



java 20강 - 입출력(I/O)

양 명 속

[now4ever7@gmail.com]



목차

- 자바에서의 입출력
- 바이트 기반 스트림
- 바이트 기반의 보조 스트림
- 문자 기반 스트림
- 문자 기반의 보조 스트림
- 표준 입출력과 File
- 직렬화(Serialization)



자바에서의 입출력(I/O)

■ 입출력이란

- I/O란 Input과 Output의 약자로 입력과 출력(입출력)
- 입출력은 컴퓨터 내부 또는 외부의 장치와 프로그램간의 데이터를 주고 받는 것
- 예) 키보드로부터 데이터를 입력 받는다거나 System.out.println()을 이용해서 화면(모니터)에 출력한다거나 하는 것

■ 스트림(stream)

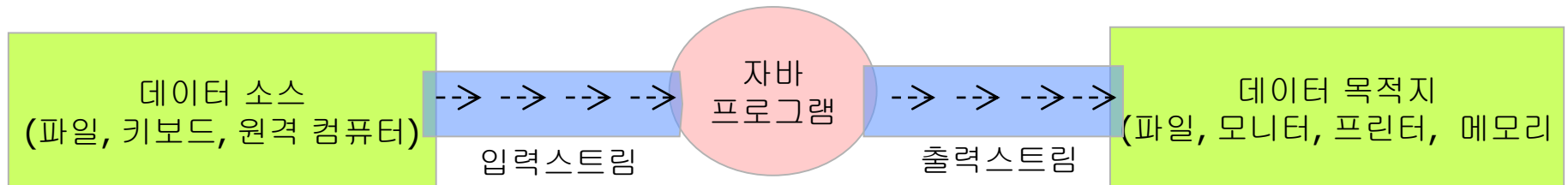
- 자바에서 입출력을 수행하려면, 즉 어느 한쪽에서 다른 쪽으로 데이터를 전달하려면, 두 대상을 연결하고 데이터를 전송할 수 있는 무언가가 필요한데 이것을 스트림이라고 함

스트림이란 데이터를 운반하는 데 사용되는 연결 통로이다.

자바에서의 입출력

■ 스트림

- 단방향 통신만 가능
 - =>하나의 스트림으로 입력과 출력을 동시에 처리할 수 없음
 - 입력과 출력을 동시에 수행하려면 **입력을 위한 입력 스트림**과 **출력을 위한 출력 스트림**, 모두 2개의 스트림이 필요함
- 먼저 보낸 데이터를 먼저 받게 되어 있으며, 중간에 건너뛸 없이 연속적으로 데이터를 주고 받음





Stream의 구분

- 1. 스트림에서 다루는 데이터가 무엇이냐에 따른 구분
 - 1) **byte 기반 스트림** (1byte 단위로 데이터를 이동시킴)
 - ~InputStream/~OutputStream
 - 2) **문자(char) 기반 스트림** (2byte 단위(문자)로 데이터가 이동)
 - ~Reader/~Writer
- 2. 스트림 안에 들어온 데이터를 그대로 보내느냐 아니면 가공 절차를 거친 뒤 보내느냐에 따른 구분
 - **Node 스트림** : 데이터 소스와 직접 연결 가능한 스트림
 - **Filter 스트림(보조 스트림)** : 데이터 소스에 직접 연결은 불가능하고, 데이터 소스에 직접 연결한 노드 스트림을 **가공하는 역할**을 하는 스트림
 - **스트림의 기능을 보완**하기 위한 보조 스트림
 - => 노드 스트림과 연결해서 써야 함.

자바에서의 입출력

■ 바이트 기반 스트림 – InputStream, OutputStream

- 스트림은 바이트 단위로 데이터를 전송하며 입출력 대상에 따라 다음과 같은 입출력 스트림이 있음

입력스트림	출력스트림	입출력 대상의 종류
FileInputStream	FileOutputStream	파일
ByteArrayInputStream	ByteArrayOutputStream	메모리(Byte배열)
PipelineInputStream	PipelineOutputStream	프로세스(프로세스 간의 통신)
AudioInputStream	AudioOutputStream	오디오 장치

- 모두 InputStream 또는 OutputStream의 자손들
- 읽고 쓰는데 필요한 추상 메서드를 자신에 맞게 구현해 놓았음
- 예) 파일의 내용을 읽고자 하는 경우 FileInputStream 사용
- **java.io 패키지**를 통해서 많은 종류의 입출력 관련 클래스들을 제공, **입출력을 처리할 수 있는 표준화된 방법 제공** – 입출력의 대상이 달라져도 동일한 방법으로 입출력이 가능

예제

public int read() throws IOException
- Reads a byte of data from this input stream.
- inputStream에서 1바이트씩 읽어온다
- 아스키코드를 리턴함
- 더 이상 읽어올 것이 없으면 -1을 리턴함

```
import java.io.*;
```

```
class FileViewer {  
    public static void main(String args[]) throws IOException{  
        FileInputStream fis = new FileInputStream("text/poetry2.txt")  
        int data = 0;  
        while((data=fis.read())!= -1) {  
            char c = (char)data;  
            System.out.print(c);  
        }  
        fis.close();  
    }  
}
```

해당 file 의 byte 수만큼 while문을 반복함

public void print(char c)
- **Prints a character.** The character is translated into one or more bytes according to the platform's default character encoding
- **char 단위로 출력**

- read() : 한 번에 1byte씩 파일로부터 데이터를 읽어 들임
- read()의 반환값이 int형 이긴 하지만, 더 이상 입력 값이 없음을 알리는 -1을 제외하고는 0~255 범위(1byte)의 정수값이기 때문에, char형으로 변환가능

자바에서의 입출력

```
//기본 스트림을 생성한다
FileInputStream fis = new FileInputStream("test.txt");
fis.read(); //기본스트림으로 부터 데이터를 읽는다
```

■ 보조 스트림

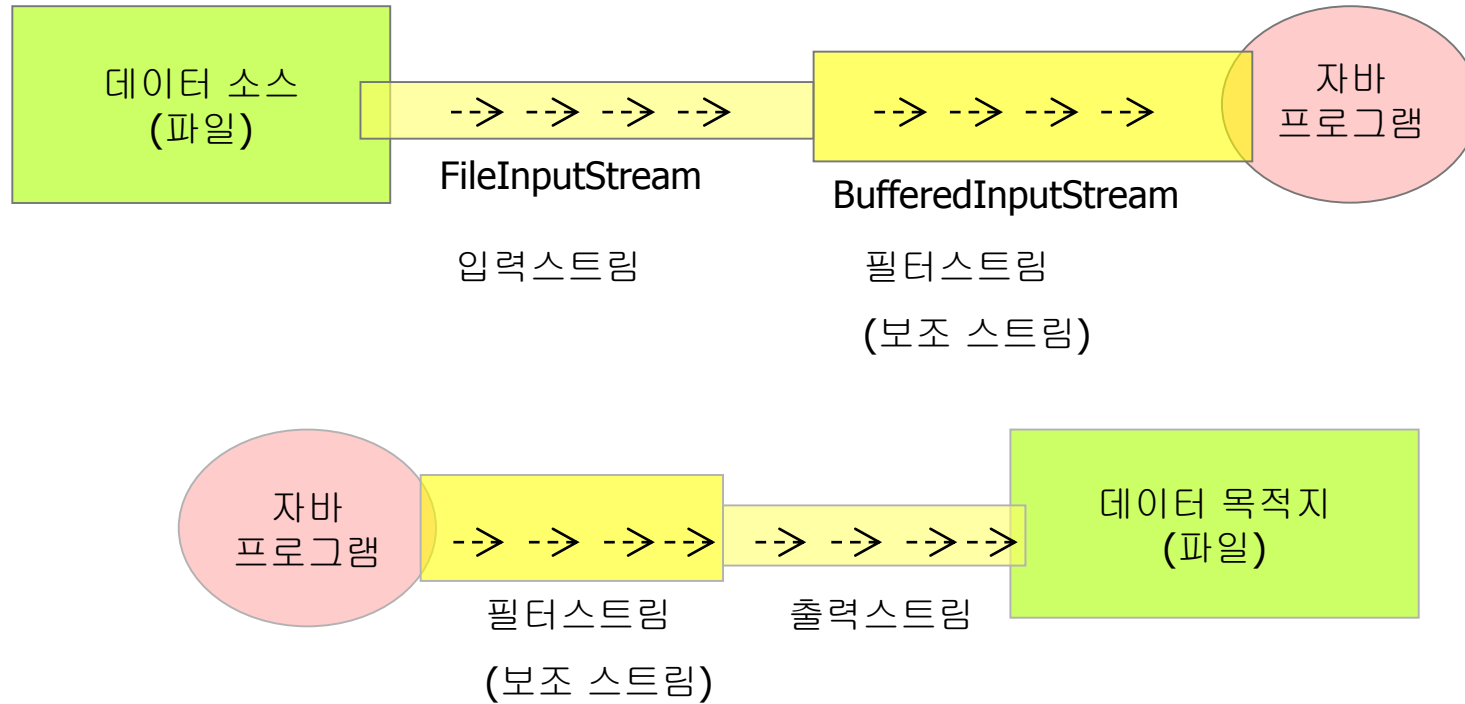
- **스트림의 기능을 보완**하기 위한 보조 스트림 제공
- 보조 스트림은 실제 데이터를 주고 받는 스트림이 아니기 때문에 데이터를 입출력할 수 있는 기능은 없지만, **스트림의 기능을 향상**시키거나 새로운 기능을 추가할 수 있음
- 보조 스트림만으로는 입출력을 처리할 수 없고, **스트림을 먼저 생성한 다음에** 이를 이용해서 보조스트림을 생성해야 함
- 예) test.txt 파일을 읽기 위해 FileInputStream 을 사용할 때, 입력 성능을 향상시키기 위해 버퍼를 사용하는 보조 스트림인 BufferedInputStream을 사용하는 코드

```
//먼저 기본 스트림을 생성한다
FileInputStream fis = new FileInputStream("test.txt");
//기본 스트림을 이용해서 보조 스트림을 생성한다
BufferedInputStream bis = new BufferedInputStream(fis);
```

```
bis.read(); //보조 스트림인 BufferedInputStream으로 부터 데이터를 읽는다
```

실제 입력기능은 BufferedInputStream과 연결된 FileInputStream이 수행, 보조 스트림인 BufferedInputStream은 버퍼만을 제공

보조 스트림





자바에서의 입출력

■ 보조 스트림의 종류

입력	출력	설명
FilterInputStream	FilterOutputStream	필터를 이용한 입출력 처리
BufferedInputStream	BufferedOutputStream	버퍼를 이용한 입출력 성능 향상
DataInputStream	DataOutputStream	int, float와 같은 기본형 단위로 데이터를 처리하는기능
SequenceInputStream	SequenceOutputStream	두 개의 스트림을 하나로 연결
LineNumberInputStream	없음	읽어 온 데이터의 라인 번호를 카운트
ObjectInputStream	ObjectOutputStream	데이터를 객체 단위로 읽고 쓰는데 사용. 주로 파일을 이용하며 객체 직렬화와 관련 있음
없음	PrintStream	버퍼를 이용하며, 추가적인 print 관련 기능
PushbackInputStream	없음	버퍼를 이용해서 읽어온 데이터를 다시 되돌리는 기능



