Character 입출력 Stream

경복대학교
소프트웨어융합과
배희호 교수
010-2369-4112
031-570-9600
hhbae@kbu.ac.kr

Character Stream

- 문자로 표현되지 않는 Data는 다루지 않음
  - 문자 Stream은 Image, Video과 같은 Binary Data는 입출력 할 수 없음
- 입출력 단위가 오직 UniCode(16 Bit) 문자나 문자열들을 읽고 쓰기 위한 Stream
- JAVA에서 File을 읽는 방법은 한 글자씩 읽는 방법과 File을 한 줄씩 읽는 방법이 있음

	File 입력(읽기) File을 읽어 콘솔에 출력	File 출력(쓰기) File로 만들어 HDD에 저장
Byte 단위로 저장하기 위한 Stream	FileInputStream	FileOutputStream
문자(UTF-8) 단위로 저장하기 위한 Stream	FileReader	FileWriter

Character Stream

- 입력 Stream

Class 이름	대응 Class	설명
BufferedInputStream	BufferedReader	Buffer 기능이 있음
DataInputStream	없음	기본형의 Data 입력
FileInputStream	FileReader	File 입력
FilterInputStream	FilterReader	Filter Class의 최상위 Class
InputStream	Reader	기본 입력 Stream

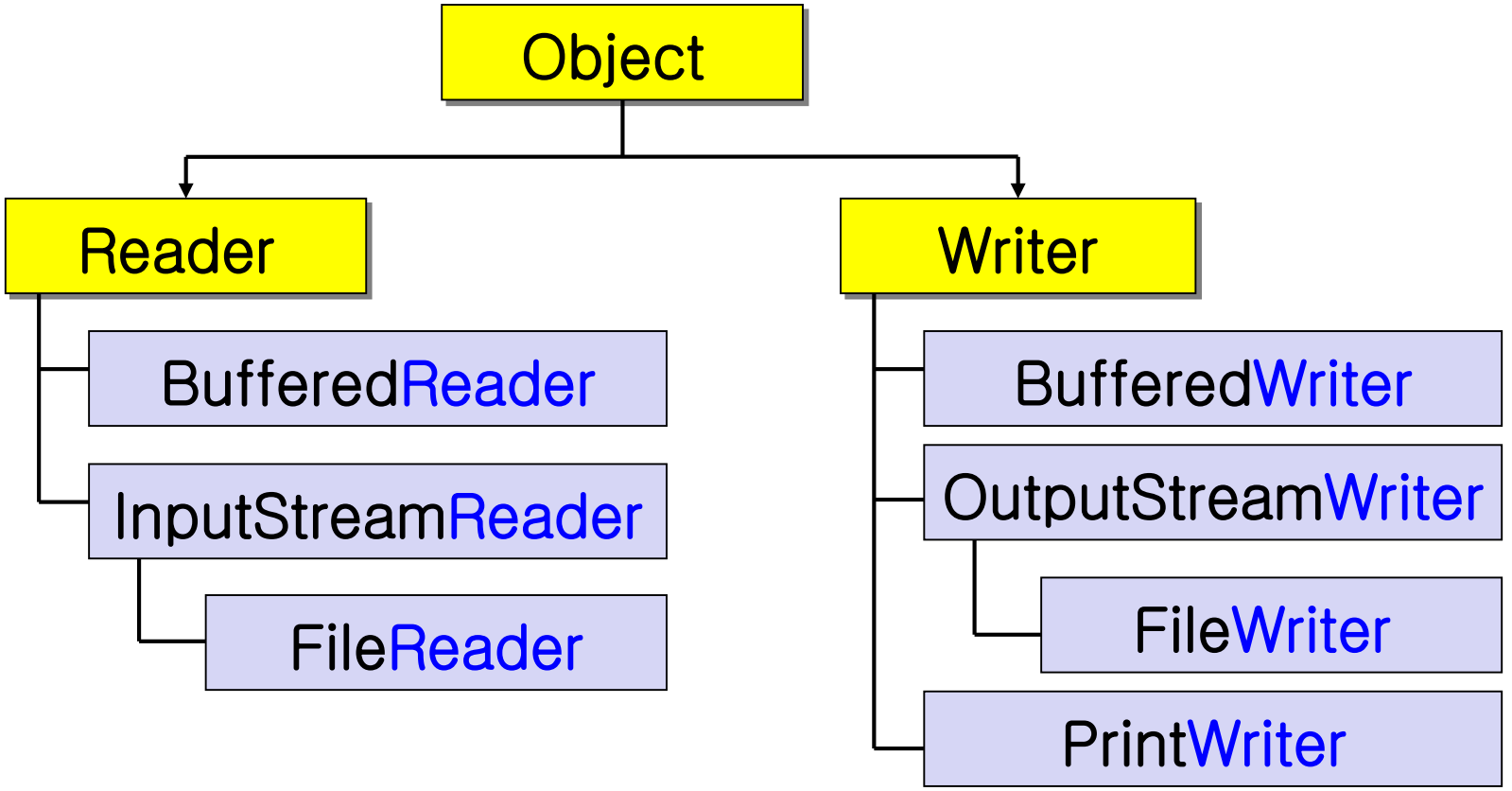
Character Stream

- 출력 Stream

Class 이름	대응 Class	설명
BufferedOutputStream	BufferedWriter	Buffer 기능이 있음
DataOutputStream	없음	기본형의 Data 출력
FileOutputStream	FileWriter	File 출력
PrintStream	PrintWriter	표준 출력
OutputStream	Writer	기본 출력 Stream

Character Stream

- Chraracter Stream과 연관된 Class
  - Class 이름에 Reader(입력)와 Writer(출력)가 붙음



Character Stream

- Reader / Writer Class
