제 17 장 고급 자바 파운데이션 클래스

자바 파운데이션 클래스는 자바의 그래픽 사용자 인터페이스의 기반이 되는 클래스들을 포괄하는 이름이다. 여기에는 추상 윈도우 툴킷, 스윙 컴포넌트, 2 차원 그래픽 등 앞에서 다룬 주제들이 모두 포함된다.
이 장에서는 AWT 와 스윙 컴포넌트에 대한 기본 지식에서 한 발 더 나아가 좀더 나은 사용자 인터페이스를 제공해주는 클립보드, 드랙앤드롭, 실행 취소/재실행, 포커스 처리 등을 다룬다.

이 장에서는 다음을 다룬다.

- 클립보드
- 드랙앤드롭
- 실행 취소/재실행
- 포커스 처리

17.1. 클립보드

17.1.1. 클립보드와 소유주

일반적으로 윈도우 시스템에서 제공하는 클립보드(clipboard)는 각 윈도우 응용 프로그램들이 공유하는 데이터 저장 장소이다.
java.awt.datatransfer 패키지의 Clipboard 클래스는 이러한 클립보드를 나타내는데, 사용자가 만든 클립보드를 자바 프로그램 내의 데이터 저장 장소로 구현할 수도 있고, 또, 시스템의 클립보드를 사용할 수도 있다.
시스템의 클립보드를 사용할 경우에는 자바 프로그램 내부의 데이터 공유는 물론, 자바가 아닌 다른 윈도우 프로그램과도 데이터 공유가 가능하다.
시스템 클립보드를 나타내는 Clipboard 객체는 다음과 같이 구할 수 있다.

```
Clipboard clipbd = Toolkit.getDefaultToolkit().getSystemClipboard();
```

Clipboard 클래스의 메소드들은 다음과 같다.

(1) public String getName(); 클립보드 이름을 반환한다. 클립보드 생성자에서 이름을 지정하며, 시스템 클립보드의 이름은 System 이다.
(2) public void setContents(Transferable contents, ClipboardOwner owner); 클립보드의 내용을 변경한다. 이때 클립보드의 소유주를 지정한다.
(3) public Transferable getContents(Object requestor); 클립보드의 내용을 가져온다. 인자로 주어지는 요청자는 사용되지 않으므로 그냥 null 을 넘겨주면 된다.

클립보드에는 소유주(owner) 개념이 있는데 클립보드에 데이터를 넣을 때 소유주를 지정하게 된다. 따라서, 다른 소유주가 클립보드에 데이터를 넣을 경우 소유권을 상실하게 되는데, 이 상황을 알 수 있도록 ClipboardOwner 라는 인터페이스가 존재한다.
ClipboardOwner 인터페이스는 하나의 메소드를 정의한다.

```
public void lostOwnership(Clipboard clipboard, Transferable contents);
```

클립보드에 setContents() 메소드를 호출할 때 현재와 다른 소유주를 지정하면 이전 소유주가 소유권을 잃게 되므로 lostOwnership() 메소드가 호출된다.

17.1.2. 전송 객체

클립보드의 내용은 항상 전송 객체를 의미하는 Transferable 인터페이스 객체이다. 클립보드의 내용은 일반적으로 여러 가지 형태로 표현 가능하다. 예를 들어, HTML 텍스트를 클립보드로 복사한 경우, HTML 텍스트, 서식 있는 텍스트, 단순한 텍스트 등의 종류로 클립보드 내용을 사용할 수 있다. 이때 자료의 종류를 DataFlavor 클래스로 나타내는데, DataFlavor 클래스는 내부적으로 MIME(다목적 인터넷 메일 확장 형식, Multipurpose Internet Mail Extensions) 유형을 사용하여 자료의 종류를 나타낸다.
자바의 DataFlavor 클래스는 미리 정의된 몇 가지 자료 종류를 가지고 있다.

(1) public static final DataFlavor stringFlavor; 단순한 텍스트
(2) public static final DataFlavor imageFlavor; 이미지
(3) public static final DataFlavor javaFileListFlavor; 파일 목록

이 외에도 MIME 유형을 사용하여 자료 종류를 정의할 수 있다.

17.1.3. 클립보드 사용 예제

클립보드를 사용하는 프로그램을 작성해보자. 이 프로그램은 사용자 클립보드와 시스템 클립보드를 모두 사용하여, 텍스트를 복사하면 사용자 클립보드와 시스템 클립보드 모두에 텍스트를 복사해넣고, 붙여넣기를 하면 현재 시스템 클립보드의 소유권을 가지고 있으면 사용자 클립보드의 내용을 붙여넣고, 소유권이 없으면 시스템 클립보드의 내용을 붙여넣는다.
대부분의 환경에서는 사용자 클립보드를 별도로 사용할 필요가 없지만, 여기에서는 클립보드 사용 방법을 보여주기 위해서 분리하였다. 만약 프로그램 내부에서 좀더 복잡한 자료 종류를 사용하고, 프로그램 외부로는 단순한 자료 종류만 내보내고 복잡한 자료 종류는 내보내지 않길 원하는 상황이라면 이렇게 클립보드를 분리해서 사용하는 것이 효율적일 수 있다.

<따라하기 시작 – 클립보드 사용 예제>
이 프로그램은 두 개의 텍스트에리어를 사용한다. 하나는 복사와 붙여넣기를 통해 클립보드 데이터를 변경하기 위한 것이며, 다른 하나는 클립보드 데이터의 내용을 표시하기 위한 것이다.
자바에서 시스템 클립보드의 내용 변화를 항상 추적할 수는 없기 때문에 클립보드의 내용을 항상 보여줄 수는 없지만, 일반적으로 프로그램 내에서 붙여넣기를 할 때 무엇이 붙여질 것인지를 미리 아는 것은 시스템 클립보드의 소유권 변화와 해당 프로그램 윈도우의 포커스 이벤트를 사용하여 검사할 수 있다.
시스템 클립보드로 데이터를 복사해 넣으면 시스템 클립보드의 소유주가 되는데 시스템 클립보드의 소유권을 잃게 되는 상황은 다른 프로그램이 시스템 클립보드에 데이터를 변경한 때문이다. 붙여넣기를 하려면 윈도우가 포커스를 가지고 있어야 하므로 소유권을 잃은 경우에는 윈도우가 포커스를 받을 때 시스템 클립보드에 어떤 내용이 있는지 확인하면 된다.

1. 먼저 스윙 컴포넌트를 사용하여 사용자 인터페이스를 구현한다.
COPY 와 PASTE 라는 레이블을 가진 두 개의 버튼이 있으며, 이 버튼의 액션 이벤트를 처리하여 클립보드로 데이터를 전송하고 또 클립보드 데이터를 가져와서 붙여넣기를 할 것이다.
입력을 받기 위한 텍스트에리어인 editText 멤버 필드는 JTextArea 클래스가 시스템 클립보드 사용을 하도록 copy(), cut(), paste() 메소드들이 구현되어 있기 때문에 이를 오버라이드하여 사용하지 못하도록 하였다. 따라서 윈도우의 경우 Ctrl+C, Ctrl+V, Ctrl+X 등의 키 이벤트가 발생하더라도 아무런 일도 하지 않는다.
클립보드 내용을 보여줄 텍스트에리어인 clipText 멤버 필드는 편집을 목적으로 하지 않으므로 편집을 할 수 없도록 setEditable(false)를 호출하였다.
boolean 값인 lostOwnership 멤버 필드는 시스템 클립보드 소유권을 잃었는지 여부를 알려준다. 초기값은 시스템 클립보드를 사용하지 않았으므로 소유권이 없기 때문에 true 이다.

```
package yoonforh.clipbd;

import java.awt.*;
import java.awt.datatransfer.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

/**
```

```
 * 클립보드 사용 예제.
 * 자체 클립보드와 시스템 클립보드를 별도로 관리하면서 동일한 데이터를
 * 가질 수 있도록 한다.
 */

public class ClipboardEx extends JFrame
    implements WindowFocusListener, ClipboardOwner, ActionListener {
  JTextArea editText; // 입력을 받는 텍스트에리어
  JTextArea clipText; // 붙여넣어질 클립보드 내용을 미리 보여주는 텍스트에리어
  Clipboard clipbd; // 자체 클립보드
  boolean lostOwnership = true; // 시스템 클립보드 소유권 여부

  public ClipboardEx(String title) {
    super(title);

    // JTextArea는 CUT/COPY/PASTE 기능을 내장하고 있어
    // 직접 시스템 클립보드를 사용하므로 이것을 막도록 오버라이드한다.
    editText = new JTextArea() {
       public void cut() { }
       public void copy() { }
       public void paste() { }
      };

    // 클립보드 데이터를 보여줄 텍스트에리어. 입력을 받지 않게 한다.
    clipText = new JTextArea();
    clipText.setEditable(false);

    // 스플릿페인의 왼쪽에 들어갈, 레이블과 텍스트에리어,
    // 그리고 버튼 패널을 배치한다.
    JPanel leftPanel = new JPanel();
    leftPanel.setLayout(new BorderLayout());
    leftPanel.add(new JLabel("Edit Area"), BorderLayout.NORTH);
    leftPanel.add(new JScrollPane(editText), BorderLayout.CENTER);

    JPanel buttonPanel = new JPanel();
    JButton button = new JButton("Copy");
```

```
    button.addActionListener(this);
    buttonPanel.add(button);
    button = new JButton("Paste");
    button.addActionListener(this);
    buttonPanel.add(button);
    leftPanel.add(buttonPanel, BorderLayout.SOUTH);

    // 스플릿페인의 오른쪽에 들어갈 레이블과 텍스트에리어를 배치한다.
    JPanel rightPanel = new JPanel();
    rightPanel.setLayout(new BorderLayout());
    rightPanel.add(new           JLabel("Clipboard           Contents"),
BorderLayout.NORTH);
    rightPanel.add(new JScrollPane(clipText));

    JSplitPane splitter = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                    true, /* 구분막대를 움직이는 동안에도 계속 렌더 */
                    leftPanel,
                    rightPanel);
     // 스플릿페인의 구분막대가 가운데 오도록 한다.
    splitter.setResizeWeight(0.5);

    setContentPane(splitter);

    // 자체 클립보드
    clipbd = new Clipboard("local clipboard");

    // 윈도우 닫으면 종료
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // 윈도우가 포커스를 잃거나 얻을 때 이벤트 처리
    // 시스템 클립보드 내용이 바뀌는 것을 알 수는 없으나,
    // 클립보드의 소유권이 없는 것은 알 수 있으므로,
    // 소유권이 없을 경우에는 포커스를 받을 때 클립보드 데이터를 가져와서 보여준다.
    addWindowFocusListener(this);
  }
```

```
public static void main(String[] args) {
  JFrame f = new ClipboardEx("clipboard example");

  f.setBounds(10, 10, 600, 400);
  f.setVisible(true);
  }

}
```

2. 윈도우 포커스 이벤트를 처리한다.

포커스를 얻은 경우 현재 시스템 클립보드의 소유주가 아니면, 시스템 클립보드의 데이터를 가져와 clipText 컴포넌트에 그 내용을 보여준다. 포커스를 잃은 경우에는 별도의 처리가 필요없다.

```
/**
 * 윈도우가 포커스를 얻은 경우 처리
 */
public void windowGainedFocus(WindowEvent evt) {
  // 현재 시스템 클립보드의 소유권을 잃은 상태이면
  // 시스템 클립보드의 내용을 가져와 보여준다.
  if (lostOwnership) {
    Transferable t
      = getToolkit().getSystemClipboard().getContents(null);
    // clipText 필드에 전송 객체 내용을 보여주는 메소드
    showTransferable("window gained focus", t);
  }
}

/**
 * 윈도우가 포커스를 잃은 경우 처리
 */
public void windowLostFocus(WindowEvent evt) {}
```

3. showTransferable() 메소드를 구현한다.

이 메소드는 Transferable 전송 객체의 내용을 clipText 컴포넌트에 나타낸다.

먼저 전송 객체가 지원하는 모든 자료 종류를 구하여 각, 자료 종류가 InputStream 이나

Reader 로 나타내지거나 텍스트 종류이면 그 내용이 텍스트인 것으로 간주하여 getReaderForText() 메소드를 사용하여 그 내용을 가져온다.
이 외의 자료 종류에 대해서는 간단하게 지원하지 않는 자료 종류로 표시한다.

```
/**
 * Transferable의 내용을 표시
 */
void showTransferable(String message, Transferable t) {
  clipText.setText(message + "\n");

  // 지원하는 데이터 종류를 모두 구하여 하나씩 출력한다.
  DataFlavor[] flavors = t.getTransferDataFlavors();
  for (int i = 0; i < flavors.length; i++) {
    try {
      clipText.append("Flavor : " + flavors[i] + "\n");
      if (flavors[i].isRepresentationClassInputStream()
        || flavors[i].isRepresentationClassReader()
        || flavors[i].isFlavorTextType()) {
        BufferedReader reader = new
BufferedReader(flavors[i].getReaderForText(t));
        String data = null;
        while ((data = reader.readLine()) != null) {
          clipText.append(data + "\n");
        }
      } else {
        clipText.append("unknown flavor. data is "
              + t.getTransferData(flavors[i]) + "\n");
      }
      clipText.append("\n\n");
    } catch (UnsupportedFlavorException e) {
      System.err.println("unsupported flavor error : " + flavors[i]);
    } catch (IOException e) {
      System.err.println("io error - " + e.getMessage());
    }
  }
}
```

4. 클립보드 소유권 이벤트를 처리한다.

이때 lostOwnership() 메소드는 AWT 이벤트에 의해 처리되는 것이 아니므로, 이벤트 처리 쓰레드에서 호출된다는 보장이 없다는 것에 주의해야 한다. 스윙 컴포넌트의 경우 이벤트 처리 쓰레드에서 모든 변경 동작이 일어나도록 사용해야 하므로 현재 쓰레드가 이벤트 처리 쓰레드인지 확인하고, 만약 이벤트 처리 쓰레드가 아닐 경우에는 EventQueue.invokeLater() 메소드를 사용하여 이벤트 처리 쓰레드에서 이 메소드를 다시 호출하도록 하였다.