자바에서의 입출력

- 문자 기반 스트림 - Reader, Writer
 - 바이트 기반 스트림 - 입출력의 단위가 1byte
 - java에서는 한문자를 의미하는 char 형이 2byte이기 때문에 바이트 기반의 스트림으로는 2byte인 문자를 처리하는 데 어려움이 있음
 - => 문자 기반의 스트림이 제공됨
 - 문자 데이터를 입출력할 때는 바이트 기반 스트림 대신 문자 기반 스트림을 사용

InputStream => Reader
OutputStream => Writer



자바에서의 입출력

■ 바이트 기반 스트림과 문자 기반 스트림의 비교

바이트 기반 스트림	문자기반 스트림
FileInputStream FileOutputStream	FileReader FileWriter
ByteArrayInputStream ByteArrayOutputStream	CharArrayReader CharArrayWriter
PipedInputStream PipedOutputStream	PipedReader PipedWriter
StringBufferInputStream(deprecated) StringBufferOutputStream(deprecated)	StringReader StringWriter



자바에서의 입출력

- 바이트 기반 보조스트림과 문자 기반 보조스트림

바이트 기반 보조스트림	문자기반 보조스트림
BufferedInputStream BufferedOutputStream	BufferedReader BufferedWriter
FilterInputStream FilterOutputStream	FilterReader FilterWriter
LineNumberInputStream(deprecated)	LineNumberReader
PrintStream	PrintWriter
PushbackInputStream	PushbackReader

스트림 구분

	1바이트 기반 스트림(1byte)		문자기반 스트림(2byte)	
Abstract class	InputStream	OutputStream	Reader	Writer
Node Stream	FileInputStream System.in	FileOutputStream System.out	FileReader PipeReader StringReader	FileWriter PipeWriter StringWriter
		다리(bridge)역할을 수행하는 스트림	InputStreamReader	OutputStreamWriter
Filter Stream (보조 스트림)	DataInputStream BufferedInputStream CheckedInputStream DigestInputStream InflaterInputStream LineNumberInputStream PushbackInputStream ProgressMonitorInputStream	DataOutputStream BufferedOutputStream	BufferedReader PrintReader	BufferedWriter PrintWriter



바이트 기반 스트림



바이트 기반 스트림

- InputStream과 OutputStream
 - InputStream과 OutputStream은 모든 바이트 기반 스트림의 조상

InputStream 메서드

메서드명	설명
int available()	스트림으로부터 읽어 올 수 있는 데이터의 크기를 반환한다
void close()	스트림을 닫음으로써 사용하고 있던 자원을 반환한다
void mark(int readlimit)	현재위치를 표시해 놓는다. 후에 reset()에 의해서 표시해 놓은 위치로 다시 돌아갈 수 있다. readlimit은 되돌아갈 수 있는 byte의 수이다
boolean markSupported()	mark()와 reset()을 지원하는지를 알려준다. mark()와 reset()을 사용하기 전에 markSupported()를 호출해서 지원여부를 확인해야 함
abstract int read()	1byte를 읽어옴(0~255사이의 값). 더 이상 읽어 올 데이터가 없으면 -1을 반환함.
int read(byte[] b)	배열 b의 크기만큼 읽어서 배열을 채우고 읽어 온 데이터의 수를 반환함
int read(byte[] b, int off, int len)	최대 len개의 byte를 읽어서, 배열 b의 지정된 위치 (off) 부터 저장한다. 실제로 읽어 올 수 있는 데이터가 len개보다 적을 수 있다
void reset()	스트림에서의 위치를 마지막으로 mark()이 호출되었던 위치로 되돌린다
long skip(long n)	스트림에서 주어진 길이(n)만큼 건너뛰다.



OutputStream 메서드

메서드명	설명
<code>void close()</code>	입력 소스를 닫음으로써 사용하고 있던 자원을 반환한다
<code>void flush()</code>	스트림의 버퍼에 있는 모든 내용을 출력 소스에 쓴다
<code>abstract void write(int b)</code>	주어진 값을 출력소스에 쓴다.
<code>void write(byte[] b)</code>	주어진 배열 b에 저장된 모든 내용을 출력소스에 쓴다.
<code>void write(byte[] b, int off, int len)</code>	주어진 배열 b에 저장된 내용 중에서 off번째부터 len개 만큼만을 읽어서 출력소스에 쓴다.



InputStream과 OutputStream

- InputStream과 OutputStream
 - flush() : 버퍼가 있는 출력 스트림의 경우에만 의미가 있으며, OutputStream에 정의된 flush()는 아무런 일도 하지 않음
 - 프로그램이 종료될 때, 사용하고 닫지 않은 스트림을 JVM이 자동적으로 닫아 주기는 하지만, 스트림을 사용해서 모든 작업을 마치고 난 후에는 close()를 호출해서 반드시 닫아 주어야 함



FileInputStream과 FileOutputStream

- FileInputStream과 FileOutputStream

- 파일에 입출력을 하기 위한 스트림

- 생성자

- FileInputStream(File file)
- FileInputStream(String name)
- FileOutputStream(File file)
- `FileOutputStream(File file, boolean append)`
- FileOutputStream(String name)
- FileOutputStream(String name, boolean append)

true => append 기능

예제

public int read() throws IOException
- Reads a byte of data from this input stream.
- inputStream에서 1바이트씩 읽어온다
- 아스키코드를 리턴함
- 더 이상 읽어올 것이 없으면 -1을 리턴함

```
import java.io.*;
```

```
class FileViewer {  
    public static void main(String args[]) throws IOException{  
        FileInputStream fis = new FileInputStream("text/poetry2.txt")  
        int data = 0;  
        while((data=fis.read())!= -1) {  
            char c = (char)data;  
            System.out.print(c);  
        }  
        fis.close();  
    }  
}
```

해당 file 의 byte 수만큼 while문을 반복함

public void print(char c)
- Prints a character. The character is translated into one or more bytes according to the platform's default character encoding

- **char** 단위로 출력

- **read()** : 한 번에 1byte씩 파일로부터 데이터를 읽어 들임
- **read()**의 반환값이 int형 이긴 하지만, 더 이상 입력 값이 없음을 알리는 -1을 제외하고는 0~255 범위(1byte)의 정수값이기 때문에, char형으로 변환가능

```

import java.io.*;

class FileViewer {
    public static void main(String[] args) {
        FileInputStream fis =null;
        try {
            fis= new FileInputStream("text/poetry2.txt");
            int data=0;
            int count=0;
            while((data=fis.read())!=-1){
                char ch = (char)data;
                System.out.print(ch);
                count++;
            }
            System.out.println("반복횟수:"+count);
            File myfile = new File("text/poetry2.txt");
            System.out.println("파일의 byte수:"+myfile.length());
        } catch (FileNotFoundException e) {e.printStackTrace();}
        } catch (IOException e) { e.printStackTrace();}
    }finally{
        try {
            if(fis !=null) fis.close();
        } catch (IOException e) {e.printStackTrace();}
    }
}

}

```

O Me! O Life! - by Walt Whitman

O me. O life. Of the questions of these recurring,
Of the endless trains of the faithless,
Of cities filled with the foolish.
What good amid these o me, o life?
The answer, that you are here.
That life exists and identity.
That the powerful play goes on.
and you may contribute a verse.

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 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 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160 1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1360 1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420 1421 1422 1423 1424 1425 1426 1427 1428 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1540 1541 1542 1543 1544 1545 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1599 1600 1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655 1656 1657 1658 1659 1660 1661 1662 1663 1664 1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1720 1721 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1860 1861 1862 1863 1864 1865 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040 2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2160 2161 2162 2163 2164 2165 2166 2167 2168 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220 2221 2222 2223 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336 2337 2338 2339 2340 2341 2342 2343 2344 2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400 2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460 2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518 2519 2520 2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560 2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573 2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586 2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599 2600 2601 2602 2603 2604 2605 2606 2607 2608 2609 2610 2611 2612

예제2

- `FileInputStream` 과 `FileOutputStream` 을 사용해서 `poetry2.txt` 파일의 내용을 그대로 `poetry2.bak`으로 복사하는 예제
- 단순히 `poetry2.txt`의 내용을 `read()`로 읽어서 `write(int b)`로 `poetry2.bak`에 출력한다
=> 텍스트 파일을 다루는 경우에는 문자기반 스트림인 `FileReader/FileWriter` 를 사용하는 것이 더 좋다

```
import java.io.*;
```

```
class FileCopy {
    public static void main(String[] args) {
        //poetry2.txt 파일을 바이트 단위로 읽어서 poetry2.bak 파일로 출력하기(바이트 단위로 출력)
        FileInputStream fis=null;
        FileOutputStream fos=null;
        try {
            fis= new FileInputStream("text/poetry2.txt");
            fos=new FileOutputStream("text/poetry2.bak",true); //append
            int data=0;
            while((data=fis.read())!=-1){ //1바이트씩 읽기
                //1바이트씩 출력하기
                fos.write(data); // void write(int b)
            }
        } catch (FileNotFoundException e) {e.printStackTrace();}
        } catch (IOException e) { e.printStackTrace();}
        }finally{
            try {
                if(fis !=null) fis.close();
                if(fos!=null)fos.close();
            } catch (IOException e) { e.printStackTrace();}
        }
        System.out.println("파일 카피 성공!!");
    } //main
}
```

```
public void write(int b)
-Writes the specified byte to this stream.
- byte 단위로 출력
```

```
public FileOutputStream(String name, boolean append)
- append 변수에 true 를 지정하면 파일의 내용이 추가
(append)됨
```



예제3

public int read(byte[] b)throws IOException

-InputStream에서 읽어서 byte배열에 넣는다(byte배열의 길이만큼)

-읽어온 개수를 리턴함

public void write(byte[] b,int off,int len)throws IOException

- byte배열에서 시작위치 off에서 len개 만큼 OutputStream에 출력

```
import java.io.*;
```

```
class FileInStreamTest{
```

```
    public static void main(String[] args) throws IOException{
```

```
        String filename="text/poetry4.txt";
```

```
        FileInputStream fis=new FileInputStream(filename);
```

```
        FileOutputStream fos=new FileOutputStream("text/poetry4.bak");
```

```
        int cnt=0, count=0, total=0;
```

```
        byte buf[]=new byte[1024];
```

```
        while ((cnt=fis.read(buf))!=-1)
```

```
        {
```

```
            System.out.write(buf, 0, cnt); //buf 배열에서 0부터 cnt개 읽어서 출력소스에 쓴다
            fos.write(buf, 0, cnt);
```

```
            System.out.println("\n----cnt: " +cnt +"\n");
```

```
            total+=cnt; //1024*1009
```

```
            count++;
```

```
        }//while
```

```
        System.out.println("\n\n총 " + total + "바이트"); //2033
```

```
        System.out.println("\n반복횟수 : " + count); //2
```

```
        fis.close();
```

```
        fos.close();
```

```
    }
```

```
}
```

```
System.out.write(buf);
fos.write(buf);
```


And roads diverged in a wood, and I
I took the one less traveled by,
And that has made all the difference.??

