19-21th-week

Part 04. 데이터 입출력

> **개념**
>
> 자바의 입출력 시스템은 자바의 입출력 스트림을 통해 구현된다. 이 시스템은 데이터의 입력과 출력을 관리하고, 다양한 유형의 데이터를 다룰 수 있게 도와준다.

데이터 입출력

- 입출력 스트림

> 키보드를 통해 입력되거나, 데이터가 모니터로 출력되는 것, 파일에 저장되거나 다른 프로그램으로 전송되는 것을 총칭하여 **데이터 입출력**이라고 한다.

구분	바이트 스트림		문자 스트림	
	입력 스트림	출력 스트림	입력 스트림	출력 스트림
최상위 클래스	InputStream	OutputStream	Reader	Writer
하위 클래스	XXXInputStream (FileInputStream)	XXXOutputStream (FileOutputStream)	XXXReader (FileReader)	XXXWriter (FileWriter)

> **스트림 종류**
>
> 바이트 스트림 : 그림, 멀티미디어, 문자 등 모든 종류의 데이터를 입출력할 때 사용
> 문자 스트림 : 문자만 입출력할 때 사용

1. 이미지는 바이트이므로 InputStream으로만 읽을 수 있다.
2. Reader의 read() 메소드는 1문자를 읽는다.

3. InputStream의 read() 메소드는 1바이트를 읽는다.
4. InputStreamReader를 이용하면 InputStream을 Reader로 변환시킬 수 있다.

- 바이트 출력 스트림

> 데이터를 바이트 단위로 출력하는 스트림이다. **OutputStream**은 바이트 출력 스트림의 **최상위 클래스**로 **추상 클래스**이다. 모든 바이트 출력 스트림 클래스는 OutputStream 클래스를 상속받아서 만들어진다.

```
package org.example;

import java.io.OutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class Box {
    public static void main(String[] args) {
        try {
            OutputStream os = new FileOutputStream("test.txt");
            byte[] array = {10, 20, 30};

            os.write(array);
            os.flush();
            os.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- 바이트 입력 스트림

> 바이트 단위로 데이터를 입력하는 스트림이다. **InputStream**은 바이트 입력 스트림의 **최상위 클래스**로, 추상 클래스이다. 모든 바이트 입력 스트림은 InputStream 클래스를 상속받아 만들어진다.

리턴 타입	메소드	설명
int	read()	1byte를 읽은 후 읽은 **바이트**를 리턴

리턴 타입	메소드	설명
int	read(byte[] b)	읽은 바이트를 매개값으로 주어진 배열에 저장 후 읽은 **바이트 수**를 리턴
void	close()	입력 스트림을 닫고 사용 메모리 해제

read(byte[] b)

매개값 b에는 읽은 데이터가 저장된다.
읽은 수 있는 바이트 수가 정해져 있다.
매개값 b에는 이전에 읽은 바이트가 남아있을 수 있다.

```
package org.example;

import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.OutputStream;

public class Box {
    public static void main(String[] args) throws Exception {
        String originFileNm = "/path/Downloads/test.png";
        String tmpFileNm = "tmp.png";

        InputStream is = new FileInputStream(originFileNm);
        OutputStream os = new FileOutputStream(tmpFileNm);

        byte[] data = new byte[1024];
        while (true) {
            int num = is.read(data);
            if(num == -1) break;
            os.write(data, 0, num);
        }

        os.flush();
        os.close();
        is.close();