소유권을 잃은 클립보드가 시스템 클립보드이면 lostOwnership 필드를 true 로 하고, clipText 컴포넌트에 현재 시스템 클립보드 내용을 나타낸다.

```
/**
 * 클립보드 소유권을 상실할 경우에 호출된다.
 * 이 프로그램에서는 자체 클립보드는 공유되지 않으므로,
 * 시스템 클립보드 소유권을 상실할 경우에만 호출된다.
 * 이 메소드는 이벤트 처리 쓰레드에 의해 호출되는 것이 아니어서
 * 쓰레드 처리에 주의해야 한다.
 */
public void lostOwnership(final Clipboard clipboard,
            final Transferable contents) {
  // 만약 이벤트 처리 쓰레드가 아니면 다시 이벤트 처리 쓰레드에서
  // 호출되도록 전송.
  if (!EventQueue.isDispatchThread()) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
          lostOwnership(clipboard, contents);
        }
      });
    return;
  }

  // 클립보드 소유권 상실의 경우를 처리
  // 다른 누군가가 클립보드 소유권을 취득했으므로, 해당 클립보드 데이터가 바뀌었다.
  showTransferable("lost ownership of " + clipboard.getName(),
contents);
```

```
  Clipboard sysclip = getToolkit().getSystemClipboard();
  if (clipboard == sysclip) {
    lostOwnership = true;
    Transferable t
      = sysclip.getContents(null);
    showTransferable("lost ownership", t);
  }
}
```

5. 버튼의 액션 이벤트를 처리한다.

복사의 경우에는 editText 컴포넌트의 현재 선택된 텍스트 내용을 내부 클립보드와 시스템 클립보드로 각각 붙여넣는다. 시스템 클립보드의 내용을 지정하여 시스템 클립보드의 소유주가 되므로 lostOwnership 필드 값은 false 로 한다.

StringSelection 클래스는 단순한 텍스트를 자료 종류로 가지는 전송 객체이다.

붙여넣기의 경우에는 현재 시스템 클립보드의 소유권을 가지는 경우에는 내부 클립보드의 전송 객체를 구하여 붙여넣고, 소유권이 없는 경우에는 시스템 클립보드의 전송 객체를 구하여 붙여넣는다.

```
/**
 * 버튼 이벤트 처리
 */
public void actionPerformed(ActionEvent evt) {
  String cmd = evt.getActionCommand();

  if (cmd.equals("Copy")) {
    copy();
  } else if (cmd.equals("Paste")) {
    paste();
  }
}

/**
 * 클립보드로 내용을 복사한다.
 */
void copy() {
```

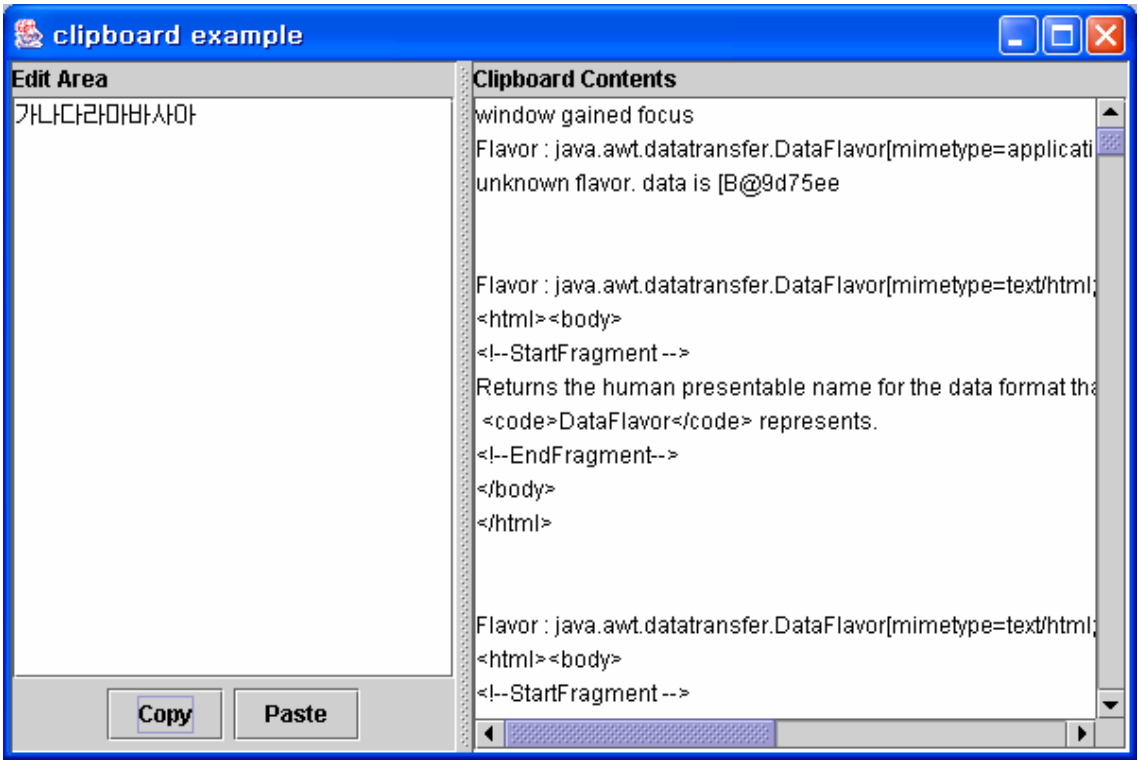
```
    String string = editText.getSelectedText();
    if (string == null) {
      return;
    }

    StringSelection sel = new StringSelection(string);

    // 자체 클립보드와 시스템 클립보드 모두에 복사
    // setContents() 메소드를 호출함으로써
    // 해당 클립보드의 소유권을 인자로 지정한 소유주가 갖게 된다.
    clipbd.setContents(sel, this);
    getToolkit().getSystemClipboard().setContents(sel, this);
    showTransferable("copy selected text", sel);
    lostOwnership = false;
  }

  /**
   * 클립보드의 내용을 붙여 넣는다.
   */
  void paste() {
    // 시스템 클립보드 소유권을 잃은 상태이면
    // 내부 클립보드의 데이터는 시스템 클립보드에 지정한 데이터보다 예전 버전이다.
    // 따라서, 시스템 클립보드로부터 직접 가져와야 한다.
    Transferable t = (lostOwnership ?
getToolkit().getSystemClipboard() : clipbd)
      .getContents(this);
    String data = getStringData(t);
    if (data != null) {
      editText.insert(data,
            editText.getCaretPosition());
    }
  }

/**
   * Transferable이 지원하는 문자열 내용을 가져온다.
   */
```

```
  String getStringData(Transferable t) {
    if (t.isDataFlavorSupported(DataFlavor.stringFlavor)) {
      try {
        return (String) t.getTransferData(DataFlavor.stringFlavor);
      } catch (UnsupportedFlavorException e) {
        e.printStackTrace();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }

    return null;
  }
```

6. 컴파일하여 실행해보자.

```
javac yoonforh\clipbd\ClipboardEx.java <엔터>

java -cp . yoonforh.clipbd.ClipboardEx <엔터>
```

프로그램이 실행되면, 먼저 왼쪽의 편집용 텍스트에리어에 텍스트를 입력하여 영역을 선택한 다음 Copy 버튼을 눌러보자.
Copy 버튼을 누름으로써 선택된 텍스트 내용을 두 개의 클립보드에 복사하게 되고, 두 클립보드의 소유주가 된다.
그 다음 다른 프로그램에서 텍스트나 이미지를 선택한 다음 복사를 해보자. 그러면 ClipboardEx 프로그램이 시스템 클립보드의 소유권을 잃게 되어 그 내용을 오른쪽의 클립보드 내용 텍스트에리어에 보여주게 된다.
이미지나 탐색기의 파일 목록의 경우 어떻게 자료 종류를 표현하는지 확인해보자.
다음 그림은 실행 예를 보여주고 있다.



[그림 1 – ClipboardEx 프로그램 실행 모습]

<따라하기 끝 – 클립보드 사용 예제>

17.2. 드래앤드롭

드래앤드롭(drag and drop)은 윈도우 시스템의 클립보드를 거치지 않고, 마우스 드래을 시작한 컴포넌트에서 마우스 드롭이 끝나는 컴포넌트로 데이터를 전송하는 특별한 사용자 인터페이스이다. 이 경우에도 역시 Transferable 전송 객체를 사용하여 데이터를 전송하게 된다.

17.2.1. 드래앤드롭의 지원 방법

드래앤드롭에는 드래을 하는 원천과 드롭이 되는 대상의 구분이 있다. 드래 원천과 드롭 대상은 각각 실제 드래이 발생하고 드롭이 일어날 컴포넌트에 대한 정보를 가지고 있으며, 드래과 드롭에 관련된 마우스 동작을 추적하여 필요한 메소드를 이벤트 방식으로 호출한다.

java.awt.dnd 패키지에는 드래앤드롭을 위한 몇 가지 인터페이스와 클래스들이 정의되어 있다.

드래앤드롭 과정에서 각 클래스와 인터페이스들의 역할을 살펴보자.

(1) DragSource 클래스 : 드래 원천을 추상화한 클래스. 이 객체에서 드래 동작 인식기를 원하는 컴포넌트와 연결시켜 만듦으로써, 컴포넌트에 발생하는 드래 동작을 추적할 수 있다.

(2) DropTarget 클래스 : 드롭할 대상을 추상화한 클래스. 드랙된 전송 객체를 컴포넌트에 드롭할 경우, 해당 컴포넌트가 드롭을 처리할 수 있으려면 이 객체를 사용하여 컴포넌트를 드롭 대상으로 등록해야 한다.

(3) DragGestureListener 인터페이스 : 드랙 원천에 등록된 컴포넌트에서 드랙 동작이 발생하면 이를 인식한 AWT 윈도우 시스템은 이 리스너 객체의 이벤트 메소드를 호출한다. 이 리스너의 이벤트 메소드에서 드랙의 시작을 설정한다.

(4) DragSourceListener 인터페이스 : 드랙 동작이 시작되면 드랙 원천의 드랙 관련 이벤트를 처리하기 위해 이 리스너 객체를 사용한다.

(5) DropTargetListener 인터페이스 : 드롭 대상이 등록한 컴포넌트 위로 드롭 관련 마우스 이벤트가 발생하면 이 리스너 객체의 이벤트 메소드들이 호출된다. 이 리스너의 이벤트 메소드에서 드롭의 성공 여부를 결정한다.

(6) DragSourceMotionListener 인터페이스 : 드랙 동작이 시작될 때 마우스가 드랙이 발생한 컴포넌트 위를 움직이면 이 리스너의 이벤트가 호출된다.

드랙앤드롭은 드랙 동작과 드롭 동작의 두 과정으로 나뉘므로, 각 동작을 지원하기 위해 필요한 과정을 살펴보자.
먼저 컴포넌트가 드랙을 지원하기 위해서는 다음을 해야한다.

(1) 드랙 원천 객체를 생성하여 드랙 동작 리스너를 컴포넌트와 함께 등록한다.
드랙 원천 객체의 생성은 DragSource 의 생성자를 이용하거나 플랫폼의 기본 드랙 원천 객체를 구한다.

```
DragSource ds = new DragSource();
```

혹은

```
DragSource ds = DragSource.getDefaultDragSource();
```

드랙 원천 객체에 드랙 동작 리스너와 컴포넌트를 등록하고 허용할 드랙 동작 유형도 함께 지정한다. 다음 메소드를 사용하면 된다.

```
ds.createDefaultDragGestureRecognizer(
  component, // 드랙이 시작하는 컴포넌트
  DnDConstants.ACTION_COPY_OR_MOVE, // 드랙 동작 유형
  new MyDragGestureListener() // 드랙 동작 리스너 객체
  );
```

(2) 사용자가 지정한 컴포넌트에서 드랙 동작을 시작하면 드랙 동작 리스너의 이 메소드가 호출된다. 이 메소드에서 드랙의 시작을 결정해야 한다.

```
public void dragGestureRecognized(DragGestureEvent dge);
```

드랙의 시작을 결정하려면 다음과 같이 DragGestureEvent 객체의 startDrag() 메소드를 호출한다. 이 메소드에서 전송될 객체를 넘겨주고, 드랙 원천 리스너를 등록하여 드랙 이벤트를 처리할 수 있게 한다.

```
dge.startDrag(cursor, // 드랙 동작에 맞는 커서
  transferable, // 드랙이 될 전송 객체
    new MyDragSourceListener() // 드랙 원천 리스너 객체
    );
```

(3) 드랙 동작 리스너 메소드에서 등록한 드랙 원천 리스너는 드랙 이벤트가 발생하는 도중에 계속해서 리스너 메소드가 호출된다. 리스너의 각 메소드를 적당히 처리함으로써 드랙 동작의 지원은 마무리된다.

컴포넌트가 드롭을 지원하기 위해서는 다음과 같은 일을 해야 한다.

(1) 먼저 드롭 대상 객체를 생성한다. 이 대상 객체에는 드롭을 받을 컴포넌트와 드롭 대상 리스너를 등록하고, 지원하는 드롭 동작을 지정한다.

```
DropTarget dt = new DropTarget(
  this, // 드롭을 받을 컴포넌트
  DnDConstants.ACTION_COPY_OR_MOVE, // 지원하는 드롭 동작
        new MyDropTargetListener() // 드롭 대상 리스너 객체
        );
```

컴포넌트에서 자신의 직접 드롭 대상 객체를 지정할 수도 있다.

```
component.setDropTarget(dt);
```

(2) 컴포넌트의 드롭 관련 이벤트가 발생하면 등록된 드롭 대상 리스너의 메소드들이 차례로 호출된다. 리스너의 각 메소드들을 적당히 처리함으로써 드롭 지원이 완성

된다.

DragSourceListener 인터페이스의 각 메소드는 다음과 같은 일을 한다.

(1) public void dragEnter(DragSourceDragEvent dsde); 드랙이 시작된 커서가 드롭 가능한 대상으로 들어간 경우 호출된다.
(2) public void dragExit(DragSourceEvent dse); 드랙이 시작된 커서가 드롭 가능한 대상을 벗어나는 경우 호출된다.
(3) public void dragOver(DragSourceDragEvent dsde) ; 드랙이 시작된 커서가 드롭 가능한 대상 위를 움직일 경우 호출된다.
(4) public void dropActionChanged(DragSourceDragEvent dsde); 사용자가 원하는 드롭 동작을 변경할 경우 호출된다. 예를 들어 마이크로소프트 윈도우에서 사용자가 마우스를 드랙하는 도중 Ctrl 키나 Shift 키 등을 누르는 경우 발생한다.
(5) public void dragDropEnd(DragSourceDropEvent dsde); 드롭이 끝난 후에 발생한다. 드롭이 성공적이었는지, 또 드롭 동작이 무엇이었는지를 알 수 있으며, 그에 따른 조치를 취할 수 있다.