O Captain! My Captain! - by Walt Whitman

I.
O CAPTAIN! my Captain! our fearful trip is done;
The ship has weather'd every rack, the prize we sought is won;
The port is near, the bells I hear, the people
----cnt: 1024

all exulting,
While follow eyes the steady keel,
But O heart! heart! heart!

You've fallen cold and dead.

III.
My Captain does not answer, his lips are pale and still;
My father does not feel my arm, he has no pulse nor will;
The ship is anchor'd safe and sound, its voyage closed and done;
From fearful trip, the victor ship, comes in with object won;
Exult, O shores, and ring, O bells!
But I, with mounful tread,
Walk the deck my Captain lies,

Fallen cold and dead.
----cnt: 1009

총 2033바이트

반복횟수 : 2

```
import java.awt.Button;
import java.awt.Frame;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;
public class TextTest extends Frame {
    String fileName;
    TextArea content;
    Button btn;
    TextTest(String title) {
        super(title);
        content = new TextArea();
        btn = new Button("닫기");
        add(content, "Center");
        add(btn, "South");
        btn.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
    }
}
```

```

public void fileOpen(String fileName) {
    FileReader fr=null;
    BufferedReader br=null;
    try {
        fr = new FileReader(fileName);
        br= new BufferedReader(fr);

        String data = "", str="";
        while ((data=br.readLine())!=null) {
            str+=data+"\r\n";
        }
        content.setText(str);    //한글이 깨지지 않는다
    } catch(IOException e) {
        e.printStackTrace();
    }finally{
        try {
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public void fileOpen2(String fileName) {
    FileInputStream fis=null;
    BufferedInputStream bis=null;

```

```

try {
    fis = new FileInputStream(fileName);
    bis= new BufferedInputStream(fis);

    int data = 0;
    String str="";
    while ((data=bis.read())!= -1) {
        str+=Character.toString((char)data);
    }
    content.setText(str);  //한글이 깨진다
} catch(IOException e) {
    e.printStackTrace();
}finally{
    try {        bis.close();
    } catch (IOException e) { e.printStackTrace();        }
}
}

```

```

public static void main(String args[]) {
    TextTest mainWin = new TextTest("Stream 테스트");
    mainWin.setSize(300, 200);
    mainWin.setVisible(true);
    mainWin.fileOpen2("text/poetry2.txt");

```

```

}

```

```

} //class

```



바이트 기반의 보조 스트림



바이트 기반의 보조 스트림

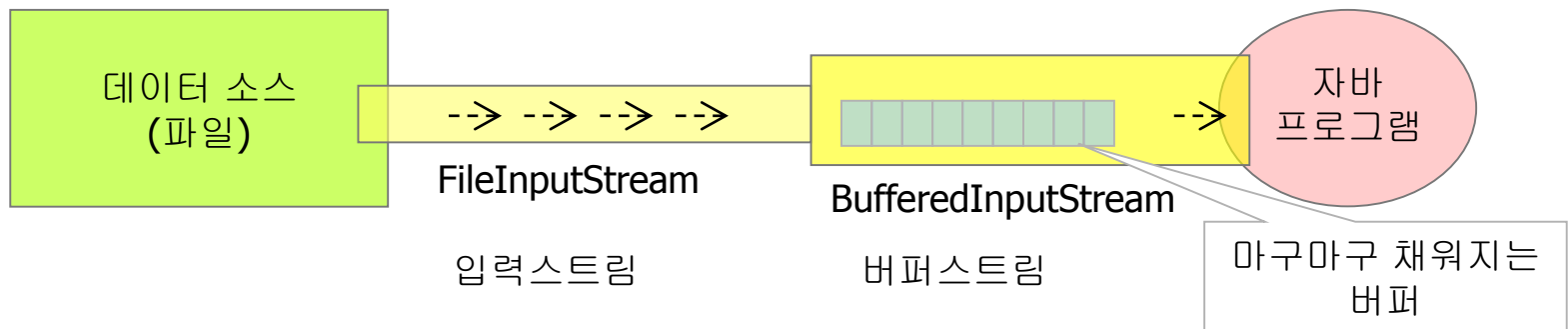
- `FilterInputStream`과 `FilterOutputStream`
 - `InputStream`/`OutputStream`의 자손이면서 모든 보조 스트림의 조상
 - 보조 스트림은 자체적으로 입출력을 수행할 수 없기 때문에 기반 스트림을 필요로 함
 - 생성자
 - `protected FilterInputStream(InputStream in)`
 - `public FilterOutputStream(OutputStream out)`

BufferedInputStream과 BufferedOutputStream

■ BufferedInputStream/ BufferedOutputStream

- 스트림의 입출력 효율을 높이기 위해 버퍼를 사용하는 보조 스트림
- 한 바이트씩 입출력하는 것 보다는 버퍼(바이트 배열)를 이용해서 **한 번**에 **여러 바이트**를 입출력하는 것이 빠르기 때문에 대부분의 입출력 작업에 사용됨

생성자	설 명
BufferedInputStream(InputStream in, int size)	주어진 InputStream인스턴스를 입력소스(input source)로 하며, 지정된 크기(byte 단위)의 버퍼를 갖는 BufferedInputStream 인스턴스를 생성함
BufferedInputStream(InputStream in)	주어진 InputStream인스턴스를 입력소스(input source)로 하며, 버퍼의 크기를 지정해 주지 않으므로 기본적으로 2048byte 크기의 버퍼를 갖게 됨





BufferedInputStream

■ BufferedInputStream

- BufferedInputStream의 버퍼크기는 **입력소스로부터 한 번에 가져올 수 있는 데이터의 크기**로 지정하면 좋다
- 입력소스가 파일인 경우 작게는 512부터 1024 또는 2048 정도의 크기로 하는 것이 보통임

■ 입력 작업 순서

- 프로그램에서 입력소스로부터 데이터를 읽기 위해 처음으로 read 메서드를 호출하면, BufferedInputStream 은 **입력소스로부터 버퍼 크기만큼의 데이터를 읽어다 자신의 내부버퍼에 저장한다.**
- 이제 프로그램에서는 BufferedInputStream의 버퍼에 저장된 데이터를 읽는다
- 외부의 입력소스로부터 읽는 것보다 내부의 버퍼로부터 읽는 것이 훨씬 빠르기 때문에 그만큼 작업 효율이 높아짐
- 프로그램에서 버퍼에 저장된 모든 데이터를 다 읽고 그 다음 데이터를 읽기 위해 read 메서드가 호출되면, BufferedInputStream 은 입력소스로부터 다시 버퍼크기 만큼의 데이터를 읽어다 버퍼에 저장해 놓는다.
- 이와 같은 작업이 계속해서 반복된다.



BufferedOutputStream

■ BufferedOutputStream

메서드/생성자	설 명
BufferedOutputStream(OutputStream out, int size)	주어진 OutputStream인스턴스를 출력소스(output source)로 하며, 지정된 크기(byte 단위)의 버퍼를 갖는 BufferedOutputStream 인스턴스를 생성함
BufferedOutputStream(OutputStream out)	주어진 OutputStream인스턴스를 출력소스(output source)로 하며, 버퍼의 크기를 지정해 주지 않으므로 기본적으로 2048byte 크기의 버퍼를 갖게 됨
flush()	버퍼의 모든 내용을 출력소스에 출력한 다음, 버퍼를 비운다.
close()	flush()를 호출해서 버퍼의 모든 내용을 출력소스에 출력하고, BufferedOutputStream인스턴스가 사용하던 모든 자원을 반환한다.



BufferedOutputStream

■ BufferedOutputStream

- BufferedOutputStream 도 버퍼를 이용해서 출력소스와 작업을 하게 됨

■ 출력 작업 순서

- 입력소스로부터 데이터를 읽을 때와는 반대로, 프로그램에서 write 메서드를 이용한 출력이 BufferedOutputStream의 버퍼에 저장된다.
- 버퍼가 가득 차면, 그 때 버퍼의 모든 내용을 출력소스에 출력한다
- 그리고는 버퍼를 비우고 다시 프로그램으로부터의 출력을 저장할 준비를 한다.
- 버퍼가 가득 찼을 때만 출력소스에 출력을 하기 때문에, 마지막 출력부분이 출력소스에 쓰여지지 못하고 BufferedOutputStream의 버퍼에 남아있는 채로 프로그램이 종료될 수 있다는 점을 주의해야 함
- 프로그램에서 모든 출력작업을 마친 후 BufferedOutputStream에 close()나 flush()를 호출해서 마지막에 버퍼에 있는 모든 내용이 출력소스에 출력되도록 해야 함



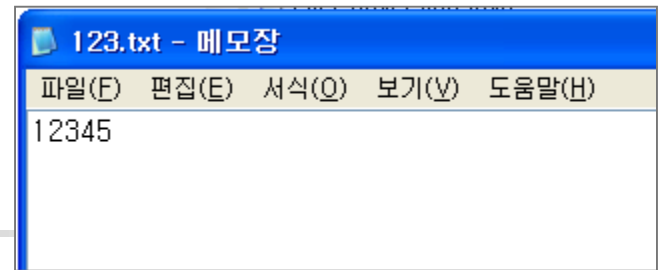
예제 1

```
import java.io.*;
class BufferedInputStreamTest {
    public static void main(String[] args) throws IOException{
        //파일에서 데이터를 읽어와서 화면출력 - 보조 스트림 이용
        FileInputStream fis = new FileInputStream("text/poetry2.txt");
        BufferedInputStream bis = new BufferedInputStream(fis, 1024); //버퍼 사이즈 생략하면
        2048

        int data=0;
        while ((data = bis.read()) != -1)
        {
            System.out.print((char)(data));
        }

        bis.close();
    }
}
```

