

```
for object to mirror_mod.mirror_object
operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True
```

```
@selection at the end -add
mirror_ob.select= 1
modifier_ob.select=1
context.scene.objects.active
= bpy.context.selected_object
data.objects[one.name].select
print("please select exactly one mirror")
```

```
-- OPERATOR CLASSES -----
bpy.types.Operator):
    X mirror to the selected
    object.mirror_mirror_x"
    mirror X"
```

Java 기초

입출력

1. I/O의 개요

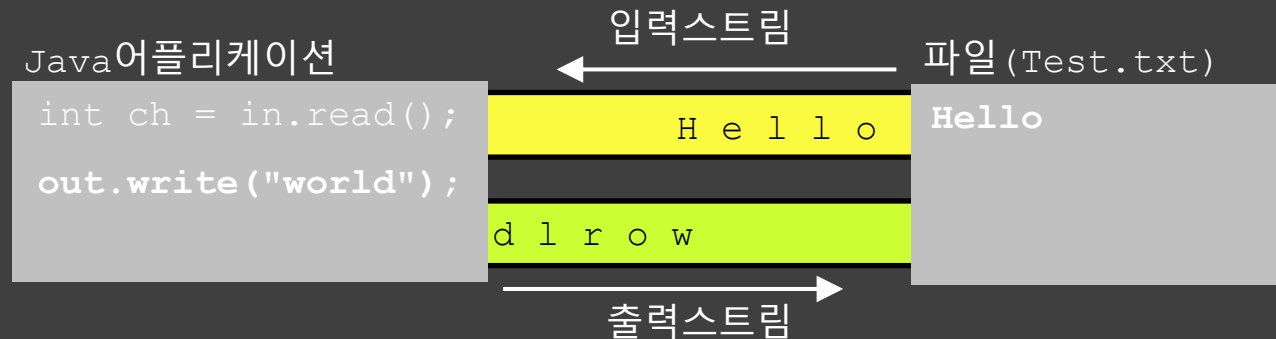
1. 입출력(I/O)와 스트림(Stream)

▶ 입출력(I/O)

- 입출력 : 두 개의 단말기 혹은 대상간에 데이터를 주고받는 것

▶ 스트림

- 데이터를 운반(입출력)하는데 사용되는 연결통로
- 연속적인 데이터의 흐름을 물(stream)에 비유해서 붙여진 이름
- 하나의 스트림으로 입출력을 동시에 수행할 수 없다.(단방향 통신)
- 입출력을 동시에 수행하려면, 2개의 스트림이 필요하다.



1. I/O의 개요

2. 바이트 기반 스트림

- 데이터를 바이트 단위로 주고 받는다

InputStream	OutputStream
abstract int read()	abstract void write(int b)
int read(byte[] b)	void write(byte[] b)
int read(byte[] b, int off, int len)	void write(byte[] b, int off, int len)

입력스트림	출력스트림	대상
FileInputStream	FileOutputStream	파일
ByteArrayInputStream	ByteArrayOutputStream	메모리
PipedInputStream	PipedOutputStream	프로세스
AudioInputStream	AudioOutputStream	오디오장치

1. I/O의 개요

3. 보조 스트림

- 스트림의 기능을 향상시키거나 새로운 기능을 추가하기 위해 사용
- 독립적으로 입출력을 수행할 수 없다.

입력	출력	설명
FilterInputStream	FilterOutputStream	필터를 이용한 입출력 처리
BufferedInputStream	BufferedOutputStream	버퍼를 이용한 입출력 성능향상
DataInputStream	DataOutputStream	int, float와 같은 기본형 단위(primitive type)로 데이터를 처리하는 기능
SequenceInputStream	SequenceOutputStream	두 개의 스트림을 하나로 연결
LineNumberInputStream	없음	읽어 온 데이터의 라인 번호를 카운트 (JDK 1.1부터 LineNumberReader로 대체)
ObjectInputStream	ObjectOutputStream	데이터를 객체단위로 읽고 쓰는데 사용. 주로 파일을 이용하며 객체 직렬화와 관련있음
없음	PrintStream	버퍼를 이용하며, 추가적인 print관련 기능(print, printf, println메서드)
PushbackInputStream	없음	버퍼를 이용해서 읽어 온 데이터를 다시 되돌리는 기능 (unread, push back to buffer)

1. I/O의 개요

4. 문자기반 스트림 – Reader, Writer

- 입출력 단위가 문자(char, 2 byte)인 스트림.

바이트기반 스트림	문자기반 스트림	대상	바이트기반 보조스트림	문자기반 보조스트림
<code>FileInputStream</code> <code>FileOutputStream</code>	<code>FileReader</code> <code>FileWriter</code>	파일	<code>BufferedInputStream</code> <code>BufferedOutputStream</code>	<code>BufferedReader</code> <code>BufferedWriter</code>
<code>ByteArrayInputStream</code> <code>ByteArrayOutputStream</code>	<code>CharArrayReader</code> <code>CharArrayWriter</code>	메모리	<code>FilterInputStream</code> <code>FilterOutputStream</code>	<code>FilterReader</code> <code>FilterWriter</code>
<code>PipedInputStream</code> <code>PipedOutputStream</code>	<code>PipedReader</code> <code>PipedWriter</code>	프로세스	<code>LineNumberInputStream</code>	<code>LineNumberReader</code>
<code>StringBufferInputStream</code> <code>StringBufferOutputStream</code>	<code>StringReader</code> <code>StringWriter</code>	메모리	<code>PrintStream</code>	<code>PrintWriter</code>
			<code>PushbackInputStream</code>	<code>PushbackReader</code>

InputStream → Reader
OutputStream → Writer

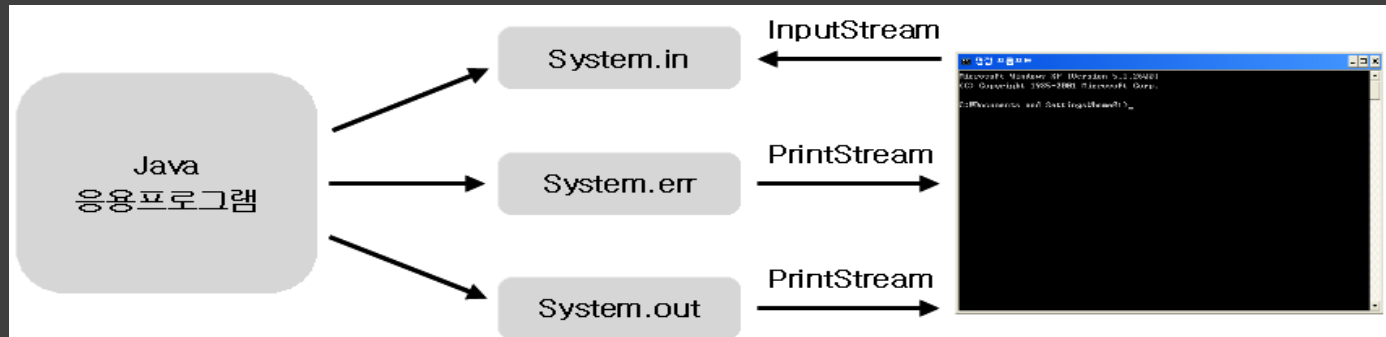
InputStream	Reader
abstract int read() int read(byte [] b) int read(byte [] b, int off, int len)	int read() int read(char [] cbuf) abstract int read(char [] cbuf, int off, int len)
OutputStream	Writer
abstract void write(int b) void write(byte [] b) void write(byte [] b, int off, int len)	void write(int c) void write(char [] cbuf) abstract void write(char [] cbuf, int off, int len) void write(String str) void write(String str, int off, int len)

1. I/O의 개요

5. 표준 입출력 – System.in, System.out, System.err

- 콘솔(console, 화면)을 통한 데이터의 입출력을 ‘표준 입출력’이라 한다.
- JVM이 시작되면서 자동적으로 생성되는 스트림이다.

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



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

```
static void setOut(PrintStream out)
static void setErr(PrintStream err)
static void setIn(InputStream in)
```

1. I/O의 개요

5. 표준 입출력 – System.in, System.out, System.err

- System.in 예제(1)

```
public class SystemInExample1 {  
    public static void main(String[] args) throws Exception {  
        System.out.println("== 메뉴 ==");  
        System.out.println("1. 예금 조회");  
        System.out.println("2. 예금 출금");  
        System.out.println("3. 예금 입금");  
        System.out.println("4. 종료 하기");  
        System.out.print("메뉴를 선택하세요: ");  
  
        InputStream is = System.in;  
        char inputChar = (char) is.read();  
        switch(inputChar) {  
            case '1':  
                System.out.println("예금 조회를 선택하셨습니다.");  
                break;  
            case '2':  
                System.out.println("예금 출금을 선택하셨습니다.");  
                break;  
            case '3':  
                System.out.println("예금 입금을 선택하셨습니다.");  
                break;  
            case '4':  
                System.out.println("종료 하기를 선택하셨습니다.");  
                break;  
        }  
    }  
}
```

1. I/O의 개요

5. 표준 입출력 – System.in, System.out, System.err

- System.in 예제(2)

```
public class SystemInExample2 {  
    public static void main(String[] args) throws Exception {  
        InputStream is = System.in;  
  
        byte[] datas = new byte[100];  
  
        System.out.print("이름: ");  
        int nameBytes = is.read(datas);  
        String name = new String(datas, 0, nameBytes-2);  
  
        System.out.print("하고 싶은말: ");  
        int commentBytes = is.read(datas);  
        String comment = new String(datas, 0, commentBytes-2);  
  
        System.out.println("입력한 이름: " + name);  
        System.out.println("입력한 하고 싶은말: " + comment);  
    }  
}
```


1. I/O의 개요

5. 표준 입출력 – System.in, System.out, System.err

- System.out 예제(1)

```
public class SystemOutExample {  
    public static void main(String[] args) throws Exception {  
        OutputStream os = System.out;  
  
        for(byte b=48; b<58; b++) {os.write(b);}  
        os.write(10);  
  
        for(byte b=97; b<123; b++) {os.write(b);}  
        os.write(10);  
  
        String hangul = "가나다라마바사아자차카타파하";  
        byte[] hangulBytes = hangul.getBytes();  
        os.write(hangulBytes);  
  
        os.flush();  
    }  
}
```

1. I/O의 개요

5. 표준 입출력 – System.in, System.out, System.err

- Scanner 예제(1)

```
public class ScannerExample {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        System.out.print("문자열 입력> ");  
        String inputString = scanner.nextLine();  
        System.out.println(inputString);  
        System.out.println();  
  
        System.out.print("정수 입력> ");  
        int inputInt = scanner.nextInt();  
        System.out.println(inputInt);  
        System.out.println();  
  
        System.out.print("실수 입력> ");  
        double inputDouble = scanner.nextDouble();  
        System.out.println(inputDouble);  
    }  
}
```

2. File

1. File

- 물리적인 File 처리에 대한 모든 기능을 갖고있는 클래스
- 물리적인 파일을 통한 각종 기능을 제공한다.

