6장 - II

테이블들

- 단순 테이블
- 테이블 모델
- 데이터베이스 레코드 출력
- 정렬 필터

셀 렌더링과 편집

- 셀 렌더링
- 셀 편집
- 사용자 편집

테이블들

JTable 컴포넌트는 객체들의 2차원 격자를 디스플레이한다. 물론, 테이블들은 사용자 인터페이스에서 아주 일반적으로 사용되는 것이다. 스윙 팀은 테이블 컨트롤에 많은 노력을 기울였다. 테이블들은 본래 복잡하다. 그러나 – 다른 스윙 클래스들보다 성공적이다.-JTable컴포넌트는 복잡함을 가려준다. 여러분은 단지 몇 줄의 코드만으로도 많은 기능을 갖는 풍부한 기능의 테이블을 만들 수 있다. 물론, 여러분은 특정 응용 프로그램을 위한 디스플레이와 행위를 커서터마이징하고 보다 많은 코드를 작성할 수 있다.

이 절에서, 우리는 단순한 테이블들을 만드는 방법과 이들 테이블과의 사용자 인터페이스 방법, 가장 일반적인 정렬 방법을 설명한다. 다른 복잡한 스윙 컨트롤처럼, 완전히 보다 자세한 부분까지 다루는 것은 불가능하다. 만일 보다 자세한 정보가 필요하다면, Kim Topley의 Core Java Foundation Classed나 David Geary의 Graphics Java를 참조하라.

단순 테이블

트리 컨트롤처럼, JTable은 자신의 데이터를 저장하지 않는다. 그러나 테이블은 테이블 모델로부터 자신의 데이터를 얻는다. 그리고 자동적으로 배열을 기본 모델로 포장한다. 이것이 우리의 첫번째 예제에서 사용할 전략이다. 이 장의 뒤에서, 우리는 테이블 모델로 선회할 것이다.

그림 6-21은 태양계의 행성의 속성을 묘사하는 테이블이다. (만약 행성이 대부분 수소와 헬륨으로 구성되어 있다면 그 행성은 기체 상태이다. 그리고 이 테이블에는 “Color” 컬럼이 있

다. 이 컬럼은 뒤의 코드에서 필요하기 때문에 추가되었다.)

그림 6-21 : 단순 테이블

예제 6-6의 코드를 보게 되면 테이블의 데이터는 Object 값의 2차원 배열로 저장되어 있다는 것을 알 수 있다.

```
Object[][] cells =
{       { "Mercury", new Double(2440), new Integer(0),
          Boolean.FALSE, Color.yellow
        },
        { "Venus", new double(6052), new Integer(0),
          Boolean.FALSE, Color.yellow
        },
        . . .
)
```

테이블은 데이터를 나타내기 위해 각 객체에 toString 메소드를 사용한다. 그 때문에 색상 데이터들이 java.awt.Color[r=...,g=...,b=...]로서 표시된다.
여러분은 별도의 문자열 배열에서 컬럼 이름을 제공해야 한다.

```
String[] columnNames =
{   "Planet", "Radius", "Moons", "Gaseous", "Color"
};
```

그리고 나서, 셀과 컬럼 이름 배열로 테이블을 생성해야 한다. 마지막으로, JScrollPane에 테이블 인자를 포장하는 일반적인 방법으로 스크롤 바를 추가한다.

```
JTable table = new JTable(cells, columnNames);
JScrollPane pane = new JscrollPane(table);
```

결과 테이블은 이미 상당히 많은 기능을 가지고 있다. 스크롤 바가 나타날 때까지 세로로 테이블 크기를 조정해 보라. 그러면, 테이블은 스크롤이 될 수 있게 된다. 그렇지만 컬럼 헤더는 뷰에서 스크롤 되지 않는다.
다음, 컬럼 헤드중의 하나를 클릭하고 그것을 왼쪽 또는 오른쪽으로 드래그 하라. 전체 컬럼이 분리될 것이다.(그림 6-22) 다른 위치에 그것을 드롭 시킬 수도 있다. 이것은 *오직 뷰에서만* 컬럼이 재정렬되는 것이고 데이터 모델은 영향을 받지 않는다.

그림 6-22: 컬럼 이동

컬럼의 크기를 재조정하기 위하여 커서 모양이 화살표로 변화될 때까지 두개의 컬럼 사이에 커서를 위치시켜라. 그리고 나서, 원하는 위치로 컬럼 경계를 드래그 하라.(그림 6-23)

그림 6-23 : 컬럼 크기 재조절

사용자가 열의 아무 곳이나 클릭함으로써 열을 선택할 수 있다. 선택된 열들은 색상이 반전된다. 선택 이벤트를 얻는 방법은 나중에 설명할 것이다. 사용자는 역시 셀을 클릭하여 편집을 함으로써 테이블 엔트리를 입력할 수 있다. 그러나 이 코드 예제에서는 밑줄 친 데이터는 변화시키지 못하도록 하고자 한다. 따라서 프로그램에서 셀들을 입력 가능하지 않게 하거나 셀 입력 이벤트를 사용하여 모델을 갱신할 수 있어야 한다. 이러한 것들에 대해서는 이 섹션의 후반부에 다룰 것이다.

예제 6-6 : PlanetTable.java

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

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

class PlanetTableFrame extends JFrame
{  public PlanetTableFrame()
   {  setTitle("PlanetTable");
      setSize(300, 200);
      addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
```

```
            {   System.exit(0);
            }
        } );

    JTable table = new JTable(cells, columnNames);

    Container contentPane = getContentPane();
    contentPane.add(new JScrollPane(table), "Center");
 }

private Object[][] cells =
    {   { "Mercury", new Double(2440),   new Integer(0),
            Boolean.FALSE, Color.yellow
        },
       { "Venus", new Double(6052), new Integer(0),
            Boolean.FALSE,  Color.yellow
        },
       { "Earth", new Double(6378), new Integer(1),
            Boolean.FALSE,  Color.blue
        },
        {  "Mars",  new Double(3397), new Integer(2),
            Boolean.FALSE,  Color.red
        },
       { "Jupiter", new Double(71492), new Integer(16),
            Boolean.TRUE,  Color.orange
        },
       { "Saturn", new Double(60268), new Integer(18),
            Boolean.TRUE,  Color.orange
        },
       { "Uranus", new Double(25559), new Integer(17),
            Boolean.TRUE,  Color.blue
        },
       { "Neptune", new Double(24766), new Integer(8),
            Boolean.TRUE,  Color.blue
        },
       { "Pluto", new Double(1137), new Integer(1),
```

```
            Boolean.FALSE, Color.black
        }
    };

  private String[] columnNames =
    {  "Planet", "Radius", "Moons", "Gaseous", "Color"
    };
}
```

javax.swing.JTable

- JTable(Objects[][] entries, Object[] columnNames)
  기본 데이블 모델로 한 테이블을 생성한다.

테이블 모델

이전 예제에서 랜더링된 객체는 2차원 배열로 저장되어 있었다. 그러나 일반적으로 그러한 방법을 사용하지 않고 테이블 모델을 적용할 것이다. 만일 여러분이 데이터를 테이블로 디스플레이하기 위해 배열로 데이터를 덤프해야 한다면, 여러분 자신의 테이블 모델을 구현하는 것에 대해 생각해야 한다.

대부분의 메소드를 제공하는 AbstractTableModel **클래스가 있기 때문에 테이블 모델을 사용하는 것은 간단하다. 여기에는 다음의 세 개만의 메소드를 사용할 필요가 있다.**

```
public int getRowCount ();
public int getColumncount ();
public Object gerValueAt (int row, int column);
```

getValueAt **메소드를 적용하는 방법은 다양하다. 계산할 수도 있고 데이터베이스나 다른 저장소에서 그 값을 검토할 수도 있다. 예제들을 보자.**

첫번째 예제에서는 몇가지 계산된 값들(투자 값이 다른 비율로 증가되고 있다.)을 보여주는 테이블을 생성한다. 즉, 다른 이익율 시나리오하에서의 투자 성장율을 보여준다.(그림 6-24 **참조).**

그림 6-24 : 투자 성장율

getValueAt **메소드는 적절한 값을 계산하여 그것을 적절한 형태로 나타낸다.**

```
public Object getValueAt(int r, int c)
{  double rate = (c + minRate) / 100.0;
```

```
        int nperiods = r;

        double futureBalance = INITIAL_BALANCE
            *Math.pow(1 + rate, nperiods);
        return
        NumberFormat.getCurrencyInstance().format(futureBalance);
    }
```

getRowCount**와** getColumnCount **메소드는 단순히 열과 행의 수를 반환한다**.

```
    public int getRowCount()
    {   return years;
    }

    public int getColumnCount()
    {   return maxRate – minRate + 1;
    }
```

만약 행 이름을 제공하지 않는다면 AbstractTableModel**의** getColumnName **메소드는 행을** A, B, C **등으로 이름 짓는다. 행 이름을 변화시키기 위하여** getColumnName **메소드를 오버라이드한다. 대개는 디폴트 기능을 오버라이드할 것이다. 이 예제에서, 우리는 각 행을 원하는 비율로 이름 지을 것이다**.

```
    public String getColumnName(int c)
    { double rate = (c + minRate) / 100.0;
      return
            NumberFormat.getPercentInstance().format(rate);
    }
```

전체 소스는 예제 6-7**에 나타내었다**.

예제 6-7 : InvestmentTable.java

```
import java.awt.*;
import java.awt.event.*;
import java.text.*;
import javax.swing.*;
import javax.swing.table.*;
```

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

/* this data model computes the cell entries each time they
   are requested
*/

class InvestmentTableModel extends AbstractTableModel
{  public InvestmentTableModel(int y, int r1, int r2)
   {  years = y;
      minRate = r1;
      maxRate = r2;
    }

   public int getRowCount()
    {  return years;
    }

    public int getColumnCount()
    {  return maxRate - minRate + 1;
    }

   public Object getValueAt(int r, int c)
   {  double rate = (c + minRate) / 100.0;
      int nperiods = r;

      double futureBalance = INITIAL_BALANCE
         * Math.pow(1 + rate, nperiods);

       return
          NumberFormat.getCurrencyInstance().format(futureBalance);
    }
```

```
   public String getColumnName(int c)
   {  double rate = (c + minRate) / 100.0;
         return
            NumberFormat.getPercentInstance().format(rate);
    }

    private int years;
   private int minRate;
   private int maxRate;

  private static double INITIAL_BALANCE = 100000.0;
}

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


      TableModel model = new InvestmentTableModel(30, 5, 10);
       JTable table = new JTable(model);
       Container contentPane = getContentPane();
       contentPane.add(new JScrollPane(table), "Center");
   }
}
```

데이터베이스 레코드 디스플레이

아마 테이블에서 표시되는 대부분의 정보는 데이터베이스 내의 레코드 셋일 것이다. 만약 데이터베이스 정보에 접근하기 위하여 전문적인 개발 환경을 사용하고자 한다면 대부분은 편리한 자바 빈을 포함할 것이다. 그러나 데이터베이스 빈즈를 사용하는 것을 원하지 않는

다면 다음 예제를 보기 바란다. 그림 6-25는 데이터베이스 테이블에서의 질의 결과를 나타내고 있다.

그림 6-25 : 테이블내에 질의 결과 디스플레이

예제 프로그램에서 우리는 데이터베이스 질의의 결과 셋으로부터 데이터를 패치하는 ResultSetTableModel을 정의한다.(4장에서 자바 데이터베이스와 결과 셋에 대한 보다 자세한 내용을 참조하라.)

우리는 *결과 셋 메타 데이터* 객체인 rsmd로부터 컬럼 숫자와 컬럼 이름을 얻을 수 있다.

```
public String getColumnName(int c)
{ try
   { return rsmd.getColunmName(c+1);
    }
   catch(SQLException e)
   {  . . .
    }
}

public int getColumnCount()
{ try
   { return rsmd.getColumnCount();
    }
   catch(SQLException e)
   {  . . .
    }
}
```

만약 데이터베이스가 커서 스크롤을 지원한다면 아주 쉽게 셀 값을 얻을 수 있다. 단지 필요한 열에 커서를 옮기고 컬럼 값을 패치하면 된다.

```
public Object getValueAt(int r, int c)
{ try
   {  ResultSet rs = getResultSet();
       rs.absolute(r+1);
      return rs.getObject(c+1);
```

```
            }
            catch(SQLException e)
            {   System.out.println("Error   " + e);
                return null;
            }
        }
```

DefaultTableModel 대신에 이러한 데이터 모델을 사용하는 것이 편리할 때가 많다. 만약 값의 배열을 생성한다면, 그 때 여러분은 데이터베이스 드라이브가 이미 관리하고 있는 캐쉬를 중복하게 만들 것이다.