예제



```
import java.io.*;

class BufferedOutputStreamEx1 {
    public static void main(String args[]) {
        try {
            FileOutputStream fos = new FileOutputStream("123.txt");

            // BufferedOutputStream의 버퍼 크기를 5로 한다.
            BufferedOutputStream bos = new BufferedOutputStream(fos, 5);

            // 파일 123.txt에 1 부터 9까지 출력한다.
            for(int i='1'; i <= '9'; i++) {
                bos.write(i);
            }

            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

크기가 5인 **BufferedOutputStream** 을 이용해서 파일 **123.txt**에 1~9까지 출력하는 예제
결과 : 5까지만 출력

<= 버퍼에 남아있는 데이터가 출력되지 못한 상태로 프로그램이 종료되었기 때문

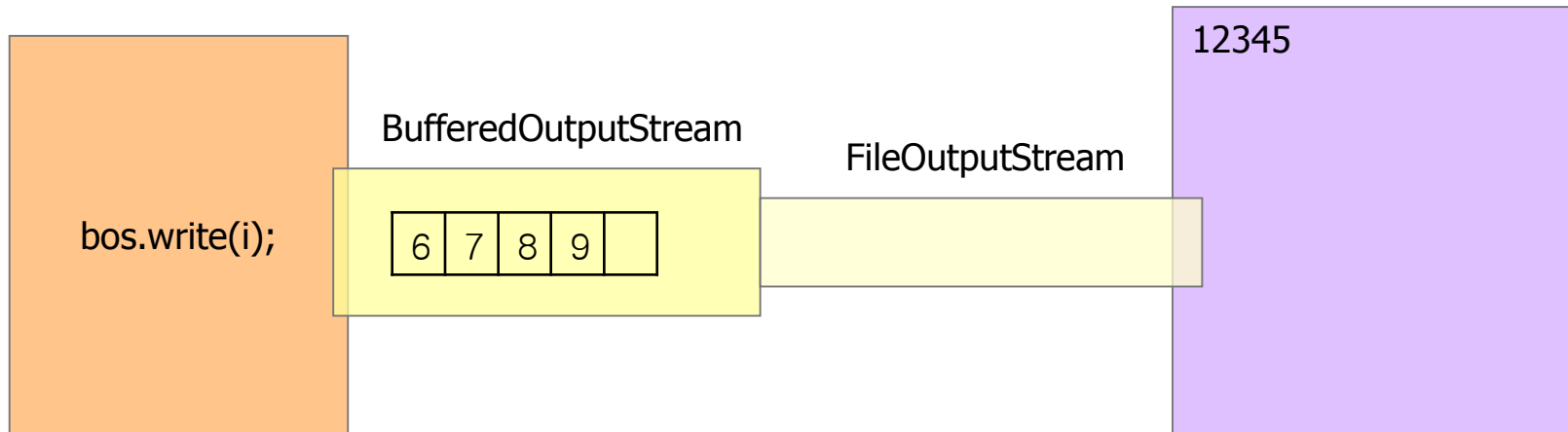
예제

■ 실행결과

자바 프로그램

BufferedOutputStreamEx1

데이터 목적지 : 123.txt



- `fos.close()` 를 호출해서 스트림을 닫아 주기는 했지만, 이렇게 해서는 `BufferedOutputStream`의 버퍼에 있는 내용이 출력되지 않는다
- `bos.close()` 를 호출해서 `BufferedOutputStream` 의 `close()`를 호출해주어 버퍼에 남아있던 모든 내용이 출력됨
- `BufferedOutputStream` 의 `close()` 는 기반 스트림인 `FileOutputStream` 의 `close()`를 호출하기 때문에 `FileOutputStream`의 `close()`는 따로 호출해주지 않아도 됨



실습

- BufferedInputStream을 이용하여 poetry3.txt를 읽어서, BufferedOutputStream 을 이용하여 poetry3_bak.txt로 출력하기



실습

- 1. 키보드로 입력한 내용을 파일(input.txt)에 저장하기
 - 1) 데이터 소스 : 키보드(System.in)
 - System.in.read() 이용
 - 2) 데이터 목적지 : 파일(FileOutputStream)
- 2. BufferedOutputStream 을 이용하여 출력하는 것으로 변경하기
 - (성능 향상을 위해 보조 스트림 이용)



DataInputStream과 DataOutputStream

- DataInputStream과 DataOutputStream
 - 데이터를 읽고 쓰는데 있어서 byte 단위가 아닌, 8가지 기본 자료형의 단위로 읽고 쓸 수 있다는 장점이 있음
 - DataOutputStream이 출력하는 형식은 각 기본 자료형 값을 16진수로 표현하여 저장함
 - 예) int 값을 출력한다면, 4byte의 16진수로 출력됨



DataInputStream과 DataOutputStream

- DataInputStream의 생성자와 메서드

메서드/생성자	설 명
<code>DataInputStream(InputStream in)</code>	주어진 <code>InputStream</code> 인스턴스를 기반스트림으로 하는 <code>DataInputStream</code> 인스턴스를 생성함
<code>boolean readBoolean()</code> <code>byte readByte()</code> <code>char readChar()</code> <code>short readShort()</code> <code>int readInt()</code> <code>long readLong()</code> <code>float readFloat()</code> <code>double readDouble()</code>	각 자료형에 알맞은 값들을 읽어 온다. 더 이상 읽을 값이 없으면 <code>EOFException</code> 을 발생시킴
<code>String readUTF()</code>	UTF 형식으로 쓰여진 문자를 읽는다. 더 이상 읽을 값이 없으면 <code>EOFException</code> 을 발생시킴
<code>int skipBytes(int n)</code>	현재 읽고 있는 위치에서 지정된 숫자(n)만큼을 건너뛰



DataInputStream과 DataOutputStream

- DataOutputStream의 생성자와 메서드

메서드/생성자	설 명
DataOutputStream(OutputStream out)	주어진 OutputStream 인스턴스를 기반스트림으로 하는 DataOutputStream 인스턴스를 생성함
void writeBoolean(boolean b) void writeByte(int b) void writeChar(int c) void writeShort(int s) void writeInt(int i) void writeLong(long l) void writeFloat(float f) void writeDouble(double d)	각 자료형에 알맞은 값들을 출력한다.
void writeUTF(String s)	UTF 형식으로 쓰여진 문자를 출력한다
void writeChars(String s)	주어진 문자열을 출력한다. writeChar(char c) 메서드를 여러 번 호출한 결과와 같다.
int size()	지금까지 DataOutputStream 에 쓰여진 byte의 수를 알려준다



예제

```
import java.io.*;
```

```
class DataOutputStreamEx1 {  
    public static void main(String args[]) {  
        FileOutputStream fos = null;  
        DataOutputStream dos = null;  
  
        try {  
            fos = new FileOutputStream("sample.dat");  
            dos = new DataOutputStream(fos);  
            dos.writeInt(10); //4byte  
            dos.writeFloat(20.0f); //4byte  
            dos.writeBoolean(true); //1byte  
            dos.writeChar('A'); //2byte  
            dos.writeUTF("hello"); //유니코드의 utf-8 형식으로 문자열을 출력하는 메서드  
  
            dos.close();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
} // main
```

- **FileOutputStream** 을 기반으로 하는 **DataOutputStream** 을 생성한 후, **DataOutputStream**의 메서드들을 이용해서 **sample.dat** 파일에 값들을 출력
- 출력한 값들은 이진데이터로 저장됨

예제2

```
10
20.0
true
A
hello
```

```
import java.io.*;
```

```
class DataInputStreamEx1 {
    public static void main(String args[]) {
```

```
        try {
            FileInputStream fis = new FileInputStream("sample.dat");
            DataInputStream dis = new DataInputStream(fis);

            System.out.println(dis.readInt());
            System.out.println(dis.readFloat());
            System.out.println(dis.readBoolean());
            System.out.println(dis.readChar());
            String str = dis.readUTF();
            System.out.println(str);
            dis.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    } // main
}
```

데이터를 읽어 올 때, 아무런 변환이나 자릿수를 셀 필요없이 **readInt** 와 같이, 읽어 올 데이터의 타입에 맞는 메서드를 사용하기만 하면 됨

- **DataOutputStream** 의 **write** 메서드들로 기록한 데이터는

DataInputStream의 **read** 메서드들로 읽는다

- 여러 가지 종류의 자료형으로 출력한 경우, **읽을 때는 반드시 쓰인 순서대로 읽어야** 한다.



예제3

```
import java.io.*;

class DataOutputStreamEx3 {
    public static void main(String args[]) {
        int[] score = { 100, 90, 95, 85, 50 };

        try {
            FileOutputStream fos = new FileOutputStream("score.dat");
            DataOutputStream dos = new DataOutputStream(fos);
            for(int i=0; i<score.length;i++) {
                dos.writeInt(score[i]);
            }
            dos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    } // main
}
```

int 형 배열 **score**의 값들을 **DataOutputStream**을 이용해서 **score.dat** 파일에 출력하는 예제

예제 4

```
100
90
95
85
50
점수의 총합은 420입니다.
```

- score.dat 파일을 읽어서 데이터의 총합을 구하는 예제
- `DataInputStream`의 `readInt()` 와 같이 데이터를 읽는 메서드는 더 이상 읽을 데이터가 없으면 **`EOFException`** 을 발생시킴

```
import java.io.*;

class DataInputStreamEx2 {
    public static void main(String args[]) {
        int sum = 0;
        int score = 0;
        try {
            FileInputStream fis = new FileInputStream("score.dat");
            DataInputStream dis = new DataInputStream(fis);

            while(true) {
                score = dis.readInt();
                System.out.println(score);
                sum += score;
            }
        } catch (EOFException e) {
            System.out.println("점수의 총합은 " + sum + "입니다.");
        } catch (IOException ie) {
            ie.printStackTrace();
        }
    } // main
}
```



문자 기반 스트림



문자 기반 스트림

■ 문자기반 스트림

- 문자 데이터를 다루는 데 사용
- 단순히 2byte 로 스트림을 처리하는 것만을 의미하지는 않음
- 여러 종류의 인코딩과 자바에서 사용하는 유니코드간의 변환을 자동적으로 처리해줌
 - Reader – 특정 인코딩을 읽어서 유니코드로 변환
 - Writer – 유니코드를 특정 인코딩으로 변환하여 저장

■ Reader / Writer

- 바이트 기반 스트림의 조상
 - InputStream/OutputStream
- 문자기반의 스트림의 조상
 - Reader/ Writer
- Reader/ Writer 의 메서드
 - byte 배열 대신 char 배열을 사용한다는 것 외에는 InputStream/OutputStream 의 메서드와 동일



FileReader/FileWriter

- FileReader/FileWriter
 - 파일로부터 텍스트 데이터를 읽고, 파일에 쓰는 데 사용

예제

```
Hello, ¾?³ ???¼¼¿??  
Hello, 안녕하세요?
```

- 같은 내용의 파일(test.txt)을 한번은 **FileInputStream**으로, 다른 한번은 **FileReader** 로 읽어서 화면에 출력
- **FileInputStream**을 사용했을 때는 한글이 깨져서 출력

```
import java.io.*;
```

```
class FileReaderEx1 {  
    public static void main(String args[]) {  
        try {  
            String fileName = "test.txt";  
            FileInputStream fis = new FileInputStream(fileName);  
            FileReader fr = new FileReader(fileName);  
  
            int data = 0;  
            // FileInputStream을 이용해서 파일내용을 읽어 화면에 출력한다.  
            while((data=fis.read())!=-1) {  
                System.out.print((char)data);  
            }  
            System.out.println();  
            fis.close();  
  
            // FileReader를 이용해서 파일내용을 읽어 화면에 출력한다.  
            while((data=fr.read())!=-1) {  
                System.out.print((char)data);  
            }  
            System.out.println();  
            fr.close();  
  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
} // main
```



예제2

- 파일의 공백을 모두 없애는 예제
입력스트림으로 부터 읽은 데이터를 변환해서 출력스트림
에 쓰는 작업의 예

```
import java.io.*;

