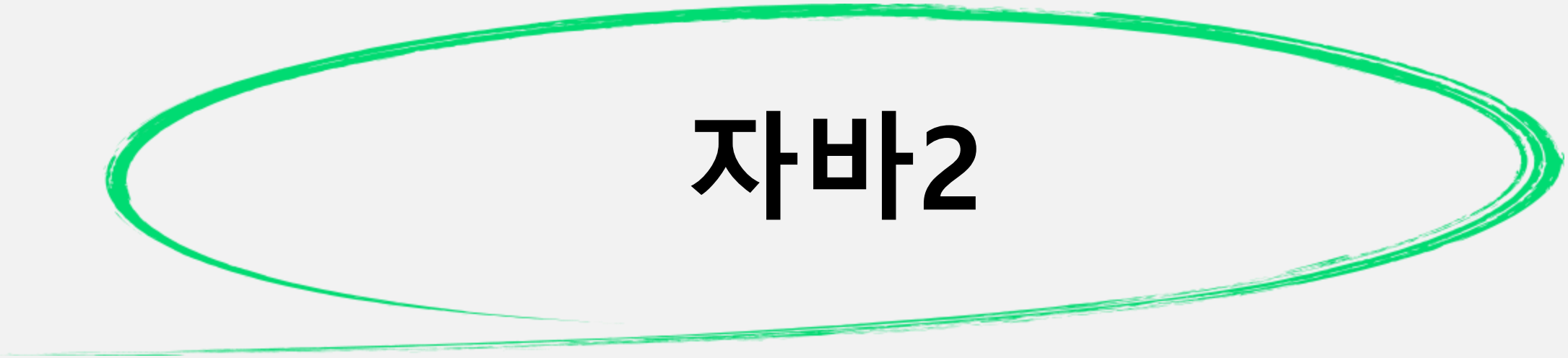


자바2

(파일 입출력)

Chapter 10
파일 입출력

Chapter10의 학습목표

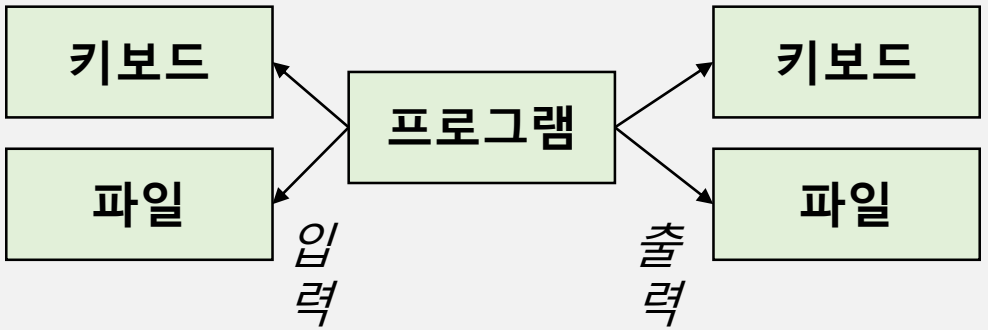
- 자바에서의 데이터의 흐름에 대해 알아본다
- 파일에 데이터를 읽고 쓰는 방법에 대해 학습한다