생성자

[File](#) ([File](#) parent, [String](#) child)

친추상 경로명 및 자식 경로명 스트링로부터 새로운 File 의 인스턴스를 생성합니다.

[File](#) ([String](#) pathname)

지정된 경로명 스트링을 추상 경로명으로 변환해, 새로운 File 의 인스턴스를 생성합니다.

[File](#) ([String](#) parent, [String](#) child)

친경로명 스트링 및 자식 경로명 스트링로부터 새로운 File 의 인스턴스를 생성합니다.

[File](#) ([URI](#) uri)

지정된 file: URI 를 추상 경로명으로 변환해, 새로운 File 의 인스턴스를 생성합니다.

2. File

2. File 메서드

메소드의 개요	
boolean	canExecute () 이 추상 경로명이 가리키는 파일을 어플리케이션을 실행할 수 있을지 여부를 판정합니다.
boolean	canRead () 이 추상 경로명이 가리키는 파일을 어플리케이션을 읽어들일 수 있을지 여부를 판정합니다.
boolean	canWrite () 이 추상 경로명이 가리키는 파일을 어플리케이션을 변경할 수 있을지 여부를 판정합니다.
int	compareTo (File pathname) 2 개의 추상 경로명을 어휘적으로 비교합니다.
boolean	createNewFile () 이 추상 경로명이 가리키는 빈 상태(empty)의 새로운 파일을 불가분 (atomic)에 생성합니다 (그 이름의 파일이 아직 존재하지 않는 경우만).
static File	createTempFile (String prefix, String suffix) 지정된 접두사와 접미사(suffix)을 파일명의 생성에 사용해, 디폴트의 임시 파일 디렉토리에 빈 상태(empty)의 파일을 생성합니다.
static File	createTempFile (String prefix, String suffix, File directory) 지정된 디렉토리에서 새로운 빈 상태(empty)의 파일을 생성해, 그 이름에는, 지정된 접두사 및 접미사(suffix)의 캐릭터 라인이 사용됩니다.
boolean	delete () 이 추상 경로명이 가리키는 파일 또는 디렉토리를 삭제합니다.
void	deleteOnExit () 이 추상 경로명이 가리키는 파일 또는 디렉토리가, 가상 머신이 종료했을 때에 삭제되도록(듯이) 요구합니다.
boolean	equals (Object obj) 이 추상 경로명이 지정된 객체와 동일한지 여부를 판정합니다.
boolean	exists () 이 추상 경로명이 가리키는 파일 또는 디렉토리가 존재할지 여부를 판정합니다.
File	getAbsoluteFile () 이 추상 경로명의 절대 형식을 돌려줍니다.
String	getAbsolutePath () 이 추상 경로명의 절대 경로명 스트링을 돌려줍니다.

2. File

2. File 메서드

메소드의 개요	
File getCanonicalFile ()	이 추상 경로명의 정규 형식을 돌려줍니다.
String getCanonicalPath ()	이 추상 경로명의 정규의 경로명 스트링을 돌려줍니다.
long getFreeSpace ()	이 추상 경로명으로 <u>지정되는</u> 파티션내에서 할당되지 않은 바이트수를 돌려줍니다.
String getName ()	이 추상 경로명이 가리키는 파일 또는 디렉토리의 이름을 돌려줍니다.
String getParent ()	이 추상 경로명의 부모 경로명 스트링을 돌려줍니다.
File getParentFile ()	이 추상 경로명의 부모의 추상 경로명을 돌려줍니다.
String getPath ()	이 추상 경로명을 경로명 스트링로 변환합니다.
long getTotalSpace ()	이 추상 경로명으로 <u>지정되는</u> 파티션의 사이즈를 돌려줍니다.
long getUsableSpace ()	이 추상 경로명으로 <u>지정되는</u> 파티션상에서, 이 가상 머신을 이용할 수 있는 바이트수를 돌려줍니다.
int hashCode ()	이 추상 경로명의 해시 코드를 계산합니다.
boolean isAbsolute ()	이 추상 경로명이 절대인가 어떤가를 판정합니다.
boolean isDirectory ()	이 추상 경로명이 가리키는 파일이 디렉토리일지 여덟지를 판정합니다.
boolean isFile ()	이 추상 경로명이 가리키는 파일이 보통 파일인가 어떤가를 판정합니다.
boolean isHidden ()	이 추상 경로명이 가리키는 파일이 숨겨진 파일인가 어떤가를 판정합니다.
long lastModified ()	이 추상 경로명이 가리키는 파일이 마지막에 변경된 시각을 돌려줍니다.

2. File

2. File 메서드

메소드의 개요	
long length ()	이 추상 경로명으로 지정되고 있는 파일의 길이를 돌려줍니다.
String [] list ()	이 추상 경로명이 가리키는 디렉토리에 있는 파일 및 디렉토리를 나타내는 캐릭터 라인의 배열을 돌려줍니다.
String [] list (FilenameFilter filter)	이 추상 경로명이 가리키는 디렉토리에 있는 파일 및 디렉토리 중(안)에서, 지정된 필터의 기준을 채우는 것의 캐릭터 라인의 배열을 돌려줍니다.
File [] listFiles ()	이 추상 경로명이 가리키는 디렉토리내의 파일을 나타내는 추상 경로명의 배열을 돌려줍니다.
File [] listFiles (FileFilter filter)	이 추상 경로명이 가리키는 디렉토리에 있는 파일 및 디렉토리 중(안)에서, 지정된 필터의 기준을 채우는 것의 추상 경로명의 배열을 돌려줍니다.
File [] listFiles (FilenameFilter filter)	이 추상 경로명이 가리키는 디렉토리에 있는 파일 및 디렉토리 중(안)에서, 지정된 필터의 기준을 채우는 것의 추상 경로명의 배열을 돌려줍니다.
static File[] listRoots ()	유효한 파일 시스템의 루트를 리스트 표시합니다.
boolean mkdir ()	이 추상 경로명이 가리키는 디렉토리를 생성합니다.
boolean mkdirs ()	이 추상 경로명이 가리키는 디렉토리를 생성합니다.
boolean renameTo (File dest)	이 추상 경로명이 가리키는 파일의 이름을 변경합니다.
boolean setExecutable (boolean executable)	이 추상 경로명에 소유자의 실행 권한을 설정하는 편리한 메소드입니다.
boolean setExecutable (boolean executable, boolean ownerOnly)	이 추상 경로명에 소유자 또는 전원의 실행 권한을 설정합니다.
boolean setLastModified (long time)	이 추상 경로명이 가리키는 파일 또는 디렉토리가 변경된 시각을 설정합니다.

2. File

2. File 메서드

메서드의 개요	
boolean	setReadable (boolean readable) 이 추상 경로명에 소유자의 읽기 권한을 설정하는 편리한 메소드입니다.
boolean	setReadable (boolean readable, boolean ownerOnly) 이 추상 경로명에 소유자 또는 전원의 읽기 권한을 설정합니다.
boolean	setReadOnly () 이 추상 경로명이 가리키는 파일 또는 디렉토리에 마크를 설정해, 읽기 조작만이 허가되도록(듯이) 합니다.
boolean	setWritable (boolean writable) 이 추상 경로명에 소유자의 쓰기 권한을 설정하는 편리한 메소드입니다.
boolean	setWritable (boolean writable, boolean ownerOnly) 이 추상 경로명에 소유자 또는 전원의 쓰기 권한을 설정합니다.
String	toString () 이 추상 경로명의 경로명 스트링을 돌려줍니다.
URI	toURI () 이 추상 경로명을 나타내는 file: URI 를 구축합니다.
URL	toURL () 추천 되고 있지 않습니다. 이 메소드에서는, URL 내에서 사용할 수 없는 이스케이프 문자는 자동적으로 변환할 수 없습니다. 새로운 코드의 추상 경로명을 URL 로 변환하려면, 우선 toURI 메소드를 사용해 URI 로 변환하고 나서, URI.toURL 메소드를 사용해 URL 로 변환하는 것을 추천합니다.

2. File

3. File 쓰기 예제

```
public class IOTest {  
    public static void main(String[] args) {  
        File f = new File("C:/test/cxx.txt");  
  
        if(!f.exists()){//해당 파일이 존재하는지 유뮤  
            try {  
                f.createNewFile();//파일 생성  
                //파일인지 아닌지 확인하는 문구  
                System.out.println(f.isFile());  
                //절대경로  
                System.out.println(f.getAbsolutePath());  
                //상대경로  
                System.out.println(f.getCanonicalPath());  
                System.out.println(f.getPath());  
                //파일 이름  
                System.out.println(f.getName());  
                //상위 Path  
                System.out.println(f.getParent());  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```



```
true  
C:\test\cxx.txt  
C:\test\cxx.txt  
C:\test\cxx.txt  
cxx.txt  
C:\test
```

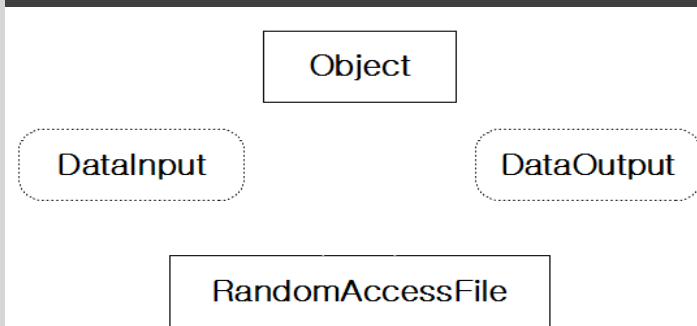

2. File

4. RandomAccessFile

- 하나의 스트림으로 파일에 입력과 출력을 모두 수행할 수 있는 스트림
- 다른 스트림들과 달리 Object의 자손이다.

* **DataInput**인터페이스의 메서드

```
boolean readBoolean()  
byte    readByte()  
int     readInt()  
void    readFully(byte[] b)  
String  readLine()  
...
```



* **DataOutput**인터페이스의 메서드

