<JSTORM>

자바 프린팅 모델 2부

중앙대학교 컴퓨터공학과
자바 동호회
JSTORM
http://www.jstorm.pe.kr

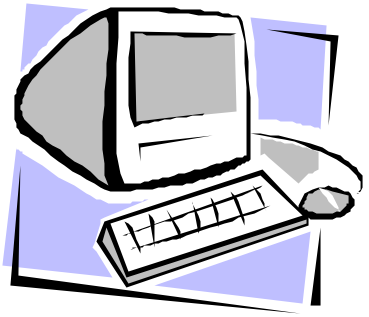
Document Information

Document title:	자바 프린팅 모델 2부
Document file name:	JavaPrintingModel2_JunoYoon.doc
Revision number:	<1.0>
Issued by:	<윤준호> junoyoon@orgio.net
Issue Date:	<2000/12/21>
Status:	final

Content Information

Audience	자바 중급이상
Abstract	5부에 걸친 자바 프린팅 모델 강좌 중 2번째 강좌에서 저자는 독자에게 어떻게 프린팅을 해볼 수 있는지 알려줄 것이다. 또한 Java 고급 Printing API의 사용법도 알아볼 것이다.
Reference	Printing in Java, Part 2 (http://www.javaworld.com/javaworld/jw-10-2000/jw-1020-print_p.html) 번역 by **Jean-Pierre Dubé**
Benchmark information	

Table of Contents



파트 2

일단 페이지를 출력해보고 복잡한 문서를 렌더링 해보자.

이번 달에는 파트 1에서 배웠던 것들을 실습해볼 차례이다. 파트 1에서는 Printable과 Pageable 2가지 프린팅 모델에 대하여 알아보았다. 또 문서를 만들기 위해서 Book 클래스를 어떻게 사용하는지도 살펴보았다. 파트 2 에서는 실제로 페이지를 한번 출력해 볼 것이고 자바 2D API를 이용하여 좀 더 복잡한 문서를 렌더링 해볼 것이다. 또한 지난달에 약속했듯이 폰트를 어떻게 사용 하는 지와 그 관련클래스에 대해서도 설명할 것이다. 다른 플랫폼에서의 프린팅과 관련된 이슈들에 대해서도 살펴 볼 것이다. 여기 있는 프로그램들은 Sun의 Java 1.3으로 Windows 2000과 Linux상에서 컴파일하고 실행되었다.

첫번째 페이지를 출력해 보자.

파트 1에서 이미 Printable과 Pageable 모델에 대하여 알아 보았다. Printable을 통해서는 간단히 문서를 출력할 수 있다. 그러나 Printable 모델은 몇 가지 한계를 가지고 있다. 대표적으로 Printable 모델로 출력할 때는 모든 페이지들이 같이 포맷 이어야 한다. 이와는 반대로 Pageable 모델은 좀 더 유연하다. Book 클래스와 결합하여 각각 다른 포맷을 가진 여러 페이지 문서를 만들 수 있다.
리스트 1은 Printable 모델을 이용하여 어떻게 프린트를 하는지 보여주고 있다.

리스트 1

```
1|
2|/**
3| * Class: Example1 <p>
4| *
```

```
 5| * @author Jean-Pierre Dube <jpdube@videotron.ca>
 6| * @version 1.0
 7| * @since 1.0
 8| * @see Printable
 9| */
10|
11|import java.awt.*;
12|import java.awt.geom.*;
13|import java.awt.print.*;
14|
15|
16|public class Example1 implements Printable {
17|
18| // Private 인스턴스 선언
19| private final double INCH = 72;
20|
21|
22|
23| /**
24| * Constructor: Example1 <p>
25| *
26| */
27| public Example1 () {
28|
29| //--- printerjob 객체를 만든다.
30| PrinterJob printJob = PrinterJob.getPrinterJob ();
31|
32| //--- Example1이 Printable 인터페이스를 구현했기 때문에
33| //--- Printable 클래스로 this를 할당한다.
34| printJob.setPrintable (this);
35|
36| //--- 유저에게 프린트 다이얼로그를 보여준다. 만약 유저가 프린트
37| //--- 버튼을 클릭하면 프린트 된다. 그렇지 않으면 프린트 작업은
38| //--- 취소된다.
39| if (printJob.printDialog()) {
40|         try {
41|                 printJob.print();
42|         } catch (Exception PrintException) {
43|                 PrintException.printStackTrace();
44|         }
45| }
46|
47| }
```

```
48|
49|
50| /**
51| * Method: print <p>
52| *
53| * 이 클래스는 주어진 파라메터를 이용하여 페이지를 렌덩링하는
54| * 역할을 담당한다. 결과로 각 셀이 1/2 인치 * 1/2 인치인
55| * 그리드가 출력되게 된다.
56| *
57| * @param g a value of type Graphics
58| * @param pageFormat a value of type PageFormat
59| * @param page a value of type int
60| * @return a value of type int
61| */
62| public int print (Graphics g, PageFormat pageFormat, int page) {
63|
64| int i;
65| Graphics2D g2d;
66| Line2D.Double line = new Line2D.Double ();
67|
68| // 페이지번호를 확인한다. 여기서는 단지 첫번째 페이지만을 출력한다.
69| if (page == 0) {
70|
71| //--- graphic2D 객체를 만들고 디폴트 파라메터를 설정한다.
72| g2d = (Graphics2D) g;
73| g2d.setColor (Color.black);
74|
75| //--- 기준점을 (0,0)으로 변경한다.
76| g2d.translate(pageFormat.getImageableX(),
pageFormat.getImageableY());
77|
78| //--- 세로 선들을 출력한다.
79| for (i = 0; i < pageFormat.getWidth (); i += INCH / 2) {
80|         line.setLine (i, 0, i, pageFormat.getHeight ());
81|         g2d.draw (line);
82| }
83|
84| //--- 가로 선들을 출력한다.
85| for (i = 0; i < pageFormat.getHeight (); i += INCH / 2) {
86|         line.setLine (0, i, pageFormat.getWidth (), i);
87|         g2d.draw (line);
88| }
89|
```

```
90| return (PAGE_EXISTS);
91| }
92| else
93| return (NO_SUCH_PAGE);
94| }
95|
96|} //Example1
```

리스트 1의 소스는 디폴트 여백 설정사항을 사용하고 1/2인치 * 1/2인치 짜리 그리드들이 출력되게 된다. 이 예제를 자바 1.2로 돌려보고자 할 때는 마진이 정확히 맞지 않는다는 것을 주의하라. 자바 1.2 프린팅 API의 버그가 이런 결과를 만들어 내고 있다.

이제 Printable 모델을 이용하여 프린팅할 때 필요한 단계들을 살펴보도록 하자.

- PrinterJob 객체를 만든다이 객체는 페이지, 프린트 다이얼로그를 디스플레이하고,프린트 액션을 초기화 함으로써 프린트 프로세스를 제어한다.
- 적합한 다이얼로그 박스를 출력한다. 다이얼로그 박스 종류로는 프린트, 페이지 다이얼로그가 있다.
- Printable의 print() 메소드를 구현한 클래스를 만든다.
- 렌더링될 페이지 번호를 확인한다.
- Graphics 파라미터를 이용하여 페이지를 렌더링한다.

페이지가 렌더링 되었다면 PAGE_EXIT 값을 리턴하고 렌더링되지 못했다면 NO_SUCH_PAGE 값을 리턴한다.

Pageable과 Books을 사용하여 프린트 하자.

앞의 예제에서 살펴봤듯이 Printable을 이용한 프린팅은 직관적이다. 그러나 Printable모델은 좀 더 복잡한 문서를 다룰 때에 유연성을 제공하지 못하고 있다. 이러한 작업을 할 때에는 Pageable과 Book 클래스가 여러모로 쓸모가 많을 것이다. 이번 연재의 파트 1에서 설명했듯이 Book 클래스는 각 Book의 페이지가 서로 다른 페인터(painter)를 사용하거나 다른 페이지의 페인터를 사용할 것인지 선택하는 것을 가능하게 해준다. Pageable과 Book을 어떻게 사용하는지 예제를 통해 살펴보도록 하자.