DropTargetListener 인터페이스의 각 메소드는 다음과 같은 일을 한다.

(1) public void dragEnter(DropTargetDragEvent dtde); 드랙 동작이 진행되는 도중 이 리스너를 등록한 드롭 대상 컴포넌트로 커서가 들어올 때 호출된다.
(2) public void dragExit(DropTargetEvent dte) ; 드랙 동작이 진행되는 도중 이 리스너를 등록한 드롭 대상 컴포넌트를 커서가 벗어날 때 호출된다.
(3) public void dragOver(DropTargetDragEvent dtde); 드랙 동작이 진행되는 도중 이 리스너를 등록한 드롭 대상 컴포넌트 위를 마우스가 움직일 때 호출된다.
(4) public void dropActionChanged(DropTargetDragEvent dtde); 사용자가 원하는 드롭 동작을 변경할 경우 호출된다.
(5) public void drop(DropTargetDropEvent dtde); 드랙 동작이 이 리스너를 등록한 드롭 대상 컴포넌트 위에서 드롭으로 끝날 때 호출된다. 이 메소드에서 드롭을 실제로 허용하고 전송 객체를 처리하는 일을 수행해야 한다.

17.2.2. 커스텀 전송 객체

자바의 드랙앤드롭에서는 같은 자바 가상 머신 내에서의 드랙앤드롭과 서로 다른 자바 가상 머신 사이에서의 드랙앤드롭, 그리고 자바 프로그램과 자바가 아닌 윈도우 프로그램과의 드랙앤드롭을 처리하는 방식이 조금씩 다르다.
드랙 원천이 원격지로부터의 것인지 아닌지 여부는 DropTargetListener 의 drop() 메소드에서

확인할 수 있다.
여기에서 만들 전송 객체는 뒤이어 만들 드랙앤드롭 예제에서 사용할 것이다. 드랙앤드롭의 전송 객체는 드롭이 끝나면 더 이상 사용할 수 없기 때문에 drop() 메소드 안에서 이 내용을 복사해서 그 후로 드랙 원천의 전송 객체로 사용할 수 있는 복제 전송 객체를 만들 것이다. 이 복제 전송 객체는 내부적으로 문자열과 이미지, 파일 목록 세 가지 자료 종류를 지원하고, 원격지 전송이 가능하다.
전송 객체는 Transferable 인터페이스를 구현해야 하는데 이 인터페이스는 세 가지 메소드를 선언하고 있다.

(1) public DataFlavor[] getTransferDataFlavors(); 지원하는 자료 종류를 배열로 반환한다.
(2) public boolean isDataFlavorSupported(DataFlavor flavor); 인자로 주어진 자료 종류를 지원하는지 여부를 알려준다.
(3) public Object getTransferData(DataFlavor flavor) throws UnsupportedFlavorException, IOException; 인자로 주어진 자료 종류에 맞는 데이터를 반환한다.

<따라하기 시작 – 커스텀 전송 객체>
여기에서 만들 전송 객체는 드랙앤드롭을 통해 넘어온 전송 객체의 내용 중 지원하는 자료 종류만 별도로 복사하여 저장하는 저장 객체이다. 드롭에서 넘겨진 전송 객체는 드롭 메소드가 종료하면 더 이상 사용할 수 없으므로, 이후 사용하려면 그 내용을 저장해둬야 한다. 아울러, 별도의 자료 종류를 하나 지정하여 지역 전송과 원격지 전송에서 모두 사용 가능하도록 선언하는 방법을 다룬다.

1. 먼저 클래스를 선언하고 멤버 필드를 정의한다.
Transferable 인터페이스를 구현하도록 선언하고, 멤버 필드로 지원하는 자료 종류를 저장할 ArrayList 객체를 선언한다.
그리고 해당하는 실제 데이터가 들어갈 필드를 각각 선언한다.
이 전송 객체는 문자열과 이미지, 파일 목록을 지원한다.
클래스 필드인 octetFlavor 는 서로 다른 자바 가상 머신 간의 원격지 전송일 경우에 사용할 자료 종류로, 바이트 배열 스트림을 나타낸다.

```
package yoonforh.dnd;

import java.awt.Image;
import java.awt.datatransfer.*;
import java.util.*;
import java.io.*;
```

```
/**
 * 전송 객체를 복제하는 전송 객체
 */

public class CopyTransferable implements Transferable {
  /**
   * 원격지 자바 가상 머신에서 호출될 경우 사용할 자료 종류
   * 이 자료 종류를 사용할 경우 내용이 되는 자바 객체는 직렬화가능하여야 한다.
   */
  public static DataFlavor octetFlavor = null;

  static { // static 초기화 블록
    try {
      octetFlavor = new DataFlavor("application/octet-stream");
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    }
  }

  /** 자료 종류 목록 */
  ArrayList flavorList = new ArrayList();

  // 전송 가능 자료들
  String string;
  Image image;
  List fileList;
  Object object;
}
```

2. 생성자를 구현한다. 생성자는 먼저 기본 자료 종류를 확인하기 위해 findDefaultFlavors() 메소드를 호출하고, 지원하는 자료 종류가 없을 경우, octetFlavor 자료 종류를 사용하여 드롭 객체를 스트림으로 전송 받도록 findOctetFlavor() 메소드를 호출한다.

```
  /**
```

```
   * 전송 객체를 복사하는 생성자
   */
  public         CopyTransferable(Transferable         t)         throws
UnsupportedFlavorException {
    // 먼저 지원하는 자료 종류를 찾는다.
    findDefaultFlavors(t);

    // 지원하는 자료 종류가 없으면 octetFlavor를 검사한다.
    if (flavorList.size() == 0) {
           // 스트림으로부터 실제 데이터를 전송 받아 전송 종류를 등록한다.
      try {
        findOctetFlavor(t);
      } catch (UnsupportedFlavorException e) {
        System.err.println("unsupported    flavor    error    :    "    +
e.getMessage());
      } catch (IOException e) {
        System.err.println("io error - " + e.getMessage());
      }
    }

    if (flavorList.size() == 0) {
      throw                                                          new
UnsupportedFlavorException(t.getTransferDataFlavors()[0]);
    }

    // 자료 종류의 원격지 유형을 추가한다.
    flavorList.add(octetFlavor);
    }
```

3. 기본 자료 종류를 확인하는 findDefaultFlavors() 메소드는 다음과 같다.
먼저 복사될 전송 객체가 지원하는 자료 종류를 살펴본 후, 지원하는 자료 종류인 문자열과 이미지, 파일 목록이 포함되어 있으면 이를 flavorList 멤버 필드에 추가한다. 그리고 각 실제 데이터를 읽어들여 각 필드에 저장한다.

```
/**
   * 전송 객체의 자료 종류 중 지원하는 자료 종류를 등록
```

```
   */
  private void findDefaultFlavors(Transferable t) {
    DataFlavor[] flavors = t.getTransferDataFlavors();

    for (int i = 0; i < flavors.length; i++) {
      try {
        if (flavors[i].equals(DataFlavor.stringFlavor)) {
          // 문자열 자료 종류 처리
          StringBuffer buffer = new StringBuffer();
          BufferedReader reader = new
BufferedReader(flavors[i].getReaderForText(t));
          try {
            int ch = -1;
            while ((ch = reader.read()) >= 0) {
              buffer.append((char) ch);
            }
          } finally {
            reader.close();
          }

          // 문자열 데이터 저장하고 자료 종류 목록에 추가
          string = buffer.toString();
          flavorList.add(DataFlavor.stringFlavor);
        } else if (flavors[i].equals(DataFlavor.imageFlavor)) {
          // 이미지 데이터 저장하고 자료 종류 목록에 추가
          image = (Image) t.getTransferData(flavors[i]);
          flavorList.add(DataFlavor.imageFlavor);
        } else if (flavors[i].equals(DataFlavor.javaFileListFlavor)) {
          // 파일 목록 데이터 저장하고 자료 종류 목록에 추가
          fileList = (List) t.getTransferData(flavors[i]);
          flavorList.add(DataFlavor.javaFileListFlavor);
        } else {
          // 지원하지 않는 자료 종류는 무시한다.
          System.out.println("unknown flavor - " + flavors[i]);
        }
      } catch (UnsupportedFlavorException e) {
```

```
      System.err.println("unsupported flavor error : " + flavors[i]);
    } catch (IOException e) {
      System.err.println("io error - " + e.getMessage());
    }
  }
  }
```

4. 지원하는 기본 자료 종류에 해당하는 자료 종류가 없을 경우 octetFlavor 를 지원하는지 검사한다.

   octetFlavor 자료 종류의 경우에는 실제 자료 객체를 드랙 원천으로부터 스트림 방식으로 넘겨 받는다. 여기에서는 구현을 간단하게 하기 위해 객체 직렬화를 사용하였다.

   객체 직렬화는 책의 뒷부분에서 다룬다.

   이 메소드는 스트림 방식으로 객체를 내보내고 다시 스트림으로부터 객체를 생성한다.

```
 /**
  * 원격지 전송(다른 자바 가상 머신 사이)에서도 사용 가능한 스트림 방식의 자료 종류
  * 드랙 원천이 CopyTransferable의 경우만 원격지 전송을 지원
  */
  private void findOctetFlavor(Transferable t)
      throws IOException, UnsupportedFlavorException {
   // 먼저 스트림으로부터 데이터 객체를 전송받아 지역 객체로 변환해서 가지고 있어야 한다.

   Object tobject = null; // 전송되어 올 객체
   if (t.isDataFlavorSupported(octetFlavor)) {
     Object data = null;
     data = t.getTransferData(octetFlavor);

     if (data instanceof InputStream) {
       ObjectInputStream in = null;
       try {
         in = new ObjectInputStream((InputStream) data);
         tobject = in.readObject();
       } catch (ClassNotFoundException e) {
```

```
        UnsupportedFlavorException ufe
          =                                                    new
UnsupportedFlavorException(t.getTransferDataFlavors()[0]);
        ufe.initCause(e);
       } finally {
        if (in != null) {
          in.close();
        }
       }
     }
   }

   if (tobject == null) {
     return;
   }

   // 지원하는 자료 종류의 자료형이면 해당 자료 종류로
   // 등록하고, 그렇지 않으면 octetFlavor로 등록한다.
   if (tobject instanceof String) {
     string = (String) tobject;
     flavorList.add(DataFlavor.stringFlavor);
   } else if (tobject instanceof Image) {
     image = (Image) tobject;
     flavorList.add(DataFlavor.imageFlavor);
   } else if (tobject instanceof List) {
     fileList = (List) tobject;
     flavorList.add(DataFlavor.javaFileListFlavor);
   } else {
     object = tobject;
     flavorList.add(octetFlavor);
   }
   }
```

5. 각 인터페이스 메소드를 구현한다.

getTransferDataFlavors() 메소드는 지원하는 자료 종류를 배열로 반환해야 하므로 flavorList 멤버 필드를 배열로 변환하여 반환한다.

isDataFlavorSupported() 메소드는 flavorList 멤버 필드가 해당 자료 종류를 포함하는지 여부를 알려준다.

```
  // 컬렉션의 배열 자료형을 지정하기 위한
  // 원소 없는 배열
private final static DataFlavor[] ZERO_FLAVORS = new DataFlavor[0];
/**
 * 지원하는 자료 종류
 */
public DataFlavor[] getTransferDataFlavors() {
  return (DataFlavor[]) flavorList.toArray(ZERO_FLAVORS);
}

/**
 * 자료 종류를 지원하는지 여부
 */
public boolean isDataFlavorSupported(DataFlavor flavor) {
  return flavorList.contains(flavor);
}
```

6. 마지막으로 getTransferData() 인터페이스 메소드를 구현한다. 이 메소드는 드랙앤드롭에서 드롭이 발생할 때 드랙 원천에서 지정한 전송 객체에 대해 호출될 것이다.

실제 데이터를 넘겨주면 되므로, 각 자료 종류에 따라 데이터를 반환한다. 원격지 전송의 경우인 octetFlavor 의 경우는 해당 데이터를 객체 직렬화를 통해 바이트 배열로 저장하고 바이트 배열 입력 스트림을 만들어 반환한다.

```
/**
 * 자료 종류에 따른 실제 데이터
 */
public Object getTransferData(DataFlavor flavor)
     throws UnsupportedFlavorException, IOException {
  if (flavor.equals(DataFlavor.stringFlavor)) {
    return string;
  } else if (flavor.equals(DataFlavor.imageFlavor)) {
    return image;
```

```
    } else if (flavor.equals(DataFlavor.javaFileListFlavor)) {
      return fileList;
    } else if (flavor.equals(octetFlavor)) {
      // 스트림으로 자바 객체를 넘겨준다.
      ByteArrayInputStream in = null;
      ByteArrayOutputStream out = null;
      Object tobject = this.object; // 보낼 자료 (임의의 객체)

      try {
        out = new ByteArrayOutputStream();
        new ObjectOutputStream(out).writeObject(tobject);
        in = new ByteArrayInputStream(out.toByteArray());
        out.close();
      } catch (IOException e) {
        throw e;
      } finally {
        if (out != null) {
          out.close();
        }
      }
           return in;
    }

    throw new UnsupportedFlavorException(flavor);
  }
```

7. 컴파일과 테스트는 드랙앤드롭 예제에서 함께 하도록 한다. 커스텀 자료 종류인 octetFlavor 는 일반적인 자바 객체의 전송에 사용될 수 있으며, 지역 객체에 대한 참조를 사용하지 않고 객체 직렬화 스트림을 사용하기 때문에 서로 다른 자바 가상 머신에서 실행된 프로그램 간의 드랙앤드롭을 지원할 수 있다. 일반적으로 자바 객체의 드랙앤드롭에 사용되는 자료 종류는 이와 같은 방식으로 구현된다.

<따라하기 끝 – 커스텀 전송 객체>

17.2.3. AWT 컴포넌트에서의 드랙앤드롭

AWT 컴포넌트를 사용하여 드랙앤드롭을 구현해보자.

자바 가상 머신이 서로 다른 두 개의 자바 프로그램 사이의 드랙앤드롭과 다른 윈도우 응용 프로그램과의 드랙앤드롭을 모두 지원하도록 구현한다.

<따라하기 시작 – AWT 드랙앤드롭 프로그램>
이 프로그램은 전송 객체를 내부적으로 가지고 있으면서 그 내용을 보여주는 캔버스 컴포넌트 두 개로 구성된다.
보여줄 수 있는 전송 객체의 자료 종류는 문자열, 이미지, 그리고 파일 목록이다.
또, 각 캔버스는 동시에 드랙 원천이자 드롭 대상으로 동작한다.