```
void write(byte[] b)  
void write(int b)  
void writeBoolean(boolean b)  
void writeInt(int v)  
void writeBytes(String s)  
...
```

생성자 / 메서드	설 명
RandomAccessFile(File file, String mode) RandomAccessFile(String fileName, String mode)	주어진 file에 읽기 또는 읽기와 쓰기를 하기 위한 RandomAccessFile인스턴스를 생성한다. mode에는 "r"과 "rw" 두 가지 값이 지정가능하다. "r" - 파일로부터 읽기(r)만을 수행할 때 "rw" - 파일에 읽기(r)와 쓰기(w)
long getFilePointer()	파일 포인터의 위치를 알려 준다.
long length()	파일의 크기를 얻을 수 있다.(단위 byte)
void seek(long pos)	파일 포인터의 위치를 변경한다. 위치는 파일의 첫 부분부터 pos크기만큼 떨어진 곳이다.(단위 byte)
void setLength(long newLength)	파일의 크기를 지정된 길이로 변경한다.(byte단위)
int skipBytes(int n)	지정된 수만큼의 byte를 건너뛴다.

3. 바이트 기반 스트림

1. InputStream과 OutputStream

- InputStream(바이트기반 입력스트림의 최고 조상)의 메서드

메서드명	설 명
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 알려 준다. mark()와 reset()기능을 지원하는 것은 선택적이므로, mark()와 reset()을 사용하기 전에 markSupported()를 호출해서 지원여부를 확인해야한다.
abstract int read()	1 byte를 읽어 온다(0~255사이의 값). 더 이상 읽어 올 데이터가 없으면 -1을 반환한다. abstract메서드라서 InputStream의 자손들은 자신의 상황에 알맞게 구현해야한다.
int read(byte[] b)	배열 b의 크기만큼 읽어서 배열을 채우고 읽어 온 데이터의 수를 반환한다. 반환하는 값은 항상 배열의 크기보다 작거나 같다.
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)만큼을 건너뛰다.

- OutputStream(바이트기반 출력스트림의 최고 조상)의 메서드

메서드명	설 명
void close()	입력소스를 닫음으로써 사용하고 있던 자원을 반환한다.
void flush()	스트림의 버퍼에 있는 모든 내용을 출력소스에 쓴다.
abstract void write(int b)	주어진 값을 출력소스에 쓴다.
void write(byte[] b)	주어진 배열 b에 저장된 모든 내용을 출력소스에 쓴다.
void write(byte[] b, int off, int len)	주어진 배열 b에 저장된 내용 중에서 off번째부터 len개 만큼만 읽어서 출력소스에 쓴다.

3. 바이트 기반 스트림

2. ByteArrayInputStream과 ByteArrayOutputStream

- 바이트배열(byte[])에 데이터를 입출력하는 바이트기반 스트림

```
public class ByteArrayTest {  
    public static void main(String[] args) {  
        byte[] insrc = {1,2,3,4,5,6,7,8,9};  
        byte[] outsrc = null;  
  
        ByteArrayInputStream bis = new ByteArrayInputStream(insrc);  
        ByteArrayOutputStream bos = new ByteArrayOutputStream();  
  
        int temp = 0;  
        while((temp=bis.read())!=-1){bos.write(temp);}   
  
        outsrc = bos.toByteArray();  
        System.out.println("bis : "+Arrays.toString(insrc));  
        System.out.println("bos : "+Arrays.toString(outsrc));  
    }  
}
```

(data = input.read()) != -1

① data = input.read() // read()를 호출한 반환값을 변수 data에 저장한다.

② data != -1 // data에 저장된 값이 -1이 아닌지 비교한다.

abstract int read()

1 byte를 읽어 온다(0~255사이의 값).
더 이상 읽어 올 데이터가 없으면 -1
을 반환한다.

Result)

bis : [1, 2, 3, 4, 5, 6, 7, 8, 9]

bos : [1, 2, 3, 4, 5, 6, 7, 8, 9]

3. 바이트 기반 스트림

3. **FileInputStream**과 **FileOutputStream**

- 파일(file)에 데이터를 입출력하는 바이트기반 스트림

FileInputStream 생성자

[FileInputStream](#) ([File](#) file)

파일 시스템으로 File 객체 file 에 의해 지정하는 실제의 파일에의 접속을 여는 것으로,FileInputStream 를 작성합니다.

[FileInputStream](#) ([FileDescriptor](#) fdObj)

파일 시스템의 실제의 파일에의 기존의 접속을 나타내는 파일 기술자 fdObj 를 사용해,FileInputStream 를 작성합니다.

[FileInputStream](#) ([String](#) name)

파일 시스템으로 경로명 name 에 의해 지정하는 실제의 파일에의 접속을 여는 것으로,FileInputStream 를 작성합니다.

FileOutputStream 생성자

[FileOutputStream](#) ([File](#) file)

지정된 File 객체에 의해 나타내지는 파일에 기입하기 위한 파일 출력 스트림을 작성합니다.

[FileOutputStream](#) ([File](#) file, boolean append)

지정된 File 객체에 의해 나타내지는 파일에 기입하기 위한 파일 출력 스트림을 작성합니다.

[FileOutputStream](#) ([FileDescriptor](#) fdObj)

파일 시스템의 실제의 파일에의 기존의 접속을 나타내는, 지정된 파일 기술자에게 기입하기 위한 출력 파일 스트림을 작성합니다.

[FileOutputStream](#) ([String](#) name)

지정된 이름의 파일에 기입하기 위한 파일 출력 스트림을 작성합니다.

[FileOutputStream](#) ([String](#) name, boolean append)

지정된 name 의 파일에 기입하기 위한 파일 출력 스트림을 작성합니다.

3. 바이트 기반 스트림

4-1. FileInputStream과 FileOutputStream 를 이용한 파일 읽기

```
public class FileRead {  
    public static void main(String[] args) {  
        File file = new File("c:/test/cac.txt");  
        try{  
            FileInputStream fis = new FileInputStream(file);  
            int temp = 0;  
            while ((temp=fis.read())!=-1){  
                System.out.print((char)temp);  
            }  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

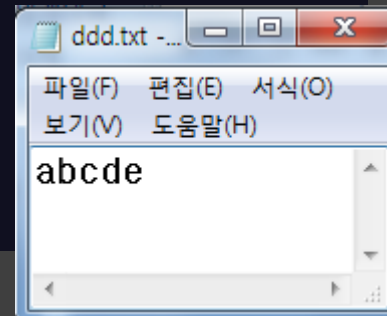
Result)

abcde

3. 바이트 기반 스트림

4-2. FileInputStream과 FileOutputStream 를 이용한 파일 쓰기

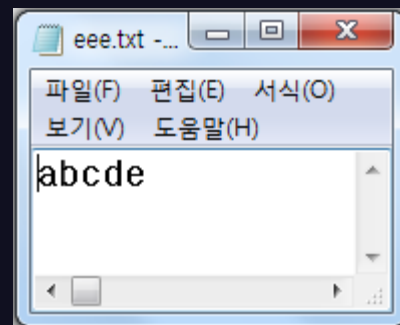
```
public class FileRead {  
    public static void main(String[] args) {  
        File file = new File("c:/test/cac.txt");  
        try{  
            FileInputStream fis = new FileInputStream(file);  
            int temp = 0;  
            while ((temp=fis.read())!=-1){  
                System.out.print((char)temp);  
            }  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```



3. 바이트 기반 스트림

4-3. FileInputStream과 FileOutputStream 를 이용한 파일 카피

```
public class FileCopyClass {  
    public static void main(String[] args) {  
        try{  
            FileInputStream fis = new FileInputStream("C:/test/ccc.txt");  
            FileOutputStream fos = new FileOutputStream("C:/test/eee.txt");  
            byte[] b = new byte[1024];  
            fis.read(b);  
            fos.write(b);  
            fis.close();  
            fos.close();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```



4. 바이트기반 보조스트림

1. FilterInputStream과 FilterOutputStream

- 모든 바이트기반 보조스트림의 최고조상
- 모든 보조스트림은 자체적으로 입출력을 수행할 수 없다.
- 생성자 인자로 반드시 IO 바이트기반 스트림객체를 받아서 처리한다.

protected	FilterInputStream (InputStream in) 나중에 사용할 수 있도록(듯이) 인수 in 를 this.in 필드에 할당하는 것에 의해,FilterInputStream 를 작성합니다.
public	FilterOutputStream (OutputStream out) 지정된 기본이 되는 출력 스트림의 상위에 출력 스트림 필터를 작성합니다.

FilterInputStream의 자손 - BufferedInputStream, DataInputStream, PushbackInputStream 등
FilterOutputStream의 자손 - BufferedOutputStream, DataOutputStream, PrintStream 등

4. 바이트기반 보조스트림

2. BufferedInputStream/BufferedOutputStream

- 입출력 효율을 높이기 위해 사용하는 보조스트림
- 해당 입출력 대상을 버퍼단위로 제어할 수 있도록 해준다

생성자의 개요

BufferedInputStream (InputStream in)

BufferedInputStream 를 작성해, 그 인수인 입력 스트림 in 를 나중에 사용할 수 있도록(듯이) 보존합니다.

BufferedInputStream (InputStream in, int size)

지정된 버퍼 사이즈를 가지는 BufferedInputStream 를 작성해, 그 인수인 입력 스트림 in 를 나중에 사용할 수 있도록(듯이) 보존합니다.

BufferedOutputStream (OutputStream out)

지정된 기본이 되는 출력 스트림에 데이터를 기입하기 위한 버퍼링 된 출력 스트림을 작성합니다.

BufferedOutputStream (OutputStream out, int size)

지정된 기본이 되는 출력 스트림에 데이터를 기입하기 위한 버퍼링 된 출력 스트림을, 지정된 버퍼 사이즈로 작성합니다.

4. 바이트기반 보조스트림

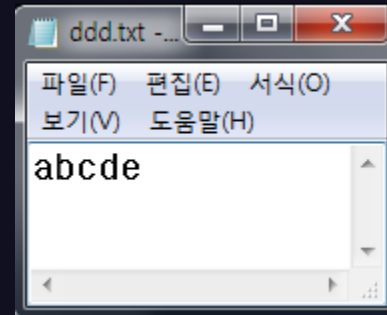
2. BufferedInputStream/BufferedOutputStream 주요 메서드

BufferedInputStream 메소드의 개요	
int available ()	이 입력 스트림의 메소드의 다음의 호출에 의해, 블록 하지 않고 이 입력 스트림로부터 읽어들이 수가 있는 (또는 스킵 할 수 있다) 추정 바이트수를 돌려줍니다.
void close ()	이 입력 스트림을 닫아, 그 스트림에 관련하는 모든 system resource를 해제합니다.
void mark (int readlimit)	이 입력 스트림의 현재 위치에 마크를 설정합니다.
boolean markSupported ()	입력 스트림이 mark 와 reset 메소드를 지원하고 있을지 여부를 판정합니다.
abstract int read ()	입력 스트림로부터 데이터의 다음의 바이트를 읽어들이합니다.
int read (byte[] b)	입력 스트림로부터 수바이트를 읽어들이, 그것을 버퍼 배열 b 에 포함합니다.
int read (byte[] b, int off, int len)	최대 len 바이트까지의 데이터를, 입력 스트림로부터 바이트 배열에 읽어들이합니다.
void reset ()	이 스트림의 위치를, 입력 스트림로 마지막에 mark 메소드가 불러 갔을 때의 마크 위치에 재설정합니다.
long skip (long n)	이 입력 스트림로부터 n 바이트분을 스킵 및 파기합니다.
BufferedOutputStream 메소드의 개요	
void flush ()	버퍼링 된 출력 스트림을 플래시 합니다.
void write (byte[] b, int off, int len)	지정된 바이트 배열의 오프셋(offset) off 로부터 시작되는 len 바이트를, 버퍼링 된 출력 스트림에 기입합니다.
void write (int b)	지정된 바이트수를 버퍼링 된 출력 스트림에 기입합니다.