만약 데이터베이스가 커서 스크롤을 지원하지 않거나 JDBC 1 드라이브를 사용한다면 쿼리 데이터는 수동으로 캐쉬되어야 한다. 예제 프로그램은 그런 수동 캐쉬를 포함하고 있다. 기본 클래스 ResultSetTableModel에서 공통 기능을 분리해 내었다. 서브클래스 ScrollingResultSetTableModel은 커서 스크롤을 사용한다. 그리고 서브클래스 CachingResultSetTableModel은 결과 셋을 캐쉬한다.

예제 6-8 : ResultSetTable.java

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

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

/* this class is the base class for the scrolling and the
   caching result set table model. It stores the result set
```

```
   and its metadata.
*/

abstract class ResultSetTableModel extends AbstractTableModel
{  public ResultSetTableModel(ResultSet aResultSet)
   {  rs = aResultSet;
      try
      {  rsmd = rs.getMetaData();
      }
      catch(SQLException e)
      {  System.out.println("Error " + e);
      }
   }

   public String getColumnName(int c)
   {  try
      {  return rsmd.getColumnName(c + 1);
      }
      catch(SQLException e)
      {  System.out.println("Error " + e);
         return "";
      }
   }

   public int getColumnCount()
   {  try
      {  return rsmd.getColumnCount();
      }
      catch(SQLException e)
      {  System.out.println("Error " + e);
         return 0;
      }
   }

   protected ResultSet getResultSet()
   {  return rs;
```

```
    }

   private ResultSet rs;
   private ResultSetMetaData rsmd;
}

/* this class uses a scrolling cursor, a JDBC 2 feature
*/

class ScrollingResultSetTableModel extends ResultSetTableModel
{  public ScrollingResultSetTableModel(ResultSet aResultSet)
   {   super(aResultSet);
   }

  public Object getValueAt(int r, int c)
   {   try
      {   ResultSet rs = getResultSet();
          rs.absolute(r + 1);
           return rs.getObject(c + 1);
       }
      catch(SQLException e)
      {   System.out.println("Error " + e);
          return null;
       }
   }

   public int getRowCount()
   {   try
      {   ResultSet rs = getResultSet();
           rs.last();
          return rs.getRow();
       }
      catch(SQLException e)
      {   System.out.println("Error " + e);
          return 0;
       }
```

```
    }
}

/* this class caches the result set data; it can be used
   if scrolling cursors are not supported
*/

class CachingResultSetTableModel extends ResultSetTableModel
{   public CachingResultSetTableModel(ResultSet aResultSet)
    {   super(aResultSet);
        try
        {   cache = new ArrayList();
            int cols = getColumnCount();
            ResultSet rs = getResultSet();

            /* place all data in an array list of Object[] arrays
              We don't use an Object[][] because we don't know
              how many rows are in the result set
             */

            while (rs.next())
            {   Object[] row = new Object[cols];
                for (int j = 0; j < row.length; j++)
                  row[j] = rs.getObject(j + 1);
                cache.add(row);
             }
        }
        catch(SQLException e)
        {   System.out.println("Error " + e);
        }
    }

   public Object getValueAt(int r, int c)
    {   if (r < cache.size())
            return ((Object[])cache.get(r))[c];
        else
```

```
            return null;
    }

    public int getRowCount()
    {   return cache.size();
    }

    private ArrayList cache;
}

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

        /* find all tables in the database and add them to
           a combo box
        */

        Container contentPane = getContentPane();
        tableNames = new JComboBox();
        tableNames.addActionListener(this);
        JPanel p = new JPanel();
        p.add(tableNames);
        contentPane.add(p, "North");

        try
        {   Class.forName("com.pointbase.jdbc.jdbcDriver");
            // force loading of driver
            String url = "jdbc:pointbase:corejava";
```

```
        String user = "PUBLIC";
        String password = "PUBLIC";
        con = DriverManager.getConnection(url, user,
              password);
        if (SCROLLABLE)
            stmt = con.createStatement(
                ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_READ_ONLY);
        else
            stmt = con.createStatement();
        DatabaseMetaData md = con.getMetaData();
        ResultSet mrs = md.getTables(null, null, null,
            new String[] { "TABLE" });
        while (mrs.next())
            tableNames.addItem(mrs.getString(3));
        mrs.close();
    }
    catch(ClassNotFoundException e)
    {   System.out.println("Error " + e);
    }
    catch(SQLException e)
    {   System.out.println("Error " + e);
    }
}

public void actionPerformed(ActionEvent evt)
{   if (evt.getSource() == tableNames)
    {   // show the selected table from the combo box

        if (scrollPane != null)
            getContentPane().remove(scrollPane);
        try
        {   String tableName
                = (String)tableNames.getSelectedItem();
            if (rs != null) rs.close();
            String query = "SELECT * FROM " + tableName;
```

```
                rs = stmt.executeQuery(query);
                if (SCROLLABLE)
                   model = new ScrollingResultSetTableModel(rs);
                else
                   model = new CachingResultSetTableModel(rs);

               JTable table = new JTable(model);
               scrollPane = new JScrollPane(table);
                getContentPane().add(scrollPane, "Center");
                 pack();
                 doLayout();
             }
            catch(SQLException e)
           {   System.out.println("Error " + e);
             }
         }
     }

   private JScrollPane scrollPane;
   private ResultSetTableModel model;
   private JComboBox tableNames;
    private JButton nextButton;
   private JButton previousButton;
   private ResultSet rs;
   private Connection con;
   private Statement stmt;

  private static boolean SCROLLABLE = false;
     // set to true if your database supports scrolling cursors
}
```

정렬 필터

마지막 두 예제는 테이블이 셀 데이터를 저장하지 않는다는 것을 설명한다. 그 예제들에서 셀 데이터는 모델로부터 얻어낸다. 모델도 데이터를 저장할 필요가 없다. 그것은 셀 값을 계산할 수 있거나 다른 어딘가에서 데이터들을 패치할 수 있다.

이 절에서는 다른 유용한 기술인 *필터 모델*을 소개한다. 필터 모델은 다른 테이블로부터의 정보를 다른 폼으로 제공한다. 예제에서는 테이블의 열들을 *정렬*할 것이다. 예제 6-9의 프로그램을 실행하고 컬럼 헤더중의 하나를 더블 클릭하라. 컬럼 엔트리가 정렬되는 것을 볼 수 있다.(그림 6-26)

그림 6-26 : 테이블의 행들에 대한 정렬

그러나 우리는 데이터 모델에서 물리적으로 열들을 재배열하지 않을 것이다. 대신에 배열이 치환된 열 인덱스를 가지도록 하는 *필터 모델*을 사용할 것이다.

필터 모델은 실질적인 테이블 모델에 대한 참조를 저장한다. JTable이 값을 찾아낼 필요가 있을 때 필터 모델은 실질적인 열 인덱스를 계산하고 모델로부터 그 값을 얻어낸다. 예를 들면,

```
public Object getValueAt(int r, int c)
{   return model.getValueAt(actual row index, c);
}
```

모든 다른 메소드는 간단히 본래의 모델로 넘겨진다.

```
public String getColumnName(int c)
{   return model.getColumnName(c);
}
```

그림 6-27은 필터가 JTable 객체와 실질적인 테이블 모델 사이에 필터가 놓여지는 방법을 보여준다.

그림 6-27 : 테이블 모델 필터

이와 같은 정렬 필터를 적용할 때 두 가지 문제가 존재한다. 먼저 사용자가 컬럼 헤더중의 하나를 더블 클릭할 때 통지할 필요가 있다. 이러한 기술적인 부분에 대해 너무 상세하게 접근하지는 않을 것이다. 예제 6-9의 SortFilterModel의 addMouseListener 메소드에서 그 코드를 볼 수 있다. 코드 이면에 숨겨진 생각은 다음과 같다. 먼저 테이블 헤더 컴포넌트를 얻

어 마우스 리스너를 부착하라. 더블 클릭이 탐지되면 어떤 테이블 컬럼에서 마우스가 클릭되는 지를 알아낼 필요가 있다. 그리고 나서 테이블 컬럼을 모델 컬럼으로 전환할 필요가 있다. 만약 테이블 컬럼을 옮겼다면 그것들은 달라질 수 있다. 만약 모델 컬럼을 알고 있다면 테이블 열의 정렬을 시작할 수 있다.

테이블 열을 정렬하는데 문제가 하나 있다. 우리는 물리적으로 열을 재정렬하기를 원하지 않는다. 우리가 원하는 것은 우리에게 재정렬하는 방법을 알려주는 열 인덱스의 순서이다. 그러나 Arrays와 Collections 클래스에서 정렬 알고리즘은 우리에게 그 요소를 재정렬하는 방법을 알려주지는 않는다. 물론 정렬 알고리즘을 재적용하고 객체 재배열의 방법을 사용할 수 있다. 그러나 훨씬 더 명쾌한 방법이 있다. 그 방법은 관례적인 객체와 관례적인 비교 방법에 따라 라이브러리 정렬 알고리즘을 서비스에 끼워 넣는 것이다.

우리는 Row 객체를 정렬할 것이다. Row 객체는 모델에서 열의 인덱스 r을 포함한다. 다음과 같은 두개의 객체를 비교하라. 모델에서 요소를 찾아서 그것들을 비교하라. 즉 Row 객체의 compareTo 메소드는

model.getValueAt(r1, c).compareTo(model.getValueAt(r2, c))

를 계산한다. 여기서 r1와 r2는 Row 객체의 열 인덱스이다. 그리고 c는 정렬하고자 하는 요소의 컬럼이다.
우리는 compareTo 메소드가 현재의 모델과 컬럼에 접근할 필요가 있기 때문에 Row 클래스를 SortFilterModel의 내부 클래스로 만들 것이다.

```
class SortFilterModel extends AbstractTableModel
{ . . .
   private class Row implements Comparable
   {  public int index;
      public int compareTo(Object other)
      {  Row otherRow =  (Row)other;
         Object a = model.getValueAt(index, sortColumn);
         Object b = model.getValueAt(otherRow.index, sortColumn);
         if (a instanceof Comparable)
            return ((Comparable) a).compareTo(b);
         else
           /* compare index with otherRow.index and
              return < 0, 0 or > 0 value
            */
            return index – otherRow.index;
```

```
        }
    }
    private TableModel model;
    private int sortColumn;
    private Row[] rows;
}
```

생성자에서 우리는 배열인 rows**를 생성하고** rows[i]**를** i**로 초기화한다**.

```
    public SortFilterModel(TableModel m)
    {   model = m;
        rows = new Row[model.getRowCount()];
        for (int i = 0; i < rows.length; i++)
           {
              rows[i] = new Row();
              rows[I].index = I;
           }
    }
```

sort **메소드에서** Arrays.sort **알고리즘을 호출한다**. **그것은** Row **객체를 정렬한다**. **비교 기준을 적용하면** row[0]**는 그 행에서 가장 적은 요소의 인덱스를 포함하고** row[1]**은 그 다음 적은 요소의 인덱스를 포함 하는 식으로 요소들은 배열된다**.

배열이 정렬될 때 모든 테이블 모델 리스너에게(특히, JTable), **테이블 내용이 변화되어 다시 표시되었다는 것을 통지한다**.