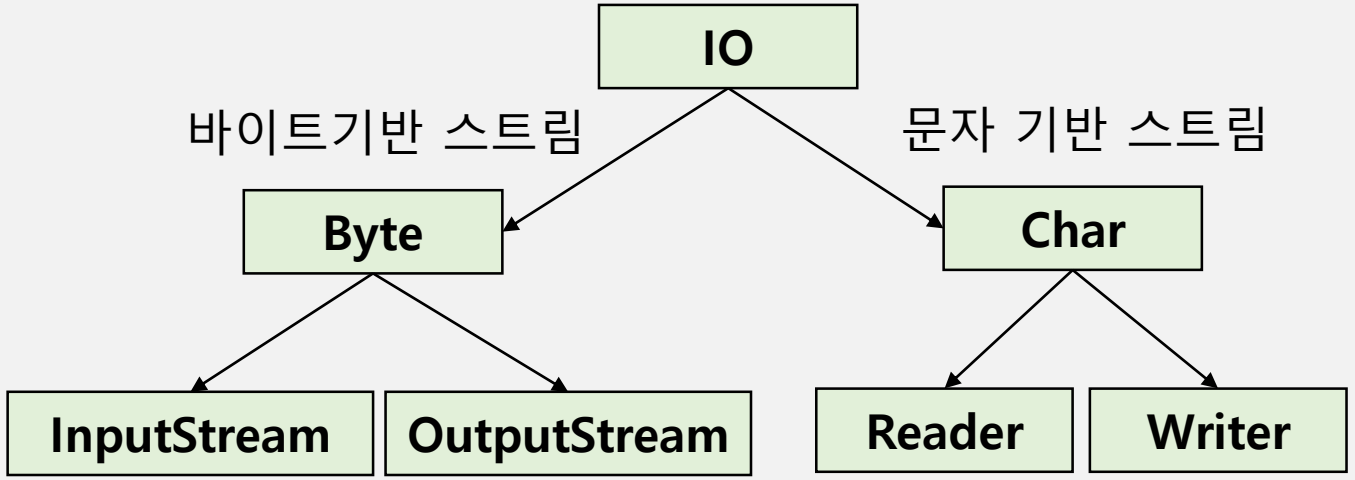
IO

IO와 스트림

- IO (Input Output) : 프로그램으로 들어오는 모든 값 -> Input, 밖으로 나가는 값 -> Output
  자바에서는 입출력을 처리하기 위해 IO package를 제공
- 스트림 (Stream) : 데이터의 흐름

1. **FIFO구조 : 스트림의 데이터는 순차적으로 흘러가며 순차적 접근 밖에서는 허용되지 않아, 데이터의 순서가 바뀌지 않는다는 특징이 있음. 먼저 들어온 데이터가 먼저 나가는 FIFO구조를 가짐**
2. **단방향 : 스트림은 읽기, 쓰기가 동시에 되지 않음. 데이터 통신은 한 쪽 방향으로만 가능함.**
3. **지연상태 : 스트림에 넣어진 데이터가 처리되기 전까지는 스트림에 사용되는 스레드는 지연상태에 빠짐.**





IO

File 클래스

- 로컬에 있는 파일이나 디렉토리 경로를 추상화한 클래스
- 파일의 생성과 삭제, 파일의 수정 날짜 기록 등 다양한 기능을 제공

생성자	설명
File(String pathname)	문자열 pathname을 가지고 경로를 생성하여 File객체를 생성
File(String parent, String child)	Parent와 child 문자열을 연결한 문자열로 경로를 생성하여 File 객체를 생성
File(File parent, String child)	Parent의 파일 객체와 child 문자열을 연결한 문자열로 경로를 생성하여 File 객체를 생성

IO

File 클래스 메소드

메소드)	반환형	설명
canRead()	boolean	파일을 읽을 수 있으면 true, 아니면 false를 반환
canWrite()		파일을 쓸 수 있으면 true, 아니면 false를 반환
createNewFile()		파일을 새로 생성하면 true, 아니면 false를 반환
delete()		파일을 지우면 true, 아니면 false를 반환
exists()		파일이나 디렉토리가 존재하면 true, 아니면 false를 반환
getAbsolutePath()	String	파일의 절대 경로 반환
getCanonicalPath()		파일의 정규 경로를 반환
getName()		파일명을 반환
isDirectory()	boolean	디렉토리면 true, 아니면 false를 반환
ifFile()		파일이면 true, 아니면 false를 반환
length()	long	파일의 크기를 바이트로 반환
mkdir()	Boolean	디렉토리를 생성하면 true, 생성하지 못하면 false를 반환(기존에 있을 경우도 false)
list()	String[]	특정 디렉토리의 모든 파일과 자식 디렉토리를 String 배열로 반환

