEvent event)

  속성 거부 예외상황을 구성한다.

  전달인자: reason **거부 이유**

  event **거부 이벤트**