        System.out.println("복사가 잘 되었습니다.");
    }
}
```

- 문자 입출력 스트림

> 문자 단위로 데이터를 입출력하는 스트림이다. 바이트 입출력 스트림인 InputSteam과 OutputStream에 대응하는 문자 입출력 스트림으로 Reader와 Writer가 있다.

리턴 타입	메소드	설명
void	write(int c)	매개값으로 주어진 한 문자를 출력
void	write(charp[] cbuf)	매개값으로 주어진 배열의 모든 문자를 출력
void	write(charp[] cbuf, int off, int len)	매개값으로 주어진 배열에서 cbuf[off]부터 len개까지의 문자를 출력
void	write(String str)	매개값으로 주어진 문자열을 출력
void	write(String str, int off, int len)	매개값으로 주어진 문자열에서 off 순번부터 len개까지의 문자를 출력
void	flush()	버퍼에 잔류하는 모든 문자를 출력
void	close()	출력 스트림을 닫고 사용 메모리를 해제

```java
package org.example;

import java.io.*;

public class Box {
    public static void main(String[] args) throws Exception {
        Writer writer = new FileWriter("test2.txt");

        char a = 'A';
        char[] arr = {'C', 'D'};

        writer.write(a);
        writer.write(arr);
        writer.write("STR");

        // 버퍼에 잔류하고 있는 문자들을 출력하고, 버퍼를 비움
        writer.flush();
        // 출력 스트림 닫고 메모리 해제
        writer.close();
    }
}
```

- 보조 스트림

> 다른 스트림에 연결되어 추가적인 기능을 제공하는 스트림. 예를 들어, 버퍼링, 압축, 인코딩 등의 기능을 제공한다.

입력 스트림 FileInputStream에 보조 스트림 InputStreamReader 를 연결

```
InputStream is = new FileInputStream("...");
InputStreamReader reader = new InputStreamReader(is);
```

문자 변환 보조 스트림 InputStreamReader에 보조 스트림 BufferedReader 보조 스트림 연결

```
InputStream is = new FileInputStream("...");
InputStreamReader reader = new InputStreamReader(is);
BufferedReader br = new BufferedReader(reader);
```

- 문자 변환 스트림

> 바이트와 문자 사이의 변환을 수행하는 스트림이다. 바이트 스트림(InputStream, OutputStream)에서 입출력할 데이터가 문자라면 문자 스트림(Reader와 Writer)로 변환해서 사용하는 것이 좋다. 문자로 바로 입출력하는 편리함이 있고, 문자셋의 종류를 지정할 수 있기 때문이다.

InputStream을 Reader로 변환

```
InputStream is = new FileInputStream("/path/test.txt");
Reader reader = new InputStreamReader(is);
```

OutputStream을 Writer로 변환

```
OutputStream os = new FileOutputStream("/path/test.txt");
Writer writer = new OutputStreamWriter(os);
```

```
package org.example;

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

```java
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.io.Reader;

public class Box {
    public static void main(String[] args) throws Exception {
        write("문자 변환 스트림을 사용합니다.");
        String data = read();
        System.out.println(data);
    }

    public static void write(String str) throws Exception {
        // 보조 스트림 연결
        OutputStream os = new FileOutputStream("test2.txt");
        Writer writer = new OutputStreamWriter(os, "UTF-8");

        // 보조 스트림을 이용하여 문자 출력
        writer.write(str);
        writer.flush();
        writer.close();
    }

    public static String read() throws Exception {
        // 보조 스트림 연결
        InputStream is = new FileInputStream("test2.txt");
        Reader reader = new InputStreamReader(is, "UTF-8");
        // 보조 스트림 이용하여 문자 입력
        char[] data = new char[100];
        int num = reader.read(data);
        reader.close();
        // 읽은 문자 수만큼 문자열로 변환
        String str = new String(data, 0, num);
        return str;
    }
}
```

- 성능 향상 스트림

> 입출력 성능을 향상시키는 스트림이다. 버퍼링을 통해 입출력 작업을 최적화한다.
> **BufferedInputStream**과 **BufferedOutputStream** 클래스가 이에 해당한다.