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

        try {

            FileReader fis = new FileReader("text/poetry.txt");
            FileWriter fos = new FileWriter("text/poetry_bak.txt");
            int data = 0;

            while((data=fis.read())!=-1) {
                if(data!='\t' && data!='\n' && data!=' ' && data !='\r')
                    fos.write(data);
            }

            fis.close();
            fos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    } // main
}
```



StringReader/StringWriter

- StringReader/StringWriter
 - 입출력 대상이 메모리인 스트림
 - StringWriter에 출력되는 데이터는 내부의 StringBuffer에 저장됨
 - 저장된 데이터를 얻어오는 메서드
 - StringBuffer getBuffer()
 - StringWriter에 출력한 데이터가 저장된 StringBuffer를 반환함
 - String toString()
 - StringWriter에 출력된(StringBuffer에 저장된) 문자열을 반환함



예제

```
Input Data :ABCD
Output Data :ABCD
```

```
import java.io.*;

class StringReaderWriterEx
{
    public static void main(String[] args)
    {
        String inputData = "ABCD";
        StringReader input = new StringReader(inputData);
        StringWriter output = new StringWriter();

        int data = 0;

        try {
            while((data = input.read())!=-1) {
                output.write(data);    // void write(int b)
            }
        } catch(IOException e) {}

        System.out.println("Input Data  :" + inputData);
        System.out.println("Output Data :" + output.toString());
        System.out.println("Output Data :" + output.getBuffer().toString());
    }
}
```



예제2

```
import java.io.*;

class StringReaderWriterEx2
{
    public static void main(String[] args)
    {
        try {
            String fileName = "c:WWjavaWWpoetry3.txt";
            FileReader input = new FileReader(fileName);
            StringWriter sw = new StringWriter();

            int data = 0;
            while((data = input.read())!=-1) {
                sw.write(data);          // void write(int b)
            }//while

            System.out.println("---Output Data---\n\n" + sw.toString()); //TextArea에
            보여줄때도 사용
        } catch(IOException e) {
            e.printStackTrace();
        }
    }//main
}
```

---Output Data---

To the Virgins to Make Much of Time - by Walt Whitman

Gather ye rosebuds while ye may,
Old time is still a-flying,
And this same flower that smiles today,
Tomorrow will be die."

시간이 있을 때 장미 봉우리를 거두라.
시간은 흘러
오늘 핀 꽃은
내일이면 질 것이니
계속하려면 아무 키나 누르십시오 . . .



문자 기반의 보조 스트림



문자 기반의 보조 스트림

- BufferedReader/ BufferedWriter
 - 버퍼를 이용해서 입출력의 효율을 높일 수 있도록 해주는 역할
 - BufferedReader의 **readLine()** 을 사용하면 데이터를 라인단위로 읽어 올 수 있다는 장점
 - BufferedWriter의 **newLine()** - 줄바꿈 해주는 메서드

BufferedReader/ BufferedWriter

```
import java.io.*;
class BufferedReaderEx1
{
    public static void main(String
    {
        try {

            FileReader fr = new FileReader("BufferedReaderEx1.java");
            BufferedReader br = new BufferedReader(fr);

            String line = "";
            for(int i=1;(line = br.readLine())!=null;i++) {
                // ";"를 포함한 라인만 출력한다.
                if(line.indexOf(";")!=-1)
                    System.out.println(i+": "+line);
            }
        } catch(IOException e) {}
    } // main
}
```

- **BufferedReader** 의 **readLine()**을 이용해서 파일을 라인단위로 읽은 다음 **indexOf()**를 이용해서 ';'를 포함하고 있는지 확인하여 출력하는 예제
- 파일에서 특정 문자 또는 문자열을 포함한 라인을 쉽게 찾아낼 수 있음을 보여줌

```
C:\WINDOWS\system32\cmd.exe
1:import java.io.*;
8:      FileReader fr = new FileReader("BufferedReaderEx1.java");
;
9:      BufferedReader br = new BufferedReader(fr);
11:     String line = "";
12:     for(int i=1;(line = br.readLine())!=null;i++) {
13:         // ";"를 포함한 라인을 출력한다.
14:         if(line.indexOf(";")!=-1)
15:             System.out.println(i+": "+line);
계속하려면 아무 키나 누르십시오 . . .
```



InputStreamReader /OutputStreamWriter



InputStreamReader/OutputStreamWriter

- InputStreamReader/OutputStreamWriter
 - 바이트 기반 스트림을 문자기반 스트림으로 연결시켜 주는 역할을 함
 - 바이트 기반 스트림의 데이터를 지정된 인코딩의 문자 데이터로 변환하는 작업을 수행함
 - 한글 윈도우에서 중국어로 작성된 파일을 읽을 때 InputStreamReader(InputStream in, String encoding)를 이용해서 인코딩이 중국어로 되어 있다는 것을 지정해주어야 파일의 내용이 깨지지 않고 올바르게 보임
 - 인코딩을 지정해주지 않는다면 OS에서 사용하는 인코딩을 사용해서 파일을 해석해서 보여주기 때문에 원래 작성된 데로 볼 수 없음

• 시스템 속성에서 `sun.jnu.encoding` 의 값을 보면 OS에서 사용하는 인코딩의 종류를 알 수 있음

```
Properties prop = System.getProperties();  
System.out.println(prop.get("sun.jnu.encoding"));
```

MS949



InputStreamReader/OutputStreamWriter

- InputStreamReader의 생성자와 메서드

생성자/ 메서드	설 명
InputStreamReader(InputStream in)	OS에서 사용하는 기본 인코딩의 문자로 변환하는 InputStreamReader 를 생성
InputStreamReader(InputStream in, String encoding)	지정된 인코딩을 사용하는 InputStreamReader를 생성
String getEncoding()	InputStreamReader의 인코딩을 알려줌



InputStreamReader/OutputStreamWriter

- OutputStreamWriter의 생성자와 메서드

생성자/ 메서드	설 명
OutputStreamWriter(OutputStream out)	OS에서 사용하는 기본 인코딩의 문자로 변환하는 OutputStreamWriter 를 생성
OutputStreamWriter(OutputStream out, String charsetName)	지정된 인코딩을 사용하는 OutputStreamWriter를 생성
String getEncoding()	OutputStreamWriter의 인코딩을 알려줌



예제

```
import java.io.*;
```

```
class OutputStreamWriterTest  
{
```

```
    public static void main(String[] args) throws IOException
```

```
    {        //2byte 기반 스트림
```

```
        FileReader fr=new FileReader("poetry.txt");
```

```
        //도스 콘솔에 출력, System.out을 사용하면 1바이트 기반
```

```
        //2바이트 단위로 출력해주는 bridge스트림으로 출력
```

```
        OutputStreamWriter osw=new OutputStreamWriter(System.out);
```

```
        //1바이트로 출력하는 데이터를 2바이트로 만들어서 Writer 객체에 전달
```

```
        int data=0;
```

```
        while ((data=fr.read()) != -1){
```

```
            //System.out.write(data); //2바이트 기반 스트림을 1바이트 기반 스트림으로 출력했기 때문에 한글은 깨짐
```

```
            osw.write(data);
```

```
            //osw.flush();
```

```
        }//while
```

```
        osw.close();
```

```
        fr.close();
```

```
        System.out.close();
```

```
    }//main()
```

```
}
```



예제

```
System.out.println("입력하세요");
try {
    int input = 0;

    while((input=System.in.read())!=-1) {
        System.out.println("input : " + input
            + ", (char)input : " + (char)input);
        //System.out.write(input);
        //System.out.print((char)input);
    }
} catch(IOException e) {
    e.printStackTrace();
}
```

//키보드로 부터 입력 받은 값을 출력 => 한글은 깨짐



예제2

```
import java.io.*;
class InputStreamReaderEx
{
    public static void main(String[] args)
    {
        String line = "";

        try {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);

            System.out.println("사용중인 OS의 인코딩 : " + isr.getEncoding());

            System.out.print("문장을 입력하세요. ");
            while((line = br.readLine()) != null) {
                System.out.println("입력하신 문장 : "+line);
            }

            br.close();
            System.out.println("프로그램을 종료합니다.");
        } catch(IOException e) {}
    } // main
}
```

사용중인 OS의 인코딩 :MS949
문장을 입력하세요. 안녕
입력하신 문장 : 안녕
hello java
입력하신 문장 : hello java
프로그램을 종료합니다.



예제

- BufferedReader의 readLine()을 이용해서 사용자의 화면 입력을 라인단위로 입력받으면 편리
- BufferedReader 와 InputStream인 System.in 을 연결하기 위해 InputStreamReader 를 사용
 - jdk 1.5 부터는 Scanner 가 추가되어 간단하게 처리 가능
- 현재 사용중인 OS의 인코딩을 확인하려면 생성자 InputStreamReader(InputStream in) 를 이용해서 InputStreamReader의 인스턴스를 생성한 다음, getEncoding() 을 호출하면 됨
- 한글 윈도우즈 에서 사용하는 인코딩의 종류 - MS949



실습1

- BufferedReader을 이용하여 poetry.txt를 읽어서, BufferedWriter 을 이용하여 poetry_bak.txt로 출력하기
 - BufferedReader 의 readLine()을 이용한 경우에는 BufferedWriter의 newLine()도 이용한다
 - read()를 이용한 경우에는 write()만 써도 됨



실습2

- 1. 키보드로 입력한 내용을 파일(result.txt)에 저장하기
 - 1) 데이터 소스 : 키보드(System.in) - 1바이트 기반
 - 2) 데이터 목적지 : 파일(FileWriter) - 2바이트 기반
 - 한글도 깨지지 않고 저장되도록 해야 함
 - InputStreamReader 이용하여 System.in을 2바이트 기반으로 변환해야 함
- 2. BufferedWriter 을 이용하여 출력하는 것으로 변경하기



표준 입출력과 File



표준 입출력 - System.in, System.out, System.err

■ 표준 입출력

- 콘솔(Console, 도스창)을 통한 데이터 입력과 콘솔로의 데이터 출력을 의미함
- 표준 입출력을 위해 3가지 입출력 스트림 System.in, System.out, System.err 을 제공
 - 이들은 자바 어플리케이션의 실행과 동시에 사용할 수 있게 자동적으로 생성되기 때문에 개발자가 별도로 스트림을 생성하는 코드를 작성하지 않고도 사용 가능

System.in - 콘솔로부터 데이터를 입력받는데 사용
System.out - 콘솔로 데이터를 출력하는데 사용
System.err - 콘솔로 데이터를 출력하는 데 사용



표준 입출력

```
public final class System{  
    public final static InputStream in = new InputStream();  
    public final static PrintStream out = new PrintStream();  
    public final static PrintStream err = new PrintStream();  
}
```

PrintStream - **OutputStream**의 자식클래스

- 출력용으로 사용하는 **print()**, **println()** 메소드를 사용할 수 있도록 기능을 보강한 클래스



예제

```
import java.io.*;
```

```
class StandardIOEx1  
{
```

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

```
        try {
```

```
            int input = 0;
```

```
            while((input=System.in.read())!=-1) {
```

```
                System.out.println("input :" + input + ", (char)input :" + (char)input);
```

```
                //System.out.write(input);
```

```
            }
```

```
        } catch(IOException e) {
```

```
            e.printStackTrace();
```

```
        }
```

```
    } // main
```

```
}
```

```
abc  
input :97, <char>input :a  
input :98, <char>input :b  
input :99, <char>input :c  
input :13, <char>input :  
input :10, <char>input :  
12
```