Listing 2

```
1|
2|/**
3| * Class: Example2 <p>
4| *
5| * Print in Java series for the JavaWorld magazine.
6| *
```

```
 7| * @author Jean-Pierre Dube <jpdube@videotron.ca>
 8| * @version 1.0
 9| * @since 1.0
10| */
11|
12|import java.awt.*;
13|import java.awt.font.*;
14|import java.awt.geom.*;
15|import java.awt.print.*;
16|
17|
18|public class Example2 {
19|
20| //--- Private 인스턴스 선언
21| private final static int POINTS_PER_INCH = 72;
22|
23|
24| /**
25| * Constructor: Example2 <p>
26| *
27| */
28| public Example2 () {
29|
30| //--- PrinterJob 객체를 만든다.
31| PrinterJob printJob = PrinterJob.getPrinterJob ();
32|
33| //--- 페이지(page)가 추가될 새로운 book 인스턴스를 선언한다.
34| Book book = new Book ();
35|
36| //--- 이 프린트 Job의 디폴트 페이지 포맷을 사용하는 겉장 페이지를 추가한다.
37| book.append (new IntroPage (), printJob.defaultPage ());
38|
39| //--- 문서(document) 페이지를 가로 페이지 포맷을 이용하여 추가한다.
40| PageFormat documentPageFormat = new PageFormat ();
41| documentPageFormat.setOrientation (PageFormat.LANDSCAPE);
42| book.append (new Document (), documentPageFormat);
43|
44| //--- printJob에 Pageable 객체로 Book을 사용하겠다고 알린다.
45| printJob.setPageable (book);
46|
47| //--- 프린트 다이얼로그 박스를 출력한다. 유저가 print 버튼을 누르면
48| //--- 프린트를 진행시키고 cancel 버튼을 누르면 프린트 프로세스를
49| //--- 취소한다.
```

```
50| if (printJob.printDialog()) {
51|         try {
52|                 printJob.print();
53|         } catch (Exception PrintException) {
54|                 PrintException.printStackTrace();
55|         }
56| }
57| }
58|
59|
60| /**
61| * Class: IntroPage <p>
62| *
63| * 이 클래스는 Printable 인터페이스를 구현함으로써
64| * 겉장 페이지(cover page)의 페인터(painter)를 정의한다.. <p>
65| *
66| * @author Jean-Pierre Dube <jpdube@videotron.ca>
67| * @version 1.0
68| * @since 1.0
69| * @see Printable
70| */
71| private class IntroPage implements Printable {
72|
73|
74| /**
75| * Method: print <p>
76| *
77| * @param g a value of type Graphics
78| * @param pageFormat a value of type PageFormat
79| * @param page a value of type int
80| * @return a value of type int
81| */
82| public int print (Graphics g, PageFormat pageFormat, int page) {
83|
84| //---  Graphics2D 인스턴스를 생성한다.
85| Graphics2D g2d = (Graphics2D) g;
86|
87| //--- 좌상측(0, 0)으로 좌표축(기준점)을 옮긴다.
88| g2d.translate (pageFormat.getImageableX (), pageFormat.getImageableY
());
89|
90| //--- 디폴트 드로잉 색상을 검정색으로 설정한다.
91| g2d.setPaint (Color.black);
```

```
 92|
 93| //--- 페이지의 경계선을 그린다.
 94| Rectangle2D.Double border = new Rectangle2D.Double (0,
 95| 0,
 96| pageFormat.getImageableWidth (),
 97| pageFormat.getImageableHeight ());
 98| g2d.draw (border);
 99|
100| //--- 제목을 출력한다.
101| String titleText = "Printing in Java Part 2";
102| Font titleFont = new Font ("helvetica", Font.BOLD, 36);
103| g2d.setFont (titleFont);
104|
105| //--- 페이지의 가로 중심점을 계산한다.
106| FontMetrics fontMetrics = g2d.getFontMetrics ();
107| double titleX = (pageFormat.getImageableWidth () / 2) -
(fontMetrics.stringWidth (titleText) / 2);
108| double titleY = 3 * POINTS_PER_INCH;
109| g2d.drawString (titleText, (int) titleX, (int) titleY);
110|
111| return (PAGE_EXISTS);
112| }
113| }
114|
115|
116|
117| /**
118| * Class: Document <p>
119| *
120| * 이 클래스는 document 내용의 페인터이다..<p>
121| *
122| *
123| * @author Jean-Pierre Dube <jpdube@videotron.ca>
124| * @version 1.0
125| * @since 1.0
126| * @see Printable
127| */
128| private class Document implements Printable {
129|
130|
131| /**
132| * Method: print <p>
133| *
```

```
134| * @param g a value of type Graphics
135| * @param pageFormat a value of type PageFormat
136| * @param page a value of type int
137| * @return a value of type int
138| */
139| public int print (Graphics g, PageFormat pageFormat, int page) {
140|
141|
142| //--- Graphics2D 인스턴스를 만든다.
143| Graphics2D g2d = (Graphics2D) g;
144|
145| //--- 좌상측(0, 0)으로 좌표축(기준점)을 옮긴다.
146| g2d.translate (pageFormat.getImageableX (),
pageFormat.getImageableY ());
147|
148| //--- 드로잉 색상을 검정색으로 설정한다.
149| g2d.setPaint (Color.black);
150|
151| //--- 페이지 주변으로 12포인트의 두께를 가진 경계선을 그린다.
152| g2d.setStroke (new BasicStroke (12));
153| Rectangle2D.Double border = new Rectangle2D.Double (0,
154| 0,
155| pageFormat.getImageableWidth (),
156| pageFormat.getImageableHeight ());
157|
158| g2d.draw (border);
159|
160| //--- 텍스트를 좌상측 마진으로부터 1인치 위치에 출력한다.
161| g2d.drawString ("This the content page", POINTS_PER_INCH,
POINTS_PER_INCH);
162|
163| //--- 페이지를 Validate 한다.
164| return (PAGE_EXISTS);
165|
166| }
167| }
168|
169|} // Example2
```

이 예제는 겉장(cover) 페이지 페인터와 문서(document) 자체의 페인터를 이용하여 2페이지의 문서를 출력하게 된다. 겉장 페이지는 세로로 출력되게 되고 2번째 페이지는 가로로 출력되게 되고 각 페인터는 1페이지만 출력한다. 1페이지 이상을 출력하고 싶을 경우에는 다음 예제를 살펴보도록 하자.