4. 바이트기반 보조스트림

3-1. BufferedInputStream/BufferedOutputStream 쓰기

```
public class FileWriteByBuffer {  
    public static void main(String[] args) {  
        File file = new File("c:/test/ddd.txt");  
        //아스키코드로 abcde를 찍는다  
        byte[] b = {97,98,99,100,101};  
        try {  
            FileOutputStream fos = new FileOutputStream(file);  
            BufferedOutputStream bos = new BufferedOutputStream(fos);  
            bos.write(b);  
            bos.close();  
            fos.close();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```



4. 바이트기반 보조스트림

3-1. BufferedInputStream/BufferedOutputStream 읽기

```
public class FileReadByBuffer {  
    public static void main(String[] args) {  
        try{  
            FileInputStream fis = new FileInputStream("c:/test/cac.txt");  
            BufferedInputStream bis = new BufferedInputStream(fis);  
            int temp = 0;  
            while ((temp=bis.read())!=-1){  
                System.out.print((char)temp);  
            }  
            bis.close();  
            fis.close();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

abcde

4. 바이트기반 보조스트림

3-1. BufferedInputStream/BufferedOutputStream 읽기

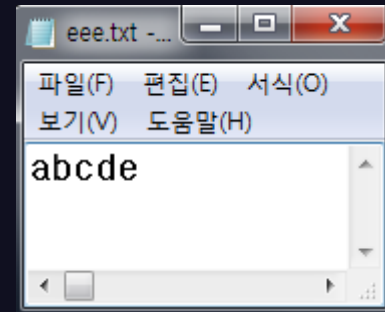
```
public class FileReadByBuffer {  
    public static void main(String[] args) {  
        try{  
            FileInputStream fis = new FileInputStream("c:/test/ccc.txt");  
            BufferedInputStream bis = new BufferedInputStream(fis);  
            int temp = 0;  
            while ((temp=bis.read())!=-1){  
                System.out.print((char)temp);  
            }  
            bis.close();  
            fis.close();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

abcde

4. 바이트기반 보조스트림

3-1. BufferedInputStream/BufferedOutputStream 복사

```
public class FileCopyByBuffer {  
    public static void main(String[] args) {  
        try{  
            BufferedInputStream bis = new BufferedInputStream(new FileInputStream("C:/test/ccc.txt"));  
            BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("C:/test/eee.txt"));  
            byte[] temp = new byte[1024];  
            while(bis.read(temp) != -1){  
                bos.write(temp);  
            }  
            bis.close();  
            bos.close();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```



4. 바이트기반 보조스트림

4. DataInputStream과 DataOutputStream

- 기본형 단위로 읽고 쓰는 보조스트림
- 각 자료형의 크기가 다르므로 출력할 때와 입력할 때 순서에 주의

메서드 / 생성자	설 명
DataInputStream(InputStream in)	주어진 InputStream인스턴스를 기반스트림으로 하는 DataInputStream인스턴스를 생성한다.
boolean readBoolean() byte readByte() char readChar() short readShort() int readInt() long readLong() float readFloat() double readDouble()	각 자료형에 알맞은 값들을 읽어 온다. 더 이상 읽을 값이 없으면 EOFException을 발생시킨다.
String readUTF()	UTF형식으로 쓰여진 문자를 읽는다. 더 이상 읽을 값이 없으면 EOFException을 발생시킨다.
int skipBytes(int n)	현재 읽고 있는 위치에서 지정된 숫자(n) 만큼을 건너 뛴다.

메서드 / 생성자	설 명
DataOutputStream(OutputStream out)	주어진 OutputStream인스턴스를 기반스트림으로 하는 DataOutputStream인스턴스를 생성한다.
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)	UTF형식으로 문자를 출력한다.
void writeChars(String s)	주어진 문자열을 출력한다. writeChar(char c)메서드를 여러 번 호출한 결과와 같다.
int size()	지금까지 DataOutputStream에 쓰여진 byte의 수를 알려 준다.

4. 바이트기반 보조스트림

5. SequenceInputStream

- 여러 입력스트림을 연결해서 하나의 스트림처럼 다룰 수 있게 해준다.

메서드 / 생성자	설 명
SequenceInputStream(Enumeration e)	Enumeration에 저장된 순서대로 입력스트림을 하나의 스트림으로 연결한다.
SequenceInputStream(InputStream s1, InputStream s2)	두 개의 입력스트림을 하나로 연결한다.

```
public class FileRoadBySeq {
    public static void main(String[] args) {
        try {
            FileInputStream fis1 = new FileInputStream("C:/test/ccc.txt");
            FileInputStream fis2 = new FileInputStream("C:/test/ddd.txt");
            SequenceInputStream sis = new SequenceInputStream(fis1, fis2);
            BufferedInputStream bis = new BufferedInputStream(sis);
            int b = 0;
            while((b=bis.read())!=-1){
                System.out.print((char)b);
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

abcdeabcde

4. 바이트기반 보조스트림

6. PrintStream

- 데이터를 다양한 형식의 문자로 출력하는 기능을 제공하는 보조스트림
- System.out과 System.err이 PrintStream이다.
- PrintStream보다 PrintWriter를 사용할 것을 권장한다.

생성자 / 메서드		설 명
PrintStream(File file) PrintStream(File file, String csn) PrintStream(OutputStream out) PrintStream(OutputStream out,boolean autoFlush) PrintStream(OutputStream out,boolean autoFlush, String encoding) PrintStream(String fileName) PrintStream(String fileName, String csn)		지정된 출력스트림을 기반으로 하는 PrintStream 인스턴스를 생성한다. autoFlush의 값을 true로 하면 println메서드가 호출되거나 개행문자가 출력될 때 자동으로 flush된다. 기본값은 false이다.
boolean checkError()		스트림을 flush하고 에러가 발생했는지를 알려 준다.
void print(boolean b) void print(char c) void print(char[] c) void print(double d) void print(float f) void print(int i) void print(long l) void print(Object o) void print(String s)	void println(boolean b) void println(char c) void println(char[] c) void println(double d) void println(float f) void println(int i) void println(long l) void println(Object o) void println(String s)	인자로 주어진 값을 출력소스에 문자로 출력한다. println메서드는 출력 후 줄바꿈을 하고, print메서드는 줄을 바꾸지 않는다.
void println()		줄바꿈 문자(line separator)를 출력함으로써 줄을 바꾼다.
PrintStream printf(String format, Object... args)		정형화된(formatted) 출력을 가능하게 한다.
protected void setError()		작업 중에 오류가 발생했음을 알린다.(setError()를 호출한 후에, checkError()를 호출하면 true를 반환한다.)

4. 바이트기반 보조스트림

7. PrintStream 특수문자 및 연산자

format	설 명	결 과(int i=65)
%d	10진수(decimal integer)	65
%o	8진수(octal integer)	101
%x	16진수(hexadecimal integer)	41
%c	문자	A
%s	문자열	65
%5d	5자리 숫자. 빈자리는 공백으로 채운다.	65
%-5d	5자리 숫자. 빈자리는 공백으로 채운다.(왼쪽 정렬)	65
%05d	5자리 숫자. 빈자리는 0으로 채운다.	00065

format	설 명	결 과
%s	문자열(string)	ABC
%5s	5자리 문자열. 빈자리는 공백으로 채운다.	ABC
%-5s	5자리 문자열. 빈자리는 공백으로 채운다.(왼쪽 정렬)	ABC

format	설 명
\t	탭(tab)
\n	줄바꿈 문자(new line)
%%	%

format	설 명	결과
%e	지수형태표현(exponent)	1.234568e+03
%f	10진수(decimal float)	1234.56789
%3.1f	출력될 자리수를 최소 3자리(소수점포함), 소수점 이하 1자리(2번째 자리에서 반올림)	1234.6
%8.1f	소수점이상 최소 6자리, 소수점 이하 1자리. 출력될 자리수를 최소 8자리(소수점포함)를 확보한다. 빈자리는 공백으로 채워진다.(오른쪽 정렬)	1234.6
%08.1f	소수점이상 최소 6자리, 소수점 이하 1자리. 출력될 자리수를 최소 8자리(소수점포함)를 확보한다. 빈자리는 0으로 채워진다.	001234.6
%-8.1f	소수점이상 최소 6자리, 소수점 이하 1자리. 출력될 자리수를 최소 8자리(소수점포함)를 확보한다. 빈자리는 공백으로 채워진다.(왼쪽 정렬)	1234.6

format	설 명	결 과
%tR %tH:%tM	시분(24시간)	21:05 21:05
%tT %tH:%tM:%tS	시분초(24시간)	21:05:33 21:05:33
%tD %tm/%td/%ty	연월일	02/16/07 02/16/07
%tF %tY-%tm-%td	연월일	2007-02-16 2007-02-16

5. 문자기반 입력스트림

1. Reader와 Writer

▶ Reader(문자기반 입력스트림의 최고 조상)의 메서드

메서드	설 명
abstract void close()	입력스트림을 다음으로써 사용하고 있던 자원을 반환한다.
void mark(int readlimit)	현재위치를 표시해 놓는다. 후에 reset()에 의해서 표시해 놓은 위치로 다시 돌아갈 수 있다.
boolean markSupported()	mark()와 reset()을 지원하는지를 알려 준다.
int read()	입력소스로부터 하나의 문자를 읽어 온다. char의 범위인 0~65535범위의 정수를 반환하며, 입력스트림의 마지막 데이터에 도달하면, -1을 반환한다.
int read(char[] c);	입력소스로부터 매개변수로 주어진 배열 c의 크기만큼 읽어서 배열 c에 저장한다. 읽어 온 데이터의 개수 또는 -1을 반환한다.
abstract int read(char[] c, int off, int len)	입력소스로부터 최대 len개의 문자를 읽어서, 배열 c의 지정된 위치(off)부터 읽은 만큼 저장한다. 읽어 온 데이터의 개수 또는 -1을 반환한다.
boolean ready()	입력소스로부터 데이터를 읽을 준비가 되어있는지 알려 준다.
void reset()	입력소스에서의 위치를 마지막으로 mark()가 호출되었던 위치로 되돌린다.
long skip(long n)	현재 위치에서 주어진 문자 수(n)만큼을 건너뛴다.

▶ Writer(문자기반 출력스트림의 최고 조상)의 메서드

메서드	설 명
abstract void close()	출력스트림을 다음으로써 사용하고 있던 자원을 반환한다.
abstract void flush()	스트림의 버퍼에 있는 모든 내용을 출력소스에 쓴다.(버퍼가 있는 스트림에만 해당됨)
void write(int b)	주어진 값을 출력소스에 쓴다.
void write(char[] c)	주어진 배열 c에 저장된 모든 내용을 출력소스에 쓴다.
abstract void write(char[] c, int off, int len)	주어진 배열 c에 저장된 내용 중에서 off번째부터 len길이만큼만 출력소스에 쓴다.
void write(String str)	주어진 문자열(str)을 출력소스에 쓴다.
void write(String str, int off, int len)	주어진 문자열(str)의 일부를 출력소스에 쓴다.(off번째 문자부터 len개 만큼의 문자열)

5. 문자기반 입력스트림

2. FileReader와 FileWriter

- 문자기반의 파일 입출력. 텍스트 파일의 입출력에 사용한다.