1. 전송 객체의 내용을 그리는 클래스를 정의하자. 각 캔버스는 paint() 메소드에서 이 클래스의 paintTransferable() 메소드를 사용하여 전송 객체를 그려서 보여줄 것이다.

문자열과 이미지, 파일 목록을 각각 지원하며, 파일 목록의 경우 문자열로 보여준다.

```
package yoonforh.dnd;

import java.awt.*;
import java.awt.datatransfer.*;
import java.util.*;
import java.util.List; // java.awt.List와 구분하기 위해 명시적으로 임포트
import java.io.*;

/**
 * 전송 객체의 내용을 컴포넌트에 그린다.
 */

public class TransferablePainter {
  Component c = null;

  public TransferablePainter(Component c) {
    this.c = c;
  }

  /**
   * 전송 객체의 지원 데이터 종류를 검사하여 칠한다.
   */
```

```
  public void paintTransferable(Graphics2D g2, Transferable t)
       throws UnsupportedFlavorException {
    if (t == null) {
      return;
    }

    // 지원하는 데이터 종류를 모두 구하여 하나씩 출력한다.
    DataFlavor[] flavors = t.getTransferDataFlavors();
    for (int i = 0; i < flavors.length; i++) {
      try {
        if (flavors[i].isFlavorTextType()) {
          StringBuffer buffer = new StringBuffer();
          BufferedReader reader = new
BufferedReader(flavors[i].getReaderForText(t));
          try {
            int ch = -1;
            while ((ch = reader.read()) >= 0) {
              buffer.append((char) ch);
            }
          } finally {
            reader.close();
          }

          paintString(g2, buffer.toString());
          return;
        } else if (flavors[i].equals(DataFlavor.imageFlavor)) {
          Image img = (Image) t.getTransferData(flavors[i]);
          paintImage(g2, img);
          return;
        } else if (flavors[i].isFlavorJavaFileListType()) {
          List list = (List) t.getTransferData(flavors[i]);
          paintFileList(g2, list);
          return;
        } else {
          System.out.println("unknown flavor. data is "
                   + t.getTransferData(flavors[i]));
```

```
      }
    } catch (UnsupportedFlavorException e) {
      System.err.println("unsupported flavor error : " + flavors[i]);
    } catch (IOException e) {
      System.err.println("io error - " + e.getMessage());
    }
  }

  throw new UnsupportedFlavorException(flavors[0]);
}

/**
 * 문자열을 칠한다.
 */
void paintString(Graphics2D g2, String string) {
  Font font = new Font("Helvetica", Font.PLAIN, 12);
  FontMetrics fm = g2.getFontMetrics(font);
  g2.setFont(font);
  g2.translate(10, 10 + fm.getAscent());

  int from = -1,
    to = 0;
  while (true) {
    to = string.indexOf('\n', ++from);
    if (to >= from) {
      g2.drawString(string.substring(from, to), 0, 0);
      g2.translate(0, fm.getHeight());
    } else {
      g2.drawString(string.substring(from), 0, 0);
      break;
    }
    from = to;
  }
}

/**
```

```
   * 이미지를 칠한다.
   */
  void paintImage(Graphics2D g2, Image img) {
    g2.drawImage(img, 0, 0, c);
  }

  /**
   * 파일 목록을 표현한다.
   */
  void paintFileList(Graphics2D g2, List list) {
    StringBuffer buffer = new StringBuffer("File List ---\n\n");
    Iterator it = list.iterator();
    while (it.hasNext()) {
      File file = (File) it.next();
      buffer.append(file.isDirectory()
             ? "Directory : "
             : "File : ");
      try {
        buffer.append(file.getCanonicalPath()).append('\n');
      } catch (IOException e) {
        System.err.println("io error : " + e.getMessage());
      }
    }

    paintString(g2, buffer.toString());
  }
}
```

2. 드랙 원천과 드롭 대상을 모두 지원하도록 캔버스 클래스를 구현한다.
먼저 드랙 원천을 지원하도록 구현해보자. 드랙 원천 객체를 생성하여 드랙 동작 인식 리스너를 등록하면 드랙 원천을 지원할 수 있다.

```
package yoonforh.dnd;

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

```
import java.awt.dnd.*; // 드래앤드롭 지원
import java.awt.datatransfer.*;
import java.io.IOException;

/**
 * 드래앤드롭을 지원하는 캔버스
 */

public class DragNDropCanvas extends Canvas {
  /**
   * 드롭된 전송 객체
   */
  Transferable xfer = null;
  Point pt = null; // 드롭이 일어난 위치
  TransferablePainter painter = null;
  // 드롭이 일어난 컴포넌트
  Component dropComponent = null;

  /**
   * 생성자. 드랙 원천과 드롭 대상을 지정한다.
   */
  public DragNDropCanvas() {
    // 드랙 원천 객체 생성
    DragSource ds = new DragSource();
    ds.createDefaultDragGestureRecognizer(
      this, // 드랙이 시작하는 컴포넌트
      DnDConstants.ACTION_COPY_OR_MOVE, // 드랙 동작 유형
      new MyDragGestureListener() // 드랙 동작 리스너
      );

    // 이 프로그램에서는 하는 일이 없지만
    // 드랙 원천의 드랙 움직임 이벤트를 처리하려면 이 리스너를 등록한다.
    ds.addDragSourceMotionListener(new MyDragSourceMotionListener());

    // 드롭 대상 지정 코드를 여기에 넣을 것이다.
```

```
  // 전송 객체 내용을 칠하는 객체
  painter = new TransferablePainter(this);
}

/**
 * 캔버스 내용을 칠한다.
 */
public void paint(Graphics g) {
  if (xfer == null) {
    return;
  }

  // 그래픽 객체를 복사한다.
  Graphics2D g2 = (Graphics2D) g.create();
  try {
    if (pt != null) {
      g2.translate(pt.x, pt.y);
    }

    painter.paintTransferable(g2, xfer);
  } catch (UnsupportedFlavorException e) {
    e.printStackTrace();
    setTransferable(null, null);
  } finally {
    // 복사한 그래픽 장치 리소스를 해지한다.
    g2.dispose();
  }
}

/**
 * 캔버스에 보여줄 전송 객체를 변경
 */
void setTransferable(Transferable t, Point pt) {
  this.xfer = t;
  this.pt = pt;
  repaint();
```

```
  }

/**
 * 액션에 해당하는 커서를 반환한다.
 */
Cursor getActionCursor(int action, boolean acceptable) {
  switch (action) {
  case DnDConstants.ACTION_COPY :
    if (acceptable) {
      return DragSource.DefaultCopyDrop;
    } else {
      return DragSource.DefaultCopyNoDrop;
    }

  case DnDConstants.ACTION_MOVE :
  case DnDConstants.ACTION_COPY_OR_MOVE :
    if (acceptable) {
      return DragSource.DefaultMoveDrop;
    } else {
      return DragSource.DefaultMoveNoDrop;
    }

  case DnDConstants.ACTION_LINK :
  case DnDConstants.ACTION_NONE :
    if (acceptable) {
      return DragSource.DefaultLinkDrop;
    } else {
      return DragSource.DefaultLinkNoDrop;
    }
  }

  return null;
}

/**
 * 디버그 용도로 사용할 수 있는 액션의 문자열 표시
```

```
   */
  String getActionString(int action) {
    switch (action) {
    case DnDConstants.ACTION_COPY :
      return "COPY";

    case DnDConstants.ACTION_MOVE :
      return "MOVE";

    case DnDConstants.ACTION_COPY_OR_MOVE :
      return "COPY_OR_MOVE";

    case DnDConstants.ACTION_LINK :
      return "LINK";

    case DnDConstants.ACTION_NONE :
      return "NONE";
    }

    return "UNKNOWN_ACTION";
  }

  /**
      * DragSourceMotionListener 인터페이스 구현한
      * 내부 클래스. 이 프로그램에서는 하는 일이 없다.
   */
  class MyDragSourceMotionListener implements DragSourceMotionListener
{
    public void dragMouseMoved(DragSourceDragEvent dsde) {  }
  }

  }
```

3. 드랙 동작 인식 리스너에서 드랙 동작이 감지되면 startDrag() 메소드를 호출해 줘야 실제 드랙이 시작된다.

각 드랙앤드롭의 리스너들은 이해를 돕기 위해 모두 별도의 내부 클래스로 구현하

였다.

내부에 전송 객체를 가지고 있을 때에만 드랙을 시작하게 하고, 드랙이 시작될 때의 커서는 드랙 동작에 맞는 커서를 지정한다. 드랙이 될 전송 객체는 현재 캔버스가 xfer 필드로 가지고 있는 전송 객체이며, 또한 드랙 원천 리스너를 등록한다.

```
/**
 * DragGestureListener 인터페이스 구현
 */
class MyDragGestureListener implements DragGestureListener {
  public void dragGestureRecognized(DragGestureEvent dge) {
    // 드랙 시작
    if (xfer != null) {
      dge.startDrag(getActionCursor(dge.getDragAction(), true), xfer,
            new MyDragSourceListener());
    }
  }
}
```

4. 드랙 원천 리스너를 구현함으로써 드랙 동작의 지원은 모두 구현된다.

드롭 동작이 변경되는 것을 감지할 경우 해당하는 드롭 동작이 허용되는지 여부를 자료 종류와 드롭 동작을 참고로 판단하여 그에 맞는 커서를 표현한다. 드롭이 완료되고 나면 dragDropEnd() 메소드가 호출되는데, 이 메소드에서는 드롭의 성공 여부와 드롭 동작을 판단하여 드롭의 이동 동작이 성공으로 끝나고, 드롭 컴포넌트가 자신이 아닌 경우에는 캔버스의 전송 객체를 null 로 하여 다른 컴포넌트로의 이동을 표시하였다.

```
/**
 * DragSourceListener 인터페이스 구현
 */
class MyDragSourceListener implements DragSourceListener {
  public void dragEnter(DragSourceDragEvent dsde) {  }

  public void dragOver(DragSourceDragEvent dsde) {  }

  public void dropActionChanged(DragSourceDragEvent dsde) {
```

```
      // 사용자 액션대로 커서 표시
      dsde.getDragSourceContext().setCursor(
        getActionCursor(dsde.getUserAction(),
              isDropActionAllowed(dsde, dsde.getUserAction())));
    }

    public void dragExit(DragSourceEvent dse) {  }

    public void dragDropEnd(DragSourceDropEvent dsde) {
      int action = dsde.getDropAction();

      // 드롭이 성공하고, 그 액션이 이동이었을 경우
      // 드롭 목적 컴포넌트가 이 컴포넌트 자신이 아니면
      // 데이터를 삭제한다.
      if (dsde.getDropSuccess()
        && dsde.getDropAction() == DnDConstants.ACTION_MOVE
        && (dropComponent == null
          || dropComponent != DragNDropCanvas.this)) {
        setTransferable(null, null);
      }
    }

    /**
     * DragSourceDragEvent에서 액션을 허용할 것인지를 결정한다.
     */
    private boolean isDropActionAllowed(DragSourceDragEvent dsde, int
action) {
      // 파일 목록일 경우에는 복사만 허용 (실수로 파일이 지워지지 않도록... ㅠ_ㅠ)
      // 이미지와 문자열일 경우에는 복사와 이동을 모두 허용
      // 나머지는 거부
      Transferable t = dsde.getDragSourceContext().getTransferable();
      if (t.isDataFlavorSupported(DataFlavor.javaFileListFlavor)) {
        return action == DnDConstants.ACTION_COPY;
      } else if (t.isDataFlavorSupported(DataFlavor.stringFlavor)
          || t.isDataFlavorSupported(DataFlavor.imageFlavor)) {
        return true;
```

```
    } else {
      return false;
    }
  }

}
```

5. 드롭 동작 지원은 드롭 대상 객체를 생성하는 것에서부터 시작한다. 생성자에 드롭 대상 객체를 생성하는 코드를 추가한다.

```
// 드롭할 대상 지정
  try {
    DropTarget dt
      = new DropTarget(this, // 드롭을 지원할 컴포넌트
              DnDConstants.ACTION_COPY_OR_MOVE,
              new MyDropTargetListener());
          // 명시적으로 호출하지 않아도 DropTarget의
          // 생성자에서 컴포넌트가 지정된다.
    setDropTarget(dt);
  } catch (UnsupportedOperationException e) {
    e.printStackTrace();
  }
```

6. 드롭 대상 객체의 생성자에서 지정한 드롭 대상 리스너를 구현함으로써 드롭 동작에 대한 지원이 완성된다.

드랙 중인 마우스 커서가 드롭 대상 컴포넌트 위로 오면 dragEnter() 메소드가 호출되는데 여기에서 드롭을 허용할 것인지를 결정하여 허용하거나 거부한다. 또 dragEnter()와 dragExit() 메소드에서는 컴포넌트를 드랙 커서가 들어오거나 벗어날 때 드롭 대상 컴포넌트가 바뀌었음을 반영한다.

dropActionChanged() 메소드에서는 사용자가 드롭 동작을 변경할 때 동작에 따라 허용하거나 거부한다.

드롭 대상 리스너의 가장 중요한 메소드는 drop() 메소드로 실제 이 메소드 안에서 드롭 동작을 처리해야 한다. 여기에서 이벤트 객체에 대해 acceptDrop() 혹은 rejectDrop() 메소드를 호출해야 하며, 또 acceptDrop()을 호출한 경우에는 반드시 dropComplete() 메소드를 호출하여 드롭의 종료를 알려줘야만 한다.