```
input :49, <char>input :1  
input :50, <char>input :2  
input :13, <char>input :  
input :10, <char>input :  
12
```



예제

- 콘솔 입력은 버퍼를 가지고 있기 때문에 Backspace 키를 이용해서 편집이 가능하며 한 번에 버퍼의 크기만큼 입력이 가능함
- Enter 키나 입력의 끝을 알리는 [Ctrl + Z] 를 누르기 전까지는 아직 데이터가 입력 중인 것으로 간주되어 커서가 입력을 계속 기다리는 상태에 머무르게 됨
- 콘솔에 데이터를 입력하고 Enter키를 누르면 입력대기상태에서 벗어나 입력된 데이터를 읽기 시작하고 입력된 데이터를 모두 읽으면 다시 입력대기 상태가 됨
- [Ctrl + Z]를 입력하면 read()는 입력이 종료되었음을 인식하고 -1을 반환
- Enter 키를 누르는 것은 두 개의 특수문자 '\r', '\n'이 입력된 것으로 간주됨
 - '\r' - (carriage return), 커서를 현재 라인의 첫 번째 컬럼으로 이동시킴
 - '\n' - 커서를 다음 줄로 이동시키는 줄바꿈(new line)을 함



예제2

- 키보드 입력, 도스 콘솔창에 출력

Node Stream : System.in/ System.out (1바이트 기반)

Bridge Stream : InputStreamReader / OutputStreamWriter (2바이트로 변환)

```
import java.io.*;

class StandardInOutTest
{
    public static void main(String[] args) throws IOException
    {
        InputStream is = System.in;
        PrintStream ps = System.out;

        InputStreamReader ir = new InputStreamReader(is); //한글 입력도 가능
        OutputStreamWriter ow = new OutputStreamWriter(ps);

        int data = 0;
        while ((data = ir.read()) != -1)
        {
            //System.out.print((char)data);

            ow.write(data);
            ow.flush(); //반드시 해줘야 함
        } //while

        ir.close(); ow.close();
        is.close(); ps.close();
    } //main()
}
```



예제3

- 키보드 입력에 도스 콘솔창에 출력

Node Stream : System.in/ System.out (1바이트 기반)

Bridge Stream : InputStreamReader / OutputStreamWriter (2바이트로 변환)

Filter Stream : BufferedReader / BufferedWriter

```
import java.io.*;
class StandardInOutTest2
{
    public static void main(String[] args) throws IOException{
        /*InputStreamReader ir=new InputStreamReader(System.in);
        OutputStreamWriter ow=new OutputStreamWriter(System.out);
        BufferedReader br=new BufferedReader(ir);
        BufferedWriter bw=new BufferedWriter(ow);*/

        BufferedReader br
            =new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bw
            =new BufferedWriter(new OutputStreamWriter(System.out));

        //줄(line)단위로 입력 받기
        String line="";
        while ((line=br.readLine()) != null)
        {
            bw.write(line);
            bw.newLine(); //줄바꿈 주기
            bw.flush(); //반드시 써야 함
        }//while
        br.close();
        bw.close();
    }
}
```



File 클래스



File

- File 클래스를 통해서 파일과 디렉토리를 다룰 수 있다
 - File 인스턴스는 파일일 수도 있고, 디렉토리일 수도 있다

File의 생성자와 경로와 관련된 메서드

생성자/ 메서드	설 명
<code>File(String fileName)</code>	주어진 문자열(fileName)을 이름으로 갖는 파일을 위한 File인스턴스를 생성 파일뿐만 아니라 디렉토리도 같은 방법으로 다룸 fileName - 주로 경로(path)를 포함해서 지정 파일 이름만 사용하면 프로그램이 실행되는 위치가 경로로 간주됨
<code>File(String pathName, String fileName)</code> <code>File(File pathName, String fileName)</code>	파일의 경로와 이름을 따로 분리해서 지정할 수 있도록 한 생성자.
<code>String getName()</code>	파일이름을 String으로 반환
<code>String getPath()</code>	파일의 경로(path)를 String으로 반환
<code>String getAbsolutePath()</code> <code>File getAbsoluteFile()</code>	파일의 절대경로를 String으로 반환 파일의 절대경로를 File로 반환
<code>String getParent()</code> <code>File getParentFile()</code>	파일의 조상 디렉토리를 String으로 반환 파일의 조상 디렉토리를 File로 반환
<code>String getCanonicalPath()</code> <code>File getCanonicalFile()</code>	파일의 정규경로를 String으로 반환 파일의 정규경로를 File로 반환



경로와 관련된 File의 멤버변수

멤버변수	설 명
static String pathSeparator	OS에서 사용하는 경로 구분자. 윈도우 ";", 유닉스 ":"
static char pathSeparatorChar	OS에서 사용하는 경로 구분자. 윈도우 ";", 유닉스 ":"
static String separator	OS에서 사용하는 이름 구분자. 윈도우 "W", 유닉스 "/"
static char separatorChar	OS에서 사용하는 경로 구분자. 윈도우 "W", 유닉스 "/"

예제

시스템 속성 중에서 **user.dir**의 값을 확인하면 현재 프로그램이 실행중인 디렉토리를 알 수 있다

```
import java.io.*;
class FileEx1
{
    public static void main(String[] args) throws IOException
    {
        File f = new File("E:\\WW\\java\\WWsrc\\WWFileTest.java");
        if (!f.exists()){
            System.out.println("원본 파일이 없습니다!!");
            return;
        }
        String fileName = f.getName();
        int pos = fileName.lastIndexOf(".");

        System.out.println("경로를 제외한 파일이름 - " + f.getName());
        System.out.println("확장자를 제외한 파일이름 - " + fileName.substring(0,pos));
        System.out.println("확장자 - " + fileName.substring(pos+1));

        System.out.println("경로를 포함한 파일이름 - " + f.getPath());
        System.out.println("파일의 절대경로 - " + f.getAbsolutePath());
        System.out.println("파일이 속해 있는 디렉토리 - " + f.getParent());
        System.out.println();
        System.out.println("File.pathSeparator - " + File.pathSeparator);
        System.out.println("File.pathSeparatorChar - " + File.pathSeparatorChar);
        System.out.println("File.separator - " + File.separator);
        System.out.println("File.separatorChar - " + File.separatorChar);
        System.out.println();
        System.out.println("user.dir="+System.getProperty("user.dir"));
        System.out.println("sun.boot.class.path=" + System.getProperty("sun.boot.class.path"));
    }
}
```



예제

```
File f2 = new File("e:\\java\\src","newFile.txt");
f2.createNewFile();

if (f2.exists())
{
    System.out.println("파일 생성됨!!");
    //f2.delete();
}
```



```
경로를 제외한 파일이름 - FileTest.java
확장자를 제외한 파일이름 - FileTest
확장자 - java
경로를 포함한 파일이름 - E:\java\src\FileTest.java
파일의 절대경로 - E:\java\src\FileTest.java
파일이 속해 있는 디렉토리 - E:\java\src
```

```
File.pathSeparator - ;
File.pathSeparatorChar - ;
File.separator - \
File.separatorChar - \
```

```
user.dir=E:\java\src
sun.boot.class.path=C:\Program Files\Java\jdk1.6.0_04\jre\lib\resources.jar;C:\Program Files\Java\jdk1.6.0_04\jre\lib\rt.jar;C:\Program Files\Java\jdk1.6.0_04\jre\lib\sunrsasign.jar;C:\Program Files\Java\jdk1.6.0_04\jre\lib\jsse.jar;C:\Program Files\Java\jdk1.6.0_04\jre\lib\jce.jar;C:\Program Files\Java\jdk1.6.0_04\jre\lib\charsets.jar;C:\Program Files\Java\jdk1.6.0_04\jre\classes
```

- File 인스턴스를 생성했다고 해서 파일이나 디렉토리가 생성되는 것은 아님
- 파일명이나 디렉토리명으로 지정된 문자열이 유효하지 않더라도 컴파일 에러나 예외를 발생시키지 않음
- 새로운 파일을 생성하기 위해서는 File 인스턴스를 생성한 다음, 출력스트림을 생성하거나 createNewFile 을 호출해야 함

- 이미 존재하는 파일을 참조할 때
File f = new File("d:\java\src","test.txt");
- 기존에는 없는 파일을 새로 생성할 때
File f = new File("d:\java\src","newFile.txt");
f.createNewFile(); //새로운 파일이 생성된다.

File의 메서드

메서드	설 명
boolean canRead() boolean canWrite()	읽을 수 있는 파일인지 검사한다. 쓸 수 있는 파일인지 검사한다.
boolean exists()	파일이 존재하는지 검사한다.
boolean isDirectory() boolean isFile()	디렉토리인지 확인한다. 파일인지 확인한다.
int compareTo(File pathname)	주어진 파일 또는 디렉토리를 비교한다. 같으면 0을 반환, 다르면 1 또는 -1을 반환
boolean createNewFile()	아무런 내용이 없는 새로운 파일을 생성한다.(이미 존재하면 생성되지 않는다.)
boolean delete()	파일을 삭제한다. 삭제하는데 성공하면 true, 실패하면 false반환
String[] list()	디렉토리의 파일목록(디렉토리 포함)을 String 배열로 반환
String[] list(FilenameFilter filter)	FilenameFilter인스턴스에 구현된 조건에 맞는 파일을 String 배열로 반환함
File[] listFiles()	디렉토리의 파일목록(디렉토리 포함)을 File배열로 반환함
long length()	파일 크기
boolean mkdir() boolean mkdirs()	디렉토리 생성
boolean renameTo(File dest)	이름변경

e:>java FileEx2 e:□java□src

지정한 디렉토리에 포함된 파일과 디렉토리의 목록을 보여주는 예제

예제

```
import java.io.*;
import java.util.Scanner;
class FileEx2{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("디렉토리를 입력하세요");
        String dir=sc.nextLine();

        File f = new File(dir);

        if(!f.exists() || !f.isDirectory()) {
            System.out.println("유효하지 않은 디렉토리입니다.");
            System.exit(0);
        }

        File[] files = f.listFiles();

        for(int i=0; i < files.length; i++) {
            String fileName = files[i].getName();
            System.out.println(files[i].isDirectory() ? "["+fileName+"]" : fileName + ", " + files[i].length());
        }
    } // main
}
```

```
FileEx1.java
FileEx2.class
FileEx2.java
FileEx3.java
FileEx4.java
FileEx5.java
FileEx6.java
FileEx7.java
FileEx8.java
FileEx9.java
FileMerge.java
FileReaderEx1.java
FileSplit.java
FileTest.java
FileViewer.java
[path1]
[rain]
[star]
[sun]
```

예제2

```
import java.io.*;
```

```
//File 클래스
```

```
class FileTest
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
    String filename=args[0];
```