```
public class CompareByFileRead {
    public static void main(String[] args) {
        try{
            FileInputStream fis = new FileInputStream("C:/test/ccc.txt");
            FileReader fr = new FileReader("C:/test/ddd.txt");
            int b=0;
            while((b=fis.read())!=-1){
                System.out.print((char)b);
            }
            System.out.println();
            int c=0;
            while((c=fr.read())!=-1){
                System.out.print((char)c);
            }
            fis.close();
            fr.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

abcde%?3???%/%? ?
abcde안녕하세요

5. 문자기반 입력스트림

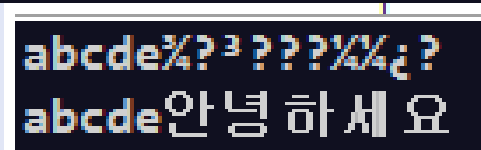
3. StringReader와 StringWriter

- 메모리에 저장된 문자 기반의 문자열을 입출력 시키기 위해 사용한다.

StringBuffer getBuffer() : StringWriter에 출력한 데이터가 저장된 StringBuffer를 반환한다.

String toString() : StringWriter에 출력된 (StringBuffer에 저장된) 문자열을 반환한다.

```
public class CompareByFileRead {
    public static void main(String[] args) {
        try{
            FileInputStream fis = new FileInputStream("C:/test/ccc.txt");
            FileReader fr = new FileReader("C:/test/ddd.txt");
            int b=0;
            while((b=fis.read())!=-1){
                System.out.print((char)b);
            }
            System.out.println();
            int c=0;
            while((c=fr.read())!=-1){
                System.out.print((char)c);
            }
            fis.close();
            fr.close();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```



6. 문자기반 보조스트림

1. BufferedReader와 BufferedWriter

- 입출력 효율을 높이기 위해 버퍼를 사용하는 보조스트림.
- 라인(line)단위의 입출력을 용이하게 하는 기능을 갖고있다.

`String readLine()` - 한 라인을 읽어온다. (BufferedReader의 메서드)

`void newLine()` - '라인 구분자(개행문자)'를 출력한다. (BufferedWriter의 메서드)

```
public class BufferedReadWrite {  
  
    public static void main(String[] args) {  
        try{  
            BufferedReader br = new BufferedReader(new FileReader("C:/test/ccc.txt"));  
            BufferedWriter bw = new BufferedWriter(new FileWriter("C:/test/ddd.txt"));  
            String line = "";  
            while((line = br.readLine())!=null){  
                bw.write(line);  
                System.out.println(line);  
            }  
            br.close();  
            bw.close();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

안녕하세요 반갑습니다

6. 문자기반 보조스트림

2. InputStreamReader와 OutputStreamWriter

- 바이트기반스트림을 문자기반스트림처럼 쓸 수 있게 해준다.
- 인코딩(encoding)을 변환하여 입출력할 수 있게 해준다.

생성자 / 메서드	설 명
InputStreamReader(InputStream in)	OS에서 사용하는 기본 인코딩의 문자로 변환하는 InputStreamReader를 생성한다.
InputStreamReader(InputStream in, String encoding)	지정된 인코딩을 사용하는 InputStreamReader를 생성한다.
String getEncoding()	InputStreamReader의 인코딩을 알려 준다.

생성자 / 메서드	설 명
OutputStreamWriter(OutputStream in)	OS에서 사용하는 기본 인코딩의 문자로 변환하는 OutputStreamWriter를 생성한다.
OutputStreamWriter(OutputStream in, String encoding)	지정된 인코딩을 사용하는 OutputStreamWriter를 생성한다.
String getEncoding()	OutputStreamWriter의 인코딩을 알려 준다.

6. 문자기반 보조스트림

2. InputStreamReader와 OutputStreamWriter

```
public class IncodingBufferedToString {  
    public static void main(String[] args) {  
        try{  
            BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream("C:/test/ccc.txt")));  
            BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(new FileOutputStream("C:/test/eee.txt")));  
            String line = "";  
            while((line = br.readLine())!=null){  
                bw.write(line);  
                System.out.println(line);  
            }  
            br.close();  
            bw.close();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

안녕하세요 반갑습니다

7. 직렬화(serialization)

· 1. 직렬화란?

- 객체를 '연속적인 데이터'로 변환하는 것. 반대과정은 '역직렬화'라고 함
- 객체의 인스턴스 변수들의 값을 일렬로 나열하는 것
- 객체를 저장하기 위해서는 객체를 직렬화해야 한다.
- 객체를 저장한다는 것은 객체의 모든 인스턴스변수의 값을 저장하는 것



7. 직렬화(serialization)

2. **ObjectInputStream, ObjectOutputStream**

- 객체를 직렬화하여 입출력할 수 있게 해주는 보조스트림

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

- 객체를 파일에 저장하는 방법

```
FileOutputStream fos = new FileOutputStream("objectfile.ser");  
ObjectOutputStream out = new ObjectOutputStream(fos);  
  
out.writeObject(new UserInfo());
```

- 파일에 저장된 객체를 다시 읽어오는 방법

```
FileInputStream fis = new FileInputStream("objectfile.ser");  
ObjectInputStream in= new ObjectInputStream(fis);  
  
UserInfo info = (UserInfo)in.readObject();
```

7. 직렬화(serialization)

3. 직렬화 클래스 구현

- java.io.Serializable을 상속하여 구현해야만 직렬화가 가능하다.

```
public class UserInfo {  
    String name;  
    String password;  
    int age;  
}
```

```
public class UserInfo  
    implements java.io.Serializable {  
    String name;  
    String password;  
    int age;  
}
```

```
public interface Serializable { }
```

- 제어자 transient가 붙은 인스턴스변수는 직렬화 대상에서 제외된다.

```
public class UserInfo implements Serializable {  
    String name;  
    transient String password; // 직렬화 대상에서 제외된다.  
    int age;  
}
```

- Serializable을 구현하지 않은 클래스의 인스턴스도 직렬화 대상에서 제외

```
public class UserInfo implements Serializable {  
    String name;  
    transient String password;  
    int age;  
  
    Object obj = new Object(); // Object객체는 직렬화할 수 없다.  
}
```

7. 직렬화(serialization)

3. 직렬화 클래스 구현

- Serializable을 구현하지 않은 조상의 멤버들은 직렬화 대상에서 제외

```
public class SuperUserInfo {  
    String name;      // 직렬화되지 않는다.  
    String password; // 직렬화되지 않는다.  
}  
  
public class UserInfo extends SuperUserInfo implements Serializable {  
    int age;  
}
```

- readObject()와 writeObject()를 오버라이딩하면 직렬화를 마음대로...

```
private void writeObject(ObjectOutputStream out)  
    throws IOException {  
    out.writeUTF(name);  
    out.writeUTF(password);  
    out.defaultWriteObject();  
}
```

```
private void readObject(ObjectInputStream in)  
    throws IOException, ClassNotFoundException {  
    name = in.readUTF();  
    password = in.readUTF();  
    in.defaultReadObject();  
}
```

7. 직렬화(serialization)

3. 직렬화 클래스 버전 관리

- 직렬화했을 때와 역직렬화 했을 때의 클래스가 같은지 확인할 필요가 있다.

```
java.io.InvalidClassException: UserInfo; local class incompatible: stream
classdesc serialVersionUID = 6953673583338942489, local class
serialVersionUID = -6256164443556992367
...
```

- 직렬화할 때, 클래스의 버전(serialVersionUID)을 자동계산해서 저장한다.
- 클래스의 버전을 수동으로 관리하려면, 클래스 내에 정의해야 한다.

```
class MyData implements java.io.Serializable {
    static final long serialVersionUID = 3518731767529258119L;
    int value1;
}
```

- serialver.exe는 클래스의 serialVersionUID를 자동생성해준다.

```
C:\jdk1.5\work\ch14>serialver MyData
MyData:    static final long serialVersionUID = 3518731767529258119L;
```

8. JAVA NIO

1. 기존 IO의 단점

- 기존 IO는 OS의 커널로 직접적인 접근이 불가능함
- Blocking(블록형) IO여서 많은 오버헤드가 생기게 되어 버퍼를 잡아먹음
- Garbage Collector의 영향으로 인한 오버헤드의 문제도 생김
- 즉 CPU가 관여하게 되면서 퍼포먼스의 문제가 발생한다.
- CPU를 사용하지 않고 커널영역의 DMA를 사용해서 파일을 전송한다면 CPU의 자원을 낭비하지 않고 효율적으로 복사가 가능하다.

8. JAVA NIO

2. NIO란?

- 기존 IO의 오버헤드를 줄이기 위해 나온 IO
- NIO에서 System Call을 간접적으로 사용하게 해준다.
- NIO는 불특정 다수의 클라이언트 연결 또는 멀티 파일들을 넘 블로킹이나 비동기로 처리할 수 있기 때문에 과도한 스레드 생성을 피하고 스레드를 효과적으로 재사용한다는 점에서 큰 장점이 있다.

8. JAVA NIO

3. 경로 정의(path)

- 기존 IO의 java.io.File 클래스에 대응되는 인터페이스

사용 예

```
Path path = Paths.get(String first, String...more)
```

```
Path path = Paths.get(URI uri);
```


8. JAVA NIO

3. 경로 정의(path)

리턴타입	메소드(매개변수)	설명
int	compareTo(Path other)	파일경로가 동일하면 0을 리턴 상위 경로면 음수, 하위 경로면 양수를 리턴, 음수와 양수 값의 차이는 문자열의 수
Path	getFileName()	부모 경로를 제외한 파일 또는 디렉토리 이름만 가진 Path 리턴
FileSystem	getFileSystem()	FileSystem 객체 리턴
Path	getName(int index)	C:/Temp/dir/file.txt일 경우 index가 0이면 "Temp"의 Path 객체 리턴 index가 1이면 "dir"의 Path 객체 리턴 index가 2이면 "file.txt"의 Path 객체 리턴
int	getNameCount()	중첩 경로 수, C:/Temp/dir/file.txt일 경우 3 리턴
Path	getParent()	바로 위 부모 폴더의 Path 리턴
Path	getRoot()	루트 디렉토리의 Path 리턴

8. JAVA NIO

3. 경로 정의(path)

리턴타입	메소드(매개변수)	설명
Iterator<Path>	iterator()	경로에 있는 모든 디렉토리와 파일을 Path 객체로 생성하고 반복자를 리턴
Path	normalize()	상대 경로로 표기할 때 불필요한 요소를 제거 C:/Temp/dir1/.../dir2/file.txt -> C:/Temp/dir2/file.txt
WatchKey	register(...)	WatchService를 등록
File	toFile()	java.io.File 객체로 리턴
String	toString()	파일 경로를 문자열로 리턴
URI	toURI()	파일 경로를 URI 객체로 리턴

8. JAVA NIO

3. 경로 정의(path)

- 예제