IO

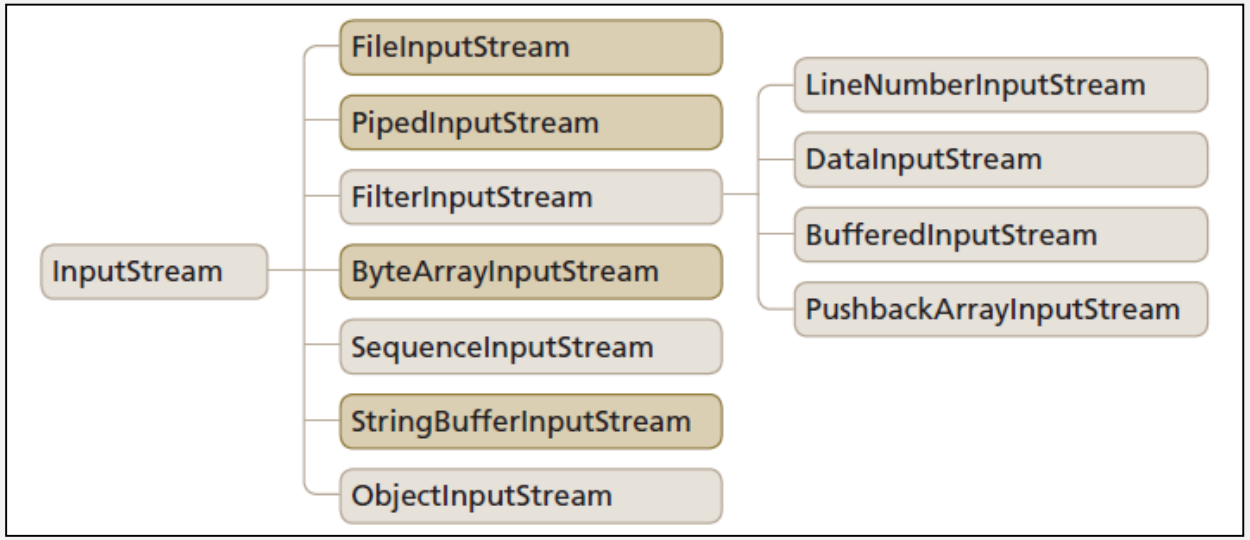
File 클래스 – 현재 경로의 모든 폴더와 파일을 가져오는 예제

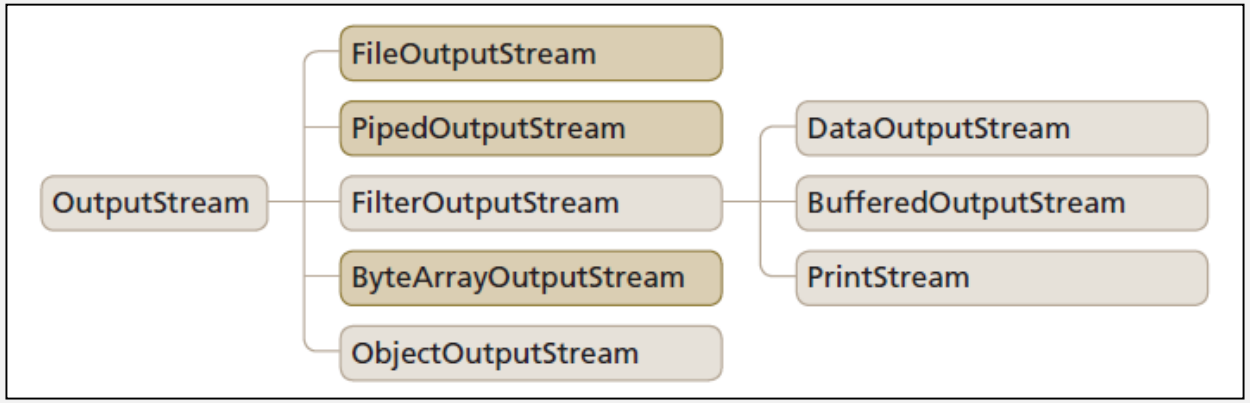
```
import java.io.File;

public class FileClassTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String filePath = ".\\";
        File f1 = new File(filePath);
        String list[] = f1.list(); //filePath 디렉토리의 모든 파일과 폴더를 가져옴
        for (int i = 0; i < list.length; i++) {
            File f2 = new File(filePath, list[i]);
            if(f2.isDirectory()) {
                System.out.println(list[i] + " : 디렉토리");
            }else {
                System.out.printf("%s : 파일 (%,dbyte) %n",list[i] ,f2.length());
            }
        }

    }
}
```

IO

바이트기반 스트림 (Byte stream)

- 바이트 단위로 데이터를 입출력하는 스트림 (1byte)
- 이진 데이터를 읽고 쓰기 위하여 사용 (이미지, 동영상, 음악 파일 등)





메소드명	설명
abstract int read()	한 바이트를 읽어서 반환(0~255사이의 정수) 만약 파일의 끝이면 -1을 반환
abstract void write(int b)	한 바이트(b)를 특정한 장치에 씀
close()	스트림을 닫음

IO

파일 입출력 바이트 스트림

- FileInputStream/FileOutputStream 클래스를 사용
- FileInputStream : 파일에서 바이트를 읽음
- FileOutputStream : 파일에 바이트를 씀

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class A_Class {
    public static void main(String[] args) throws IOException{
        FileInputStream in = null;
        FileOutputStream out = null;
        try {
            in = new FileInputStream("input.txt");
            out = new FileOutputStream("output.txt");
            int c = in.read();

