7장

고급 AWT

- 파이프라인 방식의 렌더링
- 형태(shape)
- 영역(area)
- 스트로크
- 페인트
- 좌표 변환
- 클리핑
- 투명도와 컴포지션
- 렌더링 힌트
- 이미지 조작
- 프린팅
- 클립보드
- 드레그-앤-드롭

제1권에서, 간단한 그리기를 할 때 Graphics 클래스의 메소드를 사용하는 방법을 살펴 보았다. 이러한 메소드들은 간단한 애플릿과 애플리케이션에는 충분하지만, 복잡한 형태나 그래픽 외형 전체를 관리할 경우에는 부족한 점이 많이 있다. 자바 2D API를 사용하면 보다 최근 클래스 라이브러리로 향상된 그리기를 할 수 있다. 이번 장에서는 이러한 API에 대해서 간략히 살펴 본다.
그리고 프린팅 쪽으로 주제를 옮겨서 프로그램 상에서 프린팅을 하는 방법을 살펴본다. 마지막으로, 프로그램들사이에서 데이터를 옮기는 두 가지 테크닉:시스템 클립보드와 드레그-앤-드롭(drag-and-drop) 매커니즘을 다룬다.

파이프라인 방식의 렌더링

원래 JDK1.0에는 아주 간단한 메커니즘의 그리기 형태만을 제공한다. 색상과 페인트 모드를 고르고, drawRect 나 fillOval같은 Graphics 클래스의 메소드를 호출한다. Java 2D API는 보다 많은 옵션들을 지원한다.

- 광범위하게 다양한 형태(*shape*)를 쉽게 만들어 낼 수 있다.
- 스트로크(Stroke)를 제어해서 펜이 형태 가장자리를 찾게 한다.
  입체색상, 다양한 빛깔 그리고 반복되는 패턴으로 형태를 채울 수 있다(Fill).
  형태 변환(Transformation)을 이용해, 형태를 이동하고, 크기를 조절하고 회전시키고 잡아

당겨 늘릴 수 있다.
컴포지션 규칙(Composition rule)을 선택해서 새로운 형태의 Pixel들과 기존의 픽셀들을 조합하는 방법을 나타낼 수 있다.
랜더링 힌트를 주어 속도와 그리기 특성을 상쇄(trade-off)시킬 수 있다.

형태를 그리기 위해서, 다음의 단계를 밟는다.

1. Graphics2D 클래스의 객체를 얻는다. 이 클래스는 Graphics 클래스의 하위 클래스이다. Java 2D가 가능한 JDK 버전을 사용하고 있다면, paint와 paintComponent 같은 메소드들은 Graphics2D 클래스의 객체를 자동으로 받아온다. 간단히 형변환을 사용하면, 아래와 같다.
2. setRenderingHints 메소드를 사용해서 랜더링 힌트를 설정해 속도와 그리기 특성간의 차이를 상쇄시킨다.
3. setStroke 메소드를 사용해서 stroke을 설정한다. 스트로크는 형태의 아웃라인을 그리는데 사용한다. 두껍고 일정한 점선들 중에서 선택 가능할 수 있다.
4. setPaint 메소드를 사용해서 paint을 설정한다. 페인트는 스트로크 패스나 형태 내부 같은 영역을 채울 때 사용된다. 고른 색상의 페인트와 변화하는 빛깔 또는 타일로 채워진 패턴을 만들어 낼 수 있다.
5. setClip 메소드를 사용해서 오려낼 부분을 설정한다.
6. setTransform 메소드를 사용해서 사용자 부분에서 디바이스 부분까지의 변형을 설정한다. 사용자가 픽셀 조정을 하는 것 보다 사용자 정의 조정 시스템에서 형태를 정의하는 것이 보다 편하다면 변형을 할 수 있다.
7. setComposite 메소드를 사용하여 컴포지션 규칙(Composition rule)을 설정해 새로운 픽셀과 기존의 픽셀들을 결합하는 방법을 나타낸다.
8. 형태를 만들어낸다. Java 2D API는 많은 형태의 객체와 메소드들을 제공하여 형태들을 결합한다.
9. 그리고 형태를 채운다. 형태를 그릴 때 해당 아웃라인을 긋고, 형태를 채울 때 해당 내부가 칠해 진다.

물론, 대부분의 실제 상황 하에서는 이 모든 단계를 밟을 필요가 없다. 2D 그래픽 관계를 설정함에 있어서 적당한 디폴트 값이 있다. 디폴트 값을 바꾸고 싶다면, 설정만 바꿔주면 된다.

다음 절에서는 형태, 스트로크, 페인트, 형태변형 그리고 컴포지션 규칙(Composition rule)을 구현하는 방법을 살펴본다.

다양한 set 메소드는 단순히 2D 그래픽 관계 상태를 설정한다. 이러한 메소드들은 어떤 그리기 동작도 일으키지 않는다. 비슷하게, shape 객체를 구성할 때도 그리기 동작은 발생하지

않는다. draw 나 fill을 호출하면 단순히 형태가 만들어 진다. 이 때, 새로운 형태는 파이프라인 방식의 랜더링으로 만들어진다. (그림 7-1 참조)

그림 7-1: 파이프라인 방식의 렌더링

파이프라인 방식의 랜더링에서, 형태를 만들기 위한 단계는 다음과 같다.

1. 형태의 패스가 선으로 그어 진다.
2. 형태가 변형된다.
3. 형태가 잘리고, 형태와 잘린 부분에 교차점이 없으면 여기서 멈춘다.
4. 잘린 다음 나머지 형태가 채워진다.
5. 채워진 형태의 픽셀들은 기존에 있던 픽셀들로 구성된다. (그림 7-1, 원은 기존에 있던 픽셀들의 일부이고, 컵 모양의 형태가 그 위에 덮어 씌워진다.)

다음 절에서, 형태들을 정의하는 방법을 살펴본다. 그런 다음, 우리는 2D 그래픽 컨텍스의 설정으로 복귀할 것이다.

java.awt.Graphics2D

- void draw(Shape s)
  현재 그어진 선으로 주어진 형태의 아웃라인을 그린다.
- void fill(Shape s)
  현재 페인트로 주어진 형태의 내부를 채운다.

형태(shape)

Graphics 클래스에는 형태를 그리는 몇 가지 메소드가 있다..

drawLine
drawRectangle
drawRoundRect
draw3DRect
drawPolygon
drawPolyline
drawOval
drawArc

이에 상응하는 fill 메소드 또한 있다. 이러한 메소드들은 JDK1.0 이후 Graphics 클래스 내에 있었다. Java 2D API 는 완전히 다르며, 객체지향 접근을 한다. 메소드들 대신에, 다음 클래스들이 존재한다.

Line2D

Rectangle2D

RoundRectangle2D

Ellipse2D

Arc2D

QuadCurve2D

CubicCurve2D

GeneralPath

이러한 클래스들은 모두 Shape **인터페이스를 구현한다**.

형태를 그리기 위해서, Shape **인터페이스를 구현하는 클래스의 객체를 만들고** Graphics 2D **클래스의** draw **메소드를 호출한다**.

Line2D, Rectangle2D, RoundRectangle2D, Ellipse2D **와** Arc2D **클래스가** drawLine, drawRectangle, drawRoundRect, drawOval, drawArc **메소드와 동일하다**.(' 3D Rectangle' **의 개념은 크게 유용한 것은 아니다**. -- draw3Drect **메소드 같은 것은 없다**.) Java 2D API**는 두개의 부가적인 클래스: 이차곡선과 입체곡선을 제공한다**:. **이번 절의 후반부에서 이러한 형태에 대해 논의할 것이다**. Polygon 2D **클래스는 없다**. **대신에**, GeneralPath **클래스가 일차**, **이차곡선 그리고 입체곡선을 구성하는 방법을 설명하다**. **다각형**(Polygon)**을 그릴 때** GeneralPath**가 사용될 수 있다**.:**이번 절의 후반부에서 설명될 것이다**.

형태(shape) 클래스 계층도

Java 2D **형태 클래스를 사용하기 매우 복잡하다**. JDK1.0**에서** draw **메소드가 정수 픽셀 좌표를 사용한 것과는 달리**, Java 2D **형태에서는 부동소수점 좌표를 사용한다**. (**밀리미터나 인치들 처럼**)**여러분에게 의미있는 좌표들로 형태들을 명시하고 이를 픽셀로 변환해주기 때문에 대단히 편리하다**.

Java 2D **라이브러리는 많은 내부의 부동소수점 계산시에 단일 정밀 부동숫자**(single precision float)**를 사용한다**. **단일정밀 숫자가 충분하지만**, **결국은 기하 산술의 최종 목적은 스크린상이나 프린터에 설정하는 것이다**. **한 픽셀에 어떤 큰 에러가 있더라도**, **눈에 보이는 결과는 크게 영향을 주는 것 아니다**. **게다가**, **어떤 폴랫폼에서는 부동소수 산술식이 보다 빠르며**, **부동소수값이** double value**보다 절반의 저장 공간밖에 차지 하지 않는다**.

그러나, **자바프로그램밍 언어가** double value**을** float value**로 변환시에 형변환에 대해** adament

하기 때문에 부동 소수값을 사용하는 것은 간혹 자바 프로그래머에게 불편함을 준다.

```
float f = 1.2; //error
```

이 문장은 컴파일 시 에러가 난다. 상수 1.2는 더블 타입(double type)으로, 컴파일러는 정밀성을 잃는 것에 대해서 긴장하게 된다. 해결책은 F을 말미에 붙여서 부동소수화 시키는 것이다.

```
Float f = 1.2F; //ok
```

이번에는 다음 문장을 보자:

```
Rectangle r = ..
float f = r.getWidth();// Error
```

이 문장 또한 같은 이유로 컴파일 되지 않는다. getWidth 메소드는 double값을 돌려준다. 이번에는 형변환 해주면 된다.:

```
float f = (float)r.getWidth(); // ok
```

접미어와 형변환을 하는 것은 좀 불편하지만, 2D 라이브러리 디자이너는 각 형태 클래스에 두개의 버전을 제공하기로 했다: 부지런한 프로그래머에게는 부동좌표와 함께, 게으른 프로그래머에게는 더블 좌표와 함께.(이 책에서, 우리는 두번째 부류에 속하며 가능한 한 더블 좌표를 사용한다.)

라이브러리 디자이너들은 처음에는 혼란스럽고 짜증나는, 이러한 선택들을 묶기 위한 메소드를 선택했다. Rectangle2D 클래스를 고려해 보자. 이 클래스는 두개의 하위 클래스를 가지고 있는데, 모두 정적 내부(static inner) 클래스이다.

```
Rectangle2D.Float
Rectangle2D.Double
```

그림7-2 는 상속 다이어그램을 보여준다.

그림7-2: 2D rectangle 클래스들

두개의 추상적인 클래스가 정적 내부(static inner) 클래스라는 사실은 무시하는게 가장 좋다.- 그것은 FloatRectangle2D 와 DoubleRectangle2D와 같은 이름들을 피하기 위한 속임수이다. (정적 내부 클래스에 대한 보다 자세한 정보는 6장에 있다.)

Rectangle2D.Float 객체을 구성할 때, 부동소수점으로 좌표값을 제공한다. Rectangle2D.Double

객체에 있어서는, double값을 제공한다..

```
Rectabgle2D.Float floatRect = new Rectangle2D.Float(10.0F , 25.0F, 22.5F, 20.0F);
Rectabgle2D.Double doubleRect = new Rectangle2D.Double(10.0 , 25.0, 22.5, 20.0);
```

실제로, Rectangle2D.Float 와 Rectangle2D.Double가 동일한 Rectangle2D로 확장된 이후로, 하위 클래스에서 메소드들은 단순히 Rectangle2D 상위 클래스에서 메소드들을 오버라이드하고, 정확한 형태 타입을 기억하는 것은 무의미하다. 단순히 Rectangle2D변수들을 이용해서 직사각형의 참조를 구한다.

```
Rectabgle2D float floatRect = new Rectangle2D.Float(10.0F , 25.0F, 22.5F, 20.0F);
Rectabgle2D double doubleRect = new Rectangle2D.Double(10.0 , 25.0, 22.5, 20.0);
```

그것은, 형태 객체를 구성시에 성가신 내부 클래스를 사용할 필요가 있다는 것이다.
구성 전달인자는 맨왼쪽 코너, 두께, 직사각형의 높이를 나타낸다.

노트 : 사실, Rectangle2D.Float 클래스는 Rectangle2D 다시 말해, setRect(float x, float y, float h, float w) 로부터 상속받지 않는 부가적인 메소드를 가지고 있다. Rectangle2D 변수에 Rectangle2D.Float 참조를 저장하면 그 메소드를 잃어 버린다. 그러나 큰 손실이 아니고, Rectangle2D 클래스는 double 전달인자를 가진 setRect 메소드를 가진다.

Rectangle2D 메소드는 double 전달인자를 사용하여 값을 반환한다. 예를 들면, getWidth 메소드는 너비가 Rectangle2D.Float 객체에 부동 소수 값으로 저장되어 있어도, 더블 값을 반환한다.

팁 : 플로팅 값으로 계산하지 않기 위해서 단순히 Double 형태 클래스를 사용한다. 그러나, 수천 개의 형태 객체를 생성한다면, 메모리 보존차원에서 플로팅 클래스 사용을 고려할 수 있다.

Rectangle2D 클래스에 대해 논의했던 모든 것이 기타 형태 클래스에서 마찬가지이다. 게다가,

하위 클래스 Point2D.Float와 Point2D.Double을 가지는 Point2D 클래스가 있다. 이것이 포인트 클래스를 만든 것이다.

```
Point p = new Point2D.Double(10, 20);
```

팁 : Point2D 클래스는 매우 유용하다-구별자 X와 Y값을 사용하는 것 보다 Point2D 오브잭트을 이용하는 것이 보다 객체 지향적이다. 많은 생성자와 메소드들은 Point2D 전달인자를 받아들인다. 가능하면 Point2D 객체를 사용하기를 권장하며, 기하산술식을 보다 쉽게 이해하게 한다.

다음 클래스는 모두 동일한 상위 클래스 RectanglarShape 으로 부터 상속된다.

Rectangle2D

RoundRectangle2D

Ellipse2D

Arc2D

부가적으로, 타원과 호는 정확히 구형은 아니지만, 그것들은 바운딩 구형이다. (그림 7-3 참조)

그림 7-3: 타원과 호의 바운딩 구형

RectanglarShape 클래스는 getWidth, getHeight, getCenterX, getCenterY같은 유용한 메소드들중에서 이러한 형태에 공통적인 20개가 넘는 클래스를 정의한다. (그러나, 불행히도 이번에 getCenter 메소드는 중앙을 Point2D 객체로 반환하지 않는다.)

최종적으로, 형태 클래스 계층에 맞는 JDK1.0에서의 리거시 클래스 2개는 Rectangle 과 Point 클래스로, 정수 좌표를 가진 직사각형과 점을 저장하고 Rectangle2D 와 Point2D 클래스로 확장된다. Polygon 클래스는 현재 shape 인터페이스를 구현하고 있다. 그럼에도 불구하고 새로운 코드에는 이러한 클래스의 사용을 자제하는 것이 좋다.

우리는 다음 절에서 아래과 같이 남아있는 형태 클래스들에 대해 논의할 것이다.

QuadCurve2D,

CubicCurve2D

GeneralPath

Area

그림 7-4는 형태 클래스들간의 관계를 보여준다. 그러나, Double과 Float 하위 클래스는 생략되었다. 리거시 클래스들은 회색으로 채워져 표시되어 있다.

그림 7-4 : 형태 클래스들간의 관계

java.awt.geom..RectanglarShape

- double getCenterX()
- double getCenterY()
- double getMinX()
- double getMinY()
- double getMaxX()
- double getMaxY()

  은 둘러싼 직사각형의 중심, 최소, 최대 X 혹은 Y값을 반환한다.
- double getWidth()
- double getHeight()

  은 둘러싼 직사작형의 너비와 높이를 반환한다.
- double getX()
- double getY()

  는 둘러싼 직사각형의 맨위 왼쪽 코너의 x 혹은 Y 좌표값을 반환한다.

java.awt.geom..Rectangle2D.Double

- Rectangle2D.Double(double X, double y, double w, double h)는 주어진 맨 위 왼쪽 좌표값과 너비, 높이로 직사각형을 생성한다.

java.awt.geom..Rectangle2D.Float

- Rectangle2D.Float(float x, float y, float w, float h)는 주어진 맨 위 왼쪽 좌표에 주어진 높이와 너비의 직사작형을 생성한다.

형태 클래스들 사용하기

Rectangle2D, RoundRectangle2D와 Ellipse2D 객체는 생성하기에 매우 단순하다.

- 맨 왼쪽 코너의 X, Y좌표값
- 너비와 높이

만을 명시하면 된다.

예를 들면,

```
Rectangle2D r = new Rectangle2D.Double(150, 200, 100, 50);
```

은 (150, 200) 좌표인 지점에서 너비 100, 높이 50인 직사각형을 생성한다.

RoundRetangle2D 형태에서, 둥근부분에 대한 X, Y 너비가 명시되어야 한다.(그림 7-5 참조).

예를 들면,

```
RoundRectangle2D r = new RoundRectangle2D.Double(150, 200, 100, 50, 20, 20);
```

은 각 코너마다 반지름이 20인 원형의 사각형을 생성한다.

그림 7-5: RoundRectangle2D 생성하기

그러나 때때로, 즉시 가능한 맨 위 왼쪽 코너값이 없을 수 있다. 직사각형에서 두 개의 대각선을 갖는 것이 일반적이지만, 맨 위 왼쪽과 맨 아래 오른쪽 코너가 아닐 수 도 있다.

```
Rectangle2D r = new Rectangle2D.Double(px, py, qx – px, qy – py); //오류
```

는 간단하게 직사각형을 생성할 수 없다.

P가 맨 위 왼쪽이 아니라면, 하나 아니 두개의 좌표값의 차이는 음수가 되고 직사각형은 비게된다. 이러한 경우, 첫번째로 빈 직사각형이 되면 setFrameFromDiagonal 메소드를 사용한다.

```
Rectangle2D r = new Rectangle2D.Double();
r.setFrameFromDiagonal (px, py, qx, qy);
```

혹은 보다 좋은 방법으로, Point2D 객체 p와 q의 코너 좌표를 안다면,

```
r.setFrameFromDiagonal(p, q);
```

타원을 생성할때는, 바운딩 직사각형의 코너 포인트가 아닌(바운딩 직사각형이 타원에 누워있지 않으면) 일반적으로 중심, 너비, 높이만을 알 것이다. 중심점을 이용한 setFrameFrom Center 메소드가 있지만, 사방 코너점의 하나을 요구할 것이다. 그래서, 다음과 같이 타원형을 생성하면서 끝을 낸다.

```
Ellipse2D e = new Ellipse2D.Double(centerX – width / 2,
 CenterY – height / 2, width, height );
```

호를 생성할때는, 시작각도와 호만큼 기운 각도와 종결타입, Arc2D.OPEN, Arc2D.PIE, 혹은 Arc2D.CHORD 중의 하나에 따른 바운딩 박스를 명시한다.

```
Arc2D a = new Arc2D(x, y, width, height,
        Startangle, arcAngle, closureType);
```

그림 7-6: 타원호의 생성

그림 7-7 **은 호 타입을 나타낸다.**

그림 7-7 : 호 타입

그러나, 각도는 정도에 따라 주어진 만큼 간단하지 않다. 각도가 틀어졌지만, 45**도 정도의 각도는 너비와 높이가 같지 않더라도 대각선위치을 나타낸다. 원형호(예를 들어 파이차트에서) 를 그린다면, 이것에 대한 염려는 필요없다. 그러나, 타원호의 경우는 삼각법** (Trigonometry)**을 경험할 준비를 해야 한다.– 자세한 것은 사이드 바를 참조하라.**

타원호에서 각도 나타내기.==========

타원호를 그리는 알고리즘은 뒤틀린 각도를 사용하는데, Caller**는 미리 계산되어져야 한다. 이 사이드 바는 어떻게 구현하는지 알려준다. 본인이 타원호를 그려본 적이 없는 많은 프로그래머에 속해있다면, 사이드 바를 생략하라. 그러나, 공식적인 기록문서는 이러한 주제를 잘 정리놓았기 때문에, 우리는 이 정보가 몇시간의 삼각법의 고통을 필요로 하는 이들을 구제하기 위해 기록되었다고 생각한다.**

아래의 공식으로 실제 각도를 뒤틀린 각도들로 전환시킨다.

```
distortedAngle = Math.atan2(Math.sin(angle) * width,
        Math.cos(angle) * height );
```

예를 들면 (이번 장의 끝부분에 있는 예제 같은), 호의 끝점을 알거나, 타원의 중심과 끝점이 접하는 선상의 다른 점을 알고 있다. 이런 경우, 첫번째 산식은

```
dx = p.getX() – center.getX();
dy = p.getY() – center.getY();
```

이고, 뒤틀린 각도는

```
distortedAngle = Math.atan2(-dy * width, dx * height );
```

(dy**앞에 있는 마이너스 부호는 픽셀 좌표내에 있으므로 필요한 것이다.** Y **축 점은 아래를 가르키며 시계방향으로 돌면서 각도 측정을 한다. 그러나, 측정된 시계 반대 방향의 각도는 주어져야 한다.**)

결과 값을 라디안에서 각도로 변환한다.:

```
distortedAngle = Math.toDegrees(distortedAngle);
```

결과값은 –180과 180사이의 값이다.
이러한 식으로 뒤틀린 시작과 끝각도를 계산한다. 그리고, 두 뒤틀린 각도 사이의 차이를 계산한다.
시작 각도나 차이가 음수이면, 360을 더한다. 그러면 호 생성자에 시작 각도와 각도 차이를 준다.

```
Arc2D a = new Arc2D(x, y, width, height,
 DistortedStartAngle, distortedAngleDifference, closureType);
```

비틀림에서는 문서화의 모호성뿐 아니라, 뒤틀린 각도의 차이값인 '호의 각도'을 호출하는 잘못을 유출해 낸다. 원호의 경우를 제외하고는, 이러한 값은 실제 호의 각도가 아닐 뿐 아니라 뒤틀림 또한 아니다.
이번 절 끝부분의 예제 프로그램을 실행시키면, 이러한 계산은 호 생성자에 대한 정확한 값을 산출함을 체크할 수 있다.(그림 7-10)

==========

Java 2D 패키지는 이차 커브(quadratic)과 입체(Cubic) 커브를 제공한다.. 이번 장에서, 이러한 커브에 수학적 계산까지는 가지 않는다. 예제 7-1의 프로그램을 실행하면 커브가 어떻게 보일까 하는 생각을 하길 권유한다. 그림 7-8과 7-9을 보면, 이차커브 및 입체커브가 두개의 끝점과 하나 혹은 두개의 제어점으로 표시되어 있다. 제어점을 움직이면 커브의 형태를 변환시킨다.

그림 7-8: 이차 곡선

그림 7-9 : 입체 곡선

이차 및 입체 곡선을 생성하기 위해서, 끝점과 제어점의 좌표를 준다. 예를 들면,

```
QuadCurve2D q = new QuadCurve2D.Double(startX, startY, controlX, controlY, endX, endY);
CubicCurve2D c = new CubicCurve2D.Double(startX, startY, controlY, control1X, control1Y, control2X, control2Y, endX, endY);
```

이차 곡선은 그리 탄력적이지 않고, 실제로 많이 쓰이지 않는다. 입체 곡선(CubicCurve3D 클래스에 의해 그려진 Bezier 곡선)은 그러나, 매우 많이 쓰이다. 많은 입체 곡선을 결합하면, 결합점의 경사가 맞춰지고, 복잡하지만 부드럽게 보이는 곡선 형태를 만들수 있다. 보다 자

세한 사항은, Computer Graphics: Principles and practce, Second edition in C, James D.Foley, Andries van Dam, Steven K.Feiner, et al.[Addison Wesley 1995]을 참고하기 바람.

선 조각들과 이차 곡선들, 그리고 입체 커브의 임의 순서를 정하고, GeneralPath 객체에 그것들을 저장한다. MoveTo 메소드로 첫번째 패스의 좌표을 명시한다. 예를 들면,

```
GeneralPath path = new GeneralPath();
Path.moveTo(10, 20);
```

그리고, lineTo, quadTo, 또는 curveTo 메소드들중의 하나를 호출함으로써 패스를 확장한다. 이러한 메소드들은 라인, 이차곡선, 혹은 입체곡선에 의해 진로를 확장시킨다. LineTo 을 호출하면, 끝점을 제공한다. 두개의 곡선 메소드의 경우, 제어점들과 끝점을 준다. 예를 들면,

```
path.lineTo(20, 30);
path.curveTo(control1X, control1Y, control2X, control2Y, endX, endY);
```

closePath 메소드을 호출하여 패스를 끝낸다. 이는 마지막 moveTo까지 꺼꾸로 선을 그린다.
다각형을 만들기 위해서는, 다른 코너점을 방문하는 lineTo에 반복되는 호출을 따라하면서 간단히 moveTo 을 호출하고 첫번째 코너점으로 갑니다. 마지막으로, closePath을 호출해서 다각형을 닫습니다. 예제 7-1에 있는 프로그램은 이것을 자세히 알려준다.
일반적인 패스는 연결되어져 있지 않는다. 언제든지 moveTo을 호출해서 새로운 패스를 시작한다.

마침내, append 메소드를 이용해 임의의 Shape 객체를 패스에 붙인다. 형태의 아웃라인은 패스의 끝에 덧붙여진다. Append 메소드의 두번째 전달인자는 새로은 형태가 패스상의 마지막 포인트에 연결될 때 참(true)이 되고, 연결되지 않으면 거짓(false) 이 된다. 예를 들면, 호출,

```
Rectangle2D r = ...;
path.append(r, false);
```

는 기존 패스에 연결하지 않고 직사각형의 아웃라인을 패스에 덧붙인다. 그러나,

```
path.append(r, false);
```

는 패스의 끝점에서 직사각형의 시작점을 곧바로 연결하고, 직사각형의 아웃라인을 패스에 붙인다.

예제 7-1의 프로그램은 간단한 패스를 만들 수 있게 한다. 그림 7-8과 7-9는 예제로 프로그램을 실행한 것 이다. 콤보 박스에서 형태 만들기를 선택하다. 프로그램은

- 직선
- 직사각형, 둥근 사각형 그리고 타원형
- 호(호 자체 뿐만 아니라 바운딩 사각형과 시각과 끝 각도를 보여주는 선)
- 다각형(GeneralPath을 이용)

- 이차 및 입체 곡선

에 대한 형태 만들기를 포함하고 있다.

마우스를 이용해 제어점들을 조정한다. 그것들을 움직이면서, 형태는 스스로 계속적인 리페인트를 하다.
프로그램은 다소 복잡한데, 이는 형태의 다양성을 다루고 제어점의 드래깅을 제공하고 있기 때문이다.)
추상적인 상위 클래스인 ShapeMaker는 형태 생성기 클래스의 공통성을 함축하고 있다. 각 형태는 제어점을 고정된 수만큼 가지고, 사용자가 이리저리 움직이게 하다. getPointCount메소드는 그 값을 돌려준다. 추상적인 메소드

Shape makeShape(Point2D[] points)

는 실제 형태, 제어점들의 현재 위치들을 계산해 낸다. toString 메소드는 클래스 이름을 반환해서 ShapeMaker 객체가 간단히 JcomboBox로 점프될 수 있게 한다.
제어점의 드래깅을 가능하게 할려면, ShapePanel 클래스가 MouseListener와 MouseMotionListener 의 인터페이스를 구현한다. 마우스로 직사각형의 위를 누르면, 뒤따라 마우스 드래그가 직사각형을 움직임니다.

형태 생성자의 많은 부분이 간단한 것이다.-그들의 makeShape 메소드들은 단지 생성하고 요청된 형태를 반환해준다. 그러나, ArcMaker 클래스는 뒤틀린 시작 각도와 끝각도을 산출해야 한다. 더구나, 계산이 옳음을 설명할려면, 반환된 형태가 호의 중심에서부터 각도 제어 포인트까지의 호 자체, 바운딩 직사각형 그리고 선을 포함하는 GeneralPath이다. (그림 7-10)

그림7-10: ShapeTest 프로그램

예제 7-1: ShapeTest.java

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

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

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

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

        Container contentPane = getContentPane();

        panel = new ShapePanel();
        contentPane.add(panel, "Center");
        comboBox = new JComboBox();
        comboBox.addItem(new LineMaker());
        comboBox.addItem(new RectangleMaker());
        comboBox.addItem(new RoundRectangleMaker());
        comboBox.addItem(new EllipseMaker());
        comboBox.addItem(new ArcMaker());
        comboBox.addItem(new PolygonMaker());
        comboBox.addItem(new QuadCurveMaker());
        comboBox.addItem(new CubicCurveMaker());
        comboBox.addActionListener(this);
        contentPane.add(comboBox, "North");
    }

   public void actionPerformed(ActionEvent event)
   {  ShapeMaker shapeMaker =
         (ShapeMaker)comboBox.getSelectedItem();
```

```
        panel.setShapeMaker(shapeMaker);
    }

    private JComboBox comboBox;
    private ShapePanel panel;
}

class ShapePanel extends JPanel
    implements MouseListener, MouseMotionListener
{   public ShapePanel()
    {   addMouseListener(this);
        addMouseMotionListener(this);
        current = -1;
    }

    public void setShapeMaker(ShapeMaker aShapeMaker)
    {   shapeMaker = aShapeMaker;
        int n = shapeMaker.getPointCount();
        points = new Point2D[n];
        for (int i = 0; i < n; i++)
        {   double x = generator.nextDouble() * getWidth();
            double y = generator.nextDouble() * getHeight();
            points[i] = new Point2D.Double(x, y);
        }
        repaint();
    }

    public void paintComponent(Graphics g)
    {   super.paintComponent(g);
        if (points == null) return;
        Graphics2D g2 = (Graphics2D)g;
        for (int i = 0; i < points.length; i++)
        {   double x = points[i].getX() - SIZE / 2;
            double y = points[i].getY() - SIZE / 2;
            g2.fill(new Rectangle2D.Double(x, y, SIZE, SIZE));
        }
```

```
      g2.draw(shapeMaker.makeShape(points));
   }

public void mousePressed(MouseEvent event)
{   Point p = event.getPoint();
    for (int i = 0; i < points.length; i++)
    {   double x = points[i].getX() - SIZE / 2;
        double y = points[i].getY() - SIZE / 2;
        Rectangle2D r = new Rectangle2D.Double(x, y, SIZE, SIZE);
          if (r.contains(p))
          {   current = i;
              return;
          }
     }
   }

public void mouseReleased(MouseEvent event)
{   current = -1;
   }

public void mouseEntered(MouseEvent event)
   {
   }

public void mouseExited(MouseEvent event)
   {
   }

public void mouseClicked(MouseEvent event)
   {
   }

public void mouseMoved(MouseEvent event)
   {
   }
```

```
    public void mouseDragged(MouseEvent event)
    {   if (current == -1) return;
        points[current] = event.getPoint();
         repaint();
     }


    private Point2D[] points;
   private static Random generator = new Random();
   private static int SIZE = 10;
    private int current;
    private ShapeMaker shapeMaker;
}

abstract class ShapeMaker
{   public ShapeMaker(int aPointCount)
    {   pointCount = aPointCount;
     }

    public int getPointCount()
    {   return pointCount;
     }

   public abstract Shape makeShape(Point2D[] p);

    public String toString()
    {   return getClass().getName();
     }

   private int pointCount;
}

class LineMaker extends ShapeMaker
{    public LineMaker() { super(2); }
```

```
   public Shape makeShape(Point2D[] p)
   {   return new Line2D.Double(p[0], p[1]);
    }
}

class RectangleMaker extends ShapeMaker
{  public RectangleMaker() { super(2); }

   public Shape makeShape(Point2D[] p)
    {   Rectangle2D s = new Rectangle2D.Double();
        s.setFrameFromDiagonal(p[0], p[1]);
        return s;
    }
}

class RoundRectangleMaker extends ShapeMaker
{  public RoundRectangleMaker() { super(2); }

   public Shape makeShape(Point2D[] p)
    {   RoundRectangle2D s
          = new RoundRectangle2D.Double(0, 0, 0, 0, 20, 20);
        s.setFrameFromDiagonal(p[0], p[1]);
        return s;
    }
}

class EllipseMaker extends ShapeMaker
{  public EllipseMaker() { super(2); }

   public Shape makeShape(Point2D[] p)
    {   Ellipse2D s = new Ellipse2D.Double();
        s.setFrameFromDiagonal(p[0], p[1]);
        return s;
    }
}
```