Listing 3

```
 1|
 2|
 3|/**
 4| * Class: Example3 <p>
 5| *
 6| * @author Jean-Pierre Dube <jpdube@videotron.ca>
 7| * @version 1.0
 8| * @since 1.0
 9| */
10|
11|import java.awt.*;
12|import java.awt.font.*;
13|import java.awt.geom.*;
14|import java.awt.print.*;
15|
16|
17|public class Example3 {
18|
19| //--- Private 인스턴스를 선언한다.
20| private final static int POINTS_PER_INCH = 72;
21|
22|
23| /**
24| * Constructor: Example3 <p>
25| *
26| */
27| public Example3 () {
28|
29| //--- PrinterJob 인스턴스를 만든다.
30| PrinterJob printJob = PrinterJob.getPrinterJob ();
31|
32| //--- 페이지를 담을 Book 인스턴스를 만든다.
33| Book book = new Book ();
34|
35| //--- 이 printjob의 디폴트 포맷을 이용하여 출력되는  커버 페이지를 추가한다.
36| book.append (new IntroPage (), printJob.defaultPage ());
37|
38| //--- 가로 페이지 포맷을 이용하여 출력되는 document 페이지를 추가한다.
39| PageFormat documentPageFormat = new PageFormat ();
40| documentPageFormat.setOrientation (PageFormat.LANDSCAPE);
41| book.append (new Document (), documentPageFormat);
```

```
42|
43| //--- 같은 painter를 이용하는 세번째 페이지를 추가한다.
44| book.append (new Document (), documentPageFormat);
45|
46| //--- printJob에 Pageable 객체로 Book을 사용하라고 알린다.
47| printJob.setPageable (book);
48|
49| //--- 프린트 다이얼로그 박스를 출력하고 유저가 프린트 버튼을 누르면
50| //--- 프린트를 진행하고 cancel을 누르면 프로세스를
51| //--- 취소한다.
52| if (printJob.printDialog()) {
53|         try {
54|                 printJob.print();
55|         } catch (Exception PrintException) {
56|                 PrintException.printStackTrace();
57|         }
58| }
59|
60| }
61|
62| /**
63| * Class: IntroPage <p>
64| *
65| * 이 클래스는 Printable 인터페이스를 구현함으로써
66| * 겉장 페이지의 페인터를 정의한다. <p>
67| *
68| * @author Jean-Pierre Dube <jpdube@videotron.ca>
69| * @version 1.0
70| * @since 1.0
71| * @see Printable
72| */
73| private class IntroPage implements Printable {
74|
75|
76| /**
77| * Method: print <p>
78| *
79| * @param g a value of type Graphics
80| * @param pageFormat a value of type PageFormat
81| * @param page a value of type int
82| * @return a value of type int
83| */
84| public int print (Graphics g, PageFormat pageFormat, int page) {
```

```
 85|
 86| //--- Graphics2D 인스턴스를 생성한다.
 87| Graphics2D g2d = (Graphics2D) g;
 88|
 89| //--- 좌표계(기준점)을 좌상단 (0,0)으로 옮긴다.
 90| g2d.translate (pageFormat.getImageableX (), pageFormat.getImageableY
());
 91|
 92| //--- 디폴트 드로잉 색상을 검은색으로 설정한다.
 93| g2d.setPaint (Color.black);
 94|
 95| //--- 페이지 경계선을 그린다.
 96| Rectangle2D.Double border = new Rectangle2D.Double (0,
 97| 0,
 98| pageFormat.getImageableWidth (),
 99| pageFormat.getImageableHeight ());
100| g2d.draw (border);
101|
102| //--- 제목을 출력한다.
103| String titleText = "Printing in Java Part 2";
104| Font titleFont = new Font ("helvetica", Font.BOLD, 36);
105| g2d.setFont (titleFont);
106|
107| //--- 페이지의 가로 중심을 계산한다.
108| FontMetrics fontMetrics = g2d.getFontMetrics ();
109| double titleX = (pageFormat.getImageableWidth () / 2) -
(fontMetrics.stringWidth (titleText) / 2);
110| double titleY = 3 * POINTS_PER_INCH;
111| g2d.drawString (titleText, (int) titleX, (int) titleY);
112|
113| return (PAGE_EXISTS);
114| }
115| }
116|
117|
118|
119| /**
120| * Class: Document <p>
121| *
122| * 이 클래스는 document 의 내용을 위한 페인터이다..<p>
123| *
124| *
125| * @author Jean-Pierre Dube <jpdube@videotron.ca>
```

```
126| * @version 1.0
127| * @since 1.0
128| * @see Printable
129| */
130| private class Document implements Printable {
131|
132|
133| /**
134| * Method: print <p>
135| *
136| * @param g a value of type Graphics
137| * @param pageFormat a value of type PageFormat
138| * @param page a value of type int
139| * @return a value of type int
140| */
141| public int print (Graphics g, PageFormat pageFormat, int page) {
142|
143|
144| //--- Graphics2D 인스턴스를 만든다.
145| Graphics2D g2d = (Graphics2D) g;
146|
147| //--- 좌표계(기준점)을 좌상측 (0, 0)으로 옮긴다.
148| g2d.translate (pageFormat.getImageableX (),
pageFormat.getImageableY ());
149|
150| //--- 드로잉 색상을 검정색으로 설정한다.
151| g2d.setPaint (Color.black);
152|
153| //--- 12 포인트 두께를 가진 경계선을 페이지 가장자리에 그린다.
154| g2d.setStroke (new BasicStroke (12));
155| Rectangle2D.Double border = new Rectangle2D.Double (0,
156| 0,
157| pageFormat.getImageableWidth (),
158| pageFormat.getImageableHeight ());
159|
160| g2d.draw (border);
161|
162|
163| //---페이지 1을 출력한다.
164| if (page == 1) {
165| //--- 좌상측에서 1인치씩 떨어진 곳에 텍스트를 출력한다.
166| g2d.drawString ("This the content page of page: " + page,
POINTS_PER_INCH, POINTS_PER_INCH);
```

```
167| return (PAGE_EXISTS);
168| }
169|
170| //--- 페이지 2를 출력한다.
171| else if (page == 2) {
172| //--- 좌상측에서 1인치씩 떨어진 곳에 텍스트를 출력한다.
173| g2d.drawString ("This the content of the second page: " + page,
POINTS_PER_INCH, POINTS_PER_INCH);
174| return (PAGE_EXISTS);
175| }
176|
177|
178| //--- 페이지를 Validate 한다.
179| return (NO_SUCH_PAGE);
180|
181| }
182| }
183|
184|} // Example3
```

리스트 3 코드의 44번 줄에서 첫번째 페이지와 같은 페인터를 이용하여 출력하는 세 번째 페이지를 추가하였다. 164줄에서부터 171줄까지의 if 문에서는 렌더링할 정확한 페이지를 체크 한다. 이 예제는 간단하기 때문에 렌더링 할 페이지를 확인하는 작업이 어렵지 않다. 그러나 앞의 예제에서 알아 봤듯이 문서를 페이징 하는 것은 대단히 복잡해 질 수 있다. 다음 연재에서 소개될 프린트 프레임 웍은 이러한 작업을 신경 쓸 필요가 없이 출력작업에만 신경 쓰게 하는 기능을 담게 될 것이다.

이제 프린팅 API를 어떻게 사용하여 페이지를 렌더링 하는 지 알게 되었다.
이제는 새로운 것을 알아보도록 하자. 프린터가 여러 개 있을 경우 이중 어느 프린터로 출력 되야 하는지를 유저가 정하게 하는 기능이 필요하다. 또한 페이지 포맷과 여백을 직접 유저가 조정할 수 있는 기회를 주는 것도 중요하다. 자바 프린팅 API에서는 프린터 다이얼로그와 페이지 다이얼로그 총 2개의 다이얼로그 윈도우를 제공하고 있다.

프린트, 페이지 설정 다이얼로그 박스 사용하기

자바의 이번 버전에서는 프린트 다이얼로그 박스만이 필요하였다. 그러나 자바 1.2이 나오면서부터 프린트 다이얼로그 박스 외에 페이지 다이얼로그 박스까지 사용되게 되었다. 페이지 다이얼로그 박스가 등장하면서 아직 페이지 설정을 다 마치지 않은 프린트 작업(print job)을 생성할 수 있다. 일반적인 OS에서 제공하는 다이얼로그 박스를 사용함으로써 유저에게 친근감을 제공할 수 있다. 그림 1의 프린트 다이얼로그 박스를 살펴보자.

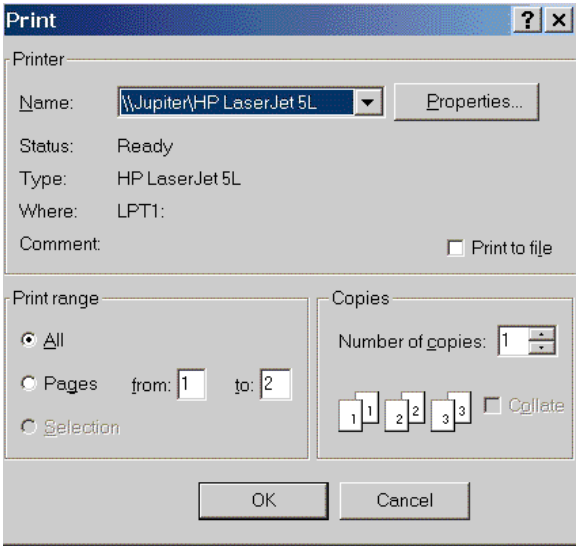


그림 1. 프린트 다이얼로그 박스

프린트 다이얼로그 박스를 사용하여 유저는 프린터를 선택할 수 있고 설정을 변경할 수 있다. 또한 유저는 프린트할 문서의 개수와 페이지 범위를 설정할 수도 있다. 이 다이얼로그 박스에서 유저는 프린트를 계속할 것인지 또는 그만둘 것인지 선택할 수 있다.

아래의 프린트 다이얼로그 박스를 출력하는 간단한 코드를 살펴보도록 하자.

```
if (printJob.printDialog()) {
  try {
    printJob.print();
  }
  catch (Exception PrintException) {
    PrintException.printStackTrace();
  }
}
```

PrinterJob 클래스의 printDialog 메소드는 프린트 다이얼로그 박스를 유저에게 보여주게 된다. 이 메소드는 Boolean 값을 리턴한다. 만약 유저가 Print 버튼을 클릭하게 되면 printDialog() 는 true를 리턴하고 프린팅 하는 반면 cancel을 클릭하면 false를 리턴하고 프린팅 작업은 더 이상 진행되지 않는다. printDialog() 메소드는 Boolean 값으로만 유저 어플리케이션과 상호작용(interact)하게 된다. 이 다이얼로그박스에서 설정한 어떠한 파라미터도 PrinterJob 객체에는 아무런 영향을 끼치지 않는다.

두 번째 다이얼로그 박스는 page 다이얼로그이다. 이 다이얼로그를 사용하면 유저는 페이지 설정을 변경할 수 있게 된다. 그림 2에서 페이지 다이얼로그 박스의 예를 볼 수있다.