```
    File file=new File(filename);
```

```
    System.out.println("파일명: " + file.getName());
```

```
    System.out.println("파일 상대경로: " + file.getPath());
```

```
    System.out.println("파일 절대경로: " + file.getAbsolutePath());
```

```
    System.out.println("파일 크기: " + file.length() + " byte");
```

```
    File file2=new File("path1", "test.txt"); //디렉토리, 파일명
```

```
    //File file3=new File("path1\\test2.txt");
```

```
    File file3=new File("path1" + File.separator + "test2.txt");
```

```
        //운영체제에 따른 디렉토리 구분자
```

```
    System.out.println("파일의 상위부모 경로: " + file2.getParent());
```

```
    //boolean 형태 정보를 제공하는 메서드
```

```
    System.out.println("파일 존재 여부: " + file2.exists());
```

```
    System.out.println(file3.exists()? "존재함": "존재하지 않음");
```

```
파일명: FileTest.java
파일 상대경로: FileTest.java
파일 절대경로: E:\java\src\FileTest.java
파일 크기: 1619 byte
파일의 상위부모 경로: path1
파일 존재 여부: true
존재하지 않음
file이 파일인지 여부: true
디렉토리 아님
쓰기 가능
읽기 가능
false
file5 삭제여부 : true
```



예제2

```
System.out.println("file0이 파일인지 여부: " +file.isFile());
System.out.println(file.isDirectory()? "디렉토리임": "디렉토리 아님");
System.out.println(file.canWrite()? "쓰기 가능": "쓰기 불가");
System.out.println(file.canRead()? "읽기 가능": "읽기 불가");
```

```
File file4=new File("rain");
file4.mkdir(); //디렉토리 생성
```

```
File file5=new File("sun", "moon");
//sun디렉토리 생성하고, 그 아래에 하위 디렉토리 moon 생성
file5.mkdirs();
```

```
boolean r= file4.renameTo(new File("star")); //rain 을 star로 이름 변경
System.out.println(r); //true, false 반환됨, 이름변경 성공하면 true
boolean r2= file5.delete(); //디렉토리 삭제시 비어있어야 함
System.out.println("file5 삭제여부 : "+r2); //moon이 삭제됨
```

```
}
```

```
}
```

실습

- d:\Wjava\W 아래 있는 파일과 디렉토리 목록을 도스 콘솔에 출력하기
 - `public File[] listFiles()` 이용
- d:\Wjava\W 아래의 디렉토리 목록을 얻어오는데, 디렉토리이면 DIR 이라고 표시하고, 파일이면 파일 크기 bytes 를 표시한다.
- 파일인 경우 확장자가 java 인 파일과 txt 파일만 콘솔에 출력
 - `endsWith(".txt")` 이용

파일\DIR 명	size
FileEx1.java	1147bytes
FileEx2.java	568bytes
FileEx3.java	1421bytes
FileEx4.java	852bytes
FileEx5.java	2986bytes
FileEx6.java	1617bytes
FileEx7.java	686bytes
FileEx8.java	1130bytes
FileEx9.java	742bytes
FileMerge.java	1137bytes
FileReaderEx1.java	700bytes
FileSplit.java	978bytes
FileTest.java	1619bytes
FileViewer.java	267bytes
path1	DIR
star	DIR
sun	DIR



직렬화(Serialization)

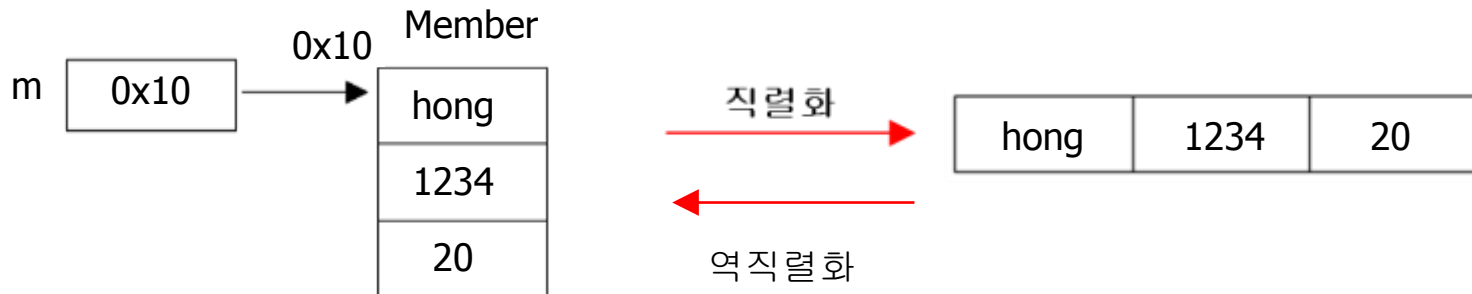
직렬화(Serialization)

■ 직렬화(serialization)

- 객체를 네트워크를 통해 전송하거나, 파일로 저장하고자 할 때 직렬화해야 함
- 객체에 저장된 데이터를 스트림에 쓰기위해 '연속적인 데이터'로 변환하는 것
- 객체의 인스턴스변수들의 값을 일렬로 나열하는 것

■ 역직렬화(deserialization)

- 반대로 스트림으로부터 데이터를 읽어서 객체를 만드는 것



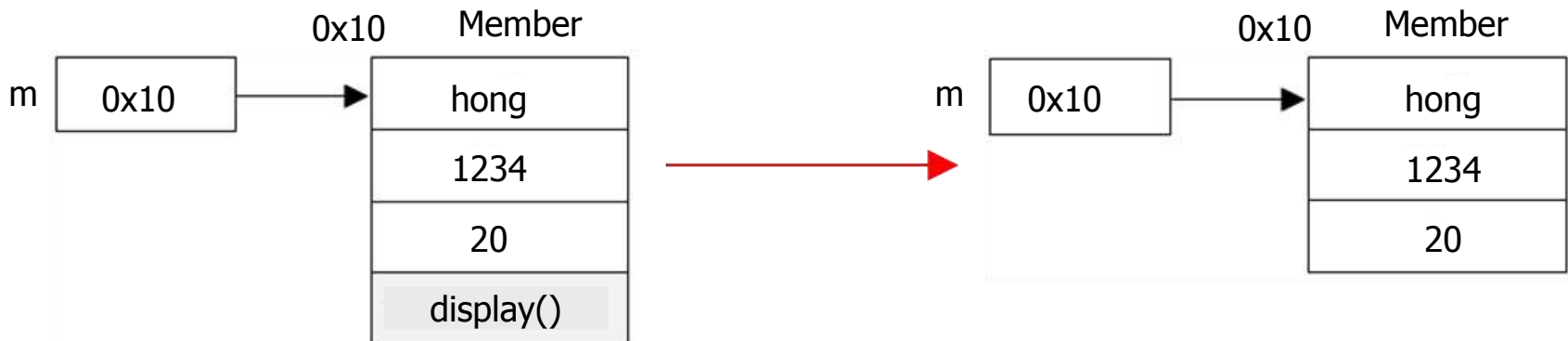
```
Member m = new Member("hong", "1234", 20);
```

userid, pwd, age

직렬화(Serialization)

■ 직렬화(serialization)

- 객체를 저장하기 위해서는 객체를 직렬화해야 함
- 객체를 저장한다는 것은 **객체의 모든 인스턴스변수의 값을 저장하는 것**



저장했던 객체를 다시 생성하려면, 객체를 생성한 후에 저장했던 값을 읽어서 생성한 객체의 인스턴스 변수에 저장하면 됨



ObjectInputStream, ObjectOutputStream

- ObjectInputStream, ObjectOutputStream
 - 객체를 직렬화하여 입출력할 수 있게 해주는 보조스트림

```
ObjectInputStream(InputStream in);  
ObjectOutputStream(OutputStream out);
```

- 객체를 파일에 저장(직렬화)하는 방법

```
FileOutputStream fos=new FileOutputStream("objfile.ser");  
ObjectOutputStream out=new ObjectOutputStream(fos);  
out.writeObject(new Member());
```

- 파일에 저장된 객체를 다시 읽어오는 방법

```
FileInputStream fis=new FileInputStream("objfile.ser");  
ObjectInputStream in=new ObjectInputStream(fis);  
Member m = (Member)in.readObject();
```



직렬화가 가능한 클래스 만들기

- 직렬화하고자 하는 클래스가 `java.io.Serializable` 인터페이스를 구현하도록 하면 됨
- 제어자 `transient`가 붙은 인스턴스변수는 직렬화 대상에서 제외됨

```
public class Member implements Serializable
{
    String userid;
    transient String pwd; //직렬화 대상에서 제외
    int age;
}
```



직렬화 가능한 클래스 만들기

- Serializable을 구현하지 않은 클래스의 인스턴스도 직렬화 대상에서 제외
- Serializable을 구현하지 않은 조상의 멤버들은 직렬화 대상에서 제외된다.



```
public class Member implements Serializable //직렬화 가능 클래스(네트워크 통해 전송한다고 표시)
{
    private String userid;
    private transient String pwd; //직렬화 대상에서 제외(비밀번호는 보내지 않겠다, pwd는 null이 출
    력됨)
    private int age;

    public Member(){

    }

    public Member(String userid, String pwd, int age){
        this.userid=userid;
        this.pwd = pwd ;
        this.age=age;
    }

    public void display(){
        System.out.println("-----" + userid + "의 정보-----");
        System.out.println("아이디: " + userid);
        System.out.println("비밀번호 : " + pwd);
        System.out.println("나이 : " + age);
    }
}

//
```

예제-WriteMember.java

```
import java.io.*;
import java.awt.*;
import java.util.*;
class WriteMember //객체를 직렬화 - 객체를 파일로 저장
{
    public static void main(String[] args) throws IOException
    {
        Member m1=new Member("hong", "1234", 25);
        Member m2=new Member("kim", "5678", 20);
        ArrayList<Member> list = new ArrayList<Member>();
        list.add(m1);
        list.add(m2);

        String fileName="member.ser";
        FileOutputStream fos=new FileOutputStream(fileName);
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        //DataOutput 인터페이스를 구현하고 있는 스트림.
        oos.writeObject(m1);
        oos.writeObject(m2);
        oos.writeObject(new Date());
        Frame f=new Frame("Object Stream 실습");
        oos.writeObject(f);
        oos.writeObject(list);

        //oos.flush();
        //fos.close();
        oos.close();

        System.out.println("직렬화가 잘 끝났습니다.");
    }
}
//
```

생성한 객체를 직렬화하여 파일
(member.ser)에 저장하는 예제

writeInt(), writeDouble()

예제-ReadMember.java

```
import java.io.*;
import java.awt.*;
import java.util.*;
class ReadMember{
    public static void main(String[] args) throws Exception{
        FileInputStream fis=new FileInputStream("member.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);