```
class ArcMaker extends ShapeMaker
{  public ArcMaker() { super(4); }

   public Shape makeShape(Point2D[] p)
   {  double centerX = (p[0].getX() + p[1].getX()) / 2;
      double centerY = (p[0].getY() + p[1].getY()) / 2;
       double width = Math.abs(p[1].getX() - p[0].getX());
       double height = Math.abs(p[1].getY() - p[0].getY());

        double distortedStartAngle
            = Math.toDegrees(Math.atan2(-(p[2].getY() - centerY)
               * width, (p[2].getX() - centerX) * height));
        double distortedEndAngle
            = Math.toDegrees(Math.atan2(-(p[3].getY() - centerY)
               * width, (p[3].getX() - centerX) * height));
        double distortedAngleDifference
           = distortedEndAngle - distortedStartAngle;
       if (distortedStartAngle < 0)
           distortedStartAngle += 360;
       if (distortedAngleDifference < 0)
           distortedAngleDifference += 360;

      Arc2D s = new Arc2D.Double(0, 0, 0, 0,
            distortedStartAngle, distortedAngleDifference,
            Arc2D.OPEN);
       s.setFrameFromDiagonal(p[0], p[1]);

       GeneralPath g = new GeneralPath();
        g.append(s, false);
       Rectangle2D r = new Rectangle2D.Double();
         r.setFrameFromDiagonal(p[0], p[1]);
        g.append(r, false);
       Point2D center = new Point2D.Double(centerX, centerY);
       g.append(new Line2D.Double(center, p[2]), false);
       g.append(new Line2D.Double(center, p[3]), false);
        return g;
```

```
    }
}

class PolygonMaker extends ShapeMaker
{  public PolygonMaker() { super(6); }

   public Shape makeShape(Point2D[] p)
   {  GeneralPath s = new GeneralPath();
      s.moveTo((float)p[0].getX(), (float)p[0].getY());
      for (int i = 1; i < p.length; i++)
         s.lineTo((float)p[i].getX(), (float)p[i].getY());
      s.closePath();
      return s;
   }
}

class QuadCurveMaker extends ShapeMaker
{  public QuadCurveMaker() { super(3); }

   public Shape makeShape(Point2D[] p)
   {  return new QuadCurve2D.Double(p[0].getX(), p[0].getY(),
         p[1].getX(), p[1].getY(), p[2].getX(), p[2].getY());
   }
}

class CubicCurveMaker extends ShapeMaker
{  public CubicCurveMaker() { super(4); }

   public Shape makeShape(Point2D[] p)
   {  return new CubicCurve2D.Double(p[0].getX(), p[0].getY(),
         p[1].getX(), p[1].getY(), p[2].getX(), p[2].getY(),
         p[3].getX(), p[3].getY());
   }
}
```

java.awt.geom..RectanglarShape

- void setFrameCenter(double centerX, double centerY, double cornerX, double cornerY)
- void setFrameCenter(Point2D center, Point2D corner)

주어진 중심과 코너 포인터로 새로운 직사각형에 이 직사각형 형태의 바운딩 직사각형을 설정한다. 코너 포인트는 맨 위 왼쪽의 포인트일 필요는 없다.

- void setFrameFromDiagonal(double corner1X, double corner1Y, double corner2X, double corner2Y)
- void setFrameFromDiagonal(Point2D corner1, Point2D corner2)

주어진 diametrically **반대 코너 포인터를 가진 새로운 직사각형에 이 직사각형 형태의 바운딩 직사각형을 설정한다.**

java.awt.geom..RoundRectangle2D.Double

- RoundRectangle2D.Double(double x, double y, double w, double h, double arcWidth, double arcHeight)

주어진 바운딩 직사각형과 호 너비로 둥근 직사각형을 생성한다.

(영문 첫글자 대문자로 바뀌지 않게 하기)

전달인자:	x, y	**바운딩 직사각형의 맨 위 왼쪽 코너**
	w,h	**바운딩 직사각형의 너비와 높이**
	arcWidth	**중심에서 타원 바깥 호의 끝까지 수평 거리**
	arcHeight	**중심에서 타원 바깥 호의 끝까지 수직거리**

java.awt.geom..Ellipse2D.Double

- Ellipse2D.Double(double x, double y, double w, double h)

주어진 바운딩 직사각형으로 둥근 직사각형을 생성한다.

전달인자 :	x,y	**바운딩 직사각형의 맨 위 왼쪽 코너**
	w, h	**바운딩 직사각형의 너비와 높이**

java.awt.geom..Arc2D.Double

- Arc2D.Double(double x, double y, double w, double h, double startAngle, double arcAngle, int type)

주어진 바운딩 직사각형, 처음 및 호 각도 그리고 호 타입으로 호를 생성한다.

전달인자 :	x, y	**바운딩 직사각형의 맨 위 왼쪽 코너**
	w,h	**바운딩 직사각형의 너비와 높이**
	startAngle	**x축과 반지름내에서 호의 시각점을 가진 바운딩 직사각형의 중심과 인접한 라인과의 각도 측정치는 뒤틀린다. 그래서 π/4의 "angle"은 x축과 바운딩 직사각형의 중심과 위 오른쪽 코너에 인접한 라인사이의 각도와 동일하다..**
	arcAngle	**뒤틀린것과 시작각도 사이의 차이** – 470 **페이지의 사이**

드바를 참조하세요. 원형 호의 경우, 이 값은 호에 swept out된 각도와 동일하다.

type Arc2D.OPEN, Arc2D.PIE 그리고 Arc2D.CHORD중의 하나

java.awt.geom.QuadCurve2D.Double

- QuadCurve2D.Double(double x1, double y1, double ctrlx, double ctrly, double x2, double y2)

시작점, 제어점, 그리고 끝점에서 이차 곡선을 생성한다.

전달인자 : x1, y1 시작점

ctrlx, ctrly 제어점

x2, y2 끝점

java.awt.geom..CubicCurve2D.Double

- CubicCurve2D.Double(double x1, double y1, double ctrlx1, double ctrly1, double ctrlx2, double ctrly2, double x2, double y2)

시작점, 두개의 제어점 그리고 끝점에서 입체 곡선을 생성한다.

전달인자 : x1, y1 시작점

ctrlx1, ctrly1 첫번째 제어점

ctrlx2, ctrly2 두번째 제어점

x2, y2 끝점

java.awt.geom..GeneralPath

- GeneralPath()

빈 일반적인 패스를 생성한다.

- void moveTo(float x, float y)

다음 세크먼트의 시작점등 현 지점 (x,y)을 만든다.

- void lineTo(float x, float y)
- void quadTo(float ctrlx, float ctrly, float x, float y)
- void curveTo(float ctrlx, float ctrly, float ctrl2x, float ctrl2y, float x, float y)

현재 지점에서 끝점(x,y)까지 라인, 이차곡선 및 입체 곡선을 그리고 그 끝점을 현재 점으로 만든다.

- void append(Shape s, Boolean connect)

주어진 형태의 아웃라인에 일반적인 패스를 덧붙인다. Connect이 true이면, 일반적인 포인트의 현지점은 직선에 의해 덧붙여진 형태의 시작점과 연결된다.

- void closePath()

패스상의 현재 시점에서 첫번째 점까지 직선을 그리면서 패스를 끝낸다.

영역(area)

앞 장에서, 선과 곡선으로 이루어진 일반적인 패스를 생성해서 복잡한 형태를 구현하는가에 대해 살펴보았다. 충분한 선과 곡선을 사용해서, 필요한 어떤 형태도 그릴 수 있다. 예를 들면, 화면이나 프린트 된 상태에서 보여지는 폰트상의 문자 형태들은 모두 선과 입체 곡선으로 이루어져 있다.

그러나, 때때로 직사각형, 다각형 또는 타원 같은 영역(area)들로 구성해서 형태를 나타내는 것이 쉬울 때도 있다. Java 2D API 는 두개의 영역을 결합해서 새로운 하나의 영역을 생성 영역 기하배열(Constructive area geometry)을 제공한다.

- add – 결합된 영역은 첫번째 또는 두번째 영역안에 있는 모든 점들을 포함하다.
- subtract – 결합된 영역은 첫번째 영역에는 있지만 두번째 영역안에는 없는 모든 점들을 포함하다.
- intersect – 결합된 영역은 첫번째과 두번째 영역안에 있는 모든 점들을 포함하다.
- exclusiveOr – 결합된 영역은 첫번째 영역 또는 두번째 영역, 그러나 둘다는 아닌 영역안에 있는 모든 점들을 포함하다.

그림 7-11은 이러한 기능을 보여준다.

그림 7-11 : Constructive Area Geometry 기능

복잡한 영역을 생성하려면, 디폴트 영역 객체로 시작한다.

```
Area a = new Area();
```

그리고, 그 영역을 어떠한 형태와 결합한다.

```
a. add( new Rectangle2D.Double(….));
a. subtract(path);
```

Area 클래스는 Shape 인터페이스를 구현한다. Draw 메소드로 영역의 가장자리를 그을 수 있거나 Graphics2D 클래스의 fill 메소드로 내부를 채울수 있다.

예제 7-2의 프로그램은 constructive area geometry 기능을 보여준다. 네 가지 기능중 하나를 선택하고 선택한 기능으로 타원과 직사각형을 결합한 결과를 보자. (그림 7-12)

그림 7-12 : AreaTest .java

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

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

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

class AreaTestFrame extends JFrame
   implements ActionListener
{  public AreaTestFrame()
    {  setTitle("AreaTest");
       setSize(400, 400);
       addWindowListener(new WindowAdapter()
          {  public void windowClosing(WindowEvent e)
              {  System.exit(0);
              }
          } );

       Container contentPane = getContentPane();
       canvas = new AreaPanel();
       contentPane.add(canvas, "Center");

       JPanel buttonPanel = new JPanel();
       ButtonGroup group = new ButtonGroup();

         addButton = new JRadioButton("Add", true);
         buttonPanel.add(addButton);
         group.add(addButton);
         addButton.addActionListener(this);

       subtractButton = new JRadioButton("Subtract", false);
         buttonPanel.add(subtractButton);
         group.add(subtractButton);
```

```
        subtractButton.addActionListener(this);

      intersectButton = new JRadioButton("Intersect", false);
        buttonPanel.add(intersectButton);
        group.add(intersectButton);
        intersectButton.addActionListener(this);

        exclusiveOrButton = new JRadioButton("Exclusive Or", false);
        buttonPanel.add(exclusiveOrButton);
        group.add(exclusiveOrButton);
        exclusiveOrButton.addActionListener(this);

       contentPane.add(buttonPanel, "North");
    }

   public void actionPerformed(ActionEvent event)
   {  Object source = event.getSource();
      if (source == addButton)
            canvas.addAreas();
      else if (source == subtractButton)
            canvas.subtractAreas();
      else if (source == intersectButton)
            canvas.intersectAreas();
      else if (source == exclusiveOrButton)
            canvas.exclusiveOrAreas();
    }


   private AreaPanel canvas;
   private JRadioButton addButton;
   private JRadioButton subtractButton;
   private JRadioButton intersectButton;
   private JRadioButton exclusiveOrButton;
}

class AreaPanel extends JPanel
```

```
{   public AreaPanel()
    {   area1
          = new Area(new Ellipse2D.Double(100, 100, 150, 100));
        area2
          = new Area(new Rectangle2D.Double(150, 150, 150, 100));
        addAreas();
    }

   public void paintComponent(Graphics g)
    {   super.paintComponent(g);
       Graphics2D g2 = (Graphics2D)g;
        g2.draw(area1);
        g2.draw(area2);
        g2.fill(area);
    }

   public void addAreas()
   {   area = new Area();
        area.add(area1);
        area.add(area2);
        repaint();
    }

   public void subtractAreas()
   {   area = new Area();
        area.add(area1);
        area.subtract(area2);
        repaint();
    }

   public void intersectAreas()
   {   area = new Area();
        area.add(area1);
        area.intersect(area2);
        repaint();
    }
```

```
   public void exclusiveOrAreas()
   {  area = new Area();
        area.add(area1);
        area.exclusiveOr(area2);
        repaint();
    }

   private Area area;
   private Area area1;
    private Area area2;
}
```

java.awt.geom..Area

- void add(Area other)
- void subtract(Area other)
- void intersect(Area other)
- void exclusiveOr(Area other)

이 형태와 다른 형태로 constructive area geometry **기능을 수행하고 영역을 결과에 설정한다**.

스트로크

Graphics2D **클래스의** draw **기능은 현재 선택된** stroke**으로 형태의 가장자리를 그린다**. **디폴트로**, **스트로크는 고른 선으로 하나의 픽셀만 한다**. setStroke **메소드를 호출해서 다른 스트로크를 선택할 수 있다**. **클래스의 객체를 제공해서** Stroke **인터페이스를 구현한다**. Java2D API **는** BasicStroke**라 불리는 이러한 하나의 클래스를 정의한다**. **이번 장에서**, BasicStroke **클래스의 용도들을 알아본다**.

임의 굵은 스트로크를 생성한다. **예를 들면**, **여기서 어떻게** 10 **픽셀 크기의 라인을 그리는지를 보여준다**.

```
g2.setStroke(new BasicStroke(10.0F));
g2.draw(new Line2D.Double(..));
```

스트로크가 하나의 픽셀보다 두꺼울 때, **스트로크의 끝은 다른 스타일을 갖게 된다**. **그림** 7-13**은 이른바** end cap styles**을 보여준다**. **세가지의 선택이 있다**:

- butt cap **은 단순히 이 스트로크의 끝점에서 끝난다**.
- round cap**은 스트로크의 끝에 반원을 붙인 것이다**.
- square cap**은 스트로크의 끝에 반사각형을 붙인 것이다**.

그림7-13: End Cap Styles

두개의 두꺼운 스트로크가 만나며, join style에는 3개의 선택이 있다. (그림 7-14)

- bevel join은 두 스트로크사이 각도의 이등분선에 교차하는 직선이 있는 스트로크를 결합한다.
- round join은 각각의 스트로크를 펼쳐 round cap을 갖게 된다.
- miter join은 "spike"을 덧붙여 각각의 스트로크를 펼친다.

그림 7-14: Join Styles

miter join은 만나는 각도가 작은 선에서는 부적절하다. 두선 만났을 때의 각도가 miter limit 보다 낮으면, bevel join이 대신 쓰이다. 이러한 방법은 극히 긴 "spike"을 막기 위합이다. 디폴트로 miter limit은 10도이다.

예를 들어, BasicStroke 생성자안에 이러한 선택들을 명시한다.:

```
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 15.0F/*miter limit*/));
```

마지막으로, dash pattern을 설정해서 점괘선을 명시할 수 있다. 예제 7-3의 프로그램에서, 모스 코드의 SOS 판독하여 대시 패턴을 선택할 수 있다. 대시 패턴은 'on' 'off' 스트로크의 길이를 담고 있는 숫자로 float[ ]배열이다. (그림 7-15)

그림 7-15 : 대시 패턴

BasicStroke을 생성할 때 대시 패턴과 dash phase을 명시한다. Dash phase는 대시 패턴안에서 어디쯤에서 선이 시작하는지를 가리킵니다. 정상적으로는, 이 값을 0으로 설정한다.

```
float[ ] dash Pattern = { 10, 10, 10, 10, 10, 10, 30, 10, 30 ……..};
g2.setStroke(new BasicStroke(10.0F, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 10.0F /* miter limit**/, dashPattern, 0 /** dash phase */));
```

노트 : End cap 스타일은 대시 패턴내의 각 대시 끝에 적용된다.

예제 7-3의 프로그램은 end cap 스타일, join 스타일, 그리고 대시된 선을 나타낼 수 있게 한다. (그림 7-16). 선 마디의 끝을 움직여 miter limit을 테스트 한다: miter join을 선택하고, 선마디를 뽀족한 각도를 만든다. 그러면 miter join이 bevel join으로 바뀜을 볼 수 있다.

그림 7-16 : The StrokeTest 프로그램

이 프로그램은 예제 7-1의 프로그램과 유사하다. 선 마디의 끝점을 클릭했다면 마우스 리스너가 기억을 하고 있고, 마우스 모션 리스너가 끝점의 드래깅을 모니터한다. 라디오 버튼이 end cap style, join style, 굵은 또는 대시드된 선에 대한 사용자의 선택을 신호로 보낸다. StrokePanel 클래스의 paintComponent 메소드는 세 개의 점으로 조인되어 두 라인으로 구성된 GeneralPath을 생성해서 사용자가 마우스를 가지고 움직일 수 있게 한다. 그러면, 사용자의 선택에 따라서 BasicStroke을 생선하고 마침내는 패스를 그리게 된다.

예제 7-3 : StrokeTest.java

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

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

class StrokeTestFrame extends JFrame
   implements ActionListener
{  public StrokeTestFrame()
   {  setTitle("StrokeTest");
      setSize(400, 400);
      addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
```

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

 Container contentPane = getContentPane();
 canvas = new StrokePanel();
  contentPane.add(canvas, "Center");

JPanel buttonPanel = new JPanel();
 buttonPanel.setLayout(new GridLayout(3, 3));
ButtonGroup group1 = new ButtonGroup();

buttCapButton = new JRadioButton("Butt Cap", true);
  buttonPanel.add(buttCapButton);
  group1.add(buttCapButton);
  buttCapButton.addActionListener(this);

  roundCapButton = new JRadioButton("Round Cap", false);
  buttonPanel.add(roundCapButton);
  group1.add(roundCapButton);
  roundCapButton.addActionListener(this);

squareCapButton = new JRadioButton("Square Cap", false);
  buttonPanel.add(squareCapButton);
  group1.add(squareCapButton);
  squareCapButton.addActionListener(this);

ButtonGroup group2 = new ButtonGroup();

bevelJoinButton = new JRadioButton("Bevel Join", true);
  buttonPanel.add(bevelJoinButton);
  group2.add(bevelJoinButton);
  bevelJoinButton.addActionListener(this);

miterJoinButton = new JRadioButton("Miter Join", false);
  buttonPanel.add(miterJoinButton);
```

```
        group2.add(miterJoinButton);
        miterJoinButton.addActionListener(this);

      roundJoinButton = new JRadioButton("Round Join", false);
        buttonPanel.add(roundJoinButton);
        group2.add(roundJoinButton);
        roundJoinButton.addActionListener(this);

      ButtonGroup group3 = new ButtonGroup();

       solidLineButton  = new JRadioButton("Solid Line", true);
        buttonPanel.add(solidLineButton);
        group3.add(solidLineButton);
        solidLineButton.addActionListener(this);

      dashedLineButton = new JRadioButton("Dashed Line", false);
        buttonPanel.add(dashedLineButton);
        group3.add(dashedLineButton);
        dashedLineButton.addActionListener(this);

       contentPane.add(buttonPanel,  "North");
   }

  public void actionPerformed(ActionEvent event)
  {   Object source = event.getSource();
       if (source == buttCapButton)
            canvas.setCap(BasicStroke.CAP_BUTT);
      else if (source == roundCapButton)
            canvas.setCap(BasicStroke.CAP_ROUND);
      else if (source == squareCapButton)
            canvas.setCap(BasicStroke.CAP_SQUARE);
       else if (source == bevelJoinButton)
            canvas.setJoin(BasicStroke.JOIN_BEVEL);
      else if (source == miterJoinButton)
            canvas.setJoin(BasicStroke.JOIN_MITER);
      else if (source == roundJoinButton)
```

```
            canvas.setJoin(BasicStroke.JOIN_ROUND);
        else if (source == solidLineButton)
            canvas.setDash(false);
        else if (source == dashedLineButton)
            canvas.setDash(true);
    }

    private StrokePanel canvas;
    private JRadioButton buttCapButton;
    private JRadioButton roundCapButton;
    private JRadioButton squareCapButton;
    private JRadioButton bevelJoinButton;
    private JRadioButton miterJoinButton;
    private JRadioButton roundJoinButton;
    private JRadioButton solidLineButton;
    private JRadioButton dashedLineButton;
}

class StrokePanel extends JPanel
    implements MouseListener, MouseMotionListener
{   public StrokePanel()
    {   addMouseListener(this);
        addMouseMotionListener(this);
        points = new Point2D[3];
        points[0] = new Point2D.Double(200, 100);
        points[1] = new Point2D.Double(100, 200);
        points[2] = new Point2D.Double(200, 200);
        current = -1;
        width = 10.0F;
    }

    public void paintComponent(Graphics g)
    {   super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        GeneralPath path = new GeneralPath();
        path.moveTo((float)points[0].getX(),
```

```
            (float)points[0].getY());
    for (int i = 1; i < points.length; i++)
            path.lineTo((float)points[i].getX(),
                (float)points[i].getY());
        BasicStroke stroke;
        if (dash)
        {   float miterLimit = 10.0F;
          float[] dashPattern = { 10F, 10F, 10F, 10F, 10F, 10F,
              30F, 10F, 30F, 10F, 30F, 10F,
              10F, 10F, 10F, 10F, 10F, 30F };
            float dashPhase = 0;
            stroke = new BasicStroke(width, cap, join,
                miterLimit, dashPattern, dashPhase);
        }
        else
          stroke = new BasicStroke(width, cap, join);
        g2.setStroke(stroke);
        g2.draw(path);
    }

public void setJoin(int j) { join = j; repaint(); }

public void setCap(int c) { cap = c; repaint(); }

public void setDash(boolean d) { dash = d; repaint(); }

    public void mousePressed(MouseEvent event)
    {   Point p = event.getPoint();
        for (int i = 0; i < points.length; i++)
        {   double x = points[i].getX() - SIZE / 2;
            double y = points[i].getY() - SIZE / 2;
            Rectangle2D r = new Rectangle2D.Double(x, y, SIZE, SIZE);
            if (r.contains(p))
            {   current = i;
                return;
            }
```

```
      }
   }

public void mouseReleased(MouseEvent event)
{   current = -1;
 }

public void mouseEntered(MouseEvent event)
 {
 }

public void mouseExited(MouseEvent event)
 {
 }

public void mouseClicked(MouseEvent event)
 {
 }

public void mouseMoved(MouseEvent event)
 {
 }

public void mouseDragged(MouseEvent event)
{   if (current == -1) return;
    points[current] = event.getPoint();
     repaint();
 }

private Point2D[] points;
private static int SIZE = 10;
private int current;
private float width;
private int cap;
private int join;
private boolean dash;
```

}

java.awt.Graphics2D

- void setStroke(Stroke s)
  Stroke 인터페이스를 구현하는 주어진 객체에 이 그래픽 컨텍스의 스트로크를 설정한다.

java.awt.geom..BasicStroke

- BasicStroke(float width)
- BasicStroke(float width, int cap, int join)
- BasicStroke(float width, int cap, int join, float miterlimit)
- BasicStroke(float width, int cap, int join, float miterlimit, float [ ] dash, float dashphase)

주어진 특징의 스트로크 객체를 생성한다.

전달인자 :	width	펜의 두께
	cap	CAP_BUTT, CAP_ROUND, CAP_SQUARE 중의 하나인 end cap style
	join	JOIN_REVEL, JOIN_MITER, JOIN_ROUND의 하나인 JOIN style
	miterlimit	bevel join으로 랜더리된 miter join 이하의 정도에서의 각도
	dash	대신 채워진 길이와 대시된 스트로크이 공백에 해당하는 배열
	dashPhase	대시 패턴의 '상'. 스트로크의 출발점보다 앞서는 이 길이의 부분은 이미 적용된 대시 패턴이 있음을 가정한다.

페인트

형태를 채울 때, 내부는 paint로 덮인다. Paint 인터페이스를 구현하고 있는 클래스의 객체에 패인트 스타일을 설정하려면, setPaint를 사용한다. Java 2D API에서, 3가지의 이러한 클래스가 있다:

- Color 클래스는 Paint 인터페이스를 구현한다. Solid 선으로 형태를 채우려면
  g2.setPaint(Color.red);
  같은 Color 객체를 가진 setPaint를 호출한다.
- GradientPaint 클래스는 두개의 주어진 색상 값을 새로 만들어 색상을 다양화한다.(그림 7-17)
- TexturePaint 클래스는 계속적인 이미지로 영역을 채운다. (그림 7-18)

그림 7-17 : Gradient paint

그림 7-18 : Texture paint

이러한 두 점에서 주어지는 두개의 점과 색상들로 원하는 GradientPaint 객체를 생성한다.

```
g2.setPaint( new GradientPaint(p1, Color.red, p2, Color.blue));
```

색상은 두 점에 인접한 선 사이에서 새로워 진다. 색상은 이러한 인접한 선에 수직한 선들 사이에서는 일정하다. 선의 끝점을 벗어난 점들에게는 끝점에서의 색상이 주어진다.
선택적으로 cyclic 전달인자에 참인 GradientPaint 생성자를 호출하면,

```
g2.setPaint(new GradientPaint(p1, Color.red, p2, Color.blue, true));
```

색상 변화는 순환하면서 끝점을 넘어서까지 계속적인 변화를 계속한다.

TexturePaint 객체를 생성하려면, BufferedImage 와 anchor 직사각형을 명시할 필요가 있다. Anchor Rectangle은 x와 y 방향으로 무한히 확대되어 전체 좌표면에 붙여준다. 이미지는 anchor에 적당한 크기로 맞추어지고 각 타일에 복사된다.

뒷 장에서 이미지에 대해서 자세히 설명할 때 BufferedImage 클래스를 소개하고자 한다. 이미지 크기와 이미지 타입을 제공하여 BufferedImage 객체를 만든다. 가장 일반적인 이미지 타입은 TYPE_INT_ARGB이고, 각 픽셀은 alpha 도는 투명성, RGB 값을 나타내는 정수에 의해 나타난다. 예를 들면,

```
BufferedImage bufferedImage = new BufferedImage(width, height, TYPE_INT_ARGB);
```

그리고, 그래픽 컨텍스을 구해 버퍼드된 이미지에 그릴 수 있다.

```
Graphics2D g2 = bufferedImage.createGraphics();
```

이제 g2 에서의 모든 그리기 연산은 버퍼드 이미지를 픽셀로 채운다. 여러분이 작업을 마쳤을 때, TextPaint 객체를 생성할 수 있다.:

```
g2.setPaint(new TexturePaint(bufferedImage, anchorRectangle));
```

예제 7-4의 프로그램은 사용자가 solid color paint, gradient paint, texture paint 중에서 선택할 수 있게 한다,

그러면, 타원형은 지정된 페인트로 채워진다. Texture paint는 GIF 파일에서 이미지를 읽는다. 이미지 파일을 읽기 위해서는 미디어 추적기가 필요하다는 것을 1권의 7장에서 알 수 있다.

```
Image image = ToolKit.getDefaultToolkit().getImage ("blue-ball.gif");
MediaTracker tracker = new MediaTracker(this);
tracker.addImage(image, 0);
try { tracker.waitForID(0); }
catch( InterruptedException e) { }
```

이미지를 포착하게 되면, BufferedImage 안에 그릴 수 있다:

```
bufferdImage = new BufferedImage(image.getWidth(null), image.getHeight(null), BufferedImage.TYPE_INT_ARGB);
Graphics2D g2 = bufferedImage.createGraphics();
g2.drawImage(image, 0, 0, null);
```

anchor 직사각형의 의미를 보여주고자, 우리는 이미지 크기를 두 배로 확대한 anchor를 명시한다.

```
Rectangle2D anchor = new Rectangle2D.Double(0, 0, 2 * bufferedImage.getWidth(), 2 * bufferedImage.getHeight());
paint = new TexturePaint(bufferedImage, anchor);
```

'Texture Paint'를 선택한 때를 알 수 있듯이, 이미지는 anchor에 맞게 크기가 조정된다. 그리고 복사되어 형태를 채우고, 형태의 가장자리에서 만나는 타일은 잘린다.

예제 7-4 : PaintTest.java

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

public class PaintTest
```

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

class PaintTestFrame extends JFrame
   implements ActionListener
{  public PaintTestFrame()
    {  setTitle("PaintTest");
       setSize(400, 400);
       addWindowListener(new WindowAdapter()
          {  public void windowClosing(WindowEvent e)
             {  System.exit(0);
             }
          } );

       Container contentPane = getContentPane();
       canvas = new PaintPanel();
       contentPane.add(canvas, "Center");

       JPanel buttonPanel = new JPanel();
       ButtonGroup group = new ButtonGroup();

       colorButton = new JRadioButton("Color", true);
        buttonPanel.add(colorButton);
        group.add(colorButton);
        colorButton.addActionListener(this);

        gradientPaintButton
          = new JRadioButton("Gradient Paint", false);
        buttonPanel.add(gradientPaintButton);
        group.add(gradientPaintButton);
        gradientPaintButton.addActionListener(this);

        texturePaintButton
```

```
            = new JRadioButton("Texture Paint", false);
        buttonPanel.add(texturePaintButton);
        group.add(texturePaintButton);
        texturePaintButton.addActionListener(this);

        contentPane.add(buttonPanel, "North");
    }

    public void actionPerformed(ActionEvent event)
    {   Object source = event.getSource();
        if (source == colorButton)
            canvas.setColor();
        else if (source == gradientPaintButton)
            canvas.setGradientPaint();
        else if (source == texturePaintButton)
            canvas.setTexturePaint();
    }

    private PaintPanel canvas;
    private JRadioButton colorButton;
    private JRadioButton gradientPaintButton;
    private JRadioButton texturePaintButton;
}

class PaintPanel extends JPanel
{   public PaintPanel()
    {   Image image = Toolkit.getDefaultToolkit().getImage
            ("blue-ball.gif");
        MediaTracker tracker = new MediaTracker(this);
        tracker.addImage(image, 0);
        try { tracker.waitForID(0); }
        catch (InterruptedException e) {}
        bufferedImage = new BufferedImage(image.getWidth(null),
            image.getHeight(null), BufferedImage.TYPE_INT_ARGB);
        Graphics2D g2 = bufferedImage.createGraphics();
        g2.drawImage(image, 0, 0, null);
```

```
        setColor();
    }

    public void paintComponent(Graphics g)
    {   super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.setPaint(paint);
        Ellipse2D circle
          = new Ellipse2D.Double(0, 0, getWidth(), getHeight());
        g2.fill(circle);
    }

    public void setColor()
    {  paint = Color.red; // Color implements Paint
        repaint();
    }

    public void setGradientPaint()
    {  paint = new GradientPaint(0, 0, Color.red,
            (float)getWidth(), (float)getHeight(), Color.blue);
        repaint();
    }

    public void setTexturePaint()
    {  Rectangle2D anchor = new Rectangle2D.Double(0, 0,
            2 * bufferedImage.getWidth(),
            2 * bufferedImage.getHeight());
       paint = new TexturePaint(bufferedImage, anchor);
        repaint();
    }

    private Paint paint;
    private BufferedImage bufferedImage;
}
```

java.awt.Graphics2D

- void setPaint(Paint s)

  Paint 인터페이스을 구현하는 주어진 객체에 대한 그래픽 컨텍스의 페인트를 설정한다.

java.awt.geom..GradientPaint

- GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2)
- GradientPaint(float x1, float y1, Color color1, float x2, float y2, Color color2, Boolean cyclic)
- GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2)
- GradientPaint(Point2D p1, Color color1, Point2D p2, Color color2, Boolean cyclic)

출발점이 color1로, 끝점이 color2로 색칠되고, 이 사이의 색상은 1차적으로 새로 생성된 이러한 색상으로 형태를 채운 gradient paint 객체를 생성한다. 색상들은 출발점과 끝점이 조인되는 라인과 직각을 이루는 선들과 함께 한다. 기본적으로, gradient paint는 주기적이지 않다. 즉, 시작점과 끝점을 넘는 점들은 출발점과 끝점과 같이 같은 색상으로 칠해진다. Gradient paint가 주기적이면, 색상은 계속적으로 새로 변하며 처음 출발점으로 반환되고는 무한히 양방향으로 순환을 계속한다.

전달인자 :	x1, y1, or p1	시작점
	color1	출발점에서 쓰인 색상
	x2, y2, or p2	끝점
	color2	끝점에 쓰인 색상
	cyclic	색상 변화가 계속되면 true, 출발점과 끝점을 넘은 색상이 일정하면 false

java.awt.geom..TexturePaint

- TexturePaint(BufferedImage texture, Rectangle2D anchor)

  texture paint 객체를 생성한다.

전달인자 :	texture	형태를 채울 때 쓰인 texture
	anchor	색칠된 공간의 타일링iling)를 정의하는 anchor 사각형. 사각형은 x와 y방향으로 무한정 반복되고 texture 이미지는 각 타일을 채우도록 스케일된다.

좌표 변환

자동차와 같은 물체를 그려야 할 필요가 있다고 가정하자. 제작자의 명세표에서, 높이, 앞뒤 축 사이의 거리, 총 길이 등을 알 수 있다. 물론, 미터당 몇 픽셀인지 가정하고 총 픽셀 위치를 파악할 수 있다. 그러나, 더욱 쉬운 방법이 있다. 그래픽 컨텍스를 변환해주는 것이다.

```
g2.scale(pixelsPerMeter, pixelsPerMeter);
```

```
g2.draw(new Line2D.Double (coordinates in meters));
        // 픽셀로 바꾸고 크기만한 선을 긋는다.
```

Graphics2D 클래스의 scale 메소드는 그래픽 컨텍스의 좌표변환으로 크기변환을 설정한다. 이러한 변환은 사용자 좌표(사용자가 명시한 단위)에서 디바이스 좌표(픽셀)로 바꿔준다. 그림 7-19는 변환이 어떻게 일어나는지 보여준다.