            while(c != -1) {
                System.out.print((char)c);
                out.write(c);
                c = in.read();
            }
        } finally {
            if (in != null)
                in.close();
            if (out != null)
                out.close();
        }
    }
}
```

txt파일에 연결된 파일 스트림을 생성

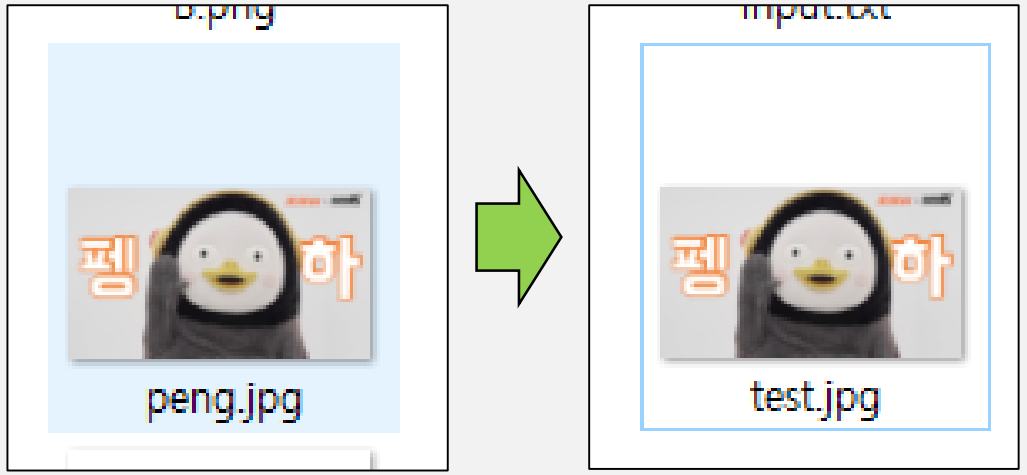
파일이 끝나면 -1을 반환

IO

이미지 파일 복사하기

하나의 이미지 파일을 다른 이미지 파일로 복사하는 프로그램을 작성 (이미지 파일은 이진 파일)

```
public class A_Class {
    public static void main(String[] args) throws IOException{
        Scanner scan = new Scanner(System.in);

        System.out.println("원본 파일 이름을 입력하시오");
        String inputFileName = scan.next();

        System.out.println("복사 파일 이름을 입력하시오");
        String outputFileName = scan.next();

        try (InputStream inputStream = new FileInputStream(inputFileName);
                OutputStream outputStream = new FileOutputStream(outputFileName)){
            int c = inputStream.read();

            while(c != -1) {
                outputStream.write(c);
                c = inputStream.read();
            }
        }

        System.out.println(inputFileName + "을 " + outputFileName + "로 복사하였습니다.");