```
           public void sort(int c)
           {  sortcolumn = c;
              Arrays.sort(rows);
              FireTableDataChanged();
           }
```

마침내, **필터 클래스의** getValueAt **메소드의 정확한 계산을 볼 수 있다**. **그것은 열 인덱스** r **을 모델 열 인덱스** rows[r].index **로 간단히 전환한다**:

```
                    public Object getValueAt(int r, int c)
                    {  return model.getValueAt(rows[r].index, c);
                    }
```

정렬 모델 필터는 다시 모델-뷰 패턴의 파워를 다시 보여준다. **데이터와 디스플레이가 분리되어 있기 때문에 둘 사이의 매핑을 변화시킬 수 있다**.

예제 6-9: TableSortTest.java

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

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

class SortFilterModel extends AbstractTableModel
{   public SortFilterModel(TableModel m)
    {   model = m;
        rows = new Row[model.getRowCount()];
        for (int i = 0; i < rows.length; i++)
        {   rows[i] = new Row();
            rows[i].index = i;
        }
    }

    public void sort(int c)
    {   sortColumn = c;
        Arrays.sort(rows);
        fireTableDataChanged();
    }

    public void addMouseListener(final JTable table)
    {   table.getTableHeader().addMouseListener(new MouseAdapter()
            {   public void mouseClicked(MouseEvent event)
                {   // check for double click
                    if (event.getClickCount() < 2) return;
```

```
                // find column of click and
                int tableColumn
                    = table.columnAtPoint(event.getPoint());

              // translate to table model index and sort
                int modelColumn
                    = table.convertColumnIndexToModel(tableColumn);
                sort(modelColumn);
              }
          });
  }

/* compute the moved row for the three methods that access
     model elements
  */

 public Object getValueAt(int r, int c)
  {  return model.getValueAt(rows[r].index, c);
  }

 public boolean isCellEditable(int r, int c)
  {  return model.isCellEditable(rows[r].index, c);
  }

 public void setValueAt(Object aValue, int r, int c)
  {  model.setValueAt(aValue, rows[r].index, c);
  }

 /* delegate all remaining methods to the model
  */

 public int getRowCount()
  {  return model.getRowCount();
  }

 public int getColumnCount()
```

```
    {   return model.getColumnCount();
    }

   public String getColumnName(int c)
    {   return model.getColumnName(c);
    }

    public Class getColumnClass(int c)
    {   return model.getColumnClass(c);
    }

  /* this inner class holds the index of the model row
     Rows are compared by looking at the model row entries
      in the sort column
   */

   private class Row implements Comparable
    {   public int index;
       public int compareTo(Object other)
       {   Row otherRow = (Row)other;
          Object a = model.getValueAt(index, sortColumn);
          Object b = model.getValueAt(otherRow.index, sortColumn);
           if (a instanceof Comparable)
               return ((Comparable)a).compareTo(b);
            else
              return index - otherRow.index;
        }
    }

   private TableModel model;
   private int sortColumn;
   private Row[] rows;
}

class TableSortFrame extends JFrame
{   public TableSortFrame()
```

```
{   setTitle("TableSortTest");
    setSize(300, 200);
    addWindowListener(new WindowAdapter()
       {  public void windowClosing(WindowEvent e)
          {  System.exit(0);
          }
       } );

    // set up table model and interpose sorter

    DefaultTableModel model
       = new DefaultTableModel(cells, columnNames);
   SortFilterModel sorter = new SortFilterModel(model);

    // show table

   JTable table = new JTable(sorter);
    Container contentPane = getContentPane();
    contentPane.add(new JScrollPane(table), "Center");

  // set up double click handler for column headers

    sorter.addMouseListener(table);
}

private Object[][] cells =
   {  { "Mercury", new Double(2440),  new Integer(0),
          Boolean.FALSE, Color.yellow
       },
      { "Venus", new Double(6052), new Integer(0),
          Boolean.FALSE, Color.yellow
       },
      { "Earth", new Double(6378), new Integer(1),
          Boolean.FALSE, Color.blue
       },
      { "Mars", new Double(3397), new Integer(2),
```

```
            Boolean.FALSE, Color.red
          },
        { "Jupiter", new Double(71492), new Integer(16),
            Boolean.TRUE, Color.orange
          },
         { "Saturn", new Double(60268), new Integer(18),
            Boolean.TRUE, Color.orange
          },
        { "Uranus", new Double(25559), new Integer(17),
            Boolean.TRUE, Color.blue
          },
        { "Neptune", new Double(24766), new Integer(8),
            Boolean.TRUE, Color.blue
          },
        { "Pluto", new Double(1137), new Integer(1),
            Boolean.FALSE, Color.black
          }
      };

  private String[] columnNames =
    {  "Planet", "Radius", "Moons", "Gaseous", "Color"
     };
}
```

javax.swing.table.TableModel

- int getRowCount()
- int getColumnCount()
  테이블 모델내의 열과 행의 숫자를 반환한다.
- Object getValueAt(int row, int column)
  주어진 행과 열의 값을 가져온다.
- void setValueAt(Object newValue, int row, int column)
  주어진 행과 열에 새로운 값을 설정한다.
- boolean isCellEditable(int row, int column)
  만일 주어진 행과열에 있는 셀이 편집가능하면 true**를 반환한다**.
- String getColumnName(int column)
  컬럼 제목을 가져온다.

javax.swing.table.AbstractTableModel

- void fireTableDataChanged()
  테이블 데이터가 변경됐음을 모든 테이블 모델 리스너들에게 통보한다.

javax.swing.JTable

- JTableHeader getTableHeader()
  해당 테이블의 테이블 헤더 컴포넌트를 반환한다.
- int columnAtPoint(Point p)
  픽셀 위치 p **아래에 있는 테이블 컬럼의 숫자를 반환한다.**
- int convertColumnIndexToModel(int tableColumn)
  주어진 인덱스를 갖는 컬럼의 모델 인덱스를 반환한다. 만일 몇몇 테이블 컬럼들이 이동되거나 감춰졌으면 해당 값은 tableColumn **과 다르다.**

셀 렌더링과 편집

다음 예제에서 다시 행성 데이터를 나타낼 것이다. 그러나 이번에는 테이블에 행 타입에 대해 더 많은 정보를 제공할 것이다. 만약 컬럼 타입을 묘사하는 클래스를 반환하기 위하여 테이블 모델의 메소드

Class getColumnClass(int columnIndex)

을 정의한다면 JTable **클래스는 그 클래스를 위해 적절한** renderer**를 선택할 것이다. 테이블** 6-1**은** JTable **클래스가 디폴트로 제공하는 랜드러를 나타낸다.**

표 6-1 : 기본 렌더러

타입	**..로 렌더링된**
ImageIcon	image
Boolean	check box
Object	string

그림 6-28**에서 체크박스와 이미지를 볼 수 있다.** (**행성 이미지를 제공해 준 것에 대해** Jim Evins**에 감사한다.** http://www.clark.net/pub/evins/Main/)

그림 6-28 : 셀 렌더러를 갖는 테이블

다른 타입에 대해 자신의 셀 랜더러를 제공할 수 있다. 테이블 셀 랜더러는 이전에 본 트리 셀 랜더러와 유사하다. 그것은 TableCellRenderer 인터페이스를 적용한다.

```
Component getTableCellRendererComponent(JTable table,
   Object value, Boolean isSelected, Boolean hasFocus,
   int row, int column)
```

해당 메소드는 트리가 셀을 나타낼 필요가 있을 때 호출된다. 셀 영역을 채우기 위하여 사용되는 paint 메소드를 사용하는 컴포넌트를 반환한다.

셀 타입 Color을 나타내기 위하여 color 객체에 설정된 배경색을 패널에 반환할 수 있다. 색상은 value 인수로 전달된다.

```
class ColorTableCellRenderer implements TableCellRenderer
{   public component getTableCellRendererComponent(JTable table,
          Object value, Boolean isSelected, Boolean hasFocus,
          int row, int column)
     {   panel.setBackground((Color)value);
         return panel;
}
```

타입 Color의 모든 객체를 가진 이러한 랜더러를 사용하도록 테이블에 지시할 수 있다. JTable 클래스의 setDefaultRenderer 메소드는 이러한 연관성을 얻을 수 있게 한다. 그러기 위해서 Class 객체와 그 랜더러를 제공해야 한다.

```
table.setDefaultRenderer(Color.class,
   new ColorTableCellRenderer());
```

해당 랜더러는 이제 주어진 타입의 모든 객체에 사용되어진다.

노트 : 더 유용한 셀 랜더러를 위해 셀이 선택되어 편집 포커스를 가지고 있을 때 눈에 보이는 표식이 필요하다. 그것을 위해 선택과 포커스를 나타내기 위한 셀 크기와 색상이 필요하다. 자바 룩앤필에서는 선택은 옅은 푸른 배경색으로 그것을 지시한다.

셀 크기를 사용하기 위하여 JTable 클래스의 getCellRect 메소드를 호출하라. 색상을 선택하기 위하여 getSelectionBackground와 getSelectionForeground를 호출한다.

그러나, 포커스를 나타내기 위하여 동일한 기준이 있지 않는 것처럼 보인다. 만약 예제 프로

그램을 실행시켜 세가지 편집 가능한 컬럼 중에서 셀을 클릭하면 텍스트 필드, 콤보 박스 그리고 체크 박스 각각이 포커스를 나타내기 위하여 다른 색상 구조를 사용하는 것을 볼 수 있을 것이다.

팁 : 만약 랜더러가 텍스트 문자열 또는 아이콘을 표현한다면 DefaultTableCellRenderer 클래스를 상속할 수 있다. 그것을 가지고 선택에 대한 처리와 포커스를 다룰 수 있다.

셀 편집

셀 편집을 가능하게 하기 위하여 테이블 모델은 isCellEditable 메소드를 정의함으로써 어떤 셀이 편집가능한지에 대해 알려주어야 한다. 컬럼이 편집 가능하도록 할 필요가 있을 것이다. 예제 프로그램에서는 4 개의 컬럼에 편집을 허용하고 있다.

```
public boolean isCellEditable(int r, int c)
{   return c == NAME_COLUMN
            || c == MOON_COLUMN
            || c == GASEOUS_COLUMN
            || c == COLOR_COLUMN;
}
private static final int NAME_COLUMN = 0;
private static final int MOON_COLUMN = 2;
private static final int GASEOUS_COLUMN = 3;
private static final int COLOR_COLUMN = 4;
```

노트 : AbstractTableModel은 false를 반환하기 위하여 isCellEditable 메소드를 정의한다.
반면에, DefaultTableModel은 true를 반환하기 위하여 isCellEditable 메소드를 오버라이드한다.

만약 예제 6-10의 프로그램을 실행시키면, Gaseous 컬럼의 체크 박스를 클릭해서 체크 마크를 on 또는 off로 바꿀 수 있다. 만약 Moon 컬럼을 클릭하면 콤보 박스가 나타난다.(그림 6-29) 그러한 콤보 박스가 셀 편집기로 설치되는 방법을 상세하게 볼 수 있을 것이다.

마지막으로, 첫번째 컬럼의 셀을 클릭해서 포커스를 얻어라. 타이핑을 해서 셀 내용을 변화시킬 수 있다.

그림 6-29 : 셀 편집기

작동되는 것은 DefaultCellEditor 클래스의 세가지 변수이다. DefaultCellEditor는 JTextField, JCheckBox 또는 JComboBox를 가지고 생성될 수 있다. JTable 클래스는 자동적으로 자신의 랜더러를 공급하지 않는 모든 편집 가능한 셀을 위해 Boolean 셀을 위한 체트 박스 편집기와 텍스트 필드 편집기를 설치한다. 텍스트 필드는 사용자에게 테이블 모델의 getValueAt 메소드의 반환 값에 toString 메소드를 적용하여 얻어지는 문자열을 편집하게 해 준다.
편집이 완성될 때 셀 편집기는 setValueAt 메소드를 호출함으로써 결과를 모델에 되돌려 넘긴다. 편집된 결과를 적절한 값으로 넘기기 위하여 여러분 모델의 setValueAt 메소드를 정의하는 것은 여러분에게 달려 있다.

주의 : 디폴트 텍스트 필드 편집기에서 셀을 문자열로 바꾸는 것은 쉽다. getValueAt를 호출하고 toString를 적용해서 그러한 작업을 한다. 그 역으로 바꾸기 위해서는 편집이 끝났을 때 셀 편집기가 결과 값인 String 객체를 사용해서 setValueAt를 호출한다. SetValueAt 메소드는 그 문자열을 파서할 수 있어야 한다. 예를 들면, 셀이 정수를 저장하려면 Integer.parseInt를 호출해야 한다.

콤보 박스 편집기를 사용하기 위하여 수동으로 셀 편집기를 설치할 수 있다. JTable 컴포넌트는 어떤 값이 특정한 타입에 적당한 지를 모른다. Moon 컬럼에서 사용자가 0과 20 사이의 임의의 값을 선택 가능하도록 하고 싶다. 다음에 콤보 박스를 초기화하는 코드가 있다.

```
JComboBox moonCombo = new JComboBox();
for (int i = 0; i <= 20; i++)
   moonCombo.addItem(new Integer(i));
```

DefaultCellEditor을 생성하기 위하여 생성자에 콤보 박스를 제공한다.

```
TableCellEditor moonEditor = new DefaultCellEditor(moonCombo);
```

다음, 우리는 편집기를 설치할 필요가 있다. 색상 셀 랜더러와 달리 이 편집기는 객체 *타입*에 의존하지 않는다. 타입 Integer의 모든 객체를 필수적으로 사용하는 것은 원하지 않는다. 대신에 특정 컬럼에 그것을 설치할 필요가 있다. JTable 클래스는 TableColumn 타입의 객체에 테이블 컬럼에 대한 정보를 저장한다. TableColumnModel 클래스는 컬럼을 관리한다.(그림 6-30은 가장 중요한 테이블 클래스 사이의 관계를 보여 준다.) 만약 동적으로 컬럼을 삽입하거나 제거하기를 원하지 않는다면 테이블 컬럼 모델을 많이 사용하지는 않을 것이다. 그러나 특정한 TableColumn 객체를 얻기 위하여 컬럼 객체에 그것을 요청하도록 컬럼 모델을 사용할 필요가 있다.

```
TableColumnModel columnModel = table.getColumnModel()
```

```
TableColumn moonColumn
    = columnModel.getColumn(PlanetTableModel.MOON_COLUMN);
```

그리하여 셀 편집기를 설치할 수 있다.

```
MoonColumn.setCellEditor(moonEditor);
```

만약 디폴트 값보다 더 큰 셀을 원한다면 열 높이를 설정해 주어야 한다.

```
Table.setRowHeight(100);
```

테이블의 모든 열은 같은 높이이다.