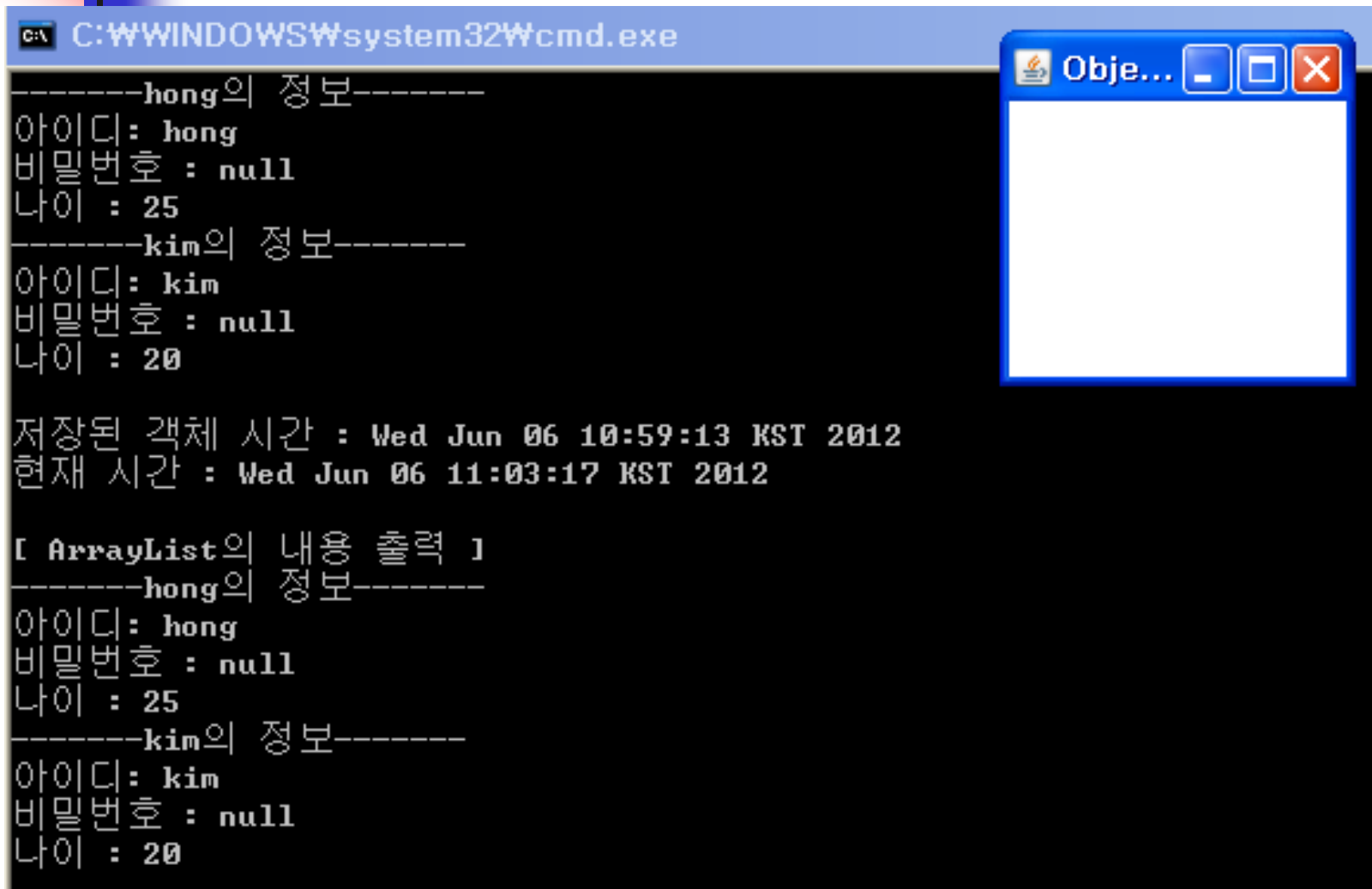
        // 객체를 읽을 때는 출력한 순서와 일치해야한다.
        Object o1= ois.readObject();
        Member m1=(Member)o1;
        Member m2=(Member)ois.readObject();

        Date d=(Date)ois.readObject();
        Frame f=(Frame)ois.readObject();
        ArrayList<Member> list = (ArrayList<Member>)ois.readObject();

        m1.display();
        m2.display();
        Date date = new Date();
        System.out.println("\n저장된 객체 시간 : " + d);
        System.out.println("현재 시간 : " + date );
        f.setSize(150, 150);
        f.setVisible(true);
        System.out.println("\n[ ArrayList의 내용 출력 ]");
        for (Member m : list){
            m.display();
        }
        //fis.close();
        ois.close();
    }
}
```

- 직렬화한 객체를 역직렬화하는 예제
- 객체를 역직렬화할 때는 직렬화할 때의 순서와 일치해야 함
- 직렬화할 객체가 많을 때는 각 객체를 개별적으로 직렬화하는 것보다 **ArrayList**와 같은 컬렉션에 저장해서 직렬화하는 것이 좋다.

예제



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The command prompt displays the following text:

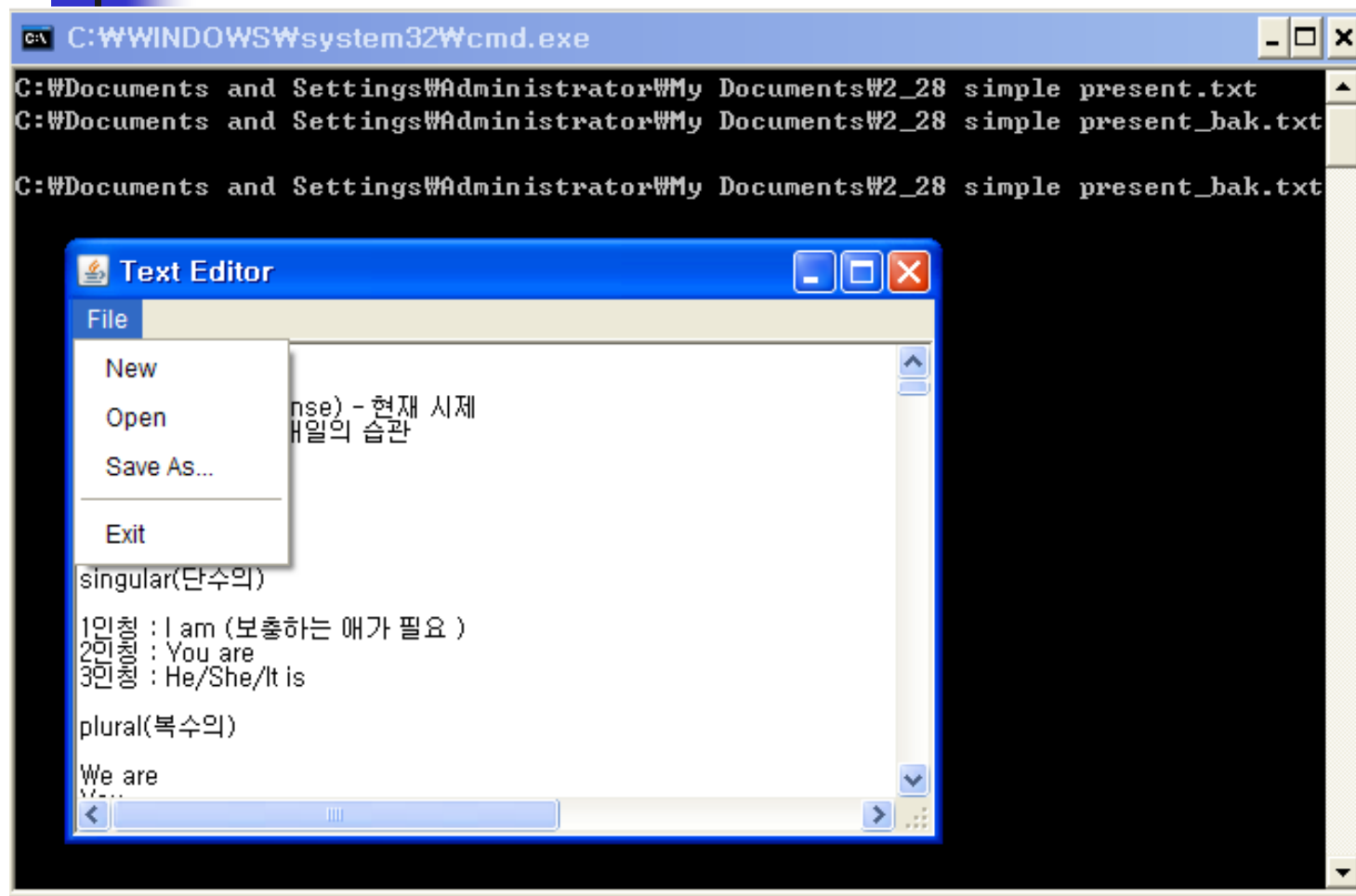
```
-----hong의 정보-----  
아이디: hong  
비밀번호 : null  
나이 : 25  
-----kim의 정보-----  
아이디: kim  
비밀번호 : null  
나이 : 20  
  
저장된 객체 시간 : Wed Jun 06 10:59:13 KST 2012  
현재 시간 : Wed Jun 06 11:03:17 KST 2012  
  
[ ArrayList의 내용 출력 ]  
-----hong의 정보-----  
아이디: hong  
비밀번호 : null  
나이 : 25  
-----kim의 정보-----  
아이디: kim  
비밀번호 : null  
나이 : 20
```

There is a small, empty window titled "Obje..." floating on the right side of the command prompt.

■ 메뉴 만들기

- Open 클릭시 선택한 파일의 내용을 읽어서 TextArea에 보여준다
 - FileReader, BufferedReader, StringWriter(sw) 이용
 - `ta.setText(sw.toString());`
- Save 클릭시 TextArea의 내용을 지정된 파일에 저장한다
 - FileWriter, BufferedWriter 이용

예





인코딩 및 유니코드

■ 인코딩

- 문자코드를 컴퓨터가 이해할 수 있는 0과1의 바이너리 값을 가지는 연속적인 비트 형태로 대응시켜주는 작업
- 컴퓨터가 이해할 수 있는 코드 형태로 만들어 주는 것

■ 문자 코드(Character code)

- 문자들의 집합과 이 문자들을 나타내기 위해 정한 숫자 (문자코드)들을 1대1로 연결시켜 놓은 것
- 예) ASCII 문자 코드
 - A : 65, a : 97



문자코드

■ EUC-KR 인코딩

- 유닉스 운영체제에서 영어는 KS C 5636(ASCII문자에 대한 표준)을, 한글은 KS C 5601을 사용하는 것
- ASCII 문자코드는 1바이트로 표현,
- 한글 문자코드는 2바이트로 표현



유니코드

■ 유니코드

- 인간이 사용하는 모든 언어를 표현할 수 있도록 하기 위하여 만들어짐
- 기존 언어의 인코딩 체계를 모두 포함할 수 있도록 고안된 커다란 문자 집합
- 두 개의 대표적 문자 인코딩이 있음

■ UTF-8

- ASCII 문자코드는 1바이트로 인코딩, 다른 문자들은 2바이트나 그 이상으로 인코딩하는 방식
- 한글은 3바이트로 인코딩
- 기존의 ASCII 문자코드 체계와 그대로 호환이 가능
 - 인터넷상에서 문서를 교환하기 위한 기본적인 인코딩으로 환영받음
 - XML 문서 : 디폴트로 UTF-8 인코딩 사용

■ UTF-16

- 간단하게 2 바이트를 사용하여 모든 문자 코드를 표현



예제

- 다음 배열에서 확장자만 출력하시오

```
String[] fileArr={"test.java","member","write.jsp","sun.gif"};
```