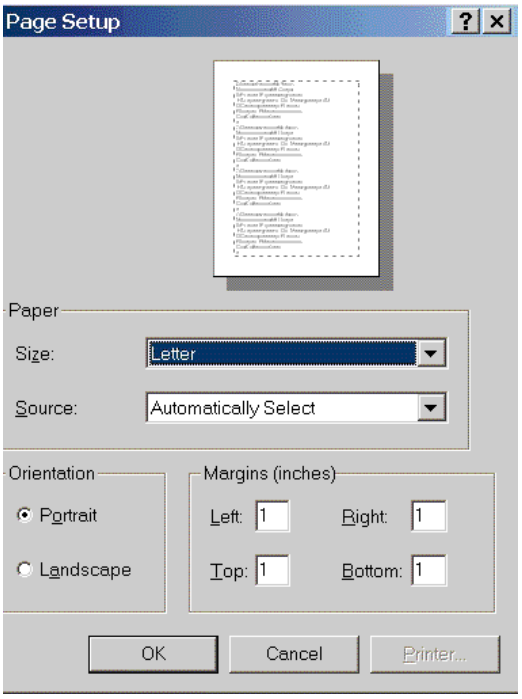


그림 2. 페이지 다이얼로그 박스

이 다이얼로그 박스를 사용하면 유저는 종이의 사이즈, 종이 종류, 페이지의 방향, 여백 등의 모든 PageForamt 객체의 파라메터를 설정할 수 있다.
아래의 코드에서 페이지 다이얼로그 박스를 어떻게 호출하는지 살펴볼 수 있을 것이다.

```
...

PageFormat documentPageFormat = new PageFormat ();
documentPageFormat = printJob.pageDialog (documentPageFormat);
book.append (new Document (), documentPageFormat);
...
```

만약 유저가 OK 버튼을 누르면 pageDialog() 메소드는 유저가 설정한 내용을 반영하여 PageFormat 객체의 복사본(clone)을 리턴 하게 된다. 유저가 Cancel 버튼을 선택하면 이 메소드는 PageFormat 객체를 변경하지 않고 바로 복사본을 리턴 한다.
이제 자바에서 프린팅 하는 방법을 알아보는 첫번째 단계를 끝냈다. 그렇다면 이제부터는 자바가 텍스트를 어떻게 다루는지 알아보도록 하자.

텍스트와 폰트를 다루자.

텍스트를 렌더링 하는 것은 프린팅에서 가장 복잡한 부분이라고 할 수 있다. 예전에 도트 매트릭스 프린터를 사용할 때는 단지 인치 당 몇 개의 문자가 들어가고 또 페이지당

몇 개의 문자가 들어 가는지 만 신경 쓰면 되었다. 행간을 정하는 것도 마찬가지로 간단하였고 세로로 텍스트를 정렬하기 위해서 단지 CR/LF 코드만을 프린터에 보내어 주면 다른 줄에서부터 출력되었다.

그러나 프린팅 방법은 오늘날 많이 바뀌었다. 텍스트의 2줄을 출력하기 위해서는 CR/LF 만이 아닌 더욱 많은 정보가 요구된다. 폰트를 어떻게 다루는지 까지도 알아야 한다.

폰트

만약 눈에 확 띄는 텍스트를 렌더링 하고 싶다면 먼저 폰트의 구조부터 이해해야만 한다. 그림 3은 폰트의 레이아웃을 보여주고 있다. 각 문자들은 baseline이라고 불리는 선을 기준으로 정렬되어 있다. 이 폰트가 위치할 때 세로의 참조점을 제공하는 역할을 한다. ascend는 baseline으로부터 가장 큰 문자의 위쪽 경계까지의 거리를 나타낸다. baseline과 폰트의 가장 낮은 경계가지의 공간은 descend라고 부른다. leading은 2개의 문자간의 세로 거리를 나타낸다. 그리고 폰트 height는 ascend에 leading과 descend를 더한 값을 가르킨다. 마지막으로 string의 길이는 advance라고 부른다. 이러한 모든 파라미터들은 폰트에 따라 다르다. 즉 예를 들어 Arial 폰트를 사용한 text의 값과 Lucida 폰트를 사용한 text값이 서로 다르다는 것이다.

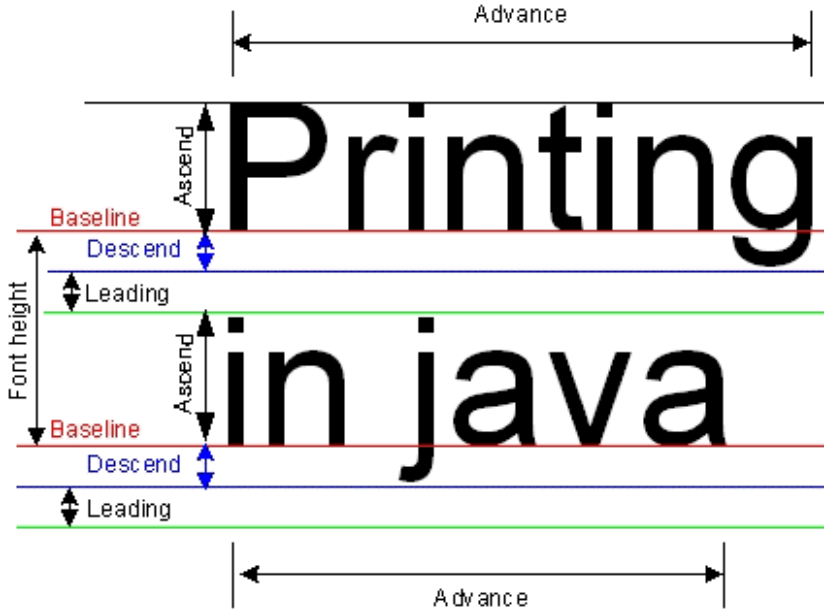


그림 3. 폰트의 정의

폰트를 위치시키는 작업은 일반적인 도형이나 그림들을 위치시키는 것과는 좀 다르다. 사각형 같은 경우 좌상측이 사각형의 기준점이 되나 폰트의 경우 baseline의 y좌표와 text의 시작 문자의 x 좌표가 폰트의 기준위치가 된다. 그림 4를 보면 폰트 기준좌표에 대해서 좀 더 자세히 알 수 있을 것이다.

그림 4. 폰트의 기준점

폰트를 사용할 때 2가지 스타일이 가능하다.

✍✍ Serif 폰트는 글자의 위쪽과 아래쪽의 꺾어 지는 부분에 약간의 튀어나온 부분이 있는 것이다. Times New Roman 폰트가 serif 폰트의 대표적인 예이다.

ABCD

✍✍ Sans serif 폰트는 장식용 요소를 가지고 있지 않다. Arial 폰트가 sans serif 폰트의 대표적인 예이다.

ABCD

이러한 폰트는 각각 둘 다 동일한 문자 넓이를 갖거나(monospaced) 문자넓이가 서로 다르게(proportional) 될 수 있다. monospaced 폰트는 모든 문자가 동일한 넓이를 가지고 있는 문자를 말한다. 도트 매트릭스 프린터 같은 구형 프린터에서는 monospaced 폰트를 사용하고 있다. proportional 폰트 내의 각 문자들은 각각 다른 넓이를 가질 수 있다. 레이저 프린터가 등장하고 나서부터 proportional 폰트가 등장하였고 이 폰트를 사용하는 모니터와 프린터에 사용한 첫번째 컴퓨터는 Apple의 Lisa와 맥킨토시 였다.

폰트에 접근하기

자바는 한번 작성하고 나면 어느 곳에서든 실행할 수 있도록(WORA) 디자인 되어 있기 때문에 자바를 만든 사람들은 모든 자바 지원 플랫폼에서 공통적으로 사용할 수 있는 폰트들(font family)을 제작해야만 했다. 또 더 중요한 점은 대응되는 운영체제가 해당하는 폰트들을 가지고 있건 아니건 간에 디폴트 폰트를 제공해야만 한다. 자바는 8개의 기본 폰트들을 제공하고 있다.: Serif, Sans Serif, Dialog, Dialog Input, Lucida Sans, Lucida Sans Typewriter, Lucida Bright와 Monospace.