좌표변환은 실제적으로 매우 유용하다. 이 변환은 편리한 좌표값으로 작업하게 해준다. 그래픽 컨텍스는 좌표변환을 픽셀로 바꿔주는 불편한 부분에 까지 신경을 써 준다.

그림 7-19 : 사용자 및 디바이스 좌표

4가지의 기본적인 변환이 있다.

- Scaling : 고정된 점에서부터 거리까지 위로 늘리고,수축한다.
- Rotation : 고정된 중심 주위의 모든 점을 돌린다.
- Translation : 고정된 픽셀로 모든 점들을 옮긴다.
- Shear : 한 라인을 고정시켜놓고, 고정된 선에서 거리가 일정한 픽셀들에 의해 "미끄러지듯(sliding)" 선을 수평하게 한다.

그림 7-20은 시각형에서 이러한 4가지 기본적인 변형을 동작하는 것을 보여준다.

그림 7-20 : The fundamental transformations

Graphics2D 클래스의 scale, rotate, translate, shear 메소드들은 그래픽 컨텍스의 좌표변환을 이러한 기본적인 변환의 하나로 설정한다.

변경을 구성할 수 있다. 예를 들면, 형태를 돌리고 크기를 두배로 하고자 할 때, 회전과 크기 변형을 해 주어야 한다.

```
g2.rotate(angle);
g2.scale(2, 2);
g2.draw(..);
```

이 경우, 여러분이 제공하는 변형 순서는 문제가 되지 않는다. 그러나, 대부분의 변형의 경

우 순서가 영향을 미친다. 예를 들면, 회전을 시키고 전단변형을 하고자 한다면, 처음 변형하는 것이 무언가에 따라 차이가 난다. 의도가 무엇인지를 파악해야 한다. 그래픽 컨텍스는 사용자가 변형을 하는 순서 그대로 변형을 적용한다.

원하는 만큼 많은 변형을 해보도록 하겠다. 예를 들면, 변형의 다음 순서를 생각해 보자.

```
g2.translate(-x, -y);
g2.rotate(a);
g2.translate(x, y);
```

첫번째 변형은 (x, y)의 위치에서 원래의 위치로 움직이다. 두번째 변형은 원형을 a 각도 만큼 회전시킨다. 세번째 변형은 되돌아간 곳에서 (x, y)로 움직이다. 전체적인 영향은 (x, y)라는 중심점을 가지고 회전하는 것이다. (그림 7-21) 원 위치가 아닌 다른 점에서 회전하는 것은 흔한 동작으로, 보다 빠른 방법이 있다:

```
g2.rotate(a, x, y);
```

사용자가 행렬(matrix)의 이론을 안다면, 모든 회전이나 옮김, 크기 변형, 전단변형, 이들의 합성이 행렬의 형태 변형으로 나타날 수 있다는 것을 인지했을 것이다.

그림 7-21 : 변형의 합성

이런 변형을 affine (관계) 변형이라 불린다. Java 2D API에서, AffineTransform 클래스는 이러한 변형을 기술한다. 특정 변형 행렬의 컴포넌트를 알고 있다면, 다음과 같이 바로 생성 할 수 있다.

```
AffineTransform t = new AffineTransform(a, b, c, d, e, f);
```

또한 행렬을 생성하는 getRotateInstance, getScaleInstance, getTranslateInsatnce, getShearInstance 가 있어 이러한 변형 형태를 나타낸다. 예를 들면, 호출

```
t = AffineTransform.getScaleInstance(2.0F, 0.5F);
```

은 변형되어 아래의 행렬과 상응된다.

(행렬 표시 부탁합니다.)

마지막으로, setToTotation, setToScale, setToTranslation, setToToShear 의 인스턴스 메소드가 변형 객체에 새로운 타입으로 설정한다. 예를 들면,

t.setToRotation(angle); // t에 회전을 설정한다.

그래픽 컨텍스의 좌표변환에 AffineTransform 객체를 설정한다.

g2.setTransform(t); // 현재의 변형을 대신하다.

그러나, 실제적으로 여러분은 그래픽 구문가 가지고 있는 모든 기존의 잘린 형태를 대체하기 때문에 setTransform 연산을 호출하고 싶지 않다. 예를 들면, 배경모드에서 출력에 대한 그래픽 컨텍스는 이미 90 도의 회전 변형을 했다. setTransform을 호출하면, 그 회전을 지우게 된다. 대신에 transform 메소드를 호출한다.

g2.transform(t); // t로 현재 변형을 구성한다.

새로운 AffineTransform 객체를 지닌 기존의 변형을 구성한다.

잠시동안만 변형을 적용하고자 한다면, 먼저 기존의 변형을 보존하고 새로운 변형으로 구성 후 마침내 기존의 변형을 되돌린다.

```
AffineTransform oldTransform = g2.getTransform();
        // 기존 변형을 저장한다.
g2.transform(t) ;  // 일시적인 변형을 적용하다.
        // now draw on g2
g2.setTransform(oldTransform) ;  // 기존의 변형을 되돌린다.
```

예제 7-5의 프로그램은 사용자가 4개의 기본적인 변형 중에서 선택할 수 있게 한다. paintComponent 메소드는 사각형을 그리고, 선택한 변형을 적용하고 사각형을 다시 그린다. 그러나, 보기 좋게 하기 위해, 사각형을 갖고 디스 플레이 패널의 중심에 이 변형이 나타나게 한다. 여러가지 이유로, paintComponent 메소드는 좌표변형으로 변환을 설정한다.

```
g2.translate(getWidth() / 2, getHeight() / 2);
```

이 변환은 원위치를 컴포넌트의 중심으로 움직이다.

그러면, paintComponent 메소드는 사각형을 그려 원 위치 주위의 중심에 위치시킨다.

```
square = new Rectangle2D.Double(-50, -50, 100, 100);
. . .
g2.setPaint(Color.gray);
g2.draw(square);
```

그러나, 그래픽 컨텍스가 형태에 전환을 적용하기 때문에 시각형은 실제적으로 컴포넌트의

중심에 위치하면서 그 자신의 중심을 갖고 그려진다.

다음은, 사용자가 선택한 변형이 현재의 변형에 형성되고 사각형은 다시 한번 그려진다.

```
g2.transform(t);
g2.setPaint(Color.black);
g2.draw(square) ;
```

원래 사각형은 회색으로 그려지고, 변형된 것은 검은색으로 그려진다. (그림7-22)

그림 7-22 : The transformTest 프로그램

예제 7-5 : TransformTest.java

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


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

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

```
        } );

    Container contentPane = getContentPane();
    canvas = new TransformPanel();
     contentPane.add(canvas, "Center");

    JPanel buttonPanel = new JPanel();
    ButtonGroup group = new ButtonGroup();

    rotateButton = new JRadioButton("Rotate", true);
      buttonPanel.add(rotateButton);
      group.add(rotateButton);
      rotateButton.addActionListener(this);

     translateButton = new JRadioButton("Translate", false);
      buttonPanel.add(translateButton);
      group.add(translateButton);
      translateButton.addActionListener(this);

    scaleButton = new JRadioButton("Scale", false);
      buttonPanel.add(scaleButton);
      group.add(scaleButton);
      scaleButton.addActionListener(this);

    shearButton = new JRadioButton("Shear", false);
      buttonPanel.add(shearButton);
      group.add(shearButton);
      shearButton.addActionListener(this);

     contentPane.add(buttonPanel, "North");
 }

public void actionPerformed(ActionEvent event)
{   Object source = event.getSource();
    if (source == rotateButton) canvas.setRotate();
    else if (source == translateButton) canvas.setTranslate();
```

```
        else if (source == scaleButton) canvas.setScale();
        else if (source == shearButton) canvas.setShear();
    }

    private TransformPanel canvas;
    private JRadioButton rotateButton;
    private JRadioButton translateButton;
    private JRadioButton scaleButton;
    private JRadioButton shearButton;
}

class TransformPanel extends JPanel
{   public TransformPanel()
    {   square = new Rectangle2D.Double(-50, -50, 100, 100);
        t = new AffineTransform();
        setRotate();
    }

    public void paintComponent(Graphics g)
    {   super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.translate(getWidth() / 2, getHeight() / 2);
        g2.setPaint(Color.gray);
        g2.draw(square);
        g2.transform(t);
        /* we don't use setTransform because we want
              to compose with the current translation
        */
        g2.setPaint(Color.black);
        g2.draw(square);
    }

    public void setRotate()
    {   t.setToRotation(Math.toRadians(30));
        repaint();
    }
```

```
    public void setTranslate()
    {   t.setToTranslation(20, 15);
        repaint();
    }

    public void setScale()
    {   t.setToScale(2.0, 1.5);
        repaint();
    }

    public void setShear()
    {   t.setToShear(-0.2, 0);
        repaint();
    }

    private Rectangle2D square;
    private AffineTransform t;
}
```

java.awt.geom..AffineTransform

- AffineTransform(double a, double b, double c, double d, double e, double f)
- AffineTransform(float a, float b, float c, float d, float e, float f)

행렬을 가진 affine **변형을 생성한다**.

(행렬 좀 그려주세요?)

- AffineTransform(double [ ] m)
- AffineTransform(float [ ] m)

행렬을 가진 affine **변형을 생성한다**.

(행렬 좀 그려주세요?)

- static AffineTransform getRotateInstance(double a)

a**라는 각도로 원위치를 시계반대방향으로 회전시킨다. 변형 행렬은 다음과 같다**.

(행렬 좀 그려주세요?)

- static AffineTransform getRotateInstance(double a, double x, double y)

a라는 각도로 (x, y) 위치 둘레로 시계반대방향 회전시킨다

- static AffineTransform getScaleInstance(double sx, double sy)

x축으로 sx만큼, y축으로 sy만큼 변경하는 크기 변형을 한다. 변형 행렬은 다음과 같다.

(행렬 좀 그려주세요?)

- static AffineTransform getShearInstance(double shx, double shy)

x축에서 shx만큼, y축에서 shy만큼 전단해서 전단변형을 한다. 변형 행렬은 다음과 같다.

(행렬 좀 그려주세요?)

- static AffineTransform getTranslateInstance(double tx, double ty)

x축에서 tx만큼, y축에서 ty만큼 전환해서 움직인다. 전환 행렬은 다음과 같다.

(행렬 좀 그려주세요?)

- void setToRotation(double a)
- void setToRotation(double a, double x, double y)
- void setToScale(double sx, double sy)
- void setToShear(double sx, double sy)
- void setToTranslation(double tx, double ty)

주어진 전달인자를 가진 기본적인 변형에 affine transformation을 설정한다. 기본적인 변형과 그의 전달인자의 설명에 대해서는 getXxxInstance 메소드을 참조하라.

java.awt.Graphics2D

- void setTransform(AffineTransform t)

그래픽 컨텍스의 좌표변환을 t로 대체한다.

- void transform(AffineTransform t)

해당 그래픽 컨텍스의 기존 좌표변환에 t을 구성시킨다.

- void rotate(double a)
- void rotate(rotate a, double x, double y)
- void scale(double sx, double sy)
- void shear(double sx, double sy)
- void translate(double tx, double ty)

주어진 전달인자로 그래픽 컨텍스의 기존 좌표변환에 기본적인 변형을 한다. 기본적인 변

형과 전달인자들에 대한 설명은 AffineTransform을 참고하라.

클리핑

그래픽 컨텍스에서 잘린 형태를 설정함으로써, 그리기 작업중에 잘려진 형태의 내부에 대한 모든 그리기 연산을 제약할 수 있다.

```
g2.setClip(clipShape); // 밑을보세요
g2.draw(shape);
        //잘린 부분의 내부에 떨러진 부분만 그린다.
```

그러나 실제, 그래픽 컨텍스가 가지고 있는 기존의 모든 잘린 형태을 대체하기 때문에 여러분은 setClip 작업을 원하지 않는다. 예를 들면, 이장에 후반에서 보겠지만, 프린팅시에 그래픽 컨텍스는 가장자리를 그리지 않는다는 것을 알수 있는 clip rectangle이 온다. 대신에 clip 메소드를 호출한다.

```
g2.clip(clipShape); // better
```

clip 메소드는 기존의 클리핑 형태에 제공한 새로운 것을 교차시킨다.
일시적으로 클리핑 영역을 적용시키고자 하면, 예전 clip을 저장하고 새로운 clip을 적용, 작업이 끝나면 예전 것을 되돌려 가져온다.

```
Shape oldClip = g2.getClip() ;  // 기존 clip을 저장한다.
g2.clip(ClipShape) ; // 일시적인 clip을 적용한다.
        // g2에 새로 그린다.
g2.setClip(oldClip) ;  // 기존 clip으로 되돌린다.
```

예제 7-6에서, 우리는 복잡한 형태, 즉 일련의 문자들의 아웃라인처럼 복잡한 형태에 의해서 잘려진 라인 패턴을 그리는 클리핑 기능을 보여준다.(그림 7-23 참조)

그림 7-23 : ClipTest 프로그램

문자 아웃라인을 구할려면, font render context가 필요하다. Graphics2D 클래스의 getFontRenderContext 메소드을 이용한다.

```
FontRenderContext context = g2.getFontRenderContext() ;
```

다음은,

- A string
- A font
- The font render context

로부터

```
textLayout layout = newTextLayout("Hello", font, context);
```

TextLayout 객체를 생성한다.

이 텍스트 레이아웃 객체는 특정한 폰트 랜더 구문에 의해 만들어진 문자들의 순서의 레이아웃을 기술한다. 레이아웃은 랜더 구문에 따라 달라진다 – 같은 문자들도 스크린이나 프린터위에 다르게 나타날 수 있다.

TextLayout 클래스의 메소드들

```
getAdvance()
getAscent()
getDescent()
getLeading()
```

은 문자의 수평, 수직 길이을 기술하다 (그림 7-24). 이러한 메소드들은 1권의 7장에서 보았던 FontMetrics 클래스의 방법을 대신한다. FontMetrics 클래스는 정수 측정치를 돌려주는데, 측정치가 small point 길이에 흔한 fractional(분수) 값이면 불충분하다.

그림7-24 : Text layout measurements

현재 애플리케이션에서 보다 중요한 것은, getOutLine 메소드가 텍스트 레이아웃에서 문자들의 아웃라인 형태를 명시하는 Shape 객체를 돌려준다는 것이다. 아웃라인 형태는 원래(0,0)에서 출발하는데, 이는 대부분의 그리기 작업에 부적절하다. 그래서, getOutLine작업에 affine 변형을 해서 아웃라인이 나타날 곳을 명시하는 것이 필요하다.
단순히 변형을 주면 기준점이 (0, 100)으로 이동하다.

```
AffineTransform transform
        = AffineTransform.getTranslateInstance(0, 100) ;
Shape outline = layout.getOutline(transform) ;
```

그러면, 아웃라인을 클리핑 형태에 덧붙인다.

```
GeneralPath clipShape = new GeneralPath() ;
clipShape.append(outline, false) ;
```

결국, 클리핑 형태를 설정하고 선들을 그린다. 선들은 문자 경계선 내부에서만 나타난다.

```
g2.setClip(clipShape)
Point2D p = new Point2D.Doubt(0, 0) ;
for (int i = 0; i < NLINES ; i ++)
{       double x =  . . .;
        double y = . . .  ;
```

```
            Point2D q = new Point2D.Double(x, y) ;
            g2.draw(new Line2D.Double(p, q)) ; // lines are clipped
        }
```

완전한 프로그램은 아래와 같다.

예제 7-6 : ClipTest.java

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

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

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

      Container contentPane = getContentPane();
      canvas = new ClipPanel();
      contentPane.add(canvas, "Center");
```

```
        JPanel buttonPanel = new JPanel();
        ButtonGroup group = new ButtonGroup();

        noClipButton = new JRadioButton("No Clip", true);
          buttonPanel.add(noClipButton);
          group.add(noClipButton);
          noClipButton.addActionListener(this);

        clipButton = new JRadioButton("Clip", false);
          buttonPanel.add(clipButton);
          group.add(clipButton);
          clipButton.addActionListener(this);

         contentPane.add(buttonPanel, "North");
    }

    public void actionPerformed(ActionEvent event)
    {   Object source = event.getSource();
        if (source == clipButton) canvas.setClip(true);
        else if (source == noClipButton) canvas.setClip(false);
    }

    private JRadioButton clipButton;
    private JRadioButton noClipButton;

    private ClipPanel canvas;
}

class ClipPanel extends JPanel
{   public void paintComponent(Graphics g)
    {   super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;

        if (clip)
        {   FontRenderContext context = g2.getFontRenderContext();
```

```
        Font f = new Font("Serif", Font.PLAIN, 100);
        GeneralPath clipShape = new GeneralPath();

        TextLayout layout = new TextLayout("Hello", f, context);
        AffineTransform transform
            = AffineTransform.getTranslateInstance(0, 100);
        Shape outline = layout.getOutline(transform);
        clipShape.append(outline, false);

        layout = new TextLayout("World", f, context);
        transform
            = AffineTransform.getTranslateInstance(0, 200);
        outline = layout.getOutline(transform);
        clipShape.append(outline, false);

        g2.draw(clipShape);
        g2.clip(clipShape);
      }

    final int NLINES =50;
    Point2D p = new Point2D.Double(0, 0);
    for (int i = 0; i < NLINES; i++)
    {   double x = (2 * getWidth() * i) / NLINES;
        double y = (2 * getHeight() * (NLINES - 1 - i))
            / NLINES;
        Point2D q = new Point2D.Double(x, y);
        g2.draw(new Line2D.Double(p, q));
      }
  }

public void setClip(boolean c)
{   clip = c;
    repaint();
  }

private boolean clip;
```

}

java.awt.Graphics

- Shape getClip()

현재의 클리핑 형태을 돌려준다.

java.awt.Graphics2D

- void clip(Shape s)

현재의 클리핑 형태에 형태 s을 교차시킨다.

- void setClip(Shape s)

현재의 클리핑 형태를 형태 s에 설정한다.

- FontRenderContext getFontRenderContext()

TextLayout **객체를 생성시키는데 필요한 폰트 랜더 컨텍스를 반환한다.**

java.awt.font.TextLayout

- TextLayout(String s, Font f, FontRenderContext context)

특정 디바이스에 대한 폰트 특성을 얻기위해 폰트 랜터 컨텍스를 이용해 주어진 스트링과 폰트로부터 텍스트 레이아웃 객체를 생성한다.

- float getAdvance ()

해당 텍스트 레이아웃의 너비를 반환한다.

- float getAscent( )
- floatgetDescent ( )

기준선 위와 아래에 있는 텍스트 레이아웃의 높이를 반환한다.

- float getLeading ( )

텍스트 레이아웃에서 쓰인 폰트내의 연속선사이의 거리를 반환한다.

투명도와 컴포지션

표준 RGB **색상 모델에서, 각 색상은** red, green, blue **컴포넌트로 설명된다. 투명하거나 부분적으로 투명한 영역을 설명할 때 편리할 수 있다. 그러나, 기존 그림에 새로운 그림을 얹었을 때, 투명한 픽셀들은 그들 밑에 있는 픽셀들을 가려 감추지 않는다. 반면, 부분적으로 투명한 픽셀들은 그들 아래에 있는 픽셀과 섞이게 된다. 그림** 7-25**는 이미지 위에 부분적으로 투명한 사각형을 오버레이하는 효과를 보여준다. 아직까지 사각형 밑으로부터 이미지 빛을 자세히 볼 수 있다.**

Java2D API**에서는, 투명성이** alpha channel**에 의해 설명되어지고 있다. 각 픽셀은** red, green, blue **색상외에** 0**(완전히 투명한)과** 1**(완전히 불투명한) 사이의 값인 알파값을 갖는다. 예를**

들면, 그림 7-25의 시각형은 50% 의 투명성으로 연한 노란색으로 채워져 있다.

그림 7-25 : 이미지 위에 부분적으로 투명한 직사각형을 겹치기

이제, 두개의 형태들을 덧씌우면 어떻게 될지 보기로 한다. 원래 픽셀과 원하는 픽셀의 색상과 알파값을 섞거나 합한다. 컴퓨터 그래픽 분야의 연구자들인 Peter 와 Duff는 이러한 섞는 과정에서 12개의 가능한 합성규칙을 만들었다. Java2D API는 이러한 규칙중 8개를 구현한다. 좀더 진도를 나가기전에, 이들중 2개가 실제로 중요함을 지적하고자 하다.
룰이 이상하고 애매하면, SRC_OVER 룰을 사용하면 된다. Graphics2D 객체에 대한 디폴트 룰로, 가장 직감적인 결과를 낸다.

룰에 대한 가설이 있다. 만약 알파값 as를 가진 원래 픽셀이 있다면, 이미 알파값 ad을 갖는 목적 픽셀이 있다. 두 값을 합성하고자 하다. 그림 7-26의 다이어그램은 어떻게 합성룰을 디자인하는지를 보여준다.

Peter와 Duff는 픽셀 색상이 쓰여야 하는다는 확률로써 알파값을 생각했다. 원래 픽셀에서부터 생각하자면, 원래 색상을 쓰고자 하는 확률 as와 그렇지 않고자 하는 확률 1-as가 있다. 둘 다 목적 픽셀에서는 유효하다. 색상을 합성할 때, 확률이 독립적이라고 가정하자. 그러면 그림 7-26에서 네가지 경우가 있다. 원래 소스가 원래 색상을 쓰고자 하고 destination에서는 그렇지 않는 경우, 소스가 그대로 가지고 있으면 되는 것 같이 보인다. 그것은 다이어그램의 위 오른쪽 코너가 S로 라벨되어 있는 이유이다. 이 이벤트의 확률은 as(1 – ad)이다. 유사하게, 아래 왼쪽 코너는 D로 라벨되어 있다. Destination과 소스 모두가 그들의 색상을 고르고자 한다면 무엇을 해야 할까? 그래서 Peter-Duff의 룰이 나온 이유이다. 소스가 더 중요하다면, S를 아래 오른쪽에 또한 라벨하다. 이러한 룰을 SRC_OVER라 하다. 이 룰에서 as의 가중치를 가진 소스와 (1-as)ad (원문 수식으로 표현해 주세요!) 의 가중치를 가진 destination을 조합한다.

시각적인 효과는 소스에 주어진 전제에 따라 소스와 destination을 섞는 것이다. 특히, as 가 1 이면, 대상 색상은 전혀 고려되지 않는다. as 가 0이면, 소스 픽셀은 완전히 투명하고 대상 색상은 변함이 없다.

그림 7-26 : 합성룰 디자인

확률 다이어그램 상자에 무슨 글자를 넣을지에 따라 다르는 다른 룰이 있다. 표 7-1과 그림 7-27은 java2D API에 의해 지원되는 모든 룰들을 보여준다. 그림상의 이미지들은 0.75의

알파값을 갖는 사각 소스 지역이 1.0의 알파값을 갖는 타원형의 대상 지역과 조합할 때 룰의 결과를 보여준다.

표 7-1 : Peter-Duff 합성 룰

CLEAR	소스가 대상을 지운다.
SRC	소스가 대상을 덮어 씌우고 빈 픽셀이 된다.
SRC_OVER	소스가 대상과 섞기고 빈 픽셀을 덮어 씌운다.
DST_OVER	소스가 빈 픽셀을 덮어씌운다.
SRC_IN	소스가 대상을 덮어씌운다.
SRC_OUT	소스가 대상을 지우고 빈 픽셀을 덮어 씌운다
DST_IN	소스 알파가 대상을 변경한다.
DST_OUT	소스 알파 보색이 대상을 변경한다.

지금 볼 수 있듯이, 대부분의 룰은 유용하지 않다. 극단적인 경우에는 DST_IN 룰을 고려한다. 소스 색상은 전혀 고려되지 않지만, 소스의 알파값을 이용하여 대상에 영향을 준다. SRC 룰은 소스 색상이 사용되게 하고, 대상에 섞이지 않게 하므로 - 잠재적으로 유용하다. Poter-Duff 룰에 대한 보다 자세한 내용은 17장 6.1에 있는 Foley, van Dam, Feiner, et al.,의 예제를 참조하기 바란다.

그림 7-27 : Poter-Duff 합성 룰

Composite 메소드를 구현시에 클래스의 객체를 설정하는 Graphics2D 클래스의 setComposite 메소드를 사용한다. Java2D API 는 AlphaComposite 같은 하나의 클래스를 제공하여 그림 7-27 의 여덟개의 Poter-Duff 룰을 구현한다.

AlphaComposite 클래스의 팩토리 메소드 getInstance는 AlphaComposite을 만들어 낸다. 소스 픽셀에서 쓰일 룰과 알파값을 제공해야 한다. 예를 들어 아래의 코드를 생각해 보자.

```
int rule = AlphaComposite.SRC_OVER;
float alpha = 0.5
g2.setComposite(AlphaComposite.getInstance(rule, alpha) ) ;
g2.setPaint(Color.blue) ;
```

g2.fill(rectangle) ;

이러면, 시각형은 Blue 색상과 알파 값이 0.5로 채워진다. 합성룰이 SRC_OVER이기 때문에, 기존의 이미지에 투명하게 덮어 씌워 진다.
예제 7-7의 프로그램은 이러한 합성룰을 알아 볼 수 있게 한다. 콤보 박스에서 룰을 선택하고 AlphaComposite 객체의 알파값을 설정키 위해 슬라이더를 사용한다.

여기에, 프로그램은 각 룰의 구두 설명까지 보여준다. 설명이 합성룰 다이어그램에서 나온 것임을 유념하자. 예를 들면 두번째 줄의 DS는 "Blends with destination"을 나타낸다.

프로그램은 한가지 중요한 꼬임이 있다. 그래픽 컨텍스가 알파 채널을 가진 스크린에 상응되는지를 보장받지 못한다. (사실, 일반적으로 그렇지 못한다.) 픽셀이 알파 채널없이 대상에 저장되는 경우, 픽셀 색상은 알파값으로 나누어지고 알파값은 버려진다. poter-duff 룰의 몇개가 대상의 알파값에 쓰인 이후, 대상 알파 채널은 매우 중요해 졌다. 이러한 이유로, 형태를 합성하려면 ARGB 색상 모델을 가진 버퍼드 이미지를 사용한다. 이미지들이 합성된 이후, 결과 이미지를 화면에 그린다.

```
BufferedImage image = new BufferedImage(getWidth ( ), getHeight( ), BufferedImage.TYPE_INT_ARGB ) ;
Graphics2D gImage – image.createGraphics( ) ;
// gImage에 그린다.
g2.drawImage (image, null, 0, 0) ;
```

프로그램의 전체 코드가 여기에 있다. 그림 7-28은 스크린 화면을 보여준다. 프로그램은 실행시키고, 합성된 형태위의 효과를 볼려면 알파 슬라이더를 왼쪽에서 오른쪽으로 움직인다. 특히, DST_IN과 DST_OUT 룰 사이의 다른 하나의 차이가 소스 알파를 바꿀 때 어떻게 대상 색상을 바꾸는지에 유념해야 한다.

그림 7-28 : CompositeTest 프로그램

예제 7-7 : CompositeTest.java

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

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

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

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

       Container contentPane = getContentPane();
       canvas = new CompositePanel();
        contentPane.add(canvas, "Center");

       ruleCombo = new JComboBox();
        ruleCombo.addItem("CLEAR");
        ruleCombo.addItem("SRC");
        ruleCombo.addItem("SRC_OVER");
        ruleCombo.addItem("DST_OVER");
        ruleCombo.addItem("SRC_IN");
        ruleCombo.addItem("SRC_OUT");
        ruleCombo.addItem("DST_IN");
        ruleCombo.addItem("DST_OUT");
        ruleCombo.addActionListener(this);
```

```
        alphaSlider = new JSlider();
        alphaSlider.addChangeListener(this);
        JPanel panel = new JPanel();
        panel.add(ruleCombo);
        panel.add(new JLabel("Alpha"));
        panel.add(alphaSlider);
        contentPane.add(panel, "North");

        explanation = new JTextField();
        contentPane.add(explanation, "South");

        canvas.setAlpha(alphaSlider.getValue());
        canvas.setRule(ruleCombo.getSelectedItem());
        explanation.setText(canvas.getExplanation());
    }

    public void stateChanged(ChangeEvent event)
    {   canvas.setAlpha(alphaSlider.getValue());
    }

    public void actionPerformed(ActionEvent event)
    {   canvas.setRule(ruleCombo.getSelectedItem());
        explanation.setText(canvas.getExplanation());
    }

    private CompositePanel canvas;
    private JComboBox ruleCombo;
    private JSlider alphaSlider;
    private JTextField explanation;
}

class CompositePanel extends JPanel
{   public CompositePanel()
    {   shape1 = new Ellipse2D.Double(100, 100, 150, 100);
        shape2 = new Rectangle2D.Double(150, 150, 150, 100);
```

```
 }

public void paintComponent(Graphics g)
 {   super.paintComponent(g);
     Graphics2D g2 = (Graphics2D)g;

   BufferedImage image = new BufferedImage(getWidth(),
        getHeight(), BufferedImage.TYPE_INT_ARGB);
   Graphics2D gImage = image.createGraphics();
     gImage.setPaint(Color.red);
     gImage.fill(shape1);
     AlphaComposite composite
       = AlphaComposite.getInstance(rule, alpha);
     gImage.setComposite(composite);
     gImage.setPaint(Color.blue);
     gImage.fill(shape2);
   g2.drawImage(image, null, 0, 0);
 }

public void setRule(Object r)
 {  if (r.equals("CLEAR"))
    {   rule = AlphaComposite.CLEAR;
        porterDuff1 = "   ";
        porterDuff2 = "   ";
     }
    else if (r.equals("SRC"))
    {   rule = AlphaComposite.SRC;
       porterDuff1 = " S";
       porterDuff2 = " S";
     }
    else if (r.equals("SRC_OVER"))
    {   rule = AlphaComposite.SRC_OVER;
       porterDuff1 = " S";
        porterDuff2 = "DS";
     }
    else if (r.equals("DST_OVER"))
```

```
    {   rule = AlphaComposite.DST_OVER;
        porterDuff1 = " S";
        porterDuff2 = "DD";
     }
    else if (r.equals("SRC_IN"))
    {   rule = AlphaComposite.SRC_IN;
        porterDuff1 = "   ";
        porterDuff2 = " S";
     }
    else if (r.equals("SRC_OUT"))
    {   rule = AlphaComposite.SRC_OUT;
        porterDuff1 = " S";
        porterDuff2 = "   ";
     }
    else if (r.equals("DST_IN"))
    {   rule = AlphaComposite.DST_IN;
        porterDuff1 = "   ";
        porterDuff2 = " D";
     }
    else if (r.equals("DST_OUT"))
    {   rule = AlphaComposite.DST_OUT;
        porterDuff1 = "   ";
        porterDuff2 = "D ";
     }
     repaint();
 }

public void setAlpha(int a)
{   alpha = (float)a / 100.0F;
     repaint();
 }

public String getExplanation()
{   String r = "Source ";
    if (porterDuff2.equals("   "))
        r += "clears";
```

```
        if (porterDuff2.equals(" S"))
            r += "overwrites";
        if (porterDuff2.equals("DS"))
            r += "blends with";
        if (porterDuff2.equals(" D"))
            r += "alpha modifies";
        if (porterDuff2.equals("D "))
            r += "alpha complement modifies";
        if (!porterDuff2.equals("DD"))
        {   r += " destination";
            if (!porterDuff1.equals("   ")) r += " and ";
         }
        if (porterDuff1.equals(" S"))
            r += "overwrites";
        if (!porterDuff1.equals("   "))
            r += " empty pixels";
        return r + ".";
    }

    private Shape shape1;
    private Shape shape2;
    private float alpha;
    private int rule;
   private String porterDuff1; // row 1 of the rule diagram
   private String porterDuff2; // row 2 of the rule diagram
}
```

java.awt.Graphics2D

- void setComposite(Composite s)

Composite **인터페이스를 구현하는 주어진 객체에 해당 그래픽 컨텍스를 설정한다**.

java.awt.AlphaComposite

- static AlphaComposite getInstance(int rule)
- static AlphaComposite getInstance(int rule, float alpha)

알파 composite **객체를 생성한다**.

전달인자 : rule CLEAR, SRC, SRC_OVER, DST_OVER, SRC_IN, SRC_OUT, DST_IN, DST_OUT 중의 하나

alpha 소스 픽셀에 대한 알파 값.

렌더링 힌트

앞서 랜더링 과정이 매우 복잡하다는 것을 보았다. Java2D API가 대부분의 경우 놀라울 정도로 빠른 반면, 속도와 quality 사이를 상쇄하고 싶은 경우가 있다. 이 경우 랜더링 힌트로 상쇄시킬 수가 있다. Graphics2D 클래스의 setRenderingHint 메소드는 단일 힌트를 설정하게 한다. 힌트 키와 값은 RenderingHints 클래스에 선언되어 진다. 테이블 7-2는 선택 사항들을 요약했다.

테이블 7-2 : 랜더링 힌트

원문 표를 그대로 첨부해 주세요!