```
public class PathExample {  
    public static void main(String[] args) throws Exception {  
        Path path = Paths.get("src/sec02/exam01_path/PathExample.java");  
        System.out.println("[파일명] " + path.getFileName());  
        System.out.println("[부모 디렉토리명]: " + path.getParent().getFileName());  
        System.out.println("중첩 경로수: " + path.getNameCount());  
  
        System.out.println();  
        for(int i=0; i<path.getNameCount(); i++) {  
            System.out.println(path.getName(i));  
        }  
  
        System.out.println();  
        Iterator<Path> iterator = path.iterator();  
        while(iterator.hasNext()) {  
            Path temp = iterator.next();  
            System.out.println(temp.getFileName());  
        }  
    }  
}
```

8. JAVA NIO

4. 경로 정의(FileSystem)

- 운영체제의 파일 시스템은 FileSystem 인터페이스를 통해 접근할 수 있다.
- FileSystem 구현객체는 FileSystem의 정적메소드인 getDefault()로 얻을 수 있다

사용 예

```
FileSystem fileSystem = FileSystem.getDefault();
```

- FileSystem은 다음과 같은 메소드를 제공한다.

리턴타입	메소드(매개변수)	설명
Iterator<FileStore>	getFileStores()	드라이버 정보를 가진 FileStore 객체들을 리턴
Iterable<Path>	getRootDirectories()	루트 디렉토리 정보를 가진 Path 객체들을 리턴
String	getSeparator()	디렉토리 구분자 리턴

8. JAVA NIO

4. 경로 정의(FileSystem)

- FileStore는 드라이버를 표현한 객체로 다음과 같은 메소드를 제공한다.

리턴타입	메소드(매개변수)	설명
long	getTotalSpace()	드라이버 전체 공간 크기(단위: 바이트) 리턴
long	getUnallocatedSpace()	할당되지 않은 공간 크기(단위: 바이트) 리턴
long	getUseableSpace()	사용 가능한 공간 크기, getUnallocatedSpace()와 동일한 값
boolean	isReadOnly()	읽기 전용 여부
String	name()	드라이버명 리턴
String	type()	파일 시스템 종류

8. JAVA NIO

4. 경로 정의(FileSystem)

- FileSystem 예제.

```
public class FileSystemExample {  
    public static void main(String[] args) throws Exception {  
        FileSystem fileSystem = FileSystems.getDefault();  
        for(FileStore store : fileSystem.getFileStores()) {  
            System.out.println("드라이버명: " + store.name());  
            System.out.println("파일시스템: " + store.type());  
            System.out.println("전체 공간: \t\t" + store.getTotalSpace() + " 바이트");  
            System.out.println("사용 중인 공간: \t" +  
                (store.getTotalSpace() - store.getUnallocatedSpace()) + " 바이트");  
            System.out.println("사용 가능한 공간: \t" + store.getUsableSpace() + " 바이트");  
            System.out.println();  
        }  
  
        System.out.println("파일 구분자: " + fileSystem.getSeparator());  
        System.out.println();  
  
        for(Path path : fileSystem.getRootDirectories()) {  
            System.out.println(path.toString());  
        }  
    }  
}
```

8. JAVA NIO

5. 파일 속성 읽기 및 파일, 디렉토리 생성/삭제

- java.nio.file.Files 클래스는 파일과 디렉토리의 생성 및 삭제, 그리고 이들의 속성을 읽는 메소드를 제공하고 있다.

리턴타입	메소드(매개변수)	설명
long 또는 Path	copy(...)	복사
Path	createDirectories(...)	모든 부모 디렉토리 생성
Path	createDirectory(...)	경로의 마지막 디렉토리만 생성
Path	createFile(...)	파일 생성
void	delete(...)	삭제
boolean	deleteIfExists(...)	존재한다면 삭제
boolean	exist(...)	존재여부
FileStore	getFileStore(...)	파일이 위치한 FileStore(드라이브)리턴
FileTime	getLastModifiedTime(...)	마지막 수정시간을 리턴
UserPrincipal	getOwner(...)	소유자 정보를 리턴
boolean	isDirectory(...)	디렉토리인지 여부

8. JAVA NIO

5. 파일 속성 읽기 및 파일, 디렉토리 생성/삭제

리턴타입	메소드(매개변수)	설명
boolean	isExecutable(...)	실행 가능 여부
boolean	isHidden(...)	숨김 여부
boolean	isReadable(...)	읽기 가능 여부
boolean	isRegularFile(...)	일반 파일인지 여부
boolean	isSameFile(...)	같은 파일인지 여부
boolean	isWritable(...)	쓰기가 가능 여부
Path	move(...)	파일 이동
BufferedReader	newBufferedReader(...)	텍스트 파일을 읽는 BufferedReader 리턴
BufferedReader	newBufferedWriter(...)	텍스트 파일에 쓰는 BufferedWriter 리턴
SeekableByteChannel	newByteChannel(...)	파일에 읽고 쓰는 바이트 채널을 리턴
DirectoryStream<Path>	newDirectoryStream(...)	디렉토리의 모든 내용을 스트림으로 리턴
InputStream	newInputStream(...)	파일의 InputStream 리턴
OutputStream	newOutputStream(...)	파일의 OutputStream 리턴

8. JAVA NIO

5. 파일 속성 읽기 및 파일, 디렉토리 생성/삭제

리턴타입	메소드(매개변수)	설명
boolean	notExists(...)	존재하지 않는지 여부
String	probeContentType(...)	파일의 MiME 타입을 리턴
byte[]	readAllBytes(...)	파일의 모든 바이트를 읽고 배열로 리턴
List<String>	readAllLines(...)	텍스트 파일의 모든 라인을 읽고 리턴
long	size(...)	파일의 크기 리턴
Path	write(...)	파일의 바이트나 문자열을 저장

8. JAVA NIO

5. 파일 속성 읽기 및 파일, 디렉토리 생성/삭제

- 예제(1).

```
public class FileExample {  
    public static void main(String[] args) throws Exception {  
        Path path = Paths.get("src/sec02/exam03_file_directory/FileExample.java");  
        System.out.println("디렉토리 여부: " + Files.isDirectory(path));  
        System.out.println("파일 여부: " + Files.isRegularFile(path));  
        System.out.println("마지막 수정 시간: " + Files.getLastModifiedTime(path));  
        System.out.println("파일 크기: " + Files.size(path));  
        System.out.println("소유자: " + Files.getOwner(path).getName());  
        System.out.println("숨김 파일 여부: " + Files.isHidden(path));  
        System.out.println("읽기 가능 여부: " + Files.isReadable(path));  
        System.out.println("쓰기 가능 여부: " + Files.isWritable(path));  
    }  
}
```

8. JAVA NIO

5. 파일 속성 읽기 및 파일, 디렉토리 생성/삭제

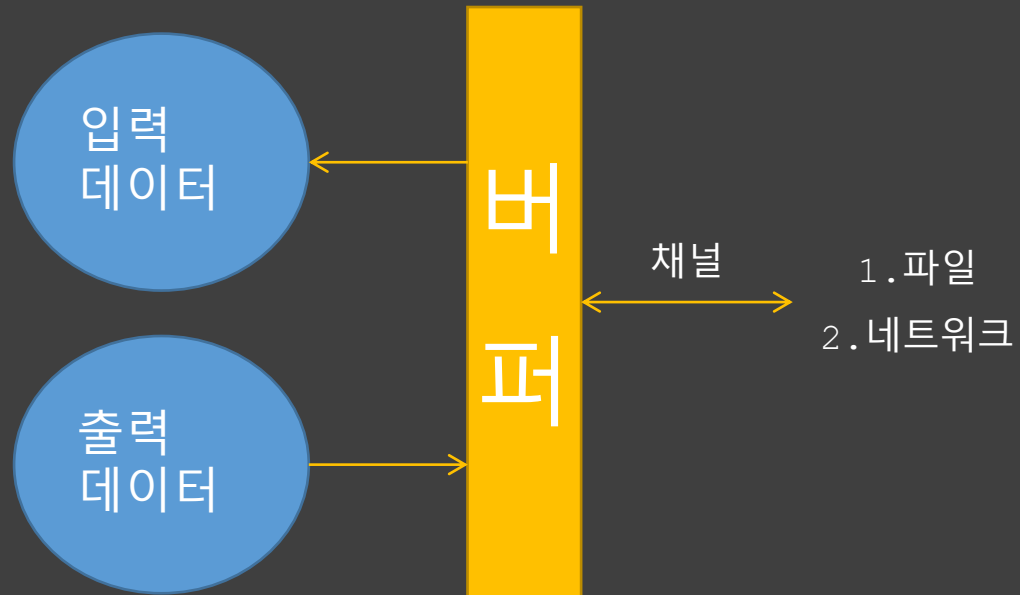
- 예제(2).

```
public class DirectoryExample {  
    public static void main(String[] args) throws Exception {  
        Path path1 = Paths.get("C:/Temp/dir/subdir");  
        Path path2 = Paths.get("C:/Temp/file.txt");  
  
        if(Files.notExists(path1)) {Files.createDirectories(path1);}  
        if(Files.notExists(path2)) {Files.createFile(path2);}  
  