WORA의 대세를 유지하기 위해서 자바는 논리적인 폰트 이름을 사용한다. 이러한 이름들은 자동적으로 운영 체제의 폰트와 맵핑되고 이는 JDK 디렉토리의 jre/lib/font.properties 파일에서 일어난다.. 이 파일을 살펴보면 특정 운영 체제의 폰트가 자바의 Serif 폰트와 어떻게 맵핑이 일어나는지 알 수 있을 것이다. 예를 들어 논리적인

폰트 이름은 자바에 의해 운영체제 폰트에 할당된 이름을 말한다. font.properties 파일을 살펴보면 논리적인 폰트 이름은 = 의 왼쪽에 나타나고 있음을 볼 수 있다. 예를 들어 Serif 는 내 컴퓨터의 Times New Roman에 맵핑된 논리적인 폰트 이름이다.

논리적인 폰트 이름을 제외하고도 유저는 폰트의 모양 이름과 폰트 이름을 이용하여 폰트에 접근할 수 있다. 폰트 이름은 Arial, Courier, Times Roman 또는 Lucida 같이 폰트를 만든 사람에 의해서 주어진 이름을 말한다. 폰트 패밀리가 Arial일 경우를 살펴보면 Arial Bold, Arial Italic 그리고 Arial Regular 가 Arial 폰트 패밀리의 멤버이다.

만약 어플리케이션이 좀 더 많은 폰트에 접근해야 한다면

```
java -Djava.awt.fonts=[fonts directory]
```

처럼 실행 시에 Type1(포스트 스크립트) 폰트 또는 트루 타입 폰트를 담고 있는 폰트 디렉토리를 전달하여 java.awt.fonts의 파라메터로 설정하면 된다. 이렇게 함으로써 폰트들을 운영체제에 추가하지 않고 어플리케이션의 폰트들을 확장하는 편리한 방법을 제공하고 있다. 만약 어플리케이션이 여러 플랫폼에서 동작하고 특정 폰트를 사용하려면 어플리케이션 jar파일에 폰트를 같이 넣으면 된다.

폰트관련 클래스들

자바 그래픽 API는 폰트를 조작하는 일을 간단하게 해주는 많은 클래스들을 제공한다. 그 중 가장 유용하게 쓰이는 클래스들의 정의를 아래에서 살펴볼 수 있다.

이름	타입	설명
Font	Class	폰트 클래스는 폰트의 모양을 대표하는 클래스이다. 최종적으로 작동될 시스템에서 사용 가능한 폰트들로 새로운 폰트를 생성하는데 이 클래스를 사용하라.
FontMetrics	Class (Abstract)	이 클래스는 폰트와 렌더링 디바이스에 대한 정보를 가지고 있다. FontMetrix는 많은 유틸리티들을 포함하고 있는데 예를 들어 스트링의 넓이를 구할 수도 있다. 이 클래스가 abstract인 것을 주의해라. 이 클래스의 인스턴스를 얻기 위해서는 Graphics.getFontMetrics를 호출해야 한다. 폰트의 descend, ascend, leading, advance 같은 사이즈 정보를 알고 싶을 때 FontMetric를 사용하라.
FontRenderContext	Class	이 클래스는 렌더링 디바이스와 관련된 폰트와 이러한 폰트를 렌더링 하는데 필요한 규칙(Rule)들에 대한 정보를 제공한다. 규칙(Rule)은 렌더링 품질을 정의하고 있다.

자바 그래픽 API에 포함된 클래스들

각각의 클래스에 대한 자세한 내용을 알고 싶다면 JDK 다큐먼트를 참조해보기 바란다. 이 클래스들을 어떻게 사용하는지 알려면 이 문서에 있는 예제 코드를 살펴보는 것도 좋을 것이다.

잘 정돈되지 않은 폰트들을 사용하여 텍스트로 구성된 문단을 렌더링 하는 것은 쉬운 일이 아니다. 폭이 다른(proportional) 폰트를 사용하고 있는 텍스트의 행간을 고르는 작업을 하는 알고리즘을 작성하는 것은 정말 고역이 아닐 수 없다. 또한 다른 나라의 문자에 대한 지원을 추가하면 더더욱 복잡하게 될 것이다.
이것이 바로 텍스트를 렌더링 할 때 TexLayout과 LineBreakMeasurer 클래스를 사용하는 이유이다.

TextLayout을 사용하여 편하게 프로그램 해보자.

TextLayout 클래스는 고급 텍스트 렌더링시 도움이 되는 많은 기능들을 제공 한다. 이 클래스는 좌에서 우로 글을 써나가는 영어와는 달리 우에서 좌로 글을 써나가는 일본어 텍스트를 렌더링할 수 있다. TextLayout 클래스는 이번 연재를 통해서는 살펴보는 것 외에도 여러 추가 기능들을 제공하고 있다. 텍스트 입력이나 커서 위치 또는 입력 테스팅 등의 기능은 문서를 출력하는 데는 필요치 않은 것들이다. 그러나 이러한 기능들이 존재하고 있다는 것을 알고 있는 것이 좋다.

TextLayout 클래스는 문단의 레이아웃을 잡는 클래스이다. 그러나 이 클래스 혼자서 작동되지 않는다. 특정 넓이로 텍스트를 정렬하기 위해서는 LineBreakMeasurer 클래스의 도움이 필요하다. LineBreakMeasurer 클래스는 이미 정의된 넓이로 맞추기 위해서 텍스트 스트링을 감싸는(wrap) 역할을 한다. 이 클래스는 여러 나라 말을 지원하도록 되어 있는 클래스 이기 때문에 각 나라 언어의 규칙에 따라 텍스트의 라인을 어디서 중단(break)해야 하는지 이미 알고 있다. LineBreakMeasurer 클래스 또한 혼자서 작동하는 클래스는 아니다. 이 클래스는 FontRenderContext에서 정보를 얻어와야 한다. 이 클래스 주요기능으로써 정확한 폰트 매트릭스를 리턴하는 역할을 담당한다. 텍스트를 효과적으로 측정하기 위해서 FontRenderContext는 사용되는 폰트 타입과 목적 디바이스(targeted device)에 대한 렌더링에 대한 약간의 정보를 알고 있어야 한다,.

TextLayout 어플리케이션의 첫번째 예제를 설명하기 전에 AttributedString이라는 새로운 클래스를 소개하겠다. 이 클래스는 스트링에 첨부된 어트리뷰트 정보를 원할 때 아주 유용하게 쓰일 수 있다. 예를 들어 다음과 같은 문장을 출력한다고 가정해 보자.

This is a **Bold** attribute.

만약 위의 문장을 AttributedString 클래스가 없이 프린트 한다면, Graphics.drawString() 메소드를 사용하여 다음과 같이 출력해야 될 것이다.

```
Font normalFont = new Font ("serif", Font.PLAIN, 12);
Font boldFont = new Font ("serif", Font.BOLD, 12);
g2.setFont (normalFont);
g2.drawString ("This is a ");
g2.setFont (boldFont);
g2.drawString ("bold ");
g2.setFont (normalFont);
g2.drawString ("attribute", 72, 72);
```

이 간단한 문장을 출력하기 위해서 얼마나 많은 코드를 작성했는지 보라. 만약 전체 문단을 렌더링 한다고 상상해 보라. 얼마나 끔찍한 작업이 되겠는가? 이 예제는 AttributedString 객체를 사용하여 좀 더 간단하게 변형할 수 있다. AttributedString은 String에 대한 Attribute를 할당할 수 있는 클래스이다. 이를 위해 attribute의 시작 위치와 종료 위치를 할당해 줘야 한다.

아래는 AttributedString을 이용하여 위의 예제를 간단히 한 것이다.

```
AttributedString attributedString = new AttributedString ("This is
a Bold attribute");
attributedString.addAttribute (TextAttribute.WEIGHT,
TextAttribute.WEIGHT_BOLD, 11, 14);
g2.drawString (attributeString.getIterator (), 72, 72);
```

위의 예제에서는 Bold라는 단어는 두꺼운 글씨체이다. Bold 단어에 이 글씨체를 적용하기 위해서 addAttribute() 메소드를 사용한다. 첫번째 파라메터는 설정하려는 어트리뷰트의 페밀리를 설정하는 어트리뷰트 키이다. 예를 들어 TextAttribute.WEIGHT 어트리뷰트 기는 weight 타입의 어트리뷰트 집합을 가르킨다.

두 번째 파라미터는 설정하려는 어트리뷰트이다. 만약 첫 번째 파라메터를 TextAttribute.WEIGHT로 설정했다면 두 번째 파라미터에 쓸 수 있는 어트리뷰트는 t WEIGHT_DEMIBOLD, WEIGHT_DEMILIGHT, WEIGHT_EXTRA_LIGHT, WEIGHT_EXTRABOLD, WEIGHT_HEAVY, WEIGHT_LIGHT, WEIGHT_MEDIUM, WEIGHT_REGULAR, WEIGHT_SEMIBOLD 과 WEIGHT_ULTRABOLD 이다.