이러한 설정의 가장 유용한 점은 antialiasing과 관계가 있다. 비스듬한 선과 곡선에서 'jaggies'(들쑥 날쑥한 부분) 을 제거하는 테크닉이다. 그림 7-29에서 볼 수 있듯이, 비스듬한 선은 픽셀의 'staircase"로써 그려져야 된다. 저해상도 스크린에서는 특히 보기 좋지 않다. 전체적으로 각 픽셀을 on or off로 그리기보다는, 선을 덮고 있는 픽셀의 영역과 비례한 색상값으로 부분적으로 덮여진 픽셀에 색상이 있으면, 결과는 더욱 부드럽게 보인다. 이러한 기법은 antialiasing(안티 알리어싱) 이라 한다. 물론, 이 기법은 이러한 색상값을 도출해 내는데 시간이 걸리므로 보다 오래 걸린다.

그림 7-29 : Antialiasing

예를 들여, 아래는 antialiasing의 사용 요청을 하는 방법을 보여준다.

```
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, VALUE_ANTIALIAS_ON);
```

폰트에 antialiasing 을 쓸 수 있다.

```
g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
VALUE_TEXT_ALTIALIAS_ON) ;
```

그밖의 랜더링 힌트는 빈번하게 쓰이지 않는다.
또한 키/값 힌트 의 묶음을 맵에 둘수 있고, setRenderingHints 메소드을 호출하면서 한번에 설정할 수 있다. 맵 인터페이스를 구현하는 어느 컬렉션 클래스도 실행되지만, RenderingHints 클래스 자체를 또한 사용할 수 있다. 이는 맵 인터페이스를 구현하고 생성자에 NULL을 전달시키면 디폴트 맵 구현이 제공된다. 예를 들면,

```
RenderingHints hints = new RenderingHints(null) ;
hints.put(RenderingHints.KEY_ANTIALIASING, VALUE_ANTIALIAS_ON) ;
hints.put(RenderingHints.KEY_TEXT_ANTIALIASING,
VALUE_TEXT_ANTIALIAS_ON) ;
g2.setRenderingHints(hints) ;
```

이것은 예제 7-8에서 사용한 기법이다. 프로그램은 힌트의 몇가지 장점을 생각하고 이미지를 그린다. 다양한 랜더링 힌트를 on 또는 off 할 수 있다. 모든 플랫폼이 모든 힌트를 제공하는 것은 아니기 때문에, 설정된 각각 하나에서의 효과를 기대해서는 안된다. Windows 98에서는 antialiasing 이 타원을 부드럽게 하고, 텍스트 antialiasing은 이탤릭 폰트을 보기좋게 하다. 텍스트 antialiasing을 on 한다면, fractional 텍스트 행렬을 선택하여 스트링의 문자를 움직여 가깝게 만든다. 이밖의 힌트는 Windows98에서는 아무런 영향이 없지만, 기타 플랫폼에서나 보다 복잡한 이미지에서는 영향이 있을 수 있다.

그림 7-30은 프로그램의 스크린 캡쳐를 보여준다..

그림 7-30 : RenderingHints 프로그램

예제 7-8 : RenderingQualityTest.java

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

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

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

class RenderQualityTestFrame extends JFrame
    implements ActionListener
{   public RenderQualityTestFrame()
    {   setTitle("RenderQualityTest");
        setSize(400, 400);
        addWindowListener(new WindowAdapter()
           {   public void windowClosing(WindowEvent e)
               {   System.exit(0);
               }
           } );

        checkBoxContainer = Box.createVerticalBox();

        antiAliasingBox
            = makeCheckBox("Antialiasing");
        qualityRenderingBox
           = makeCheckBox("Quality rendering");
        ditheringBox
            = makeCheckBox("Dithering");
        textAntiAliasingBox
           = makeCheckBox("Font antialiasing");
        fractionalMetricsBox
            = makeCheckBox("Fractional font metrics");
        qualityAlphaInterpolationBox
           = makeCheckBox("Quality alpha interpolation");
        qualityColorRenderingBox
           = makeCheckBox("Quality Color rendering");

        Container contentPane = getContentPane();
        canvas = new RenderQualityPanel();
        contentPane.add(canvas, "Center");
```

```
    contentPane.add(checkBoxContainer, "North");
 }

JCheckBox makeCheckBox(String title)
{  JCheckBox box = new JCheckBox(title);
     box.addActionListener(this);
     checkBoxContainer.add(box);
    return box;
 }

public void actionPerformed(ActionEvent event)
{  // get values from all check boxes
   RenderingHints hints = new RenderingHints(null);
     hints.put(RenderingHints.KEY_ANTIALIASING,
        antiAliasingBox.isSelected()
           ? RenderingHints.VALUE_ANTIALIAS_ON
           : RenderingHints.VALUE_ANTIALIAS_OFF);
     hints.put(RenderingHints.KEY_RENDERING,
        qualityRenderingBox.isSelected()
           ? RenderingHints.VALUE_RENDER_QUALITY
           : RenderingHints.VALUE_RENDER_SPEED);
     hints.put(RenderingHints.KEY_DITHERING,
        ditheringBox.isSelected()
           ? RenderingHints.VALUE_DITHER_ENABLE
           : RenderingHints.VALUE_DITHER_DISABLE);
     hints.put(RenderingHints.KEY_TEXT_ANTIALIASING,
        textAntiAliasingBox.isSelected()
           ? RenderingHints.VALUE_TEXT_ANTIALIAS_ON
           : RenderingHints.VALUE_TEXT_ANTIALIAS_OFF);
     hints.put(RenderingHints.KEY_FRACTIONALMETRICS,
        fractionalMetricsBox.isSelected()
           ? RenderingHints.VALUE_FRACTIONALMETRICS_ON
           : RenderingHints.VALUE_FRACTIONALMETRICS_OFF);
     hints.put(RenderingHints.KEY_ALPHA_INTERPOLATION,
        qualityAlphaInterpolationBox.isSelected()
           ? RenderingHints.VALUE_ALPHA_INTERPOLATION_QUALITY
```

```
                : RenderingHints.VALUE_ALPHA_INTERPOLATION_SPEED);
        hints.put(RenderingHints.KEY_COLOR_RENDERING,
            qualityColorRenderingBox.isSelected()
                ? RenderingHints.VALUE_COLOR_RENDER_QUALITY
                : RenderingHints.VALUE_COLOR_RENDER_SPEED);
        canvas.setRenderingHints(hints);
    }

    private RenderQualityPanel canvas;
    private JCheckBox antiAliasingBox;
    private JCheckBox qualityRenderingBox;
    private JCheckBox ditheringBox;
    private JCheckBox textAntiAliasingBox;
    private JCheckBox fractionalMetricsBox;
    private JCheckBox qualityAlphaInterpolationBox;
    private JCheckBox qualityColorRenderingBox;
    private Box checkBoxContainer;
}

class RenderQualityPanel extends JPanel
{   public RenderQualityPanel()
    {   Random generator = new Random();
        color1 = new Color(0.7F, 0.7F, 0.0F, 0.5F);
        color2 = new Color(0.0F, 0.3F, 0.3F, 0.5F);
        image = Toolkit.getDefaultToolkit().getImage
            ("clouds.jpg");
        MediaTracker tracker = new MediaTracker(this);
        tracker.addImage(image, 0);
        try { tracker.waitForID(0); }
        catch (InterruptedException e) {}
    }

    public void paintComponent(Graphics g)
    {   super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.setRenderingHints(hints);
```

```
        g2.drawImage(image, 0, 0, null);
        g2.draw(new Ellipse2D.Double(0, 0,
             image.getWidth(null), image.getHeight(null)));
        g2.setFont(new Font("Serif", Font.ITALIC, 40));
        g2.drawString("Hello", 75, 75);
         g2.setPaint(color1);
         g2.translate(0,-80);
        g2.fill(new Rectangle2D.Double(100, 100, 200, 100));
         g2.setPaint(color2);
        g2.fill(new Rectangle2D.Double(120, 120, 200, 100));
    }

    public void setRenderingHints(RenderingHints h)
    {   hints = h;
        repaint();
    }

   private RenderingHints hints = new RenderingHints(null);
    private Color color1;
    private Color color2;
    private Image image;
}
```

java.awt.Graphics2D

- void setRenderingHint (RenderingHints.Key key, Object value)

해당 그래픽 컨텍스를 위한 랜더링 힌트를 설정한다.

- void setRenderingHints (Map m)

맵에 저장되어있는 key/**값 쌍의 랜더링 힌트를 설정한다**.

java.awt.RenderingHints

- RenderingHints(Map m)

랜더링 힌트를 저장하기 위해 랜더링 힌트 맵을 구성한다. m **이** null**이면**, **디폴트 맵 구현이 제공된다**.

이미지 조작

사용자가 이미지를 갖고 있고 외형을 더욱 예쁘게 하고 한다고 가정하자. **이미지 각 픽셀에**

접근해서 다른 픽셀로 대치해야 할 필요가 있다. 아니면 예를 들어, scratch에서 이미지의 픽셀을 계산해서 물리적인 측정치나 수학적인 계산식을 보여주길 원할 수 있다. BufferedImage 클래스는 사용자가 이미지 내부의 픽셀을 제어 할 수 있게 하고 이미지를 변형할 수 있게 하는 BufferedImageOp 인터페이스를 구현할 수 있게 한다.

이것은 JDK1.0의 이미지 지원에서의 큰 변화이다. 그때, 이미지 클래스들은 incremental rendering 을 지원하고자 최적화되었다. 클래스들의 원래 목적은 웹에서 다운받은 GIF, JPEG 이미지들을 랜더링하여 부분 이미지 데이터가 사용 가능하자마자 바로 스캔하기 위한 것 이었다. 실제로, 짝수 스캔라인이 먼저 오고 홀수 스캔라인이 나중에 오면 스캔 라인들은 교차될 수 있다. 이러한 메커니즘은 브라우저가 이미지 데이탕의 리마인더를 가져오는 동안 재빨리 이미지의 근사값을 나타낼 수 있게 한다. JDK 1.0에서 ImageProducer, ImageFilter, ImageConsumer는 증가적인 랜더링의 복잡성을 모두 나타낸다. 프레임웍에 잘 맞추기 위한 이미지 manipulation은 아주 복잡하다.

다행히, 이러한 클래스들을 사용할 필요가 없다. Java 2 플랫폼은 JDK 1.0의 "push model"에 "direct" 모델로 대치하여 바로 손쉽게 픽셀을 접근 할 수 있게 한다. 이번 장에서는 다이렉트 모델만 설명하고자 하다. 다이렉트 모델의 유일한 맹점은 모든 이미지 픽셀이 메모리에 들어가게 한다는 것이다.(실제로, "push model"도 같은 제약이 있었다. 처리된 픽셀들이 사용가능하게 하도록 이미지 manipulation 알고리즘써서 복잡한 컨닝(Cunning)을 요구했었다. 옛 모델 사용자의 대부분이 이를 처리하기전에 전체 이미지를 단순히 버퍼시켰다.) java 플레폼의 다음 버전은 픽셀들이 필요할 때, 파이프라인 처리가 메모리 소비을 감소시키고 픽셀의 가져오거나 처리하는 속도를 빠르게 하는 "Pull"모델을 지원할지도 모른다.

이미지 데이터 접근

다루게 되는 대부분의 데이터는 – 디지털 카메라나 스캐너 같은 디바이스에 의해 생성되거나 그리기 프로그램에서 생선된 - 이미지 파일에서 단순히 읽는 것이다. 이번 장에서는, 이미지를 생성하는, 즉 다시 말해 한 픽셀에 한번에 이미지를 생성하는 다른 기법을 보여주고자 한다.

이미지를 생성하기 위해서, 일반적인 방법으로 BufferedImage 객체를 생성한다.

```
image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB) ;
```

이제는 객체의 형태인 WritableRaster을 얻기위해서 getRaster 메소드를 호출한다. 사용자는 이미지의 픽셀에 접근하여 수정할 때 이 객체를 사용 가능하다.

```
WritableRaster raster = image.getRaster( ) ;
```

setPixel 메소드는 각각의 픽셀을 설정하게 한다, 여기서 복잡성은 사용자가 단순히 픽셀을

색상값에 설정할 수 없게해서 버퍼드된 이미지가 어떤 색상값을 나타내는지를 알아야 하다. 이것은 이미지의 타입에 따라 다르다. 사용자의 이미지가 TYPE_INT_ARGB 타입이면, 각 픽셀은 0과 255사이인 red, green, blue 그리고 알파인 4가지의 값으로 설명되어 진다. 사용자는 정수 네개인 배열로 이들을 제공할 필요가 있다.

```
int [ ] black = {0, 0, 0, 255} ;
raster.setPixel ( I , j, black ) ;
```

Java 2D API의 전문용어로 이러한 값들을 픽셀의 ‘ sample values’ 로 불린다.

주의 : float [ ] 와 double [ ]의 배열 전달인자를 갖는 setPixel 메소드들도 또한 있다. 그러나, 이러한 배열에 대치시키기 위해 필요한 값들은 0.0과 1.0사이의 정규화된 색상 값이 아니다.

```
float [ ] red = { 0.1F, 0.0F, 0.0F, 1.0F } ;
raster.setPixel ( I, j, red ) ; //에러
```

배열 타입이 무엇인지 간에, 사용자는 1과 255사이의 값을 제공해야 한다.

사용자는 setPixel 메소드와 함께 픽셀들의 배치를 제공한다.. 사용자가 설정하고자 하는 시각형의 시작 픽셀 위치와 두께, 높이를 명시한다. 그리고, 모든 픽셀에 대한 sample values를 담고 있는 배열을 제공한다. 예를 들면, 버퍼드 이미지가 TYPE_INT_ARGB 타입이면, 첫 픽셀의 red, green, blue, 알파값, 그리고 두번째 픽셀의 red, green, blue 그리고 알파값 등 등을 제공한다.

```
int [ ] pixels = new int [ 4 * width * height ] ;
pixels[0]= …… // 첫번째 픽셀의 red 값
pixels[1] = ……// 첫번째 픽셀의 green 값
pixels[2] = ……// 첫번째 픽셀의 blue 값
pixels[3] = ……// 첫번째 픽셀의 알파 값
…….
raster.setPixels(x, y, width, height, pixels) ;
```

반대로, 픽셀을 읽기 위해서, getPixel 메소드를 사용한다. sample values를 얻으려면 4개의 정수의 배열을 제공한다..

```
int [ ] sample = new int[4];
raster.getPixel(x, y, sample) ;
Color c = new Color(sample[0], sample[1], sample[2], sample[3] ) ;
```

getPixel 메소드를 갖는 여러 개의 픽셀을 읽을 수 있다.

```
raster.getPixels(x, y, width, height, samples) ;
```

사용자가 TYPE_INT_ARGB와는 다른 이미지 타입을 사용하고 픽셀 값을 나타내는 방법을 안다면, getPixel/setPixel 메소드 사용이 간단하다. 그러나, 특정한 이미지 타입에서 sample values의 인코딩을 알아야 하다.
임의의, 알 수없는 이미지 타입의 이미지를 다룰 필요가 있으면, 약간 어렵게 작업해야 한다. 모든 이미지 타입은 샘플 값 배열과 표준 RGB 색상 모델 간을 전환할 수 있는 색상 모델을 가지고 있다.

노트 : RGB 색상 모델은 사용자가 생각하는 것처럼 그렇게 표준적이지 않다. 색상값을 자세히 보면 이미징하는 디바이스의 특성에 따라 다르다. 디지털 카메라, 스캐너, 모니터, 그리고 LCD(액정 디스플레이) 모두가 그들만의 독특한 표현 양식을(idiosyncrasies) 가지고 있다. 결과적으로, 같은 RGB 값도 다른 디바이스에 따라 다르게 보일 수 있다. 국제 색상 학회(www.color.org)는 색상맵을 1931년 CIE XYZ 색상 명세서 같은 표준화하는 방법에 대해서 명시해 놓은 ICC 프로파일을 따르도록 제안하다. 이 명세서는 조명과 색상에 관한 기술적인 지도을 제공하는 책임을 맡고있는 국제기구인 CIE(www.cie.co.at/cie)에서 1931에 만들어 졌다. 명세서는 x, y, z 으로 불리는 좌표로서 사람의 눈이 지각할 수 있는 모든 색상을 나타내는 표준 방법이다. (예를 들면, 1931 CIE XYZ 명세서상의 정보를 알고자 하면, 13장의 Foley, van Dam, Feiner, et al을 참고하라.) 그러나, ICC 프로파일는 복잡하다. sRGB(http://www.w3.org/Graphics/Color/sRGB.html)라 불리는 보다 간단히 제안된 표준은 RGB 값과 기존의 컬러 모니터에 잘 작동되도록 디자인된 1931 CIE XYZ 값들 사이를 정확히 맵핑한 것을 나타내 준다. Java 2D API 는 RGB 와 기타 다른 색상 공간사이의 변환일 때 이 맵핑을 이용한다.

---

getColorModel 메소드는 컬러 모델을 반환한다.

```
ColorModel model = image.getColorModel( ) ;
```

픽셀의 색상값을 찾기위해서, 사용자는 Raster 클래스의 getDataElements 메소드를 호출한다. 이 호출은 색상 값의 색상 모델 특정한 설명을 포함한 Object를 반환한다.

```
Object data = raster.getDataElement(x, y, null) ;
```

노트 : getDataElement 메소드에 의해 반환된 객체는 실제로 sample values의 배열이다. 그것을 처리하는 과정을 알 필요는 없지만, 메소드가 getDataElements로 불리는지는 알 수 있다.

색상 모델은 객체를 표준 ARGB 값으로 전환하다. getRGB 메소드는 각각 8 비트의 4블록으로 싸여진 알파, red, green, blue값을 가진 정수값을 반환한다. 사용자는 Color(int argb, Boolean has Alpha) 생성자를 가진 정수로부터 Color 값을 생성 할 수 있다.

```
int argb = model.getRGB(data) ;
Color color = new Color(argb, true) ;
```

픽셀을 특정한 색상에 설정할 때, 이러한 과정을 반대로 해야 한다. Color 클래스의 getRGB 메소드는 알파, red, green, blue값을 가진 정수값을 산출하고, 그 값을 ColorModel 클래스의 getDataElement 메소드에 적용하다. 반환값은 색상 값의 색상-모델 특정의 설명을 담고 있는 Object이다. 이 Object를 WritableRaster 클래스의 setDataElements 메소드의 전달하다..

```
int argb = color.getRGB ( ) ;
Object data = model.getDataElements(argb null) ;
raster.setDataElements(x, y, data) ;
```

각각의 픽셀로부터 이미지를 만들기 원한 이러한 모델의 사용을 설명하자면, we bow to tradition and 그림 7-31에서 보여주듯이 Mandelbrot set를 그린다.

그림 7-31 : A Mandelbrot set

Mandelbrot set의 개념은 순차적으로 평면에 있는 각 점을 연상하면 된다. 순서가 둘레에 머물러 있으면, 포인트에 색칠하다. “escapes to infinity”이면, 투명한 채로 둠니다. 순어에 대한 공식은 복소수의 수학적 계산에서 나옵니다. 이것은 당연한 것으로 간주하다. fractal의 수학에 관해서는 많은 책들이 나와 있다: 그중에 하나는 두껍고 광범위하게 설명된 ‘Chaos and fractals:New Frontiers of science 로 저자들은 Heinz-Otto Peitgen, Hartmut Jurgens, Dietmar Saupe, [Springer Verlag 1992] 이다.

여기서 가장 간단한 Mandelbrot set의 구성 방법을 알아보겠습니다. 각 점(a, b)마다 (x, y) = (0, 0) 로 시작되는 순서이며 반복됨을 알수 있다.:

(수학공식을 쓸 것)

$$x_{new} = x^2 - y^2 + a$$
$$y_{new} = 2xy + b$$

순서가 바운드에 있는지 x와 y가 계속 커져서 무한대로 되는지 체크하다. x또는 y가 2보다 커지면 순서가 무한하다는 것이 판명된다. bounded sequence 으로 오는 (a, b)점에 상응하는 픽셀들 만이 색칠된다.

예제 7-9는 코드이다. 이 프로그램에서, 색상 값을 픽셀 값으로 전환 할 때 ColorModel 클래스를 어떻게 사용하는지 보여준다. 이 과정은 이미지 형태와는 무관하다. 재미삼아, 버퍼드 이미지의 색상 타입을 TYPE_BYTE_GRAY로 변환하여 보자. 다른 코드들은 변환할 필요가 없다. - 이지미의 색상 모델이 자동으로 색상을 샘플값으로의 전환을 해준다.

예제 7-9 : MandelbrotTest.java

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

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

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

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

class MandelbrotPanel extends JPanel
{  public void paintComponent(Graphics g)
```

```
{   super.paintComponent(g);
    BufferedImage image = new BufferedImage(getWidth(),
        getHeight(), BufferedImage.TYPE_INT_ARGB);
     generate(image);
    g.drawImage(image, 0, 0, null);
}

public void generate(BufferedImage image)
{   int width = image.getWidth();
    int height = image.getHeight();
    WritableRaster raster = image.getRaster();
    ColorModel model = image.getColorModel();

    Color fractalColor = Color.red;
    int argb = fractalColor.getRGB();
    Object colorData = model.getDataElements(argb, null);

   for (int i = 0; i < width; i++)
      for (int j = 0; j < height; j++)
      {   double a = XMIN + i * (XMAX - XMIN) / width;
          double b = YMIN + j * (YMAX - YMIN) / height;
            if (!escapesToInfinity(a, b))
                raster.setDataElements(i, j, colorData);
          }
}

private boolean escapesToInfinity(double a, double b)
{   double x = 0.0;
    double y = 0.0;
    int iterations = 0;
     do
    {   double xnew = x * x - y * y + a;
        double ynew = 2 * x * y + b;
         x = xnew;
         y = ynew;
          iterations++;
```

```
            if (iterations == MAX_ITERATIONS) return false;
          }
        while (x <= 2 && y <= 2);
         return true;
      }

    private static final double XMIN = -2;
    private static final double XMAX = 2;
    private static final double YMIN = -2;
    private static final double YMAX = 2;
    private static final int MAX_ITERATIONS = 16;
}
```

java.awt.image.BufferedImage

- BufferedImage(int width, int height, int imageType)

버퍼드 이미지 객체를 생성한다.

전달인자 : width, height **이미지 너비**

imageType TYPE_INT_RGB, TYPE_INT_ARGB, TYPE_BYTE_GRAY, TYPE_BYTE_INDEXED, TYPE_USHORT_555_RGB**등의 타입들**

- ColorModel getColorModel( )

해당 버퍼드 이미지의 색상모델을 반환한다.

- WritableRaster getRaster( )

버퍼드 이미지의 픽셀에 접근하고 수정하기 위한 래스터를 얻는다.

java.awt.image.Raster

- Object getDataElements(int x, int y, Object data)

색상 모델에 따라 요소 타입과 길의의 배열내부에서 래스터 포인트에 대한 샘플 데이터를 반환한다. 데이터가 NULL**이 아니면, 샘플 데이터를 저장하기에 적당한 배열로 보고 채워진다. 데이터가** NULL**이면, 새로운 배열이 할당된다.**

전달인자 : x, y **픽셀 할당**

data **픽셀에 대한 샘플 데이터가 채워지기에 적당한** NULL **또는 배열. 이 요소 타입과 길이는 색상모델에 따라 달라진다.**

- int [ ] getPixel(int x, int y, int w, int h, int [ ] sampleValues)
- float [ ] getPixel(int x, int y, int w, int h, float [ ] sampleValues)

- double [ ] getPixel (int x, int y, int w, int h, double [ ] sampleValues)
- int [ ] getPixels(int x, int y, int w, int h, int [ ] sampleValues)
- float [ ] getPixels(int x, int y, int w, int h, float [ ] sampleValues)
- double [ ] getPixels (int x, int y, int w, int h, double [ ] sampleValues)

색상 모델에 의존하는 길이의 배열 내부에서 래스터 포인트의 사각형 이나 래스터 포인트에 대한 샘플값을 반환한다. sampleValues**가** NULL**이 아니면, 샘플값을 담기에 충분히 길다고 보고 채워진다.** sampleValues **가** NULL**이면, 새로운 배열이 할당된다. 이러한 모델들은 색상 모델의 샘플값의 아는 경우에만 유용하다.**

전달인자 :	x, y	**래스터 포인트 위치, 또는 사각형의 위 왼쪽 코너**
	w, h	**래스터 포인트 사각형의 너비와 높이**
	sampleValues	**샘플값이 채워지기에 충분히 긴 배열이거나** NULL

java.awt.image.WritableRaster

- void setDataElements(int x, int y, Object data)

래스터 포인트에 대한 샘플값을 설정한다.

전달인자 :	x, y	**픽셀 위치**
	data	**픽셀에 대한 샘플데이타가 채워진 배열.**
		이 요소 타입과 길이는 색상 모델에 의존한다.

- void setPixel(int x, int y, int w, int h, int [ ] sampleValue)
- void setPixel(int x, int y, int w, int h, float [ ] sampleValues)
- void setPixel (int x, int y, int w, int h, double [ ] sampleValues)
- void setPixels(int x, int y, int w, int h, int [ ] sampleValues)
- void setPixels(int x, int y, int w, int h, float [ ] sampleValues)
- void setPixels(int x, int y, int w, int h, double [ ] sampleValues)

래스터 포인트나 래스터 포인트 사각형에 대한 샘플 값을 설정한다. 이러한 방법들은 색상 모델에 대한 샘플값의 인코딩을 알 때 유용하다.

전달인자 :	x, y	**래스터 포인트 위치, 사각형의 위쪽 왼쪽 코너**
	w, h	**래스터 포인트 사각형의 너비와 높이**
	sampleValues	**샘플 데이터로 채워진 배열. 이 요소타입과 길이는 색상 모델에 의존한다.**

java.awt.image.ColorModel

- int getRGB(Object data)

데이터에 들어온 샘플 데이터에 상응하는 ARGB 값을 반환한다.

전달인자 : data 픽셀에 대한 샘플 데이터로 채워진 배열
이 요소 타입과 길이는 색상 모델에 의존한다.

- Object getDataElements(int argb, Object data)

색상값에 대한 샘플 데이타를 반환한다. 데이터가 NULL이 아니면, 샘플데이타를 담기에 적당한 배열이라 간주하고 채워진다. 데이터가 NULL이면, 새로운 배열이 할당된다.

전달인자 : argb 색상값
data 색상값에 대한 샘플 데이터로 채워지기에 적당한 배열 또는 NULL. 이 요소타입과 길이는 색상모델에 의존한다.

java.awt.Color

- Color (int argb, Boolean has Alpha)

has Alpha 가 참이면 표시된 조합 ARGB 값이거나 has Alpha 가 거짓일 때 표시된 RGB 값을 가진 색상을 만든다.

- int getRGB ( )

해당 색상에 상응하는 ARGB 색상값을 반환한다.

이미지 필터링

앞 장에서, scratch에서 이미지를 어떻게 만드는지를 보았다. 그러나, 종종 어떠한 이유:이미 이미지를 가지고 있고 다른 방법으로 더 향상시키고자 하다.- 로 이미지 데이터를 접근하고자 하다:

물론, 앞장에서 이미 이미지 데이터를 읽고, 조작하고 되돌려 쓰는 방법을 보았기에 getPixel/getDataElements 메소드를 사용할 수 있다. 불행히 Java2 기술은 이미 동일한 이미지 과정을 수행하는 많은 필터들을 제공하고 있다.

모든 이미지 조작은 BufferedImageOp 인터페이스를 수행한다. 수행후에, 단순히 이미지를 다른 걸로 변환할 때 filter 메소드를 호출한다.

```
BufferedImageOp op = ...;
BufferedImage filteredImage
= new BufferedImage(image.getWidth( ), image.getHeight( ), image.getType ( ) ) ;
op.filter( image, filteredImage );
```

어떤 과정들은 (op.filter(image, image))위치의 이미지를 변환할 수 있지만, 대부분은 그러지 못하다.

BufferedImageOp 인터페이스를 수행하는 5가지의 클래스들이 있다.

AffineTransformOp

RescaleOp

LookupOp

ColorConvertOp

convolveOp

AffineTransformOp는 픽셀에서 affine 변형을 수행한다. 예를 들면, 이미지를 중심에 대해서 회전할 수 있음을 보여준다.

```
AffineTransform transform
        = AffineTransform.getRotateInstance(Math.toTadians(angle),
        image.getWidth( ) / 2, image.getHeight( ) / 2 ) ;
AffineTransformOp op
        = new AffineTransformOp(transform, interpolation ) ;
op.filter(image, filteredImage ) ;
```

AffineTransformOp 생성자는 affine 변형과 Interpolation 전략을 요구한다. 소스 픽셀이 타겟 픽셀의 어딘가에서 변형된다면, Interpolation은 타겟 이미지 안의 픽셀을 결정하는데 필요하다. 예를 들면, 소스 픽셀을 회전시킨다면, 픽셀들이 정확히 타겟 픽셀이 떨어지지 않을 것이다. 여기 2가지의 interpolation 전략이 있다: AffineTransformOp.TYPE_BILINEAR, AffineTransformOp.TYPE_NEAREST_NEIGHBOR. Bilinear interpolation은 약간 시간은 걸리지만 보기에 좋다.

예제 7-10상의 프로그램은 5도 만큼 이미지를 회전할 수 있게 한다. (그림 7-32)

그림 7-32 : 회전된 이미지

RescaleOp 는 이미지 내부의 샘플 값 x 에 대한 스케일링 작업을 수행한다.

xnew = a.x + b (수식 표현)

리스케일링 이후 너무 크거나 작은 샘플값들은 가장 크거나 가장 작은 legal 값을 설정된다. 이미지가 ARGB 포맷안에 있으면, 알파값만 빼고 red, green, blue 값에 대해서 각각 스케일링이 수행된다. a > 1 리스케일링의 영향은 이미지를 밝게 하기 위해서이다. 스케일링 전달인자와 선택적인 랜터링 힌트를 명시함으로써 RescaleOp을 구성한다. 예제 7-10에서,

```
float a = 1.5f;
float b = -20.0f ;
RescaleOp op = new RescaleOp (a, b, null );
```

을 사용한다.
LookupOp 사용은 샘플값의 arbitrary 맵핑을 명시할 수 있게 한다. 각 값들이 어떻게 맵핑되는지 명시하는 테이블을 제공한다. 예제 프로그램에서, 색상 C에서 255-c로 바꾸면서 모든 색상의 음수 값을 생성한다.

LookupOp 생성자는 선택 힌트의 맵과 타입 객체인 LookupTable을 요구한다. LookupTable 클래스는 추상적이다. ByteLookupTable과 ShortLookupTable 2개의 구체적인 하위 클래스들이 있다. RGB 색상이 바이트인 이후, ByteLookupTable을 사용한다. 사용자는 바이트 배열과 그 배열에 대한 정수 offset으로 부터 이러한 테이블을 구성한다. 예제 프로그램에 대한 LookupOp을 어떻게 구성하는지 보여준다.

```
byte negative[ ] = new byte[256] ;
for ( int I = 0 ; I < 256; I ++)
        negative[I] = (byte) (255 –1 ) ;
ByteLookupTable table = new ByteLookupTable(0, negative ) ;
LookupOp op = new LookupOp ( table, null ) ;
```

lookup은 개별적으로 각 색상값에 적용되지만 알파 값에는 적용되지 않는다.

노트 : 인덱스 색상 모델에 있는 이미지에는 LookupOp를 적용할 수 없다. (이러한 이미지들은 각 샘플값이 색상 팔레트로 상쇄된다.)

ColorConvertOp는 색상 공간 변형시에 유용하다. 여기서 이에 대해 논의는 하지 않는다.

변형중에서 가장 강력한 것은 ConvolveOp로, 수학적인 convolution(나선)을 수행한다. 우리는 convolution의 수학적인 자세한 설명까지는 들어가지 않지만, 기본 개념은 간단하다. 예를 들어, blur filter을 생각해 보자(그림 7-33)

그림 7-33 : Blurring an image

흐려지는 것은 픽셀과 이에 인접한 8개 픽셀로부터 평균값으로 각 픽셀을 대치시킨다. 직관적으로, 이러한 작동이 그림을 흐리게 하는지 이해가 된다. 수학적으로, 평균은 아래의 커널과의 convolution 동작으로 나타내어 진다:

(행렬 표현해 주세요?)

convolution의 커널은 행렬로 어떤 가중치가 인접한 값에 적용되는지를 알려준다. 위의 커널은 흐린 이미지를 나타낸다:
(행렬 표현해 주세요?)

다른 커널은 색상변화의 공간에 위치한 edge detection을 수행한다.(그림 7-34) edge detection은 그림 이미지를 분석할 때 가장 중요한 기법이다.