```
/**
 * DropTargetListener 인터페이스 구현
 */
class MyDropTargetListener implements DropTargetListener {
  public void dragEnter(DropTargetDragEvent dtde) {
    dropComponent = dtde.getDropTargetContext().getComponent();

     // 액션을 검사하여 허용
    checkDropTargetDropAction(dtde);
  }

  public void dragOver(DropTargetDragEvent dtde) {  }

  public void dropActionChanged(DropTargetDragEvent dtde) {
     // 액션을 검사하여 허용
    checkDropTargetDropAction(dtde);
  }

  public void dragExit(DropTargetEvent dte) {
    // dropComponent가 null이면 드롭 대상이 다른 컴포넌트임을 뜻한다.
    Component component = dte.getDropTargetContext().getComponent();
    if (dropComponent == component) {
      dropComponent = null;
    }
  }

  public void drop(DropTargetDropEvent dtde) {
    // 파일 목록일 경우에는 복사만 허용 (실수로 파일이 지워지지 않도록... ㅠ_ㅠ)
    // 이미지와 문자열일 경우에는 복사와 이동을 모두 허용
    // 나머지는 거부
    if (dtde.isDataFlavorSupported(DataFlavor.javaFileListFlavor)) {
      if (dtde.getDropAction() == DnDConstants.ACTION_COPY) {
        dtde.acceptDrop(DnDConstants.ACTION_COPY);
      } else {
        dtde.rejectDrop();
        return;
```

```
    }
  } else if (dtde.isDataFlavorSupported(DataFlavor.stringFlavor)
      || dtde.isDataFlavorSupported(DataFlavor.imageFlavor)) {
    dtde.acceptDrop(dtde.getDropAction());
  } else {
    dtde.rejectDrop();
    return;
  }

  Graphics2D g2 = (Graphics2D) DragNDropCanvas.this.getGraphics();
  try {
    // 드랙앤드롭의 전송 객체는 드롭할 당시에만 유효하므로
    // 이 프로그램의 경우에는 복사해서 보관한다.
    Transferable t = new CopyTransferable(dtde.getTransferable());
    Point p = dtde.getLocation();
    g2.translate(p.x, p.y);
    painter.paintTransferable(g2, t); // 칠할 수 있는지 테스트함
    dtde.dropComplete(true); // 드롭 성공
    setTransferable(t, p);
  } catch (UnsupportedFlavorException e) {
    e.printStackTrace();
    dtde.dropComplete(false); // 드롭 실패
  } finally {
    g2.dispose();
  }
}

/**
 * DropTargetDragEvent에서 액션을 허용할 것인지를 결정한다.
 */
private void checkDropTargetDropAction(DropTargetDragEvent dtde) {
  // 파일 목록일 경우에는 복사만 허용 (실수로 파일이 지워지지 않도록... ㅠ_ㅠ)
  // 이미지와 문자열일 경우에는 복사와 이동을 모두 허용
  // 나머지는 거부
  if (dtde.isDataFlavorSupported(DataFlavor.javaFileListFlavor)) {
    if (dtde.getDropAction() == DnDConstants.ACTION_COPY) {
```

```
        dtde.acceptDrag(DnDConstants.ACTION_COPY);
        return;
      }
    } else if (dtde.isDataFlavorSupported(DataFlavor.stringFlavor)
        || dtde.isDataFlavorSupported(DataFlavor.imageFlavor)) {
      dtde.acceptDrag(dtde.getDropAction());
      return;
    }

    dtde.rejectDrag();
  }
 }
```

7. 드랙앤드롭을 지원하는 캔버스의 구현이 끝났으므로 간단한 테스트 프로그램을 작성하여 실행해보자.

이 프로그램은 프레임에 위에서 구현한 캔버스를 두 개 좌우로 배치하여 드랙앤드롭을 테스트할 수 있게 하였다.

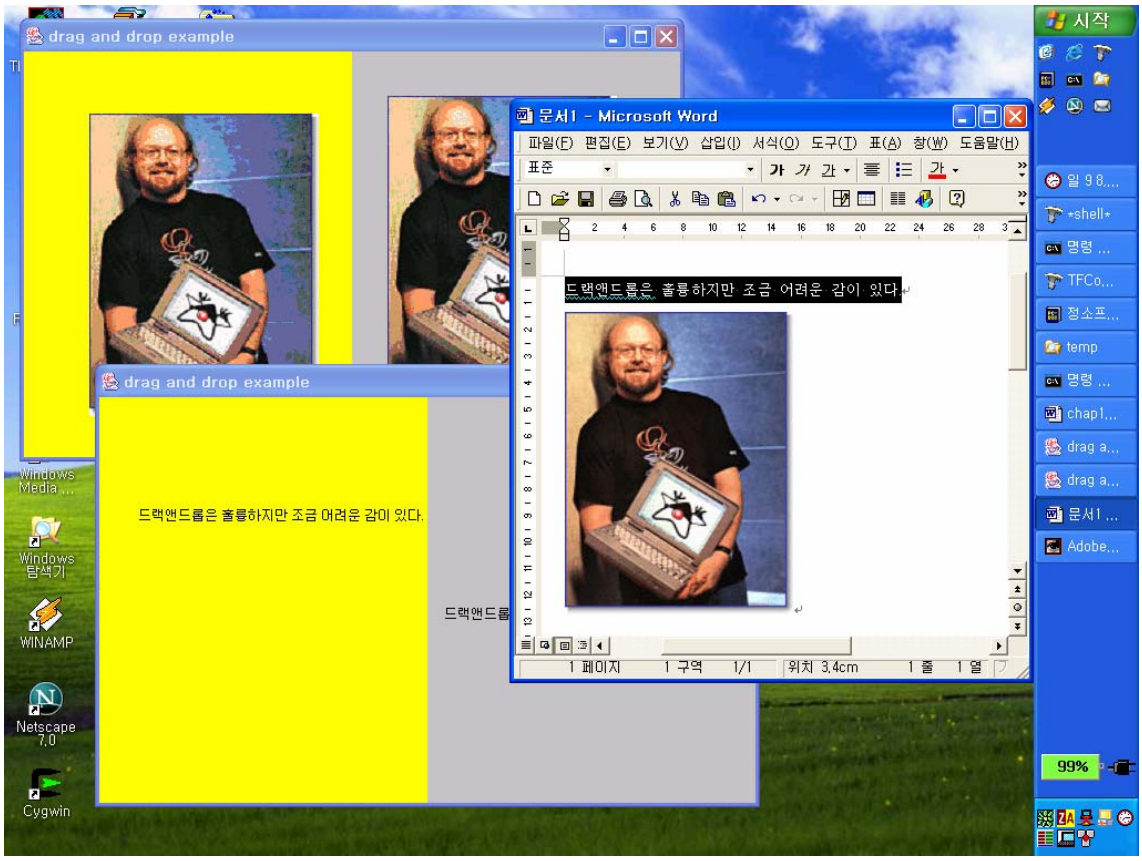
```
package yoonforh.dnd;

import java.awt.*;
import java.awt.event.*;

/**
 * 드랙앤드롭 예제 프로그램
 */

public class DragNDropEx extends Frame {
  public DragNDropEx(String title) {
    super(title);

    setLayout(new GridLayout(1, 2));
    // 두 개의 캔버스를 구분하기 위해 배경색을 서로 다르게 지정한다.
    DragNDropCanvas c1 = new DragNDropCanvas();
    c1.setBackground(Color.yellow);
    DragNDropCanvas c2 = new DragNDropCanvas();
```

```
    c2.setBackground(Color.lightGray);
    add(c1);
    add(c2);

    // 윈도우 종료 시 프로그램 종료하도록
    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent evt) {
          dispose();
          System.exit(0);
        }
      });
  }

  public static void main(String[] args) {
    Frame f = new DragNDropEx("drag and drop example");

    f.setBounds(10, 10, 600, 400);
    f.setVisible(true);
  }
}
```

8. 컴파일하여 실행해보자.

```
javac yoonforh\dnd\*.java <엔터>

java -cp . yoonforh.dnd.DragNDropEx <엔터>
```

실행한 상태에서는 이 프로그램은 전송 객체를 가지고 있지 않으므로 드랙을 시작할 수 없다.
다른 윈도우 프로그램이나 탐색기를 사용하여 텍스트, 이미지, 파일 목록을 드랙하여 이 프로그램 위로 드롭해보자.
또, 원격 전송이 동작하는지 확인하기 위해 이 프로그램을 두 개 띄워 텍스트와 파일 목록을 서로에게 드랙앤드롭해보자.
다음 그림은 마이크로소프트 워드를 사용하여 이미지와 텍스트를 드랙하여 이 프로그램 위로 드롭한 상태를 보여준다.



[그림 2 –마이크로소프트 워드로부터 이미지와 텍스트를 예제 프로그램으로 드랙앤드롭한 모습]

<따라하기 끝 – AWT 드랙앤드롭 프로그램>

17.2.4. 스윙 드랙앤드롭

스윙에서도 앞에서 다룬 드랙앤드롭 기능을 동일하게 사용할 수도 있지만, 스윙 컴포넌트 프레임웍에서는 드랙앤드롭 기능을 전담하는 전송 처리기 TransferHandler 객체를 사용하여 좀더 쉽게 드랙앤드롭 기능을 지원할 수 있게 해준다.

이 전송 처리기 객체는 각 스윙 컴포넌트별로 사용할 수 있으며 스윙 컴포넌트들의 부모 클래스인 JComponent 클래스에는 전송 처리기 객체를 지정하거나 현재의 전송 처리기 객체를 가져오는 메소드가 정의되어 있다.

```
public void setTransferHandler(TransferHandler newHandler);
public TransferHandler getTransferHandler();
```

전송 처리기 객체는 드랙앤드롭 뿐만 아니라 클립보드 사용도 쉽게 해준다. 대부분의 스윙

컴포넌트들은 전송 처리기 객체를 사용하여 드랙앤드롭과 클립보드 기능의 일부 혹은 전체를 지원하고 있으며, 커스텀 스윙 컴포넌트를 구현할 때에 전송 처리기 객체를 사용하여 쉽게 드랙앤드롭과 클립보드 기능을 추가할 수 있다.

먼저 전송 처리기 객체를 사용하여 드랙 기능을 지원하는 방법을 알아보자.
내부적으로 드랙 기능을 내장한 일부 컴포넌트들은 드랙 기능을 활성화시키는 별도의 메소드를 제공한다. JColorChooser, JFileChooser, JList, JTable, JTree, JTextComponent 등의 컴포넌트에서 다음 메소드가 제공된다.

```
public void setDragEnabled(boolean b);
public boolean getDragEnabled();
```

드랙 기능이 내장되어 있지 않은 커스텀 컴포넌트에서 드랙 기능을 지원하려면 다음과 같이 한다.

(1) 먼저 TransferHandler 객체를 지정한다.
TransferHandler 클래스는 자바 빈즈 속성을 전송 객체로 사용할 경우 생성자에서 지정하고, 그렇지 않을 경우에는 클래스를 상속하여 createTransferable() 메소드를 오버라이드하여 전송 객체를 지정하여야 한다. 자바 빈즈에 대해서는 이 책의 뒷 부분에서 자세하게 다룰 것이다. 이외에도 드랙 동작 중 지원하는 동작을 지정하려면 getSourceActions() 메소드를 오버라이드한다. 기본으로 복사 동작만 지원한다.

```
component.setTransferHandler(new TransferHandler() {
  protected Transferable createTransferable(JComponent c) {
    // 원하는 전송 객체를 반환하도록 구현한다.
  }
});
```

(2) 드랙 동작을 인식하여 exportAsDrag() 메소드를 호출한다.
마우스가 눌린 동작을 드랙 동작의 시작으로 인식하여 처리하는 방법이 있다.

```
component.addMouseListener(new MouseAdapter() {
  public void mousePressed(MouseEvent e) {
      JComponent c = (JComponent) e.getSource();
      TransferHandler th = c.getTransferHandler();
      th.exportAsDrag(c, e, TransferHandler.COPY);
```

```
  }
});
```

(3) 드롭이 끝났을 때의 처리를 위해 TransferHandler 의 exportDone() 메소드를 오버라이드한다.

드랙앤드롭의 경우처럼 이동 동작이었으면 드랙 원천의 데이터를 지우는 등의 처리가 필요하다.

```
protected void exportDone(JComponent source,
    Transferable data, int action) {
  // 오버라이드하여 드롭 동작 완료 이후를 처리한다.
}
```

전송 처리기 객체를 사용하여 드롭 기능을 지원하는 것은 더욱 간단하다.
드롭 기능을 내장하고 있는 스윙 컴포넌트에서는 전송 처리기 객체를 사용하는 DropTarget 이 정의되어 간단한 드롭 기능 지원이 되므로, 별도의 처리가 필요없다.
커스텀 스윙 컴포넌트에서도 JComponent 에 이미 간단한 기본 DropTarget 이 정의되어 있으므로 TransferHandler 객체를 오버라이드하여 필요한 드랙 기능과 드롭 기능이 동작하도록 하면 된다. 좀더 세밀한 드롭 동작에 대한 제어가 필요한 경우 앞의 AWT 예제에서처럼 DropTarget 을 별도로 생성하여 드롭 대상 리스너를 구현하여야 한다.
TransferHandler 클래스의 메소드들 중 드롭과 관련 있는 것은 다음 두 메소드이다.

(1) public boolean canImport(JComponent comp, DataFlavor[] transferFlavors); 자료 종류를 임포트할 수 있는지 즉, 드롭이나 클립보드로부터 붙여넣기가 가능한지 여부를 반환한다.
(2) public boolean importData(JComponent comp, Transferable t); 전송 객체를 임포트한다. 드롭이나 클립보드로부터 붙여넣기를 실행할 때 호출된다.

<따라하기 시작 – 스윙 드랙앤드롭 프로그램>
앞에서 AWT 컴포넌트로 구현한 프로그램을 스윙 컴포넌트를 사용하여 구현해보자. TransferHandler 를 사용하면 구현이 훨씬 간단해지고, 소스 양이 많이 줄지만, 세밀한 제어가 필요할 때에는 다시 앞의 DragSource 와 DropTarget 방식으로 돌아가야 한다.
여기에서는 TransferHandler 만을 사용하여 구현해본다.

1. JPanel 을 상속하여 드랙 원천과 드롭 대상 모두를 지원하는 컴포넌트를 설계해보자. 먼저 기본 골격부터 잡는다.

AWT 의 경우에 사용했던 TransferablePainter 클래스와 CopyTransferable 클래스를 여

기에서도 사용할 것이다.

생성자에서는 먼저 컴포넌트의 커스텀 전송 처리기를 등록하기 위해 setTransferHandler() 메소드를 호출한다.

그리고, 드랙의 시작을 감지하기 위해 마우스 리스너를 등록하였다. 마우스가 눌려지면 TransferHandler 객체의 exportAsDrag() 메소드를 호출하여 드랙이 시작하도록 한다.

패널의 내용을 칠하는 것은 AWT 의 버전과 크게 다르지 않으나, 여기에서는 드롭이 발생한 마우스 위치를 바로 구할 수가 없어 항상 원점에서 내용을 칠한다. 마우스 위치를 구하려면 AWT 버전의 경우처럼 DropTarget 을 사용하여 드롭 대상 리스너를 등록하면 된다.