  - java.io Package에 포함
  - Abstract Class
  - 문자 입출력 처리를 위한 기능을 가진 모든 Class의 Super Class
- InputStreamReader / OutputStreamReader Class
  - 이 Class는 Byte Stream과 문자 Stream을 연결시켜 줌
  - 지정된 문자 집합을 이용
  - InputStreamReader
    - Byte들을 읽어 지정된 문자 집합 내의 문자로 Encoding함
  - OutputStreamReader
    - 문자를 Byte들로 Decoding하여 Stream으로 출력함

Character Stream

- FileReader / FileWriter Class
  - FileReader와 FileWriter는 Text File로부터 직접 문자 Data를 읽고 기록함
  - FileReader
    - FileInputStream과 연결하여 읽어 들인 Byte를 문자 Data로 변환함
  - FileWriter
    - FileOutputStream을 이용하여 문자 Stream을 Byte Stream으로 변환하여 File에 기록함

Character Stream

- Reader

Class	설 명	Stream
Reader	문자 입력 Stram을 위한 추상 Class	2차 Stream
BufferedReader	문자 Buffer 입력, 라인 해석	2차 Stream
LineNumberReader	문자 입력 시 Line Number를 유지	2차 Stream
CharArrayReader	문자 배열에서 읽어 들임	1차 Stream
InputStreamReader	Byte Stream을 Character Stream으로 변환	2차 Stream
FileReader	File에서 Byte를 읽어 들여 Character Stream으로 변환	1차 Stream
FilterReader	Filter 적용(filtered) 문자 입력을 위한 추상 Class	2차 Stream
PushbackReader	읽어들인 문자를 되돌림 (pushback)	2차 Stream
PipedReader	PipedWriter에서 읽어 들임	1차 Stream
StringReader	문자열에서 읽어 들임	1차 Stream