그림 6-30 : 테이블 클래스들 사이의 관계

커스텀 편집기

예제 프로그램을 다시 실행시키고 색상을 클릭하라. 행성에 새로운 색을 넣도록 색상 선택기가 팝업된다. 색상을 선택하고 OK를 클릭하라. 셀 색상이 갱신된다.(그림 6-31을 보라)

그림 6-3 : 색상 선택기를 갖는 셀 색상 편집

색상 셀 편집기는 표준 테이블 셀 편집기가 아니고 일반적으로 지원되는 것이다. 색상 셀 편집기를 생성하기 위하여 TableCellEditor 인터페이스를 적용해야 한다. 이 인터페이스는 약간 지루하다. 이 섹션의 끝부분에는 편집이 완성된 경우 모델을 갱신하는 방법을 설명하고 있는데 만약 일반적인 편집기에 관심이 없다면 이 섹션의 끝부분을 읽지 않고 넘어 가고 싶을 것이다.

노트 : http://java.sun.com/docs/books/tutorial/uiswing/components/table.html에서 자바 스윙 튜토리얼(tutorial)은 일반적인 셀 편집기를 적용하기 위하여 DefaultCellEditor를 상속할 수 있는 방법을 보여준다. DefaultCellEditor 슈퍼클래스를 가상의 컴포넌트(텍스트 필드, 체크 박스 또는 콤보 박스로 생성되어야 한다.)에 제공해야 하기 때문에 그 기술은 애매한 면이 있다.

TableCellEditor 인터페이스의 getTableCellEditorComponent 메소드는 셀을 랜더하기 위한 컴포넌트를 요구한다. 그것은 focus 인수가 없다는 것을 제외하고 TableCellRenderer의 getTableCellRenderer 메소드와 꼭 같다. 셀이 편집될 때 포커스를 가지고 있는 것으로 여겨진다. 편집기 컴포넌트는 임시로 랜더러를 대신한다. 그렇지만 팝업 편집기를 가지고 있다면 랜더러를 상속한 편집기 클래스를 가질 수 있어서 포커스에 대해 true 인자를 가진 getTableCellRendererComponent를 호출한다.

```
class ColorTableCellEditor
```

```
    extends ColorTableCellRenderer
    implements TableCellEditor
{ public Component getTableCellEditorComponent(JTable table,
      Object value, Boolean isSelected, int row, int column)
    {return getTableCellRendererComponent(table, value,
          isSelected, true, row, column);
    }
}
```

다음으로 사용자가 셀을 클릭할 때 편집기를 팝업되게 해 보자.

JTable 클래스는 이벤트가 편집 프로세스를 받을 수 있는지 아닌지를 인식할 수 있는(마우스 클릭과 같은) 이벤트를 사용하는 편집기를 호출한다. 편집기는 까다롭지 않아서 모든 이벤트를 받아들인다.

```
public boolean isCellEditable(EventObject anEvent)
{  return true;
}
```

그러나 만약 이 메소드가 false를 반환한다면 테이블은 번거롭게 편집기 컴포넌트를 삽입하지 않을 것이다.

일단 편집기 컴포넌트가 설치되면 동일한 이벤트로써 shouldSelectCell 메소드가 호출된다. 예를 들면, 외부의 다이얼로그를 팝업하는 등의 방법으로 해당 메소드에서 편집을 초기화해야 한다.

```
public boolean shouldselectcell(EventObject anEvent)
{   colorDialog.setVisible(true);
    return true;
}
```

노트 : 이 예제에서, 우리는 JColorChooser 스윙 다이얼로그를 사용한다. 1권의 9장에서 다이얼로그 박스에 대한 섹션에서 그 다이얼로그를 설명하지 않았다. 그러나, 다이얼로그는 사용하기 쉽다. 색상 값을 사용할 필요가 있을 때 정적인 showDialog 메소드를 사용하면 된다. 그것은 모델 색상 다이얼로그를 나타낸다. 그 메소드는 사용자가 색상 값을 선택하고 다이얼로그를 취소했을 때 값을 반환한다. 반환 값은 선택된 색상이나 null 값(사용자가 다이얼로그를 취소할 경우)이다.

```
Color selected = JcolorChooser.showDialog(parent,
```

```
        "Title", defaultColor);
```

우리 예제에서는 더 복잡한 다이얼로그를 사용한다. 수동으로 setVisible 메소드를 사용해서 다이얼로그를 보여주거나 숨긴다. 그리고 사용자가 선택을 했는지 다이얼로그를 취소했는지 통지하기 위하여 동작 리스너를 제공한다.

만약 테이블에서 편집을 취소하거나 멈추려고 한다면(왜냐하면 사용자가 다른 테이블 셀을 클릭했기 때문에) cancelCellEditing 또는 stopCellEditing 메소드를 호출한다. 그 때 다이얼로그는 숨겨야 한다. stopCellEditing 메소드가 호출될 때 테이블에 부분적으로 편집된 값을 사용하고자 할 때가 있다. 만약 현재 값이 유효하다면 true를 반환해야 한다. 색상 선택기에서는 어떠한 값도 유효하다. 그러나 만약 다른 데이터를 편집할 때는 유효한 값만 편집기에서 받을 수 있다.

테이블은 여러분의 편집기에 CellEditorListener를 설치할 것이다. 리스너를 추가하거나 제거하고 이벤트를 발생시키기 위해 메소드를 사용해야 할 필요가 있다. 예제 6-10에서 이것을 위한 코드를 볼 수 있다.

다이얼로그 작업이 끝났을 때, 사용자가 다이얼로그를 승인했는지 취소했는지에 따라 편집기는 "editing stopped" 또는 "editing canceled" 이벤트를 발생시킨다. 색상 다이얼로그를 생성하면 승인과 취소 응답을 사용할 수 있다.

```
colorDialog = JColorChooser.createDialog(null,
    "Planet Color", false, colorChooser,
    new ActionListener()  //OK button listener
    {  public void actionPerformed(ActionEvent event)
        {    fireEditingStopped();
        }
    },
    new ActionListener()  // Cancel button listener
    {  public void actionPerformed(ActionEvent event)
        {   fireEditingCanceled();
        }
    });
```

다이얼로그 작업을 끝내고 발생된 이벤트를 트리가 받거나, 편집 작업을 멈추었을 때 편집은 완성된다. 트리에 편집 프로세스로부터 얻은 값을 사용하고자 한다면 getCellEditorValue 메소드를 호출하면 된다. 단지 사용자가 선택한 색상을 반환한다.

```
public Object getCellEditorValue()
{   return colorChooser.getColor();
}
```

이렇게 해서 편집기를 일반적으로 적용하는 것에 대한 설명을 마무리 하였다.
이제 셀을 편집 가능하게 하는 방법과 편집기를 설치하는 방법은 알게 되었다. 그러나 사용자가 편집한 값을 가지고 모델을 갱신하는 방법에 대한 설명이 남아 있다. 편집이 완성될 때 JTable 클래스는 테이블 모델의 다음 메소드를 호출한다.

```
void setValueAt(Object value, int r, int c)
```

새로운 값을 저장할 메소드를 오버라이드 할 필요가 있다. value 인자는 셀 편집기에 의해 반환된 객체이다. 만약 셀 편집기가 체크 박스라면 그것은 Boolean이고 텍스트 필드라면 문자열이다. 만약 콤보 박스로부터 그 값을 얻어낸다면 그것은 사용자가 선택한 객체가 된다.

만약 value 객체가 적절한 타입을 가지지 않는다면 그것을 전환할 필요가 있다. 그것은 숫자가 텍스트 필드에서 편집될 때 가장 흔하게 일어난다. 예제에서는 콤보 박스가 Integer 객체를 사용하는데 이때 어떤 전환 작업도 필요 없다.

예제 6-10 : TableCellRenderTest.java

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

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

/* the planet table model specifies the values, rendering
```

```
   and editing properties for the planet data
*/

class PlanetTableModel extends AbstractTableModel
{  public String getColumnName(int c)
    {   return columnNames[c];
    }

   public Class getColumnClass(int c)
    {   return cells[0][c].getClass();
    }

   public int getColumnCount()
    {   return cells[0].length;
    }

   public int getRowCount()
    {   return cells.length;
    }

   public Object getValueAt(int r, int c)
    {   return cells[r][c];
    }

   public void setValueAt(Object obj, int r, int c)
    {   cells[r][c] = obj;
    }

   public boolean isCellEditable(int r, int c)
    {  return c == NAME_COLUMN
          || c == MOON_COLUMN
          || c == GASEOUS_COLUMN
          || c == COLOR_COLUMN;
    }

   public static final int NAME_COLUMN = 0;
```

```
public static final int MOON_COLUMN = 2;
public static final int GASEOUS_COLUMN = 3;
public static final int COLOR_COLUMN = 4;

private Object[][] cells =
    {   { "Mercury", new Double(2440),   new Integer(0),
            Boolean.FALSE,  Color.yellow,
            new  ImageIcon("Mercury.gif")
          },
        { "Venus", new Double(6052), new Integer(0),
            Boolean.FALSE,  Color.yellow,
            new ImageIcon("Venus.gif")
          },
        { "Earth", new Double(6378), new Integer(1),
            Boolean.FALSE,  Color.blue,
            new  ImageIcon("Earth.gif")
          },
        { "Mars", new Double(3397), new Integer(2),
            Boolean.FALSE, Color.red,
            new  ImageIcon("Mars.gif")
          },
        { "Jupiter", new Double(71492), new Integer(16),
            Boolean.TRUE,  Color.orange,
            new  ImageIcon("Jupiter.gif")
          },
         { "Saturn", new Double(60268), new Integer(18),
            Boolean.TRUE,  Color.orange,
            new  ImageIcon("Saturn.gif")
          },
        { "Uranus", new Double(25559), new Integer(17),
            Boolean.TRUE,  Color.blue,
            new  ImageIcon("Uranus.gif")
          },
        { "Neptune", new Double(24766), new Integer(8),
            Boolean.TRUE,  Color.blue,
            new  ImageIcon("Neptune.gif")
```

```
            },
          { "Pluto", new Double(1137), new Integer(1),
               Boolean.FALSE, Color.black,
               new ImageIcon("Pluto.gif")
            }
        };

   private String[] columnNames =
      {  "Planet", "Radius", "Moons", "Gaseous", "Color", "Image"
       };
}

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

      TableModel model = new PlanetTableModel();
       JTable table = new JTable(model);

      // set up renderers and editors

        table.setDefaultRenderer(Color.class,
           new ColorTableCellRenderer());
        table.setDefaultEditor(Color.class,
           new ColorTableCellEditor());

       JComboBox moonCombo = new JComboBox();
      for (int i = 0; i <= 20; i++)
           moonCombo.addItem(new Integer(i));
       TableColumnModel columnModel = table.getColumnModel();
```

```
        TableColumn moonColumn
            = columnModel.getColumn(PlanetTableModel.MOON_COLUMN);
        moonColumn.setCellEditor(new DefaultCellEditor(moonCombo));

        // show table

        table.setRowHeight(100);
        Container contentPane = getContentPane();
        contentPane.add(new JScrollPane(table), "Center");
    }
}

class ColorTableCellRenderer implements TableCellRenderer
{  public Component getTableCellRendererComponent(JTable table,
       Object value, boolean isSelected, boolean hasFocus,
       int row, int column)
    {   panel.setBackground((Color)value);
        return panel;
    }

  /* the following panel is returned for all cells, with
     the background color set to the Color value of the cell
   */

  private JPanel panel = new JPanel();
}

class ColorTableCellEditor extends ColorTableCellRenderer
    implements TableCellEditor
{   ColorTableCellEditor()
    {  // prepare color dialog

       colorChooser = new JColorChooser();
       colorDialog = JColorChooser.createDialog(null,
           "Planet Color", false, colorChooser,
           new ActionListener() // OK button listener
```

```
        {   public void actionPerformed(ActionEvent event)
            {   fireEditingStopped();
            }
         },
        new ActionListener() // Cancel button listener
        {   public void actionPerformed(ActionEvent event)
            {   fireEditingCanceled();
            }
         });
 }

public Component getTableCellEditorComponent(JTable table,
   Object value, boolean isSelected, int row, int column)
{  /* this is where we get the current Color value
       We store it in the dialog in case the user starts editing
     */
     colorChooser.setColor((Color)value);
    return getTableCellRendererComponent(table, value,
       isSelected, true, row, column);
 }

public boolean isCellEditable(EventObject anEvent)
 {   return true;
 }

public boolean shouldSelectCell(EventObject anEvent)
 {   // start editing
     colorDialog.setVisible(true);