        Path path3 = Paths.get("C:/Temp");  
        DirectoryStream<Path> directoryStream = Files.newDirectoryStream(path3);  
        for(Path path : directoryStream) {  
            if(Files.isDirectory(path)) {  
                System.out.println("[디렉토리] " + path.getFileName());  
            } else {  
                System.out.println("[파일] " + path.getFileName()  
                    + " (크기:" + Files.size(path) + ")");  
            }  
        }  
    }  
}
```

8. JAVA NIO

6. 버퍼

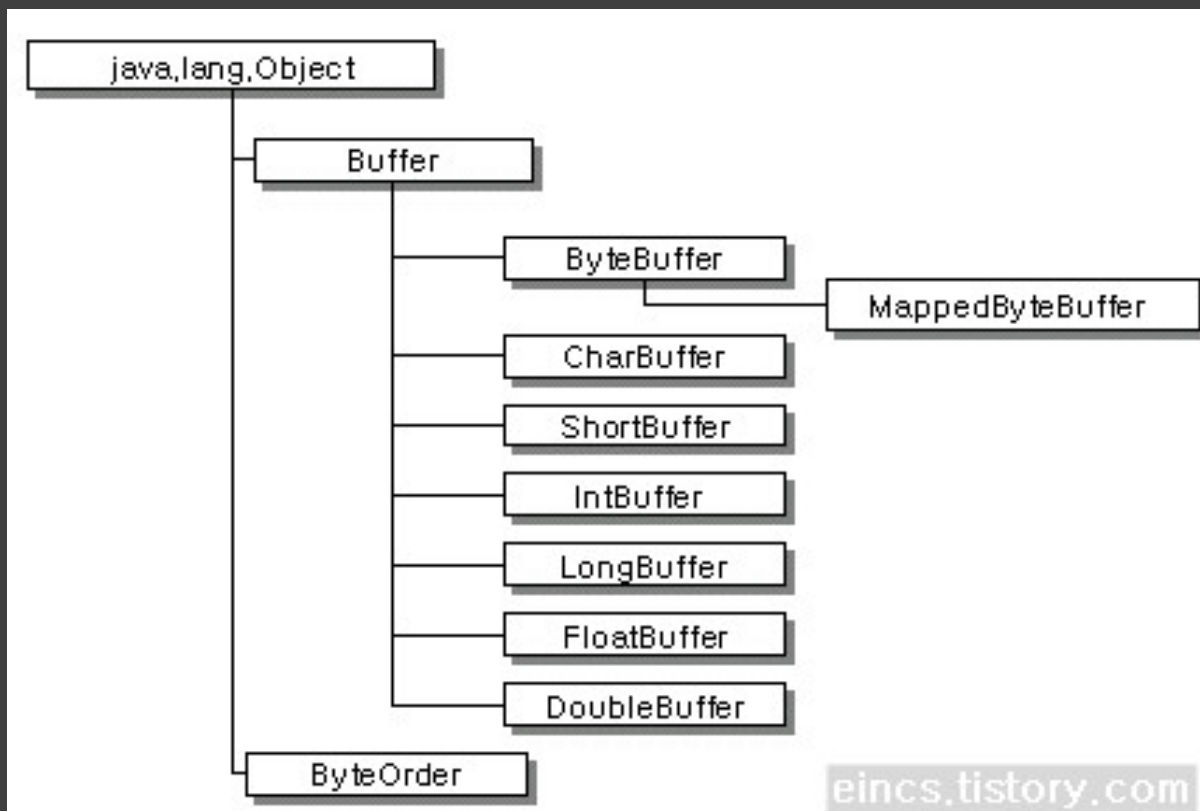
- NIO에서는 데이터를 입출력하기 위해 항상 버퍼를 사용해야 한다. 버퍼는 읽고 쓰기가 가능한 메모리 배열이다.



8. JAVA NIO

6. 버퍼

- 데이터 타입에 따른 버퍼.



<< java.nio의 Buffer 계층도 >>

8. JAVA NIO

6. 버퍼

- 버퍼가 사용하는 메모리의 위치에 따라서 넌다이렉트와 다이렉트 버퍼로 나뉜다.
- 넌 다이렉트 버퍼는 JVM이 관리하는 힙 메모리 공간을 이용하는 버퍼이고, 다이렉트 버퍼는 운영체제가 관리하는 메모리 공간을 이용하는 버퍼이다.

구분	넌다이렉트 버퍼	다이렉트 버퍼
사용하는 메모리 공간	JVM의 힙 메모리	운영체제의 메모리
버퍼의 생성시간	버퍼 생성이 빠르다	버퍼 생성이 느리다
버퍼의 크기	작다	크다(큰데이터를 처리할 때 유리)
입출력 성능	낮다	높다(입출력이 빈번할 때 유리)

```
ByteBuffer directBuffer = ByteBuffer.allocateDirect(200 * 1024 * 1024);
System.out.println("다이렉트 버퍼가 생성되었습니다.");

ByteBuffer nonDirectBuffer = ByteBuffer.allocate(200 * 1024 * 1024);
System.out.println("넌다이렉트 버퍼가 생성되었습니다.");
```

8. JAVA NIO

7. 버퍼 생성

- 각 데이터 타입별로 닌다이렉트 버퍼를 생성하기 위해서는 각 Buffer 클래스의 `allocate()`와 `wrap()` 메소드를 호출하면 되고, 다이렉트 버퍼는 `ByteBuffer`의 `allocateDirect()` 메소드를 호출하면 된다.

`allocate()`

```
ByteBuffer byteBuffer = ByteBuffer.allocate(100);  
CharBuffer charBuffer = CharBuffer.allocate(100);
```

`wrap()`

```
byte[] byteArray = new byte[100];  
ByteBuffer byteBuffer = ByteBuffer.wrap(byteArray)  
char[] charArray = new char[100];  
CharBuffer charBuffer = CharBuffer.wrap(charArray, 0, 50)
```

8. JAVA NIO

7. 버퍼 생성

- allocateDirect() 방식으로 선언하는 방법은 아래와 같다..

```
allocateDirect()
```

```
ByteBuffer byteBuffer = ByteBuffer.allocateDirect(100);
```

```
CharBuffer charBuffer = ByteBuffer.allocateDirect(100).asCharBuffer();
```

```
public class DirectBufferCapacityExample {  
    public static void main(String[] args) {  
        ByteBuffer byteBuffer = ByteBuffer.allocateDirect(100);  
        System.out.println("저장용량: " + byteBuffer.capacity() + " 바이트");  
  
        CharBuffer charBuffer = ByteBuffer.allocateDirect(100).asCharBuffer();  
        System.out.println("저장용량: " + charBuffer.capacity() + " 문자");  
  
        IntBuffer intBuffer = ByteBuffer.allocateDirect(100).asIntBuffer();  
        System.out.println("저장용량: " + intBuffer.capacity() + " 정수");  
    }  
}
```


8. JAVA NIO

8. 버퍼의 위치속성

- Buffer를 사용하려면 Buffer의 위치 속성 개념과 위치 속성이 언제 변경되는지 알고 있어야 한다..

속성	설명
position	현재 읽거나 쓰는 위치값이다. 인덱스 값이기 때문에 0부터 시작하며, limit 보다 큰 값을 가질 수 없다. 만약 position과 limit의 값이 같아진다면 더 이상 데이터를 쓰거나 읽을 수 없다는 뜻이 된다.
limit	버퍼에서 읽거나 쓸 수 있는 위치의 한계를 나타낸다. 이 값은 capacity보다 작거나 같은 값을 가진다. 최초에 버퍼를 만들었을 때는 capacity와 같은 값을 가진다.
capacity	버퍼의 최대 데이터 개수(메모리 크기)를 나타낸다. 인덱스 값이 아니라 수량임에 주의
mark	reset() 메소드를 실행했을 때에 돌아오는 위치를 지정하는 인덱스로써 mark() 메소드로 지정할 수 있다. 주의할 점은 반드시 position 이하의 값으로 지정해주어야 한다. position이나 limit의 값이 mark 값보다 작은 경우 mark는 자동 제거된다. mark가 없는 상태에서 reset() 메소드를 호출하면 InvalidMarkException이 발생한다.

8. JAVA NIO

9. BufferMethod

- Buffer의 공용 메서드는 아래와 같다...

리턴타입	메소드(매개변수)	설명
Object	array()	버퍼가 wrap한 배열을 리턴
int	arrayOffset()	버퍼의 첫 번째 요소가 있는 내부 배열의 인덱스를 리턴
int	capacity()	버퍼의 전체 크기를 리턴
Buffer	clear()	버퍼의 위치 속성을 초기화
Buffer	flip()	limit를 position으로, position을 0 인덱스로 이동
boolean	hasArray()	버퍼가 래핑한 배열을 가지고 있는지 여부
boolean	hasRemaining()	position과 limit 사이에 요소가 있는지 여부
boolean	isDirect()	운영체제의 버퍼를 사용하는지 여부
boolean	isReadOnly()	버퍼가 읽기 전용인지 여부
int	limit()	limit() 위치를 리턴
Buffer	limit(int newLimit)	newLimit으로 limit 위치를 설정
Buffer	mark()	현재 위치를 mark로 표시

8. JAVA NIO

9. BufferMethod

- Buffer의 공용 메서드는 아래와 같다...

리턴타입	메소드(매개변수)	설명
int	position()	position 위치를 리턴
Buffer	position(int new Position)	newPosition 위치로 position 위치를 설정
int	remaining()	position과 limit 사이의 요소의 개수
Buffer	reset()	position을 mark 위치로 이동
Buffer	rewind()	position을 0 인덱스로 이동

8. JAVA NIO

9. BufferMethod

- 데이터를 읽고 저장하는 메소드는 put()이고, 데이터를 읽는 메소드는 get()이다.
- Buffer의 추상 클래스에는 위의 두 메서드가 존재하지 않으며 각 타입별 하위 Buffer클래스가 가지고 있다.
- get()과 put() 메소드는 상대적(Relative)와 절대적(Absolute)으로 구분되며 버퍼내의 현재 위치 속성인 position에서 데이터를 읽고, 저장할 경우는 상대적이고 position과 관계없이 주어진 인덱스에서 데이터를 읽고, 저장할 경우는 절대적이다.
- 상대적 get()과 put() 메소드를 호출하면 position의 값은 증가하지만, 절대적 get()과 put() 메소드를 호출하면 position 값은 증가되지 않는다.
- 보통 상대적 get은 인자가 존재가 없거나 배열을 인자로 받지만 절대적 get은 인자에 반드시 int index가 들어간다.
- put은 상대적으로 인자가 value 값만 존재하거나 byte 배열을 받는 인자가 존재하지만 절대적 put은 앞에 int index가 붙는다.

8. JAVA NIO

9. BufferMethod

- 예제(1)

```
public class BufferExample {  
    public static void main(String[] args) {  
        System.out.println("[7바이트 크기로 버퍼 생성]");  
        ByteBuffer buffer = ByteBuffer.allocateDirect(7);  
        printState(buffer);  
  
        buffer.put((byte)10);  
        buffer.put((byte)11);  
        System.out.println("[2바이트 저장후]");  
        printState(buffer);  
  
        buffer.put((byte)12);  
        buffer.put((byte)13);  
        buffer.put((byte)14);  
        System.out.println("[3바이트 저장후]");  
        printState(buffer);  
  
        buffer.flip();  
        System.out.println("[flip() 실행후]");  
        printState(buffer);  
    }  
}
```

8. JAVA NIO

9. BufferMethod

- 예제(1)

```
buffer.get(new byte[3]);
System.out.println("[3바이트 읽은후]");
printStats(buffer);

buffer.mark();
System.out.println("-----[현재 위치를 마크 해놓음]");

buffer.get(new byte[2]);
System.out.println("[2바이트 읽은후]");
printStats(buffer);

buffer.reset();
System.out.println("-----[position을 마크 위치로 옮김]");
printStats(buffer);

buffer.rewind();
System.out.println("[rewind() 실행후]");
printStats(buffer);

buffer.clear();
System.out.println("[clear() 실행후]");
printStats(buffer);
}
```

8. JAVA NIO

9. BufferMethod

- 예제(1)

```
public static void printState(Buffer buffer) {  
    System.out.print("\tposition:" + buffer.position() + ", ");  
    System.out.print("\tlimit:" + buffer.limit() + ", ");  
    System.out.println("\tcapacity:" + buffer.capacity());  
}  
}
```

8. JAVA NIO

9. BufferMethod

- 예제(2)