Character Stream

- Writer

Class	설 명	Stream
Writer	문자 출력 Stream을 위한 추상 Class	2차 Stream
BufferedWriter	문자 Stream에 Buffer 출력, 줄 바꿈 사용	2차 Stream
CharArrayWriter	문자 Stream에 문자 배열 출력	1차 Stream
OutputStreamWriter	문자 Stream을 Byte Stream으로 변환	2차 Stream
FileWriter	문자 Stream을 Byte File로 변환	1차 Stream
FilterWriter	Filter 적용(filtered) 문자 출력을 위한 추상 Class	2차 Stream
PipedWriter	PipedReader에 출력	1차 Stream
StringWriter	문자열 출력	1차 Stream
PrintWriter	Writer 값과 Object를 프린트	2차 Stream

Reader 클래스

- 문자 Stream을 읽어 들이기 위한 추상 Class
- 모든 Character 기반 입력 Class는 Reader Class를 상속받아 만들어지며, FileReader, BufferedReader, InputStreamReader Class가 있음

Reader 클래스

- Method

Method	설명
void close()	Stream 닫고 그것과 관련된 System Resource를 방출
void mark (int readAheadLimit)	Stream에 현재 위치를 표시
boolean markSupported()	Stream이 mark() 를 지원하는지 여부 확인
int read()	Stream에서 단일 문자를 읽음
int read(char[] cbuf)	Stream에서 cbuf 배열 크기만큼 읽어 buf에 저장하고 그 크기를 반환
int read(char[] cbuf, int off, int len)	Stream에서 len에 지정된 크기만큼 문자를 읽어 cbuf 버퍼의 off 위치에 저장하고 크기를 반환
int read(CharBuffer target)	CharBuffer형인 target에서 문자열일 읽어옴
boolean ready()	다음 read()문을 수행할 수 있는지 판별, Stream이 준비되었는지 반환
void reset()	Stream의 시작위치를 가장 가까운 mark 위치로 이동
long skip(long n)	n만큼의 문자들을 건너뜀

Writer 클래스

- 문자 기반 출력 Stream의 최상위 추상 Class
- 모든 Character 기반 출력 Class는 Writer Class를 상속받아 만들어지며, FileWriter, BufferedWriter, PrintWriter, OutputStreamWriter Class가 있음

Writer 클래스

- Method

Method	설명
Writer append(char c)	Writer에 문자를 추가
Writer append (CharSequence csq)	Writer에 CharSequence scq를 추가
Writer append(CharSequence csq, int start, int end)	Writer에 CharSequence csq의 start부터 end까지의 문자를 추가
void close()	Stream을 닫음
void flush()	Buffer에 남은 출력 Stream을 출력
void write(char[] cbuf)	배열 cbuf의 내용을 출력
void write(char[] cbuf, int off, int len)	배열 cbuf에서 off를 시작지점으로 len 만큼의 문자를 출력
void write(int c)	c만큼의 글자 출력
void write(String str)	문자열 str을 출력
void write(String str, int off, int len)	문자열 str에서 off를 시작지점으로 len만큼 출력

Reader/Writer 예제

```
public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        Reader reader = new InputStreamReader(inputStream);
        OutputStream outputStream = System.out;
        Writer writer = new OutputStreamWriter(outputStream);

        int count;
        char[] buffer = new char[100];
        while (true) {
            try {
                writer.write("입력해주세요 ");
                writer.flush();
                if (!((count = reader.read(buffer)) != EOF))
                    break;
                String data = new String(buffer, 0, count);
                writer.write(data);
                writer.flush();
```