그림 7-34 : Edge detection

convolution 작업을 생성하려면, 커널에 대한 값의 배열을 먼저 설정하고 커널 객체를 구성한다. 그리고 커널에서 ConvolveOp 객체를 구성하고 필터링에 사용한다.

```
float [ ] elements =
        { 0.0f, -1.0f, 0,0f, -1,0f, 4.f, -1.0f, 0.0f, -1.0f, 0.0f } ;
Kernel kernel = new Kernel(3, 3, elements) ;
ConvolveOp op = new ConvolveOp (kernel);
op.filter(image, filterImage) ;
```

예제 7-10의 프로그램은 GIF, JPEG 이미지를 로드하여 이미 토의했던 이미지 조작을 수행한다. Java 2D API가 제공하는 이미지 조작 기능 덕분에, 프로그램은 매우 간단하다.

노트 : 이것을 쓸 시점에, 자바 플랫폼은 GIF, JPEG 형태를 이미지를 저장하는 공식적인 지원은 없다. 그러나, Sun Microsystems 의 JDK 의 일부인 com.sun.image.codec.jpeg 패키지가 있는데, 이미지 저장을 지원하다. 이 패키지에 대한 정보는 http://java.sun.com/products/jdk/1.2/docs/guide/2d/api-jpeg/overview-summary.html 에서 찾아 볼 수 있다. www.acme.com/java 에서 GIF 엔코더를 찾을 수 있다.

예제 7-10 : ImageProcessingTest.java

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

```
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.io.*;

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

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

      Container contentPane = getContentPane();
      panel = new ImageProcessingPanel();
      contentPane.add(panel, "Center");

      JMenu fileMenu = new JMenu("File");
      openItem = new JMenuItem("Open");
      openItem.addActionListener(this);
      fileMenu.add(openItem);

      exitItem = new JMenuItem("Exit");
      exitItem.addActionListener(this);
      fileMenu.add(exitItem);
```

```
    JMenu editMenu = new JMenu("Edit");
    blurItem = new JMenuItem("Blur");
     blurItem.addActionListener(this);
     editMenu.add(blurItem);

    sharpenItem = new JMenuItem("Sharpen");
     sharpenItem.addActionListener(this);
     editMenu.add(sharpenItem);

    brightenItem = new JMenuItem("Brighten");
     brightenItem.addActionListener(this);
     editMenu.add(brightenItem);

    edgeDetectItem = new JMenuItem("Edge detect");
     edgeDetectItem.addActionListener(this);
     editMenu.add(edgeDetectItem);

    negativeItem = new JMenuItem("Negative");
     negativeItem.addActionListener(this);
     editMenu.add(negativeItem);

    rotateItem = new JMenuItem("Rotate");
     rotateItem.addActionListener(this);
     editMenu.add(rotateItem);

     JMenuBar menuBar = new JMenuBar();
     menuBar.add(fileMenu);
     menuBar.add(editMenu);
     setJMenuBar(menuBar);
 }

public void actionPerformed(ActionEvent evt)
{   Object source = evt.getSource();
    if (source == openItem)
     {    JFileChooser chooser = new JFileChooser();
          chooser.setCurrentDirectory(new File("."));
```

```
            chooser.setFileFilter(new
                javax.swing.filechooser.FileFilter()
              {   public boolean accept(File f)
                  {   String name = f.getName().toLowerCase();
                      return name.endsWith(".gif")
                          || name.endsWith(".jpg")
                          || name.endsWith(".jpeg")
                          || f.isDirectory();
                   }
                  public String getDescription()
                  {   return "Image files";
                   }
                });

            int r = chooser.showOpenDialog(this);
            if(r == JFileChooser.APPROVE_OPTION)
            {   String name
                    = chooser.getSelectedFile().getAbsolutePath();
                panel.loadImage(name);
            }
        }
      else if (source == exitItem) System.exit(0);
      else if (source == blurItem) panel.blur();
      else if (source == sharpenItem) panel.sharpen();
      else if (source == brightenItem) panel.brighten();
      else if (source == edgeDetectItem) panel.edgeDetect();
      else if (source == negativeItem) panel.negative();
      else if (source == rotateItem) panel.rotate();
    }

    private ImageProcessingPanel panel;
    private JMenuItem openItem;
    private JMenuItem exitItem;
    private JMenuItem blurItem;
    private JMenuItem sharpenItem;
```

```
    private JMenuItem brightenItem;
    private JMenuItem edgeDetectItem;
    private JMenuItem negativeItem;
    private JMenuItem rotateItem;
}

class ImageProcessingPanel extends JPanel
{  public void paintComponent(Graphics g)
    {   super.paintComponent(g);
        if (image != null)
            g.drawImage(image, 0, 0, null);
    }

    public void loadImage(String name)
    {   Image loadedImage
            = Toolkit.getDefaultToolkit().getImage(name);
        MediaTracker tracker = new MediaTracker(this);
        tracker.addImage(loadedImage, 0);
        try { tracker.waitForID(0); }
        catch (InterruptedException e) {}
        image = new BufferedImage(loadedImage.getWidth(null),
            loadedImage.getHeight(null), BufferedImage.TYPE_INT_RGB);
        Graphics2D g2 = image.createGraphics();
        g2.drawImage(loadedImage, 0, 0, null);

        repaint();
    }

    private void filter(BufferedImageOp op)
    {   BufferedImage filteredImage
            = new BufferedImage(image.getWidth(), image.getHeight(),
                image.getType());
        op.filter(image, filteredImage);
        image = filteredImage;
        repaint();
    }
```

```
private void convolve(float[] elements)
 {    Kernel kernel = new Kernel(3, 3, elements);
    ConvolveOp op = new ConvolveOp(kernel);
      filter(op);
 }

public void blur()
{   float weight = 1.0f/9.0f;
    float[] elements = new float[9];
   for (int i = 0; i < 9; i++)
          elements[i] = weight;
      convolve(elements);
 }

public void sharpen()
{   float[] elements =
         {   0.0f,  -1.0f, 0.0f,
             -1.0f,   5.f, -1.0f,
             0.0f,  -1.0f, 0.0f
          };
      convolve(elements);
 }

void edgeDetect()
{   float[] elements =
         {   0.0f,  -1.0f, 0.0f,
             -1.0f,   4.f, -1.0f,
             0.0f,  -1.0f, 0.0f
          };
      convolve(elements);
 }

public void brighten()
{   float a = 1.5f;
    float b = -20.0f;
```

```
        RescaleOp op = new RescaleOp(a, b, null);
        filter(op);
    }

    void negative()
    {  byte negative[] = new byte[256];
      for (int i = 0; i < 256; i++)
          negative[i] = (byte)(255 - i);
      ByteLookupTable table = new ByteLookupTable(0, negative);
        LookupOp op = new LookupOp(table, null);
        filter(op);
    }

    void rotate()
    {  AffineTransform transform
          = AffineTransform.getRotateInstance(Math.toRadians(5),
            image.getWidth() / 2, image.getHeight() / 2);
        AffineTransformOp op = new AffineTransformOp(transform,
            AffineTransformOp.TYPE_BILINEAR);
        filter(op);
    }

    private BufferedImage image;
}
```

java.awt.image.BufferedImageOp

- BufferedImage = filter(BufferedImage source, BufferedImage dest)

이미지 작업을 소스 이미지에 적용하고 대상 이미지애 결과를 저장한다. 대상이 NULL **면, 새로운 대상 이미지가 생성된다. 대상 이미지는 반환된다.**

java.awt.image.AffineTransformOp

- AffineTransformOp(AffineTransform t, int interpolationType)

affine **변형 연산자를 구성한다.**

전달인자 : t Affine **변형**

interpolationType TYPE_BILINEAR **또는** TYPE_NEAREST_NEIGHBOR **중 하나**

java.awt.image.RescaleOp

- RescaleOp(float a, float b, RenderingHints hints)

리스케일 연산자를 구성한다.

전달인자	a, b	**변형 계수** xnew = ax +b **가 샘플 값에 적용된다**.
	hints	**색상 매칭에서의 랜더링 힌트**; NULL**일 수 있다**.

java.awt.image.LookupOp

- LookupOp(LookupTable table, RenderingHints hints)

lookup **연산자를 구성한다**.

전달인자	table	**샘플 값을 맵핑키 위한 테이블**
	hints	**색상 매칭을 위한 랜더링 힌트** NULL **일수도 있다**.

java.awt.image.ByteLookupTable

- ByteLookupTable(int offset, byte [ ] data)

byte lookup **테이블을 생성한다**.

전달인자	offset	**첫번째 데이터 값이 쓰이는 위치**
	data	**테이블 데이터**

java.awt.image.ConvolveOp

- ConvolveOp(Kernel kernel)
- ConvolveOp(Kernel kernel, int edgeCondition, RenderingHints hints)

convolution **연산자를 구성한다**.

전달인자 :	kernel	convolution **에 대한 커널 행렬**
	edgeCondition	edge **값이 다뤄지는 방법을 명시한다**. : EDGE_NO_OP, EDGE_ZERO_FILL. Edge**값은** convolution**을 연산하기 위해 충분히 인접 값들을 가지고 있지 않기 때문에 특정하게 취급된다**. **디폴트는** EDGE_ZERO_FILL **이다**.
	hints	**색상 매칭시의 랜더링 힌트** : NULL**일 수 있다**.

java.awt.image.Kernel

- Kernel (int width, int height, float [ ] data)

커널을 생성한다.

전달인자:	width, height	**커널 행렬의 너비**
	data	**커널 행렬의 엔트리**.

프린팅

본래의 자바 개발 킷은 전혀 프린팅 기능을 지원하지 않는다. 애플릿에서 프린트하는 것은 불가능하고 응용 프로그램에서 프린트를 할려면 다른 제3의 라이브러리를 이용해야 한다. JDK1.1은 아주 단순한 프린팅 기능을 소개했다. 여러분이 아주 질좋은 프린트 기능을 원하지 않는한 JDK1.1에서 단순한 프린트 출력은 충분하다. 1.1 프린팅 모델은 부라우져 벤터들이 웹 페이지상에 나타나는 애플릿의 외관을 출력할 수 있도록 설계되었다.(그러나 부라우져 벤더들은 이를 수용하지 않는다.)

자바2 프린팅 모델은 2D 그래픽과 완벽하게 통합된 강건한 프린팅 모델의 시작이다. 이 절에서 우리는 여러분에게 단일 시트상에 그려진 것을 쉽게 출력하는 방법을 알려준다. 그러나 여러분은 다중 페이지의 출력을 관리할 수 있으며 자바 2D 이미징 모델에서 제공하는 이점, 그리고 프린트 미리보기 다이얼로그를 쉽게 생성할 수 있다.

노트 : 자바2 플랫폼은 또한 사용자 인터페이스 컴포넌트들의 출력을 지원한다. 우리는 이 주제를 다루지 않는다. 왜냐하면 이 주제는 대부분 부라우져 구현자들과 스트린 감지자들의 관심이기 때문이다. 컴포넌트들을 프린팅하는 방법에 대한 자세한 정보는 다음 URL에서 제공하는 마티 헐의 튜토리얼을 참조하라.

www.apl.jhu.edu/~hall/java/Swing-Tutorial/Swing-Tutorial-Printing.html

---

단일 시트 프린팅

프린트 출력을 생성하기 위해, 여러분은 적어도 두개의 아이템을 필요로 한다.

- Printable 인터페이스를 구현한 객체를 지원할 필요가 있다.
- 프린트 작업을 시작할 필요가 있다.

Printable 인터페이스는 단일 메소드를 가지고 있다.

```
int print(Graphics g, PageFormat format, int page)
```

이 메소드는 프린트 엔진이 프린팅을 위해 페이지를 포맷할 필요가 있을 때마다 호출된다. 여러분의 코드는 그래픽 컨텍스상에 출력할 필요가 있는 텍스트와 이미지를 그린다. 페이지 포맷은 여러분에게 페이퍼 크기와 프린트 마진을 말해준다. 페이지 숫자는 여러분이 렌더할 필요가 있는 페이지가 어느 페이지인지를 말해준다.

프린트 작업을 시작하기 위해, 여러분은 PrinterJob 클래스를 사용한다. 먼저, 여러분은 프린트 작업 객체를 얻기위해 getPrinterJob 정적 메소드를 호출한다. 기본 페이지 포맷을 얻기위

해 defaultPage 메소드를 호출한 다음 setPrintable 메소드를 사용하여 페이지 컨텐트를 렌더하는 Printable 객체를 설정한다.

```
PrinterJob printJob = PrinterJob.getPrinterJob();
pageFormat = printJob.defaultPage();
printJob.setPrintable(printable, pageFormat);
```

주의 : PrintJon 클래스는 JDK1.1 스타일의 프린팅을 처리한다. 이 클래스는 이제 다소 모호하다. PrinterJob 클래스와 혼돈하지 말라.

프린트 작업을 시작하기 전에, printDialog 메소드를 호출하여 프린트 다이얼로그를 출력한다.(그림 7-35 참조) 해당 다이얼로그는 사용자에게 프린트하고자 하는 페이지 범위를 선택할 기회를 제공한다. 더욱 중요한 것은, 프린트 다이얼로그는 사용자에게 프린터 설정을 하게 해준다는 것이다. 프린트 다이얼로그는 프린터 제조업체에서 (색상, 양면 인쇄처럼)고급 프린팅 기능을 설정하는 , 호스트 운영 체제에 보여지도록 작성된 제공하는 커스텀 다이얼로그를 제공하기 때문에 네이티브 다이얼로그이다.

노트 : 만일 자바 플랫폼이 페이지 선택을 위한 자신의 다이얼로그를 제공하고 오직 고유 다이얼로그를 프린터 설정을 위해서만 사용한다면 보다 일반적일 것이다. 이것은 머지않은 미래에 이뤄질 것이다.

그림 7-35 : 프린트 다이얼로그

여러분이 Printable 객체의 프린트 작업을 수행할 때, 프린트 작업은 여러분이 얼마나 많은 페이지를 프린트하길 원하는지 알지 못한다. 이것은 단지 print 메소드를 계속해서 호출한다. Print 메소드가 Printable.PAGE_EXISTS를 반환하는 한 , 프린트 작업은 페이지를 계속해서 생성해 낸다. Print 메소드가 Printable.NO_SUCH_PAGE를 반환할 때, 프린트 작업을 중단한다.

경고 : 프린트 작업이 print 메소드에게 전달하는 페이지 번호들은 0에서부터 시작한다.

그러므로, 프린트 작업은 출력작업이 종료된 후라야 정확한 페이지 숫자를 알게 된다. 이러한 이유 때문에, 프린트 다이얼로그는 1-19999사이의 페이지 범위를 보여준다. 여러분은 다음 절에서 프린트 작업에 Book객체를 제공하는 것에 의해서 이러한 결점을 피하는 방법을 보게 될 것이다.

PrintDialog 메소드는 사용자가 "OK"를 클릭하면 true를 다이얼로그를 취소하면 flase를 반환한다. 만일 사용자가 받아들이면, PrinterJob 클래스의 print 메소드는 프린트 작업을 시작한다. print 메소드는 PrinterException을 발생시킨다. 다음은 프린팅 코드의 윤곽이다.

```
if(ptintJob.printDialog())
{       try
        {       printJob.print();
        }
        catch(PrinterException exception)
        {       …
        }
}
```

프린팅 작업 동안, 프린트 작업은 반복적으로 Printable 객체의 print 메소드를 호출한다. 프린트 작업은 같은 페이지에 대해 다중 호출을 가능하게 한다. 그러므로 print 메소드내에서 페이지들을 세는 게 아니라 항상 페이지 숫자 전달인자에 의존해야 한다. 프린트 작업은 같은 페이지에 대해 반복해서 print 메소드를 호출할 수 있기 때문에 이것은 좋은 이유이다. 도트 프린터와 잉크젝 같은 몇몇 프린터들은 밴딩(banding)을 사용한다. 이들 프린터는 한번에 한 밴드를 프린트한다. 그런 다음 다음 밴드를 프린트 한다. 프린트 작업은 한번에 전체 페이지를 출력하는 레이저 프린터에 대해서 조차도 밴딩을 사용한다-이것은 프린트 작업에게 스풀 파일의 크기를 관리할 방법을 제공해 준다.

만일 프린터 작업이 한 밴드를 출력할 Printable 객체를 필요로 한다면 요청된 밴드에 대한 그래픽 컨텍스의 클립 영역을 설정하고 print 메소드를 호출한다. 그리기 기능은 밴드 사각형에 대해 클립을 할 것이다. 그리고 오직 밴드내에 나타나는 이러한 그리기 항목들을 렌더링할 것이다. 여러분의 print 메소드는 이러한 과정을 알 필요가 없다. 한가지 주의사항: 클립 영역을 방해해서는 안된다.

경고 : 여러분의 print 메소드가 얻는 Graphics 객체는 페이지 마진에 대해 클립한다. 만일 여러분이 클립 영역을 대체하면 마진들의 주변을 그릴 수 있다. 특히, 프린터 그래픽 컨텍스내에서 클릭핑 영역은 보다 제약을 받는다. 만일 여러분이 클립 영역을 제거하면 여러분의

print 메소드를 시작할 때 getClip을 호출해야 한다. 그리고 해당 클립 영역을 복귀해야 한다.

---

print 메소드의 PageFormat 전달인자는 출련된 페이지에 대한 정보를 담고 있다. 다음 메소드들은 포인트(point) 단위의 종이 크기를 반환한다.

getWidth
getHeight

한 포인트는 인치의 1/72 이다. (인치는 25.4 밀리미터이다.) 가령, A4 크기는 거의 595 by 842 포인트이고 US 레터는 612 by 792 포인터이다.

포인터는 미국내 프린팅 거래의 측정 단위이다. 다른 나라에서, 프린팅 패키지는 두가지 목적에서 포인터 단위를 사용한다. 종이 크기와 종이 마진은 포인트로 측정된다. 그리고 모든 포인트 그래픽 컨텍스들을 위한 기본 단위는 1 포인트이다. 여러분은 이 절의 끝에 있는 예제 프로그램에서 이를 검증할 수 있다. 프로그램은 텍스트의 두 라인을 서로 72 유닛 떨어져서 프린트한다. 예제 프로그램을 수행하면 베이스 라인사이의 간격을 측정할 수 있다. 이 간격은 정확이 1인치 또는 25.4 밀리 미터 이다.

PageFormat 클래스의 getWidth와 getHeight 메소드는 여러분에게 완벽한 페이퍼 크기를 제공한다. 모든 페이퍼 영역이 프린트 가능한 것은 아니다. 기본적으로 사용자는 마진을 선택한다. 그리고 비록 선택을 하지 않더라도 프린터들은 다소 출력할 페이퍼의 시트들을 확보할 필요가 있다. 그러므로 가장자리 주변에 프린터되지 않는 작은 영역을 갖고 있다. 다음 메소드들은 여러분에게 여러분이 실제 채울 수 있는 영역의 차원을 말해 준다.

getImageableWidth
getImageableHeight

그러나 마진이 대칭일 필요는 없다, 그래서 여러분은 다음 메소드를 이용해서 그릴수 있는 영역(Imageable area)의 상단 외쪽코너를 알 필요가 있다(그림 7-36참조).

getImageableX
getImageableY

팁 : 여러분이 print 메소드에서 받는 그래픽 컨텐스는 마진을 배제하고 클립한다. 그러나 우

리가 테스트한 JDK 버전은 좌표 측정 시스템의 출발점이 결코 페이퍼의 상단 왼쪽 코너가 아니다. 좌표 측정 시스템을 그릴 수 있는 영역의 상단 외쪽의 코너에서 시작하도록 변환하는 것이 좋다. 여러분의 print 메소드를 다음처럼 손쉽게 시작하라.

```
g.translate(pageFormat.getImageableX() , pageForamt.getImageableY());
```

그림 7-36 : 페이지 포맷 측정

만일 여러분이 여러분의 사용자가 페이지 마진들을 설정할 수 있도록 해주거나 가로 방향이나 세로 방향간의 전환을 선택할 수 있게 해주질 원한다면 PrintJob 클래스의 pageDialog 메소드를 호출할 수 있다. 그림7-37은 윈도우 운영체제를 위한 페이지 포맷 다이얼로그이다.

여러분은 기본 PageFormat 객체를 pageDialog 메소드에 전달한다. 이 메소드는 해당 객체를 복제하고 다이얼로그의 사용자 선택에 따라 이를 변형한다. 그리고 복제된 객체를 반환한다.

```
PageFormat defaultFormat = printJob.defalutPage();
PageForamt selectedFormat = printJob.pageDialog(defaultFormat);
```

노트 : 윈도우상에서 우리사 테스트한 JDK 버전에서, 페이지 포맷 다이얼로그는 pageDialog 메소드에게 전달한 페이지 포맷의 값들을 적절히 초기화하기 못한다. 이것은 향후 개량될 것이다. 또한 JDK의 다음 버전에서는 고유 다이얼로그에 의존하는 대신 페이지 설정을 위한 자신의 다이얼로그를 제공할 것이다.

그림 7-37 : 페이지 셋업 다이얼로그

예제 7-11은 여러분에게 스크린과 출려된 페이지상에 나타나는 일련의 모양들을 렌더링하는 방법을 보여준다. JPanel의 하위 클래스는 Printable 인터페이스를 구현한다. PaintComponent와 print 메소드는 모두 실제 그림그리기를 수행하는 같은 메소드를 호출한다.

```
class PrintPanel extends JPanel implements Printable
{  public void paintComponent(Graphics g)
   {   super.paintComponent(g);
```

```
        Graphics2D g2 = (Graphics2D)g;
        drawPage(g2);
    }

  public int print(Graphics g, PageFormat pf, int page)
        throws PrinterException
   {  if (page >= 1) return Printable.NO_SUCH_PAGE;
        Graphics2D g2 = (Graphics2D)g;
        g2.translate(pf.getImageableX(), pf.getImageableY());
        drawPage(g2);
        return Printable.PAGE_EXISTS;
    }

  public void drawPage(Graphics2D g2)
    {       // shared drawing code goes here
            …
    }
    …
}
```

경고 : JDK1.2는 프린터 그래픽 컨텍스를 설정할 때 검정에 대한 포인트를 설정하지 않는다. 만일, 여러분의 페이지가 모두 흰색이면 print 메소드의 상단에 gc.setPaint(Color.black) 라인을 추가하라. JDK1.3에서 이 문제는 해결되었다.

이 예제는 예제 7-6처럼 같은 이미지를 출력하고 프린트한다. 즉, 라인들의 패턴을 위한 클리핑 영역으로 사용되어지는 "Hello, World" 메시지의 윤곽이다. (그림 7-23)

프린팅을 시작하기 위해 프린트 버튼을 클릭하거나 페이지 셋업 다이얼로그르르 가져오기 위해 페이지 셋업 버튼을 클릭한다. 예제 7-11은 코드를 보여준다.

경고 : 이 프로그램은 그릴 수 있는 영역 주변에 사각형을 프린트한다. 우리는 몇몇 프린터들 상에서 심지어 동일한 마진을 명시했는 데도 불구하고 그릴수 있는 영역이 적절히 중간에 위치하지 않는다는 것을 발견했다. 우리는 프린팅 메커니즘이 적합한 마진 주변에 프린

트가 불가능한 영역을 고려하는 데 실패했다.

예제 7-11 : PrintTest.java

```
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.awt.print.*;
import java.util.*;
import javax.swing.*;

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

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

      Container contentPane = getContentPane();
      canvas = new PrintPanel();
      contentPane.add(canvas, "Center");

      JPanel buttonPanel = new JPanel();
```

```
    printButton = new JButton("Print");
     buttonPanel.add(printButton);
     printButton.addActionListener(this);

    pageSetupButton = new JButton("Page setup");
     buttonPanel.add(pageSetupButton);
     pageSetupButton.addActionListener(this);

     contentPane.add(buttonPanel, "North");
 }

public void actionPerformed(ActionEvent event)
{   Object source = event.getSource();
    if (source == printButton)
    {   PrinterJob printJob = PrinterJob.getPrinterJob();
        if (pageFormat == null)
            pageFormat = printJob.defaultPage();
         printJob.setPrintable(canvas, pageFormat);
         if (printJob.printDialog())
         {   try
             {   printJob.print();
             }
            catch (PrinterException exception)
             {   JOptionPane.showMessageDialog(this, exception);
             }
         }
     }
    else if (source == pageSetupButton)
    {   PrinterJob printJob = PrinterJob.getPrinterJob();
        if (pageFormat == null)
            pageFormat = printJob.defaultPage();
        pageFormat = printJob.pageDialog(pageFormat);
     }
 }

private JButton printButton;
```

```
    private JButton pageSetupButton;

    private PrintPanel canvas;
    private PageFormat pageFormat;
}

class PrintPanel extends JPanel
    implements Printable
{   public void paintComponent(Graphics g)
    {    super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
         drawPage(g2);
    }

   public int print(Graphics g, PageFormat pf, int page)
        throws PrinterException
   {  if (page >= 1) return Printable.NO_SUCH_PAGE;
        Graphics2D g2 = (Graphics2D)g;
         g2.setPaint(Color.black);
        g2.translate(pf.getImageableX(), pf.getImageableY());
        g2.draw(new Rectangle2D.Double(0, 0,
             pf.getImageableWidth(), pf.getImageableHeight()));

         drawPage(g2);
        return Printable.PAGE_EXISTS;
    }

   public void drawPage(Graphics2D g2)
   {   FontRenderContext context = g2.getFontRenderContext();
       Font f = new Font("Serif", Font.PLAIN, 72);
       GeneralPath clipShape = new GeneralPath();

         TextLayout layout = new TextLayout("Hello", f, context);
        AffineTransform transform
           = AffineTransform.getTranslateInstance(0, 72);
       Shape outline = layout.getOutline(transform);
```

```
        clipShape.append(outline, false);

        layout = new TextLayout("World", f, context);
        transform
           = AffineTransform.getTranslateInstance(0, 144);
        outline = layout.getOutline(transform);
        clipShape.append(outline, false);

        g2.draw(clipShape);
        g2.clip(clipShape);

        final int NLINES =50;
      Point2D p = new Point2D.Double(0, 0);
      for (int i = 0; i < NLINES; i++)
      {  double x = (2 * getWidth() * i) / NLINES;
         double y = (2 * getHeight() * (NLINES - 1 - i))
               / NLINES;
         Point2D q = new Point2D.Double(x, y);
         g2.draw(new Line2D.Double(p, q));
       }
    }
}
```

java.awt.print.Printable

- int print(Graphics g, PageFormat format, int pageNumber)

  한 페이지를 렌더하고 PAGE_EXISTS**를 반환하거나** NO_SUCH_PAGE**를 반환한다**.

전달인자		
	g	**렌더된 페이지에 대한 그래픽 컨텍스**
	format	**그릴 페이지의 포맷**
	pageNumber	**요청한 페이지 숫자**

java.awt.print.PrinterJob

- static PrinterJob getPrinterJob()

  프린터 작업 객체를 반환한다.

- PageFormat defaultPage()

  해당 프린터를 위한 기본 페이지 포맷을 반환한다.

- PageFormat pageDialog(PageFormat defaults)

  페이지 설정 다이얼로그를 출력하고 다이얼로그에서 사용자가 요청한 변화를 반영한 defaults의 복제를 반환한다.
- boolean printDialog()

  프린트할 페이지를 선택할 기회를 사용자에게 제공하는 프린트 다이얼로그를 생성한다. 그리고 호스트 환경의 프린터 설정을 변경한다. 만일 사용자가 다이얼로그를 승낙하면 true를 반환한다.
- void setPrintable(Printable p)
- void setPrintable(Printable p, PageFormat format)

  해당 프린터 작업의 Printable과 선택사항으로 페이지 포맷을 설정한다.
- void paint()

  반복적으로 print 메소드를 호출하여 현재의 Printable을 프린트하고 더 이상 이용할 수 있는 페이지가 없을 때까지 프린터에게 렌더된 페이지를 보낸다.

java.awt.print.PageFormat

- double getWidth()
- double getHeight()

  페이지의 폭과 넓이를 반환한다.
- double getImageableWidth()
- double getImageableHeight()

  페이지의 그릴 수 있는 영역의 폭과 높이를 반환한다.
- double getImageableX()
- double getImageableY()

  그릴 수 있는 영역의 최상단 왼쪽 코너 위치를 반환한다.
- int getOrientation()

  PORTRAIT,LANDSCAPE,REVERSE_LANDSCAPE 중의 하나를 반환한다. 페이지 방향은 프로그래머에게 투명하다. 왜냐하면 페이지 포맷과 그래픽 컨텍스 설정은 자동적으로 페이지 방향을 반영하기 때문이다.

다중 페이지 프린팅

특히, 여러분은 대개 가공되지 않은 Printable 객체를 프린터 작업에게 전달하길 원하지 않는다. 대신 여러분은 Pageable 인터페이스를 구현한 클래스의 객체를 얻을 수 있다. 자바 플랫폼은 Book이라 불리는 이 같은 하나의 클래스를 제공한다. 책(book)은 절(section)들로 이루

어져 있다. 각 각의 절들은 Printable이다. 여러분은 Printable 객체와 그들의 페이지 숫자를 추가함으로써 책을 만들 수 있다.

```
Book book = new Book();
Printable coverpage = . . .;
Printable bodypage = . . .;
book.append(coverpage, pageFormat); // append 1 page
book.append(bodypage, pageFormat, pageCount);
```

그런 다음, 여러분은 Book 객체를 프린터 작업에게 전달하기 위해 setPageable 메소드를 사용한다.

```
printJob.setPageable(book);
```

이제 프린트 작업은 정확히 얼마나 많은 페이지들을 출력할 지 알고 있다. 그러면, 프린트 다이얼로그는 정확한 페이지 범위를 출력하고 사용자는 전제 범위나 일부 범위를 선택할 수 있다.

주의 : 프린터 작업이 Printable 절의 print 메소드를 호출할 때, 현재 페이지 숫자로서 책의 각 절이 아니라 현재 페이지 숫자를 전달한다. 이것은 큰 고통이다 – 각 절은 페이지 숫자 전달인자를 알기 위해 앞선 절들의 페이지 숫자를 알고 있어야 한다. 프린팅 인터페이스의 몇가지 다른 이상한 점들은 자바 디자이너들이 프린팅 이슈들에 대해 보다 자세히 이해함에 따라 인터페이스의 이러한 행위도 변할 것이다.

프로그래머의 관점에서 보면, Book 클래스를 사용하는 데 있어 가장 큰 도전은 프린트할 때 각 절에 얼마나 많은 페이지가 있는 지 여러분이 알 필요가 있다는 것이다. 여러분의 Printable 클래스는 출력된 페이지들상에 레이아웃을 계산하는 *레이아웃 알고리즘*을 필요로 한다. 프린팅을 시작하기 전에, 페이지 브레이크와 페이지 카운터를 계산하는 알고리즘을 호출한다. 여러분은 프린팅 과정 동안 이를 편하게 하기 위해 레이아웃 정보를 유지할 수 있다.

여러분은 사용자가 페이지 포맷을 변경할 가능성에 대한 주의를 기울여야 한다. 만일 그렇다면, 여러분은 비록 프린트하길 원하는 정보가 변하지 않더라도 레이아웃을 다시 계산해야 한다.

예제7-12는 다중 페이지 출력물을 생산하는 방법을 보여준다. 이 프로그램은 많은 페이지들 상에 큰 문자들의 메시지를 출력한다(그림7-38 참조). 여러분은 마진들을 정렬하고 베너를 구성하기 위해 페이지들을 함께 붙일 수 있다.

그림 7-38 : 베너

Banner 클래스의 layoutPages 메소드는 레이아웃을 계산한다. 우리는 먼저 72 포인트 폰트로 메시지 문자열을 레이아웃한다. 그런 다음, 우리는 결과 문자열의 높이를 계산하고 이 결과를 그릴 수 있는 페이지의 높이와 비교한다. 우리는 이러한 두가지 측정치로부터 스케일 요소를 고안해 낸다. 문자열을 프린팅할 때, 우리는 스케일 요소에 의해서 이를 확대한다.

경고 : 여러분의 정보를 정교하게 레이아웃하기 위해, 여러분은 일반적으로 프린터 그래픽 컨텍스에 접근할 필요가 있다. 불행스럽게도 프린팅이 실제 시작할 때까지 해당 그래픽 컨텍스를 얻을 방법이 없다. 예제 프로그램에서, 우리는 스크린 그래픽 컨텍스를 다룬다. 그리고 스크린의 폰트 메트릭스와 프린터가 일치하길 기대한다.

Banner 클래스의 getPageCount 메소드는 첫번째로 레이아웃 메소드를 호출한다. 그런 다음 문자열의 넓이를 재조정하고 이것을 각 페이지의 그릴 수 있는 넓이로 나눈다. 반올림한 나눈 몫이 바로 페이지 카운터이다.

문자들이 다중 페이지들 상에 걸쳐 쪼개져 있기 때문에 배너를 프린트하기는 어렵다. 그러나 자바 2D API의 기능 덕분에 이것은 전혀 어려운 문제가 아니라는 것이 판명되었다. 특정 페이지가 요청됐을 때, 우리는 Graphics2D 클래스의 translate 메소드를 사용하여 손쉽게 문자열의 최상단 왼쪽 코너를 왼쪽으로 이동한다. 그러면, 우리는 현재 페이지와 동일한 사각형 클립을 설정한다.(그림 7-39 참조). 마지막으로 , 우리는 그래픽 컨텍스를 레이아웃 메소드가 계산한 스케일 요소로 크기를 재조정한다.