```
public class CompactExample {  
    public static void main(String[] args) {  
        System.out.println("[7바이트 크기로 버퍼 생성]");  
        ByteBuffer buffer = ByteBuffer.allocateDirect(7);  
        buffer.put((byte)10);  
        buffer.put((byte)11);  
        buffer.put((byte)12);  
        buffer.put((byte)13);  
        buffer.put((byte)14);  
        buffer.flip();  
        printState(buffer);  
  
        buffer.get(new byte[3]);  
        System.out.println("[3바이트 읽음]");  
  
        buffer.compact();  
        System.out.println("[compact() 실행 후]");  
        printState(buffer);  
    }  
}
```


8. JAVA NIO

9. BufferMethod

- 예제(2)

```
public static void printState(ByteBuffer buffer) {  
    System.out.print(buffer.get(0) + ", ");  
    System.out.print(buffer.get(1) + ", ");  
    System.out.print(buffer.get(2) + ", ");  
    System.out.print(buffer.get(3) + ", ");  
    System.out.println(buffer.get(4));  
    System.out.print("position:" + buffer.position() + ", ");  
    System.out.print("limit:" + buffer.limit() + ", ");  
    System.out.println("capacity:" + buffer.capacity());  
}
```

8. JAVA NIO

10. Buffer 변환

- 채널이 데이터를 읽과 쓰는 버퍼는 모두 ByteBuffer이다.
- 채널을 통해 읽은 데이터를 복원하려면 ByteBuffer를 문자열 또는 다른 타입 버퍼 (CharBuffer, ShortBuffer, IntBuffer, LongBuffer, FloatBuffer, DoubleBuffer)로 변환해야 한다.
- 반대로 문자열 또는 다른 타입 버퍼의 내용을 채널을 통해 쓰고 싶다면 ByteBuffer로 변환해야 한다.

8. JAVA NIO

10. Buffer 변환

- ByteBuffer <-> String

```
public class ByteBufferToStringExample {  
    public static void main(String[] args) {  
        Charset charset = Charset.forName("UTF-8");  
  
        //문자열 -> 인코딩 -> ByteBuffer  
        String data = "안녕하세요";  
        ByteBuffer byteBuffer = charset.encode(data);  
  
        //ByteBuffer -> 디코딩 -> CharBuffer -> 문자열  
        data = charset.decode(byteBuffer).toString();  
        System.out.println("문자열 복원: " + data);  
    }  
}
```

8. JAVA NIO

10. Buffer 변환

- ByteBuffer <-> IntBuffer

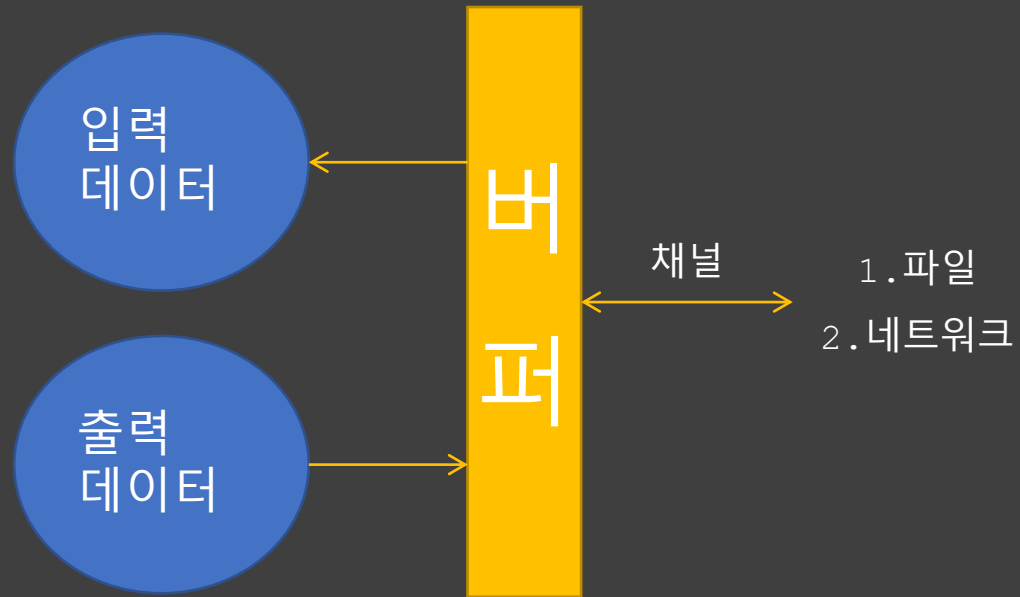
```
public class ByteBufferToIntBufferExample {
    public static void main(String[] args) throws Exception {
        //int[] -> IntBuffer -> ByteBuffer
        int[] writeData = { 10, 20 };
        IntBuffer writeIntBuffer = IntBuffer.wrap(writeData);
        ByteBuffer writeByteBuffer = ByteBuffer.allocate(writeIntBuffer.capacity());
        for(int i=0; i<writeIntBuffer.capacity(); i++) {
            writeByteBuffer.putInt(writeIntBuffer.get(i));
        }
        writeByteBuffer.flip();

        //ByteBuffer -> IntBuffer -> int[]
        ByteBuffer readByteBuffer = writeByteBuffer;
        IntBuffer readIntBuffer = readByteBuffer.asIntBuffer();
        int[] readData = new int[readIntBuffer.capacity()];
        readIntBuffer.get(readData);
        System.out.println("배열 복원: " + Arrays.toString(readData));
    }
}
```

8. JAVA NIO

11. 파일채널(Channel)

- java.nio.channels.FileChannel을 이용하면 파일 읽기와 쓰기를 할 수 있다.
- FileChannel은 동기화 처리가 되어있기 때문에 멀티 스레드 환경에서 사용해도 안전하다



8. JAVA NIO

11. 파일채널(Channel)

- FileChannel은 정적 메소드인 open()을 호출해서 얻을 수도 있지만, IO의 FileInputStream, FileOutputStream의 getChannel()메소드를 호출해서 얻을 수 있다.

사용 예

8. JAVA NIO

11. 파일채널(Channel)

- 두 번째 options 매개값은 열거 옵션 값이며 아래와 같은 상수를 나열할 수 있다

열거 상수	설명
READ	읽기용으로 파일을 연다.
WRITE	쓰기용으로 파일을 연다.
CREATE	파일이 없다면 새 파일을 생성한다.
CREATE_NEW	새 파일을 만든다. 이미 파일이 있으면 예외와 함께 실패한다.
APPEND	파일 끝에 데이터를 추가한다.(WRITE나 CREATE와 함께 사용됨)
DELETE_ON_CLOSE	채널을 닫을 때 파일을 삭제한다(임시 파일을 삭제할 때 사용)
TRUNCATE_EXISTING	파일을 0바이트로 잘라낸다(WRITE 옵션도 함께 사용됨)

8. JAVA NIO

11. 파일채널(Channel)

- 파일 생성 및 쓰기 예제

```
public class FileChannelWriteExample {  
    public static void main(String[] args) throws IOException {  
        Path path = Paths.get("C:/Temp/file.txt");  
        Files.createDirectories(path.getParent());  
  
        FileChannel fileChannel = FileChannel.open(  
            path, StandardOpenOption.CREATE, StandardOpenOption.WRITE);  
  
        String data = "안녕하세요";  
        Charset charset = Charset.defaultCharset();  
        ByteBuffer byteBuffer = charset.encode(data);  
  
        int byteCount = fileChannel.write(byteBuffer);  
        System.out.println("file.txt : " + byteCount + " bytes written");  
  
        fileChannel.close();  
    }  
}
```


8. JAVA NIO

11. 파일채널(Channel)

- 파일 읽기 예제

```
public class FileChannelReadExample {  
    public static void main(String[] args) throws IOException {  
        Path path = Paths.get("C:/Temp/file.txt");  
  
        FileChannel fileChannel = FileChannel.open(  
            path, StandardOpenOption.READ);  
        ByteBuffer byteBuffer = ByteBuffer.allocate(100);  
        Charset charset = Charset.defaultCharset();  
        String data = "";  
        int byteCount;  
  
        while(true) {  
            byteCount = fileChannel.read(byteBuffer);  
            if(byteCount == -1) break;  
            byteBuffer.flip();  
            data += charset.decode(byteBuffer).toString();  
            byteBuffer.clear();  
        }  
  
        fileChannel.close();  
        System.out.println("file.txt : " + data);  
    }  
}
```

8. JAVA NIO

11. 파일채널(Channel)

- 파일 복사 예제

```
public class FileCopyExample {  
    public static void main(String[] args) throws IOException {  
        Path from = Paths.get("src/sec04/exam02_file_copy/house1.jpg");  
        Path to = Paths.get("src/sec04/exam02_file_copy/house2.jpg");  
  
        FileChannel fileChannel_from = FileChannel.open(  
            from, StandardOpenOption.READ);  
        FileChannel fileChannel_to = FileChannel.open(  
            to, StandardOpenOption.CREATE, StandardOpenOption.WRITE);  
  
        ByteBuffer buffer = ByteBuffer.allocateDirect(100);  
        int byteCount;  
        while(true) {  
            buffer.clear();  
            byteCount = fileChannel_from.read(buffer);  
            if(byteCount == -1) break;  
            buffer.flip();  
            fileChannel_to.write(buffer);  
        }  
  
        fileChannel_from.close();  
        fileChannel_to.close();  
        System.out.println("파일 복사 성공");  
    }  
}
```

8. JAVA NIO

12. TransferTo, TransferFrom 를 이용한 파일 복사

- 기존의 ByteBuffer를 통한 파일 복사처리는 Blocking이 되었기에 IO 보다 뚜렷한 성능의 차이가 보이지 않았다.
- 하지만 Channel에서 제공하는 non-blocking 방식의 파일 처리 방법을 제공하는 메소드가 등장.
- TransferTo() 메소드와 TransferFrom 메소드는 강력한 non-blocking 방식의 FileCopy를 제공한다.

사용 예

```
fc.transferTo(시작지점(0), IO크기, 타겟IO)  
fc.transferFrom(타겟IO, 시작지점(0), IO크기)
```

8. JAVA NIO

12. TransferTo, TransferFrom 를 이용한 파일 복사

```
public class CopyWithTransfer {
    public static void main(String[] args) {
        try{
            FileChannel fc1 = new FileInputStream("C:/test/ccc.txt").getChannel();
            FileChannel fc2 = new FileOutputStream("C:/test/ggg.txt").getChannel();
            fc1.transferTo(0, fc1.size(), fc2);
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

public class CopyWithTransferFrom {
    public static void main(String[] args) {
        try{
            FileChannel fc1 = new FileInputStream("C:/test/ccc.txt").getChannel();
            FileChannel fc2 = new FileOutputStream("C:/test/hhh.txt").getChannel();
            fc2.transferFrom(fc1, 0, fc1.size());
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

8. JAVA NIO

13. Copy 메소드를 이용한 파일 복사

```
public class FilesCopyMethodExample {  
    public static void main(String[] args) throws IOException {  
        Path from = Paths.get("src/sec04/exam02_file_copy/house1.jpg");  
        Path to = Paths.get("src/sec04/exam02_file_copy/house2.jpg");  
  
        Files.copy(from, to, StandardCopyOption.REPLACE_EXISTING);  
        System.out.println("파일 복사 성공");  
    }  
}
```