Reader/Writer 예제

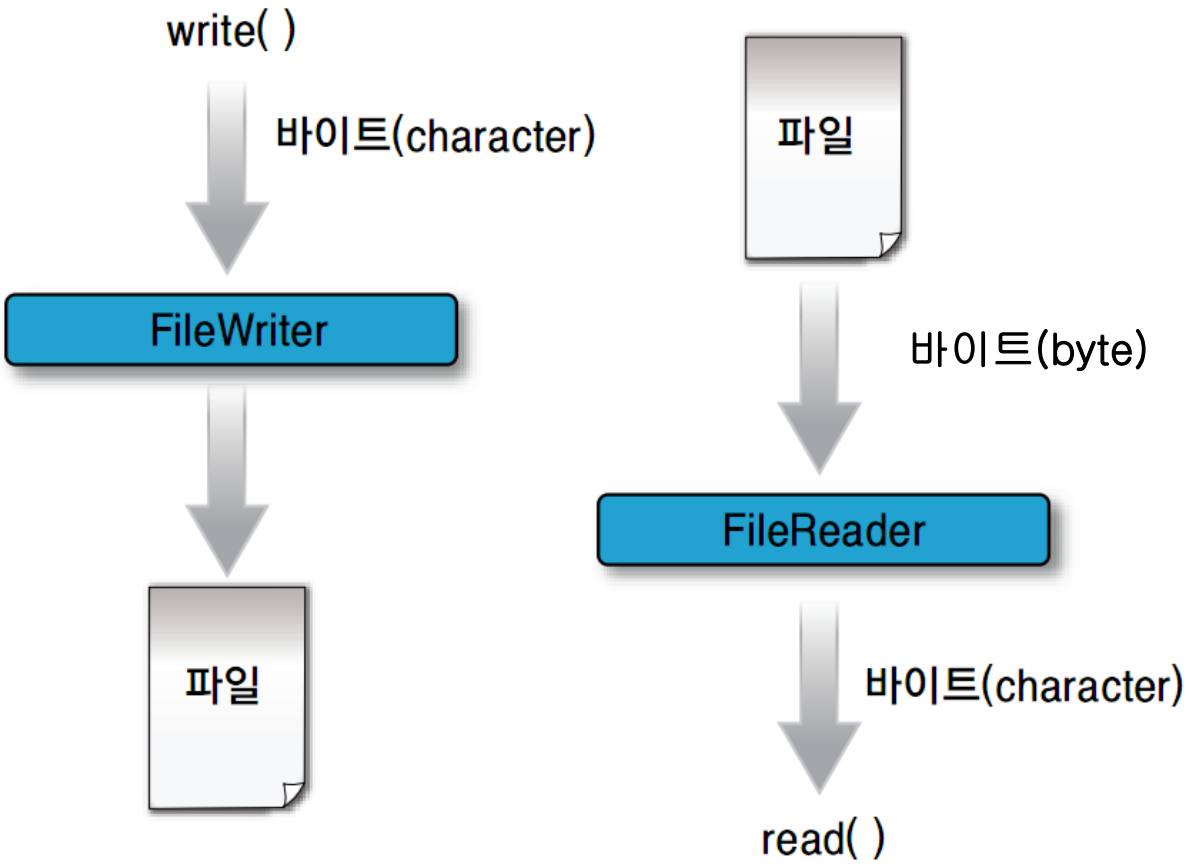
```
            } catch (IOException e) {
                System.err.println(e.getMessage());
            }
        }
        try {
            writer.close();
            reader.close();
        } catch (IOException e) {
            System.err.println(e.getMessage());
        }
    }
}
```

File Reader/FileWriter 클래스

- FileReader와 FileWriter Class는 File에 저장된 Byte를 Unicode 문자로 변환해서 읽어 들이거나 출력할 Unicode 문자를 Default 문자 Encoding의 Byte로 변환해서 File에 저장하는 데 사용되는 입출력 Class
- FileReader와 FileWriter는 각각 InputStreamReader나 OutputStreamWriter의 하위 Class로 Unicode 문자와 Byte 간의 변환 기능을 포함함

FileReader/FileWriter 클래스

- FileWriter Class와 FileReader Class의 역할



FileReader 클래스

- FileReader
  - InputStreamReader Class로부터 상속된 Class
  - File을 읽을 때는 기본적으로 java.io.FileReader가 사용
  - 생성된 File을 Object로 선언하고, 선언된 File Object를 인자로 FileReader를 선언하면 File에 직접 접근하여 File을 읽을 수 있음
  - File로부터 한 글자씩 읽어올 수 있는 기능을 제공