버퍼를 이용한 성능 향상

버퍼는 데이터가 쌓이기를 기다렸다가 꽉 차게 되면 데이터를 한꺼번에 디스크로 보냄으로써 출력 횟수를 줄여준다.

```
package org.example;

import java.io.*;

public class Box {
    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(
                new FileReader("src/main/java/org/example/Test.java")
        );

        int lineNo = 1;
        while (true) {
            String str = br.readLine();
            if(str == null) break;
            System.out.println(lineNo + "\t" + str);
            lineNo++;
        }
        br.close();
    }
}
```

- 기본 타입 스트림

> 기본 자료형에 대한 입출력을 지원하는 스트림이다. DataInputStream과 DataOutputStream 클래스를 통해 기본 자료형(boolean, char, short, int, long, float, double)의 데이터를 입출력할 수 있다.

DataInputStream과 DataOutputStream 보조 스트림을 연결

```
DataInputStream dis = new DataInputStream(바이트 입력 스트림);
DataOutputStream dos = new DataOutputStream(바이트 출력 스트림);
```

DataInputStream		DataOutputStream	
boolean	readBoolean()	void	writeBoolean(boolean v)

DataInputStream		DataOutputStream	
byte	readByte()	void	writeByte(int v)
char	readChar()	void	writeChar(int v)
double	readDouble()	void	writeDouble(double v)
float	readFloat()	void	writeFloat(float v)
int	readInt()	void	writeInt(int v)
long	readLong()	void	writeLong(long v)
short	readShort()	void	writeShort(int v)
String	readUTF()	void	writeUTF(String str)

- 프린트 스트림

> 텍스트 데이터를 간편하게 출력하는 스트림이다. PrintStream과 PrintWriter 클래스를 사용하여 포맷팅된 텍스트를 출력할 수 있다. PrintStream과 PrintWriter는 프린터와 유사하게 출력하는 pint(), println(), printf() 메소드를 가지고 있는 보조 스트림이다.

println

System.out.println() 에서 out은 PrintStream 타입이기 때문에 println()을 사용한다. 매개값의 타입에 따라 오버로딩되어 있다.

PrintStream은 바이트 출력 스트림과 연결되고, PrintWriter는 문자 출력 스트림과 연결된다.

```
PrintStream ps = new PrintStream(바이트 출력 스트림);
PrintWriter pw = new PrintWriter(문자 출력 스트림);
```

```
package org.example;

import java.io.*;

public class Box {
    public static void main(String[] args) throws Exception {
        FileOutputStream fos = new FileOutputStream("test2.txt");

        PrintStream ps = new PrintStream(fos);

        ps.print(" 마치");
```

```
            ps.println();
            ps.println("이것이 프린트다.");
            ps.print("| %6d");

            ps.flush();
            ps.close();
        }
    }
```

- 객체 스트림

> 객체를 입출력하는 스트림이다. ObjectOutputStream과 ObjectInputStream 클래스를 사용하여 객체를 직렬화하여 저장하고, 역직렬화하여 읽어올 수 있다.

직렬화와 역직렬화

객체를 출력하려면 필드값을 일렬로 늘어선 바이트로 변경해야 하는데, 이것을 직렬화라고 하고, 반대로 직렬화된 바이트를 객체의 필드값으로 복원하는 것을 역직렬화라고 한다.

ObjectInputStream과 ObjectOutputStream 보조 스트림을 연결

```
ObjectInputStream ois = new ObjectInputStream(바이트 입력 스트림);
ObjectOutputStream oos = new ObjectOutputStream(바이트 출력 스트림);
```

ObjectOutputStream으로 객체를 직렬화

```
oos.writeObject(객체);
```

읽은 바이트 역직렬화

```
객체타입 변수 = (객체타입) ois.readObject();
```

Serializable 인터페이스

> 자바는 Serializable 인터페이스를 구현한 클래스만 직렬화할 수 있도록 제안한다. Serializable 인터페이스는 멤버가 없는 빈 인터페이스지만, 객체를 직렬화할 수 있다고 표시하는 역할을 한다.

```
public class XXX implements Serializable {
        public int field1;
        protected int field2;
        int field3;
        private int field4;
        // 정적 필드는 직렬화에서 제외
        public static int field5;
        // transient로 선언된 필드는 직렬화에서 제외
        transient int field6;
}
```

SerialVersionUID 필드

- 직렬화할 때 사용된 클래스와 역직렬화할 때 상요된 클래스는 동일해야 한다. (이름과 내용이 동일해야 한다.)
- 두 클래스가 동일한 serialVersioUID 상수값을 가졌다면 역직렬화할 수 있다.