    }
}
```



IO

대표적인 바이트 스트림 클래스들

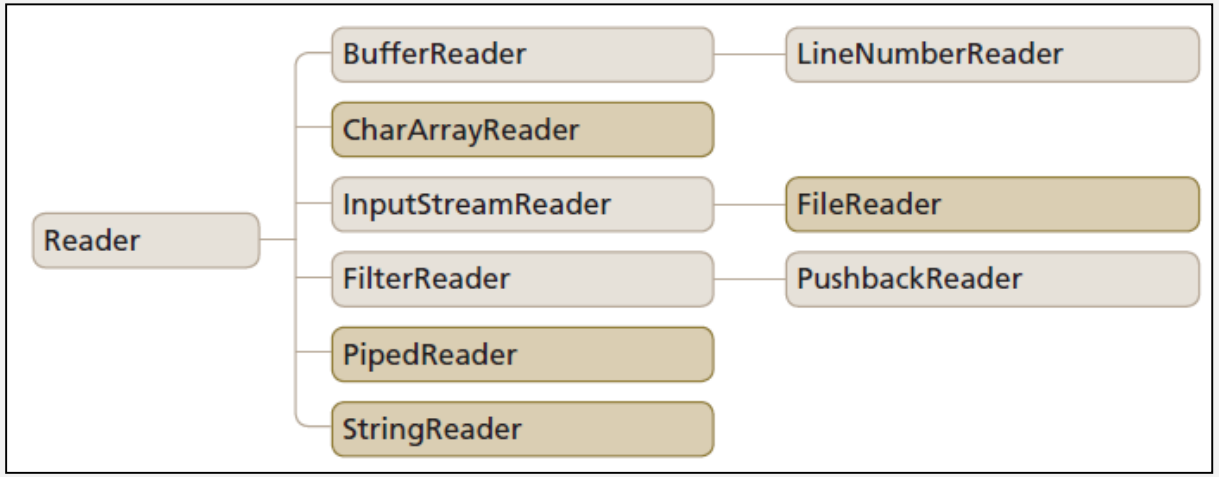
- 크게 Input과 Output으로 나뉨

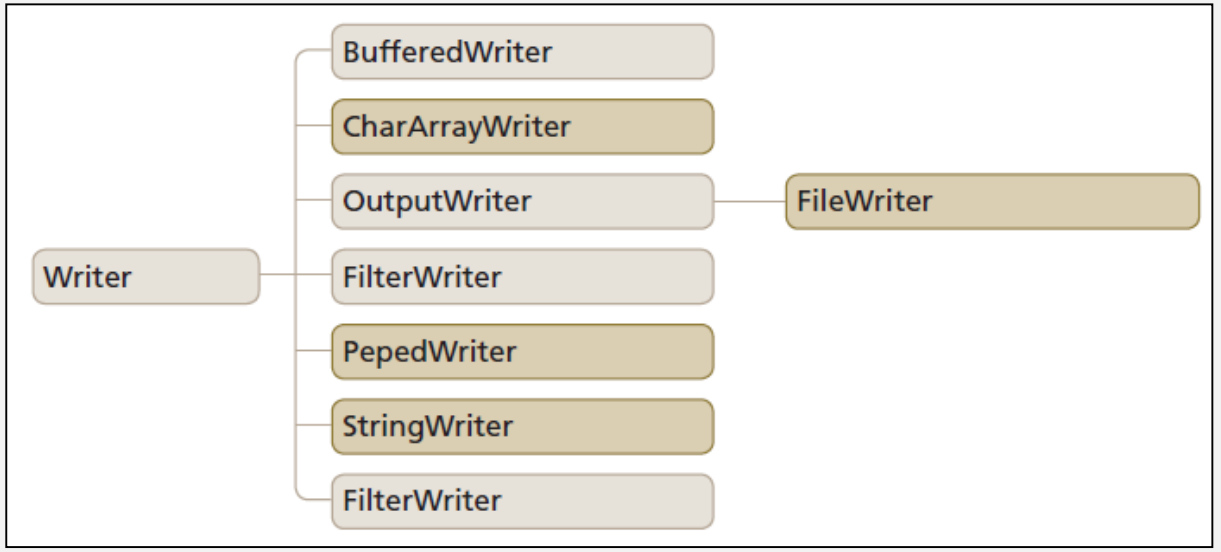
클래스명	설명
InputStream	바이트 입력을 수행하는데 필요한 메소드를 정의하는 추상 클래스
FileInputStream	시스템에 있는 모든 파일을 읽을 수 있는 기능을 제공하는 클래스
DataInputStream	입력 스트림으로부터 기본형 데이터를 읽기 위한 메소드를 제공 (int,byte,char,double 등 기본 자료형 + UTF와 같은 문자)
BufferedInputStream	입출력 수행을 향상시킨 클래스로, 수많은 논리적 데이터들이 버퍼로 입력되어 버퍼로 저장됨
OutputStream	바이트 출력을 수행하는데 필요한 메소드를 정의하는 추상 클래스
FileOutputStream	시스템에 있는 모든 파일에 쓸 수 있는 기능을 제공하는 클래스
DataOutputStream	출력 스트림에 기본형 데이터를 쓰기 위한 메소드를 제공 (int,byte,char,double 등 기본 자료형 + UTF와 같은 문자)
BufferedOutputStream	출력 속도의 향상을 위해 버퍼링을 사용하는 클래스로 출력할 바이트를 버퍼에 저장하여 버퍼가 차면 한번에 출력함
PrintStream	지정된 스트림에 텍스트 출력을 수행하는 클래스. print(), println()이 오버라이딩 되어있음

IO

문자기반 스트림 (Character stream)

- 문자 단위로 데이터를 입출력하는 스트림 (2byte)





메소드명	설명
abstract int read()	한 문자를 읽어서 반환. 파일의 끝이면 -1을 반환
abstract void write(int b)	한 바이트(b)를 특정한 장치에 씀
close()	스트림을 닫음

IO

파일 입출력 문자 스트림

- FileReader/FileWriter 클래스를 사용
- FileReader : 파일에서 문자를 읽음
- FileWriter : 파일에 문자를 씀

```
public class A_Class {
    public static void main(String[] args) throws IOException{
        FileReader inputStream = null;
        FileWriter outputStream = null;

        try {
            inputStream = new FileReader("input.txt");
            outputStream = new FileWriter("output.txt");