그림 7-39 : 베너의 페이지 프린팅

이 예제는 변형(transformation)의 강력함을 보여준다. 그려지는 코드는 단순하게 유지되고 변형은 적합한 장소에 그림을 위치시키는 모든 일을 수행한다. 마지막으로, 클립은 페이지의 외부에 떨어져 있는 이미지의 일부를 잘라낸다. 다음 절에서 여러분은 프린트 미리보기를

출력하기 위한 또다른 변형의 사용을 보게 될 것이다.

프린트 미리보기

대부분의 전문 프로그래머들은 원하지 않는 출력으로 인해 종이를 낭비하지 않기 위해 화면상에 여러분의 페이지를 보게 해주는 프린트 미리보기 메커니즘을 가지고 있다. 자바 플랫폼의 프린팅 클래스들은 표준 " 프린트 미리보기" 다이얼로그를 지원하지 않는다. 그러나 여러분 자신의 클래스를 손쉽게 디자인할 수 있다 (그림 7-40 참조). 이번 절에서, 우리는 여러분에게 이 방법을 보여준다. 예제 7-12의 PrintPreviewDialog 클래스는 완전히 기본적(Generic)이다. 여러분은 어떤 종류의 출력에 대해서도 이 클래스를 재 사용하여 미리보기를 할 수 있다.

PrintPreviewDialog를 생성하기 위해 , 여러분은 PageFormat객체와 더불어 Printable이나 Book을 제공해야 한다. 해당 다이얼로그의 표면은 PrintPreviewCanvas를 포함하고 있다. 여러분이 페이지들에 걸쳐 "Next"와 "Previous"버튼을 사용할 때, paintComponent 메소드는 요청된 페이지를 위해 Printable 객체의 print 메소드를 호출한다.

일반적으로, print 메소드는 프린터 그래픽 컨텍스상에 페이지 컨텍스를 그린다. 그러나 우리는 조그만한 스크린 사각형내부에 프린트된 페이지 전체가 맞도록 하기 위해 적절히 규모조정된 스크린 그래픽 컨텍스를 전달한다.

```
float xoff = . . . ; // left of page
float yoff = . . . // top of page
float scale = . . . to fit printed page onto screen
g2.translate(xoff, yoff);
g2.scale(scale, scale);
Printable printable = book.getPrintable(currentPage);
Printable.print(g2, pageFormat, currentPage);
```

print 메소드는 결코 실제 생산해 내는 페이즈를 알아야 만 한다. 스크린상에 프린트 미리보기를 만들어냄으로서 그래픽 컨텍스상에 손쉽게 그린다. 이것은 아주 설득력있는 자바 2D 이미징 모델의 기능을 보여준다.

그림 7-40 : 프린트 미리보기 다이얼로그

예제 7-12**는 배너 프린팅 프로그램과 프린트 미리보기 다이얼로그를 위한 코드를 포함하고 있다. 텍스트 필드내에** "Hello, World!"**를 입력하고 프린트 미리보기를 보면 배너를 출력할 수 있다.**

예제 7-12 : BookTest.java

```
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.awt.print.*;
import java.util.*;
import javax.swing.*;

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

class BookTestFrame extends JFrame
   implements ActionListener
{  public BookTestFrame()
   {  setTitle("BookTest");
      setSize(300, 100);
      addWindowListener(new WindowAdapter()
         {  public void windowClosing(WindowEvent e)
            {  System.exit(0);
            }
         } );

      Container contentPane = getContentPane();
      text = new JTextField();
      contentPane.add(text, "South");
```

```
    JPanel buttonPanel = new JPanel();

    printButton = new JButton("Print");
      buttonPanel.add(printButton);
      printButton.addActionListener(this);

    pageSetupButton = new JButton("Page setup");
      buttonPanel.add(pageSetupButton);
      pageSetupButton.addActionListener(this);

    printPreviewButton = new JButton("Print preview");
      buttonPanel.add(printPreviewButton);
      printPreviewButton.addActionListener(this);

     contentPane.add(buttonPanel, "North");
 }

public Book makeBook()
{   if (pageFormat == null)
    {   PrinterJob printJob = PrinterJob.getPrinterJob();
        pageFormat = printJob.defaultPage();
     }
    Book book = new Book();
    String message = text.getText();
    Banner banner = new Banner(message);
     int pageCount
         = banner.getPageCount((Graphics2D)getGraphics(),
             pageFormat);
   book.append(new CoverPage(message + " (" + pageCount
        + " pages)"), pageFormat);
     book.append(banner, pageFormat, pageCount);
     return book;
 }

 public void actionPerformed(ActionEvent event)
{   Object source = event.getSource();
```

```
        if (source == printButton)
        {   PrinterJob printJob = PrinterJob.getPrinterJob();
            printJob.setPageable(makeBook());
            if (printJob.printDialog())
            {   try
                {   printJob.print();
                }
                catch (Exception exception)
                {   JOptionPane.showMessageDialog(this, exception);
                }
            }
        }
        else if (source == pageSetupButton)
        {   PrinterJob printJob = PrinterJob.getPrinterJob();
            if (pageFormat == null)
                pageFormat = printJob.defaultPage();
            pageFormat = printJob.pageDialog(pageFormat);
        }
        else if (source == printPreviewButton)
        {   PrintPreviewDialog dialog
                = new PrintPreviewDialog(makeBook());
            dialog.show();
        }
    }

    private JButton printButton;
    private JButton pageSetupButton;
    private JButton printPreviewButton;

    private JTextField text;
    private PageFormat pageFormat;
}

class Banner implements Printable
{   public Banner(String m)
    {   message = m;
```

```
 }

public int getPageCount(Graphics2D g2, PageFormat pf)
 {  if (message.equals("")) return 0;
     layoutPages(g2, pf);
    float width = scale * layout.getAdvance();
    int pages = (int)Math.ceil(width / pf.getImageableWidth());
     return pages;
 }

public int print(Graphics g, PageFormat pf, int page)
     throws PrinterException
 {  Graphics2D g2 = (Graphics2D)g;
     g2.setPaint(Color.black);
    if (page > getPageCount(g2, pf))
         return Printable.NO_SUCH_PAGE;
     g2.translate(pf.getImageableX(), pf.getImageableY());

     drawPage(g2, pf, page);
     return Printable.PAGE_EXISTS;
 }

 public void layoutPages(Graphics2D g2, PageFormat pf)
 {  if (message.equals("")) return;
    FontRenderContext context = g2.getFontRenderContext();
    Font f = new Font("Serif", Font.PLAIN, 72);
    layout = new TextLayout(message, f, context);
     float ascent = layout.getAscent();
     float descent = layout.getDescent();
    float height = ascent + descent;
    scale = (float)pf.getImageableHeight() / height;
 }

public void drawPage(Graphics2D g2, PageFormat pf, int page)
 {  if (message.equals("")) retum;
     page--; // account for cover page
```

```
        layoutPages(g2, pf);

        drawCropMarks(g2, pf);
        g2.clip(new Rectangle2D.Double(0, 0,
            pf.getImageableWidth(), pf.getImageableHeight()));
         g2.translate(-page * pf.getImageableWidth(), 0);
        g2.scale(scale, scale);
        AffineTransform transform
            = AffineTransform.getTranslateInstance(0,
                layout.getAscent());
        Shape outline = layout.getOutline(transform);
         g2.draw(outline);
    }

   public void drawCropMarks(Graphics2D g2, PageFormat pf)
   {  final double C = 36; // crop mark length = 1/2 inch
      double w = pf.getImageableWidth();
      double h = pf.getImageableHeight();
      g2.draw(new Line2D.Double(0, 0, 0, C));
       g2.draw(new Line2D.Double(0, 0, C, 0));
      g2.draw(new Line2D.Double(w, 0, w, C));
      g2.draw(new Line2D.Double(w, 0, w - C, 0));
      g2.draw(new Line2D.Double(0, h, 0, h - C));
      g2.draw(new Line2D.Double(0, h, C, h));
      g2.draw(new Line2D.Double(w, h, w, h - C));
       g2.draw(new Line2D.Double(w, h, w - C, h));
    }

   private String message;
   private float scale;
   private float width;
   private TextLayout layout;
}

class CoverPage implements Printable
{  public CoverPage(String t)
```

```
    {   title = t;
    }

    public int print(Graphics g, PageFormat pf, int page)
        throws PrinterException
    {   if (page >= 1) return Printable.NO_SUCH_PAGE;
        Graphics2D g2 = (Graphics2D)g;
        g2.setPaint(Color.black);
        g2.translate(pf.getImageableX(), pf.getImageableY());
        FontRenderContext context = g2.getFontRenderContext();
        Font f = g2.getFont();
        TextLayout layout = new TextLayout(title, f, context);
        float ascent = layout.getAscent();
        g2.drawString(title, 0, ascent);
        return Printable.PAGE_EXISTS;
    }

    private String title;
}

class PrintPreviewDialog extends JDialog
    implements ActionListener
{   public PrintPreviewDialog(Printable p, PageFormat pf,
        int pages)
    {   Book book = new Book();
        book.append(p, pf, pages);
        layoutUI(book);
    }

    public PrintPreviewDialog(Book b)
    {   layoutUI(b);
    }

    public void layoutUI(Book book)
    {   setSize(200, 200);
```

```
    Container contentPane = getContentPane();
    canvas = new PrintPreviewCanvas(book);
     contentPane.add(canvas, "Center");

    JPanel buttonPanel = new JPanel();

    nextButton = new JButton("Next");
     buttonPanel.add(nextButton);
     nextButton.addActionListener(this);

    previousButton = new JButton("Previous");
     buttonPanel.add(previousButton);
     previousButton.addActionListener(this);

    closeButton = new JButton("Close");
     buttonPanel.add(closeButton);
     closeButton.addActionListener(this);

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

public void actionPerformed(ActionEvent event)
{   Object source = event.getSource();
    if (source == nextButton)
      {   canvas.flipPage(1);
      }
    else if (source == previousButton)
      {   canvas.flipPage(-1);
      }
    else if (source == closeButton)
      {   setVisible(false);
      }
 }

private JButton nextButton;
private JButton previousButton;
```

```
    private JButton closeButton;
    private PrintPreviewCanvas canvas;
}

class PrintPreviewCanvas extends JPanel
{   public PrintPreviewCanvas(Book b)
    {   book = b;
        currentPage = 0;
     }

    public void paintComponent(Graphics g)
     {   super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        PageFormat pageFormat = book.getPageFormat(currentPage);

        double xoff; // x offset of page start in window
       double yoff; // y offset of page start in window
       double scale; // scale factor to fit page in window
        double px = pageFormat.getWidth();
        double py = pageFormat.getHeight();
        double sx = getWidth() - 1;
        double sy = getHeight() - 1;
       if (px / py < sx / sy) // center horizontally
        {   scale = sy / py;
           xoff = 0.5 * (sx - scale * px);
            yoff = 0;
         }
        else // center vertically
        {   scale = sx / px;
             xoff = 0;
           yoff = 0.5 * (sy - scale * py);
         }
        g2.translate((float)xoff, (float)yoff);
        g2.scale((float)scale, (float)scale);

       // draw page outline (ignoring margins)
```

```
        Rectangle2D page = new Rectangle2D.Double(0, 0, px, py);
          g2.setPaint(Color.white);
          g2.fill(page);
          g2.setPaint(Color.black);
          g2.draw(page);

         Printable printable = book.getPrintable(currentPage);
          try
         {   printable.print(g2, pageFormat, currentPage);
          }
          catch (PrinterException exception)
         {   g2.draw(new Line2D.Double(0, 0, px, py));
             g2.draw(new Line2D.Double(0, px, 0, py));
          }
      }

    public void flipPage(int by)
    {   int newPage = currentPage + by;
        if (0 <= newPage && newPage < book.getNumb erOfPages())
         {   currentPage = newPage;
              repaint();
          }
      }

    private Book book;
    private int currentPage;
}
```

java.awt.print.PrinterJob

- void setPageable(pageable p)
  (Book**처럼**) **프린트될** Pageable**을 설정한다**.

java.awt.print.Book

- void append(Printable p, PageFormat format)
- void append(Printable p, PageFormat format, int pageCount)
  해당 Book**에 절**(section)**을 추가한다**. **만일 페이지 카운트가 명시되지 않으면 첫번째 페이지가 추가된다**.

- Printable getPrintable(int page)

  명시된 페이지를 위한 Printable을 얻는다.

클립보드

(윈도우나 X 윈도우 시스템 같은) 가장 강력하고 편리한 그래픽 사용자 인터페이스 환경중 하나가 잘라 붙이기(Cut and paste)이다. 여러분은 하나의 프로그램에서 몇몇 데이터를 선택하고 이를 클립보드에 잘라내거나 복사할 수 있다. 그런 다음, 여러분은 다른 프로그램을 선택하고 클립보드 컨텐트를 해당 응용 프로그램에 붙일 수 있다. 클립보드를 사용하여, 여러분은 텍스트,이미지 또는 다른 데이터를 하나의 문서에서 다른 문서로 보내거나 같은 문서내의 한 장소에서 다른 장소로 보낼 수 있다.

그러나 JDK1.0에서는 잘라 붙이기를 지원하지 않는다. 여러분은 자바 응용프로그램간의 잘라 붙이기를 할 수 없다. 가령, 만일 여러분이 자바 언어로 작성된 부라우져를 갖고 있다면 웹 페이지로부터 텍스트와 이미지를 복사할 수 없다. 그리고 이들을 자바 기술에 기반한 워드 프로세스로 전송할 수 없다. JDK1.1은 기초적인 클립보드 메커니즘을 구현하였다. 이 메커니즘이 이 절의 주제이다(자바2 플랫폼에서 문서적으로 향상되지 않았다).

비록 클립보드가 개념적으로 단순하더라도, 클립보드 서비스 구현은 실제 여러분이 생각하는 것보다 더 어렵다. 여러분이 워드프로세스로부터 텍스트를 클립보드로 복사한다고 가정해 보자. 만일 여러분이 해당 텍스트를 다른 워드 프로세스로 붙인다면 폰트들과 포맷팅이 영향받지 않기를 기대한다. 즉, 클립보드내에 텍스트는 포맷팅 정보를 간직할 필요가 있다. 그러나 만일 여러분이 텍스트를 일반 텍스트 필드에 붙인다면, 추가적인 포맷팅 코드없이 단지 문자들을 붙이면 된다.
그래픽 사용자 인터페이스 환경의 시스템 클립보드는 다중 포맷으로 클립보드 데이터를 유지할 수 있다. 그리고 데이터를 제공하는 프로그램은 데이터를 소비하는 프로그램과 데이터 포맷에 대해 협상해야 한다.

더욱이, 일반적으로 개별 프로그램은 그들 자신의 로컬 클립보드를 가지고 있다. 여러분이 텍스트를 워드 프로세서내에 잘라 붙일 때, 워드 프로세스는 데이터를 로컬 클립보드내에 자신의 포맷으로 저장한다. 포맷된 텍스트를 교환하기 위한 전체 표준은 없다. 텍스트를 시스템 클립보드상에 붙일 때, 일반적으로 몇몇 포맷팅 정보는 잃어 버린다. 이러한 손실은 하나의 워드 프로세스에서 다른 워드 프로세스로 텍스트 교환을 할 경우에는 참을 만 하지만 하나의 워드 프로세스내에서 데이터를 이동하는 데에는 이러한 손실이 받아들여 지지 않는다. 프로그램 사용자는 문서의 한 곳에서 같은 문서내의 다른 곳으로 정보를 이동할 때 잃어버리는 정보가 없기를 기대한다. 프로그램내에서 완벽한 정보 전송을 이루기 위해, 로컬 클립보드는 클립보드내에 저장될 수 있는 데이터보다 풍부한 정보를 요구한다.

마이크로소프트 윈도우,OS2, 매킨토시의 시스템 클립보드 구현은 유사하지만 약간은 다르다. 그리고 X 윈도우 시스템 클립보드 메커니즘은 보다 제한적이다-일반 텍스트의 잘라 붙이기만을 지원한다. 이러한 차이는 자바 프로그래밍 언어로 작성된 프로그램들의 주요한 해결 과제이다. 이 들 프로그램은 많은 플랫폼들상에서 변경되지 않고 작동된다. 종종, 자바 기술은 최소 공통 분모를 제공해야 한다, 종종 최소 공통 분모는 X 윈도우 시스템이다. X 윈도우 시스템은 클립보드상에 그래픽을 저장하기 위한 표준 메소드가 없기 때문에, 자바 프로그래밍 언어에서 프로그램은 현재 오직 시스템 클립보드상에 텍스트만 저장할 수 있다. 여러분은 이 장에서 프로그램에서 시스템 클립보드상에 텍스트를 위치시키는 방법과 다른 프로그램에서 해당 데이터를 추출하는 방법을 배우게 될 것이다.

그러나, 하나의 자바 프로그램에서 다른 프로그램으로 데이터를 복사할 때, 여러분은 일반 텍스트 제약을 극복할 수 있다. 만일, 여러분이 그래픽 이미지를 전자우편 메시지에 추가한다면 무슨 일이 발생할지 고려해 보자. 첨부는 MIME(Multipurpose Internet Mail Extension) 표준을 사용하여 자동적으로 텍스트 형태로 인코딩된다.

노트 : MIME 형태를 정의한 RFC(Request for Comment)의 HTML 버전은 다음 URL을 참조하라. http://www.oac.uci.edu/indiv/ehood/MIME

MIME을 지원하는 메일러들은 첨부를 일반 텍스트로 인코딩하는 방법과 MIME 포맷으로 인코딩하는 방법, 그리고 사용자에게 첨부를 제공하는 방법을 알고 있다. 물론, 여러분이 (본래의 유닉스 mail 프로그램 같은) 과거의 메일 리더를 사용한다면 첨부들은 이상하게 보이는 텍스트로 보여질 것이다.

자바 프로그래밍 언어에서 인코딩된 데이터를 클립보드에 위치시킴으로서 한 프로그램에서 다른 프로그램으로 데이터를 전송하기 위해 이와 같은 접근 방법을 사용하는 것이 가능하다. 물론, 이러한 인코딩에 대해 알지 못하는 프로그램에 텍스트를 붙여 넣을 수 없다. 그러나 목표 프로그램이 인코딩 포맷을 설명하는 헤더를 이해할 수 있다면 정보를 디코딩할 수 있다. 현재, 자바 기술은 이러한 전송 메소드를 지원하지 않는다. 비록 MIME 데이터 타입이 데이터 포맷을 설명할 지라도, 데이터 전송 메커니즘은 현재 데이터를 인코딩하기 위해 전체 MIME 표준을 사용하지 않는다. 우리는 해당 메커니즘을 확장하고 클립보드 데이터를 MIME 포맷으로 인코딩하는 것이 합당하다는 것을 발견했다. 우리는 여러분에게 이러한 접근 방법을 사용함으로서 시스템 클립보드를 통해 자바 응용 프로그램들 간의 임의의 연속화 객체들을 쉽게 교환하는 방법을 보여주는 예제 프로그램을 제공한다.

마지막으로, 자바 데이터 전송 API는 임의의 자바 객체들을 저장할 수 있도록 로컬 클립보드를 지원한다. 본래, 프로그램들간에 데이타를 전송하는 것은 그다지 어렵지 않다. 무슨 이유에서 쉽게 전역변수에 클립보드 데이타를 저장하지 못하는 것일까? 자바 메커니즘은 또한 데이터 포맷 협정(negotiation)을 지원한다.(자바 데이터 전송 API에서는 데이터 포맷을 플래버(flavor)라 부른다.)

표7-3은 새로운 클립보드 메커니즘의 데이터 전송 능력을 요약한 표이다.

표7-3 : 자바 데이터 전송 메커니즘의 능력

전송	포맷
자바 프로그래밍 언어로 작성된 프로그램과 고유 프로그램간	텍스트
자바 프로그래밍 언어로 작성된 두개의 협력하는 프로그램간	인코딩된 텍스트 데이터
자바 프로그래밍 언어로 작성된 하나의 프로그램 내부	임의의 객체

데이터 전송을 위한 클래스들과 인터페이스들

자바 기술에서 데이터 전송은 java.awt.datatransfer 라 불리는 패키지에 구현되어 있다. 표7-4에는 이 패키지의 일부에 대한 간력한 설명이다.

표7-4 : java.awt.datatransfer 패키지의 클래스들

Transferable 인터페이스	전송될 수 있는 객체들은 이 인터페이스를 구현해야 한다.
Clipboard 클래스	클립보드를 갭슐화한 클래스. 전송가능한 객체는 클립보드에 넣고 빠질 수 있는 유일한 아이템이다. 시스템 클립보드는 Clipboard의 구체적인 예제이다.
ClipboardOwner 인터페이스	클립보드에 데이터를 복사할 수 있게 되길 원하는 클래스는 이 인터페이스를 구현해야 한다.
DataFlavor 클래스	클립보드에 위치한 데이터의 타입을 식별하기 위한 방법
StringSelection 클래스	Transferable 을 구현한 자바 1.1에서 제공하는 유일한 구체적 클래스, 편하게는 ClipboardOwner 이다. 텍스트 문자

	열을 전송하는 데 사용한다.
UnsupportedFlavorException	해당 플레버에서 여러분에게 데이터를 제공할 수 없을 때 Transferable에 의해서 발생한다.

텍스트 전송

데이터 전송 클래스들을 손쉽게 익히는 최선의 방법은 지원되는 가장 단순한 상황으로 시작하는 것이다 : 시스템 클립보드로 텍스트를, 시스템 클립보드로 부터 텍스트를 전송. 다음 프로그램의 아이디어는 단순한다. 먼저 시스템 클립보드에 대한 참조를 얻는다.

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

문자열을 클립보드로 전송하기 위해, 문자열들을 StringSelection 객체로 포장할 필요가 있다. 생성자는 여러분이 전송하길 원하는 텍스트를 받는다.

```
StringSelection selection = new StringSelection(text);
```

실제 전송은 전달인자로 StringSelection 객체와 ClipBoardOwner를 취하는 setContents를 호출하는 것에 의해서 이뤄진다. 만일 여러분이 클립보드 소유자를 알려주길 원하지 않는다면 두번째 전달인자를 null로 설정하면 된다.

```
Clipboard.setContents(selection, null);
```

단순한 텍스트에 대해, 여러분은 클립보드 소유자를 필요로 하지 않는다. 클립보드 소유자는 복잡한 데이터의 "지연 포맷팅(delayed formatting)"을 가능하게 한다. 만일 하나의 프로그램이 (문자열 같은) 단순한 데이터를 전송한다면, 단순히 클립보드 컨텐트를 설정하고 다음 작업으로 이동하면 된다. 그러나 만일, 프로그램이 다중 플레버로 포맷팅될 수 있는 복잡한 데이터를 클립보드에 두고 싶다면, 실제 모든 플레버들을 준비하길 원치 않을 것이다. 왜냐하면 결코 이들 모두가 실제 필요하지 않기 때문이다. 그러나, 플레버들을 클립보드 데이터에 남겨둘 필요가 있다. 왜냐하면 나중에 요청받았을 때 플래버들을 생성할 수 있도록 하기 위해서 이다. 클립보드의 내용이 변경됐을 때 (lostOwnership 메소드를 호출함으로서) 클립보드 소유자는 이를 통보받는다.

우리의 예제 프로그램에서, 우리는 클립보드 소유자에 대해 걱정할 필요가 없다.

다음은 한 문자열을 시스템 클립보드에 위치시키기 위한 일련의 완전한 지시사항이다.

```
String text = . . .;
StringSelection selection = new StringSelection(text);
Clipboard.setContents(selection, null);
```

반대로 클립보드로부터 한 문자열을 읽는 기능에 대해 살펴보자. 전송가능한 객체를 얻기위해 클립보드의 getContents 메소드를 호출한다.

```
Transferable contents = clipboard.getContents(this);
```

getContents의 전달인자는 null 일 수 있다. 이 값은 클립보드가 비어있거나 자바 플랫폼이 추출하는 방법을 알고 있는 데이터가 없다는 것임을 알려준다.

Transferable 객체는 여러분에게 어느 플레버 클립보드 정보가 이용가능한지 알려 줄 수 있다. 우리는 나중에 보다 자세히 플레버를 찾는 방법에 대해 논의할 것이다. 지금까지, 우리는 오직 DataFlavor.stringFlavor라 불리는 표준 플레버를 원했다. 여러분은 isDataFlavorSupported 메소드를 사용하여 해당 플레버가 실제 이용하길 원하는 것인지 발견할 수 있다.

마지막으로, 여러분은 원하는 플레버들을 getTransferData 메소드에 전달한다. 이 메소드 호출은 try 블록내에 포함되어야 한다. 왜냐하면 요청받은 플레버가 이용할 수 없다면 UnsupportedFlavorException을 발생시키거나 데이터를 읽을 수 없다면 IOException을 발생시키기 때문이다.

```
DataFlavor flavor = DataFlavor.stringFlavor;
if(selection.isDataFlavorSupported(flavor))
{       try
        {       String text = (String)(selection.getTransferData(flavor);
                do something with text;
        }
        catch(Exception  e)
        {       …
        }
}
```

예제 7-13은 자바 응용 프로그램과 시스템 클립보드간의 잘라 붙이기 데모 프로그램이다. 그림 7-41은 스크린 샷을 보여준다. 만일 여러분이 텍스트 영역내에 텍스트를 선택하고 Copy를 선택하면 선택된 덱스트는 시스템 클립보드로 복사된다. 그림 7-42가 보여주듯이, 복

사된 텍스트는 진정으로 시스템 클립보드에 저장된다. 계속해서 Paste 버튼을 선택할 때, 클립보드의 내용들은 커져 위치에 붙여진다.

그림 7-41 : TextTransferTest 프로그램

그림 7-42 : 복사한 후의 윈도우 클립보드 뷰어

예제 7-13: TextTransferTest.java

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

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

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

```
    Container contentPane = getContentPane();

    textArea = new JTextArea();
     contentPane.add(new JScrollPane(textArea), "Center");
      JPanel panel = new JPanel();
    copyButton = new JButton("Copy");
      panel.add(copyButton);
      copyButton.addActionListener(this);
    pasteButton = new JButton("Paste");
      panel.add(pasteButton);
      pasteButton.addActionListener(this);
     contentPane.add(panel, "South");
      sysClipboard
          = Toolkit.getDefaultToolkit().getSystemClipboard();
 }

public void actionPerformed(ActionEvent event)
{   Object source = event.getSource();
    if (source == copyButton) copy();
     else if (source == pasteButton) paste();
 }

private void copy()
{   String text = textArea.getSelectedText();
    if (text.equals("")) text = textArea.getText();
    StringSelection selection = new StringSelection(text);
      sysClipboard.setContents(selection, null);
 }

private void paste()
{   String text;
    Transferable contents = sysClipboard.getContents(this);
    if (contents == null) return;
      try
    {   text = (String)(contents.getTransferData
              (DataFlavor.stringFlavor));
```

```
            textArea.replaceSelection(text);
        }
        catch(Exception e)
        {   textArea.append("Error: " + e);
        }
    }

    private JTextArea textArea;
    private JButton copyButton;
    private JButton pasteButton;
    private Clipboard sysClipboard;
}
```

java.awt.Toolkit

- Clipboard getSystemClipboard()
  시스템 클립보드를 얻는다.

java.awt.datatransfer.Clipboard

- Transferable getContents(Object requester)
  클립보드 내용들을 얻는다.
  전달인자: requester **클립보드 내용을 요청하는 객체**
- void setContents(Transferable contents, ClipboardOwner owner)
  클립보드상에 컨텐트를 넣는다.
  전달인자: contents **컨텐트를 캡슐화한** Transferable
  owner **새로운 정보가 클립보드상에 추가됐을** (lostOwnership **메소드를 통해**) **통지되는 객체**

java.awt.datatransfer.ClipboardOwner

- void lostOwnership(Clipboard clipboard, Transferable contents)
  더 이상 클립보드 내용들의 소유자가 아닐 때 해당 객체를 통지한다.
  전달인자: clipboard **컨텐트가 위치한 클립보드**
  contents **해당 소유자가 클립보드상에 위치시킨 아이템**

java.awt.datatransfer.Transferable

- boolean isDataFlavorSupported(DataFlavor flavor)
  명시된 플레버가 지원되는 데이터 플레버들 중의 하나이면 true**를**, **아니면** false**를 반환한다**.

- Object getTransferData(DataFlavor flavor)
  요청된 플레버에 포맷된 데이터를 반환한다. 만일 요청된 플레버가 지원되지 않으면 UnsupportedFlavorException을 발생한다.

전송가능하게 구성하기

클립보드를 경유하여 전송하길 원하는 객체들은 Transferable 인터페이스를 구현해야 한다. StringSelection 클래스는 현재 자바 표준 라이브러리에서 Transferable 인터페이스를 구현한 유일한 공용 클래스이다. 우리는 먼저 로컬 클립보드를 통해 이미지를 전송하는 데 사용될 수 있는 ImageSelection 클래스를 먼저 구성할 것이다. 마지막으로, 우리는 연속화된 자바 객체들을 위한 Transferable을 구성하는 방법을 구축할 것이다.

전송가능한 인터페이스와 데이터 가공
여러분은 전송가능한 객체가 지원하는 모든 데이터 플레버를 발견할 수 있다.

```
DataFlavor[] flavors = transferable.getTransferDataFlavors()
```

DataFlavor는 두개의 특징에 의해 정의된다.

- MIME 형식 이름(예, "image/gif")
- 데이터에 접근하기 위한 대표 클래스(예, InputStream)

게다가 , 모든 데이터 플레버는 사람이 읽을 수 있는 이름을 가지고 있다(예, "GIF Image").
대표 클래스는 MIME 타입에서 class 전달인자로 명시될 수 있다. 예를 들면,

```
image/gif;          class=java.awt.Image
```

노트 : 이것은 단지 문법을 보여주는 예제이다. 현재 GIF 이미지 데이터 전송을 지원하지 않는다.

만일 class 전달인자가 주어지지 않으면, 대표 클래스는 InputStream 이다.

선 마이크로시스템은 연속화된 자바 객체들을 위해 다음처럼 MIME 타입을 정의했다.

```
application/x-java-serialized-object
```

노트 : x 접두사는 이것이 표준 MIME 타입 이름을 할당해 주는 조직인 IANA에 의해서 인가되지 않는, 실험적인 이름이라는 것을 나타낸다.

예를 들면, 표준 stringFlavor 데이터 플레버는 다음의 MIME 타입으로 표현된다.

application/x-java-serialized-object; class=java.lang.String

java.awt.datatransfer.DataFlavor

- DataFlavor(String mimeType, String humanPresentableName)
  MIME 타입의 포맷으로 스트림 데이터를 설명하는 데이터 플레버를 생성한다.
  전달인자: mimeType MIME 타입 문자열
  humanPresentableName 읽을 수 있는 이름 버전
- DataFlavor(Class class, String humanPresentableName)
  자바 플랫폼 클래스를 설명하는 데이터 플레버를 생성한다. MIME 타입은 application/x-java-serialized-object; class= class name 과 같다.
  전달인자: class transferable로부터 연역된 클래스
  humanPresentableName 읽을 수 있는 이름 버전
- String getMimeType()
  해당 데이터 플레버를 위한 MIME 타입 문자열을 반환한다.
- boolean isMimeTypeEqual(String mimeType)
  해당 데이터 플레버가 주어진 MIME 타입인지를 검사한다.
- String getHumanPresentableName()
  해당 데이터 플레버의 데이터 포맷에 대한 사람이 읽을 수 있는 이름을 반환한다.
- Class getRepresentationClass()
  해당 데이터 플레버를 호출할 때 반황할 Transferable 객체의 클래스를 대표하는 Class 객체를 반환한다. MIME 타입의 class 전달인자 이거나 InputStream 이다.

java.awt.datatransfer.Transferable

- DataFlavor[] getTransferDataFlavors()
  지원되는 플레버들의 배열을 반환한다.

이미지 전송가능하게 구성하기

전송가능한 이미지 클래스를 설계하고 동작시키기 위해 앞선 절의 정보를 상기하라. 지금, 우리는 오직 단일 프로그램 내부에서 이미지들을 전송하길 원한다. 즉, 우리는 단지 로컬 클립보드를 사용할 것이다.

다음은 여러분이 해야 할 것 들이다.