```
package yoonforh.dnd;

import java.awt.*;
import java.awt.event.*;
import java.awt.dnd.*; // 드랙앤드롭
import java.awt.datatransfer.*; // 클립보드와 전송 객체
import javax.swing.*;
import java.io.IOException;

/**
 * 드랙앤드롭을 지원하는 커스텀 스윙 컴포넌트
 */

public class SwingDNDPanel extends JPanel {

  /**
   * 드롭된 전송 객체
   */
  Transferable xfer = null;
  TransferablePainter painter = null; // 전송 객체를 보여주는 클래스
  JComponent dropComponent = null; // 드롭 대상 컴포넌트

  /**
   * 생성자. 전송 처리기 객체를 지정하고 드랙과 드롭을 지원한다.
   */
```

```
public SwingDNDPanel() {
  // 전송 객체 내용을 칠하는 객체
  painter = new TransferablePainter(this);

  // 커스텀 전송 처리기를 등록한다.
  setTransferHandler(new MyTransferHandler());

  // 드랙 시작을 알기 위해 마우스 리스너를 사용한다.
  addMouseListener(new MouseAdapter() {
      /**
       * 마우스가 눌리면 드랙 시작으로 간주
       */
      public void mousePressed(MouseEvent e) {
        // 전송 객체가 비어 있으면 드랙을 시작하지 않는다.
        if (xfer == null) {
          return;
        }

                // 드랙이 시작되므로 드롭 대상 컴포넌트를
                // null로 초기화한다.
        dropComponent = null;
        TransferHandler th = SwingDNDPanel.this.getTransferHandler();
        th.exportAsDrag(SwingDNDPanel.this, e, TransferHandler.COPY);
      }
    });
}

/**
 * 패널 내용을 칠한다.
 */
public void paintComponent(Graphics g) {
  super.paintComponent(g);

  if (xfer == null) {
    return;
  }
```

```
    // 그래픽 객체를 복사한다.
    Graphics2D g2 = (Graphics2D) g.create();
    try {
      painter.paintTransferable(g2, xfer);
    } catch (UnsupportedFlavorException e) {
      e.printStackTrace();
      setTransferable(null);
    } finally {
      // 복사한 그래픽 장치 리소스를 해지한다.
      g2.dispose();
    }
  }

  /**
   * 패널에 보여줄 전송 객체를 변경
   */
  void setTransferable(Transferable t) {
    this.xfer = t;
    repaint();
  }

}
```

2. 커스텀 전송 처리기를 내부 클래스로 구현하여 드래과 드롭을 모두 지원한다. 먼저 드랙과 관련된 메소드들을 오버라이드하여 구현한다.

```
/**
   * 전송 처리기 클래스.
   */
  class MyTransferHandler extends TransferHandler {
    /**
     * 드랙할 전송 객체를 만든다.
     */
    protected Transferable createTransferable(JComponent c) {
      // 전송 객체는 패널의 멤버 필드인 xfer를 사용한다.
```

```
    return xfer;
  }

  /**
   * 지원하는 드랙 동작을 정의한다. 여기에서는 복사와 이동을 지원한다.
   */
  public int getSourceActions(JComponent c) {
    return DnDConstants.ACTION_COPY_OR_MOVE;
  }

  /**
   * 드롭이 완료된 후를 처리한다.
   */
  protected void exportDone(JComponent source,
              Transferable data, int action) {
          // 드롭 대상 컴포넌트가 null이거나 드랙 원천과
          // 다르면 다른 컴포넌트로 이동된 것이므로
          // 이 컴포넌트의 전송 객체를 지운다.
    if (action == DnDConstants.ACTION_MOVE
      && (dropComponent == null
        || dropComponent != source)) {
      setTransferable(null);
    }
  }
}
```

3. 전송 처리기의 드롭과 관련된 메소드를 오버라이드한다.

```
  /**
   * 자료 종류를 임포트할 수 있는지 즉, 드롭이나 클립보드로부터
   * 붙여넣기가 가능한지 여부를 반환한다.
   */
  public boolean canImport(JComponent comp, DataFlavor[] flavors) {
    for (int i = 0; i < flavors.length; i++) {
      // 파일 목록, 이미지, 문자열, octetFlavor만 허용
      if (flavors[i].equals(DataFlavor.javaFileListFlavor)
```

```
        || flavors[i].equals(DataFlavor.stringFlavor)
        || flavors[i].equals(DataFlavor.imageFlavor)
        || flavors[i].equals(CopyTransferable.octetFlavor)) {
        return true;
      }
    }

    return false;
  }

  /**
   * 전송 객체를 임포트한다. 드롭이나 클립보드로부터 붙여넣기를
   * 실행할 때 호출된다.
   */
  public boolean importData(JComponent comp, Transferable t) {
    //
    dropComponent = comp;
    Graphics2D g2 = (Graphics2D) SwingDNDPanel.this.getGraphics();
    try {
      // 드랙앤드롭의 전송 객체는 드롭할 당시에만 유효하므로
      // 이 프로그램의 경우에는 복사해서 보관한다.
      Transferable tcopy = new CopyTransferable(t);
      painter.paintTransferable(g2, tcopy); // 칠할 수 있는지 테스트함
      DataFlavor[] fs = tcopy.getTransferDataFlavors();
      setTransferable(tcopy);

      if  (tcopy.isDataFlavorSupported(DataFlavor.javaFileListFlavor))
{
        // 자료 종류가 파일 목록일 경우 이 프로그램을 테스트하다가
        // 실수로 이동이 되어 파일이 삭제되는 것을 막기 위해
        // 여기에서는 항상 false를 반환한다. ㅠ_ㅠ
        // 결과적으로는 파일이 복사만 되는 효과가 난다.
        return false;
      }
      return true;
    } catch (UnsupportedFlavorException e) {
```

```
        e.printStackTrace();
      } finally {
        g2.dispose();
      }

      return false;
    }
```

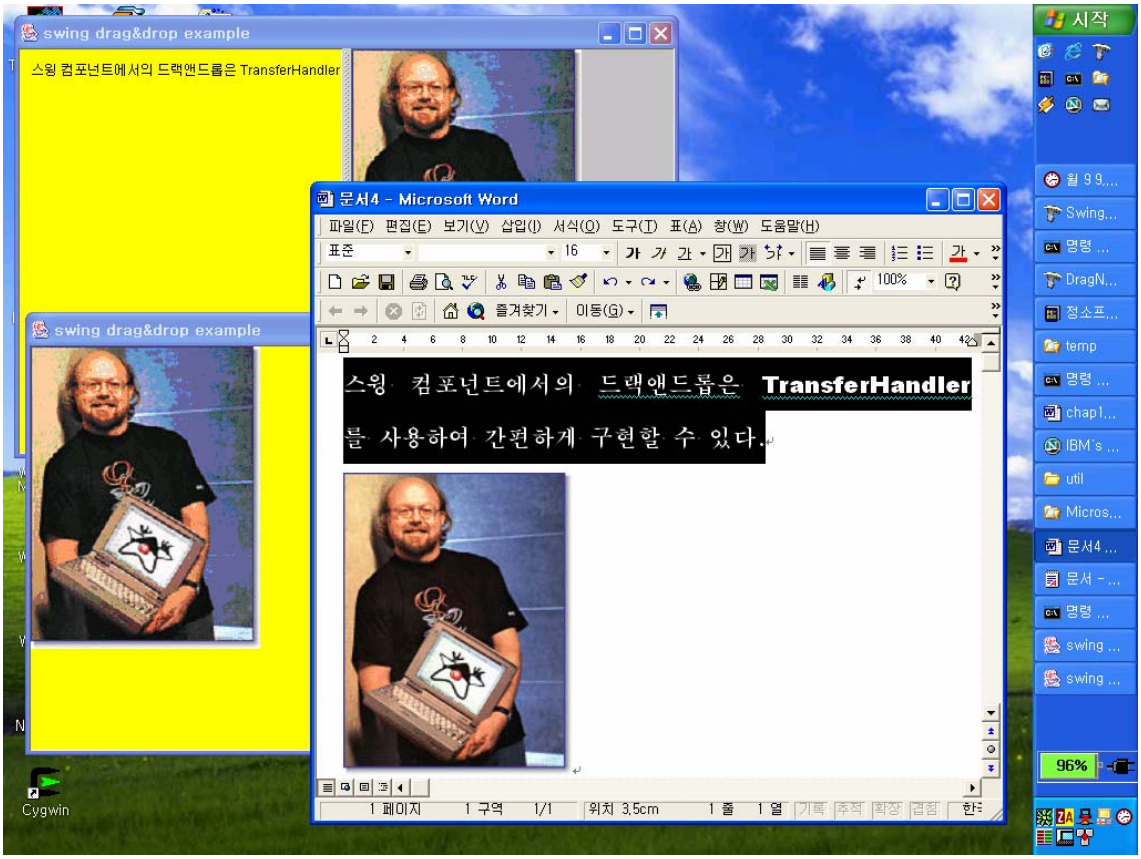
4. AWT 버전과 비교하여 드롭 마우스 위치를 쉽게 구할 수 없는 등 제한이 있지만 훨씬 간단해졌음을 볼 수 있다. 간단한 테스트 프로그램을 작성하여 이 패널 컴포넌트를 추가하자.

```
package yoonforh.dnd;

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

/**
 * 스윙 컴포넌트에서의 드랙앤드롭 예제 프로그램
 */

public class SwingDNDEx extends JFrame {
  public SwingDNDEx(String title) {
    super(title);

    SwingDNDPanel leftPanel = new SwingDNDPanel();
    leftPanel.setBackground(Color.yellow);
    SwingDNDPanel rightPanel = new SwingDNDPanel();
    rightPanel.setBackground(Color.lightGray);

    JSplitPane splitter = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                    true, /* 구분막대를 움직이는 동안에도 계속 렌더 */
                    leftPanel,
                    rightPanel);
     // 스플릿페인의 구분막대가 가운데 오도록 한다.
```

```
    splitter.setResizeWeight(0.5);

    // JFrame의 컨텐트 페인으로 스플릿페인을 사용
    setContentPane(splitter);

    // 윈도우 닫으면 종료
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }

  public static void main(String[] args) {
    JFrame f = new SwingDNDEx("swing drag&drop example");

    f.setBounds(10, 10, 600, 400);
    f.setVisible(true);
  }
}
```

5. 컴파일하여 실행해보자. AWT 버전에 사용했던 TransferablePainter 클래스와 CopyTransferable 클래스 등이 모두 클래스 경로에 있어야 한다.

```
javac yoonforh\dnd\*.java <엔터>

java -cp . yoonforh.dnd.SwingDNDEx <엔터>
```



[그림 3 – 스윙 드랙앤드롭 프로그램 실행 모습]

<따라하기 끝 – 스윙 드랙앤드롭 프로그램>

17.3. 실행 취소/재실행

편집 가능한 프로그램을 작성할 때 실행 취소와 재실행은 반드시 필요한 인터페이스 중 하나이다.

실행 취소와 재실행을 지원하기 위해서는 각 편집 행위마다 그 행위의 실행 취소 행위와 재실행 행위를 함께 정의해야 한다.

스윙 컴포넌트 프레임웍에서는 javax.swing.undo 패키지에 UndoableEdit 이라는 인터페이스를 사용하여 각 편집 행위의 실행 취소 및 재실행 행위 정보를 지정하게 하고, UndoManager 라는 관리자 클래스를 통해 등록하여 실행 취소와 재실행을 지원한다.

스윙에서 실행 취소와 재실행을 지원하기 위해서는 먼저 각 행위에 대한 UndoableEdit 을 구현한 클래스를 정의해야 하는데 AbstractUndoableEdit 클래스는 이 인터페이스를 단순 구현한 클래스로 이 클래스를 상속하여 필요한 메소드만 재정의하면 된다.

UndoManager 클래스는 각 실행 취소 정보를 저장하고, 실행 취소, 재실행 등의 기능을 지원한다. 다음은 UndoManager 클래스의 주요 메소드이다.

(1) public boolean addEdit(UndoableEdit anEdit); 실행 취소 정보를 추가한다.
(2) public void undo(); 실행을 취소한다.
(3) public void redo(); 재실행한다.
(4) public boolean canUndo(); 실행 취소 가능한 상태인지를 알려준다.
(5) public boolean canRedo(); 재실행 가능한 상태인지를 알려준다.
(6) public boolean canUndoOrRedo(); 실행 취소가 가능하거나 재실행이 가능하면 true 를 반환한다.
(7) public void discardAllEdits(); 등록된 실행 취소 정보들을 모두 삭제한다.

<따라하기 시작 – undo/redo 예제 프로그램>
숫자를 1 씩 증가시키고 감소시키는 동작에 대한 undo/redo 기능을 생각해보자. 1 을 증가시킨 동작에 대한 undo() 기능은 1 을 감소시키는 동작이 되고, redo() 기능은 다시 1 을 증가시키는 동작이 된다. 마찬가지로 1 을 감소시킨 동작에 대해서는 undo()와 redo()가 반대가 된다.
이 동작을 지원하는 UndoableEdit 인터페이스를 구현하여 undo 와 redo 를 지원하는 카운터 프로그램을 만들어보자.

1. 먼저 사용자 인터페이스를 구성한다. 스윙 컴포넌트를 사용한다.
현재 숫자를 보여줄 JLabel 컴포넌트와 증가, 감소, undo, redo, 실행 취소 정보 삭제 등의 액션을 사용하는 모두 다섯 개의 JButton 컴포넌트들로 구성된다.