   // tell caller it is ok to select this cell
    return true;
 }

public void cancelCellEditing()
 {   // editing is canceled--hide dialog
     colorDialog.setVisible(false);
```

```
}

public boolean stopCellEditing()
{   // editing is complete--hide dialog
     colorDialog.setVisible(false);

  // tell caller is is ok to use color value
    return true;
}

public Object getCellEditorValue()
{   return colorChooser.getColor();
}

public void addCellEditorListener(CellEditorListener l)
{   listenerList.add(CellEditorListener.class, l);
}

public void removeCellEditorListener(CellEditorListener l)
{   listenerList.remove(CellEditorListener.class, l);
}

protected void fireEditingStopped()
{   Object[] listeners = listenerList.getListenerList();
   for (int i = listeners.length - 2; i >= 0; i -= 2)
        ((CellEditorListener)listeners[i+1]).
                editingStopped(event);
}

protected void fireEditingCanceled()
{   Object[] listeners = listenerList.getListenerList();
   for (int i = listeners.length - 2; i >= 0; i -= 2)
        ((CellEditorListener)listeners[i+1]).
                editingCanceled(event);
}
```

```
    private Color color;
    private JColorChooser colorChooser;
    private JDialog colorDialog;
    private EventListenerList listenerList
        = new EventListenerList();
    private ChangeEvent event
        = new ChangeEvent(this);
}
```

java.swing.JTable

- void setRowHeight(int height)
  height **픽셀로 테이블의 모든 행의 높이를 설정한다**.
- Rectangle getCellRect(int row, int column, boolean includeSpacing)
  한 테이블 셀의 사각형 바운드를 반환한다.
  전달인자: row, column **셀의 행과 열**
  includeSpacing **셀주변의 공간이 포함되면** true
- Color getSelectionBackground()
- Color getSelectionForeground()
  선택된 셀을 위해 사용되는 배경색과 전경색을 반환한다.

java.swing.table.TableModel

- Class getColumnClass(int columnIndex)
  해당 컬럼내에 값들을 위한 클래스를 가져온다. 해당 정보는 셀 렌더러와 편집기에 의해서 사용된다.

java.swing.table.TableCellRenderer

- Component getTableCellRendererComponent(JTable table, Object value, boolean selected, Boolean hasFocus, int row, int column)
  테이블 셀을 렌더러하기 위해 호출되는 paint **메소드를 갖는 컴포넌트를 반환한다**.
  전달인자: table **렌더링할 셀을 포함하고 있는 테이블**
  value **레더링할 셀**
  selected **현재 셀이 선택되어 있으면** true
  hasFocus **현재 셀이 포커스되어 있으면** true
  row, column **셀의 행과 열**

javax.swing.table.TableColumn Model

- TableColumn getColumn(int index)

  주어진 인덱스를 갖는 컬럼을 설명하는 테이블 컬럼 객체를 얻어온다.

javax.swing.TableColumn

- void setCellEditor(TableCellEditor editor)
- void setCellRenderer(TableCellRenderer renderer)

  해당 컬럼내에 모든 셀들에 대한 셀 편집기와 렌더러를 설정한다.

java.swing.table.DefaultCellEditor

- DefaultCellEditor(JComboBox comboBox)

  선택된 셀 값들을 위한 콤보 박스를 제공하는 셀 편집기를 구성한다.

java.swing.CellEditor

- boolean isCellEditable(EventObject event)

  만일 이벤트가 해당 셀에 대한 편집 과정을 초기화하는 데 적합하면 true**를 반환한다.**
- boolean shouldSelectCell(EventObject anEvent)

  편집 과정을 시작한다. 만일 편집된 셀이 선택되었으면 true**를 반환한다. 일반적으로, 여러분은** true**를 반환하길 원한다. 그러나 여러분이 셀 선택을 변경할 편집 과정을 원치 않으면** flase**를 반환한다.**
- void cancelCellEditing()

  편집 과정을 취소한다. 여러분은 부분적인 편집을 포기할 수 있다.
- boolean stopCellEditing()

  결과를 사용할 의도로 편집 과정을 중단한다. 편집된 값이 추출하는 데 적합한 상태라면 true**를 반환한다.**
- Object getCellEditorValue()

  편집된 결과를 반환한다.
- void addCellEditorListener(CellEditorListener l)
- void removeCellEditorListener(CellEditorListener l)

  기본 셀 편집기 리스너를 추가하고 제거한다.

java.swing.table.TableCellEditor

- Component getTableCellEditorComponent(JTable table, Object value, boolean selected, int row, int column)

  테이블 셀을 렌더링할 paint **메소드를 갖는 컴포넌트를 반환한다.**

  전달인자:

	table	**렌더링할 셀을 포함하고 있는 테이블**
	value	**렌더링할 셀**
	selected	**현재 셀이 선택되어 있다면** true

row, column 셀의 행과 열

java.swing.JColorChooser

- JColorChooser()
  초기 값으로 흰색을 갖는 색상 선택자를 구성한다.
- Color getColor()
- void setColor(Color c)
  해당 색상 선택자의 현재 색상을 설정하거나 가져온다.
- static JDialog createDialog(Component parent, String title, boolean modal, JColorChooser chooser, ActionListener okListener, ActionListener cancelListener)
  색상 선택자를 포함하는 다이얼로그 박스를 생성한다.
  전달인자: parent 다이얼로그를 팝업할 컴포넌트
  title 다이얼로그 박스 프레임의 제목
  modal 해당 셀이 다이얼로그를 종료할 때까지 블록되면 true
  chooser 다이얼로그에게 추가될 색상 선택자
  okListener, cancelListener OK와 Cancel 버튼들의 리스너들
- static Color showDialog(Component parent , String title, Color initialColor)
  색상 선택자를 포함하고 있는 모달 다이얼로그 박스를 생성하고 보여준다.
  전달인자: parent 다이얼로그를 팝업할 컴포넌트
  title 다이얼로그 박스 프레임의 제목
  initialColor 초기 선택된 색상

행과 열로 작업하기

이 절에서 여러분은 테이블의 행과 열을 다루는 방법을 보게 될 것이다. 본 절을 통해 읽게 되는 것처럼, 여러분은 스윙 테이블이 아주 비대칭적이라는 것을 알게 된다-행과 열상에 수행할 수 있는 여러가지 다른 연산들이 존재한다. 테이블 컴포넌트는 객체들의 임의의 2차원 격자가 아니라 데이터베이스 질의 결과 같은 같은 구조를 갖는 정보의 행들을 디스플레이하기 위해 최적화 되었다. 여러분은 이러한 것을 본절을 통해 보게 될 것이다.

컬럼 재조절

TableColumn 클래스는 여러분에게 컬럼들의 크기 재조절 기능을 제공한다. 여러분은 다음 메소드들을 통해 선호 크기와 최소, 최대 크기를 설정할 수 있다.

void setPreferredWidth(int width)

void setMinimunWidth(int width)

void setMaximumWidth(int width)

이 정보는 컬럼들을 레이아웃하는 테이블 컴포넌트에 의해서 사용된다.

사용자에게 컬럼을 재조정을 허락할 것인지 조절하기 위해 다음 메소드를 사용하라.

void setResizable(boolean resizable)

다음 메소드로 컬럼을 재조절 할 수 있다.

void setWidth(int width)

컬럼이 재조정될 때, 기본은 테이블의 전체 크기를 변경하지 않고 남겨둔다. 물론, 재조정된 컬럼의 폭 증가나 김소는 다른 컬럼들에게 전파되야 한다. 기본 행위는 모든 컬럼들의 크기를 재조정된 컬럼의 오른쪽에 맞춰 변경하는 것이다. 이것은 아주 좋은 기본 정책이다. 왜냐하면 사용자에게 원하는 모든 컬럼들을 원하는 폭으로 왼쪽에서 오른쪽으로 이동할 수 있도록 해주기 때문이다.

여러분은 JTable 클래스의 다음 메소드를 사용하여 표6-2에 있는 여러 행위를 설정할 수 있다.

void setAutoResizeMode(int mode)

표 6-2 : 재조정 모드

모드	행위
AUTO_RESIZE_OFF	다른 컬럼들을 재조정하지 말라; 테이블의 크기를 변경한다.
AUTO_RESIZE_NEXT_COLUMN	오직 다음 컬럼만을 재조정한다.
AUTO_RESIZE_SUBSEQUENT_COLUMNS	모든 연속되는 컬럼들을 동등하게 재조정한다. 이것은 기본 행위이다.
AUTO_RESIZE_LAST_COLUMN	오직 마지막 컬럼만을 재조정한다.
AUTO_RESIZE_ALL_COLUMNS	테이블내의 모든 컬럼들을 재조정한다. 이것은 좋은 선택이 아니다. 왜냐하면 사용자들이 원하는 크기로 복수 개의 컬럼들을 정렬하게 끔 만들기 때문이다.

행과 열과 셀들 선택하기

선택 모드에 따라, 사용자는 테이블내의 행과 열 또는 개별적인 셀들을 선택할 수 있다. 기

본으로, 행 선택이 활성화 된다. 셀 내부를 선택함으로써 전체 행을 선택한다. 다음 호출은 행 선택을 금지한다.

```
table.setRowSelectionAllowed(flase)
```

행 선택이 가능해 질 때, 여러분은 사용자가 단일 행을 선택하거나 연속적인 행을 선택하거나 임의의 일련의 행들을 선택할 것인지를 조정할 수 있다. 여러분은 *선택 모델*을 추출하고 이 모델의 setSelectionMode 메소드를 사용할 필요가 있다.

```
table.getSelectionMode().setSelectionMode(mode);
```

mode는 다음 세가지 값들 중 하나이다.

```
ListSelectionModel.SINGLE_SELECTION
ListSelectionModel.SINGLE_INTERVAL_SELECTION
ListSelectionModel.MULTIPLE_ INTERVAL_SELECTION
```

열 선택은 기본으로 금지된다. 여러분은 다음 메소드 호출을 통해 해지할 수 있다.

```
table.setColumnSelectionAllowed(true);
```

만일, 행과 열 선택이 모두 해지되면, 모든 마우스 클릭은 "십자가(+)" 영역을 선택할 수 있다(그림6-32참조).

그림 6-32 : 행과 열 선택

여러분은 어느 행과 열들이 getSelectedRows와 getSelectedColumns 메소드에 의해서 호출되었는지 발견할 수 있다. 이들 모두 선택된 아이템들의 인덱스의 int[] 배열을 반환한다.

여러분은 여러분의 사용자들에게 다음 호출에 의해 행이나 열을 선택하는 것 대신에 개별적인 셀들을 선택하게 할 수 있다.

```
table.setCellSelectionEnabled(true);
```

일단 셀 선택이 가능해지면, 행과 열 선택을 위한 설정은 무시된다. 그림 6-33은 셀 선택이 가능한 테이블을 보여준다.

그림 6-33 : 셀들의 범위 선택

여러분은 작동중인 셀 선택을 보기위해 예제 6-11의 프로그램을 수행할 수 있다. 선택 모드에서 행과 열 또는 셀 선택을 가능하게 하고 어떻게 행위가 변경하는지 지켜보라.

열 숨기기와 출력하기

JTable 클래스의 removeColumn 메소드는 테이블 뷰로부터 한 컬럼을 삭제한다. 컬럼 데이터는 실제 모델로부터 제거되는 것은 아니다 – 단지 뷰로부터 감춰진다. RemoveColumn 메소드는 TableColumn 전달인자를 갖는다. 만일 여러분이 컬럼 숫자를 갖는다면(가령, getSelectedColumns를 호출하여), 실제 테이블 컬럼 객체를 위한 테이블 모델을 요청할 필요가 있다:

```
table.addColumn(new TableColumn(modelColumnIndex));
```

여러분은 모델의 같은 컬럼을 뷰하는 다중 테이블 컬럼들을 갖을 수 있다. 그러나 실제 모델에 새로운 컬럼을 추가하거나 모델로부터 컬럼을 삭제하는 JTable의 메소드는 없다. 또한 행들을 감추거나 보여주는 것도 없다. 만일 여러분이 행들을 감추길 원한다면, 이전에 보았던 정렬 필터와 유사한 필터 모델을 생성해야 한다.

기본 테이블 모델에서 행 추가와 삭제

DefaultTableModel은 TableModel 인터페이스를 구현한 구체적인 클래스로서 2차원 격자에 객체들을 저장한다. 만일 여러분이 이미 테이블 정렬에 데이터를 갖고 있다면, 모든 데이터를 기본 테이블 모델에 복사하는 하는 것보다 작은 데이터 셋으로부터 테이블을 재빨리 만드는 것이 더 편리하다. DefaultTableModel 클래스는 행과 열을 추가하거나 행들을 제거하는 메소드를 가지고 있다.

addRow와 addColumn 메소드는 새로운 데이터의 행이나 열을 추가한다. 여러분은 새로운 데이터를 유지하는 Object[] 배열이나 벡터를 제공한다. 또한 여러분은 addColumn 메소드로 새로운 컬럼에 대한 이름을 제공할 필요가 있다. 이들 메소드들은 새로운 데이터를 격자의 끝에 추가한다. 중간에 행을 추가할 때 insertRow 메소드를 사용한다. 격자의 중간에 컬럼을 추가하는 메소드는 제공하지 않는다.

반대로, removeRow 메소드는 모델로부터 한 열을 제거한다. 컬럼을 제거하는 메소드는 제공하지 않는다.

JTable객체는 테이블 모델 리스너 그 자체로서 등록되기 때문에, 모델은 데이터가 삽입되거나 삭제될 때 테이블에게 통보한다. 이 때, 테이블은 디스플레이를 갱신한다.

예제 6-11의 프로그램은 작동중인 선택과 편집을 모두 보여준다. 기본 테이블 모델은 단훈한 데이터 셋을 포함한다. 편집 메뉴는 다음 명령을 포함하고 있다

- 모든 선택될 컬럼을 숨긴다.
- 이전에 숨겼던 모든 컬럼들을 보여준다.
- 모델로부터 선택된 행들을 제거한다.
- 모델의 끝에 데이터의 한 행을 추가한다.

이 예제로서 스윙 테이블에 대한 논의를 마친다. 테이블들은 개념적으로 트리보다 이해하기 쉽다. 왜냐하면 하부의 데이터 모델(2차원 격자)이 쉽게 가시화되기 때문이다. 그러나 테이블 컴포넌트는 실제 트리 컴포넌트 보다 아주 복잡하다. 컬럼 헤더, 재조정 가능한 컬럼들, 그리고 특정 컬럼에 만 해당하는 렌더러 그리고 편집기는 이를 더욱 복잡하게 한다. 이 절에서, 우리는 여러분이 실제적으로 만나게 되는 이들 주제들에 관심을 두었다 : 데이터 베이스 정보 디스플레이, 정렬, 그리고 커스텀 셀 렌더링 그리고 편집. 만일 여러분이 보다 고급 기능들이 필요하다면, 우리는 다시 한번 , Kim Topley의 Core Java Foundation Classed나 David Geary의 Graphics Java를 추천한다.

예제 6-11 : TableSelectionTest.java

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

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

class TableSelectionFrame extends JFrame
   implements ActionListener
{  public TableSelectionFrame()
   {  setTitle("TableSelectionTest");
      setSize(300, 200);
```

```
 addWindowListener(new WindowAdapter()
      {   public void windowClosing(WindowEvent e)
          {   System.exit(0);
          }
      } );

// set up multiplication table

model = new DefaultTableModel(10, 10);

for (int i = 0; i < model.getRowCount(); i++)
    for (int j = 0; j < model.getColumnCount(); j++)
          model.setValueAt(
          new Integer((i + 1) * (j + 1)), i, j);

table = new JTable(model);

Container contentPane = getContentPane();
contentPane.add(new JScrollPane(table), "Center");

// create menu

JMenuBar menuBar = new JMenuBar();
  setJMenuBar(menuBar);

JMenu selectionMenu = new JMenu("Selection");
  menuBar.add(selectionMenu);

rowsItem = new JCheckBoxMenuItem("Rows");
  rowsItem.setSelected(table.getRowSelectionAllowed());
  rowsItem.addActionListener(this);
  selectionMenu.add(rowsItem);

columnsItem = new JCheckBoxMenuItem("Columns");
  columnsItem.setSelected(table.getColumnSelectionAllowed());
  columnsItem.addActionLis tener(this);
```

```
        selectionMenu.add(columnsItem);

      cellsItem = new JCheckBoxMenuItem("Cells");
        cellsItem.setSelected(table.getCellSelectionEnabled());
        cellsItem.addActionListener(this);
        selectionMenu.add(cellsItem);

       JMenu  tableMenu = new JMenu("Edit");
        menuBar.add(tableMenu);

      showColumnsItem = new JMenuItem("Show Columns");
        showColumnsItem.addActionListener(this);
        tableMenu.add(showColumnsItem);

       hideColumnsItem = new  JMenuItem("Hide Columns");
        hideColumnsItem.addActionListener(this);
        tableMenu.add(hideColumnsItem);

      addRowItem = new JMenuItem("Add Row");
        addRowItem.addActionListener(this);
        tableMenu.add(addRowItem);

       removeRowsItem = new  JMenuItem("Remove Rows");
        removeRowsItem.addActionListener(this);
        tableMenu.add(removeRowsItem);

       clearCellsItem = new  JMenuItem("Clear Cells");
        clearCellsItem.addActionListener(this);
        tableMenu.add(clearCellsItem);
   }

  public void actionPerformed(ActionEvent event)
  {  Object source = event.getSource();
     if (source == rowsItem)
       {   table.setRowSelectionAllowed(rowsItem.isSelected());
           table.clearSelection();
```

```
  }
else if (source == columnsItem)
  {   table.setColumnSelectionAllowed(columnsItem.isSelected());
      table.clearSelection();
  }
else if (source == cellsItem)
  {   table.setCellSelectionEnabled(cellsItem.isSelected());
      table.clearSelection();
  }
else if (source == hideColumnsItem)
{   int[] selected = table.getSelectedColumns();
    TableColumnModel columnModel = table.getColumnModel();

   /* remove columns from view, starting at the last
       index so that the column numbers aren't affected
      */

    for (int i = selected.length - 1; i >= 0; i--)
     {   TableColumn column
             = columnModel.getColumn(selected[i]);
         table.removeColumn(column);

       // store removed columns for "show columns" command

         removedColumns.add(column);
      }
  }
else if (source == showColumnsItem)
{   // restore all removed columns
   for (int i = 0; i < removedColumns.size(); i++)
         table.addColumn((TableColumn)removedColumns.get(i));
      removedColumns.clear();
  }
else if (source == removeRowsItem)
{   int[] selected = table.getSelectedRows();
```

```
      /* remove rows from model, starting at the last
         index so that the row numbers aren't affected
        */

      for (int i = selected.length - 1; i >= 0; i--)
         model.removeRow(selected[i]);
    }
  else if (source == addRowItem)
  {  // add a new row to the multiplication table in the model

     Integer[] newCells = new Integer[model.getColumnCount()];
    for (int i = 0; i < newCells.length; i++)
         newCells[i]
          = new Integer((i + 1) * (model.getRowCount() + 1));
     model.addRow(newCells);
    }
  else if (source == clearCellsItem)
  {  // set all selected cells to 0

    for (int i = 0; i < table.getRowCount(); i++)
       for (int j = 0; j < table.getColumnCount(); j++)
          if (table.isCellSelected(i, j))
             table.setValueAt(new Integer(0), i, j);
    }
}

private DefaultTableModel model;
private JTable table;

private JMenuItem showColumnsItem;
private JMenuItem hideColumnsItem;

private JMenuItem addRowItem;
private JMenuItem removeRowsItem;

private JMenuItem clearCellsItem;
```

```
    private JCheckBoxMenuItem rowsItem;
    private JCheckBoxMenuItem columnsItem;
    private JCheckBoxMenuItem cellsItem;

   private ArrayList removedColumns = new ArrayList();
}
```

javax.swing.JTable

- void setAutoResizeMode(int mode)
  테이블 컬럼들의 자동 크기 재조절을 위한 모드를 설정한다.
  전달인자: mode AUTO_RESIZE_OFF, AUTO_RESIZE_NEXT_COLUMN, AUTO_RESIZE_SEQUENT_COLUMN, AUTO_RESIZE_LAST_COLUMN, AUTO_RESIZE_ALL_COLUMN **중의 하나**
- ListSelectionModel getSelectionModel()
  리스트 선택 모델을 반환한다. 여러분은 행과 열 그리고 셀 선택간의 선택을 위해 이 모델이 필요하다.
- void setCellSelectionEnabled(boolean b)
  만일 b**가** true**이면, 개별적인 셀들이 선택된다. 만일** false**이면 행과 열들이 선택될 수 있다.**
- void setRowSelectionAllowed(boolean b)
  만일 b**가** true**이면, 사용자가 셀들을 선택할 때 행들이 선택될 수 있다.**
- void setColumnSelectionAllowed(boolean b)
  만일 b**가** true**이면, 사용자가 셀들을 선택할 때 컬럼들이 선택될 수 있다.**
- void addColumn(TableColumn column)
  테이블 뷰에 한 컬럼을 추가한다.
- void moveColumn(int from, int to)
  from **테이블 인덱스의 컬럼을** to**로 이동한다. 오직 뷰만이 영향을 받는다.**
- void removeColumn(TableColumn column)
  뷰로 부터 주어진 컬럼을 제거한다.

javax.swing.table.TableColumn

- TableColumn(int modelColumnIndex)
  주어진 인덱스를 갖는 모델 컬럼을 보여주기 위한 테이블을 구성한다.
- void setPreferredWidth(int width)

- void setMinimumWidth(int width)
- void setMaximunWidth(int width)
  해당 테이블 컬럼의 선호하는, 최소 그리고 최대 폭을 width로 설정한다.
- void setWidth(int width)
  해당 컬럼의 실제 폭을 width로 설정한다.
- void setResizable(boolean b)
  b가 true이면, 해당 컬럼은 재조정 가능하다.

javax.swing.ListSelectionModel

- void setSelectionMode(int mode)
  전달인자: mode SINGLE_SELECTION, SINGLE_INTERVAL_SELECTION, 그리고 MULTIPLE_INTERVAL_SELECTION 중의 하나

javax.swing.DefaultTableModel

- void addRow(Object[] rowData)
- void addColumn(Object columnName, Object[] columnData)
  테이블 모델의 끝에 데이터의 행과 열을 추가한다.
- void insertRow(int row, Object[] rowData)
  row 인덱스에 데이터의 행을 추가한다.
- removeRow(int row)
  모델로부터 주어진 행을 제거한다.
- void moveRow(int start, int end, int to)
  start와 end 인덱스 사이의 모든 행들을 새로운 위치인 to로 옮긴다.

스타일 텍스트 컴포넌트

1권에서 기본 텍스트 컴포넌트 클래스인 JtextField와 JTextArea 클래스를 논의했었다. 물론 이들 클래스는 사용자로부터의 입력 텍스트를 얻을 때 아주 유용하다. 다른 유용한 클래스인 Jeditorpane은 HTML과 RFT 형식의 텍스트를 편집하고 디스플레이한다.(RTF는 "rich text format"으로 문서 교환을 위해 많은 많은 마이크로소프트 응용 프로그램에 의해서 사용된다. 이 포맷은 마이크로소프트 자신의 응용 프로그램들간에도 제대로 작동하지 않는 좋지 않은 문서 포맷이다. 우리는 이 책에서 RTF의 기능에 대해서는 다루지 않는다. 선은 오직 제한된 지원만을 한다고 주장한다. 우리는 앞으로 이 기능이 빠지더라도 놀라지 않는다. 왜냐하면 선은 HTML 지원을 강화하는 데 모든 자원들을 투자하고 있기 때문이다.)

솔직하게, 이 시점에서 JeditorPane은 아주 제한적이다. HTML 렌더러는 단순한 파일만을 디

스플레이한다. 그러나 여러분이 웹 상에서 발견하는 많은 복잡한 페이지들은 제대로 지원하지 않는다. HTML편집기는 불안정하고 아주 협소하다. 물론 HTML편집기를 제외한 몇몇 응용 프로그램들만이 사용자에게 HTML 텍스트 편집을 요구한다. 따라서 이것은 주요한 제약사항은 아니다.

우리는 JEditorPane의 완벽한 응용 프로그램이 HTML 포맷으로 된 도움말을 디스플레이하는 프로그램이라고 생각한다. 여러분은 직접 제공한 도움말 파일들을 조절할 수 있기 때문에 JeditorPane이 제대로 표현하지 못하는 기능들을 피해갈 수 있다.

노트 : 강력한 도움말 시스템에 대한 자세한 정보는 JavaHelp를 참조하라.

http://java.sun.com/products/javahelp/index.html

노트 : JEditorPane의 JTextPane 하위 클래스는 특정 폰트와 텍스트 포맷 뿐만 아니라 내장된 컴포넌트들을 갖는 스타일 텍스트를 유지할 수 있다. 우리는 이 책에서 해당 컴포넌트를 다루지는 않을 것이다. 만일 여러분이 여러분 프로그램의 사용자가 스타일 텍스트를 입력하게 해주는 컴포넌트를 구현할 필요가 있다면, JDK에 포함되어 있는 스타일패드 데모의 구현을 살펴보라.

예제 6-12의 프로그램은 HTML 페이지의 내용들을 보여주는 편집 패인을 포함하고 있다. 텍스트 필드에 URL을 입력하라. URL은 “http:”나 “file:”로 시작해야 한다. 그런 다음, 로드 버튼을 클릭한다. 선택된 HTML 페이지는 편집 패인에서 디스플레이된다(그림6-34).