            int c;
            while((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }

        } finally {
            if (inputStream != null)
                inputStream.close();
            if (outputStream != null)
                outputStream.close();
        }
    }
}
```

txt파일에 연결된 파일 스트림을 생성

IO

대표적인 문자 스트림 클래스들

- 크게 Reader와 Writer로 나뉨

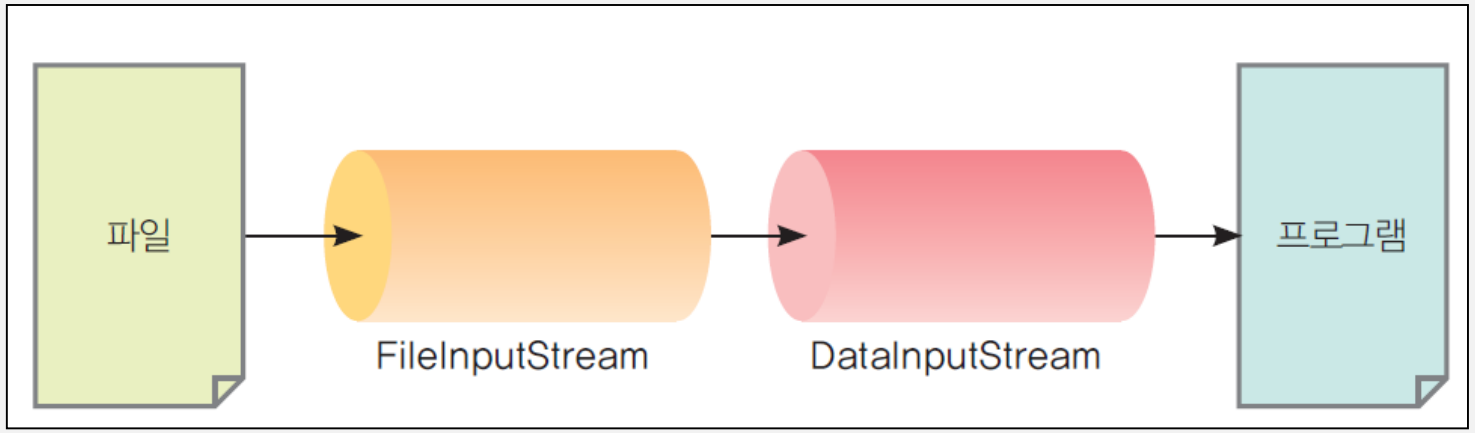
클래스명	설명
Reader	문자 입력 스트림의 최상위 추상 클래스. InputStream 클래스와 거의 동일
FileReaader	시스템에 있는 문자 파일을 읽을 수 있는 기능을 제공
BufferedReader	바이트 기반 스트림의 BufferedInputStream과 동일한 기능을 제공하는 클래스
Writer	문자 출력 스트림의 최상위 추상 클래스. OutputStream 클래스와 거의 동일
FileWriter	문자 파일에 출력(쓰기)할 때 사용되는 클래스
BufferedWriter	바이트 스트림의 BufferedOutputStream과 동일한 기능을 제공하는 클래스
PrintWriter	바이트 출력 스트림(OutputStream)과 문자 출력(Writer) 스트림을 모두 매개변수로 받을 수 있는 생성자를 제공하는 클래스

IO

스트림 결합하기

- 스트림들끼리 결합이 가능
- 스트림을 통하여 흘러가는 데이터에 대해 다양한 가공 처리가 가능

```
FileInputStream fileSt = new FileInputStream("sample.dat") //정수들이 저장되어 있는 파일을 읽음
DataInputStream dataSt = new DataInputStream(fileSt); //정수를 읽기 위해 DataInputStream을 결합
int i = dataSt.readInt(); //readInt()메소드를 통해 정수를 읽음
```

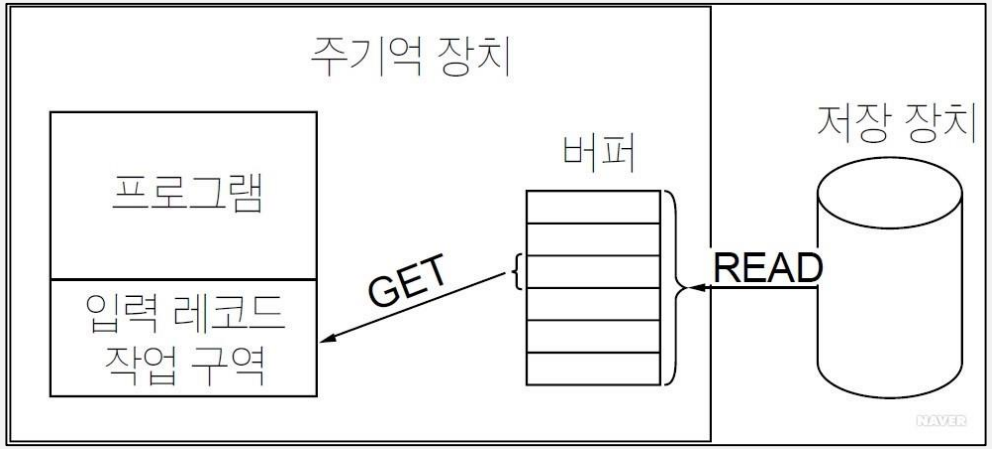


IO

버퍼 스트림

- 각 read/write 요청은 운영 체제에 의하여 요청되는 즉시 처리됨
  – 입출력 요청은 디스크 접근과 같은 시간이 오래 걸리는 동작을 요구하기 때문에 비효율적
- 자바에서는 버퍼링된 스트림(buffered I/O)를 제공
- 버퍼 입력 스트림은 버퍼 (메모리 영역)에서 데이터를 읽음