동일한 SerialVersionUID

```
public class Member implements Serializable {
        static final long serialVersionUID = 1;
        int field1;
        int field2;
}

public class Member implements Serializable {
        static final long serialVersionUID = 1;
        int field1;
        int field2;
        int field3;
}
```

```
import java.io.*;

public class SerializationExample {
    public static void main(String[] args) {
        // 객체를 직렬화하여 파일에 저장
        Member member = new Member();
        member.field1 = 10;
        member.field2 = 20;
        member.field3 = 30;

        try (ObjectOutputStream oos = new ObjectOutputStream(new
```

```
FileOutputStream("member.ser"))) {
            oos.writeObject(member);
            System.out.println("Serialization completed. Object saved to
member.ser");
        } catch (IOException e) {
            e.printStackTrace();
        }

        // 파일에서 객체를 역직렬화하여 가져오기
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("member.ser"))) {
            Member deserializedMember = (Member) ois.readObject();
            System.out.println("Deserialization completed. Object loaded from
member.ser");
            System.out.println("field1: " + deserializedMember.field1);
            System.out.println("field2: " + deserializedMember.field2);
            System.out.println("field3: " + deserializedMember.field3); //
field3는 Serializable 인터페이스에 없는 필드이므로 0으로 초기화됨
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}

class Member implements Serializable {
    static final long serialVersionUID = 1;
    int field1;
    int field2;
    int field3;
}
```

- File과 Files 클래스

> 파일 및 파일 시스템과 관련된 작업을 수행하는 클래스들. File 클래스는 단일 파일의 경로를 나타내고, Files 클래스는 파일 및 디렉터리를 다루는 다양한 메서드를 제공한다.

입출력 예외처리

자바에서 입출력을 사용할 때 반드시 예외처리를 사용하도록 강제하고 있다.

```
boolean isExist = file.exists();
```

리턴 타입	메소드	설명
boolean	createNewFile()	새로운 파일을 생성
boolean	mkdir()	새로운 디렉토리를 생성
boolean	mkdir()	경로상에 없는 모든 디렉토리를 생성

리턴 타입	메소드	설명
File[]	listFiles(FilenameFilter filter)	디렉토리에 포함된 파일 및 서브 디렉토리 목록 중에 FilenameFilter에 맞는 것만 File 배열로 리턴
String	getName()	파일의 이름을 리턴
long	lastModified()	마지막 수정 날짜 및 시간을 리턴
long	length()	파일의 크기 리턴
boolean	isDirectory()	디렉토리인지 여부

```
package org.example;

import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;

public class FileExample {
    public static void main(String[] args) throws Exception {
        File dir = new File("data");
        File file1 = new File("data/new1.txt");
        File file2 = new File("data/new2.txt");
        File file3 = new File("data/new3.txt");

        if(dir.exists() == false) { dir.mkdir(); }
        if(file1.exists() == false) { file1.createNewFile(); }
        if(file2.exists() == false) { file2.createNewFile(); }
        if(file3.exists() == false) { file3.createNewFile(); }

        File temp = new File("data");
        if(temp.exists() == false) { dir.mkdir(); }

        File[] contents = temp.listFiles();

        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd a HH:mm");
```

```
        for(File file : contents) {
            System.out.printf("%-25s", sdf.format(new
Date(file.lastModified())));
            if(file.isDirectory()) {
                System.out.printf("%-10s%-20s", "<DIR>", file.getName());
            } else {
                System.out.printf("%-10s%-20s", file.length(),
file.getName());
            }
            System.out.println();
        }
    }
}
```