1. Transferable 인터페이스를 구현한 ImageSelection 클래스를 정의한다.

```
Class ImageSelection implements Transferable{ …}
```

2. java.awt.image 클래스를 위한 클래스 플레버로서 단일 지원 플레버를 "AWT Image"라는 이름으로 정의한다. 해당 플레버를 대표하는 정적 imageFlavor 객체를 구성한다.

```
public static final DataFlavor imageFlavor = new DataFlavor( java.awt.Image.class, "AWT Image");
```

3. DataFlavor 타입의 imageFlavor 단일 항목을 갖는 배열을 반환하는 getTransferableDataFlavors 메소드를 만든다.

```
private static DataFlavor[] flavors = { imageFlavor };
public DataFlavor[] getTransferDataFlavors()
{           return flavors;
}
```

4. 요청받은 플레버가 imageFlavor 객체와 동일한지 검사하는 isDataFlavorSupported 를 갖는다.

```
public booleab isDataFlavorSupported(DataFlavor flavor)
{           return flavor.equals(imageFlavor);
}
```

5. image 타입의 객체를 받고 저장하는 ImageSelection 생성자를 갖는다.

```
public ImageSelectoin(Image image)
{           theImage = image;
}
private Image theImage;
```

6. getTransferData의 값으로서 저장된 Image 객체에 대한 참조를 반환한다.

```
public synchronized Object getTransferData
        (DataFlavor  flavor)
        throws  UnsupportedFlavorException
    {   if(flavor.equals(imageFlavor))
        {   return  theImage;
        }
        else
        {   throw  new  UnsupportedFlavorException(flavor);
        }
```

```
}
```

예제 7-14는 완전한 예제 코드이다.
물론, 이 모든 것이 다소 흥미롭지 못할 수 있다. ImageSelection 클래스는 단순히 Image 객체들을 포장한다.

보다 흥미롭게도, 이 클래스는 하나 이상의 플레버에 해당 내용을 전달할 필요가 있다. 가령, Image 객체와 GIF 파일로서. 그러면, 여러분은 getTransferData 메소드에서 몇가지 실제 작업을 보게 될 것이다. 이것은 다음처럼 보인다:

```
public synchronized Object getTransferData
        (DataFlavor flavor)
        throws UnsupportedFlavorException
    {   if(flavor.equals(imageFlavor))
        {   return theImage;
        }
         else if(flavor.equals(gifFlavor))
         {  byte[]      gifBytes = ..;
                        // translate image to GIF format
            return new ByteArrayInputStream(gifBytes);
         }
        else
        {   throw new UnsupportedFlavorException(flavor);
        }
    }
```

우리는 여기서 이 작업을 하지는 않을 것이다. 왜냐하면 이미지의 GIF 포맷을 계산하는 것은 다소 단조롭기 때문이다.

ImageSelection 클래스 사용

예제 7-14 프로그램은 두개의 윈도우를 생성한다. 여러분은 각 윈도우에 이미지 파일을 로드하거나 Edit|Copy와 Edit|Paste를 사용하여 윈도우간에 복사와 붙이기를 할 수 있다(그림7-43). ImageSelection 클래스를 사용하여 작성한 복사와 붙이기 기능은 StringSelection 클래스를 사용하는 것과 크게 다르지 않다.

```
private void copyIt()
{   ImageSelection selection = new ImageSelection(theImage);
    localClipboard.setContents(selection, null);
 }

private void pasteIt()
{   Transferable contents
        = localClipboard.getContents(this);
     try
    {   theImage = (Image)contents.getTransferData
            (ImageSelection.imageFlavor);
         repaint();
     }
    catch(Exception e) {}
}
```

이 경우, selection**은** ImageSelection **의 인스턴스이다. 우리는 분명** getTransferData**의 반환 값을 문자열이 아니라** Image**로 형변환할 필요가 있다. 그리고 결과를 텍스트 영역에 추가하는 것보다** repaint() **호출을 사용한다. 우리가 시스템 클립보드 보다 로컬 클립보드를 사용한다는 것에 주목하라. 이것은 다음처럼 구성된다.**

```
static Clipboard localClipboard = new Clipboard("local");
```

우리는 이 시점에서 시스템 클립보드를 사용할 수 없다. 왜냐하면 자바 기술은 오직 텍스트만을 시스템 클립보드로 전송할 수 있기 때문이다.

예제 7-14**는 예제의 완전한 코드를 보여준다.**

그림 7-43 : ImageTransferTest 프로그램

예제 7-14 : ImageTransferTest.java

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

```
import javax.swing.*;

public class ImageTransferTest
{    public static void main(String[] args)
   {  JFrame frame1 = new ImageTransferFrame();
      JFrame frame2 = new ImageTransferFrame();
       frame1.setTitle("Frame 1");
       frame2.setTitle("Frame 2");
       frame1.show();
       frame2.show();
    }
}

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

      Container contentPane = getContentPane();
      label = new JLabel();
       contentPane.add(label, "Center");

      JMenu fileMenu = new JMenu("File");
      openItem = new JMenuItem("Open");
        openItem.addActionListener(this );
        fileMenu.add(openItem);

      exitItem = new JMenuItem("Exit");
        exitItem.addActionListener(this);
        fileMenu.add(exitItem);
```

```
    JMenu editMenu = new JMenu("Edit");
    copyItem = new JMenuItem("Copy");
     copyItem.addActionListener(this);
     editMenu.add(copyItem);

    pasteItem = new JMenuItem("Paste");
     pasteItem.addActionListener(this);
     editMenu.add(pasteItem);

    JMenuBar menuBar = new JMenuBar();
     menuBar.add(fileMenu);
     menuBar.add(editMenu);
     setJMenuBar(menuBar);
 }

public void actionPerformed(ActionEvent evt)
{   Object source = evt.getSource();
    if (source == openItem)
    {   JFileChooser chooser = new JFileChooser();
         chooser.setCurrentDirectory(new File("."));

          chooser.setFileFilter(new
              javax.swing.filechooser.FileFilter()
            {   public boolean accept(File f)
                {   String name = f.getName().toLowerCase();
                     return name.endsWith(".gif")
                          || name.endsWith(".jpg")
                          || name.endsWith(".jpeg")
                          || f.isDirectory();
                 }
                public String getDescription()
                {   return "Image files";
                 }
             });

         int r = chooser.showOpenDialog(this);
```

```
        if(r == JFileChooser.APPROVE_OPTION)
        {   String name
                = chooser.getSelectedFile().getAbsolutePath();
            setImage(Toolkit.getDefaultToolkit().getImage(name));
        }
    }
  else if (source == exitItem) System.exit(0);
  else if (source == copyItem) copy();
  else if (source == pasteItem) paste();
}

private void copy()
{  ImageSelection selection = new ImageSelection(theImage);
    localClipboard.setContents(selection, null);
}

private void paste()
{   Transferable contents
        = localClipboard.getContents(this);
  if (contents == null) return;
    try
   {   Image image = (Image)contents.getTransferData
            (ImageSelection.imageFlavor);
        setImage(image);
    }
   catch(Exception e) {}
}

public void setImage(Image image)
{   theImage = image;
    label.setIcon(new ImageIcon(image));
}

private static Clipboard localClipboard
   = new Clipboard("local");
private Image theImage;
```

```
    private JLabel label;
    private JMenuItem openItem;
    private JMenuItem exitItem;
    private JMenuItem copyItem;
    private JMenuItem pasteItem;
}

class ImageSelection implements Transferable
{   public ImageSelection(Image image)
    {   theImage = image;
    }

    public DataFlavor[] getTransferDataFlavors()
    {   return flavors;
    }

    public boolean isDataFlavorSupported(DataFlavor flavor)
    {   return flavor.equals(imageFlavor);
    }

    public synchronized Object getTransferData
        (DataFlavor flavor)
        throws UnsupportedFlavorException
    {   if(flavor.equals(imageFlavor))
        {   return theImage;
        }
        else
        {   throw new UnsupportedFlavorException(flavor);
        }
    }

    public static final DataFlavor imageFlavor
       = new DataFlavor(java.awt.Image.class, "AWT Image");

   private static DataFlavor[] flavors = { imageFlavor };
    private Image theImage;
```

}

시스템 클립보드를 통해 자바 객체들 전송하기

이제, 여러분들이 개인적인 클립보드가 아니라 시스템 클립보드를 사용하여 이미지를 붙이길 원한다고 가정해보자. 이러한 기능은 비록 본래의 프로그램이 종료됐을 지라도 자바 프로그래밍 언어로 개발된 다른 프로그램들간에 이미지를 붙일 수 있고 이미지를 지속화 시킬 수 있다.

불운하게도, 시스템 클립보드는 현재 오직 하나의 프로그램에서 다른 프로그램으로 StringSelection 데이터를 전송하는 데 사용된다.

이 절에서, 우리는 여러분에게 객체들을 텍스트 문자열로 인코딩하고 이들 텍스트 문자열을 시스템 클립보드상에 위치시키는 것에 의해 이러한 제약을 극복하는 방법을 보여준다. 이들 메소드들은 임의의 연속화된 자바 객체에 대해 작동한다. 우리는 다음의 단순한 텍스트 인코딩을 선택한다.

```
Content-type: application/x-java-serialized-object; class=java.util.Serializable
Content-length: length

Serialized object data in BASE64 encoding
```

예를 들면,

```
Content-type: application/x-java-serialized-object; class=java.util.Serializable
Content-length: 80311

ro0ABXNyAAZCaXRtYXA8A5/mgeUpsAIAA0kAMmhlaWdodEkABXdpZHRoWwAGcG
14ZWxzdAACW014cAAAAIwAAABqdXIAAltJTbpgJnbqsUCAAB4cAAAOfj ////////////////
```

BASE64 인코딩은 일반적으로 바이너리 데이터를 출력가능한 문자들로 인코딩하는 메소드로서 사용된다. 정확안 인코딩 스킴은 중요하지 않다. 여러분은 MIME RFC에서 설명을 찾을 수 있다. 그리고 예제 7-15의 코드는 인코딩과 디코딩을 수행하는 클래스를 포함하고 있다.

시스템 클립보드 클래스는 오직 클립보드로부터 텍스트를 얻어 올 수 있기 때문에, 우리는 특별한 MimeClipboard 클래스를 디자인 하였다. 이 클래스는 다른 클립보드, 기본적으로는 시스템 클립보드에 저정과 추출 요청을 대표한다. 그리고 데이터 인코딩과 디코딩을 처리한

다.

```
class MimeClipboard extends Clipboard
{       public MimeClipboard(Clipboard cb)
        {…
         clip = cp;
        }
        public synchronized void setContents(Trasferable contents, ClipboardOwner owner)
        {       encode data and put on clip
        }
        public synchronized Transferable getContents(Object requestor)
        {       get data from clip and decode
        }
        private Clipboard clip;
}
```

또한 우리는 연속화된 객체들에 대한 SerializableSelection 포장 객체를 전송하는 사용한다 – 이것은 정확히 앞선 절의 ImageSelection 클래스의 연속이다 Transferable 객체가 MimeClipboard에 놓을 때, 다음 일들이 발생한다.

1. 만일 Transferable 객체가 StringSelection이면, 쉽게 클립보드에 위키시긴다.
2. 그렇지 않고, 만일 SerializableSelection 타입의 Transferable 이면 객체는 일련의 바이트들로 연속화된다. 연속화된 바이트들은 BASE64로 인코딩되고 , MIME 헤더가 추가되고 결과 문자열이 클립보드에 놓인다.

```
public synchronized void setContents(Transferable contents, ClipboardOwner owner)
  {  if (contents instanceof SerializableSelection)
     {  try
        {  DataFlavor flavor
              = SerializableSelection.serializableFlavor;
           Serializable obj = (Serializable)
                contents.getTransferData(flavor);
           String enc = encode(obj);
           String header = "Content-type: "
              + flavor.getMimeType()
              + "\nContent-length: "
```

```
            + enc.length() + "\n\n";
          StringSelection selection
            = new StringSelection(header + enc);
          clip.setContents(selection, owner);
        }
        catch(UnsupportedFlavorException e){}
        catch(IOException e){}
      }
    else clip.setContents(contents, owner);
  }
```

클립보드로부터 Transferable **객체를 읽을 때, 이들 단계들이 거꾸로 된다**.

1. **시스템 클립보드로부터** StringSelection**을 얻는다**.
2. **만일 문자열이** Content-type**으로 시작하지 않으면, 문자열 선택 객체가 반환된다**.
3. **그렇지 않으면**, BASE64 **데이터는 바이너리로 변환되고 객체 스트림 데이터로서 읽는다**.

결과 객체는 SerializableSelection **전송 객체로 포자되어 반환된다**.

```
public synchronized Transferable getContents(Object requestor)
  {  Transferable contents = clip.getContents(requestor);

    if (contents instanceof StringSelection)
     {   String data = (String)contents.getTransferData
              (DataFlavor.stringFlavor);
        }

        if (!data.startsWith("Content-type: "))
          return contents;
        int start = .;;.      // skip three newlines

        Serializable obj = decode(data.substring(start));
        SerializableSelection selection
          = new SerializableSelection(obj);
        return selection;
     }
    else return contents;
  }
```

이 메커니즘은 Content-type으로 시작하는 본래의 문자열을 전송할 수 없다. 이 문제를 극복하기 위해 메커니즘을 강화하는 것은 어렵지 않다. 그러나 우리는 이 강화를 독자의 숙제로 남겨 놓는다.

여러분은 해당 클립보드를 다른 클립보드처럼 사용한다.

```
Clipboard mimeClipboard = new MimeClipboard(Toolkit.getDefaultToolkit().getSystemClipboard());
…
private void copyIt()
{       SerializableSelection selection = new SerializableSelection(theBitmap);
        mimeClipboard.setContents(selection, null);
}
```

객체를 인코딩하기 위해, 우리는 해당 객체를 스트림으로 연속화한다. Base64OutputStream 클래스는 모든 바이트를 BASE64로 인코딩한다. 우리는 이 클래스 상위에 ObjectOutputStream을 올렸다. 다음은 인코딩 메소드의 코드이다.

```
ByteArrayOutputStream bOut = new ByteArrayOutputStream();
Base64OutputStream b64Out
    = new Base64OutputStream(bOut);
ObjectOutputStream out
    = new ObjectOutputStream(b64Out);
out.writeObject(obj);
out.close();
return bOut.toString("8859_1");
```

디코드를 위해, 우리는 같은 접근 방법을 따른다. Base64InputStream은 BASE64문자들을 바이트로 변환하는 읽기 메소드를 가지고 있다. 그리고 ObjectInputStream은 해당 스트림으로부터 연속화된 객체 데이터를 읽는다. 다음은 decode 메소드의 코드이다.

```
byte[] bytes = s.getBytes("8859_1");
ByteArrayInputStream bIn
    = new ByteArrayInputStream(bytes);
Base64InputStream b64In
    = new Base64InputStream(bIn);
```

```
ObjectInputStream in
   = new ObjectInputStream(b64In);
Object obj = in.readObject();
```

이 기술을 이미지들을 전송하는 작업에 사용할 때, 사소한 기술적인 후퇴가 있다 : Image 클래스는 Serializable을 구현하지 못한다. 이 문제를 극복하기 위해, 우리는 이미지의 칙셀들을 유지하는 연속화된 Bitmap 클래스를 디자인하였다.

```
class Bitmap implements Serializable
{  public Bitmap(BufferedImage image)
   {  type = image.getType();
      width = image.getWidth();
      height = image.getHeight();
      WritableRaster raster = image.getRaster();
      data = raster.getDataElements(0, 0, width, height, null);
   }

   public BufferedImage getImage()
   {  BufferedImage image
         = new BufferedImage(width, height, type);
      WritableRaster raster = image.getRaster();
      raster.setDataElements(0, 0, width, height, data);
      return image;
   }

   private int type;
   private int width;
   private int height;
   private Object data;
}
```

제1권의 12장에서 우리는 실제로 연속화와 비연속화를 위한 메소드를 작성할 필요가 없었다. 연속화 메커니즘은 자동적으로 픽셀들의 배열과 width, heght 필드들을 연속화할 것이다.

이제 우리는 이미지를 시스템 클립보드로 복사할 수 있는 프로그램을 작성하기 위한 모든 조각들을 가지고 있다. 예제 7-15내의 프로그램은 단지 이 작업을 수행한다. 프로그램을 수

행하면 이미지를 로드하고 이를 클립보드에 복사한다. 그런 다음, 프로그램을 종료하고 다시 구동시킨다. Edit|Paste를 선택하고 어떻게 이미지가 새로운 프로그램 인스턴스에 전송하는 지 지켜보라. 그렇지 않으면 그림 7-44처럼 여러 프로그램 복사본을 수행하고 이들 간의 복사와 붙이기를 수행해 보라.

만일 여러분이 클립보드뷰어를 사용하거나 여러분의 워드 프로세스내에 Paste를 선택한다면 클립보드 데이터의 MIME 인코딩을 볼 수 있다(그림7-45참조). 그러나 여러분이 할 수 없는 일은 클립보드의 비트맵을 이미지로서 여러분의 워드프로세스에 붙일 수 없다는 것이다. 시스템 클립보드는 데이터가 실제 이미지 인지 알지 못한다. 그리고 인코딩을 이해하지 못한다. 같은 이유에서, 여러분은 여러분의 웹 부라우져내에 있는 이미지를 선택하고 이를 클립보드에 복사할 수 없다. 그리고 이를 우리의 에제 프로그램내에 붙일 수도 없다. 이전에 언급했던 것처럼, 현재 자바 프로그램언어로 작성된 프로그램과 고유한 응용 프로그램간에 텍스트 이상의 다른 것을 전송할 수 없다.

그림 7-44: 두 개의 MimeClipboardTest 프로그램 인스턴스간의 데이터 복사

그림 7-45 : 이미지를 복사한 후의 클립보드 컨텐트

예제 7-15 : MimeClipboardTest.java

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

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

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

        Container contentPane = getContentPane();
        label = new JLabel();
        contentPane.add(label, "Center");

        JMenu fileMenu = new JMenu("File");
        openItem = new JMenuItem("Open");
        openItem.addActionListener(this);
        fileMenu.add(openItem);

        exitItem = new JMenuItem("Exit");
        exitItem.addActionListener(this);
        fileMenu.add(exitItem);

        JMenu editMenu = new JMenu("Edit");
        copyItem = new JMenuItem("Copy");
        copyItem.addActionListener(this);
        editMenu.add(copyItem);

        pasteItem = new JMenuItem("Paste");
        pasteItem.addActionListener(this);
        editMenu.add(pasteItem);

        JMenuBar menuBar = new JMenuBar();
        menuBar.add(fileMenu);
```

```
    menuBar.add(editMenu);
    setJMenuBar(menuBar);
}

public void actionPerformed(ActionEvent evt)
{   Object source = evt.getSource();
   if (source == openItem)
   {   JFileChooser chooser = new JFileChooser();
       chooser.setCurrentDirectory(new File("."));

       chooser.setFileFilter(new
           javax.swing.filechooser.FileFilter()
       {   public boolean accept(File f)
           {   String name = f.getName().toLowerCase();
               return name.endsWith(".gif")
                   || name.endsWith(".jpg")
                   || name.endsWith(".jpeg")
                   || f.isDirectory();
           }
           public String getDescription()
           {   return "Image files";
           }
       });

       int r = chooser.showOpenDialog(this);
       if(r == JFileChooser.APPROVE_OPTION)
       {   String name
               = chooser.getSelectedFile().getAbsolutePath();
           setImage(Toolkit.getDefaultToolkit().getImage(name));
       }
   }
  else if (source == exitItem) System.exit(0);
   else if (source == copyItem) copy();
  else if (source == pasteItem) paste();
}
```

```
private void copy()
{   MediaTracker tracker = new MediaTracker(this);
    tracker.addImage(theImage, 0);
    try { tracker.waitForID(0); }
    catch (InterruptedException e) {}
    BufferedImage image
       = new BufferedImage(theImage.getWidth(null),
            theImage.getHeight(null),
            BufferedImage.TYPE_INT_RGB);
    Graphics2D g2 = image.createGraphics();
    g2.drawImage(theImage, 0, 0, null);

    Bitmap bitmap = new Bitmap(image);
    SerializableSelection selection
       = new SerializableSelection(bitmap);
    mimeClipboard.setContents(selection, null);
 }

private void paste()
{   Transferable selection
       = mimeClipboard.getContents(this);
    try
    {   Bitmap bitmap = (Bitmap)selection.getTransferData
            (SerializableSelection.serializableFlavor);
        setImage(bitmap.getImage());
    }
    catch(Exception e) {}
 }

public void setImage(Image image)
{   theImage = image;
    label.setIcon(new ImageIcon(image));
 }

private static Clipboard mimeClipboard
    = new MimeClipboard
```

```
            (Toolkit.getDefaultToolkit().getSystemClipboard());
    private Image theImage;
    private JLabel label;
    private JMenuItem openItem;
    private JMenuItem exitItem;
    private JMenuItem copyItem;
    private JMenuItem pasteItem;
}

class Bitmap implements Serializable
{   public Bitmap(BufferedImage image)
    {   type = image.getType();
        width = image.getWidth();
        height = image.getHeight();
        WritableRaster raster = image.getRaster();
        data = raster.getDataElements(0, 0, width, height, null);
    }

    public BufferedImage getImage()
    {   BufferedImage image
            = new BufferedImage(width, height, type);
        WritableRaster raster = image.getRaster();
        raster.setDataElements(0, 0, width, height, data);
        return image;
    }

    private int type;
    private int width;
    private int height;
    private Object data;
}

class SerializableSelection implements Transferable
{   public SerializableSelection(Serializable object)
    {   theObject = object;
    }
```

```
    public boolean isDataFlavorSupported(DataFlavor flavor)
    {   return flavor.equals(serializableFlavor);
    }

    public synchronized Object getTransferData
        (DataFlavor flavor)
        throws UnsupportedFlavorException
    {   if(flavor.equals(serializableFlavor))
        {   return theObject;
        }
        else
        {   throw new UnsupportedFlavorException(flavor);
        }
    }

    public DataFlavor[] getTransferDataFlavors()
    {   return flavors;
    }

    public static final DataFlavor serializableFlavor
        = new DataFlavor(java.io.Serializable.class,
        "Serializable Object");

    private static DataFlavor[] flavors
        = { serializableFlavor };

    private Serializable theObject;
}

class MimeClipboard extends Clipboard
{   public MimeClipboard(Clipboard cb)
    {   super("MIME/" + cb.getName());
        clip = cb;
    }
```

```
public synchronized void setContents(Transferable contents,
    ClipboardOwner owner)
{   if (contents instanceof SerializableSelection)
    {   try
        {   DataFlavor flavor
                = SerializableSelection.serializableFlavor;
            Serializable obj = (Serializable)
                contents.getTransferData(flavor);
            String enc = encode(obj);
            String header = "Content-type: "
                + flavor.getMimeType()
                + "\nContent-length: "
                + enc.length() + "\n\n";
             StringSelection selection
                = new StringSelection(header + enc);
            clip.setContents(selection, owner);
        }
        catch(UnsupportedFlavorException e)
        {}
        catch(IOException e)
        {}
    }
    else clip.setContents(contents, owner);
}

public synchronized Transferable getContents
    (Object requestor)
{   Transferable contents = clip.getContents(requestor);

    if (contents instanceof StringSelection)
    {   String data = null;
        try
        {   data = (String)contents.getTransferData
                (DataFlavor.stringFlavor);
        }
        catch(UnsupportedFlavorException e)
```

```
        { return contents; }
         catch(IOException e)
        { return contents; }

         if (!data.startsWith("Content-type: "))
             return contents;
        int start = -1;
        // skip three newlines
       for (int i = 0; i < 3; i++)
         {   start = data.indexOf('\n', start + 1);
           if (start < 0) return contents;
           }
        Serializable obj = decode(data.substring(start));
         SerializableSelection selection
             = new SerializableSelection(obj);
        return selection;
     }
    else return contents;
 }

public static String encode(Serializable obj)
 {   ByteArrayOutputStream bOut
        = new ByteArrayOutputStream();
     try
     {   Base64OutputStream b64Out
             = new Base64OutputStream(bOut);
         ObjectOutputStream out
             = new ObjectOutputStream(b64Out);
          out.writeObject(obj);
          out.close();
         return bOut.toString("8859_1");
     }
    catch (IOException exception)
     {   return null;
     }
 }
```

```
    public static Serializable decode(String s)
     {   try
        {   byte[] bytes = s.getBytes("8859_1");
            ByteArrayInputStream bIn
               = new ByteArrayInputStream(bytes);
            Base64InputStream b64In
               = new Base64InputStream(bIn);
            ObjectInputStream in
               = new ObjectInputStream(b64In);
             Object obj = in.readObject();
             in.close();
            return (Serializable)obj;
         }
        catch(Exception e)
        {   return null;
         }
     }

    private Clipboard clip;
}

/* BASE64 encoding encodes 3 bytes into 4 characters.
     |11111122|22223333|33444444|
   Each set of 6 bits is encoded according to the
   toBase64 map. If the number of input bytes is not
   a multiple of 3, then the last group of 4 characters
   is padded with one or two = signs. Each output line
    is at most 76 characters.
*/

class Base64OutputStream extends FilterOutputStream
{   public Base64OutputStream(OutputStream out)
     {   super(out);
     }
```

```
public void write(int c) throws IOException
{   inbuf[i] = c;
     i++;
   if (i == 3)
    {   super.write(toBase64[(inbuf[0] & 0xFC) >> 2]);
       super.write(toBase64[((inbuf[0] & 0x03) << 4) |
           ((inbuf[1] & 0xF0) >> 4)]);
       super.write(toBase64[((inbuf[1] & 0x0F) << 2) |
           ((inbuf[2] & 0xC0) >> 6)]);
        super.write(toBase64[inbuf[2] & 0x3F]);
        col += 4;
        i = 0;
        if (col >= 76)
         {   super.write('\n');
            col = 0;
         }
     }
 }

public void flush() throws IOException
{  if (i == 1)
   {  super.write(toBase64[(inbuf[0] & 0xFC) >> 2]);
      super.write(toBase64[(inbuf[0] & 0x03) << 4]);
        super.write('=');
        super.write('=');
     }
   else if (i == 2)
   {  super.write(toBase64[(inbuf[0] & 0xFC) >> 2]);
      super.write(toBase64[((inbuf[0] & 0x03) << 4) |
           ((inbuf[1] & 0xF0) >> 4)]);
      super.write(toBase64[(inbuf[1] & 0x0F) << 2]);
        super.write('=');
     }
   i = 0;
 }
```

```
    private static char[] toBase64 =
    {  'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
       'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
      'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
      'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f',
      'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
      'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
      'w', 'x', 'y', 'z', '0', '1', '2', '3',
       '4', '5', '6', '7', '8', '9', '+', '/'
     };

    private int col = 0;
    private int i = 0;
    private int[] inbuf = new int[3];
}


class Base64InputStream extends FilterInputStream
{   public Base64InputStream(InputStream in)
     {    super(in);
     }

     public int read(byte[] b, int off, int len) throws IOException
    {   if (len > b.length - off) len = b.length - off;
       for (int i = 0; i < len; i++)
        {   int ch = read();
            if (ch == -1) return i;
           b[i + off] = (byte)ch;
         }
        return len;

     }

    public int read(byte[] b) throws IOException
     {   return read(b, 0, b.length);
     }
```

```
public int read() throws IOException
 {   int r;
    if (i == 0)
     {   // skip whitespace
          do
         {   ch[0] = super.read();
             if (ch[0] == -1) return -1;
          }
          while (Character.isWhitespace((char)ch[0]));
         ch[1] = super.read();
         if (ch[1] == -1) return -1;
          i++;
        r = (fromBase64[ch[0]] << 2)
              | (fromBase64[ch[1]] >> 4);
     }
    else if (i == 1)
     {   ch[2] = super.read();
       if (ch[2] == '=' || ch[2] == -1) return -1;
          i++;
       r = ((fromBase64[ch[1]] & 0x0F) << 4)
             | (fromBase64[ch[2]] >> 2);
     }
     else
     {   ch[3] = super.read();
       if (ch[3] == '=' || ch[3] == -1) return -1;
         i = 0;
       r = ((fromBase64[ch[2]] & 0x03) << 6)
              | fromBase64[ch[3]];
     }
    return r;
 }

private static int[] fromBase64 =
 {   -1, -1, -1, -1, -1, -1, -1, -1,
     -1, -1, -1, -1, -1, -1, -1, -1,
```

```
        -1, -1, -1, -1, -1, -1, -1, -1,
        -1, -1, -1, -1, -1, -1, -1, -1,
        -1, -1, -1, -1, -1, -1, -1, -1,
        -1, -1, -1, 62, -1, -1, -1, 63,
       52, 53, 54, 55, 56, 57, 58, 59,
        60, 61, -1, -1, -1, -1, -1, -1,
        -1,  0,  1,  2,  3,  4,  5,  6,
         7,  8,  9, 10, 11, 12, 13, 14,
       15, 16, 17, 18, 19, 20, 21, 22,
        23, 24, 25, -1, -1, -1, -1, -1,
        -1, 26, 27, 28, 29, 30, 31, 32,
       33, 34, 35, 36, 37, 38, 39, 40,
        41, 42, 43, 44, 45, 46, 47, 48,
        49, 50, 51, -1, -1, -1, -1, -1
    };

  int i = 0;
  int[] ch = new int[4];
}
```

드레그-앤-드롭

여러분이 두개의 프로그램사이의 정보를 잘라 붙이기를 사용하여 전송할 때, 클립보드는 중간자로서 작동한다. 드레그-앤-드롭(drag and Drop)는 중간자를 잘라 두개 프로그램간에 직접 통신을 하게 한다. 자바 2 플랫폼은 드레그-앤-드롭을 기본 기능을 제공한다. 여러분은 자바 응용 프로그램과 고유 응용 프로그램간의 드레그-앤-드롭을 수행할 수 있다. 이 절은 여러분에게 목표를 놓는 자바 응용 프로그램과 소스를 끌어내는 응용 프로그램을 작성하는 방법을 보여준다.

드레그-앤-드롭을 지원하기 위한 자바 플랫폼에 대해 보다 자세히 들어가기 전에, 잠시 드레그-앤-드롭 인터페이스에 대해 살펴보자. 우리는 윈도우 익스플로러와 워드패드 프로그램을 예제들로서 사용할 것이다 – 다른 플랫폼상에서 여러분은 드레그-앤-드롭 기능을 제공하는 로컬 컴퓨터상에서 이용가능한 프로그램으로 실험을 할 수 있다.

여러분은 드레그 소스내에 드레그 연산을 초기화 한다-일반적으로, 먼저 하나 또는 그 이상의 항목들을 선택하고 초기 위치에서 다른 곳으로 끌어다 놓는다(그림 7-46).

그림 7-46 : 드래그 연산의 초기화

놓기 연산을 수용하는 놓기 목표상에 눌렀던 마우스 버튼을 놓는다. 그러면 놓기 목표는 놓기 소스에 떨어뜰릴 항목들에 대한 정보를 묻는다. 그리고 몇몇 연산을 초기화한다. 가령, 만일 여러분이 윈도우 익스플로러에서 워드 패드로 파일 아이콘을 놓는다면 워드 패드는 파일을 개방한다. 그러나 만일 여러분이 파일 아이콘을 윈도우 익스플로러의 디렉토리 아이콘의 상단에 놓는다면 파일을 디렉토리로 이동시킨다.

만일 여러분이 드레그하는 동안 시프트나 컨트롤 키를 누르고 있다면 드레그 행위의 타입은 이동 행위에서 복사 행위로 변한다. 그리로 파일의 복사본이 디렉토리에 놓인다. 만일 여러분이 시스트와 컨트롤 키를 동시에 누른다면 파일에 대한 링크가 디렉토리에 위치한다.(다른 플랫폼들에서는 이러한 연산들에 대해 다른 키보드 조합을 사용할 수 있다.)

다른 제스츄어를 갖는 세가지 놓기 행위가 존재한다.

- 이동
- 복사
- 링크

링크 행위의 목적은 놓기 항목에 대한 참조를 만드는 것이다. 이와 같은 링크는 기본적으로 호스트 운영체제에서 지원해야 한다(파일에 대한 심볼릭 링크와 문서 컴포넌트들을 위한 객체 링크 같은). 그리고 일반적으로 크로스 플랫폼 프로그램에서는 통용되지 않는다. 이 절에서 우리는 복사와 이동을 위한 드레그-앤-드롭 기능에 주안점을 둔다.

모든 프로그램이 놓기 행위 타입들에 반응하는 것은 아니다. 가령, 만일 여러분이 파일을 워드패드에 놓는다면, 행위 타입을 무시하고 항상 파일을 개방한다.

일반적으로 드레그 기능을 위한 몇몇 시각적 피드백이 존대한다. 적게는, 커져 모양이 변경될 것이다. 가능한 놓기 목표상으로 커져를 이동할 때, 커져 모양은 놓기가 가능한지 아닌지를 알려준다. 만일 놓기가 가능하다면, 커져 모양 또한 놓기 행위의 타입을 표현할 것이다. 그림7-47은 놓기 목표들에 대한 몇몇 커져 모양을 보여준다.

그림 7-47 : 그롭 목표상에서의 커서 모양

또한 여러분은 파일 아이콘들외에 다른 항목들에 대해 드레그를 수행할 수 있다. 가령, 여러분은 워드 패드내에 텍스트를 선택하고 이를 드레그 할 수 있다. 텍스트 조각을 원하는 놓기 목표에 떨어 뜨리고 어떻게 반응하는 지 살펴보라.

노트 : 이 실험은 사용자 인터페이스 매카니즘으로서 드레그-앤-드롭 기능의 단점을 보여준다. 사용자들은 드레그 가능한 것이 무엇이고, 어디에 이를 떨어뜨리고 그렇게 했을 때 무슨 일이 발생하는 가에 대해 예측하기 힘들다. 기본적인 "이동" 행위는 본래의 것을 삭제할 수 있기 때문에 많은 사용자들은 드레그-앤-드롭 기능에 대한 실험을 주의해서 이해해야 한다.

드롭 목표

이 절에서, 우리는 목표를 떨어 뜨리는 아주 단순한 자바 응용 프로그램을 구성할 것이다. 예제 프로그램은 유용하지 않다. 예제 프로그램은 단지 사용자가 놓기 초기화를 하고 싶고, 드롭을 받아 들이는 방법과 놓여진 데이터를 분석하는 방법을 감지하는 것을 알려주는 예제이다.

여러분은 모든 AWT 컴포넌트를 놓기 목표가 되도록 설계할 수 있다. 이를 위해, 여러분은 DropTarget 객체를 구성하고 컴포넌트와 놓기 목표 리스너를 생성자에게 전달할 필요가 있다. 생성자는 드레그-앤-드롭 시스템으로 객체를 등록한다.