- BufferedReader
  - FileReader의 확장판 같은 개념인 BufferedReader
  - File을 한 문장씩 읽기 때문에 FileReader 보다는 훨씬 빠름
  - File을 읽는 Code의 대부분은 BufferedReader로 만들어져 있음

FileReader 클래스

- 생성자

생성자	설명
FileReader(File file) throws IOException	file로 지정된 File에 대한 입력 Stream을 생성
FileReader(FileDescriptor fd) throws IOException	fd로 지정된 FileDescriptor에 대한 입력 Stream을 생성
FileReader(String fileName) throws IOException	fileName로 지정한 경로의 File에 대한 입력 Stram을 생성

FileReader 클래스

- Text File을 읽는 Program(미완성)

```
public static void main(String args[]) {
        FileReader reader = new FileReader(".//data//sample.txt");  // 오픈
        while (true) {
            int ch = reader.read();
            if (ch == -1)
                break;
            System.out.print((char) ch);
        }
        reader.close();                                  // 파일을 닫는 부분
    }
}
```

파일을 읽어서 처리하는 부분

FileReader 클래스

- Text File을 읽는 Program(완성)

```
public static void main(String args[]) {
        try {
            FileReader reader = new FileReader(“.//data//sample.txt");
            while (true) {
                int ch = reader.read();
                if (ch == -1)
                    break;
                System.out.print((char) ch);
            }
            reader.close();
        } catch (FileNotFoundException e) {
              System.out.println("파일이 존재하지 않습니다.");
        } catch (IOException e) {
              System.out.println("파일을 읽을 수 없습니다.");
        }
    }
```

익셉션이 발생하는 부분을 try 문으로 묶었다.

FileReader의 생성자가 발생하는 익셉션을 처리

FileReader의 close() 메소드가 발생하는 익셉션을 처리

FileReader 클래스

- FileReader Class를 이용하여 Text File을 읽는 방법
  - 1단계: File을 연다

```
FileReader reader = new FileReader("test.txt");
```

생성자에서 현재 Directory의 test.txt File을 연다

FileReader 클래스

- 2단계 : File을 읽는다

```
data = reader.read( );
```

↓

이 Method는 File에 있는 문자 하나를 읽는다

```
while (true) {
        int data = reader.read( );  // 데이터를 읽고
        if (data < 0)
            break;
        char ch = (char) data;  // char 타입으로 캐스트

}
```

Data 처리 로직

FileReader 클래스

- 3단계: File을 닫는다

```
reader.close( );
```

File을 닫는 Method

FileReader 클래스

- 한꺼번에 여러 문자를 읽는 read() Method

```
int num = reader.read(arr);
```

파일로부터 읽은 문자를 담을 char 배열

```
char arr[ ] = new char[100];
int num = reader.read(arr);
```

```
char arr[ ];
int num = reader.read(arr);
```

FileReader 예제 1

```
public static void main(String[] args) {
    String path = ".₩₩data₩₩sample.txt";
    File file = new File(path);
    if (file.exists()) {
        try {
            FileReader reader = new FileReader(path);   // 파일 여는 부분
            int ch;
            while ((ch = reader.read()) != EOF) {
                System.out.print((char) ch);
            }
            reader.close();
        }  catch (IOException e) {
            System.out.println("파일을 읽을 수 없습니다.");
        }
    } else {
        System.out.println("파일이 존재하지 않아요");
    }
}
```

FileReader 예제 1

- 한꺼번에 여러 문자를 읽는 read() 메소드

```
int num = reader.read(arr);
```

파일로부터 읽은 문자를 담을 char 배열

[올바른 예]

```
char arr[] = new char[100];
int num = reader.read(arr);
```

[잘못된 예]

```
char arr[];
int num = reader.read(arr);
```

FileReader 예제 2

- FileReader를 이용하여 컴퓨터의 windows 디렉터리에 있는 system.ini 파일을 읽고 화면에 출력하라

```
public class Main {
    public static void main(String[] args) {
        try {
            FileReader in = new FileReader("c:₩₩windows₩₩system.ini");
            int ch;
            while ((ch = in.read()) != EOF) {
                System.out.print((char) ch);
            }
            in.close();
        } catch (IOException e) {
            System.out.println("입출력 오류");
        }
    }
}
```