세 번째 파라미터는 이 어트리뷰트를 적용하려는 시작 위치를 나타내고 네 번째 파라미터는 적용이 끝나는 위치를 나타낸다. 이러한 어트리뷰트들을 전체 스트링에 하려면 세 번째, 네 번째 파라미터를 제외한 동일 메소드를 호출하면 된다. 그림 5에서는 AttributedString상에서의 텍스트 구조 예제를 보여주고 있다.

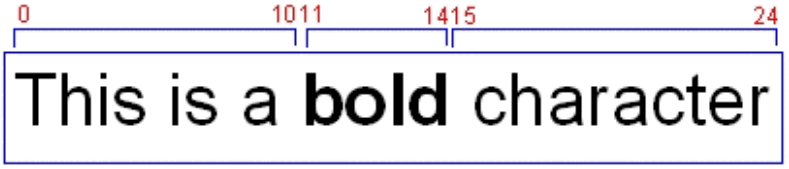


그림 5. AttributedString의 구조

TextLayout을 사용해 보자.

자 이제부터는 TextLayout 클래스를 사용할 때 필요한 여러 클래스들을 알아볼 차례이다. 첨부되는 소스 파일중의 Example4가 문단을 렌더링 하기 위해 TextLayout를 구현한 예이다. List 4는 일부만을 발췌한 것이다. 좀 더 자세한 내용을 보고 싶다면 소스코드를 살펴보기 바란다. 문단의 넓이는 7.5인치로 설정이 되어 있다. TextLayout 코드는 162 줄과 235 줄 사이에 나와있다.

Listing 4

```
118| /**
119| * Class: Document <p>
120| *
121| * 이 클래스는 문서(document) 내용을 위한 페인터이다. <p>
122| *
123| *
124| * @author Jean-Pierre Dube <jpdube@videotron.ca>
125| * @version 1.0
126| * @since 1.0
127| * @see Printable
128| */
129| private class Document implements Printable {
130|
131|
132| /**
133| * Method: print <p>
134| *
135| * @param g a value of type Graphics
136| * @param pageFormat a value of type PageFormat
137| * @param page a value of type int
138| * @return a value of type int
139| */
140| public int print (Graphics g, PageFormat pageFormat, int page) {
141|
142|
143| //--- Graphics2D 객체를 만든다.
144| Graphics2D g2d = (Graphics2D) g;
145|
146| //--- 좌상측으로 기준 좌표점을 옮긴다.
147| g2d.translate (pageFormat.getImageableX (),
pageFormat.getImageableY ());
148|
149| //--- 드로잉 색상을 검정색 으로 설정한다.
```

```
150| g2d.setPaint (Color.black);
151|
152| //--- 12포인트 두께로 경계선을 그린다.
153| g2d.setStroke (new BasicStroke (4));
154| Rectangle2D.Double border = new Rectangle2D.Double (0,
155| 0,
156| pageFormat.getImageableWidth (),
157| pageFormat.getImageableHeight ());
158|
159| g2d.draw (border);
160|
161|
162| //--- 스트링을 만들고 텍스트를 할당한다.
163| String text = new String ();
164| text += "Manipulating raw fonts would be too complicated to render
paragraphs of ";
165| text += "text. Trying to write an algorithm to fully justify text using ";
166| text += "proportional fonts is not trivial. Adding support for international
";
167| text += "characters adds to the complexity. That's why we will use the ";
168| text += "TextLayout and the LineBreakMeasurer class to ";
169| text += "render text. The TextLayout class offers a lot of ";
170| text += "functionality to render high quality text. This class is capable of ";
171| text += "rendering bidirectional text such as Japanese text where the
alignment ";
172| text += "is from right to left instead of the North American style which is
left ";
173| text += "to right. The TextLayout class offers some additional ";
174| text += "functionalities that we will not use in the course of this ";
175| text += "series. Features such as text input, caret positioning and hit ";
176| text += "testing will not be of much use when printing documents, but it's
good ";
177| text += "to know that this functionality exists. ";
178|
179| text += "The TextLayout class will be used to layout ";
180| text += "paragraphs. The TextLayout class does not work alone. To ";
181| text += "layout text within a specified width it needs the help of the ";
182| text += "LineBreakMeasurer class. This class will wrap a string of ";
183| text += "text to fit a predefined width. Since it's a multi-lingual class, it ";
184| text += "knows exactly where to break a line of text according to the
rules ";
185| text += "of the language. Then again the LineBreakMeasurer does ";
186| text += "not work alone. It needs information from the ";
```

```
187| text += "FontRenderContext class. This class' main function is to ";
188| text += "return accurate font metrics. To measure text effectively, this
class ";
189| text += "needs to know the rendering hints for the targeted device and
the font ";
190| text += "type being used. ";
191|
192|
193| //--- TextLayout 객체의 좌삭층 코너를 설정하기 위한 point 객체를 만든다.
194| Point2D.Double pen = new Point2D.Double (0.25 * POINTS_PER_INCH,
0.25 * POINTS_PER_INCH);
195|
196| //--- TextLayout 박스의 폭을 설정한다.
197| double width = 7.5 * POINTS_PER_INCH;
198|
199|
200| //--- 텍스트 스트링에서 attributed string을 만든다. LineBreakMesurer가
201| //--- 파라미터로 Iterator를 받기 때문에 attributed string을 만든다.
202| //
203| AttributedString paragraphText = new AttributedString (text);
204|
205| //--- 이 텍스트의 폰트를 설정한다.
206| paragraphText.addAttribute (TextAttribute.FONT, new Font ("serif",
Font.PLAIN, 12));
207|
208| //--- 텍스트를 감싸는 LineBreakMeasurer를 TextLayout을 위하여 만든다.
209| //--- 두번째 파라미터가 FontRendereContext인것을 주의하라.
210| //--- 안티알리아스가 가능하도록 2번째 파라미터를 true로 하였고 최상의 출력
211| //--- 품질을 얻기위하여 세번째 파라미터인 useFractionalMetrics를 true로 설
정하였다
212| LineBreakMeasurer lineBreaker = new LineBreakMeasurer
(paragraphText.getIterator(),
213| new FontRenderContext (null, true, true));
214|
215| //--- TextLayout 객체를 만든다.
216| TextLayout layout;
217|
218| //--- LineBreakMeasurer은 각 라인을 width 길이만큼 감쌀 것이고
219| //--- 이것을 TextLayout 객체에 리턴한다.
220| while ((layout = lineBreaker.nextLayout ((float) width)) != null) {
221|
222| //--- 펜의 Y좌표를 폰트의 ascend에 맞춘다. font에서는 ascending이 기준점
223| //--- (0, 0)임을 기억하기 바란다. 그림 1을 참조하라.
```

```
224| pen.y += layout.getAscent ();
225|
226| //--- 텍스트 한줄을 그린다.
227| layout.draw (g2d, (float) pen.x, (float) pen.y);
228|
229| //--- 펜의 y위치에 descent와 leading 더함으로써 다음 위치로 옮긴다.
230|
231| pen.y += layout.getDescent () + layout.getLeading ();
232| }
233|
234| //--- page를 Validate 한다.
235| return (PAGE_EXISTS);
236| }
237| }
238|
239|} // Example4
```

리스트 4를 요약해보자. 먼저 AttributedString을 만든다. addAttribute() 메소드는 폰트를 설정한다. 그러고 나서 AttributedString에서 가져온 iterator와 필요한 폰트 메트릭스를 공급 받기 위한 FontRenderContext 총 2개의 파라미터를 전달하여 LineBreakMeasurer 을 생성한다. 리턴값으로 LineBreakMeasurer은 각 라인을 감싸는 TextLaout 객체를 리턴한다. 루프상에서 텍스트의 각 줄의 위치를 계산한다. 폰트는 좌상측 코너가 기준위치가 아니라는 것을 꼭 명심해 두기 바란다. 그 대신에 y=0인 곳은 폰트의 baseline이다. 또한 정확히 한 줄 다음에 쓰고 싶다면 여기다가 descend와 leading을 함께 더해야 한다는 것도 꼭 기억해 두기 바란다. 아직도 잘 모르겠다면 그림 1을 다시 살펴보길권한다.

다음 예제는 양쪽 정렬을 구현한 예이다. 양쪽 정렬을 구현하기 위해서는 앞의 코드에서 몇 줄을 추가할 필요가 있다. Listing 5는 Example5의 소스에서 발췌한 부분이다.