```
//버퍼가 없는 스티림을 버퍼가 있는 스트림으로 변경하여 입출력 스트림 객체를 생성
inputStream = new BufferedReader(new FileReader("input.txt"));
outputStream = new BufferedWriter(new FileWriter("output.txt"));
```



IO

압축 파일 풀기

- ZipInputStream을 사용
- 압축파일 안에는 ZipEntry 타입의 객체가 여러 개 저장되어있음

InputStream으로 Zip파일을 읽음 ➡ ZipEntry타입의 객체를 얻음 ➡ FileOutputStream으로 읽은 객체를 데이터에 씀

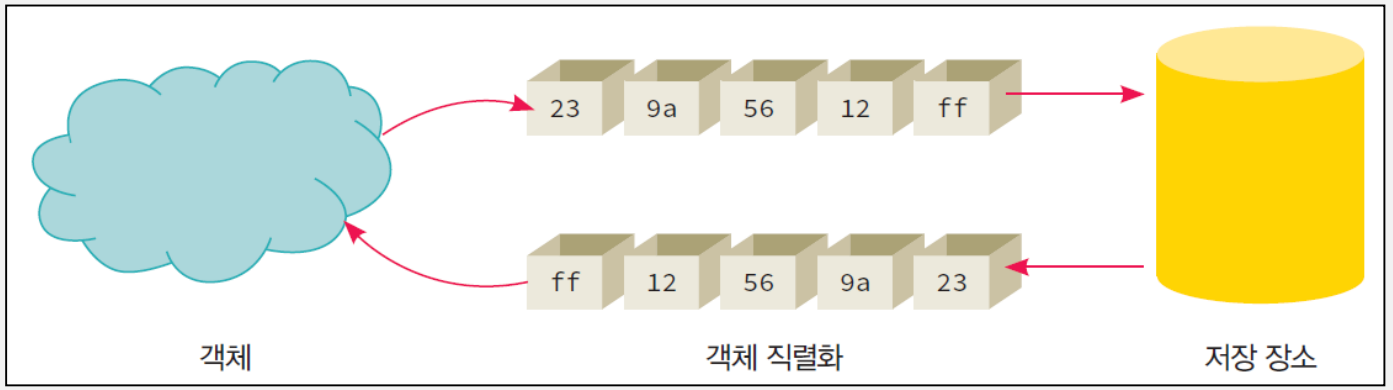
```
FileInputStream fin = new FileInputStream("test.zip");
ZipInputStream zin = new ZipInputStream(fin);
ZipEntry entry = null;
while ((entry = zin.getNextEntry()) != null) {
    System.out.println("압축 해제: " + entry.getName());
    FileOutputStream fout = new FileOutputStream(entry.getName());
    for (int c = zin.read(); c != -1; c = zin.read()) {
        fout.write(c);
    }
    zin.closeEntry();
    fout.close();
}
zin.close();
```

IO

직렬화 (Serializable) : 객체 저장하기

- 자바의 객체나 데이터를 외부의 시스템에서 사용할 수 있도록 Byte 단위의 데이터로 변환시키는 기술 혹은 그 반대의 기술
- 객체를 파일에 저장하기 위해서는 객체가 가진 데이터들을 순차적인 데이터로 반환하는 절차를 거쳐야 함
- Serializable 인터페이스는 아무런 메소드가 없고 단순히 JVM에게 정보를 전달하는 의미로 사용
- 데이터들을 한 줄로 나열시켜 cpu가 순차적으로 데이터 처리를 할 수 있도록 함

객체 전송의 단계
1. 직렬화된 객체를 바이트 단위로 분해 **(mashalling)**
2. 직렬화되어 분해된 데이터를 순서에 따라 전송
3. 전송받은 데이터를 복구(unmashalling)



데이터를 바이트 단위로 변환시키는 작업

⬇

Mashalling이 가능한 객체

1. 기본형 타입 (int, byte, char, boolean 등)
2. Serializable 인터페이스를 구현

IO

직렬화 (Serializable) – ObjectInputStream / ObjectOutputStream 예제1

- ObjectInput / ObjectOutput 인터페이스를 구현하는 클래스
- Serialzable을 구현하지 않은 경우 NotSerializableException 예외가 발생함
- 직렬화 스트림 기능을 제공하는 클래스

클래스명	메소드명	설명
ObjectOutputStream	writeObject(Object obj)	객체의 데이터를 직렬화시켜 저장
ObjectInputStream	readObject()	직렬화되어 있는 데이터를 Object 형태의 객체로 바꿔서 저장

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

class Customer implements java.io.Serializable{
    private String name;
    public Customer(String name) {
        this.name = name;
    }

    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    @Override
    public String toString() {
        return "당신의 이름 : " + name;
    }
}
```