> **입출력 스트림을 생성할 때 File 객체 활용하기**
>
> 파일 또는 폴더의 정보를 얻기 위해 File 객체를 단독으로 사용할 수 있지만, 파일 입출력 스트림을 생성할 때 경로 정보를 제공할 목적으로 사용되기도 한다.

```
// 첫 번째 방법
FileInputStream fis = new FileInputStream("./Temp");

// 두 번째 방법
File file = new File("./data");
FileInputStream fis = new FileInputStream(file);
```

Files 클래스

> Files 클래스는 정적 메소드로 구성되어 있기 때문에 File 클래스처럼 객체로 만들 필요가 없다. Files의 정적 메소드는 운영체제의 파일 시스템에서 파일 작업을 수행하도록 위임한다.

기능	관련 메소드
복사	copy
생성	createDirectories 등
이동	move
삭제	delete, deleteIfExists
존재, 검색, 비교	exists 등
속성	size, getLastModifiedTime 등

기능	관련 메소드
디렉토리 탐색	list, newDirectoryStream, walk
데이터 입출력	newInputStream 등

Path 객체

위 메소드들은 매개값으로 Path 객체를 받는다. Path 객체는 파일이나 디렉토리를 찾기 위한 경로 정보를 가지고 있는데, 정적 메소드인 get() 메소드로 다음과 같이 얻을 수 있다.

```
Path path = Paths.get("/pah/file.txt");
```

```
package org.example;

import java.io.File;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class FileInfo {
    public static void main(String[] args) {
        try {
            String data = "id: haribo \n" + "email: haribo@github.com \n" +
"tel: 010-0000-0000";
            String my_path = "tmp/user.txt";

            if(new File(my_path).exists() == false) {
                new File("tmp").mkdir();
                new File("tmp/user.txt").createNewFile();
            }

            Path path = Paths.get(my_path);

            // 파일 생성 및 데이터 저장
            Files.writeString(Paths.get("tmp/user.txt"), data,
Charset.forName("UTF-8"));

            System.out.println("파일 유형 : " + Files.probeContentType(path));
            System.out.println("파일 크기 : " + Files.size(path) + "bytes");

            // 파일 읽기
            String content = Files.readString(path, Charset.forName("UTF-8"));
```

```
            System.out.println(content);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

네트워크 입출력

- 네트워크 기초

> **네트워크란?**
>
> 여러 컴퓨터들을 통신 **회선**으로 연결한 것을 말한다.
> WAN(Wide Area Network) 은 LAN을 연결한 회선으로 이것을 인터넷(INTERNET)이라 부른다. LAN(Local Area Network) 은 가정, 회사 등 특정 영역에 존재하는 컴퓨터를 연결하고 있다.

- IP 주소 얻기

> **IP 구조**
>
> IP 주소는 네트워크 어댑터(LAN 카드)마다 할당된다. 컴퓨터 장착한 개수만큼 할당받을 수 있다.
> IP 주소는 xxx.xxx.xxx.xxx 형식으로 출력된다. 여기서 xxx는 0~255 사이의 정수다.
> 인터넷은 공인 IP로만 접속할 수 있는데, 일반 가정집의 경우 예로 KT 인터넷을 설치하게 될 경우 모뎀을 제공받는데 이것이 NAT라는 기술을 통해 사설 IP를 공인 IP주소로 변환시킨다.

아이피 주소를 확인하는 명령어