Listing 5

```
120| /**
121| * Class: Document <p>
122| *
123| * 이 클래스는 문서(document) 내용을 위한 페인터이다.
124| * 이 프린트 메소드는 텍스트 문단을 양쪽 정렬로 렌더링 할 것이다.
125| * <p>
126| *
127| *
128| * @author Jean-Pierre Dube <jpdube@videotron.ca>
129| * @version 1.0
```

```
130| * @since 1.0
131| * @see Printable
132| */
133| private class Document implements Printable {
134|
135|
136| /**
137| * Method: print <p>
138| *
139| * @param g a value of type Graphics
140| * @param pageFormat a value of type PageFormat
141| * @param page a value of type int
142| * @return a value of type int
143| */
144| public int print (Graphics g, PageFormat pageFormat, int page) {
145|
146|
147| //--- Graphics2D객체를 만든다.
148| Graphics2D g2d = (Graphics2D) g;
149|
150| //--- 기준점(좌표점)을 좌상큭 코너 0,0으로 롬긴다.
151| g2d.translate (pageFormat.getImageableX (),
pageFormat.getImageableY ());
152|
153| //--- 드로잉 색상을 검은색으로 설정한다.
154| g2d.setPaint (Color.black);
155|
156| //--- 12포인트 두께의 경계선을 그린다.
157| g2d.setStroke (new BasicStroke (4));
158| Rectangle2D.Double border = new Rectangle2D.Double (0,
159| 0,
160| pageFormat.getImageableWidth (),
161| pageFormat.getImageableHeight ());
162|
163| g2d.draw (border);
164|
165|
166| //--- 스트링을 만들고 텍스트를 할당한다.
167| String text = new String ();
168| text += "Manipulating raw fonts would be too complicated to render
paragraphs of ";
169| text += "text. Trying to write an algorithm to fully justify text using ";
170| text += "proportional fonts is not trivial. Adding support for international
```

```
";
171| text += "characters adds to the complexity. That's why we will use the ";
172| text += "TextLayout and the LineBreakMeasurer class to ";
173| text += "render text. The TextLayout class offers a lot of ";
174| text += "functionality to render high quality text. This class is capable of ";
175| text += "rendering bidirectional text such as Japanese text where the
alignment ";
176| text += "is from right to left instead of the North American style which is
left ";
177| text += "to right. The TextLayout class offers some additional ";
178| text += "functionalities that we will not use in the course of this ";
179| text += "series. Features such as text input, caret positioning and hit ";
180| text += "testing will not be of much use when printing documents, but it's
good ";
181| text += "to know that this functionality exists. ";
182|
183| text += "The TextLayout class will be used to layout ";
184| text += "paragraphs. The TextLayout class does not work alone. To ";
185| text += "layout text within a specified width it needs the help of the ";
186| text += "LineBreakMeasurer class. This class will wrap a string of ";
187| text += "text to fit a predefined width. Since it's a multi-lingual class, it ";
188| text += "knows exactly where to break a line of text according to the
rules ";
189| text += "of the language. Then again the LineBreakMeasurer does ";
190| text += "not work alone. It needs information from the ";
191| text += "FontRenderContext class. This class' main function is to ";
192| text += "return accurate font metrics. To measure text effectively, this
class ";
193| text += "needs to know the rendering hints for the targeted device and
the font ";
194| text += "type being used. ";
195|
196|
197| //--- TextLayout 객체의 좌상단을 설정하기 위해 point 객체를 만든다.
198| Point2D.Double pen = new Point2D.Double (0.25 * POINTS_PER_INCH,
0.25 * POINTS_PER_INCH);
199|
200| //--- TextLayout 박스의 넓이를 설정한다.
201| double width = 8 * POINTS_PER_INCH;
202|
203|
204| //--- 텍스트 스트링으로부터 attributed 스트링을 만든다. LineBreakMeasurer
205| //--- 가 파라메터로 Iterator를 요구하기 때문에 attributed string을 만든다.
```

```
206| //---
207| AttributedString paragraphText = new AttributedString (text);
208|
209| //--- 이 텍스트의 폰트를 설정한다.
210| paragraphText.addAttribute (TextAttribute.FONT, new Font ("serif",
Font.PLAIN, 12));
211|
212| //--- TextLayout 객체를 위해 text를 감싸는 LineBreakMeasurer 를 만든다.
213| //--- 두번째 파라미터가 FontContext 인것에 주의하라. 여기서는 두번째
214| //--- 를 true로 하여 antiAlised가 가능하도록 하고 4번째 파라미터인
215| //--- useFractionalMetrics 를 최상의 출력결과를 얻기 위해 true로 설정한다.
216| LineBreakMeasurer lineBreaker = new LineBreakMeasurer
(paragraphText.getIterator(),
217| new FontRenderContext (null, true, true));
218|
219| //--- TextLayouts 객체를 만든다.
220| TextLayout layout;
221| TextLayout justifyLayout;
222|
223| //--- 각 텍스트 라인을 임시 저장할 Vector를 만든다.
224| Vector lines = new Vector ();
225|
226| //--- LineBreakMeasurer의 출력을 받아 Vector에 저장한다.
227| while ((layout = lineBreaker.nextLayout ((float) width)) != null) {
228| lines.add (layout);
229|
230| }
231|
232| //--- 각 문단의 라인을 스캔하여 맨 마지막 줄을 제외하고 양쪽 정렬을 한다.
233| for (int i = 0; i < lines.size (); i++) {
234|
235| //--- 벡터에서 한 줄을 꺼내온다.
236| layout = (TextLayout) lines.get (i);
237|
238| //--- 마지막 라인인지 체크한다. 마지막 라인이면 정렬을 하지 않고 출력한다.
239| //---
240| if (i != lines.size () - 1)
241| justifyLayout = layout.getJustifiedLayout ((float) width);
242| else
243| justifyLayout = layout;
244|
245| //--- 펜의 Y자료를 폰트의 ascend에 맞춘다. ascend가 폰트의 기준점(0,0)
246| //--- 인것을 기억하라. 그림 1을 참조하기 바란다.
```

```
247| pen.y += justifyLayout.getAscent ();
248|
249| //--- 텍스트 한줄을 그린다.
250| justifyLayout.draw (g2d, (float) pen.x, (float) pen.y);
251|
252| //--- 펜에 폰트의 descend와 leading을 더함으로써 다음 위치로 옮긴다.
253| //---
254| pen.y += justifyLayout.getDescent () + justifyLayout.getLeading ();
255|
256| }
257|
258| //--- 페이지를 validation한다.
259| return (PAGE_EXISTS);
260| }
261| }
262|
263|} // Example5
```

리스트 5의 가장 중요한 코드부분은 219줄부터 259줄까지 이다. TextLayout 클래스가 텍스트를 양쪽 정렬하는 메소드를 가지고 있다고 해도 LineBreakMeasurer 클래스는 문단의 맨 마지막 줄을 판별하는 기능은 포함하고 있지 않다. 여기서는 맨 마지막 줄을 제외한 모든 줄을 양쪽 정렬 해야 하기 때문에 직접 코드를 쳐넣어 마지막 줄일 경우 정렬을 하지 않고 출력되도록 해야 한다.

마지막 라인임을 체크하기 위해서는 먼저 모든 라인을 Vector에 집어넣고 for 루프를 돌면서 각 줄이 문단의 끝 줄인지를 직접 확인해 줘야 한다. 아니라면 정렬하기 위해서 getJustifiedLayout() 메소드를 호출해야 하고 아니라면 호출하지 않고 그냥 프린트한다.

이미지를 프린트 하기

이미지를 프린트하는 것은 이미지를 Image객체로 로딩하여 Graphics2D.drawImage() 메소드를 이용하여 출력하는 것과 똑같다. 다음의 예제는 나사의 우주 정거장 이미지를 어떻게 프린트 하는지 보여줄 것이다. 165번 줄과 198번 줄 사이가 실제로 이미지를 프린트 하는 코드이다.