IO

직렬화 (Serializable) – ObjectInputStream / ObjectOutputStream 예제1

```
public class ObjectStreamTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ObjectInputStream ois = null;
        ObjectOutputStream oos = null;
        FileInputStream fis = null;
        FileOutputStream fos = null;
```

```
        try {
            // 두 스트림을 연결
            fos =  new FileOutputStream("object.dat");
            oos = new ObjectOutputStream(fos);
            // 객체를 파일에 씀
            oos.writeObject(new Customer("Kim"));

            fis = new FileInputStream("object.dat");
            ois = new ObjectInputStream(fis);
            // 파일에서 데이터를 읽어서 객체로 변경
            Customer m = (Customer) ois.readObject();
            System.out.println(m); //객체의 toString메소드 실행
        }catch (Exception e) {
            // TODO: handle exception
        }finally {
            try {
                if (ois != null) ois.close();
                if (oos != null) oos.close();
                if (fis != null) fis.close();
                if (fos != null) fos.close();
            }catch (Exception e) {
                // TODO: handle exception
            }
        }
    }
}
```

IO

직렬화 (Serializable) : ObjectInputStream / ObjectOutputStream 예제2

Date 클래스를 이용하여 현재 날짜를 나타내는 객체를 저장했다가 다시 읽어서 콘솔에 표시

```
ObjectInputStream in = null;
ObjectOutputStream out = null;
try {
        int c;
        out = new ObjectOutputStream(new FileOutputStream("object.dat"));
        out.writeObject(new Date());

        out.flush();
        in = new ObjectInputStream(new FileInputStream("object.dat"));
        Date d = (Date) in.readObject();
        System.out.println(d);
}
catch (Exception e) {
}
finally {
       if (in != null) {
              in.close();
       }
       if (out != null) {
              out.close();
       }
}
```

객체를 직렬화해서 데이터를 씀

IO

이미지 파일에서 RGB 값 구하기

이미지 파일에서 픽셀 값을 읽어서 GrayScale 이미지로 변환한 후 저장하기

픽셀 값 얻기

```
Color c = new Color(image.getRGB(x, y));
```

R, G, B 값 얻기

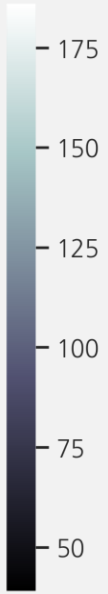
```
c.getRed();
c.getGreen();
c.getBlue();
```

컬러 이미지를 그레이스케일 이미지로 변환

```
Y = 0.299R + 0.587G + 0.114B
```

이미지 표현 형태 (2차원 배열 형태)

114	130	145	147	147	150	151	148	139	130	115	99	85	73	63
83	104	123	130	134	141	145	144	157	147	137	130	127	120	109
68	88	109	116	120	125	126	123	132	128	126	129	135	140	142
78	94	109	116	121	122	116	107	97	97	96	93	95	107	126
99	109	119	128	138	143	136	123	105	97	85	72	66	69	78
122	126	128	134	147	160	160	152	133	116	94	78	68	57	44
108	109	107	107	121	143	157	161	153	142	124	106	87	67	46
68	69	66	65	80	111	141	155	168	168	159	135	103	76	62
101	93	84	79	73	79	105	134	164	177	171	147	122	97	83
122	115	107	96	82	74	81	93	133	156	169	168	160	145	132
116	113	109	108	108	111	116	120	140	158	169	174	179	174	164
99	95	94	102	117	131	135	134	138	145	145	151	171	183	183
81	84	92	107	120	120	108	96	75	86	98	123	159	183	186
69	78	97	112	105	80	56	45	66	90	115	141	167	174	162
67	73	84	85	62	39	41	61	135	157	170	167	157	137	118



IO

이미지 파일에서 RGB 값 구하기

```
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;

public class A_Class {
    public static void main(String[] args) throws Exception{
        BufferedImage myImage = null;
        int width;
        int height;

        File ifile = new File("peng.jpg");
        try {
            myImage = ImageIO.read(ifile);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

IO

이미지 파일에서 RGB 값 구하기

```
        width = myImage.getWidth();
        height = myImage.getHeight();
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                Color c = new Color(myImage.getRGB(x, y));
                int red = (int) (c.getRed() * 0.299);
                int green = (int) (c.getGreen() * 0.587);
                int blue = (int) (c.getBlue() * 0.114);
                Color gray = new Color(red + green + blue, red + green + blue, red + green + blue);
                myImage.setRGB(x, y, gray.getRGB());
            }
        }
        File ofile = new File("gray.jpg");
        try {
            ImageIO.write(myImage, "jpg", ofile);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

수고하셨습니다.