그림 6-34 : HTML 페이지를 디스플레이하는 편집 패인

하이퍼링크들은 활성화된다: 만일 여러분이 한 링크를 클릭하면, 응용 프로그램이 이를 로드한다. Back 버튼은 이전 페이지로 되돌아간다.

이 프로그램은 사실 아주 단순한 브라우져이다. 물론 상업용 브라우져에서 기대하는 페이지 캐싱이나 북마크 리스트 같은 편안한 기능들을 갖고 있지는 않다. 또한 편집 패인은 아직 애플릿을 디스플레이 하지는 않는다.

만일 여러분이 Editable 체크 박스를 클릭하면, 편집 패인은 편집가능한 상태가 된다. 여러분은 텍스트로 입력하고 텍스트를 삭제하기 위해 백스페이스키를 사용할 수 있다. 또한 이 컴포넌트는 자르고 복사하고 붙이기 위한 CTRL+X, CTRL+C 그리고 CTRL+V 숏컷을 이해한다. 그러나 이들 단순한 명령어들 조차도 편집기상에 오작동의 원인이 되기 쉽다. 응용 프로그램을 구동시킨 윈도우를 지켜보면 종종 스택 예외상황을 보게 된다.

컴포넌트가 편집가능할 때, 하이퍼링크들은 비활성된다. 또한 편집 모두가 활성화될 때 몇몇 웹 페이지들에서 자바 스크립트 명령어와 주석 그리고 다른 태그들을 볼 수 있다(그림 6-35참조). 예제 프로그램은 여러분에게 편집 기능을 살펴볼 수 있게 해준다. 그러나 우리는 여러분의 프로그램에서 이들 기능을 빼길 추천한다.

팁 : 기본적으로, JEditorPane은 편집 모드이다. 여러분은 이를 비활성화 시키기 위해 editorPane.setEditable(false)를 호출해야 한다.

그림 6-35 : 편집 패인의 편집 모드 상태

예제 프로그램에서 본 편집 패인의 기능들은 아주 사용하기 쉽다. 여러분은 새로운 문서를 로드하기 위해 setPage 메소드를 사용한다. 전달인자는 문자열이나 URL 객체중 하나이다. JEditorPane클래스는 JExtComponent 클래스를 확장한다. 그러므로, 여러분은 setText 메소드 또한 호출할 수 있다 – 이 메소드는 단순히 일반 텍스트를 디스플레이한다.

하이퍼링크 클릭을 리슨하기 위해, 여러분은 HyperlinkListener를 추가한다. HyperlinkListener인터페이스는 사용자가 링크상에 마우스를 올려놓거나 클릭할 때 호출되는 hyperlinkUpdate라는 메소드를 가지고 있다. 이 메소드는 HyperlinkEvent 타입의 전달인자를 갖는다.

여러분은 어떤 종류의 이벤트가 발생하는 지 찾기 위해 getEventType 메소드를 호출할 필요가 있다. 다음은 세가지 가능한 반환 값들이다:

HyperlinkEvent.EventType.ACTIVATED
HyperlinkEvent.EventType.ENTERED
HyperlinkEvent.EventType.EXITED

첫번째 값은 사용자가 하이퍼링크를 클릭했다는 것을 나타낸다. 이 경우, 여러분은 기본적으

로 새로운 링크를 개발하길 원한다. 여러분은 마우스가 링크상에 있을 때 툴팁 같은 몇가지 비쥬얼 피드백을 제공하기 위해 두 번째와 세 번째 값들을 사용할 수 있다.

노트 : HyperlinkListener 인터페이스에 활성화, 엔트리 그리고 종료를 다루는 세 개의 별도 메소드들이 없는 이유는 완전한 미스터리이다.

HyperlinkEvent클래스의 getURL메소드는 하이퍼링크의 URL을 반환한다. 예를 들면, 다음은 사용자가 링크를 따라다니며 활성화시키는 하이퍼링크 리스너를 설치하는 방법이다.

```
editorPane.addHyperlinkListener(new HyperlinkListener()
        {   public void hyperlinkUpdate(HyperlinkEvent event)
            {   if (event.getEventType()
                    == HyperlinkEvent.EventType.ACTIVATED)
                {   try
                    {   // remember URL for back button
                        urlStack.push(event.getURL().toString());
                        // show URL in text field
                        url.setText(event.getURL().toString());

                        editorPane.setPage(event.getURL());
                    }
                    catch(IOException e)
                    {   editorPane.setText("Error: " + e);
                    }
                }
            }
        });
```

이벤트 핸들러는 단순히 URL을 가져오고 편집 패인을 갱신한다. setPage 메소드는 IOException을 발생할 수 있다. 이 경우, 우리는 일반 텍스트로 에러 메시지를 보여준다.

예제 6-12의 프로그램은 HTML 도움말 시스템에 필요한 모든 기능들을 보여준다. 내부 측면에서 보면 JEditorPane은 트리나 테이블 컴포넌트보다 다소 복잡하다. 그러나 만일 여러분이 텍스트 편집기나 커스텀 텍스트 포맷을 작성할 필요가 없다면 이러한 복잡성은 여러분에게

서 숨겨진다.

예제 6-12 : EditorPaneTest.java

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

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

class EditorPaneFrame extends JFrame
{  public EditorPaneFrame()
   {  setTitle("EditorPaneTest");
      setSize(600, 400);
      addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
            {  System.exit(0);
            }
         } );

     // set up text field and load button for typing in URL

      url = new JTextField(30);

      loadButton = new JButton("Load");
       loadButton.addActionListener(new ActionListener()
         {  public void actionPerformed(ActionEvent event)
            {  try
```

```
            {   // remember URL for back button
                urlStack.push(url.getText());

                editorPane.setPage(url.getText());
              }
             catch(IOException e)
             {   editorPane.setText("Error: " + e);
              }
           }
       });

// set up back button and button action

 backButton = new JButton("Back");
  backButton.addActionListener(new ActionListener()
     {   public void actionPerformed(ActionEvent event)
        {   if (urlStack.size() <= 1) return;
            try
           {   // get URL from back button
                urlStack.pop();
              // show URL in text field
              String urlString = (String)urlStack.peek();
                url.setText(urlString);

                editorPane.setPage(urlString);
              }
             catch(IOException e)
             {   editorPane.setText("Error: " + e);
             }
           }
       });

// set up editor pane and hyperlink listener

 editorPane = new JEditorPane();
  editorPane.setEditable(false);
```

```
editorPane.addHyperlinkListener(new HyperlinkListener()
    {   public void hyperlinkUpdate(HyperlinkEvent event)
        {   if (event.getEventType()
                == HyperlinkEvent.EventType.ACTIVATED)
            {   try
                {   // remember URL for back button
                    urlStack.push(event.getURL().toString());
                    // show URL in text field
                    url.setText(event.getURL().toString());

                    editorPane.setPage(event.getURL());
                }
                catch(IOException e)
                {   editorPane.setText("Error: " + e);
                }
            }
        }
      });

// set up checkbox for toggling edit mode

editable = new JCheckBox();
  editable.addActionListener(new ActionListener()
    {   public void actionPerformed(ActionEvent event)
        {   editorPane.setEditable(editable.isSelected());
        }
      });

Container contentPane = getContentPane();
 contentPane.add(new JScrollPane(editorPane), "Center");

// put all control components in a panel

JPanel panel = new JPanel();
 panel.add(new JLabel("URL"));
  panel.add(url);
```

```
        panel.add(loadButton);
        panel.add(backButton);
        panel.add(new JLabel("Editable"));
        panel.add(editable);

        contentPane.add(panel, "South");
    }

    private JTextField url;
    private JCheckBox editable;
    private JButton loadButton;
    private JButton backButton;
    private JEditorPane editorPane;
    private Stack urlStack = new Stack();
}
```

javax.swing.JEditorPane

- void setPage(URL url)
  해당 url**의 페이지를 편집 패인에 로드한다**.
- void addHyperlinkListener(HyperlinkListener listener)
  해당 편집 패인에 하이퍼링크 리스너를 추가한다.

javax.swing.event.HyperlinkListener

- void hyperlinkUpdate(hyperlinkEvent event)
  하이퍼링크가 선택될 때마다 호출된다.

javax.swing.HyperlinkEvent

- void getURL()
  선택된 하이퍼링크의 URL**을 반환한다**.

슬라이드와 진행 메터

이 절에서는 표면적으로는 유사하게 보이는 두개의 스윙 컴포넌트에 대해서 논의한다: 사용자에게 선형 규모로 일정량의 입력을 가능하게 해주는 슬라이더와 프로그램에서 선형 규모의 출력을 가능하게 해주는 진행(progress) **메터**.

슬라이드들

슬라이더는 스크롤 바와 유사하다. 그러나 스크롤 바는 일반적으로 뷰포인트를 옮기는 의향을 가지고 있다. 반면에 슬라이더는 값들을 제공하기 위해 사용된다. 이러한 행위상의 유사점에도 불구하고 API**는 다르다.**

스크롤 바는 Adjustable **인터페이스를 구현하고 조정**(adjustment) **이벤트들을 제거했다. 비록 슬라이더가 둘사이의 공통성을 표현하기 위한 좋은 인터페이스처럼 보일지라도 슬라이더는** Adjustable **인터페이스를 구현하지 않는다. 대신에** ChangeEvent **객체들을 보냄으로써 자신의 리스너에게 알린다.**

슬라이더를 구성하는 가장 일반적인 방법은 다음과 같다.

```
JSlider slider = new JSlider(min, max, initialValue);
```

만일 여러분이 최대,최소, 초기 값을 생략한다면, 이들은 0,100 **그리고** 50**으로 각기 초기화된다.**

또는, 만일 여러분이 수직 방향의 슬라이더를 원한다면 다음의 생성자 호출을 사용한다.

```
JSlider slider = new JSlider(SwingConstants.VERTICAL, min, max, initialValue);
```

이 들 생성자는 그림 6-36**의 가장 상위에 있는 슬라이더처럼 보통**(plain) **슬라이더를 생성한다. 여러분은 슬라이더를 치장하는 방법을 보게 될 것이다.**

그림 6-36 : 슬라이더들

사용자가 슬라이더 바를 움직이면, 슬라이더의 값은 최소와 최대 값사이에서 이동한다. 값이 변할 때, ChangeEvent**는 모든 변경 리스너에게 전달된다. 변경을 통보하기 위해, 여러분은** addChangeListener **메소드를 호출하고** ChangeListener **인터페이스를 구현한 객체를 설치할 필요가 있다.** ChangeListener **인터페이스는 하나의 메소드인** stateChanged**를 가지고 있다. 이 메소드에서, 여러분은 슬라이더 값을 추출할 수 있다:**

```
public void stateChanged(ChangeEvent event)
{       JSlider slider = (JSlide)event.getSource();
        int value = slider.getValue();
        …
}
```

여러분은 점(tick)들을 보여줌으로써 슬라이더를 치장할 수 있다. 예를 들면, 예제 프로그램에서 두번째 슬라이더는 다음의 설정을 사용한다:

```
slider.setMajorTickSpacing(20);
slider.setMinorTickSpacing(5);
```

슬라이더는 매 20 단위마다 큰 점 표기(tick mark)로, 매 5 단위마다 작은 점 표기로 장식된다. 단위는 픽셀이 아니라 슬라이더의 값을 가르친다.

이러한 지시 사항들은 오직 점 표기를 위한 단위만을 설정하는 것이다. 실제 점 표기를 나타나게 하려면, 다음 호출이 필요하다.

```
slider.setPaintTicks(true);
```

최대와 최소 점 표기들은 독립적이다. 예를 들면, 여러분은 최대 점 표기를 매 20단위로, 최소 점 표기를 매 7 단위로 설정할 수 있다.

여러분은 슬라이더를 점으로 순간 이동(snap to ticks)시킬 수 있다. 사용자가 스냅 모드에서 슬라이더를 드레깅할 때마다 가장 근접한 점으로 바로 이동한다. 여러분은 다음 호출을 사용하여 스냅 모드를 활성화할 수 있다.

```
slider.setSnapToTicks(true);
```

여러분은 다음을 호출하여 최대 점 표기를 위한 점 표기 라벨을 요청할 수 있다.

```
slider.setPaintLabels(true);
```

예를 들면, 0에서 100사이 범위와 최대 점 공간이 20인 슬라이더에서 점은 0,20,40,60,80, 그리고 100에 표기된다.

또한 여러분은 문자열이나 아이콘 처럼 다른 점 표기를 제공할 수 있다(그림 6-36 참조). 과정은 다소 복잡하다. 여러분은 new Integer(tickValue)와 Component 타입의 값을 갖는 해쉬 테이블을 채울 필요가 있다. 그런 다음 setLabelTable 메소드를 호출한다. 컴포넌트는 점 표기 아래 위치한다. 일반적으로 여러분은 JLable 객체를 사용한다. 다음은 점 라벨을 A,B,C,D,E,F로 만드는 방법이다.