FileWriter 클래스

- File에 문자를 쓰기 위해 활용되는 클래스로 OutputStreamWriter를 상속
- File을 쓰는 기본적인 클래스는 java.io.FileWriter
- FileReader와 쌍을 이루고 있으며 기본적으로 문자열, int 등 File에 쓰는 것이 가능
- FileWriter는 기존의 File의 내용을 모두 삭제하고, 그 위에 새로 쓰기 때문에 신중하게 사용해야 함
- FileWriter 클래스만을 활용하여 File에 내용을 쓰는 경우, FileWriter 객체를 생성하고 그 인자로 File Instance를 담은 뒤에 여기에 내용을 쓰고 flush()를 호출 함
- File 이어쓰기 기능은 FileWriter의 선언 부분에 두 번째 인자로 boolean 타입의 true를 같이 보내면 됨

FileWriter 클래스

- 생성자

생성자	설명
FileWriter(File file) throws IOException	file로 지정된 File에 대한 출력 Stream을 생성
FileWriter (File file, boolean append)	file로 지정된 File에 append 모드를 설정하여 출력 Stream을 생성 true이면 File을 추가 모드로 열면 기존 File의 내용이후부터 File이 쓰여지게 됨
FileWriter(FileDescriptor fd) throws IOException	fd로 지정된 FileDescriptor에 대한 출력 Stream을 생성
FileWriter(String fileName) throws IOException	fileName로 지정한 경로의 File에 대한 출력 Stream을 생성
FileWriter (String fileName, boolean append) throws IOException	fineName로 지정한 경로의 File에 append 모드를 설정하여 출력 Stream을 생성

FileWriter 클래스

- FileWriter Class를 이용하여 Text를 File에 쓰는 방법
  - 1단계: File을 연다

```
FileWriter writer = new FileWriter("test.txt");
```

현재 Directory에 test.txt 파일을 새로 만들어 연다

  - 2단계: File에 문자를 쓴다

```
writer.write(ch);
```

이 문자를 파일에 쓴다

  - 3단계: File을 닫는다

```
writer.close( );
```

파일을 닫는 메소드

FileWriter 클래스

- 한꺼번에 여러 문자를 출력하는 write() Method

```
writer.write(arr);
```

char 배열

arr 배열에 있는 모든 문자들을 파일로 출력

FileWriter 사용 예

- c:₩test.txt File에 문자 출력 Stream을 생성하는 Code

```
FileWriter fout = new FileWriter("c:₩₩tmp₩₩test.txt");
```

- File에 문자 출력

```
FileWriter fout = new FileWriter("c:₩₩tmp₩₩test.txt");
fout.write('A'); // 문자 'A' 출력
fout.close();
```

FileWriter 클래스

- FileWriter Class
  - OutputStreamWriter Class로부터 상속된 Class

- FileWriter Class의 생성자

형식

```
FileWriter(String filepath) throws IOException
FileWriter(String filepath, boolean append) throws IOException
FileWriter(File fileObj) throws IOException
```

- FileReader 클래스의 생성자

형식

```
FileReader(String filepath)
FileReader(File fileObj)
```

FileWriter 예제 1

```
public static void main(String[] args) {
    String path = ".₩₩data₩₩output.txt";

    char[] txt = {'경', '복', ' ', '대', '학', '교'};
    try {
        FileWriter writer = new FileWriter(path);
        for (int cnt = 0; cnt < txt.length; cnt++)
            writer.write(txt[cnt]);
        writer.flush();
        writer.close();
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }
}
```