Listing 6

```
120| /**
121| * Class: Document <p>
122| *
123| * 이 클래스는 다큐먼트 내용을 위한 페인터이다.
124| * 나사 우주 정거장을 경계선과 함께 프린트 한다..<p>
```

```
125| *
126| *
127| * @author Jean-Pierre Dube <jpdube@videotron.ca>
128| * @version 1.0
129| * @since 1.0
130| * @see Printable
131| */
132| private class Document extends Component implements Printable {
133|
134|
135| /**
136| * Method: print <p>
137| *
138| * @param g a value of type Graphics
139| * @param pageFormat a value of type PageFormat
140| * @param page a value of type int
141| * @return a value of type int
142| */
143| public int print (Graphics g, PageFormat pageFormat, int page) {
144|
145|
146| //--- Graphics2D 객체를 생성한다.
147| Graphics2D g2d = (Graphics2D) g;
148|
149| //--- 기준점(좌표점)을 (0,0)으로 옮긴다.
150| g2d.translate (pageFormat.getImageableX (),
pageFormat.getImageableY ());
151|
152| //--- 드로잉 색상을 검은색으로 설정한다.
153| g2d.setPaint (Color.black);
154|
155| //--- 12 포인트 두께의 경계선을 그린다.
156| g2d.setStroke (new BasicStroke (4));
157| Rectangle2D.Double border = new Rectangle2D.Double (0,
158| 0,
159| pageFormat.getImageableWidth (),
160| pageFormat.getImageableHeight ());
161|
162| g2d.draw (border);
163|
164|
165| //--- Media Tracker와 URL 객체를 만든다.
166| MediaTracker mt = new MediaTracker (this);
```

```
167| URL imageURL = null;
168|
169| //--- URL을 로드하기 원하는 이미지가 있는 위치로 설정한다.
170| //--- NOTE: 이미지의 위치로 패스를 변경하라.
171| //--- NOTE: 다음의 이미지는 JPEG, GIF, PNG 타입만이 가능하다.
173|
174| imageURL = new URL
("file:///c:/softdev/java/articles/javaworld/printing/part_2/ss2.jpg");
175| }
176| catch (MalformedURLException me) {
177| me.printStackTrace ();
178| }
179|
180| //--- 이미지를 로드하고 이미지가 완전히 로딩될때까지 기다린다.
181| Image image = Toolkit.getDefaultToolkit().getImage (imageURL);
182| mt.addImage (image, 0);
183| try {
184| mt.waitForID (0);
185| }
186| catch (InterruptedException e) {
187| }
188|
189| //--- 이미지를 종이에다 출력한다.
190| g2d.drawImage (image,
191| (int) (0.25 * POINTS_PER_INCH),
192| (int) (0.25 * POINTS_PER_INCH),
193| (int) (8.5 * POINTS_PER_INCH),
194| (int) (6 * POINTS_PER_INCH),
195| this);
196|
197| //--- 페이지를 Validate한다.
198| return (PAGE_EXISTS);
199| }
200| }
201|
202|} // Example6
```

이미지를 프린팅 하는 작업은 3가지 간단한 단계로 나누어 진다.

- ✍✍ 프린트하기 원하는 이미지를 가르키는 URL 작성한다.
- ✍✍ 이미지를 MediaTracker 클래스를 이용하여 로딩한다. 이 클래스를 이용하여 GIF, JPEG와 PNG 파일을 로딩할 수 있다. 만약 다른 이미지 타입을 로딩하기 원한다면 Advance Imaging API를 사용하기 바란다.
- ✍✍ Graphics2D 클래스의 drawImage() 메소드를 이용하여 이미지를 출력한다.

drawImage() 메소드는 6개의 파라미터를 가지고 있다. 첫번째는 출력될 Image 객체이고 두 번째와 세 번째는 이미지의 좌상측 코너가 위치할 좌표이다. 네 번째와 다섯 번째 파라미터는 이미지의 가로 크기와 세로 크기를 설정한다. 마지막 파라미터는 ImageObserver를 위한 것이다. 이 예제의 목적을 위하여 Document 클래스는 Component 클래스를 상속했다. 모든 Component 클래스는 ImageObserver 인터페이스를 구현하고 있다.

위의 3개의 단계를 따라 하면 쉽게 종이 위에 이미지를 렌더링 할 수 있을 것이다. 만약 좀 더 많은 이미지 포맷을 사용하길 원한다면 JavaSoft 웹 사이트에서 Advanced Imaging API를 다운 받을 수 있을 것이다.

다양한 플랫폼에서의 프린팅

자바가 WORA 언어로써의 자격을 갖추고 있을 지라도 일부는 OS가 제공하는 서비스에 의존해야만 한다. 다양한 플랫폼에서 이러한 이슈들을 확인하기 위해서 필자는 SuSe Linux 6.4에 Sun Java 1.3을 설치하였다. Windows 2000과의 비교를 위해서 같은 HP 5L 프린터를 각각의 OS가 설치된 컴퓨터에 연결하였다. 나의 주요 관심은 출력 질이 아니라 표준 다이얼로그로써 프린트, 페이지 설정 다이얼로그를 제공하고 있는 Windows와 그러한 표준 다이얼로그를 제공하고 있지 않은 Linux상에서 자바가 이러한 차이점을 어떻게 처리하느냐 였다.

처음 부딪힌 문제는 프린터 설정이었다. SuSe 리눅스는 APS 프린트 필터를 사용하고 있고 HP 5L 프린터를 600 dpi로 설정할 수 없었다. 단지 300 dpi로만 출력할 수 있었고 600 dpi로 출력하려면 좀 더 좋은 드라이버가 설치되어 있어야만 했다.

윈도우에서 작성한 코드를 아무런 수정 없이 Linux에서 수행하였다. 그리고 해상도의 차이에도 불구하여 훌륭한 품질로 프린트할 수 있었고 뛰어난 성능을 발휘하였다. 그러나 화면에 디스플레이 된 프린트, 페이지 다이얼로그 박스는 필자를 놀라게 만들었다. 윈도우와는 완전히 다른 모습을 띄고 있었다. 그림 6과 7에서 이 다이얼로그를 볼 수 있다.

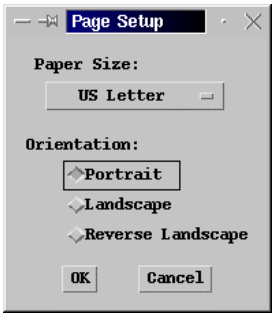


그림 6. 리눅스에서의 Page 다이얼로그 박스

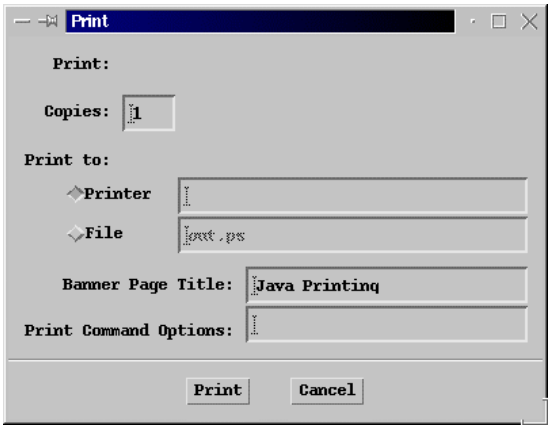


그림 7. 리눅스에서의 프린트 다이얼로그 박스

이러한 다이얼로그 박스에서의 놀라운 차이는 어플리케이션에서 몇 가지 기능을 제한하는 결과를 낳을 수 있다. 예를 들어 리눅스 상에서는 프린트 다이얼로그를 통해 출력한 페이지 범위를 지정할 수 없었다. 윈도우에서는 가능한 것이다. 또한 여백에 대하여 지원하지 않는다. 만약 다 기종에 이식할 어플리케이션을 작성한다면 이러한 차이점에 대하여 처리해야 하고 또 모든 플레폼에서 테스트 해봐야 한다. 필자는 아직 OS X가 깔려있는 Max을 사용해보지 못했기 때문에 이 플렛폼에서는 예제를 제공할 수 없었다. 곶이 새로운 OS에 한번 시도를 해볼 것이고 다음 강좌에서 결과를 알려드리겠다.

결론

파트 2는 자바 프린트 API라는 유혹 속으로 당신을 안내하였다. (예를 들어 단지 그림을 그릴 수 있는 그래픽 캔버스만 제공 했다고나 할까?) 이 API는 문단이나 이미지, 머리말, 꼬리말, 표 같은 개념이 담겨져 있지 않기 때문에 이런 기능을 추가하기 위해서는 기존적인 API들을 조합하여 구현하여야 한다.

또 이 글에서는 다양한 플랫폼에서의 프린팅이라는 이슈에 대해 건드려 봤고 리눅스하에서의 프린팅과 윈도우 하에서의 프린팅의 차이점도 살펴보았다. 파트 3에서는 앞으로 남은 3개의 연재를 통해 계속 구현해 나갈 프린트 프레임웍의 설계에 대해서 설명할 것이다. 그때까지 행복한 프린팅이 되시기를 ~