```
// 윈도우
ipconfig

// 맥
ifconfig
```

Port 번호

> **PORT 구조**
>
> IP는 네트워크 어댑터까지만 갈 수 있는 정보이기 때문에, 컴퓨터 내부에서 실행하는 서버 프로그램을 연결하기 위해서는 PORT 번호가 필요하다.
> PORT는 운영체제가 관리하는 서버 프로그램의 연결 번호로 서버는 시작할 때 특정 PORT 번호에 바인딩한다.
> 클라이언트도 서버에서 보낸 정보를 받기 위해 PORT 번호를 사용하지만 운영체제가 자동으로 부여하는 번호를 사용한다. 이 번호는 클라이언트가 서버로 요청할 때 함께 전송되어 서버가 클라이언트로 데이터를 보낼 때 사용된다.

구분명	범위	설명
Well Know Port Numbers	0~1023	국제인터넷주소관리기구가 특정 애플리케이션용으로 미리 예약한 Port
Registered Port Numbers	1024~49151	회사에서 등록해서 사용할 수 있는 Port
Dynamic Or Private Port Numbers	49152~65535	운영체제가 부여하는 동적 Port 또는 개인적인 목적으로 사용할 수 있는 Port

자바로 아이피 주소 얻기

```
InetAddress ia = InetAddress.getLocalHost();

InetAddress ia = InetAddress.getByName(String domainName);
InetAddress[] iaArr = InetAddress.getAllByName(String domainName);

String ip = InetAddress.getHostAddress();
```

- TCP 네트워킹

> **전송용 프로토콜 TCP**
>
> IP 주소로 프로그램들이 통신할 때 약속된 데이터 전송 규약이다. TCP와 UDP가 있으며 TCP는 IP와 함께 사용하기 때문에 TCP/IP 라고도 한다.
> TCP는 웹 브라우저가 웹 서버에 연결할 때 사용되며 이메일 전송, 파일 전송, DB 연동에도 사용된다.

소켓의 개념

- **소켓(Socket)**: 소켓은 네트워크 상의 두 노드 간에 통신 채널을 열기 위한 엔드포인트이며, 소켓을 통해 데이터를 주고받을 수 있다.
- **TCP 소켓**: 연결 지향적 통신을 제공한다. 데이터의 전송을 보장하고, 순서대로 도착하도록 한다. 소켓은 `java.net.Socket` 클래스로 표현된다.
- **UDP 소켓**: 비연결 지향적 통신을 제공한다. 데이터 전송의 신뢰성이나 순서를 보장하지 않는다. 소켓은 `java.net.DatagramSocket` 클래스로 표현된다.

TCP 서버 프로그램 개발

```
ServerSocket serverSocket = new ServerSocket(50001);
Socket socket = serverSocket.accept();

InetSocketAddress isa = (InetSocketAddress) socket.getRemoteSocketAddress();
String portNo = isa.getPort();

serverSocket.close();
```

TCP 클라이턴트에서 서버에 연결 요청

```
Socket socket = new Socket("IP", 50001);
Socket socket = new Socket(New InetAddress.getByName("domainName", 50001);

socket = new Socket();
socket.commect(new InetSocketAddress("domainName", 500011));

try {
        // socket 생성과 동시에 localhost의 50001 port로 연결 요청
        Socket socket = new Socket("IP", 50001);
        System.out.println("클라이언트 연결 성공");

        socket.close();
        System.out.println("클라이언트 연결 끊음");
} catch (UnknownHostException e) {
        // ip 표기 방법이 잘못되었을 경우
} catch (IOException e) {
        // ip와 port로 서버에 연결할 수 없는 경우우
}
```

- UDP 네트워킹

전송용 프로토콜 UDP

TCP는 데이터 유실이 일어났을 때 감지하고 재전송하는 매커니즘을 가지고 있지만 UDP는 이러한 메커니즘이 포함되어 있지 않다. UDP는 신뢰성 없는 비연결형 프로토콜이기 때문이다. 한 컷 정도의 손실이 문제 없는 실시간 영상 스트리밍의 경우 신뢰도 보다 속도가 중요하기 때문에 UDP를 사용할 수 있다.

```
DatagramSocket datagramSocket = new DatagramSocket(50001);
```

- 서버의 동시 요청 처리
- JSON 데이터 형식
- TCP 채팅 프로그램