```
DropTarget target = new DropTarget(component, listener);
```

여러분은 setActive 메소드로 목표를 활성화 또는 비활성화 시킬 수 있다. 기본적으로 목표는 활성화되어 있다.

```
target.setActive(b);
```

여러분은 기본적으로 받아들이길 원하는 놓기 연산을 설정하기 위해 setDefaultActions 메소드를 호출할 수 있다. 연산은 다음과 같다.

DndConstants.ACTION_COPY

DndConstants.ACTION_MOVE

DndConstants.ACTION_COPY_OR_MOVE

DndConstants.ACTION_LINK

기본적으로, 모든 연산들이 가능하다.

이제 여러분은 놓기 목표 리스너를 구현할 필요가 있다. DropTargetListener 인터페이스는 5개의 메소드들을 가지고 있다:

```
void dragEnter(DropTargetDragEvent event)
void dragExit(DropTargetEvent event)
void dragOver(DropTargetDragEvent event)
void dropActionChanged(DropTargetDragEvent event)
void drop(DropTargetDropEvent event)
```

dragEnter와 drapExit 메소드들은 커져가 놓기 목표 컴포넌트에 진입하거나 벗어날 때 호출된다. DragEnter 메소드에서, 여러분은 커져 모양을 놓기 목표가 드롭을 받아들일 수 있는지를 알려주도록 설정할 기회를 갖는다.

여러분을 일반적으로 dragExit 메소드에 대해 걱정할 필요가 없다. 그러나 만일 여러분이 비쥬얼 피드백을 위한 몇가지 고급 메커니즘을 구현한다면 이 메소드내에 배치할 수 있다.

dragOver 메소드는 사용자가 마우스를 놓기 목표 컴포넌트상에서 이동할 때 계속해서 호출된다. 여러분은 놓기의 놓기의 효과에 대한 보다 자세한 비쥬얼 피드백을 제공하기 위해 이를 사용할 수 있다. 가령 만일 놓기 목표가 트리 컴포넌트이면, 여러분은 커져하에 있는 노드를 밝게 하거나 심지어 커져가 노드를 배회할 때 하위 트리를 개방할 수 있다.

DropActionChanged 메소드는 사용자가 액션 제스쳐를 변경했을 때 호출된다. 가령, 만일 사용자가 마우스를 놓기 목표 컴포넌트상에 이동하는 동안 시프트나 컨트롤 키를 눌렀다면 DropActionChanged 메소드가 호출된다. 가장 중요한 메소드는 drop 메소드이다. 사용자가 놓기 제스츄어를 마침으로서, 일반적으로 마우스 버튼을 릴리이즈하는 것에 의해 드롭을 완료했을 때 호출된다. 이 메소드는 이전에 여러분이 드롭을 받아들일 것인지 표시하건 아니건 상관없이 호출된다는 것을 기억하라.

DropTargetListener 메소드의 전달인자들을 주의깊게 살펴보라. 이들중 세 개는 DropTarget**Drag**Event 이다. 그러나 drop 메소드는 DropTarget**Drop**Event를 받는다. 그리고

dragExit 메소드는 DropTargetEvent를 받는다. DropTargetEvent클래스는 다른 두개의 이벤트 클래스들의 상위 클래스이다. 이 클래스는 오직 하나의 메소드인 getDropTargetContext를 갖는다. dragExit 메소드의 목적이 깨끗하게 정리하는 것이기 때문에, 여러분은 아마도 전달인자를 살펴보지 않을 것이다.

DropTargetDragEvent와 DropTargetDropEvent 클래스들은 각기 다음의 메소드들을 가지고 있다.

int getDropAction()
Point getLocation()
DataFlavor[] getCurrentDataFlavors()
Boolean isDataFlavorSupported(DataFlavor flavor)

여러분은 드레그가 가능한지, 드롭을 허락하는 지, 그리고 비쥬얼 피드백을 주는 방법이 무엇인지에 대한 여부를 검사하는 데 이들 메소드들을 필요로 한다.

불운하게도, 이들 메소드들을 공통의 상위 클래스에서 이용할 수 없기 때문에 여러분은 검사 로직을 두번 구현해야 할 것이다. 한번은 드레그를 위해 그리고 드롭을 위해 다시 한번.

만일 여러분이 드레그를 제공하길 원치 않는다면 dragEnter나 dragActionChanged 메소드내에서 DropTargetDragEvent 클래스의 rejectDrag 메소드를 호출해야 한다. 그 결과로서, 커서는 경고 아이콘으로 변한다. 만일 사용자가 경고에도 불구하고 항목들을 놓는다면 DropTargetDropEvent 클래스의 rejectDrop 메소드를 호출해야 한다.

getDropAction 메소드는 사용자의 의도적인 놓기 행위를 반환한다. 많은 드레그-앤-드롭 연산에서, 여러분은 한번에 행동하길 원치않는다. 예를 들면, 여러분이 파일 아이콘을 워드 패드로 옮긴다면 파일을 삭제하는 드레그 소스를 원하지 않을 것이다. 또한 여러분은 드레그를 할 때 사용자가 키를 누르고 목표를 드롭하 길 원치않는다. 이 상황에서, 실제 행동을 갖는 DropTargetDropEvent의 acceptDrop 메소드를 호출한다. 만일 여러분이 다음을 호출하면 드롭 소스는 드레그 아이템들을 삭제할 것이다.

```
event.acceptDrop(DnDConstants.ACTION_MOVE);
// drag source deletes dragged items!
```

팁 : 여러분의 사용자가 이동과 복사 연산간의 차이를 이해하고 시프트와 컨트롤 키의 역할을 이해하길 원하는가? 사용자들이 기본 드레그 제스츄어가 소스로부터 드레그 아이템들을

삭제한다는 사실을 알고 있는가? 만일 모른다면 여러분은 드롭을 다음처럼 받아들여야 한다.

```
event.acceptDrop(DnDConstants.ACTION_COPY);
```

다른 한편으로, 만일 여러분이 실제 이동과 복사간의 구별을 원한다면, 다음을 호출하라.

```
event.acceptDrop(event.getDropEvent());
```

다음은 기본적인 놓기 목표 리스너의 윤곽이다:

```
class ADropTargetListener implements DropTargetListener
{   public boolean isDragAcceptble(DropTargetDragEvent event)
   {  look at drop action and available data flavors
    }
   public boolean isDropAcceptble(DropTargetDropEvent event)
   {  same out the same test as in isDragAcceptable
    }

   // listener methods
    public void dragEnter(DropTargetDragEvent event)
    {   if (!isDragAcceptable(event))
        {   event.rejectDrag();
            return;
        }
    }

   public void dragExit(DropTargetEvent event)
    {
    }

   public void dragOver(DropTargetDragEvent event)
    {  // you can provide visual feedback here
    }

   public void dropActionChanged(DropTargetDragEvent event)
```

```
    {   if (!isDragAcceptable(event))
        {   event.rejectDrag();
            return;
        }
    }

   public void drop(DropTargetDropEvent event)
    {   if (!isDropAcceptable(event))
        {   event.rejectDrop();
            return;
        }

        event.acceptDrop(actual action);
        process data from drag source
        event.dropComplete(true);
    }
    ….
}
```

일단 여러분이 드롭을 받아들이면, 드레그 소스로부터 데이터를 분석할 필요가 있다. DropTargetDropEvent**클래스의** getTransferable **메소드는** Transferable**객체에 대한 참조를 반환한다. 이것은 잘라 붙이기를 위해 사용하는 것과 같은 인터페이스이다.**

여러분은 이제 클립보드로부터 데이터를 붙일 때 했던 것과 같은 과정을 수행한다. 그러나 자바 플랫폼은 시스템 클립보드에서 제공했던 것보다 드레그-앤-드롭 메키니즘을 위해 보다 많은 데이터 플레버를 지원한다.

만일 DtaFlavor.javaFileListFlavor **데이터 플레버가 이용가능하다면** Transferable**객체는 파일 목록을 생성해 낸다. 이 목록은 아이템들을 갖는** java.util.List **타입의 객체로서** File **객체이다. 파일 목록은 목표에 놓여지는 일련의 파일 아이콘들을 설명한다. 다음은 파일들을 추출하는 코드이다:**

```
DataFlavor[] flavors = transferable.getTransferDataFlavors();
DataFlavor flavor = flavors[i];
if(flavor.equals(DataFlavor.javaFileListFlavor))
{       java.util.List fileList = (java.util.List)transferable.getTransferData(flavor);
        Iterator iterator = fileList.iterator();
```

```
        While(iterator.hasNext() )
        {       File f = (File)iterator.next();
                do something with f;
        }
}
```

놓기 가능한 다른 플레버는 텍스트이다. 그러나 클립보드와 달리 드레그-앤-드롭 메커니즘은 텍스트를 유니코드로 변환하기 위한 노력을 하지 않는 stringFlavor를 제공한다. 대신에 여러분은 MIME 타입 text/plain의 플레버를 얻는다. 여러분은 InputStream을 통해 텍스트를 추출한다. 물론 입력 스트림은 일련의 연속되는 바이트들 이다. 반면에 텍스트는 연속되는 문자들이다. text/plain MIME 타입은 문자 인코딩을 나타내는 charset 전달인자를 포함하고 있다. 이론적으로, 여러분은 입력 스트림과 문자 인코딩을 InputStreamReader 생성자에게 전달하고 한 번에 한 문자씩 입력을 읽을 수 있다.

```
if(flavor.isMimeTypeEqual(“text/plain”))
{       String chatset = flavor.getParameter(“charset”);
        InputStream inStream = (InputStream)transferable.getTransferData(flavor);
        InputStreamReader in = new InputStreamReader(inStream, charset);// doesn’ t work
        int ch;
        while( (ch = in.read()) != -1)
        { do something with ch
        }
}
```

그러나 MIME 인코딩 이름들은 InputStreamReader에서 기대했던 것과는 다르다.

우리는 MIME 타입들에서 두개의 문자 인코딩을 만나게 된다.

```
text/plain; charset=ascii
test/plain; chsrset=unicode
```

InputStreamReader는 이들 문자 셋을 알지 못한다. 여러분은 ascii를 8859_1로 unicode를 unicode로 변환할 필요가 있다.

경고 : 윈도우하에서, 여러분은 이러한 접근방법을 사용할 때 몇가지 문제에 봉착할 것이다. 입력 스트림은 널문자로 종료된다. 그리고 여러분은 지나간 것을 읽을 수 없다. 유니코드 데이터는 InputStreamReader가 기대했던 유니코드 바이트 순서로 시작하지 않는다. 결관,ㄴ

sun.io.MalformedInputException이다. 예제 7-16의 코드는 InputStreamReader를 사용하지 않고 일련의 바이트코드 쌍으로부터 유니코드 문자들을 구성하는 것을 보여준다. 우리는 윈도우를 나타내는 바이트 순서 마커가 없다고 가정하기 때문에 리틀 인디언(little-endian) 바이트 순서를 사용한다. 희망스럽게도, 드레그-앤-드롭 구현의 다음 버전은 이러한 문제들을 사용자들에게 감추고 편리하게 stringFlavor를 반환할 것이다.

드레그 소스에 따라, 여러분은 다음 포맷들의 데이터를 발견할 것이다.

text/html

text/enriched

여러분은 똑 같은 방법으로 데이터를 읽는다.

우리의 예제 프로그램은 다른 유용한 것을 수행하지 않는다. 단지 여러분들에게 아이템들을 텍스트 영역으로 놓을 수 있게 해준다. 일단 여러분이 드롭을 초기화하면, 놓여진 데이터가 출력된다. 만일 데이터가 텍스트 포맷이면, 프로그램은 ascii와 Unicode 인코딩을 모두 읽는다.

우리의 예제 프로그램에서, 우리는 rejectDrag 메소드를 호출할 때 자동으로 발생하는 경고 커져에 대한 변화를 넘는 드레그 과정의 비쥬얼 피드백을 제공하지 않는다.

이 프로그램의 목적은 여러분에게 실험적으로 목표를 떨어뜨리는 기능을 제공한다. 윈도우 익스플로러나 워드패드의 텍스트 조각으로부터 선택한 파일 이름들을 떨어뜨려보아라(그림 7-48참조). 또한 링크 시도가 어떻게 거절되는 지 살펴보라. 만일 여러분이 시프트와 컨트롤 키를 모두 해제하면 텍스트 영역상에 아이템을 끌어 놓을 때 경고 아이콘이 나타난다.

예제7-16은 완전한 프로그램을 보여준다.

그림 7-48 : DropTargetTest 프로그램

예제 7-16 : DropTargetTest.java

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

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

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

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

        Container  contentPane  =  getContentPane();
        JTextArea  textArea
          = new JTextArea("Drop items into this text area.\n");

        new  DropTarget(textArea,
            new  TextDropTargetListener(textArea));
        contentPane.add(new  JScrollPane(textArea),  "Center");
    }
}


class TextDropTargetListener implements DropTargetListener
{   public  TextDropTargetListener(JTextArea ta)
    {   textArea  =  ta;
    }
```

```
public void dragEnter(DropTargetDragEvent event)
{   int a = event.getDropAction();
    if ((a & DnDConstants.ACTION_COPY) != 0)
          textArea.append("ACTION_COPY\n");
    if ((a & DnDConstants.ACTION_MOVE) != 0)
          textArea.append("ACTION_MOVE\n");
    if ((a & DnDConstants.ACTION_LINK) != 0)
          textArea.append("ACTION_LINK\n");

     if (!isDragAcceptable(event))
      {   event.rejectDrag();
          return;
      }
 }

public void dragExit(DropTargetEvent event)
 {
 }

public void dragOver(DropTargetDragEvent event)
{  // you can provide visual feedback here
 }

public void dropActionChanged(DropTargetDragEvent event)
 {   if (!isDragAcceptable(event))
      {   event.rejectDrag();
          return;
      }
 }

public void drop(DropTargetDropEvent event)
 {   if (!isDropAcceptable(event))
      {   event.rejectDrop();
          return;
      }
```

```
 event.acceptDrop(DnDConstants.ACTION_COPY);

Transferable transferable = event.getTransferable();

DataFlavor[] flavors
    = transferable.getTransferDataFlavors();
for (int i = 0; i < flavors.length; i++)
{   DataFlavor d = flavors[i];
   textArea.append("MIME type=" + d.getMimeType() + "\n");

     try
     {   if (d.equals(DataFlavor.javaFileListFlavor))
        {   java.util.List fileList = (java.util.List)
                transferable.getTransferData(d);
            Iterator iterator = fileList.iterator();
             while (iterator.hasNext())
            {   File f = (File)iterator.next();
                textArea.append(f + "\n");
             }
         }
        else if (d.equals(DataFlavor.stringFlavor))
        {   String s = (String)
                transferable.getTransferData(d);
            textArea.append(s + "\n");
         }
        else if (d.isMimeTypeEqual("text/plain"))
        {   String charset = d.getParameter("charset");
            InputStream in = (InputStream)
                transferable.getTransferData(d);

            boolean more = true;
             int ch;
             if (charset.equals("ascii"))
             {   do
                 {   ch = in.read();
```

```
                    if (ch != 0 && ch != -1)
                        textArea.append("" + (char)ch);
                    else more = false;
                } while (more);
            }
            else if (charset.equals("unicode"))
            {   boolean littleEndian = true;
                    // if no byte ordering mark, we assume
                    // Windows is the culprit
                do
                {   ch = in.read();
                    int ch2 = in.read();
                    if (ch != -1 && littleEndian)
                        ch = (ch & 0xFF) | ((ch2 & 0xFF) << 8);
                    if (ch == 0xFFFE)
                        littleEndian = false;
                else if (ch != 0 && ch != -1)
                        textArea.append("" + (char)ch);
                    else more = false;
                } while (more);
            }

            textArea.append("\n");
        }
      }
      catch(Exception e)
    {   textArea.append("Error: " + e + "\n");
      }
   }
   event.dropComplete(true);
}

public boolean isDragAcceptable(DropTargetDragEvent event)
{  // usually, you check the available data flavors here
   // in this program, we accept all flavors
   return (event.getDropAction()
```

```
            & DnDConstants.ACTION_COPY_OR_MOVE) != 0;
    }

   public boolean isDropAcceptable(DropTargetDropEvent event)
   {  // usually, you check the available data flavors here
      // in this program, we accept all flavors
       return (event.getDropAction()
            & DnDConstants.ACTION_COPY_OR_MOVE) != 0;
    }

   private JTextArea textArea;
}
```

java.awt.dnd.DropTarget

- DropTarget(Component c, DropTargetListener listener)
  컴포넌트상에 드레그와 놓기 행위의 조합에 따라 놓기 목표를 구성한다.
  전달인자: c **놓기 목표**
  listener **놓기 과정에 알려질 리스너**
- void setActive(boolean b)
  해당 놓기 목표를 활성화 또는 비활성화한다.
- void setDefaultActions(int actions)
  해당 놓기 목표에 대해 기본적으로 접근가능한 행위를 설정한다. Actions는 ACTION_COPY, ACTION_MOVE, ACTION_COPY_OR_MOVE, **또는** ACTION_LINK**처럼** DnDConstants **클래스내에 정의되어 있는 상수들의 비트 마스트로 구성된다.**

java.awt.dnd.DropTargetListener

- void dragEnter(DropTargetDragEvent event)
  커져가 놓기 목표에 진입할 때 호출된다.
- void dragExit(DropTargetEvent event)
  커져가 놓기 목표에 벗어 날 때 호출된다.
- void dragOver(DropTargetDragEvent event)
  커져가 놓기 목표상으로 이동할 때 호출된다.
- void dragActionChanged(DropTargetDragEvent event)
  커져가 놓기 목표상에 있는 동안 사용자가 놓기 행동을 변경할 때 호출된다.
- void drop(DropTargetDragEvent event)
  사용자가 아이템들을 놓기 목표에 놓았을 때 호출된다.

java.awt.dnd.DropTargetDragEvent

- int getDropAction()

  현재 선택된 놓기 액션을 얻는다. 가능한 값들은 DnDConstants 클래스에 정의되어 있다.

- void acceptDrag(int action)

  만일 놓기 목표가 현재 선택된 행위와 다른 놓기 액션을 받아들이길 원할 때 이 메소드가 호출될 것이다.

- void rejectDrag()

  해당 컴포넌트가 현재 놓기 시도를 거절한다는 것을 드레그-앤-드롭 메커니즘에 통보한다.

- Point getLocation()

  놓기 목표상에 마우스의 현재 위치를 반환한다.

- DataFlavor() getCurrentDataFlavors()

  드레그 소스가 전달할 수 있는 데이터 플레버들을 반환한다.

- boolean isDataFlavorSupported(DataFlavor flavor)

  드레그 소스가 주어진 플레버를 지원하는 지에 대한 여부를 검사한다.

java.awt.dnd.DropTargetDropEvent

- int getDropAction()

  현재 선택된 놓기 행위를 얻는다. 가능한 값은 DnDConstants 클래스에 정의되어 있다.

- void acceptDrop(int action)

  이 메소드는 놓기 목표가 현재 선택된 행동과 다른 놓기 행동을 수행할 때 호출된다.

- void rejectDrop()

  해당 컴포넌트가 드롭을 거절한다는 것을 드레그-앤-드롭 메커니즘에 통보한다.

- void dropComplete(boolean success)

  놓기가 완성되고 성공했는 가에 대한 여부를 드레그 소스에 통보한다.

- Point getLocation()

  놓기 목표상에 마우스의 현재 위치를 반환한다.

- DataFlavor() getCurrentDataFlavors()

  드레그 소스가 전달할 수 있는 데이터 플레버를 반환한다.

- boolean isDataFlavorSupported(DataFlavor flavor)

  드레그 소스가 주어진 플레버를 지원하는 지 검사한다.

드래그 소스

이제 여러분은 놓기 목표를 포함하고 있는 프로그램을 구현하는 방법을 보았다. 우리는 여러분에게 드레그 소스를 구현하는 방법을 보여주길 원한다.

예제 7-17 **프로그램은** Jlist**를 현재 디렉토리에 있는 모든 파일들로 채운다(** **그림**7-49**참조). 리스트 컴포넌트는 드레그 소스이다. 여러분은 리스트 컴포넌트로부터 파일 아이템들을 일련의 파일들을 받아 들일 수 있는 모든 놓기 목표로 끌어다 놓을 수 있다.**

컴포넌트를 드레그 소스로 바꾸기 위해, 여러분은 DragSource **객체를 얻을 필요가 있다 – 여러분은** DragSource.getDefaultDragSource **정적 메소드를 호출할 수 있다. 그러면** createDefaultDragGestureRecognizer **메소드를 호출하고 다음을 제공한다.**

- **여러분이 드레그 소스로 변경하고자 하는 컴포넌트**
- **여러분이 허락하고자 하는 놓기 행위**
- DragGestureListener **인터페이스를 구현한 객체**

가령,

```
DragSource dragSource = DragSource.getDefaultDragSource();
dragSource.createDefaultDragGestureRecognizer( component,
        DnDConstants.ACTION_COPY_OR_MOVE, dragGestureListener);
```

DragGestureListener **인터페이스는** DragGestureRecognizer**라는 메소드 하나를 가지고 있다. 제스츄어 인식자는 사용자가 드레그 연산을 초기화하길 원한다는 것을 알자마자 이 메소드를 호출한다. 이 메소드에서,** 여러분은 **최종적으로** drop **메소드내에서 읽을 놓기 목표인** transferable**를 구성할 필요가 있다. 일단 여러분이** Transferable**을 모으면,** DragGestureEvent **클래스의** startDrag **메소드를 호출한다. 여러분은 추가적인 커져나 기본 드레그 커져를 원하지 않으면** null**을** Transferable**과** DragSourceListener **인터페이스와 함께 제공할 수 있다.**

```
event.startDrag(null, transferable, dragSourceListener);
```

드레그 소스 리스너는 드레그 연산이 진행될 때 반복해서 통보를 받는다. 우리는 잠시 이 부분을 살펴볼 것이다. 그러나 먼저 우리는 여러분에게 파일들의 목록을 유지할 수 있는 Transferable**을 구축하는 방법을 보여준다.**

여러분은 Transferable **인터페이스를 구현하고** DataFlavor.javaFileListFlavor **표준 플레버를 지원하는 클래스를 정의할 필요가 있다.** GetTransferData **메소드는** File **항목들을 채울**

java.util.List 인터페이스를 구현하는 클래스의 객체를 반환할 필요가 있다. 다음은 이러한 클래스의 단순한 버전이다. 우리는 단순히 ArrayList에 파일들의 배열을 저장하고 플래버가 DataFlavor.javaFileListdhk 일치할 때 목옥을 반환한다.

```
class FileListTransferable implements Transferable
{  public FileListTransferable(Object[] files)
   {  fileList = new ArrayList(Arrays.asList(files));
    }

   public DataFlavor[] getTransferDataFlavors()
    {  return flavors;
    }

   public boolean isDataFlavorSupported(DataFlavor flavor)
    {  return Arrays.asList(flavors).contains(flavor);
    }

   public synchronized Object getTransferData
       (DataFlavor flavor)
       throws UnsupportedFlavorException
    {  if(flavor.equals(DataFlavor.javaFileListFlavor))
       {  return fileList;
       }
       else if(flavor.equals(DataFlavor.stringFlavor))
       {  return fileList.toString();
       }
       else
       {  throw new UnsupportedFlavorException(flavor);
       }
    }

   private static DataFlavor[] flavors =
       {  DataFlavor.javaFileListFlavor,
          DataFlavor.stringFlavor
       };
```

```
    private java.util.List fileList;
}
```

예제 7-17의 클래스는 stringFlavor를 지원하기 때문에 다소 복잡하다.

이제 DragSourceListener 인터페이스로 돌아가 보자. 인터페이스는 5개의 메소드들을 가지고 있다.

```
void dragEnter(DragSourceDragEvent event)
void dragOver(DragSourceDragEvent event)
void dragExit(DragSourceEvent event)
void dragActionChanged(DragSourceDragEvent event)
void dragDropEnd(DragSourceDropEvent event)
```

여러분은 첫번째 4개의 메소드를 드레그 연산의 사용자 비쥬얼 피드백을 제공하는 데 사용할 수 있다. 그러나, 일반적으로 이와 같은 피드백은 놓기 목표의 역할이어야 한다. 우리는 이러한 메소드들을 우리의 예제 프로그램에서 사용할 것이다. 이들 메소드는 drop 메소드가 종료될 때 호출된다. 이동 연산을 위해, 여러분은 drop이 성공했는지 검사할 필요가 있다. 이 경우, 여러분은 드레그 소스를 갱신해야 한다.(복사 연산을 위해서는 아마도 아무것도 할 필요가 없다.)

다음은 우리의 예제 프로그램의 dragDropEnd 메소드이다. 이동이 성공했을 때, 우리는 리스트 모델로부터 이동된 아이템들을 제거한다.

```
Public void dragDropEnd(DragSourceDropEvent event)
{       if(event.getDropSuccess())
        {       int action = event.getDropAction();
                if(action == DnDConstants.ACTION_MOVE)
                {       for(int i=0; i < draggedValues.length; i++)
                                model.removeElement(draggedValues[i]);
                }
        }
}
```

이 메소드에서, 우리는 어떤 놓기가 실제 수행됐는지 알기 위해 drop 메소드에 의존한다. drop 메소드는 소스가 이동과 복사 행위를 모두 지원하면 이동 행위를 복사 행위로 변경할 수 있다. DragSourceDropEvent 클래스의 event.getDropAction은 DropTargetDropEvent의 acceptDrop 메소드를 호출했을 때 보고된 놓기 목표의 행위를 반환한다.

예제 7-16 프로그램이사 윈도우 익스플로러나 워드패드 같은 공유 프로그램에서 처럼 예제 7-17의 프로그램에서 파일 아이템들을 다양한 놓기 목표들에 끌어보아라.

노트 : 여러분이 드레그를 시도할 때, 드레그 제스츄어가 리스트 컨트롤의 일반 마우스 효과와 교감이 없다는 것에 주의해야 한다. 하나 또는 그 이상의 아이템들을 선택한다. 선택된 아이템상에 마우스 버튼을 누르고 리스트 컴포넌트를 벗어날 때까지 마우스를 이동한다. 이제 드레그 제스츄어가 인식된다. 마우스가 리스트 컴포넌트를 벗어난 후에, 놓기 행위를 변경하기 위해 시프트와 컨트롤 키를 해제한다.

경고 : 기본 놓기 행위는 이동이다. 만일 여러분이 리스트 컴포넌트로부터 파일 아이템을 끌어다 윈도우 익스플로러에 놓는다면, 익스플로러는 파일을 목표 폴더로 이동한다.

여러분이 보았듯이, 시스템 클립보드와 드레그-앤-드롭 메커니즘은 상당부분 더 진행되야 한다. 그러나 희망스럽게도 자바 플랫폼의 향후 버전에서는 보다 강건하고 설득력이쓴 지원을 제공할 것이다.

이 절에서, 우리는 드레그-앤-드롭 메커니즘의 기본을 소개했다. 보다 자세한 정보는 , 특히 비쥬얼 피드백에 대한, 다음 책을 추천한: Core Swing: Asvanced Programming , Kim Topley[Prentice Hall].

그림 7-49 : DragSourceTest 프로그램

예제 7-17 : DragSourceTest.java

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

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

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

class DragSourceFrame extends JFrame
   implements DragSourceListener, DragGestureListener
{  public DragSourceFrame()
    {  setTitle("DragSourceTest");
       setSize(300, 200);
       addWindowListener(new WindowAdapter()
          {  public void windowClosing(WindowEvent e)
             {  System.exit(0);
             }
          } );

       Container contentPane = getContentPane();
       File f = new File(".").getAbsoluteFile();
       File[] files = f.listFiles();
       model = new DefaultListModel();
      for (int i = 0; i < files.length; i++)
          model.addElement(files[i]);
       fileList = new JList(model);
       contentPane.add(new JScrollPane(fileList), "Center");
      contentPane.add(new JLabel("Drag files from this list"),
          "North");

       DragSource dragSource = DragSource.getDefaultDragSource();
        dragSource.createDefaultDragGestureRecognizer(fileList,
          DnDConstants.ACTION_COPY_OR_MOVE, this);
```

```
 }

// DragGestureListener method

 public void dragGestureRecognized(DragGestureEvent event)
{   draggedValues = fileList.getSelectedValues();
     Transferable transferable
         = new FileListTransferable(draggedValues);
    event.startDrag(null, transferable, this);
 }

// DragSourceListener methods

public void dragEnter(DragSourceDragEvent event)
 {
 }

public void dragOver(DragSourceDragEvent event)
 {
 }

public void dragExit(DragSourceEvent event)
 {
 }

public void dropActionChanged(DragSourceDragEvent event)
 {
 }

public void dragDropEnd(DragSourceDropEvent event)
 {   if (event.getDropSuccess())
    {   int action = event.getDropAction();
        if (action == DnDConstants.ACTION_MOVE)
       {   for (int i = 0; i < draggedValues.length; i++)
                model.removeElement(draggedValues[i]);
          }
```

```
        }
    }

    private JList fileList;
    private DefaultListModel model;
    private Object[] draggedValues;
}

class FileListTransferable implements Transferable
{   public FileListTransferable(Object[] files)
    {   fileList = new ArrayList(Arrays.asList(files));
    }

    public DataFlavor[] getTransferDataFlavors()
    {   return flavors;
    }

    public boolean isDataFlavorSupported(DataFlavor flavor)
    {   return Arrays.asList(flavors).contains(flavor);
    }

    public synchronized Object getTransferData
        (DataFlavor flavor)
        throws UnsupportedFlavorException
    {   if(flavor.equals(DataFlavor.javaFileListFlavor))
        {   return fileList;
        }
        else if(flavor.equals(DataFlavor.stringFlavor))
        {   return fileList.toString();
        }
        else
        {   throw new UnsupportedFlavorException(flavor);
        }
    }

    private static DataFlavor[] flavors =
```

```
        {   DataFlavor.javaFileListFlavor,
            DataFlavor.stringFlavor
        };

    private java.util.List fileList;
}
```

java.awt.dnd.DragSource

- static DragSource getDefaultDragSource()
  드레그 행위와 컴포넌트들의 조절하는 DragSource**를 얻는다**.
- DragGestureRecognizer createDefaultDragGestureRecognizer(Component component , int actions, DragGestureListener listener)
  드레그 제스츄어 인식자를 생성한다.
  전달인자: component **드레그 소스**
  actions **수행가능한 놓기 행위들**
  listener **드레그 제시츄어가 인식됐을 때 알려질 리스너**

java.awt.dnd.DragGestureListener

- void dragGestureRecognized(DragGestureEvent event)
  드레그 제스츄어 인식자가 제스츄어를 인식할 때 호출된다.

java.awt.dnd.DragGestureEvent

- void startDrag(Cursor dragCursor, Transferable transferable, DragSourceListener listener)
  전달인자: dragCursor **드롭을 위해 사용되는 선택적 커져**, null**이면 기본 커져가 사용된다**.
  transferable **놓기 목표에게 전송될 데이터**
  listener **드레그 과정을 알려 줄 리스너**

java.awt.dnd.DragSourceListener

- void dragEnter(DragSourceDragEvent event)
  드레그 커져가 드레그 소스로 진입했을 때 호출된다.
- void dragExit(DragSourceEvent event)
  드레그 커져가 드레그 소스로부터 벗어났을 때 호출된다.
- void dragOver(DragSourceDragEvent event)
  드레그 커져가 드레그 소스상에 놓여있을 때 호출된다.
- void dragActionChanged(DragSourceDragEvent event)
  사용자가 놓기 행위를 변경했을 때 호출된다.
- void dragDropEnd(DragSourceDropEvent event)

드레그-앤-드롭 연산이 종료되거나 철회됐을 때 호출된다.

java.awt.dnd.DragSourceDropEvent

- boolean getDropSuccess()

  만일 놓기 목표가 성공적으로 놓여졌다면 true를 반환한다.

- int getDropAction()

  놓기 목표를 실제 수행한 행위를 반환한다.