```
package yoonforh.undo;

import java.awt.*;
import java.awt.event.ActionEvent;
import javax.swing.*;
import javax.swing.undo.*;

/**
 * undo/redo 예제 프로그램
 */

public class UndoEx extends JFrame {
  final Action increaseAction;
  final Action decreaseAction;
  final Action undoAction;
```

```
  final Action redoAction;
  final Action discardUndoAction;

  JLabel numberLabel;
  int number = 0; // label에 표시할 숫자
  UndoManager undoman = new UndoManager(); // undo/redo 관리자

    {  // 인스턴스 초기화 블록
       // 여기에서 각 액션들을 구현한다.
    }

  /**
   * 생성자. 레이블과 버튼들을 배치한다.
   */
  public UndoEx(String title) {
    super(title);

    // 숫자를 보여주는 레이블
    numberLabel       =       new       JLabel(String.valueOf(number),
SwingConstants.CENTER);
    numberLabel.setBorder(BorderFactory.createMatteBorder(5,  5,  5,  5,
Color.orange));
    numberLabel.setOpaque(true);
    numberLabel.setBackground(Color.yellow);
    numberLabel.setFont(new Font("SansSerif", Font.BOLD, 70));

    // 툴박스. 각 버튼을 배치한다.
    Box buttonBox = Box.createHorizontalBox();
    buttonBox.add(Box.createHorizontalGlue());
    buttonBox.add(new JButton(increaseAction));
    buttonBox.add(new JButton(decreaseAction));
    buttonBox.add(new JButton(undoAction));
    buttonBox.add(new JButton(redoAction));
    buttonBox.add(new JButton(discardUndoAction));
    buttonBox.add(Box.createHorizontalGlue());
```

```
    // 버튼 상태 초기화
    updateActions();

    // 자리 배치
    getContentPane().add(numberLabel, BorderLayout.CENTER);
    getContentPane().add(buttonBox, BorderLayout.SOUTH);

    // 윈도우 닫으면 종료
    setDefaultCloseOperation(EXIT_ON_CLOSE);
  }

  public static void main(String[] args) {
    JFrame f = new UndoEx("undo example program");
    f.setBounds(10, 10, 500, 300);
    f.setVisible(true);
  }
}
```

2. 각 버튼들의 액션을 정의한다.

AbstractAction 클래스를 상속하여 각 버튼들의 액션을 정의한다.

숫자 증가 액션인 increaseAction 을 보면 액션이 실행될 때, 숫자를 1 증가시킨 후 UndoManager 의 addEdit() 메소드를 호출하여 실행 취소 정보를 등록한다. 일반적으로 실행 취소 정보는 편집 문맥에 따라 매번 다른 객체가 생성되지만, 이 프로그램의 경우는 증가 혹은 감소의 단순한 동작이어서 현재 상태 정보를 가지지 않으므로 모두 동일한 객체를 등록하였다.

그리고 난 다음에는 각 버튼들의 상태를 갱신하는 메소드를 호출하였다.

숫자 감소 액션의 경우에도 동일하게 구현된다.

실행 취소와 재실행 액션, 그리고 실행 취소 정보 폐기 액션의 경우에는 간단하게 UndoManager 의 undo(), redo() 혹은 discardAllEdits() 메소드를 호출한 다음 각 버튼들의 상태를 갱신한다.

```
  { // 인스턴스 초기화 블록
    // 증가 액션 정의
    increaseAction = new AbstractAction("++") {
        public void actionPerformed(ActionEvent evt) {
          increaseNumber();
```

```
      undoman.addEdit(new IncreaseNumberEdit());
      updateActions();
    }
  };

// 감소 액션 정의
decreaseAction = new AbstractAction("--") {
    public boolean isEnabled() {
      return number > 0;
    }

    public void actionPerformed(ActionEvent evt) {
      decreaseNumber();
      undoman.addEdit(new DecreaseNumberEdit());
      updateActions();
    }
  };

// 실행 취소 액션 정의
undoAction = new AbstractAction("Undo") {
    public boolean isEnabled() {
      return undoman.canUndo();
    }

    public void actionPerformed(ActionEvent evt) {
      undoman.undo();
      updateActions();
    }
  };

// 재실행 액션 정의
redoAction = new AbstractAction("Redo") {
    public boolean isEnabled() {
      return undoman.canRedo();
    }
```

```
      public void actionPerformed(ActionEvent evt) {
        undoman.redo();
        updateActions();
      }
    };

  // 실행 취소 정보 폐기 액션 정의
  discardUndoAction = new AbstractAction("Discard undo info") {
      public boolean isEnabled() {
        return undoman.canUndoOrRedo();
      }

      public void actionPerformed(ActionEvent evt) {
        undoman.discardAllEdits();
        updateActions();
      }
    };

}
```

3. 취소 가능한 편집 정보 클래스를 정의한다.

두 경우 모두 AbstractUndoableEdit 클래스를 상속한 내부 클래스로 구현하였다. 여기에서 redo()와 undo() 메소드 구현 시 부모 클래스인 AbstractUndoableEdit 클래스의 해당 메소드를 먼저 호출하여 내부 상태를 갱신하여야 한다.

```
/**
 * 증가에 대한 실행 취소 및 재실행 정보 클래스
 */
class IncreaseNumberEdit extends AbstractUndoableEdit {
  public void redo() {
    super.redo(); // hasBeenDone 내부 필드를 true로 만든다.

    increaseNumber();
  }
```

```
  public void undo() {
    super.undo(); // hasBeenDone 내부 필드를 false로 만든다.

    decreaseNumber();
  }
}

/**
 * 감소에 대한 실행 취소 및 재실행 정보 클래스
 */
class DecreaseNumberEdit extends AbstractUndoableEdit {
  public void redo() {
    super.redo(); // hasBeenDone 내부 필드를 true로 만든다.

    decreaseNumber();
  }

  public void undo() {
    super.undo(); // hasBeenDone 내부 필드를 false로 만든다.

    increaseNumber();
  }
  }
```

4. 나머지 메소드들을 구현한다.

```
/**
 * 레이블의 숫자를 증가시킨다.
 */
void increaseNumber() {
  number++;
  numberLabel.setText(String.valueOf(number));
}

/**
 * 레이블의 숫자를 감소시킨다.
```

```
   */
  void decreaseNumber() {
    number--;
    numberLabel.setText(String.valueOf(number));
  }

  /**
   * 각 액션의 상태를 갱신한다. 결과적으로 버튼 상태를 갱신한다.
   */
  void updateActions() {
    // increase 액션은 항상 enable 상태
    decreaseAction.setEnabled(decreaseAction.isEnabled());
    undoAction.setEnabled(undoAction.isEnabled());
    redoAction.setEnabled(redoAction.isEnabled());
    discardUndoAction.setEnabled(discardUndoAction.isEnabled());
  }
```

5. 컴파일하여 실행해보자.

```
javac yoonforh\undo\UndoEx.java <엔터>

java -cp . yoonforh.undo.UndoEx <엔터>
```

[그림 4 – 실행 취소 예제 프로그램 실행 모습]

<따라하기 끝 – undo/redo 예제 프로그램>

17.4. 포커스 처리

17.4.1. KeyboardFocusManager 클래스와 포커스 정책

여러 개의 컴포넌트를 배치한 대화상자를 구현할 때 탭 키의 입력에 따라 각 컴포넌트를 어떤 순서로 순회할 것인지를 결정하는 것은 중요한 사용자 인터페이스의 일부이다.

자바의 AWT 윈도우 시스템에서는 키보드로 인한 포커스의 변화를 관리하는 KeyboardFocusManager 클래스와 포커스 순회 순서를 정하는 FocusTraversalPolicy 클래스를 통해 포커스를 처리한다.

KeyboardFocusManager 는 포커스를 다음 컴포넌트나 이전 컴포넌트로 옮기거나 윗쪽 포커스 사이클로 이동 혹은 아랫쪽 포커스 사이클로 이동하는 등의 포커스를 순회시키는 메소드와, 포커스에 관련된 컴포넌트와 윈도우 정보를 제공하는 메소드를 제공한다.

(1) public static KeyboardFocusManager getCurrentKeyboardFocusManager(); 현재의 키보드 포커스 관리자를 반환한다.

(2) public static void setCurrentKeyboardFocusManager(KeyboardFocusManager newManager); 현재의 키보드 포커스 관리자를 변경한다.

(3) public final void focusNextComponent(); 현재 포커스를 소유한 컴포넌트의 다음 컴포넌

트로 포커스를 순회한다.

(4) public void focusNextComponent(Component aComponent); 지정된 컴포넌트의 다음 컴포넌트로 포커스를 순회한다.

(5) public final void focusPreviousComponent(); 현재 포커스를 소유한 컴포넌트의 이전 컴포넌트로 포커스를 순회한다.

(6) public void focusPreviousComponent(Component aComponent); 지정된 컴포넌트의 이전 컴포넌트로 포커스를 순회한다.

(7) public final void upFocusCycle(); 현재 포커스를 소유한 컴포넌트를 포함한 포커스 사이클의 루트 컴포넌트로 포커스를 변경한다.

(8) public void upFocusCycle(Component aComponent); 지정한 컴포넌트를 포함한 포커스 사이클의 루트 컴포넌트로 포커스를 변경한다.

(9) public final void downFocusCycle(); 현재 포커스를 소유한 컴포넌트가 사이클 루트일 경우 이 컴포넌트를 사이클 루트로 하는 기본 컴포넌트로 포커스를 변경한다.

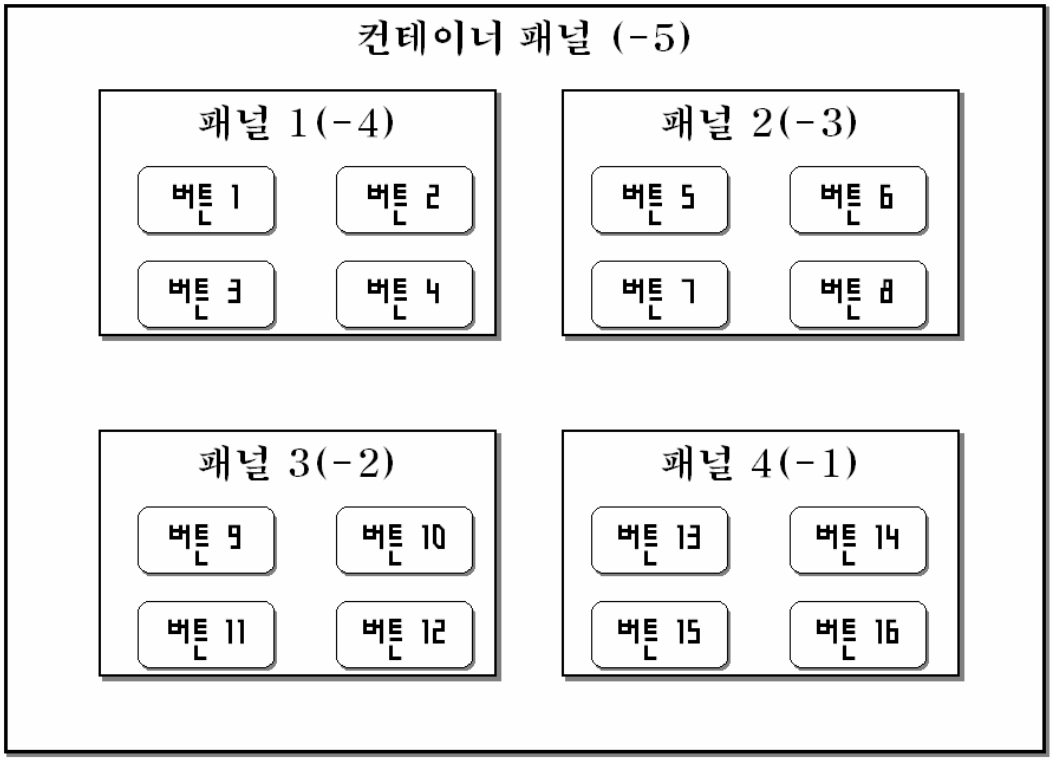
(10) public void downFocusCycle(Container aContainer); 지정한 컨테이너를 사이클 루트로 하는 기본 컴포넌트로 포커스를 변경한다.

KeyboardFocusManager 를 사용하지 않고 Component 와 Container 클래스에서 직접 포커스를 순회하고 요청하는 메소드들을 사용할 수도 있다.
Component 클래스의 transferFocus(), transferFocusBackward(), transferFocusUpCycle() 메소드와 Container 클래스의 transferFocusDownCycle() 메소드가 포커스를 순회하는 메소드들로, 예를 들어 KeyboardFocusManager 의 focusNextComponent(Component c) 메소드는 c 컴포넌트의 transferFocus () 메소드를 호출한 것과 동일한 효과를 가진다.
이 외에 Component 클래스의 requestFocusInWindow()는 해당 컴포넌트에 포커스를 요청하는 메소드이다.

AWT 컨테이너가 사용하는 기본 포커스 순회 정책은 java.awt 패키지의 DefaultFocusTraversalPolicy 이며 스윙 컨테이너가 사용하는 기본 포커스 순회 정책은 javax.swing 패키지의 LayoutFocusTraversalPolicy 이다.
DefaultFocusTraversalPolicy 는 컴포넌트의 계층 구조와 컨테이너에 컴포넌트가 등록된 순서를 기반으로 포커스 순회 순서를 정하며, LayoutFocusTraversalPolicy 는 컴포넌트의 크기와 위치를 참고하여 대강의 행과 열로 컴포넌트를 구분하고 한 행을 먼저 순회한 후, 다음 행을 순회하는 방식으로 포커스 순회 순서를 정한다.
이외에 스윙에서 사용하는 SortingFocusTraversalPolicy 는 각 컴포넌트의 어떤 값을 비교하여 그 대소관계에 따라 포커스 순회 순서를 정한다.

17.4.2. 포커스 사용 예제

여기에서 작성할 포커스 예제는 열 여섯 개의 버튼을 네 개의 패널에 배치하고 또 각 네 개의 패널은 하나의 패널에 배치되어 있다.



[그림 5 – 예제의 컴포넌트 배치]

그림에서 컨테이너 패널은 패널 1 에서 패널 4 까지를 포함하고 있는 컨테이너이며 이 다섯 컴포넌트 사이에 포커스 사이클이 구성되어 컨테이너가 사이클 루트가 된다.

네 개의 버튼을 포함하고 있는 패널들은 각각 버튼 네 개를 포함하는 다섯 컴포넌트 사이의 포커스 사이클을 구성하는 컨테이너이며 각 사이클의 루트가 된다. 이 포커스 사이클은 컨테이너 패널을 루트로 하는 다섯 패널의 포커스 사이클의 하위 사이클이 되며, 역으로 컨테이너 패널을 포함한 포커스 사이클은 이 포커스 사이클의 상위 사이클이 된다.

각 컨테이너를 사이클 루트로 만들기 위해서는 setFocusCycleRoot(true) 메소드를 각 컨테이너에 호출하면 된다.

그리고, 포커스 순회 정책 중 값의 대소를 비교하는 SortingFocusTraversalPolicy 에 사용하기 위해 각 패널은 –1 에서 –5 까지의 값을 그림과 같이 지정하였으며 버튼들은 0 에서 15 까지의 값을 실행 시에 난수로 나뉘어서 가질 것이다.

<따라하기 시작 – 포커스 순회 정책 예제>
포커스 순회 정책과 상위 사이클, 하위 사이클 등의 포커스 개념을 확인하기 위한 예제를 만들어보자.
열 여섯 개의 버튼과 다섯 개의 패널로 구성하며, 포커스 순회 정책은 스윙의 기본 정책인 LayoutFocusTraversalPolicy 와 값의 대소 비교에 따른 정책을 사용하는 SortingFocusTraversalPolicy 를 지원하도록 하였다.

1. 먼저 스윙 프로그램 골격을 구성한다.
FocusPanel 컴포넌트는 다섯 개의 패널에 사용할 컴포넌트로 포커스를 갖고 있지 않을 때에는 노란 색, 포커스를 얻으면 붉은 색을 가지도록 구현한 패널이다. 그리고, 패널에 안 여백을 약간 주어 배치된 컴포넌트들이 패널을 완전히 덮지 않도록 하여, 포커스 상태를 볼 수 있게 하였다.
또, 윈도우는 항상 포커스 사이클 루트가 되는데, 윈도우가 포커스를 받으면 컨테이너 패널로 포커스를 이동하도록 포커스 리스너를 사용하였다. 그외, 포커스 정책을 변경하고 포커스를 이동시키는 버튼과 콤보박스는 포커스를 받지 않도록 하여 포커스 정책에 영향을 미치지 않게 하였다.
SortingFocusTraversalPolicy 에 사용될 값들을 패널과 버튼에 지정하기 위해 JComponent 클래스의 putClientProperty() 메소드를 이용하였다. 이 메소드를 사용하면 스윙 컴포넌트에 임의의 값을 키와 함께 저장해둘 수 있다.