```
Hashtable labelTable = new HashTable();
LabelTable.put(new Integer(0), new JLabel("A"));
LabelTable.put(new Integer(0), new JLabel("B"));
…
LabelTable.put(new Integer(0), new JLabel("F"));
slider.setLabelTable(labelTable);
```

또한 예제 6-13은 점 라벨로 아이콘을 갖는 슬라이더를 보여준다.

팁 : 만일 여러분의 점 표기나 라벨이 보여지지 않으면 setPaintTicks(true)와 setPaintLabels(true)을 검사해 보라.

마지막으로, 여러분이 메탈 룩앤필을 사용한다면 여러분의 슬라이더에 비쥬얼 강화를 추가하여 최소 값에서부터 현재 값사이를 채울 수 있다. 그림 6-36에서 4번째와 5번째 슬라이더는 채워져 있다. 여러분은 다음처럼 클라이언트속성을 설정하여 슬라이더를 채운다.

```
slider.putClientProperty("Jslider.isFilled", Boolean.TRUE);
```

5번째 슬라이더는 다음 호출에 의해서 반대 방향을 갖게 된다.

```
slider.setInverted(true);
```

예제 프로그램은 슬라이더의 집합을 통해 이러한 모든 비쥬얼 효과를 보여준다. 각 슬라이더는 오른쪽에 위치한 텍스트 필드에 현재 슬라이더 값을 표시해 주는 변경 리스너를 가지고 있다.

예제 6-13: SliderTest.java

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

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

class SliderTestFrame extends JFrame
{  public SliderTestFrame()
   {  setTitle("SliderTest");
```

```
 setSize(400, 300);
 addWindowListener(new WindowAdapter()
    {  public void windowClosing(WindowEvent e)
         {  System.exit(0);
         }
     } );

 // set up grid bag layout and constraints

 getContentPane().setLayout(new GridBagLayout());
constraints = new GridBagConstraints();
 constraints.weighty = 100;
 constraints.gridwidth = 1;
 constraints.gridheight = 1;
  constraints.gridx = 0;
 constraints.gridy = 0;

// add sliders with various decorations

JSlider slider = new JSlider();
 addSlider(slider, "Plain");

slider = new JSlider();
  slider.setPaintTicks(true);
  slider.setMajorTickSpacing(20);
  slider.setMinorTickSpacing(5);
 addSlider(slider, "Ticks");

slider = new JSlider();
  slider.setPaintTicks(true);
  slider.setSnapToTicks(true);
  slider.setMajorTickSpacing(20);
  slider.setMinorTickSpacing(5);
 addSlider(slider, "Snap to ticks");

slider = new JSlider();
```

```
 slider.setPaintTicks(true);
 slider.setMajorTickSpacing(20);
 slider.setMinorTickSpacing(5);
slider.putClientProperty("JSlider.isFilled", Boolean.TRUE);
 addSlider(slider, "Filled");

slider = new JSlider();
 slider.setPaintTicks(true);
 slider.setMajorTickSpacing(20);
 slider.setMinorTickSpacing(5);
slider.putClientProperty("JSlider.isFilled", Boolean.TRUE);
 slider.setInverted(true);
addSlider(slider, "Inverted");

slider = new JSlider();
 slider.setPaintTicks(true);
 slider.setPaintLabels(true);
 slider.setMajorTickSpacing(20);
 slider.setMinorTickSpacing(5);
addSlider(slider, "Labels");

slider = new JSlider();
 slider.setPaintLabels(true);
 slider.setPaintTicks(true);
 slider.setMajorTickSpacing(20);
 slider.setMinorTickSpacing(5);

Hashtable labelTable = new Hashtable();
labelTable.put(new Integer(0), new JLabel("A"));
labelTable.put(new Integer(20), new JLabel("B"));
labelTable.put(new Integer(40), new JLabel("C"));
labelTable.put(new Integer(60), new JLabel("D"));
labelTable.put(new Integer(80), new JLabel("E"));
 labelTable.put(new Integer(100), new JLabel("F"));

 slider.setLabelTable(labelTable);
```

```
    addSlider(slider, "Custom labels");

    slider = new JSlider();
     slider.setPaintTicks(true);
     slider.setPaintLabels(true);
     slider.setSnapToTicks(true);
     slider.setMajorTickSpacing(20);
     slider.setMinorTickSpacing(20);

    labelTable = new Hashtable();

    // add card images

     labelTable.put(new Integer(0),
         new JLabel(new ImageIcon("9h.gif")));
     labelTable.put(new Integer(20),
         new JLabel(new ImageIcon("10h.gif")));
     labelTable.put(new Integer(40),
         new JLabel(new ImageIcon("jh.gif")));
     labelTable.put(new Integer(60),
         new JLabel(new ImageIcon("qh.gif")));
     labelTable.put(new Integer(80),
         new JLabel(new ImageIcon("kh.gif")));
     labelTable.put(new Integer(100),
         new JLabel(new ImageIcon("ah.gif")));

      slider.setLabelTable(labelTable);
     addSlider(slider, "Icon labels");
  }

 public void addSlider(JSlider s, String description)
 {  // create text field that is shown next to slider
    final TextField textField = new TextField(4);

   // update text field when the slider value changes
     s.addChangeListener(new ChangeListener()
```

```
        {   public void stateChanged(ChangeEvent event)
            {   JSlider source = (JSlider)event.getSource();
                textField.setText("" + source.getValue());
            }
        });

    // add three components into the next row

      constraints.gridx = 0;
      constraints.anchor = GridBagConstraints.WEST;
      constraints.fill = GridBagConstraints.NONE;
      constraints.weightx = 0;
      getContentPane().add(new JLabel(description), constraints);

      constraints.gridx++;
      constraints.anchor = GridBagConstraints.CENTER;
      constraints.fill = GridBagConstraints.HORIZONTAL;
      constraints.weightx = 100;
      getContentPane().add(s, constraints);

      constraints.gridx++;
      constraints.anchor = GridBagConstraints.WEST;
      constraints.fill = GridBagConstraints.NONE;
      constraints.weightx = 0;
      getContentPane().add(textField, constraints);

      // advance row
      constraints.gridy++;
   }

   private GridBagConstraints constraints;
}
```

javax.swing.Jslider

- JSlider()
- JSlider(int direction)

- JSlider(int min, int ,max)
- JSlider(int min, int ,max, int initialValue)
- JSlider(int direction ,int min, int , min, int initialValue)

  주어진 방향과 최소,최대, 초기 값들을 갖는 수평 슬라이더를 구성한다.

  전달인자: direction SwingConstants.HORIZONTAL과 SwingConstants.VERTICAL 중의 하나. 기본은 수평이다.

  min, min 슬라이더 값의 최소와 최대. 기본은 0과 100

  initialValue 슬라이더의 초기 값. 기본은 50이다.
- void setPaintTicks(Boolean b)

  b가 true이면 점이 디스플레이된다.
- void setMajorTickSpacing(int units)
- void setMinorTickSpacing(int units)

  주어진 슬라이더 단위에 반복해서 최대 또는 최소 점을 설정한다.
- void setPaintLabers(bolean b)

  b가 true이면 점의 라벨이 디스플레이된다.
- slider.setLabelTable(Dictionary table)

  점 라벨을 위해 사용되는 컴포넌트를 설정한다. 테이블내의 각 키/값 쌍은 new Integer(value) / component 형식을 갖는다.
- void setSnapToTicks(Boolean b)

  b가 true이면 슬라이더는 가장 근접한 점으로 바로 이동한다.

진행(progress) 바

진행바는 단순 컴포넌트이다 – 연산의 진행을 알려주는 색상으로 부분적으로 채워진 사각형이다. 기본적으로, 진행은 문자열 "n %"에 의해서 표기된다. 여러분은 그림 6-37의 오른쪽 하단부에서 진행바를 볼 수 있다.

그림 6-37 : 진행 바

여러분은 슬라이더 바를 만들듯이 최소와 최대 값 그리고 선택사항인 방향을 제공하여 진행바를 생성한다:

```
progressBar = new JprogressBar(0,1000);
progressBar = new JprogressBar(SwingConstants.VERTICAL ,0,1000);
```

또한 여러분은 setMinimum과 setMaximum으로 최소와 최대를 설정할 수 있다.

슬라이더와 달리, 진행 바는 사용자에 의해서 조절될 수 없다. 여러분의 프로그램은 이를 갱신하기 위해 setValue 메소드를 호출할 필요가 있다.

만일 여러분이 다음을 호출하면 진행바는 완전한 백분율을 계산하고 문자열 "n%"를 디스플레이한다.

```
progressBar.setStringPainted(true);
```

만일 여러분이 다른 문자열을 보여주길 원한다면, setString 메소드로 이를 제공할 수 있다.

```
if(progressBar.getValue() > 9000)
    progressBar.setString("Almost Done");
```

예제 6-14의 프로그램은 시간이 소요되는 연산을 모니터하는 진행바를 보여준다.

SimulatedActivity 클래스는 초당 current 값을 열번씩 증가하는 쓰레드를 구현한다. 목표 값에 도달할 때, 쓰레드는 종료한다. 만일 여러분이 목표에 도달하기 전에 쓰레드를 종료하고 싶다면 인터럽트 해야 한다.

```
class SimulatedActivity extends Thread
{   …
   public void run()
   {   while (current < target && !interrupted())
       {   try
           {   sleep(100);
           }
           catch(InterruptedException e)
           {   return;
           }
           current++;
       }
    }

   private int current;
   private int target;
}
```

여러분이 시작 버튼을 클릭할 때, 새로운 SimulatedActivity 쓰레드가 시작된다. 진행 바를 갱신하기 위해, setValue 메소드를 호출하여 활동중인 쓰레드를 모방하는 것이 쉬운 것처럼 인다. 그러나 이것은 안전한 쓰레드(thread safe)가 아니다. 1장에서 상기해 보면 여러분이 호출한 스윙 메소드들은 오직 이벤트 디스패치 쓰레드에서 파생된 것이다.

대신에, 여러분은 진행 상태와 진행 바를 갱신하기 위한 스레드를 주기적으로 폴(poll)하는 타이머를 작동시킬 수 있다. 이것은 실제적인 접근 방법이다. 왜냐하면 일반 작업 쓰레드는 진행 바의 존재를 알지 못하기 때문이다.

주의 : 만일 작업 쓰레드가 과정을 모니터하는 진행바를 알고 있다면, 진행 바 값을 직접 설정할 수 없다. 이벤트 디스패치 쓰레드에서 값을 설정하기 위해, 작업 쓰레드는 SwingUtilities.invokeLater 메소드를 사용할 수 있다.

1장을 상기해 보면 스윙 타이머는 해당 리스너들의 actionPerformed 메소드를 호출하고 이들 호출은 이벤트 디스패치 스레드에서 발생한다. 이 말은 타이머 콜백에서 스윙 컴포넌트들의 갱신은 안전하다는 것을 의미한다. 다음은 예제 프로그램의 타어머 콜백이다. 모의 활동의 현재 값은 텍스 영역과 진행 바에서 모두 디스플레이된다. 만일 끝에 도달하면, 타이머는 멈추고 시작 버튼이 다시 가능하게 된다.

```
public void actionPerformed(ActionEvent event)
        {   int current = activity.getCurrent();

            // show progress
            textArea.append(current + "\n");
             progressBar.setValue(current);

           // check if task is completed
            if (current == activity.getTarget())
              {   activityMonitor.stop();
                  startButton.setEnabled(true);
              }
         }
```

예제 6-14는 완전한 프로그램 코드를 보여준다.

예제 6-14 : ProgressBarTest.java

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

```
import javax.swing.*;
import javax.swing.event.*;

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

class ProgressBarFrame extends JFrame
{  public ProgressBarFrame()
    {   setTitle("ProgressBarTest");
        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 panel with button and progress bar

        JPanel panel = new JPanel();
        startButton = new JButton("Start");
        progressBar = new JProgressBar();
         progressBar.setStringPainted(true);
         panel.add(startButton);
         panel.add(progressBar);
        contentPane.add(new JScrollPane(textArea), "Center");
        contentPane.add(panel, "South");
```

```
   // set up the button action

     startButton.addActionListener(
        new ActionListener()
           {  public void actionPerformed(ActionEvent event)
              {  progressBar.setMaximum(1000);
                activity = new SimulatedActivity(1000);
                 activity.start();
                 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");
                  progressBar.setValue(current);

                // check if task is completed
                 if (current == activity.getTarget())
                   {  activityMonitor.stop();
                      startButton.setEnabled(true);
                   }
              }
            });
 }

private Timer activityMonitor;
private JButton startButton;
```

```
    private JProgressBar progressBar;
    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;
}
```