프로그램을 실행하면 output.txt 라는 파일이 생성되고, 그 파일을 텍스트 에디터로 열어보면 프로그램에서 출력한 내용이 들어있는 것을 확인할 수 있음

write( ) 메소드

- write(int i) Method로 한 byte를 출력하는 Method인데 이 Method의 인자는 Byte가 아닌 정수형을 사용
- write() Method의 형태는 3가지가 있는데 인자로 byte[]를 사용하는 것과 배열과 함께 시작 위치 및 크기를 지정하는 Method가 있음
- 이 Method를 사용하면 출력하고자 하는 배열의 지정된 위치에서 정해진 크기만큼 출력이 가능
- 입력과 달리 출력은 출력이 끝났음을 알려 주기 위한 Method인 flush() Method가 하나 더 있음

flush( ) 메소드

- flush() Method를 제대로 사용하려면 Buffer를 알아야 함
- Buffer는 일종의 완충 지대로 입출력을 조금 더 빨리 할 수 있게 도와줌. 모든 출력은 도착 지점으로 바로 나가지 않고 먼저 Buffer에 쌓고, Buffer에 Data가 충분히 쌓인 후 flush() 명령을 받으면 현재 Buffer에 있던 모든 내용을 도착 지점으로 내보내고 Buffer를 비워 버림
- flush() Method를 호출하지 않으면 Buffer로만 출력되기 때문에 실제로 도착 지점에서는 아무런 Data를 받지 못하는 경우가 발생할 수 있음
- 이런 이유로 일반적인 OutputStream에서는 Data를 출력하고 나면 자동적으로 flush() Method를 호출하게 만들 수 있는 기능을 제공하는 경우가 많음

FileWriter 예제 2

```
public static void main(String[] args) {
    String path = ".₩₩data₩₩kyungbok.txt";
    String message = "KOREA Fighting!₩n" +
                     "안녕하세요, 경복대학교에서 공부합니다₩n";
    char[] intxt = new char[message.length()];
    message.getChars(0, message.length(), intxt, 0);

    try {
        FileWriter writer = new FileWriter(path);
        writer.write(intxt);
        writer.append(message);
        writer.flush();
        System.out.println("파일 생성 성공");
        writer.close();
    } catch (IOException e) {
        System.err.println(e.getMessage());
    }
}
```

입출력 예외 처리

- JAVA는 입출력 시 발생할 수 있는 예외 처리를 요구
- 입출력 예외 처리 방법
  - 예외 처리는 호출한 메소드에 넘겨주는 방법
  - 메소드 내에서 직접 처리하는 방법

입출력 예외 처리

- 예외를 호출한 Method로 넘겨주는 방법

```
public static void main(String args[]) throws IOException {
    ..........
    FileWriter fw = new FileWriter(args[0]);
    ............
    fw.write("Line " + i + "₩n");
    ...........
    fw.close();
}
```

호출한 메소드에 예외를 넘김

입출력 예외 처리

- 예외를 메소드 내에서 처리하는 방법

```
public static void main(String args[]) {
   ............
   try {                                          예외 발생 가능 영역 지정
   FileWriter fw = new FileWriter(args[0]);
   ............
   fw.write("Line " + i + "₩n");
   ............
   fw.close();
   }
   catch(IOException e) {                         예외 처리 루틴 작성
      System.our.println("Exception: " + e);
   }
   ..........
}
```

입출력 예외 처리

- 입출력 예외 처리의 일반적인 예

```
public static void main(String args[]) throws IOException {
  ...// 예외를 호출한 메소드로 넘겨주는 방법
  FileWriter fw = new FileWriter(args[0]);
  ..........
  fw.write("Line " + i + "");
  ..........
  fw.close();
}

public static void main(String args[]) {    // 예외를 직접 처리하는 방법

  try   {   // 예외 발생 가능 영역 지정
    FileWriter fw = new FileWriter(args[0]);
    ..........
    fw.write("Line " + i + "");
    ..........
    fw.close();
  }
  catch(IOException e) {   // 예외 처리 루틴 작성
    System.our.println("Exception: " + e);
  }
```