```
package yoonforh.focus;

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

/**
 * 포커스 처리 예제
 */

public class FocusEx extends JFrame
    implements ActionListener, ItemListener, FocusListener, Comparator
{
  // 컴포넌트의 대소 비교에 사용할 값의 키
  final static String NUMBER_KEY = "number";
```

```
  final static Insets panelInsets = new Insets(10, 10, 10, 10); // 패널
의 안 여백
  JPanel container;
  JPanel[] panels;

  /**
   * 다섯 개의 패널에 사용될 컴포넌트
   * 포커스를 받으면 붉은 색으로 색이 바뀐다.
   */
  class FocusPanel extends JPanel {
    FocusPanel() {
      setBackground(Color.yellow);
      addFocusListener(FocusEx.this);
    }

    /**
     * 내부 여백을 사용하여 패널이 포커스를 받은 경우를 볼 수 있게 한다.
     */
    public Insets getInsets() {
      return panelInsets;
    }
  }

  /**
   * 생성자. 열 여섯 개의 버튼을 네 개의 패널에 나누어 담아 배치한다.
   * 각 버튼과 패널 그리고 컨테이너 패널에는 서로 구분되는 숫자
   * 정보를 지정한다.
   */
  public FocusEx(String title) {
    super(title);

    container = new FocusPanel();
    container.putClientProperty(NUMBER_KEY, new Integer(-5));
    container.setLayout(new GridLayout(2, 2));
    // 컨테이너 패널을 포커스 사이클 루트로 지정
    container.setFocusCycleRoot(true);
```

```
// 패널과 버튼들을 생성한다.
createComponents();

// 툴박스. 각 버튼을 배치한다.
Box buttonBox = Box.createHorizontalBox();
// 툴박스의 컴포넌트들은 포커스를 받아서는 안된다.
buttonBox.setFocusable(false);
buttonBox.add(Box.createHorizontalGlue());
JButton button = new JButton("next");
button.setFocusable(false);
button.addActionListener(this);
buttonBox.add(button);
button = new JButton("previous");
button.setFocusable(false);
button.addActionListener(this);
buttonBox.add(button);
button = new JButton("up cycle");
button.setFocusable(false);
button.addActionListener(this);
buttonBox.add(button);
button = new JButton("down cycle");
button.setFocusable(false);
button.addActionListener(this);
buttonBox.add(button);

// 포커스 순회 정책을 선택하는 콤보박스
JComboBox combo = new JComboBox(
  new Object[] { "layout order", "sorting order" });
combo.setFocusable(false);
combo.addItemListener(this);
buttonBox.add(combo);
buttonBox.add(Box.createHorizontalGlue());

// 자리 배치
getContentPane().add(container, BorderLayout.CENTER);
getContentPane().add(buttonBox, BorderLayout.SOUTH);
```

```
    // 윈도우 닫으면 종료
    setDefaultCloseOperation(EXIT_ON_CLOSE);

    addFocusListener(new FocusAdapter() {
        public void focusGained(FocusEvent evt) {
          // 윈도우가 포커스를 받으면 항상 패널 컨테이너로 포커스를 이동 요청한다.
          container.requestFocusInWindow();
        }
      });
  }

  public static void main(String[] args) {
    JFrame f = new FocusEx("focus example program");
    f.setBounds(10, 10, 500, 300);
    f.setVisible(true);
  }
}
```

2. 포커스 순회에 참여할 패널과 버튼을 생성하여 등록한다.
각 패널은 포커스 사이클 루트로 지정하며, 순회에 사용할 값을 지정한다.
각 버튼은 0 에서 15 사이의 난수 방식으로 서로 다른 값을 가지도록 하였다.

```
  /**
   * 열 여섯 개의 버튼을 네 개의 패널에 네 개씩 배치한다.
   * 각 버튼에는 0에서 15까지의 숫자값이 난수로 지정된다.
   */
  void createComponents() {
    JButton[] buttons = new JButton[16];

    for (int i = 0; i < buttons.length; i++) {
      int value = getRandomValue(); // 서로 다른 난수값을 지정
      buttons[i] = new JButton(String.valueOf(value));
      buttons[i].putClientProperty(NUMBER_KEY, new Integer(value));
      buttons[i].setBackground(Color.yellow);
      buttons[i].addFocusListener(this);
```

```
  }

  panels = new JPanel[4];
  for (int i = 0; i < panels.length; i++) {
    panels[i] = new FocusPanel();
    panels[i].putClientProperty(NUMBER_KEY, new Integer(-(i + 1)));
    panels[i].setLayout(new GridLayout(2, 2));
     panels[i].setFocusCycleRoot(true);
    for (int j = 0; j < 4; j++) {
      panels[i].add(buttons[i * 4 + j]);
    }
  }
  for (int i = 0; i < 4; i++) {
    container.add(panels[i]);
  }
}

private BitSet bitset = new BitSet(5); // 이미 지정된 값들을 기억한다.
private Random random = new Random(); // 난수 발생기

/**
 * 0에서 15 사이의 값을 랜덤하게 하나씩 지정한다.
 */
private int getRandomValue() {
  int value = -1;
  while (true) {
    value = (int) (random.nextFloat() * 16);
    if (!bitset.get(value)) {
      bitset.set(value);
      break;
    }
  }

  return value;
}
```

3. 툴박스 버튼의 액션 이벤트를 처리한다.

```
/** 버튼의 액션 이벤트 처리 */
public void actionPerformed(ActionEvent evt) {
  String cmd = evt.getActionCommand();

  KeyboardFocusManager focusman
    = KeyboardFocusManager.getCurrentKeyboardFocusManager();
  if (cmd.equals("next")) {
    focusman.focusNextComponent();
  } else if (cmd.equals("previous")) {
    focusman.focusPreviousComponent();
  } else if (cmd.equals("up cycle")) {
    focusman.upFocusCycle();
  } else if (cmd.equals("down cycle")) {
    focusman.downFocusCycle();
  }
}
```

4. 버튼과 패널의 포커스 이벤트를 처리하여 포커스를 얻거나 잃을 때 색깔이 바뀌도록 한다.

```
/**
 * 포커스를 얻은 버튼과 패널의 배경을 붉은색으로 한다.
 */
public void focusGained(FocusEvent e) {
  JComponent jc = (JComponent) e.getSource();
  jc.setBackground(Color.red);
}

/**
 * 포커스를 잃은 버튼과 패널의 배경을 노란색으로 한다.
 */
public void focusLost(FocusEvent e) {
```

```
  JComponent jc = (JComponent) e.getSource();
  jc.setBackground(Color.yellow);
}
```

5. 콤보박스의 아이템 이벤트를 처리하여 포커스 순회 정책을 변경한다.

```
/**
 * 포커스 순회 정책을 변경한다.
 * 정책을 변경할 때, 컨테이너가 포커스 사이클 루트인 경우에만
 * 새 정책이 반영된다.
 */
void changeFocusTraversalPolicy(FocusTraversalPolicy policy) {
  for (int i = 0; i < panels.length; i++) {
    panels[i].setFocusTraversalPolicy(policy);
  }
  container.setFocusTraversalPolicy(policy);

  // 컨테이너 패널로 포커스 요청
  container.requestFocusInWindow();
}

/**
 * 콤보박스 이벤트 처리
 */
public void itemStateChanged(ItemEvent evt) {
  if (evt.getStateChange() == ItemEvent.SELECTED) {
    KeyboardFocusManager focusman
      = KeyboardFocusManager.getCurrentKeyboardFocusManager();
    String value = (String) evt.getItem();

    if (value.equals("layout order")) {
      LayoutFocusTraversalPolicy policy
        = (LayoutFocusTraversalPolicy)
        focusman.getDefaultFocusTraversalPolicy();
      changeFocusTraversalPolicy(policy);
    } else if (value.equals("sorting order")) {
```

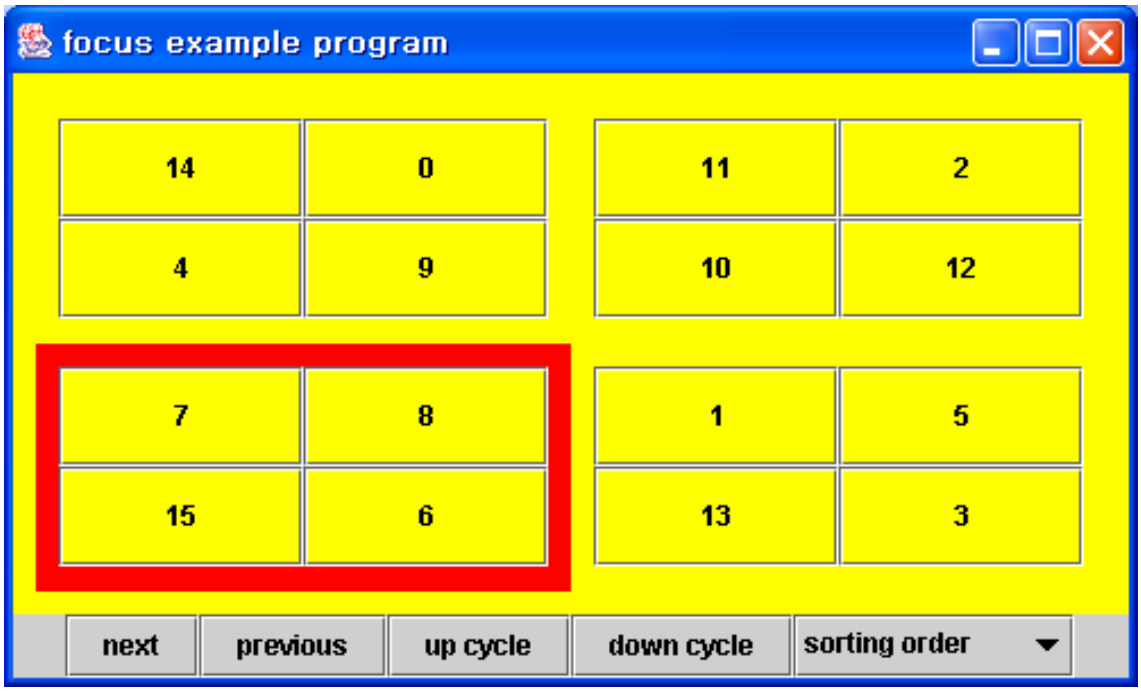
```
        SortingFocusTraversalPolicy policy
          = new SortingFocusTraversalPolicy(this);
        // 포커스 사이클 루트에서 자동으로 다운 사이클로 순회하는 것을 막음.
          policy.setImplicitDownCycleTraversal(false);
        changeFocusTraversalPolicy(policy);
      }
    }
  }

  /**
      * 정렬 순서 방식의 포커스 순회에서 사용할
      * Comparable 인터페이스 구현 메소드
   * 각 JComponent에 입력되어 있는 숫자 값을 비교한다.
   */
  public int compare(Object o1, Object o2) {
    int          v1          =          ((Integer)          ((JComponent)
o1).getClientProperty(NUMBER_KEY)).intValue();
    int          v2          =          ((Integer)          ((JComponent)
o2).getClientProperty(NUMBER_KEY)).intValue();

    return (v1 - v2);
  }
```

6. 컴파일하여 실행해보자.

```
javac yoonforh\focus\FocusEx.java <엔터>

java -cp . yoonforh.focus.FocusEx <엔터>
```

마이크로소프트 윈도우나 유닉스의 X 윈도우 시스템에서 실행할 경우 탭 키(혹은 Ctrl+탭 키)와 Shift+탭 키(혹은 Ctrl+Shift+탭 키)를 사용하여 정방향 순회와 역방향 순회를 각각 할 수 있다.
물론 툴박스의 next 버튼과 previous 버튼을 사용할 수도 있다.
상위 사이클 루트에서 진행이 안될 경우에는 각 사이클 루트에 포커스가 있을 때, “down cycle” 버튼을 눌러 하위 포커스 사이클을 순회하도록 한 후에 순회 키를 누르면 된다.
각 패널이 사이클 루트이기 때문에 하위 포커스 사이클로 들어가면 순회 키를 사용해서는

다른 패널로 이동할 수가 없다.
포커스 순회 정책에 따라 기본 포커스 순회 정책인 LayoutFocusTraversalPolicy 의 경우 왼쪽에서 오른쪽, 위에서 아래 방향으로 순회가 되고, SortingFocusTraversalPolicy 의 경우 패널에 지정한 –1 에서 –5 의 값과 버튼에 난수 발생으로 지정한 0 에서 15 까지의 값의 대소에 따라 포커스 순회 차례가 결정되는 것을 확인할 수 있다.



[그림 6 – 예제 프로그램 실행 모습]

<따라하기 끝 – 포커스 순회 정책 예제>

맺음말

이 장에서는 그래픽 사용자 인터페이스에 관련된 복잡한 기술들을 다루었다. 이 기술들을 활용하면 본격적인 고급 사용자 인터페이스를 제공할 수 있을 것이다.

다음 장에서는 그래픽에서 텍스트를 처리하는 고급 기법과 인쇄에 관해 다룬다.